Visual Computing Institute

RWTH AACHEN UNIVERSITY

The present work was submitted to the
Chair of Computer Science 13 (Computer Vision)
Faculty of Mathematics, Computer Science and Natural Sciences
Prof. Dr. Bastian Leibe

Master Thesis

Monocular 3D Human Pose Estimation using Depth as Privileged Information

presented by

Yinglun Liu
Student ID: 391562

2021-07-14

First examiner:	Prof. Dr. Bastian Leibe
Second examiner:	Prof. Dr. Leif Kobbelt

Contents

Abstract Monocular 3D human pose estimation has come a long way in recent years thanks to the introduction of deep convolutional architectures. Nevertheless, in complex scene environments, the problem of scale ambiguity and occlusion still pose a great challenge for current algorithms. In this work, we propose to improve 3D human pose estimation by taking advantage of the depth information available at train-time. More precisely, we present a privileged information learning framework that teaches a fully convolutional network to effectively extract depth cues from RGB images. The model can then be deployed at test-time to infer a 3D human pose based on a single RGB image. Moreover, we explore the possibility to make use of the unlabelled portion of our data as well, since this is inherently made feasible by our design. We conduct experiments on three different large-scale datasets to validate the efficacy of the proposed method and test out a variety of design choices.

1 Introduction

The rapid development of 3D human pose estimation algorithms in the past two decades has enabled a wide range of possibilities. For example, modern video game consoles use the technology for real-time motion analysis. From a sensor-based snapshot of the current environment, the technology infers the current full-body posture of each player. Over time, the build-in software assembles these single-frame estimations into temporal trajectories of 3D body poses. Based on these trajectories, the actions of the players are determined, thereby gone the need for a physical controller. As in Figure 1.1, the pose we talk about here is usually defined as the relative orientation and distance between adjacent body parts on a pre-defined kinematic tree.

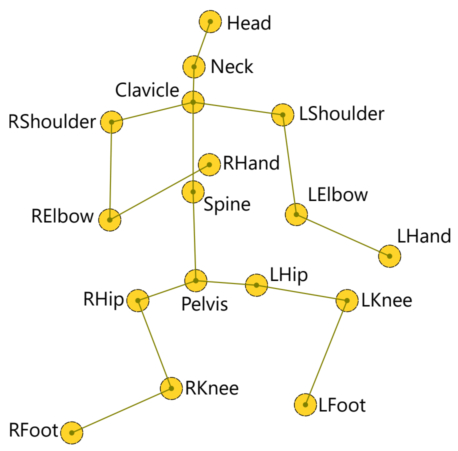
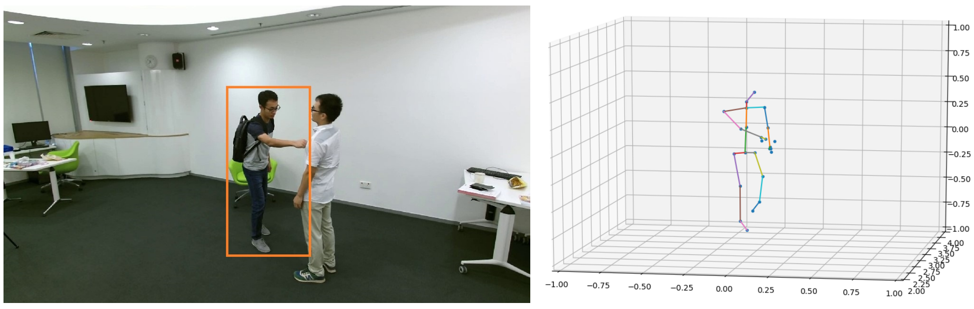


Figure 1.1: Example of a skeletal tree that defines the kinematic topology of the human body.

But the potential of 3D human pose estimation is not confined to just gaming technologies. Sign language recognition, where pose estimation plays a major role, opens the occasional possibilities of interaction and communication for the speech-impaired [RKE20]. Driver attention monitors periodically evaluates the posture of the driving personnel to assess the driver's alertness and give warnings where appropriate [XLZ+17]. Conventional motion capture systems require the subjects to wear either optical markers or inertial measurement units in order to determine an array of keypoint positions over time. Such a system is, despite its prevalence, typically expensive and tedious to deploy, whereas a more affordable alternative, on the other hand, would be to apply monocular pose estimation [MSM+20]. Yet the most crucial application of this technology lies in autonomous systems. Similar to a game console, a service robot needs to recognize and understand in real-time a user's command from his hand gestures. An autonomous vehicle monitors pedestrian actions all the time to enable timely responce under critical situations [GPB+20]. In both scenarios, pose estimation serves as the corner stone for more advanced cognitive functions.

1.1 Motivation



(a) Input image (b) 3D Estimation

Figure 1.2: Monocular 3D human pose estimation. The goal is to give a camera-space estimation of the 3D pose of the person given an image and a bounding box.

At the core of this work, we intend to improve on how 3D human pose estimation in the context of computer vision is done, utilizing convolutional neural networks [KSH12]. To be more precise, we focus on the particular sub-problem of monocular single-person pose estimation: Given a single image at inference time and a bounding box containing a person, this task demands the reconstruction of a full-body pose in the form of 3D skeletal keypoints in camera-space (see Figure 1.2). Formally, the 3D pose $\mathbf{P}$ containing J keypoints of a person is formulated as

$$\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_J) \in \mathbb{R}^{3\times J}, \tag{1.1}$$

where $\mathbf{p}_j$ is the position of the j-th skeletal keypoint in the camera-centered coordinate system. This is a not trivial task by any means mainly due to a number of obstacles:

- It is inherently impossible to infer the size of a person from just a single picture.
- Occlusion between different subjects and self-occlusion causes some body parts to be at least partially invisible.
- Samples containing rare poses and strange camera angles tend to lead to bad predictions.

1.1.1 Challenges

We will discuss in detail about each of these problems in the paragraphs below.

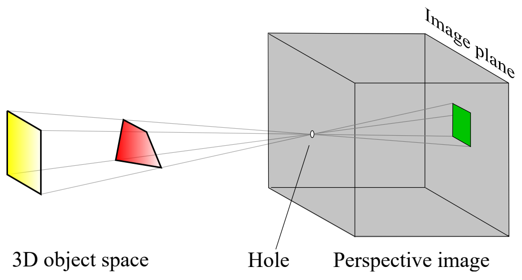


Figure 1.3: Classic pinhole camera model. Note that the projection of the yellowish object and the reddish object onto the imaging plane leads to, regardless of color, the same image.

Scale Ambiguity To get a whole picture on the problem of scale ambiguity in the context of 3D human pose estimation, we need to first understand how imaging works. In this case, it is sufficient to approximate the complicated imaging process of a modern camera with the classic pinhole camera model (see Figure 1.3). Under this model, every 3D point of an object in camera-space, if within the field of view of the camera, corresponds to a pixel on the imaging plane. The image-space location of this pixel is determined via a perspective projection. This projection is mathematically carried out in two steps. In the first step, our 3D camera-space coordinate $(x, y, z)^\intercal$ is converted into a normalized camera coordinate $(x', y', 1)^\intercal$ via

$$(x', y', 1)^\intercal = (x, y, z)^\intercal \cdot \frac{1}{z}. \tag{1.2}$$

This essentially tries to ascertain just how far away the point would deviate from the optical center on the x- and y-axis, if it were one unit away from the optical center on the z-axis. The second step obtains an image space location $(u, v, 1)^\intercal$ by multiplying this normalized camera coordinate by the intrinsic matrix of the camera:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & s_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}. \tag{1.3}$$

Here f_x and f_y are the focal lengths of the camera and c_x and c_y are principal point offsets. The former decides the number of pixels a physical unit in camera-space covers, whereas the latter simply stipulates in pixel units the location of the principal point relative to the origin of the image. The principal point is the foot of the perpendicular to the imaging plane that passes through the optical center. s_x is a parameter that describes the extent of axis skew of the camera, which causes shear distortion in the projected image. On a modern day camera this is however negligible and simply treated as zero.

It is therefore evident that the movement of a point in camera-space bears no impact on the location of the pixel it projects to, as long as it stays on the line uniquely specified by its original position and the optical center. Figure 1.3 illustrates this effect: The reddish plane and yellowish plane projects onto the same pixels on the imaging plane despite the fact that they are different objects. Now in the context of 3D human pose estimation, this implies that there is an inifinite number of hypothesis on how tall and far away from the camera the subject is, that would perfectly explain why the person looks the way he is in that image. Of course we could argue that the physical size of a human falls within a certain range, and we may always get a rough clue of how tall the person is from the ratio of his limbs and the relative size of other objects that appear in the background, but we could never know for sure.

Occlusion Occlusion happens when the sight of the camera on a body part of the subject is blocked by some other subject or himself. While a human can infer from his experience what a plausible pose could be, such occlusions often take away the visual evidence essential for an algorithm to make a reliable speculation. Moreover, when two or more subjects stand close to each other, confusion caused by the mutual presence of body parts from various people in the bounding box makes it tricky to associate the corret body part to the subject in question. Additionally, it may further compound matters if a bounding box is not given in some test cases. Modern approaches typically tackle this in a top-down fashion: A detection is performed prior to the pose inference phase to obtain a bounding box of the subject. Since heavy occlusion often hampers the performance of the detector, the pose estimator may fail to function properly due to a bounding box of below-par quality. An example of these problems is shown in Figure 1.4.

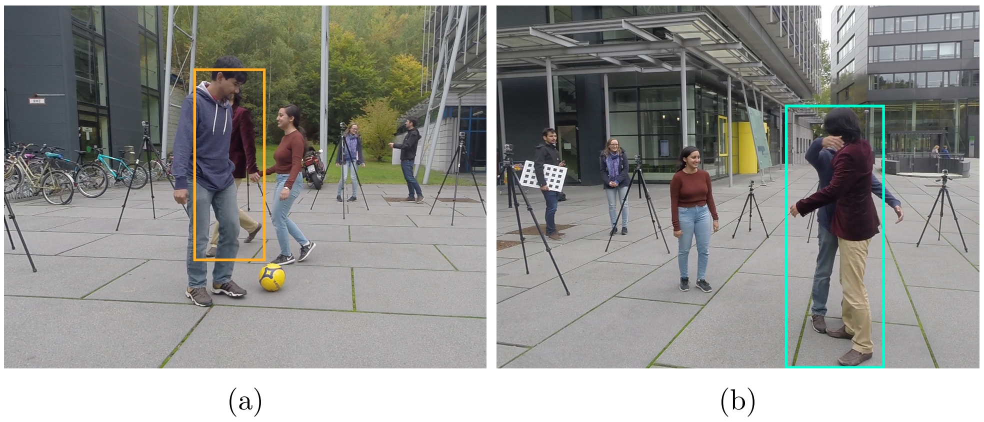


Figure 1.4: Samples that feature heavy occlusion: Case-a: Subject in question is heavily occluded by the person in the foreground. Case-b: Bounding box contains body parts from another subject.

Rare Poses and Camera Angles Rare samples at inference time pose a great challenge to almost all computer vision tasks [SGG16]. As is often the case, a model that is trained to fit an unbalanced dataset exhibits a tendency to offer the most commonplace prediction for an unfamiliar input image. Nevertheless, by virtue of the dendroid nature of 3D skeletons, 3D human pose estimation deals with a higher degree of uncertainty than many other visual recognition tasks. On one hand, the model may have trouble extracting the various body parts from visual clues on the image; on the other hand, the algorithm might simply fail to work out a semantically correct way to place those body parts in space thanks to the pinhole camera model we discussed above. Previous works attempt to restrict the degrees of freedom for each skeletal keypoint either through the adoption of a set of explicit limb-scale constraints [PZDD17] [SXW$^+$18] or via the employment of a hierarchical skeleton retrieval strategy [LCY18]. Despite such efforts, rare poses and camera angles still account for a relatively large portion of total errors at test time.

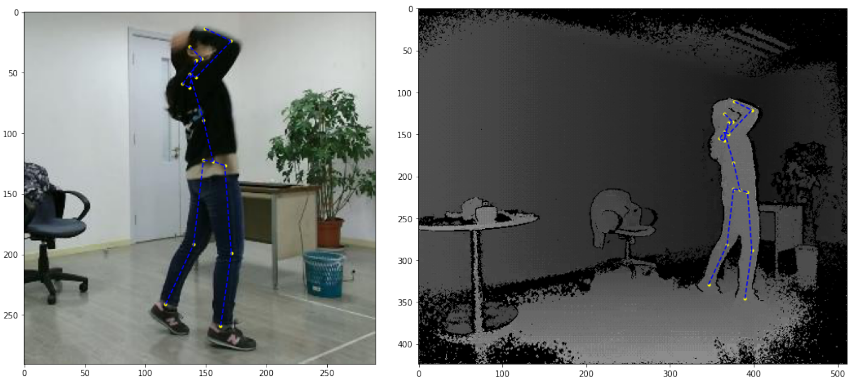
1.1.2 Further Context

On account of the challenges described in the previous section, it is a natural idea to think of offering more information to the recognition model by introducing other input modalities into this task. Of all the commonly available modalities, depth information in the form of depth maps has caught the most attention in 3D computer vision tasks. This is primarily because the ability to effectively extract depth cues from the visual disparity between both eyes is exactly what makes human vision so keen on recognizing 3D shapes in complex environments, or in the context of human pose estimation, the 3D pose. [ZWD$^+$18] and [SFEG19] are approaches for which the incorporation of depth input led to significant boost in overall performance of the algorithm. From a pragmatic standpoint, however, their methods presume the availability of depth information at inference time, which is a harsh assumption for a great many application scenarios, where having a depth sensor at disposal is a luxury. What we practically want instead is an algorithm that

a is able to, at train time, improve the model by extracting valuable knowledge from the available depth input.

b is able to, at inference time, offer a pose estimation on the basis of a single RGB image.

A natural solution to this specific problem setup would be intuitive and has already been explored by [VL20]. Given full depth annotation at train time, they propose to simplify the 3D estimation task into a 2D problem plus an additional depth map prediction step. But since depth sensors could only provide depth annotation for points on the front surface of any object, a direct lookup of predicted depth values for the skeletal keypoints is never reliable, let alone situations that involve any form of occlusion (see Figure 1.5 for an example). As such, [VL20] chose to insert a workaround step in their design as a counter-measure.



(a) RGB image (b) Depth image

Figure 1.5: Example for an unreliable depth lookup: the depth values looked up at the pixel locations of quite a few keypoints are far away from the actual depth of those keypoints by virtue of self-occlusion.

1.1.3 Objectives

By contrast, we notice that this problem setup closely fits the preconditions for privileged information learning to come into play [VI15]. In essence, privileged information describes the auxiliary source of information provided at train time, in addition to the regular features, to a student model by a stronger teacher model.

This additional explanatory cue should, at least theoretically, offer guidance for the student to gain the vital insight into how its teacher would approach the same problem, despite having no access to such counselling at test time [LPBSV15]. Here in the context of 3D human pose estimation, we argue that a good piece of privileged information could be the intermediate feature maps provided by a teacher convolutional neural network, who has exclusive access to the depth maps associated with each sample. A student network, on the other hand, should process this information in a way such that it learns how to extract such rich features on its own when only an ordinary RGB image is given as input.

The goal of this master thesis is, in the presence of fully annotated RGB-D images, to develop a learning framework for such knowledge transfer to take place. Moreover, since unlabeled RGB-D image pairs are available in abundance, we explore the possibility to acquire some weak supervision from this portion of our data as well, in the hope of a further gain in performance. We conduct experiments on two large-scale action recognition datasets [LSP$^+$19] [LHL$^+$17] and an additional test set [IPOS13] to examine whether our proposed learning paradigm is able to overcome the aforementioned problems to some extent, and to see how privileged information about depth could lead to changes to the way a convolutional neural network understands an RGB image containing a human pose.

1.2 Overview

The rest of this thesis will be organized as follows. In chapter 2, we will recapitulate the various basic aspects of deep learning and, in particular, convolutional neural networks, so as to show that the feasibility of our proposed method is well-grounded. In chapter 3, we briefly address the various topics closely related to this thesis and discuss how previous works approach similar problems.

In chapter 4, we will first give a detailed explanation of the baseline method that is the foundation of this thesis, then go on to propose our own pose estimation framework that facilitates knowledge transfer between a teacher and a student model. In chapter 5, we address details on data preparation and how our learning framework is implemented at train-time and test-time. In chapter 6, we will present the results of various experiments we carry out and analyze whether the impact of our proposed method meets our expectations. In the last chapter, we will summarize the content of this thesis and conclude on the findings we make.

2 Fundamentals

Since our proposed method is based on convolutional neural networks, we dedicate this chapter to the various fundamental aspects of these deep artificial architectures that have revolutionized computer vision for the past decade. Considering the fact that convolutional neural networks are just a branch form of artificial neural networks specialized for dealing with computer vision tasks, we start by taking a look at the more general concept of the latter.

2.1 Artificial Neural Networks

Originally a mimic of the biological neural structures in a human brain, artificial neural networks today attain a ubiquity throughout the field of artificial intelligence few in the past could have expected. From recommendation systems [FQY+18] to machine translation, from general object detection to mastering the game of Go [SSS+17], these networks are capable of developing the competence to take on tasks that require a certain level of intelligence.

2.1.1 Neurons

In contrast to its great potential, the atomic unit structures that constitutes an artificial neural network is actually quite simple: A neural network is at its core just a set of interconnected neurons. A neuron takes as input signals either from the outside or sent out by other predecessor neurons and then, based on the value of those signals, emits itself an output signal that is further propagated forward. The output signal is typically a linear combination of the inputs plus an offset. In the simplest case, a single neuron can serve as a tiny neural network whose forward propagation can be formulated as

$$y(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = \sum_{s \in S} w_s a_s + b, \tag{2.1}$$

where $y(\mathbf{x})$ is the output signal of our neuron, $\mathbf{x} = (a_1, a_2, \ldots, a_S)^\intercal$ is the array of input signal values, $\mathbf{w} = (w_1, w_2, \ldots, w_S)^\intercal$ is the array of associated weights for each input signal s, b is the bias. The weights and the bias are learnable parameters for our neuron. A illustration of a single-neuron network is given in Figure 2.1.

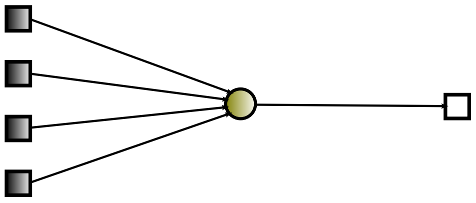


Figure 2.1: Example of a single-neuron artificial neural network. The grey squares to the left of the neuron represent input signals. The white square to the right of the neuron stands for the neuron's output.

2.1.2 Gradient Descent

We want the neuron to automatically adjust its parameters so that for any given array of input signals, the neuron's output approximates the value of a target function f that we intend to learn. To achieve this, we prepare a set of N training samples on which the neuron learns by trial and error. Each sample $i \in N$ on this training set is a tuple $(\mathbf{x}_i, t_i)$, where $\mathbf{x}_i$ is the array of input signals and t_i is the training label. Naturally it is guaranteed that $f(\mathbf{x}_i) = t_i$ holds for all samples i on this set. We enumerate over all the samples and measure for each sample, based on a chosen criterion, the extent by which the neuron's output deviates from the associated target label. We call the arithmetic mean of these measurements the network's evaluation loss on this training set. A typical criterion for quantizing this loss is the mean squared error function:

$$E(y) = \frac{1}{N}\sum_{i=1}^{N}(y(\mathbf{x}_i) - t_i)^2. \tag{2.2}$$

Now that we have a measure of how inaccurate the network's output is with respect to the target labels, we want to optimize the parameters of our network such that this loss is gradually cut down. One common way to do this is to perform gradient descent on the parameter space of our model. Essentially, the gradient of an error function with respect to a parameter points to the direction in which the error increases the fastest. By repeatedly pushing our parameters slowly in the opposing direction of the gradients, we effectively adjust our model in a way such that this evaluation loss subsides over time. The formulation for updating the weight associated with input signal $s \in S$ under such an optimization scheme is

$$w'_s = w_s - \eta \frac{\partial E}{\partial w_s}, \tag{2.3}$$

where the learning rate η is a hyperparameter that decides the extent by which we shift our weight parameters in a single iteration. The update of the bias for our neuron is carried out in similar fashion. This update scheme is repeated for a large number of iterations until, ideally, the network reaches a state of convergence, where the evaluation loss visibly stops to decrease further but oscillates around a small value instead.

2.1.3 Multi-layer Perceptron and Activation Functions

Our single-neuron network discussed above is probably able to approximate a linear target function quite well, but there's a limit to what a single-neuron can achieve due to its linear nature. To learn a more complex target function, we need to stack up multiple layers of these neurons, where each neuron in one layer takes as input the signals emitted by all neurons in the previous layer. This is the general idea behind multi-layer perceptrons (see Figure 2.2 for an example). For easier explanation, we refer to the array of signals input to and output from a network layer as features from this point on.

By organizing neurons in a hierarchical fashion, it is hoped that neurons at the front would learn to extract useful features, based on which neurons at the back are able to confidently decide on what the final output signals should be. Such an organization, however, poses a great obstacle to the optimization of the model. Whereas derivation of the gradients is straightforward for a single-neuron network, in a multi-layer perceptron, a weight parameter associated to a certain connection between two neurons can exert an influence on the final evaluation loss via an exponential number of paths each consisting of a few neural connections. To separately derive the gradient components by tracing back along each path and then aggregate them for the final weight update would induce an unmanagable computational cost. To tackle this problem, a dynamic programming algorithm termed the back propagation was designed. In this algorithm, the gradients of the evaluation loss with respect to the output features of each layer is computed in reverse order. Since the layers are arranged in a fixed sequence, by the time the derivatives with respect to the output features of one layer have been calculated, all the ingredients needed for the derivatives of the previous layer to be computed are already there. As a result, the gradients can be propagated layerwise backwards while no redundant calculations take place. After the derivatives with respect to the output features of all layers are ready, gradients with respect to the network parameters are worked out and the weight updates are carried out as before.

Despite increased strutural complexity, simply piling up more layers of neurons won't bring in much improvement, because a linear combination of a set of linear functions is indeed again a linear function. What we need is to apply a filter function on the output signal of each neuron that introduces some non-linearity into our network. Such a filter function is called an activation function and is typically differentiable so that the gradients of the loss may still be derived for each trainable parameter after its adoption.

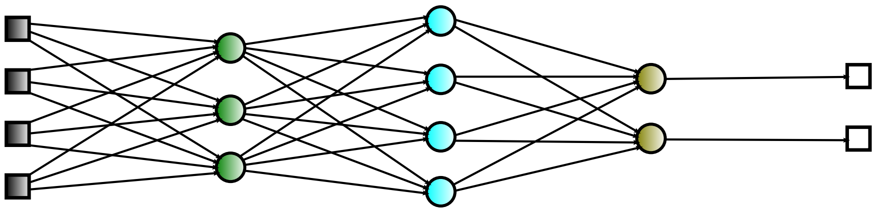


Figure 2.2: Example of a multi-layer perceptron. Multiple layers of neurons are assembled together to enable better expressiveness.

The choice of activation functions bears a significant impact on the learning capacities of neural networks. For computer vision tasks the logistic sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2.4}$$

or the hyperbolic tangent function

$$tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{2.5}$$

were assumed to be the default selection until a few years ago. However, both of these functions demonstrate saturated behaviour for input values of large magnitude. At the same time, the derivative of both functions rapidly approaches zero as the input value edges towards infinity on both ends. For deep networks that consist of a huge number of layers, this leads to the vanishing gradient problem: The gradients of the loss with respect to the trainable parameters of the earlier layers become so small that they can't be numerically represented on modern computational devices. As a consequence, the network suffers from precipitate convergence and fails to closely fit the training data.

To circumvent this issue the rectified linear activation function

$$relu(a) = max(0, a) \tag{2.6}$$

and its variations [XWCL15] were introduced. Not only does this rectifier function preserve linear behaviour for positive input values, which is crucial for keeping the gradients at a healthy scale and faster convergence, it induces less computational overhead than conventional activation functions as well. And thanks to its non-linear property, a network that implements such an activation function for its hidden layers could still enjoy the added benefit of better expressiveness.

It is worth noting that training efficacy is closely dependent on the way a neural network's trainable parameters are initialized as well. Conventionally, the network parameters are initialized to be some small random value around zero. This is because an initialization with any constant value will lead to symmetric gradients for all network parameters across the same layer on each iteration, resulting in a restricted search on parameter space and thus a degenerate optimization process. For different choices of the activation function, the best way to intialize the parameters may differ, but the general rule of thumb is that, for each layer, the variance of the output features should remain close to that of the input. For example, given the fact that the previous layer and the current layer contains m and n neurons respectively, a glorot-initialization [GB10], where each weight is drawn from the uniform distribution

$$U[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}], \tag{2.7}$$

is often preferred when the logistic sigmoid function is employed, primarily due to the function's limited window of high sensitivity around zero. On the other hand, a he-initialization [HZRS15], where each weight obeys the normal distribution

$$N[0, \sqrt{\frac{2}{m}}], \tag{2.8}$$

is nowadays the defacto standard for networks that adopt the rectified linear function, since for this activation function the linearity assumption around zero does not really hold.

2.1.4 Training Strategies

In Subsection 2.1.2 we employed the practice where, on each iteration, we compute the average evaluation error for the current model on the whole training set before the gradients are derived. Such a strategy is called batch learning. Conceptually, batch learning shifts the model parameters in the exact direction of the global optimum, since the gradients were derived from an average error computed over all samples from the training set. In practice, however, due to the non-convexity of the optimization domain, such a strategy often causes the model to be prematurely stuck in a local optima. Moreover, modern training sets for visual recognition tasks typically contain at least tens of thousands of image. The limited memory capacity of commonly affordable computational devices means that performing batch learning is most of time far from a tractable practice.

Stochastic learning provides an alternative. Instead of presenting all samples at once, the stochastic method sweeps through the training set and directly performs an update to the model parameters based on each individual sample. The gradients derived from single-sample evaluation tend to be noisier and may cause the model to move through parameter space in a swaying manner, but from time to time, they might also kick the model out of the local maxima that hinders further improvement.

Mini-batch learning strikes a balance between the two strategies above by processing a small set of randomly picked samples each time before a parameter update takes place. On one hand, the method exhibits better stability than stochastic learning due to the fact that the gradients used for model updates are averaged out over multiple samples. On the other hand, a mini-batch is still able to provide the vital randomness that helps a model avoid suboptimal basins of attraction. Additionally, since a mini-batch can most of the time fit into memory without a problem, the method opens the possibilities of vectorized computation, which greatly reduces the overall training time required by modern architectures.

2.1.5 Training Configurations

For neural networks, a successful training session relies closely upon chosing the right configuration from a range of options. For example, the learning rate determines the step size of a parameter update in the opposite direction of the gradient. The optimal magnitude of the learning rate used for each step is tricky to grasp. A large value allows the model to learn fast but may cause ocillation around the optimum in the final stages of the training process, whereas a small value guarantees a steadier learning curve at the risk of possibly having the model prematurely stuck in a suboptimal solution on the optimization surface. As such, a common practice is to gradually cut down the learning rate according to a predefined schedule as the training progresses, and there are a number of ways to do this:

- A step-decay scheduler multiplies the learning rate by a small factor every few epochs.
- A time-decay scheduler sets the learning rate to be the reciprocal of a linear function on the number of past epochs.
- An exponential-decay scheduler tunes down the learning rate with a negative exponential function on the number of past epochs.

Here an epoch refers to a full pass through all samples in a training set. Selecting a best policy to schedule the learning rate usually depends on the task at hand.

Another important decision towards the optimization of a neural network is how the parameters are updated based on the gradients. The classic update policy for stochastic gradient descent is decribed by Equation (2.3), but more advanced approaches have been proposed to enhance the efficiency of the parameter search. For example, the momentum method proposes to keep track of parameter updates that took place in the past and use this knowledge to guide future updates. This is done by keeping in memory an accumulated average of the shift vectors for previous iterations. A newly computed shift vector is always incorporated into this inertia vector before the latter is then used for the next update. Since this inertia vector essentially points in the direction of the steepest descent on average, such a strategy may dampen unnecessary oscillations and is often able to help the model steer through flat regions or steep curvature in the search space.

As an upgrade to the momentum method, the Adam optimizer proposed by [KB14] makes the further observation that, for deep architectures, the magnitude of gradients for layers close to the entrance of the network may vary significantly from those for layers close to the exit. A sensible optimization process should, therefore, consider this difference and try to update the parameters based on a somehow normalized version of the gradients. To achieve this, the Adam optimizer maintains for each layer a running estimation on the averaged squares of the gradients as well. It then applies this term to normalize the inertia vectors discussed above, before an actual update is executed. Since its release, the standard Adam optimizer has received broad recognition throughout the deep learning community.

2.2 Convolutional Neural Networks

While multi-layer perceptrons are powerful structures at appoximating functions of certain complexity, their application to the field of computer vision has been confronted with great obstacles. This is mainly by virtue of the fact that in a multi-layer perceptron, a neuron receives input from each neuron in the previous layer and outputs its activation signal to all neurons in the next. For a typical classification task, where each sample in the training set consists of an RGB image and its class label, the only way to feed the image into our perceptron as an input would be to flatten the image into a vector of floating point numbers. If we assume an image is of the shape $W \times H \times 3$ and there are m neurons on the input layer of the network, the number of trainable parameters in that layer would be $3mWH$. For tasks that require a high-resolution input, this number will get huge and induce an intractable computational cost on the optimization process, let alone the excessive time consumption.

2.2.1 Convolution Layer

In order to reduce the number of parameters in each layer without losing the dependencies between neurons that are critical for understanding an image, the convolution operation, or the convolution layer, was introduced. Taking inspiration from the neural structures inside the human visual cortex, each neuron in a convolution layer is only connected to neurons from the previous layer that are within a small local neighborhood. Moreover, since the connection rules are identical for all neurons on the same layer, the learnable weights associated with these connections are shared as well. This results in a forward propagation that is fundamentally different from the one that happens on a fully-connected layer. Essentially, computing the output, or feature map, of a convolution layer is equivalent to sliding a filter over all possible image locations and each time calculating a weighted sum of the pixel values temporarily covered by this filter. It is worth noting that in such a window-scan, the stride with which the filter advances does not necessarily have to be one. A convolution operation for which the filter moves at least two units each time is called a strided convolution. Formally, the convolution operation is defined as

$$\mathbf{G}[u,v] = \mathbf{T} * \mathbf{F} = \sum_{c=1}^{C} \sum_{i=1}^{2A+1} \sum_{j=1}^{2B+1} \mathbf{T}[c,i,j]\mathbf{F}[c,u+i,v+j], \tag{2.9}$$

where $\mathbf{G}$ is the output feature map, $\mathbf{T}$ is the filter of shape $C\times(2A+1)\times(2B+1)$, $\mathbf{F}$ is the input image or the feature map of the last convolution layer that contains C channels. To be able to extract more information and thus allow for better expressiveness, usually more than one filter is applied at the same time. These individual filters are combined into what is called a convolution kernel, which is essentially a 4-dimensional tensor that can be slid over the image space in a highly parallelized manner on modern computational devices. The output of these filter operations are combined as well into a multi-channel feature map that is passed further forward. For instance, if a convolution kernel of the shape $K\times C\times(2A+1)\times(2B+1)$ is applied to an input of the shape $C\times H\times W$, the output features of this convolution operation will be of the shape $K\times(W-2A)\times(H-2B)$. An illustration of the convolution operation is given in Figure 2.3.

Despite a reduction by orders of magnitude in the number of trainable parameters, the design of convolutional structures enables the network to capture the spatial dependencies vital for extracting from small local areas visual cues like corners and edges. However, it takes more information that those local patterns to recoginize general semantic concepts from an image like a vehicle or a horse. In a biological vision system, high-level cognitive functions are achieved via the cooperation of a large number of neurons taking on different tasks. This is implemented analogously in a convolutional neural network. By stacking multiple convolution layers in a row, neurons in later layers have access to information about a broader part of the image by gathering the features output by the small group of neurons in front of them. This allows for the detection of more complex appearance structures like the wheel of a car or the window of a house, and eventually the recognition of entire semantic objects.

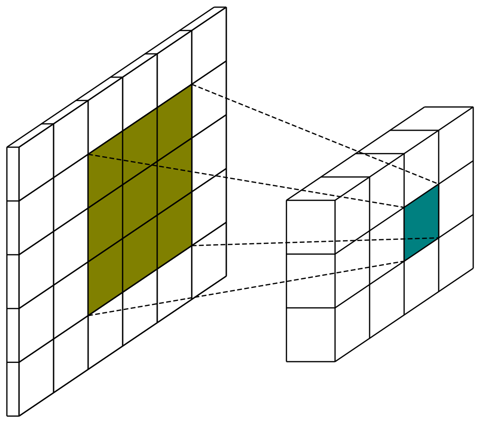


Figure 2.3: Example of a convolution layer. Values on the output feature map is computed via a window-scan with the kernel. Note that the spatial dimensions of the feature map are shortened by twice the half-width of the kernel after the filter operation.

2.2.2 Pooling Layer

But convolution layers alone wouldn't be powerful enough to make up a network capable of such visual cognition. This is because in a network that comprises only convolution layers, even the neurons in the last layer won't have access to local patterns collected from distant areas, unless the network is extremely deep. We call the portion of the original image that a neuron is able to gather information from its receptive field. Empirically, we want the neurons in middle-level layers to already have a sizable receptive field, so that a considerable part of the network is dedicated to processing global features - features output by high-level neurons who already sees the whole picture before making a decision. This is the reason behind the introduction of pooling layers.

A pooling layer spatially downscales a feature map by segmenting the image space into small patches and reducing the local features on each patch into a single value. The reduction operation is done separately but in the same way for each channel of the input features. There are two simple options for this reduction operation: Either the average or the maximum value of the input features on the same patch is taken as the output. Figure 2.4 demonstrates an example for the 2×2 pooling operation.

Pooling layers are an integral part of modern convolutional architectures for a variety of reasons. On one hand, by placing a pooling layer behind every convolution layer, not only do the receptive fields of neurons in successive layers increase exponentially, a rapid decline in the overall computational effort for the back propagation of gradients is achieved as well. On the other hand, since a max-pooling layer always chooses the dominant value from a local neighborhood as its output, its usage encourages the network to produce translation- and rotation-

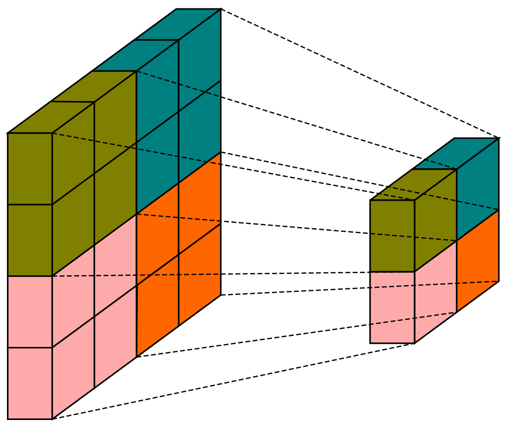


Figure 2.4: Example of a pooling layer. Each pixel entry on the output feature map is associated with a patch of pixel entries on the input. The output features are computed as the average or maximum value over the associated input patch.

invariant features, which are valuable to a range of visual recognition tasks. This is also partly why max-pooling is often preferred over average-pooling in modern convolutional networks.

2.2.3 Batch Normalization

One of the reasons why deep convolutional neural networks are difficult to optimize is that, after the loss is evaluated for a mini-batch of training samples, all the network parameters are updated simultaneously. This implies that every layer is updated under the assumption that the other layers remain unchanged, which isn't the case. For this cause, the distribution of input features for each network layer is likely to change everytime a new mini-batch arrives, which provokes the network parameters to oscillate quite a bit at the beginning stages of the training process. This undesired distribution change in the input features is also known as the internal covariate shift.

To mitigate this issue, [IS15] propose to standardize per mini-batch the input features before they are fed into activation layers. Specifically, a batch normalization layer calculates per channel the mean and standard deviation of the input features across all samples in a mini-batch. Separately for each channel, the operation then rescales the input activation values to have a mean of zero and a standard deviation of one. To enable better flexibility in pratice and allow the subsequent layers to work on features that display their preferred statistics, extra learnable parameters $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_C)^\intercal$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_C)^\intercal$ were introduced. The formulation for a batch normalization on channel c is therefore

$$\mathbf{G}[c,u,v] = \frac{\mathbf{F}[c,u,v] - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} \cdot \gamma_c + \beta_c, \tag{2.10}$$

where $\mathbf{G}$ and $\mathbf{F}$ are the input and output features for a particular sample, u and v are coordinates on the spatial axes, ϵ is a small value added for numerical stability, μ_c and σ_c are the mean and standard deviation of the respective channel. Additionally, since at test time the availability of mini-batch statistics can not be presumed, it is common practice to maintain a running mean and variance on the training samples and then apply these statistics on test samples. As has been proved by a wide range of research, the application of batch normalization layers is able to prevent the input statistics for convolution layers from changing drastically between subsequent mini-batches and thus noticably stabilize the training process.

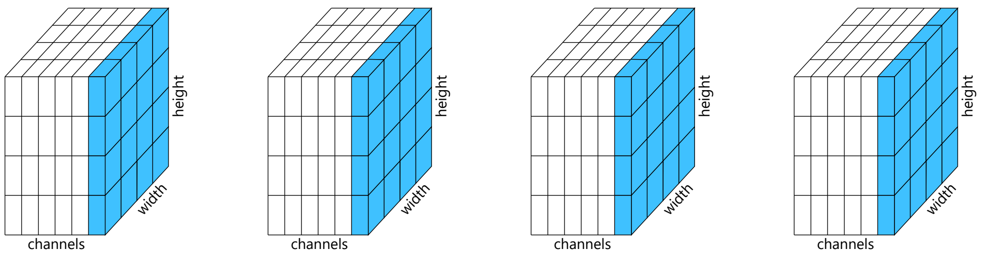


Figure 2.5: Example of a batchnorm layer working on a mini-batch of four samples. Note that the normalization is performed across all pixels on the same feature channel over all samples. Different feature channels are handled independently.

2.2.4 Common Architectures

The operations introduced up till this point can already make up deep convolutional architectures capable of a learning capacity to handle a variety of visual recognition tasks. In the following we will briefly discuss about a few classic models.

The VGG-Net presented by [SZ14] was among the very first attempts to apply convolutional neural networks for large-scale image classification. A visualization of the model is given in Figure 2.6. The authors of this paper append to the end of the convolutional stage of the network a number of fully connected layers. These layers work on the flattened feature output of the last convolution layer and separately predict a probability on the class membership of the input image for each potential class. Compared to convolutional models proposed by earlier works, VGG-Net made two novel design choices that enabled a breakthrough in the benchmarked classification accuracy on imagenet [DDS$^+$09]. On one hand, the depth of the architecture is more than doubled compared to the previous AlexNet model [KSH12] thanks to the adoption of rectified linear functions. On the other hand, only 3×3 kernels are utilized for a VGG-Net's convolution layers, which greatly reduces the number of trainable parameters in the model and speeds up the training process. But thanks to its great depth, the model is still able to maintain a large receptive field for neurons at the back. These design principles became indeed the standard for future architectures.

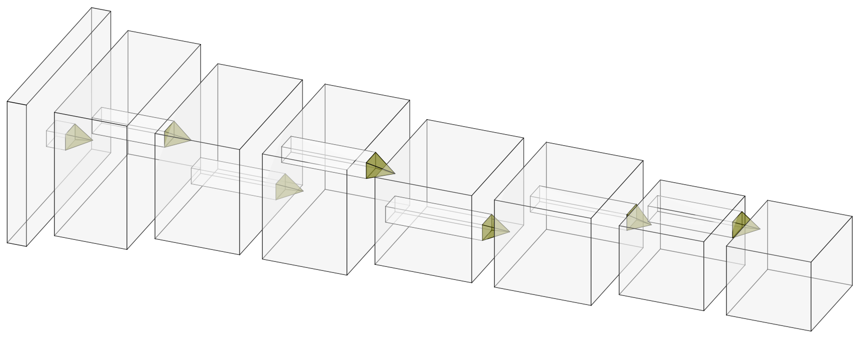


Figure 2.6: One variation of VGG-Net that contains 11 trainables layers. Each cuboid represents either the initial image or the multi-channel feature maps output by the convolution and max-pooling layers. The spatial and channel dimensions are displayed on logarithmic scales. The fully connected layers at the end of the model are omitted.

A larger number of stacked layers may enhance the expressiveness of a convolutional network due to an enlarged parameter space, but only up to a certain point. Over that mark, a further increase in depth begins to damage the performance because gradients start to vanish for layers in the front. This was the bottleneck for VGG-like structures and was addressed by [HZRS16] via the introduction and stacking of residual blocks (see Figure 2.7 for an example). In a residual block, features either pass through a few convolution layers in succession, or totally ignore these layers and only go through a 1×1 convolution for channel alignment purposes. The features that reach the end of both paths are simply joined via a pixelwise summation. Despite its concision, this structural modification has proven to be effective in countless experiments. The additional path, referred to as a skip connection, allows the gradients to propagate backwards without rapidly shrinking in magnitude, which enables the layers to the front of the network to keep on updating itself even when the evaluation loss comes down to a small value as the training proceeds. Taking full advantage of this property, the standard ResNet architecture was designed to be made up of only deep chain structures that each contains a sequence of residual blocks and led to significant advances on various recognition benchmarks.

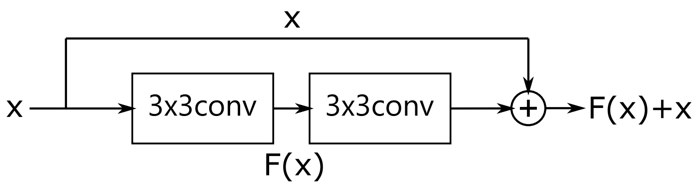


Figure 2.7: Example of a residual block implemented in ResNet. The additional skip connection at the top allows the model to learn only the residual part of an underlay target function.

But convolutional networks are more than just powerful feature extractors. For a wide range of applications including general object detection, image segmentation and human keypoint detection, specialized convolutional architectures that are adapted from classic model designs can be trained and applied at inference time in an end-to-end manner. One example of this is the fully convolutional network proposed by [LSD15] to take on the task of semantic image segmentation.

In this task, the goal is assign a class label for each pixel on a given image. However, since the features that pass through a convolutional network is typically downscaled multiple times thanks to pooling layers and convolutions with stride, giving a pixelwise prediction is not straightforward. Long et al. propose to tackle this problem by upsampling the feature maps with transposed convolution layers. A transposed convolution does the exact opposite of a normal convolution and increases the spatial resolution of a feature map when a fractional value is set as the stride. By stacking multiple such layers in a row, a prediction map that is of the same resolution as the original image can be reconstructed at the end of the network. Pixelwise loss evaluation is then carried out to offer guidance for the parameter update. Taking inspiration from [LSD15], a large number of similar network designs were proposed later for visual recognition topics that demand dense prediction maps.

3 Related Work

In this chapter we will address the various topics closely related to our work. We begin this chapter with a glance into the more well-studied problem of monocular 2D human keypoints detection. We then delve into the focus of this work, which is monocular 3D human pose estimation. Properly solving this 3D subproblem entails dealing with a number of issues that come along with the added dimension, and we will show how a number of previous works approach this topic. In particular, we will introduce the work [SLAL20] by Sarandi et al. in this section, which lays the foundation for this master thesis. We conclude this chapter with a view into the realm of privileged information learning and, specifically, how this concept has been applied to enable the proper handling of several visual recognition tasks.

3.1 Monocular 2D Human Keypoints Detection

In monocular 2D human keypoints detection, an algorithm predicts an image-space coordinate for each skeletal keypoint of all person instances within a single image. This task bears some resemblance to semantic image segmentation [CPK$^+$14], in the sense that it requires a dense prediction for each pixel on the input image. Nevertheless, a key difference lies in the fact that in 2D human keypoints detection, when multiple subjects jointly appear in the same image, an assignment of each detected keypoint to one of the person instances is needed. In general, the majority of recent methods fall into two categories in their approach to this assignment problem.

3.1.1 Top-down Approaches

The first family of methods circumvent an association attempt as a whole. They employ a pipeline that consists of a detection stage and a separate pose inference stage. The detection stage gives a bounding box for each person instance and then crops out the image content within. The second stage takes this image patch and, under the presumption that only a single person is present, gives a single coordinate prediction for each joint. One classic implementation of such a strategy is the work by Papandreou et al. [PZK+17]. In the first stage, they directly apply a *Faster R-CNN* [RHGS15] detector to generate person detection boxes. In the second stage, they propose a fully convolutional ResNet architecture that, based on box proposals delivered by the first stage, predicts jointwise heatmaps and 2D offset-maps. Ideally, a heatmap should store the probabilities with which the respective joint falls within each pixel's vicinity, whereas the 2D offset-map is supposed to store per-pixel a 2D vector that points from that pixel to the groundtruth joint location. At inference time, these two maps are fused into a highly localized activation map via a bilinear voting procedure. An illustration of this voting process is given in Figure 3.1. The peak locations of these final activation maps are then determined and assembled as the final pose prediction.

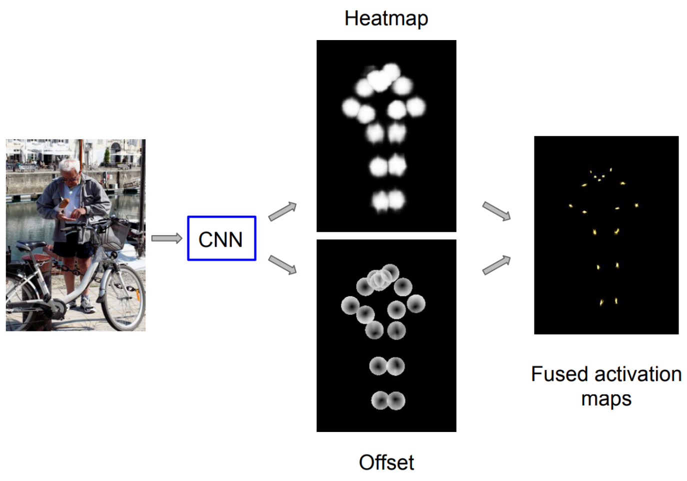


Figure 3.1: Voting procedure proposed by Papandreou et al. [PZK+17]. Every heatmap pixel casts a vote to the specific sub-pixel location its offset vector points to, with a weight equal to its confidence score. The pixel values on the final activation maps are then computed via bilinear interpolation.

3.1.2 Bottom-up Approaches

The second group of methods address the keypoint assignment problem in a bottom-up fashion. In a first step, they detect and gather keypoint candidates of every type, regardless of their ownership. In the second phase, they try to establish connections among these keypoint candidates, such that each group of associated keypoints form a full skeleton instance. The *OpenPose* keypoint detector [CSWS17] proposed by Cao et al. is a state-of-the-art embodiment of this design paradigm. They build their work off the back of [WRKS16] and make use of a fully convolutional architecture that consists of multiple prediction stages. Each prediction stage tries to refine the predictions of its predecessor. Figure 3.2 visualizes their model design. The novel aspect of this work is that they propose to regress what they call 2D part affinity fields on top of the usual jointwise heatmaps. Each part affinity field correspond to a single arc on the kinematic tree, which in most cases represent a bone that connect two joints. According to their design, each spatial location on a part affinity field stores either the null vector or a unit vector that points from the start of the target bone to its end, provided that the location is covered by an instance of that bone in the given image. In the second stage of the inference pipeline, they compute association scores between each pair of candidate joint instances that share a bone in the kinematic tree. This is achieved by calculating a path integral between the two joint candidates on the target bone's part affinity field. Based on these association scores, the hungarian algorithm is applied to establish optimal matches between joint candidates on either end of a target bone. These confirmed bone instances then go on to make up a full skeleton instance for each subject. Notably, their method can easily be extended to incorporate depth input like in [MGVCO18] and achieve comparable performances.

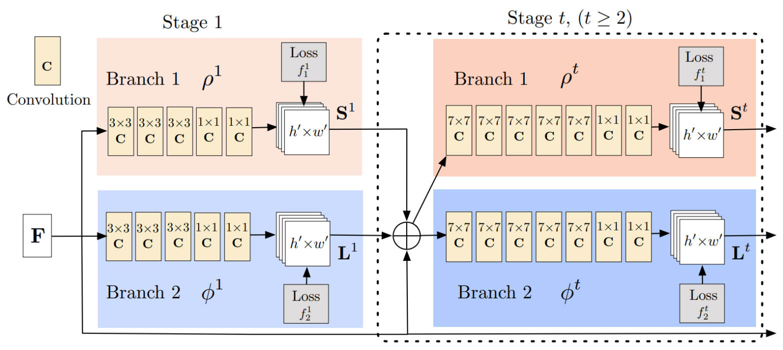


Figure 3.2: Model architecture employed by Cao et al. [CSWS17]. The multiple two-branch prediction stages feed on general features as well as jointwise heatmaps and part affinity fields output by the previous stages. This repeated design pattern gradually refines the dense prediction quality.

3.2 Monocular 3D Human Pose Estimation

The transition from image-space to camera-space makes monocular 3D human pose estimation a much harder task than its 2D counterpart in a number of aspects:

- The added degree of freedom for each skeletal keypoint leads to a significantly larger solution space.
- 3D points in the camera coordinate system and 2D locations on an image only satisfy a loose form of spatial correspondence.
- Candidate poses of different scales and at different distances to the camera may project to the same image.
- Different articulations of the same person at the same distance to the camera may project to the same image as well.

3.2.1 Direct Coordinate Regression

Several methods have been developed in recent years that build on the rapid improvement in image-space estimation since the adoption of deep convolutional networks. Dabral et al. [DGM$^+$19] attach a 3D lifter module to the back of a standard Mask R-CNN [HGDG17] framework. The 3D lifter module [MHRL17] is a simple multi-layer perceptron that, agnostic to image features, converts 2D image coordinates to a 3D root-relative skeleton. Figure 3.3 illustrates their approach. Similarly, Zhou et al. [ZHS$^+$17] propose to regress 2D keypoint activation maps as a first step. These activation maps are then combined with general features and fed as input to a depth regression module that outputs root-relative depth values for each joint. The work by Mehta et al. [MSM$^+$19] also falls into this category. They install a second branch onto the 2D keypoint detection framework in [CSWS17] that predicts a 3D bone vector field for each joint. This vector field is supposed to encode 3D metric-space displacement vectors that connect a joint to its immediate neighbors on the kinematic chain. At inference time, they lookup the displacement vectors stored at the respective pixel locations predicted for each joint and deliver this information to the fully connected lifter module as additional evidence.

3.2.2 Volumetric Heatmaps

Methods described above presume a strong 2D keypoint detector as their foundation. In practice however, scene background variations and occlusion may cause the 2D keypoint detection to fail, thus lead the 3D estimation astray. The other group of methods try to circumvent numerical coordinate regression as a whole. To this end, they discard the 3D lifters and instead train a fully convolutional network to predict dense localization probabilities in a discretized space.

[PZDD17] is amongst the first approaches to test out the possibilities of a 2.5D volumetric heatmap representation. In such a representation, each discrete slice along the z-axis is a 2D confidence map spatially aligned with the input image. The assoicated slice index, however, translates to an actual z-coordinate

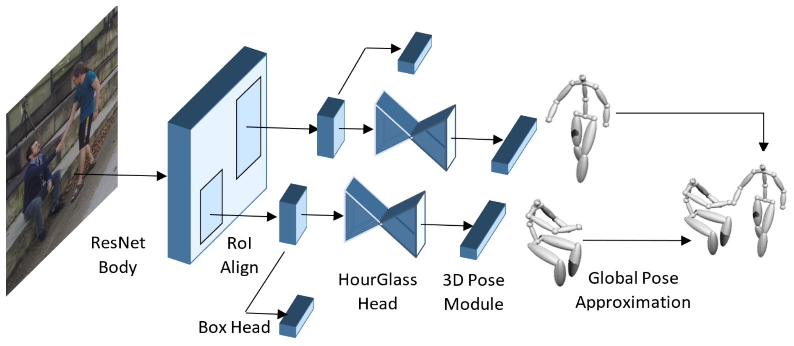


Figure 3.3: 3D pose estimation framework proposed by Dabral et al. [DGM+19]. In order to directly regress 3D root-relative coordinates, a fully connected lifter module is appended to the end of the Mask R-CNN branch for 2D keypoint detection.

in camera-space. They train a stacked hourglass architecture to produce such volumetric representations at multiple prediction stages, each time with an increase in depth resolution. At test time, the grid location where the peak value resides on each heatmap is taken as the prediction for that specific joint. Sun et al. [SXW+18] build on their idea and introduce the soft-argmax operator into the framework. This fully differentiable operator computes an average of all voxel grid coordinates as the final prediction, weighted by the activation values stored in the respective heatmap entries. Such a practice lifts the limitation on inference precision caused by a finite heatmap resolution and allows for an end-to-end framework at both train-time and test-time.

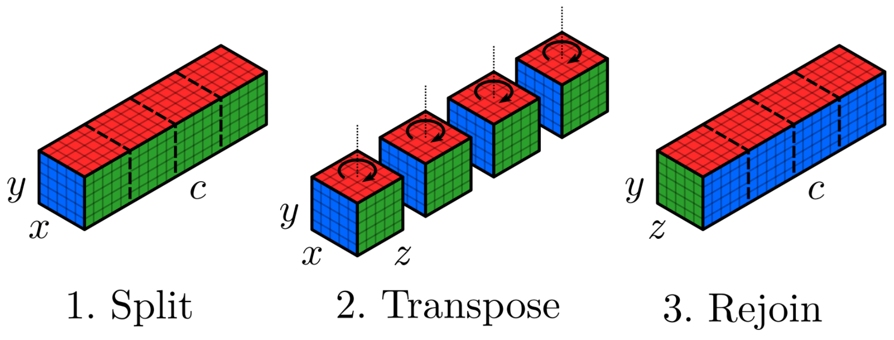


Figure 3.4: Nibali et al. [NHMP19] use axis permutation to transition between different views of the same feature blob.

The marginal heatmap regression framework proposed in [NHMP19] is another example of this genre. They propose to predict marginal heatmaps that encode the same localization information as the volumetric representation in [PZDD17], but induce lower computational expenses. In their design, the features produced by the backbone of the fully convolutional architecture undergo axis permutation before they are delivered to the respective marginal heatmap regression modules. Figure 3.4 visualizes such an operation. This axis permutation enables a transition into separate views of the same feature blob and essentially allows for the backbone to aggregate depth cues into the feature maps. At inference time, joint coordinates for each axis are computed from the marginal heatmaps via again the soft-argmax operator.

The aforementioned approaches still face limitations since they do not directly address the issue of scale ambiguity. In particular, the predicted x- and y-coordinates lie in image-space, whereas the root-relative z-coordinate lies in metric-space. The true depth of the root joint and the camera intrinsic matrix are therefore required for the full reconstruction of the root-relative skeleton. Under circumstances where root joint information isn't available, its depth value can only be determined in least squares fashion based on empirical statistics about bone length. Such anthropometric heuristics fail, however, in the presence of subjects of diverse heights. As such, Istvan et al. [SLAL20] propose to regress a volumetric heatmap whose entries directly correspond to metric-space grid voxels. That is, the central coordinate of each grid cell is measured in millimetres on all three dimensions. Such a design not only encourages the convolutional network to extract scale and depth information based on appearance cues, but it also opens the possibility for the model to implicitly reason about joints that fall outside the field of view of the camera. We take their method as the foundation of our pose estimation framework. A detailed desciption of their inference pipeline is given in Section 4.1.

3.3 Privileged Information Learning

In the context of machine learning, privileged information refers to the train-time-only soft labels provided by a teacher model to a student model. These additional informative cues encapsulate how the teacher manipulates the various resources at its disposal and should, ideally, aid the student in achieving a similar level of expertise in the learning task at hand that is not attainable otherwise. In a sense, the knowledge distillation framework [HVD15] proposed by Hinton et al. is an instance of this learning paradigm, with a special focus on model compression. In particular, they let a teacher model, compute-intensive, share its intermediate results every step of way through its task handling process. In the meanwhile, a student model, less complex, try to closely replicate these mid-level outputs when given the same task input. For deep learning models, this stepwise guidance allows the student to closely mimic the teacher's behaviour and converge to better solutions on its own parameter space.

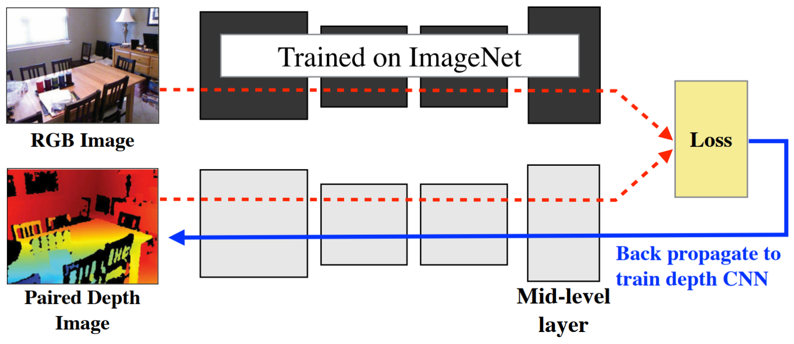


Figure 3.5: Supervision transfer framework proposed by Gupta et al. [GHM16]. Supervision is transferred from one modality to another for which there are paired images.

3.3.1 Supervision Transfer

And yet there's more possibilties. Especially under the circumstance where there is an inconsistency in the input modalities between train-time and test-time, privileged information learning can be applied to transfer knowledge between models that accept distinct inputs. [GHM16] is such an example. They introduce the concept of supervision transfer, which caters to the demand on deep learning models that work with modalities with scarce annotations. A graphical illustration for their method is given in Figure 3.5. Specifically, they assume the existence of a teacher network that is able to produce discriminative features for images from a source modality with rich annotation. To transfer the supervision to a target modality, where only a small set of labeled images are present, they prepare a wide collection of unlabelled image pairs of either modality that describe the same scenes. For each image of the source modality in the collection, they save the mid-level activation maps the teacher network produces as soft labels. These soft labels are then delivered as supervision signal to a second model, who perceives exclusively the counterpart images of the target modality and try to reproduce the associated features at some point in the architecture. The resultant model can subsequently be finetuned on a small set of labeled images from the target modality and considered ready for deployment. They test out the idea on the task of general object detection and conclude from their results that:

- The target model trained with the supervision transfer technique is able to produce feature representations that possess similar discriminative power to that of its teacher.

- The learned weights on the target model serve as a great starting point for further finetuning procedures on the target modality.

3.3.2 Knowledge Transfer

The same strategy is employed in [HGD16] [GMM18] but with a different emphasis. They try to prove that there is still value in exploiting the information carried by images from train-time-only modalities, even if there is sufficient annotation for the target modality. [GMM19] takes this argument a step further by incorporating the concept of adversarial learning into their training framework. Figure 3.6 illustrates their idea. They deal with the task of video classification via temporal convolutional networks [FPW17]. To facilitate efficient knowledge transfer between a teacher and a student that accept input images of different modalities, they introduce a separate descriminator network that plays an adversarial game with the student. Specifically, the student plays the role of the generator in a conventional adversarial learning setup and attempts to generate feature vectors that closely resembles those produced by the teacher. The discriminator, on the other hand, is instructed to classify whatever input feature vector with respect to an extended class pool. That is, the discriminator gives either a class label prediction, when it believes that the input vector was produced by the teacher, or a fake label, if it is convinced that the vector is a fake one generated by the student. As has been proven in their experiments, this unique design allows the student to extract a feature representation that not only entails visual cues from an inaccessible modality but also possesses rich discriminative power.

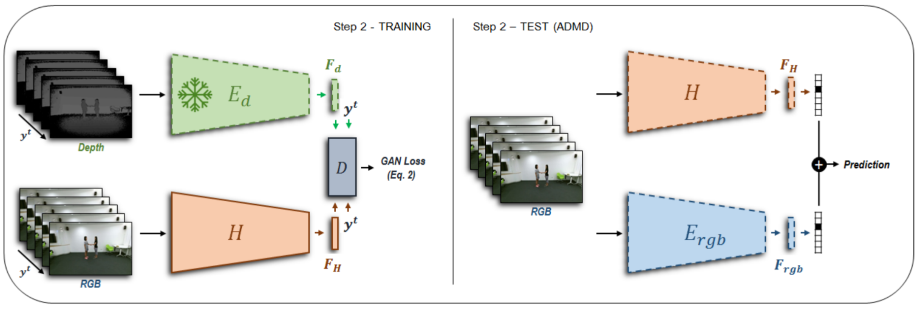


Figure 3.6: Adversarial discriminative modality distillation framework proposed by Garcia et al. [GMM19]. The conventional entrywise distillation loss is replaced by an adversarial game.

4 Method

In the following, we explain the various aspects of our proposed method for monocular 3D human pose estimation. We begin this chapter with the description of the baseline approach. Then we explain in detail how we propose to utilize the depth images available at train time to improve this baseline via privileged information learning. At last we address the multiple possibilities to modify the estimation framework in an attempt to further enhance training efficacy.

4.1 Baseline Approach

We follow [SLAL20] and address the task of monocular 3D human pose estimation by training a fully convolutional neural network to directly regress, for each skeletal keypoint, a volumetric heatmap in metric space.

4.1.1 Network Input

The baseline method take as input a tuple that consists of an RGB image, the bounding box of the person in question on that image and the intrinsic matrix of the camera that took the picture. For training samples the tuple additionally includes a ground-truth 3D skeleton in world-space and the validity masks that describes whether each skeletal keypoint is within the field of view of the camera.

Directly cropping out the image area enclosed by the given bounding box and feed the patch into the network is an option, but such a practice robs the model of the perspective information crucial for prediction accuracy. For example, the true camera-space pose of a guy who stands near the edge of the image needs to be rotated by a small angle to align perfectly with the pose of the same guy when he stands right at the center, even if he faces the camera and records the exact same pose in both cases. The algorithm should be able to notice the difference between the two and act accordingly.

To address this, we follow [SLAL20] and perform a homographic transformation on the input image. Specifically, we want to know how the image would look if the camera were to face the center of the subject when the picture was taken. To do this, we first need to know what the world-space direction of the three axes of the camera coordinate system would be if the camera were rotated towards the subject. Let $\mathbf{K}$ be the intrinsic matrix of the orginal camera configuration and $\mathbf{R}$ be the rotation matrix that describes how to represent a world-space vector under the camera coordinate system. We may compute the world-space vector that points from the optical center of the camera to the 3D position of bounding box center $(u^{center}, v^{center}, 1)$ as

$$\mathbf{a}^{center} = \mathbf{R}^\intercal \mathbf{K}^{-1} \cdot \begin{pmatrix} u^{center} \\ v^{center} \\ 1 \end{pmatrix}. \tag{4.1}$$

Since we want the camera to face the center of the subject after the rotation, we take the normalized version of this vector as the new optical axis of the camera:

$$\mathbf{a}_z^{new} = unit(\mathbf{a}^{center}) = \frac{\mathbf{a}^{center}}{\|\mathbf{a}^{center}\|_2}, \tag{4.2}$$

where $unit(\cdot)$ is the function that normalizes a vector by its euclidean norm.

Note that this rotation should not change the camera's upward-direction $\mathbf{w}$, otherwise the rotation may not be uniquely defined. It is then straightforward to define the world-space direction of the other two axes based on the right-hand rule via

$$\mathbf{a}_x^{new} = unit(\mathbf{a}_z^{new} \times \mathbf{w}) \tag{4.3}$$

and

$$\mathbf{a}_y^{new} = unit(\mathbf{a}_z^{new} \times \mathbf{a}_x^{new}). \tag{4.4}$$

These directional vectors constitute the rotation matrix $\mathbf{R}_{new}$ of the updated camera configuration, which, along with the static camera position $\mathbf{t}$, describes how to transform a world-space coordinate $\mathbf{p}_{world}$ to a coordinate $\mathbf{p}_{camera}$ in the updated camera's coordinate system:

$$\mathbf{p}_{camera} = \mathbf{R}_{new}(\mathbf{p}_{world} - \mathbf{t}), \tag{4.5}$$

where

$$\mathbf{R}_{new} = (\mathbf{a}_x^{new}, \mathbf{a}_y^{new}, \mathbf{a}_z^{new})^\intercal. \tag{4.6}$$

This is exactly what we do to the world-space groundtruth skeleton that is provided for each training sample, in order to obtain the camera-space coordinates $(\mathbf{p}_1^*, \mathbf{p}_2^*, \ldots, \mathbf{p}_J^*)$ that we need later for supervision purposes.

Now that we have obtained the extrinsic parameters for our updated camera configuration, we have all the resources needed for the homographic transformation. If we simply allow the intrinsic matrix $\mathbf{K}$ to remain unchanged, a pixel location $(u^{new}, v^{new}, 1)^\intercal$ on the projected image would correspond to pixel location

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{R}\mathbf{R}_{new}^\intercal\mathbf{K}^{-1} \cdot \begin{pmatrix} u^{new} \\ v^{new} \\ 1 \end{pmatrix} \tag{4.7}$$

on the original image. The projected pixel value can then be simply determined via a bilinear interpolation over adjacent pixel values near $(u, v, 1)^\intercal$ on the original image. To retain consistency, the image-space coordinate of the bounding box vertices should be reprojected in the same manner.

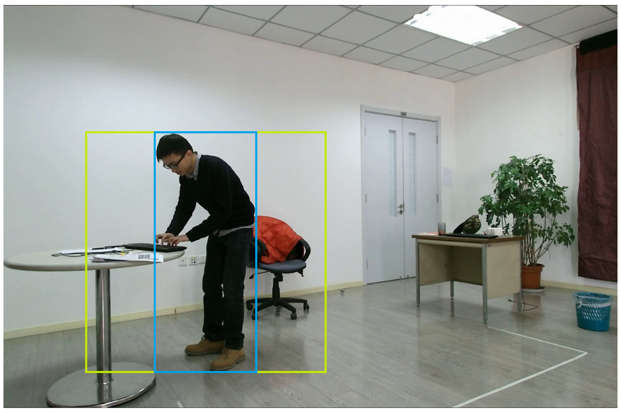


Figure 4.1: Example of a training sample. The blue rectangle is the given bounding box. The square box encloses the image area that is actually cropped out.

After this homographic transformation, we crop out the smallest concentric square image area that covers the reprojected bounding box and resize this image patch to shape 257×257 before it is fed into the convolutional network (see Figure 4.1 for an example). For a person standing nearly upright, this cropping strategy allow us to include more background context in the input image without any sacrifice in spatial resolution. The input shape is chosen so that convolution layers with stride greater than one are able to symmetrically sample the feature maps at all stages of the network, which should allow the model to learn a evenly distributed feature representation without a problem at the image borders. Optionally, various image augmentation steps may come in either before or after the cropping operation takes place. These augmentation schemes will be discussed in Subsection 4.1.4.

4.1.2 Network Design

Previous works [XWW18] [BMB20] in various fields of computer vision have shown that high resolution features are crucial for deep convolutional networks to be able to output fine prediction maps. As such, we adopt the standard ResNet-50 architecture [HZRS16] as the backbone of our fully convolutional network, with the modification that the strided convolution in the first residual block of the last chain structure is replaced by an atrous convolution [CPK+17] with a dilation rate of 2. This replacement results in an effective stride of 16 instead of 32 for the entire model, despite the fact that the receptive fields of neurons behind this convolution layer remain unchanged. As a result, the last feature map produced by the backbone of the model finds a doubled spatial resolution at 17×17. Based on this feature map, our baseline model predicts a score tensor of shape $JD \times 17 \times 17$, where J is the number of required skeletal keypoints and D is the level of discretization in the depth dimension. This is achieved by simply attaching a 3×3 convolution with no stride to the end of the backbone as the regression head. We initialize our network with the weights of a standard ResNet-50 architecture pretrained on ImageNet, with the exception that the convolutional regression head is initialized randomly. A graphical depiction of our fully convolutional network is given in Figure 4.2.

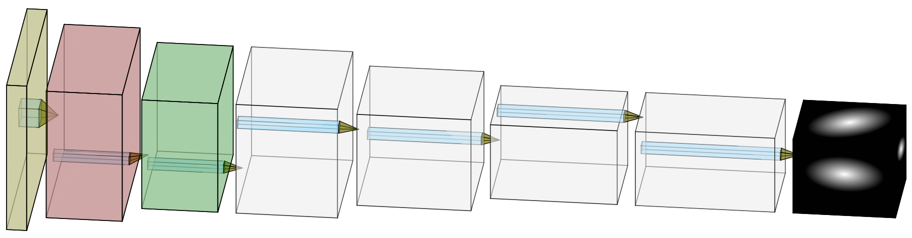


Figure 4.2: The baseline model based on the standard ResNet-50 architecture. The light-yellow cuboid represents an input image of shape $3 \times 257 \times 257 \times 3$. A 7×7 convolution and a 3×3 max-pooling both with stride 2 convert this image into a $64 \times 65 \times 65$ feature map (light green cuboid). This feature map is fed into four consecutive sequences of stacked residual blocks that extracts rich features of shape $2048 \times 17 \times 17$. Based on these features, the convolutional regression head predicts a volumetric heatmap (last cuboid) separately for each skeletal keypoint.

4.1.3 Prediction and Error Function

We rearrange this score tensor into J volumetric heatmaps $\mathbf{H}_j^{score}$, each of shape $17 \times 17 \times D$, and pass them individually into a 3D softmax operation to obtain the confidence maps $\mathbf{H}_j^{conf}$ for each joint $j \in J$:

$$\mathbf{H}_j^{conf}[u,v,d] = \frac{\exp\left(\mathbf{H}_j^{score}[u,v,d]\right)}{\sum_{u'=1}^{17}\sum_{v'=1}^{17}\sum_{d'=1}^{D} exp(\mathbf{H}_j^{score}[u',v',d'])}. \tag{4.8}$$

Semantically, each entry value in volumetric confidence map $\mathbf{H}_j^{conf}$ represents the predicted probability with which the skeletal keypoint $j \in J$ falls into the voxel in camera-space that corresponds to that particular entry. Utilizing these confidence scores, we may compute the expected normalized camera-space coordinate $\bar{\mathbf{p}}_j = (\bar{x}_j, \bar{y}_j, \bar{z}_j)^\intercal$ for each joint $j \in J$ via the soft-argmax operator. This operator essentially accumulates the 3D coordinates that correspond to the centers of each grid voxel within a unit cube, weighted by the normalized confidence scores stored in the associated confidence map entries:

$$\begin{aligned}
\bar{x}_j &= \sum_{u'=1}^{17}\sum_{v'=1}^{17}\sum_{d'=1}^{D}\mathbf{H}_j^{conf}[u',v',d'] \cdot \frac{u'-1}{17-1}, \\
\bar{y}_j &= \sum_{u'=1}^{17}\sum_{v'=1}^{17}\sum_{d'=1}^{D}\mathbf{H}_j^{conf}[u',v',d'] \cdot \frac{v'-1}{17-1}, \\
\bar{z}_j &= \sum_{u'=1}^{17}\sum_{v'=1}^{17}\sum_{d'=1}^{D}\mathbf{H}_j^{conf}[u',v',d'] \cdot \frac{d'-1}{D-1}.
\end{aligned} \tag{4.9}$$

Since we are only concerned with root-relative pose prediction, these normalized coordinates are then converted into actual camera-space coordinates within a $2000mm \times 2000mm \times 2000mm$ prediction volume, which should be suffiently large to contain the full skeleton of a person under all body configurations. This is done by simply scaling each component of the normalized coordinates by a factor of 2000:

$$\mathbf{p}_j = 2000\bar{\mathbf{p}}_j. \tag{4.10}$$

To obtain the final root-relative pose estimation $\hat{\mathbf{P}} = (\hat{\mathbf{p}}_1, \hat{\mathbf{p}}_2, \ldots, \hat{\mathbf{p}}_J)$ for evaluation purposes, we subtract the coordinate of each joint by the position of the root joint, typically the pelvis:

$$\hat{\mathbf{p}}_j = \mathbf{p}_j - \mathbf{p}_J. \tag{4.11}$$

These root-relative predictions are then evaluated against the groundtruth skeleton, which produces a single loss value to optimize:

$$L_{regression} = \frac{\sum_{j=1}^{J} m_j \cdot \left(l(\hat{x}_j, \hat{x}_j^*) + l(\hat{y}_j, \hat{y}_j^*) + l(\hat{z}_j, \hat{z}_j^*)\right)}{3\sum_{j=1}^{J} m_j}, \tag{4.12}$$

where

$$l(a,b) = \begin{cases} 0.5(a-b)^2 & |a-b| < 1.0 \\ |a-b| - 0.5 & |a-b| \geq 1.0 \end{cases} \tag{4.13}$$

is the huber loss, $\hat{x}_j^*$, $\hat{y}_j^*$ and $\hat{z}_j^*$ are the target root-relative coordinates, m_j is the boolean validity mask indicating whether joint j is within the field of view of the camera or not. Note that this design does not impose any explicit contraint on the distribution of the volumetric heatmaps. This allows the network to learn implicitly how to generate a set of probability distributions that would best predict the relative positions between each skeletal keypoint and the root joint. Moreover, all operations described above are easy to implement in a fully differentiable manner, so that gradients can be computed following the chain-rule using the standard backward propagation solver implemented in modern deep learning engines.

4.1.4 Augmentation Schemes

To account for the potential distribution changes in lighting conditions, bounding box quality and camera angles at inference time, we employ multiple image augmentation schemes throughout the training stage.

Color Augmentation Color augmentations simulate the expected lighting variations that may happen at test time. We successively induce random distortions on the brightness, contrast, hue and saturation values of a training image. Specificly, for an input image $\mathbf{F}$ in normalized RGB color space, a brightness augmentation adds a common random noise subject to a uniform distribution to all pixel values across the three image channels:

$$\mathbf{F} = \mathbf{F} + u^{brightness}, \quad u^{brightness} \sim U[-0.125, 0.125]. \tag{4.14}$$

A contrast augmentation removes 0.5 from all pixel values, scales all pixels by a random factor subject to a uniform distribution, then adds back the previously deducted amount:

$$\mathbf{F} = (\mathbf{F} - 0.5) \cdot u^{contrast} + 0.5, \quad u^{contrast} \sim U[0.8, 1.25]. \tag{4.15}$$

We convert the image to the HSV color space before the other two augmentation steps are applied. The HSV color space describes an image in terms of hue, saturation and luminance values and thus gives an interpretation that de-correlates color information from lighting intensity. Under this setup, a hue augmentation adds a common random noise to all pixel values across the hue channel of the image:

$$\mathbf{F}^{hue} = \mathbf{F}^{hue} + u^{hue}, \quad u^{hue} \sim U[-18, 18]. \tag{4.16}$$

Similarly, a saturation augmentation works exclusively on the saturation channel and scales all pixel values by a random factor:

$$\mathbf{F}^{saturation} = \mathbf{F}^{saturation} \cdot u^{saturation}, \quad u^{saturation} \sim U[0.8, 1.25]. \tag{4.17}$$

Geometry Augmentation Geometry augmentations prepare the model for possible variations in camera angle and bounding box distribution at test time. We apply random rotation, flip and zoom on our camera configuration after we turn the camera towards the center of the subject, but before the image reprojection and the calculation of camera-space groundtruth takes place. This specific order by which the operations are performed ensures that the supervision we impose on our convolutional model remain consistent with the input image.

Given the camera's current rotation matrix $\mathbf{R}$, a rotation augmentation randomly rotates the camera around its optical axis by a small angle θ that obeys the random uniform distribution

$$U[-\frac{\pi}{12}, \frac{\pi}{12}]. \tag{4.18}$$

The updated rotation matrix $\mathbf{R}_{new}$ of the camera can be computed as

$$\mathbf{R}_{new} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{R}. \tag{4.19}$$

A flip augmentation randomly flips the camera horizontally for each training image with a 50 percent chance. Internally, this is achieved by multiplying the first row of the rotation matrix $\mathbf{R}$ of the camera by -1:

$$\mathbf{R}_{new} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{R}. \tag{4.20}$$

This would result in a mirrored version of the usual image reprojection described by Equation (4.7). But additional attention is needed on the retrieval of camera-space groundtruth coordinates for a flipped sample. In this case, after Equation (4.5) is applied, coordinates for symmetric keypoints on either side of the skeletal tree are swapped to make sure that the mirrored supervision semantically complys with the input image. The validity masks are to be updated as well.

Since the principle point of the camera coincides with the center of the bounding box after the homographic transformation, a random zoom augmentation can be achieved by simply rescaling the focal lengths in the camera's intrinsic matrix by a random factor:

$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \cdot u^{zoom}, \quad u^{zoom} \sim U[0.9, 1.1]. \tag{4.21}$$

Since the cropping area stays unchanged afterwards, this essentially leads to image crops that either contain more background information at the price of a lower effective resolution, or exclude part of the person's outline.

4.2 Depth as Privileged Information

In this work, we build on the baseline method described in the previous section by incorporating depth input into the models's learning process. We imagine that the additional depth cues would inform the model of the relative distance of various body parts to the camera, thus aid the deeper neurons in understanding the general pose configuration of the subject. Moreover, due to probable limitations in real-world application scenarios, a dependency on depth sensors at inference time shall be preferably avoided. Therefore, a crucial aim of this study is to find a way for the model to be able to effectively extract such depth cues from ordinary RGB images. Inspired by the recent success of [HGD16] [GMM18] and made possible by the release of two large-scale action recognition datasets [LSP+19] [LHL+17], we present a novel 2-stage training scheme that exploits the very idea of privileged information learning. In our learning paradigm, a teacher model that is capable of extracting rich feature representations from a pair of RGB and depth images gives out a piece of advisory information for each sample, in an attempt to transfer this valuable knowledge to a student model. The student, on the other end of the framework, receives this guidance message along with an RGB input and tries to figure out just what extra depth information could be inferred about a given sample that would be beneficial for the pose regression task at hand, if an associated depth image were present. More details of our training scheme is explained in the following subsections.

4.2.1 First Stage

In the first stage of our priviledged information learning setup, we train, in the presence of fully annotated RGB-D image pairs, a teacher network that is adapted from our baseline model to estimate 3D human poses. Specifically, we mount an additional depth branch onto our baseline convolutional network that is parallel to the front part of the existing backbone. This new branch takes as input a single-channel depth image of the same spatial resolution as the input RGB image. To allow the network to handle this input format, the usual convolution kernel of shape $64 \times 3 \times 7 \times 7$ at the front of the standard ResNet-50 architecture is replaced by a new kernel of shape $64 \times 1 \times 7 \times 7$. The rest of this new branch duplicates the standard configuration up till the second chain structure of residual blocks, after which the two branches are joined together via a fusion module. Here we concatenate the features output by both branches along the channel dimension and then cut down the number of feature channels by half via a 1×1 convolution. This design choice closely follows [OVBG19] [LPV+20] and encourages both branches of the model to work independently on their own input modality and produce complementary feature representations. Whereas a number of alternative options are applicable in this circumstance, selecting the best fusion module in practice is not the core focus of our work. The rest of the teacher network (the trunk part) copies its configuration from the baseline model and is optimized in exactly the same manner. An illustration of the teacher model's training scheme is given in Figure 4.3.

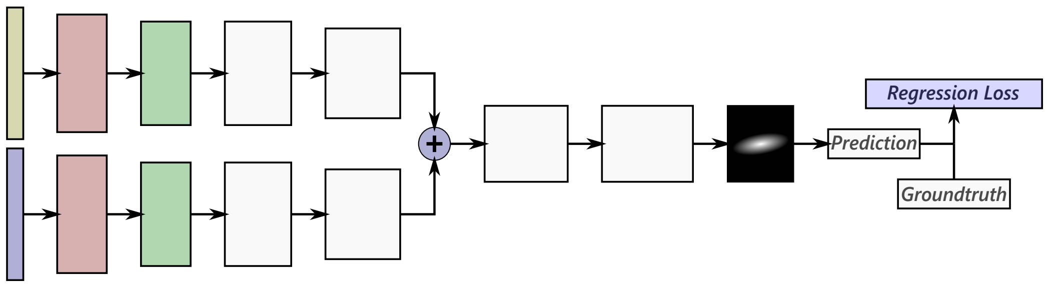


Figure 4.3: The first stage of our training scheme. The spatial and channel dimensions are displayed on logarithmic scales. The depth branch in the bottom copies its configuration from the standard ResNet-50 except for the first convolution layer and takes as input a single-channel depth image (light-blue rectangle at the front). The circle with a plus sign represents our fusion module which consists of a concatenation operation and a 1×1 convolution layer for the purpose of channel alignment. The same supervision target as the baseline model is set.

One crucial aspect of optimizing such a bi-modal convolutional network is the initialization of its parameters. A common practice today is to load the weights taken from a standard ResNet-50 model pretrained on ImageNet [DDS+09]. In practice however, this default option may not be especially optimal. Since those pretrained weights are intended for an RGB input, it could well happen that features extracted by the depth branch end up playing little row in accounting for the features reaped by our fusion module. As a consequence, a teacher network initialized in this fashion could fail to perform significantly better than our baseline. The other option is to handle the initialization of the depth branch in a separate manner. That is, we can load the weights of an adapted baseline model onto the depth branch, whose foremost convolution layer is modified in exactly the same way. We refer to this variant as our baseline depth model from this point on. It shall accept a single depth image as input and can be pretrained on labeled depth samples collected from the *NTU RGB+D* dataset. An ablation study on the initialization scheme of our teacher model is discussed in Section 6.1.

4.2.2 Second Stage

The second stage of our training scheme is where knowledge transfer takes place. In this stage, we focus on the optimization of a standalone student model that is constructed from the same configuration as our baseline network and instructed to mimic the way our teacher model approaches the task of regressing a 3D human pose. Crucially, this student model sees only an RGB image under all circumstances and is to be deployed as an independent pose predictor at inference time. For each fully annotated pair of RGB and depth images, we feed both image modalities into our teacher network and pick out a certain mid-level feature map as a piece of train-time advice for the student. We freeze the parameters of our teacher model throughout this stage, so that a fixed output is always guaranteed for the same sample. The chosen mid-level response map contains critical information about the valuable depth cues that the teacher is able to incorporate from the privileged depth input, to which a direct access for the student is denied.

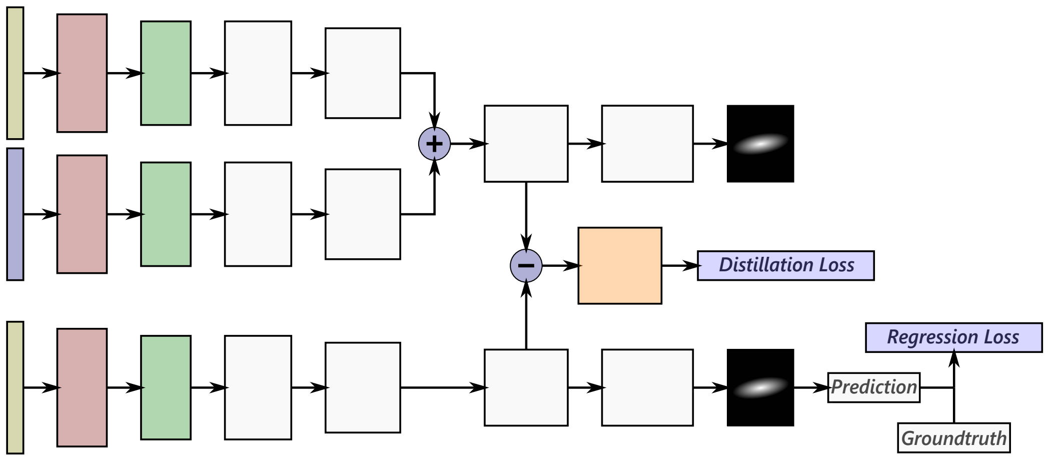


Figure 4.4: The second stage of our training scheme. We freeze the teacher network and jointly optimize the student model over two seprate supervision targets. In this depiction, the distillation loss is imposed on the last convolution layer in the third chain structure of both networks. The orange cuboid represents the difference between the respective response maps.

Given this crucial guidance, we jointly optimize the student model over two separate supervision targets. On one hand, we ask the student network to give its own prediction for the 3D pose of the subject based solely on the RGB input and evaluate how closely the estimation fits the groundtruth skeleton using Equation (4.12). In the meanwhile, we influence how the student produces its own features via the introduction of a distillation loss [HVD15] between the two models. We expect this distillation scheme to facilitate an effective knowledge transfer between the two models and enable the student to, for any test image, extract a rich feature representation somehow as if it were directly exposed to the paired depth modality.

A graphical depiction of the second stage of our learning framework is given in Figure 4.4. In particular, we establish an association between two mid-level convolution layers, one from either side, that correspond to the same target layer on the standard ResNet-50 architecture. In our experiments, the last convolution layer in the third chain structure of residual blocks turns out to be the best choice for this association. If we refer to the activation maps produced by the respective teacher layer and student layer in the matched pair as $\mathbf{F}_{teacher}$ and $\mathbf{F}_{student}$, we explicitly enforce the student model to generate a $\mathbf{F}_{student}$ that resembles $\mathbf{F}_{teacher}$ as closely as possible. Formally, our distillation loss can be formulated as

$$L_{distillation} = \|\sigma(\mathbf{F}_{teacher}) - \sigma(\mathbf{F}_{student})\|_2, \tag{4.22}$$

where $\sigma(\cdot)$ is the standard logistic sigmoid function as in Equation (2.4). Essentially, this loss function measures the pixelwise similarity between the mid-level response maps extracted by the respective networks. The sigmoid function is introduced to map the activation values to the range $[0, 1]$ and thus gain a better control over the magnitude of the loss. For a fully annotated RGB-D sample, the final loss formulation will be a weighted combination of the two loss terms described above:

$$L_{student} = L_{regression} + \alpha L_{distillation}, \tag{4.23}$$

where α is a hyperparameter that controls the relevance of the distillation term. In practice, the optimal value of α is dependent on a variety of factors and generally tricky to ascertain. Too large a value can cause the distillation term to dominate the total loss function, whereas a value too small makes the impact of our distillation scheme insignificant. In our experiments, we choose to linearly increase the magnitude of α to an empirical value, such that even contribution is made by both terms in our overall loss formulation at the end in general.

Additionally, it is worth noting that our distillation loss in Equation (4.22) does not require the presence of groundtruth 3D pose annotations. Since unlabelled RGB-D image pairs are available in abundance in the vision community, we take full advantage of this property and seek to acquire some weak supervision from the unlabelled portion of our data as well. Specifically, we propose to establish our train-time mini-batches as a hybrid of both labeled and unlabelled RGB-D image pairs. Whereas the loss formulation in Equation (4.23) is applied to the labeled portion of our samples, we compute only the distillation loss for samples devoid of annotation. We then aggregate the loss values computed over samples of either part to acquire a single optimization target.

4.2.3 Further Details

A number of implicit design choices have been made in the above description for our two-stage learning framework. In the following we will further clarify these decisions and discuss what could be expected of them as an outcome. We carry out extensive ablation studies to evaluate their practical impact and present the results in Subsection 6.1.

Distillation Target The distillation loss described by Equation (4.22) works on the response maps produced by the respective target layer instances from either model within the learning framework. It is unclear, however, which target layer in the standard architecture should be selected to optimally establish such an association. Moreover, in a modern convolutional architecture, the output features of a convolution layer typically pass through a batch normalization and a rectified linear activation function before they are processed by the subsequent convolution. Since batch normalization restores the distribution of a response map, we believe it steadies the learning process if we apply the distillation loss only after a batch normalization is performed. But it is still debatable whether the loss should be applied before the rectified linear function takes effect or after. These decisions have a decisive impact on the loss magnitude and general efficacy of the proposed method.

Distillation Loss Formulation Especially at the start of the second stage of our training scheme, the distillation term generally dominates the overall loss in terms of magnitude. To stabilize the gradients, one option is to pass the target response maps through a logistic sigmoid function before the pixelwise difference is taken, as in Equation (4.22). But if we were to apply a rectified linear function to the target features first, the sigmoid function would then map the activations to the range $[0.5, 1]$, which may not be ideal for the distillation loss to exercise its full potential. As such, we introduce an alternative loss formulation that is commonly applied in classification:

$$L_{distillation} = \sum_{c=1}^{C} \sum_{u=1}^{17} \sum_{v=1}^{17} bce\left(\mathbf{F}_{student}[c,u,v], \sigma(\mathbf{F}_{teacher}[c,u,v])\right), \tag{4.24}$$

where

$$bce(a,b) = -b\log(\sigma(a)) - (1-b)\log(1-\sigma(a)) \tag{4.25}$$

is the binary cross entropy loss function and $\sigma(\cdot)$ is again the sigmoid function.

Batch Normalization Statistics By default, the batch normalization layers in both models of our learning framework actively maintain a running estimation of mini-batch statistics. Nevertheless, it is equally interesting to see what will happen if both models freeze their accumulated mean and standard deviation vectors past a certain point in the second stage of our training scheme. This can be achieved by switching all batch normalization layers into evaluation mode in *Pytorch*. The update of learnable parameters for these layer will continue on regardless.

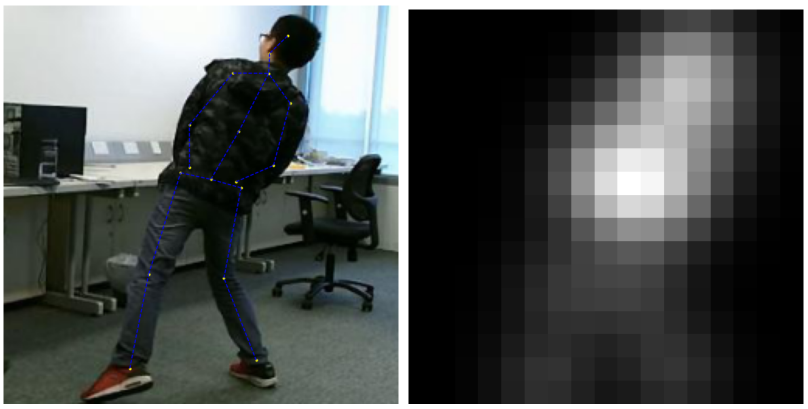

(a) sample (b) attention map

Figure 4.5: The generated artificial attention map for a given sample.

Attention Maps Attention modules [WGGH18] [JLLT18] have proven effective in their ability to guide convolutional networks towards key areas of an input image. We adopt a simple version of such strategies at train time and create an artificial attention map in an attempt to accelerate the knowledge transfer process. Specifically, for each labeled sample, we obtain the 2D image-space locations $(u_j, v_j)^\intercal$ for each skeletal keypoint $j \in J$ by projecting the 3D groundtruth skeleton onto the input image. The pixel value at location (u, v) on our attention map $\mathbf{A}$ can then be computed as

$$A[u,v] = \sum_{j=1}^{J} \exp\,(-\frac{(u-u_j)^2 + (v-v_j)^2}{5}). \tag{4.26}$$

As the next step, this attention map is normalized to the range $[0, 1]$ and resized to have the same spatial resolution as that of the target response maps on which our distillation loss is imposed. In the presence of this attention map, we replace the default formulation in Equation (4.22) for our distillation loss with

$$L_{distillation} = \|(\sigma(\mathbf{F}_{teacher}) - \sigma(\mathbf{F}_{student})) \odot \mathbf{A}\|_2, \tag{4.27}$$

where $\odot$ stands for pixelwise multiplication. Ideally, this addtional attention mechanism should encourage the student network to prioritize extracting the correct feature representations at image locations that correspond to the subject in the foreground. Figure 4.5 visualizes such an attention map for a given sample.

Partial Convolutions Partial convolutions proposed by [LRS$^+$18] excel at inpainting images that are fractionally corrupted. These convolutions re-weight the output activations based on how well the neighborhood of the current pixel is populated in the input feature map. Since the annotated depth images we are able to collect for our experiments are of limited quality, partial convolutions could benefit the teacher network in that they condition a deeper neuron's attention only on valid input pixels. To validate this idea, we take the baseline depth model defined in Subsection 4.2.1 and replace all standard convolutions up to the end of the second chain structure of residual blocks with partial convolutions. We train this adapted network on labeled depth images from the *NTU RGB+D* dataset and see if the modification brings any gain in performance.

5 Experimental Setup

Various experiments have been conducted to verify the efficacy of our proposed method. We dedicate this chapter to covering the preparation and implementation aspects of these experiments.

5.1 Datasets

We make use of two large-scale RGB-D datasets for training purposes. The *NTU RGB+D* dataset [LSP+19] and the *PKU-MMD* dataset [LHL+17] are the first datasets of their calibre to propel research on depth-based human action analysis. They both record a wide variety of relatively simple human actions, whereby the performers demonstrate considerable variations in terms of appearance. Each action sequence is simultaneously recorded by several *Micrsoft Kinect* sensor devices from different view angles. The RGB and depth videos captured by the respective sensors are in perfect sync, which allows us to easily collect an abundance of RGB-D image pairs. The statistics of both datasets can be found in Table 5.1. Sample RGB and depth images of both datasets can be found in Figure 5.1.

	camera angles	body rotations	daily + paired actions
NTU RGB+D	51	2	49 + 11
PKU-MMD	3	1	40 + 11

	subjects	RGB resolution	depth resolution
NTU RGB+D	40	1920×1080	512×424
PKU-MMD	66	1920×1080	512×424

Table 5.1: dataset statistics

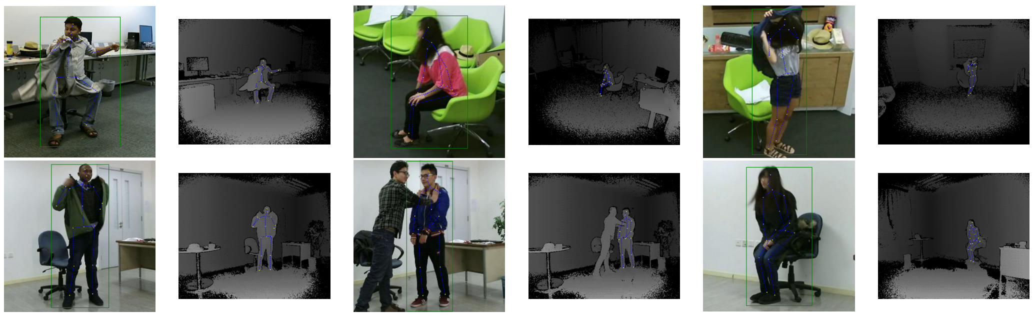


Figure 5.1: Sample poses collected from the *NTU RGB+D* dataset (first row) and the *PKU-MMD* dataset (second row).

5.1.1 Annotations

In practice, we consider the spatial misalignment between the color and depth cameras for both datasets to be negligible. In order to transform and crop the depth frames in the same way as the RGB ones for a given video sample, we need to know the intrinsic matrices for both cameras. Whereas the intrinsic parameters for RGB cameras are openly available, we rely on the default 2D and 3D annotations to compute those for the depth cameras. That is, under a fixed camera configuration, we organize the keypoint annotations in terms of 2D-3D coordinate pairs $(\mathbf{q}_i, \mathbf{p}_i)$, where $\mathbf{q}_i = (u_i, v_i)^\intercal$ and $\mathbf{p}_i = (x_i, y_i, z_i)^\intercal$. Since all these coodinate pairs are associated by the same perspective projection, the intrinsic parameters for the depth camera can be uniquely computed in least-squares fashion as

$$\begin{aligned} \begin{pmatrix} f_x \\ c_x \end{pmatrix} &= (\mathbf{A}_x^\intercal \mathbf{A}_x)^{-1} \cdot \mathbf{A}_x^\intercal \mathbf{u} \\ \begin{pmatrix} f_y \\ c_y \end{pmatrix} &= (\mathbf{A}_y^\intercal \mathbf{A}_y)^{-1} \cdot \mathbf{A}_y^\intercal \mathbf{v} \end{aligned}, \tag{5.1}$$

where

$$\begin{aligned} \mathbf{A}_x &= \begin{pmatrix} \frac{x_1}{z_1} & \frac{x_2}{z_2} & \dots & \frac{x_M}{z_M} \\ 1 & 1 & \dots & 1 \end{pmatrix}^\intercal \\ \mathbf{A}_y &= \begin{pmatrix} \frac{y_1}{z_1} & \frac{y_2}{z_2} & \dots & \frac{y_M}{z_M} \\ 1 & 1 & \dots & 1 \end{pmatrix}^\intercal \end{aligned}, \tag{5.2}$$

M is the total number of 2D-3D coordinate pairs, $\mathbf{u} = (u_1, u_2, \dots, u_M)^\intercal$ and $\mathbf{v} = (v_1, v_2, \dots, v_M)^\intercal$. The resultant parameters can then be inserted into entries of the intrinsic matrix as in Equation (1.3).

On the other hand, we notice the fact that the groundtruth skeletons in both datasets are directly taken from the output of the build-in recognition software, which means they are of below-par quality in terms of consistency and accuracy. For the actual train-time samples, we choose to discard the default annotations and use instead the 3D pose skeletons inferred by the model in [SLAL20] specialised for the ECCV workshop challenge. This brings the added benefit that the selection of keypoints our models estimate will be directly compatible with the skeletal configuration of [IPOS13]. Neither datasets provide bounding box information for the subjects. We run a Faster R-CNN detector [RHGS15] on each RGB video frame and select one detection box that fits the 2D keypoint annotation best for each sample. We then reproject the bounding box vertices onto the paired depth frame in a way similar to Equation (4.7), in order to ensure that the same image content is cropped for both frames.

5.1.2 Depth Images

Both datasets feature only indoor scenes under static lighting conditions. The individual depth frames captured by the *Microsoft Kinect* depth sensor store per-pixel the distances between the camera and the 3D points in space captured by the infrared scan. The maximum depth value stored in a depth image is, to the best of our knowledge, typically set to be around 3 metres by the points on the wall in the background. Corrupted pixels have a value of zero. To map these depth values to a range easier for our networks to process, we pass a depth map $\mathbf{D}$ through the exponential function

$$\mathbf{D}'[u,v] = \begin{cases} \exp\left(-\mathbf{D}[u,v]\right) & \mathbf{D}[u,v] < 0.1 \\ 0.0 & \mathbf{D}[u,v] \geq 0.1 \end{cases} \tag{5.3}$$

before it undergoes the same homographic transformation described in Subsection 4.1.1. Note that we treat a corrupted pixel as if it corresponds to an infinitely distant point in space. Furthermore, since we ask our convolutional networks to directly regress metric-space heatmaps, it could make more sense if the pixel values on a depth map were actual z-coordinates of the associated 3D points instead of their distances to the camera. Therefore we propose to optionally rectify the depth images via

$$\mathbf{D}'[u,v] = \mathbf{D}[u,v] \cdot \left(\left(\frac{u - c_x}{f_x} \right)^2 + \left(\frac{v - c_y}{f_y} \right)^2 + 1 \right)^{-1} \tag{5.4}$$

as an additional step before Equation (5.3) takes effect. The impact of this rectification step is discussed in Subsection 6.1.

5.1.3 Data Arrangement

The RGB-D image pairs collected from the *NTU RGB+D* dataset are assigned into two distinct partitions that do not share any subject or camera configuration.

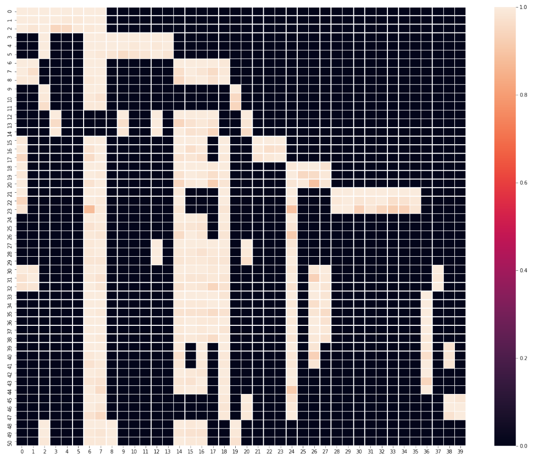


Figure 5.2: Frequency map for the *NTU RGB+D* dataset. Each entry in the graph is labeled with a saturation level proportional to the number of times a subject appears in one camera angle of a certain action sequence.

train-split subjects	P007 P008	P015 P016	P017 P018	P019 P020	P021 P022
	P023 P024	P025 P026	P027 P028	P037 P038	P039 P040
train-split configs	S003C001	S003C002	S003C003	S004C001	S004C002
	S004C003	S005C001	S005C002	S005C003	S006C001
	S006C002	S006C003	S007C001	S007C002	S007C003
	S009C001	S009C002	S009C003	S010C001	S010C002
	S010C003	S011C001	S011C002	S011C003	S012C001
	S012C002	S012C003	S013C001	S013C002	S013C003
	S014C001	S014C002	S014C003	S015C001	S015C002
	S015C003	S016C001	S016C002	S016C003	S017C001
	S017C002	S017C003			

test-split subjects	P001 P002	P003 P009	P010 P011	P029 P030	P031 P032
test-split configs	S001C001	S001C002	S001C003	S002C001	S002C002
	S002C003	S008C001	S008C002	S008C003	

Table 5.2: Subjects and camera configurations covered by our train-split and test-split of the *NTU RGB+D* dataset.

Figure 5.2 shows the number of times every subject appears in each camera angle of a certain action sequence. The camera configurations and subjects included by either partition are shown in Table 5.2. RGB-D image pairs from the first partition are used as fully annotated training samples through both stages of our training scheme, whereas the second partition is considered as a standalone test set. For the *PKU-MMD* dataset, the official cross-subject division proposal is adopted. We extract RGB-D image pairs from the train-split videos and use them as unlabelled samples in our weak-supervision setup. On the other hand, RGB-D samples collected from the test-split videos constitute another test set for model evaluation.

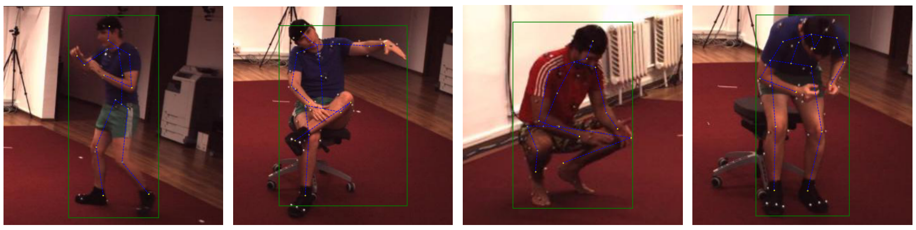


Figure 5.3: Sample poses collected from the *Human3.6M* dataset.

Additionally, we leverage the official test split of the *Human3.6M* dataset [IPOS13] to evaluate how well our models generalize to a different scene setup. This large-scale benchmark features several subjects performing a smaller range of more complex human activities. The groundtruth skeletons are gathered from a motion capture system based on attachable reflective markers, which allow for accurate reconstruction of human poses. Sample images of the *Human3.6M* dataset is shown in Figure 5.3.

5.2 Evaluation Metrics

We quantize the performance of our models by using three standard evaluation metrics that commonly appear in related literatures. Given estimated 3D root-relative poses

$$\hat{\mathbf{P}}_i = (\hat{\mathbf{p}}_{i,1}, \hat{\mathbf{p}}_{i,2}, \dots, \hat{\mathbf{p}}_{i,J}) \tag{5.5}$$

and groundtruth root-relative poses

$$\hat{\mathbf{P}}^*_i = (\hat{\mathbf{p}}^*_{i,1}, \hat{\mathbf{p}}^*_{i,2}, \dots, \hat{\mathbf{p}}^*_{i,J}), \tag{5.6}$$

where the footnote $i \in N$ indexes the test samples, the mean per-joint position error (MPJPE) computes, as its name suggests, the average euclidean distance between the predicted keypoint coordinates and the target keypoint coordinates over all test samples:

$$Q_{mpjpe} = \frac{1}{NJ} \sum_{i=1}^{N} \sum_{j=1}^{J} \left\| \hat{\mathbf{p}}_{i,j} - \hat{\mathbf{p}}^*_{i,j} \right\|_2 . \tag{5.7}$$

Naturally, lower Q_{mpjpe} on a test set indicates better estimation accuracy.

The percentage of correct keypoints (PCK) reflects the overall proportion of skeletal keypoints across all test samples that are reconstructed with an error below a given threshold T_{pck}:

$$Q_{pck} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{J} \mathbf{cond}\left(\left\| \hat{\mathbf{p}}_{i,j} - \hat{\mathbf{p}}^*_{i,j} \right\|_2 \leq T_{pck} \right)}{NJ}, \tag{5.8}$$

where **cond**$(\cdot)$ stands for the conditional function that evaluates to 1 if and only if the input condition were true. Higher Q_{pck} for a given threshold indicates better performance. In our experiments, we set T_{pck} at $50mm$ for our *NTU RGB+D* and *PKU-MMD* test sets and $150mm$ for the *Human3.6M* test set.

For a given model and fixed test set, by escalating T_{pck} from zero to an upper bound T_{auc}, we may uniquely determine a curve that describes the percentage of correct keypoints as a function of the employed threshold. The area under the curve (AUC) translates to the definite integral of this function from zero to T_{auc}:

$$Q_{auc}(T_{auc}) = \int_{0}^{T_{auc}} Q_{pck}(T_{pck})\, dT_{pck}. \tag{5.9}$$

Higher Q_{auc} for a given threshold indicates better performance. In our experiments, we set T_{auc} at $50mm$ for our *NTU RGB+D* and *PKU-MMD* test sets and $150mm$ for the *Human3.6M* test set.

The three evaluation metrics described above quantify different aspects of a model's capacity. For example, a model can well attain an inferior PCK-score when it is able to, at the same time, reconstruct poses with relatively low mean per-joint position error, since the PCK-score does not take into account very difficult samples, for which the reconstruction error is well above the applied threshold.

5.3 Implementation Details

We implement our 3D human pose estimation framework utilizing the *PyTorch* [PGM+19] deep learning library. Forward and backward propagation of convolutional networks are carried out in mixed-precision [MNA+17] both to accommodate more samples per mini-batch and to facilitate faster tensor computations. Half-precision loss values are upscaled by a factor of 32.0 before the gradients are propagated backwards. The resultant gradients are converted to full-precision and downscaled by the same factor before the network parameters are updated. The obnoxious issue of exploding gradients often accompanies mixed-precision training procedures thanks to the narrowly limited numerical range. To prevent the occurrence of gradient explosion, an additional gradient clipping step is introduced. Gradient vectors of large magnitude are clipped to have a norm of 5.0 before they are processed by the optimizer.

In all of our experiments, random he-initialization is applied to the fusion module and the regression head, since they don't match a particular layer in the standard ResNet architecture. We use standard-weights from this point on to refer to the parameters of a standard model pretrained on ImageNet. Under situations where we want to initialize a depth-based network or network branch with standard-weights, a problem arises due to the mismatch in kernel shape

dataset	*NTU RGB+D*			*PKU-MMD*			*Human3.6M*		
metric	MPJPE	PCK	AUC	MPJPE	PCK	AUC	MPJPE	PCK	AUC
stddev	0.24	0.001	0.001	0.484	0.003	0.003	1.24	0.004	0.003

Table 5.3: Model variability in terms of standard deviation under the same training configuration.

between the respective foremost convolution layers. In such cases, we opt to take the average over the three channels of the standard RGB kernel and use that to initialize its single-channel counterpart. Similarly, when we use the weights of a baseline depth model to initialize an RGB-based network, we duplicate three copies of the single-channel convolution weights, stack them along the channel-axis and then divide the resultant kernel by 3.

In all of our experiments, we train a model for 30 epochs on the train-split of the *NTU RGB+D* dataset. Before each epoch commences, the training samples are randomly reshuffled. Under situations where unlabelled RGB-D image pairs are utilized to provide weak supervision, this portion of the training data is loaded from a separate *PyTorch* Dataloader. A mini-batch always contains the largest possible number of samples that can be processed in parallel on the GPU, rounded to a multiple of 16. Unlabelled samples, if used, constitute a third of all samples in a mini-batch.

We employ the standard Adam optimizer to manage the parameter updates. Coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are applied for the accumulation of the gradients and their second moments respectively. In particular, the *PyTorch* implementation of this optimizer incorporates an additional weight decay term into the update formula, which regularizes the magnitude of weight vectors for each layer. In our experiments, a weight decay factor of 4e$-$5 is adopted for all network layers.

We employ a step-decay learning rate scheduler for all experiments. The base learning rate η_0 is set at 5e$-$5 unless stated otherwise. The actual learning rate η can be perceived as a piecewise function of the number of past epochs n_{past}:

$$\eta(n_{past}) = \begin{cases} 0.5 \cdot \eta_0 & n_{past} < 1 \\ \eta_0 & n_{past} < 15 \\ 0.2 \cdot \eta_0 & n_{past} < 20 \\ 0.04 \cdot \eta_0 & n_{past} < 25 \\ 0.008 \cdot \eta_0 & 25 \leq n_{past} \end{cases}, \tag{5.10}$$

where the first period serves as warmup phase. At the end of our experiments all models have converged to an acceptable degree.

Due to various sources of randomness, models trained under exactly the same configuration can differ in all aspects. To get an idea of how much the performances of duplicate models can vary, we conduct five identical experiments, in which we train a baseline model using only labeled RGB samples from the *NTU*

RGB+D dataset. When evaluated on each of the three test sets according to the aforementioned metrics, our duplicate models obtain similar but distinguishable performance scores. The standard deviations among these scores are listed in Table 5.3.

6 Experimental Results

In this chapter, we demonstrate the outcomes of the various studies we carry out, in an attempt to validate whether our method is of practical relevance. In the first part, we experiment with different design choices within our learning framework and reason about their impact on model performance. In a second part, we present the core experiments as well as a comprehensive analysis of the major findings.

6.1 Study on Design Choices

We test out in this section each of the training options discussed in Section 4.2. Note that experiments in this section consider only labeled training data from the *NTU RGB+D* train-split.

6.1.1 Baseline Performance

We train four baseline RGB models and report in Table 6.1 their performance on the *NTU RGB+D* test-split and the *Human3.6M* test-split. All train-time conditions, except for the few options shown as table columns, are set to be identical. Train-time color and geometry augmentations described in Subsection 4.1.4 are applied to models 1.2-4. A comparison between model 1.1 and 1.2 shows that the adoption of augmented samples allows the model to generalize significantly better to different scenes. Model 1.3 is initialized with the weights taken from a baseline depth model in Table 6.2. Model 1.4 gets its initialization from the last checkpoint of model 1.2. A comparison among models 1.2-4 shows that initialization bears a significant impact on model capacity. In particular, we observe that model 1.4 demonstrates an overall better performance than the other two models. This indicates that it is better to start with a model that accepts the same input modality and has already been finetuned on our training data.

ID	input	augmentation	init	*NTU RGB+D*	*Human3.6M*
1.1	RGB	N	standard	26.68	82.27
1.2	RGB	C + G	standard	26.31	77.98
1.3	RGB	C + G	model 2.1	25.42	78.47
1.4	RGB	C + G	model 1.2	25.45	76.55

Table 6.1: Evaluation results for baseline RGB models on two independent test sets. We report the mean per-joint position error in millimeters.

ID	input	init	rectify	partial	MPJPE	PCK [50mm]	AUC [50mm]
2.1	depth	standard	N	N	30.38	0.85	0.52
2.2	depth	standard	Y	N	30.39	0.85	0.52
2.3	depth	standard	Y	Y	30.38	0.85	0.52

Table 6.2: Evaluation results for baseline depth models on the *NTU RGB+D* test-split. We report performance in terms of all three metrics.

6.1.2 Depth Map Rectification and Partial Convolutions

We train three baseline depth models and compare their performance on the *NTU RGB+D* test-split in Table 6.2. All train-time conditions, except for the few options shown as table columns, are set to be identical. In particular, no train-time augmentation is applied. Model 2.2 and 2.3 are trained on depth images that undergo the pythagorean rectification according to Equation (5.4). A fraction of the standard convolutions in Model 2.3 are replaced with partial convolutions as described in Subsection 4.2.3. Judging by the reported numbers neither of the two proposed modifications lead to any improvement. As such, we do not apply them in any other experiment.

6.1.3 Teacher Model Initialization

We train two teacher models as described in Subsection 4.2.1 and compare their performance on the *NTU RGB+D* test-split in Table 6.3. All train-time conditions, except for the few options shown as table columns, are set to be identical. Both experiments employ train-time color and geometry augmentations. The depth branch of model 3.2 gets its initialization from the last checkpoint of model 2.1. We initialize the RGB branch and the trunk part of either model with standard-weights. It could be concluded from the numbers in the table that the initialization strategy does not exert that much of an influence on model performance as we expected. Nevertheless, we do notice that model 3.2 steadily induce a smaller regression loss during the first few training epochs, although this difference gradually wears thin as training proceeds. We visualize this finding in Figure 6.1.

ID	input	depth-branch init	MPJPE	PCK [50mm]	AUC [50mm]
3.1	RGB + depth	standard	19.77	0.94	0.66
3.2	RGB + depth	model 2.1	19.71	0.94	0.66

Table 6.3: Evaluation results for teacher models on the *NTU RGB+D* test-split. We report performance in terms of all three metrics.

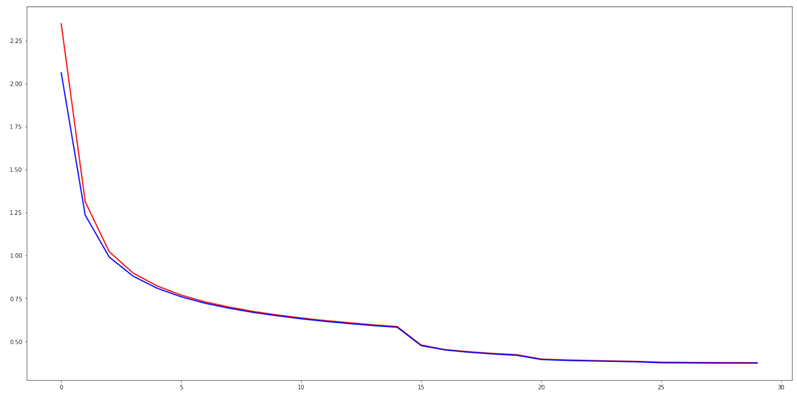


Figure 6.1: A visual comparison between the regression loss curve induced by either model in Table 6.3. The red and blue curve correspond to model 3.1 and 3.2 respectively. Note that the loss induced by model 3.2 is visibly smaller through the first few epochs.

6.1.4 Distillation Target and Sigmoid Mapping

We train four student models as described in Subsection 4.2.2 and report in Table 6.4 their performance on the *NTU RGB+D* test-split and the *Human3.6M* test-split. All train-time conditions, except for the few options shown as table columns, are set to be identical. In particular, the model 3.2 is set to be the teacher model in the privileged information learning setup. The default pixelwise similarity loss formulation in Equation (4.22) is employed for the distillation procedure. Batch normalization layers in both models are configured to actively collect mini-batch statistics. The scaling factor α of the distillation term linearly increases from a warmup value to a target value over the course of the first 10 epochs. From that point on it stays untouched until the end of the training procedure. The warmup and target values of α are set empirically as explained in Subsection 4.2.2.

For model 4.1, we impose the feature map similarity constraint at the end of the last chain structure of residual blocks. For models 4.2-4, the distillation target is set to be the ending convolution in the last residual block of the third chain structure. A direct comparison between model 4.1 and 4.2 shows that the second configuration is indeed of practical advantage. We suspect this is because an earlier distillation target affords the model more freedom to decide on what is the best way to leverage the hallucinated depth cues. Hence we stick to the second configuration for the rest of this thesis. Moreover, a comparison between model

ID	teacher	$\alpha \Rightarrow$		target	position	sigmoid
4.1	model 3.2	0.005	0.01	16-th block	after relu	N
4.2	model 3.2	0.01	0.04	13-th block	after relu	N
4.3	model 3.2	0.1	0.2	13-th block	after relu	Y
4.4	model 3.2	0.05	0.1	13-th block	before relu	Y

ID	input	init	augmentation	*NTU RGB+D*	*Human3.6M*
1.2	RGB	standard	C + G	26.31	77.98
4.1	RGB	standard	C + G	26.38	75.34
4.2	RGB	standard	C + G	25.44	74.84
4.3	RGB	standard	C + G	25.74	75.51
4.4	RGB	standard	C + G	25.27	74.93

Table 6.4: The upper chart shows the distillation configurations for several student models. The lower chart shows their evaluation results. We report the mean per-joint position error in millimeters.

1.2 and models 4.2-4 justifies the feasibility of our proposed training scheme. The student models outperform the baseline model by a considerable margin on both test sets.

The default sigmoid mapping in Equation (4.22) is excluded for model 4.2. The distillation loss is, for model 4.4, imposed on the features before they pass through the subsequent rectified linear function. A comparison among models 4.2-4 comfirms our previous speculations. On one hand, the sigmoid mapping is better skipped if we pass the target features through a rectified linear function first. On the other hand, if we were to keep the sigmoid mapping in the equation, then it is better to compute the distillation term before the subsequent $relu(\cdot)$ takes effect. In practice, the raw magnitude of the distillation term in experiment 4.2 varies significantly from that in experiment 4.4. A careful choice on each hyperparameter preconditions the efficacy of our training scheme.

6.1.5 Distillation Loss Formulation

We further train two student models with the alternative formulation for our distillation loss as in Equation (4.24). We report in Table 6.5 their performance on the *NTU RGB+D* test-split and the *Human3.6M* test-split. All train-time conditions, except for the few options shown as table columns, are set to be identical. In particular, a direct comparison between model 5.1 and 4.2 shows that the default pixelwise similarity formulation facilitates better estimation accuracy. The comparison between model 5.2 and 4.4 yields similar conclusions.

We suspect what plays a big part in this is the shape of a formulation's core function. The huber term in Equation (4.22) scales linearly as the pixelwise difference escalates, whereas the binary cross entropy term in Equation (4.24) scales in hyperbolic fashion. This makes a difference in the presence of noisy training samples that make it difficult for the student network to extract proper

ID	teacher	$\alpha \Rightarrow$		loss term	sigmoid	position
4.2	model 3.2	0.01	0.04	huber	N	after relu
5.1	model 3.2	0.01	0.1	bce	-	after relu
4.4	model 3.2	0.05	0.1	huber	Y	before relu
5.2	model 3.2	0.005	0.02	bce	-	before relu

ID	input	init	augmentation	*NTU RGB+D*	*Human3.6M*
1.2	RGB	standard	C + G	26.31	77.98
4.2	RGB	standard	C + G	25.44	74.84
5.1	RGB	standard	C + G	26.57	77.14
4.4	RGB	standard	C + G	25.27	74.93
5.2	RGB	standard	C + G	26.05	77.40

Table 6.5: The upper chart shows the distillation configurations for several student models. The lower chart shows their evaluation results. We report the mean per-joint position error in millimeters.

ID	teacher	$\alpha \Rightarrow$		const stats	*NTU RGB+D*	*Human3.6M*
4.3	model 3.2	0.1	0.2	N	25.74	75.51
6.1	model 3.2	0.1	0.2	Y	25.96	75.50

Table 6.6: Evaluation results for student models on two independent test sets. We report the mean per-joint position error in millimeters.

depth cues. In such case, the distillation loss based on the binary cross entropy term would contribute a dramatic part to the overall error, which forces the model to pay less attention to the pose regression task.

6.1.6 Batch Normalization Statistics

Next, as was discussed in Subsection 4.2.3, we test out the train-time configuration where, after the 10-th epoch, we turn off the update of mini-batch statistics for batch normalization layers in both networks. We compare the resultant model against model 4.3 in Table 6.6. All train-time conditions, except for the few options shown as table columns, are set to be identical. Judging by the reported numbers the proposed modification does not lead to any visible improvement. As such, we do not switch models into evaluation mode in any other experiment.

6.1.7 Artificial Attention Maps

Finally, we test out the possibility of applying an artificial attention map to the distillation process, as described by Equation (4.27). We compare the resultant model against model 4.3 in Table 6.7. All train-time conditions, except for the few options shown as table columns, are set to be identical. As indicated by the reported numbers, model 7.1 outperforms model 4.3 on both datasets. This could imply that our artificial attention mechanism is able to help the model focus on

ID	teacher	$\alpha \Rightarrow$		attention map	*NTU RGB+D*	*Human3.6M*
4.3	model 3.2	0.1	0.2	N	25.74	75.51
7.1	model 3.2	0.25	0.5	Y	25.46	75.15

Table 6.7: Evaluation results for student models on two independent test sets. We report the mean per-joint position error in millimeters.

ID	teacher	init	$\alpha \Rightarrow$		attention map	position	weak supervision
8.1	model 3.2	model 1.2	0.1	0.2	N	after relu	N
8.2	model 3.2	model 1.2	0.1	0.2	N	after relu	Y
8.3	model 3.2	model 1.2	0.25	0.5	Y	before relu	N
8.4	model 3.2	model 1.2	0.25	0.5	Y	before relu	Y

	NTU RGB+D			*PKU-MMD*			*Human3.6M*		
ID	MPJPE	PCK	AUC	MPJPE	PCK	AUC	MPJPE	PCK	AUC
1.4	25.45	0.91	0.63	18.55	0.95	0.69	76.55	0.89	0.56
8.1	24.69	0.91	0.64	17.82	0.95	0.70	73.82	0.90	0.57
8.2	26.39	0.91	0.63	18.68	0.95	0.69	76.07	0.90	0.56
8.3	24.58	0.91	0.64	18.01	0.95	0.69	73.95	0.90	0.57
8.4	25.70	0.91	0.63	17.98	0.95	0.69	74.60	0.90	0.56

Table 6.8: The upper chart shows the distillation configurations for several student models. The lower chart shows their evaluation results across the three independent test sets. We report performance in terms of all three metrics.

feature replication in image areas where the person resides. Furthermore, we notice in practice that, during the first few epochs, the pixelwise multiplication by our attention map cuts down the magnitude of the distillation loss by around 70 percent.

6.2 Core Study

We examine in this section to what extent privileged information is able to lift the performance of a single RGB model. Here we adopt some of the design choices that prove to be beneficial in our experiments from the previous section.

6.2.1 Student Performance

We train four student models as described in Subsection 4.2.2 and report in Table 6.8 their performance across all three test sets. All train-time conditions, except for the few options shown as table columns, are set to be identical. All experiments employ train-time color and geometry augmentations. For our distillation loss the default pixelwise similarity formulation in Equation (4.22) is employed.

Performance scores achieved by model 8.1 and 8.3 indicate that our privileged information learning setup is able to improve pose estimation accuracy across all three datasets by a considerable margin. In particular, model 8.1 outperforms

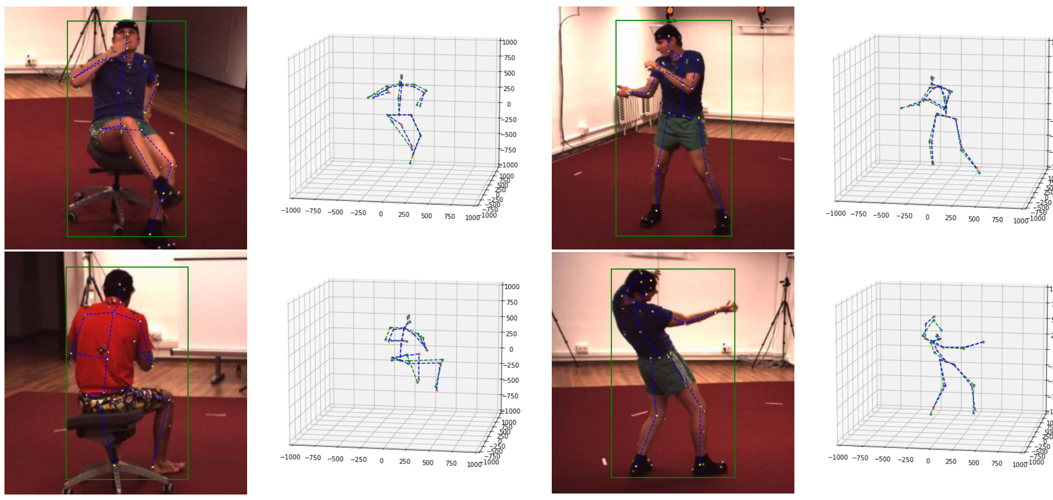


Figure 6.2: 3D pose predictions made by model 8.1 on *Human3.6M* test samples. In each 3D plot the green skeleton is the groundtruth, the blue skeleton is the prediction.

model 1.4 in terms of mean per-joint position error on the *Human3.6M* test-split by $2.7mm$. This could imply that, thanks to the knowledge transfer process, the RGB-based student is able to extract a feature representation that is more robust to variations in scene environment. We visualize in Figure 6.2 the predictions made by model 8.1 on *Human3.6M* test samples.

Model 8.2 and 8.4 receive via the distillation loss additional weak supervision from unlabelled RGB-D image pairs that we collect from the *PKU-MMD* dataset. Model 8.1 and 8.2 form a direct comparison. Judging by the reported numbers, the additional unlabelled training data rather worsened a student's test-time performance. In particular, model 8.1 surpasses model 8.2 in terms of estimation accuracy even on the *PKU-MMD* test-split, despite the fact that it had no train-time access to samples from that dataset. A comparison between model 8.3 and 8.4 yields similar observations. More experimentation is needed in this regard to ascertain the best way to exploit these unlabelled image pairs.

6.2.2 Effect of Distillation and Knowledge Transfer

In addition to performance evaluation, it is also interesting to see how well a student model is able to, given only a single RGB image, replicate the features produced by the teacher model, who has access to the paired depth image as well. For this purpose, we gather a small set of RGB-D image pairs from the *PKU-MMD* test-split. For each image pair, we take the absolute difference between the respective target features extracted by model 3.2 and 8.1, then add up the result along the channel-axis to obtain a single-channel difference map. We visualize in Figure 6.3 this difference map along with the one obtained in the same way

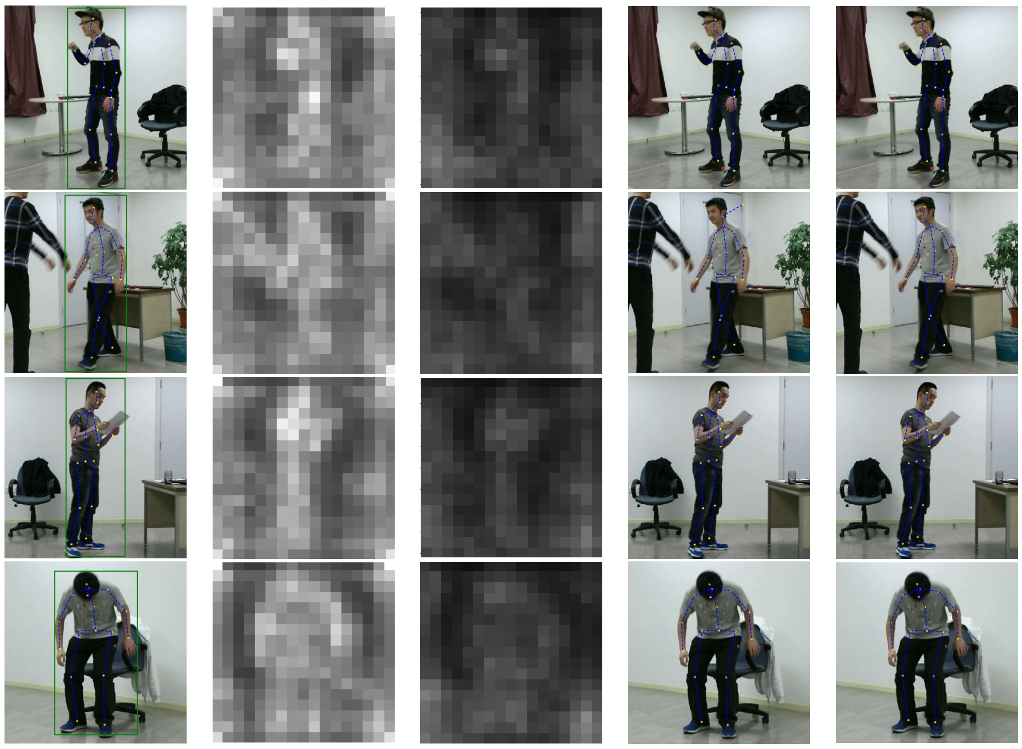


Figure 6.3: Difference map comparison for samples collected from the *PKU-MMD* test-split. Each row represents a separate test sample. The first column shows the input RGB image. The second column shows the difference map between model 3.2 and 1.4. The third column shows the difference map between model 3.2 and 8.1. Darker pixels indicate smaller absolute feature difference for that image location. The last two columns show the respective pose predictions made by model 1.4 and 8.1, projected onto the image plane.

between model 3.2 and model 1.4. Judging by pixelwise intensity, it is clear that the target features produced by student model 8.1 more closely resemble those output by teacher model 3.2. In fact, the minimum pixel value on the difference map between model 3.2 and 1.4 is often 4 to 5 times that on the difference map between model 3.2 and 8.1.

Another aspect of the learning framework we would like to study is whether or not the student model has actually learned to effectively extract depth cues from an RGB image. Whereas there is no direct way to prove or falsify this, we come up with a workaround plan. That is, we discard the depth image in each training sample and configure an RGB-based student model to receive privileged guidance from the baseline RGB model 1.4. The other configurations of this experiment closely follow the desciption in Subsection 4.2.2. We compare this model against model 8.1 in Table 6.9. All train-time conditions, except for the few options

ID	teacher	$\alpha \Rightarrow$		loss term	sigmoid	position
8.1	model 3.2	0.1	0.2	huber	Y	after relu
9.1	model 1.4	0.1	0.2	huber	Y	after relu

ID	input	init	augmentation	*NTU RGB+D*	*Human3.6M*
8.1	RGB	model 1.2	C + G	24.69	73.82
9.1	RGB	model 1.2	C + G	26.41	74.88

Table 6.9: The upper chart shows the distillation configurations for both student models. The lower chart shows their evaluation results on two independent test sets. We report the mean per-joint position error in millimeters.

shown as table columns, are set to be identical. As indicated by the reported numbers, model 8.1 outperforms model 9.1 on both datasets. Since model 1.4, teacher of model 9.1, had no access to any depth information throughout, we speculate it is the privileged depth information provided to model 8.1 that allows it to achieve this superior test-time performance.

7 Conclusion

In this work, we aimed to improve monocular 3D human pose estimation by taking advantage of the depth information available at training time. We adopted the pose estimation framework proposed by Istvan et al. [SLAL20] as a baseline method. We built on their approach to develop a privileged information learning framework that teaches the equivalent of a baseline model to effectively extract depth cues from a single RGB image.

We proposed to achieve this in two steps. In the first step, we train a teacher network to regress human poses as accurately as possible. The teacher network is adapted from the baseline model to accept one RGB image and one paired depth image as input at the entrance of either branch. The two branches are joined together via a fusion module in the middle of the original backbone. In a second step, we teach an RGB-based student network to, in the presence of labeled RGB-D image pairs, replicate the features produced by the teacher network for each input sample. This feature representation counts as privileged information and transfers knowledge to the student network about which visual cues are most crucial for the pose regression task.

We carried out a range of experiments to validate the efficacy of the proposed method. In particular, a student model trained within our framework outperforms the baseline model by a considerable margin across all three test sets, which clearly indicates that our method is of practical relevance. Furthermore, we tested out various design choices to facilitate a more efficient knowledge transfer process. Specifically, we tried an alternative distillation loss formulation, experimented with different candidate layers as the distillation target and devised an artificial attention mechanism. After a careful analysis of the experimental results, we conclude that it is generally favorable to

- initialize a student network with weights taken from a pretrained baseline RGB model.
- employ the proposed artificial attention mechanism.

- stick to the default pixelwise difference formulation for our distillation loss.
- set the convolution at the end of the third chain structure of residual blocks as the distillation target.
- impose the pixelwise similarity constraint on features before they undergo the subsequent rectified linear function.
- set the α hyperparameter such that equal contribution is made by the two loss terms in general.

7.1 Future Work

The effort to exploit unlabelled RGB-D image pairs that are abundantly available did not pay off in our experiments. The reason for this could be complex but one thing we noticed is that the teacher model we use does not perform sufficiently well on the *PKU-MMD* test-split either. This may imply that the sample distribution on the *PKU-MMD* dataset could be very different from that on the *NTU RGB+D* dataset. If that is the case, it might make sense for us to try out this weak supervision strategy on some other dataset.

We also notice that, due to the lack of pose diversity and scene variability, the evaluation error for our baseline model on the *NTU RGB+D* test-split and the *PKU-MMD* test-split are already non-realistically low. The vast majority of the training samples are so easy to overfit that they make it impossible for us to train a student model that is of practical use for in-the-wild application. If someday a new large-scale RGB-D dataset with greater pose and scene variety arises, it would certainly be interesting to repeat our experiments there and see if they open up more possibilities.

Bibliography

[BMB20] Brais Bosquet, Manuel Mucientes, and Victor M Brea. Stdnet: Exploiting high resolution feature maps for small object detection. *Engineering Applications of Artificial Intelligence*, 91:103615, 2020.

[CPK$^+$14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[CPK$^+$17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[CSWS17] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DGM$^+$19] Rishabh Dabral, Nitesh B Gundavarapu, Rahul Mitra, Abhishek Sharma, Ganesh Ramakrishnan, and Arjun Jain. Multi-person 3d human pose estimation from monocular images. In *2019 International Conference on 3D Vision (3DV)*, pages 405–414. IEEE, 2019.

[FPW17] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Spatiotemporal multiplier networks for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4768–4777, 2017.

[FQY$^+$18] Mingsheng Fu, Hong Qu, Zhang Yi, Li Lu, and Yongsheng Liu. A novel deep learning-based collaborative filtering model for recommendation system. *IEEE transactions on cybernetics*, 49(3):1084–1096, 2018.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[GHM16] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2827–2836, 2016.

[GMM18] Nuno C Garcia, Pietro Morerio, and Vittorio Murino. Modality distillation with multiple stream networks for action recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 103–118, 2018.

[GMM19] Nuno C Garcia, Pietro Morerio, and Vittorio Murino. Learning with privileged information via adversarial discriminative modality distillation. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2581–2593, 2019.

[GPB$^+$20] Joseph Gesnouin, Steve Pechberti, Guillaume Bresson, Bogdan Stanciulescu, and Fabien Moutarde. Predicting intentions of pedestrians from 2d skeletal pose sequences with a representation-focused multi-branch deep learning network. *Algorithms*, 13(12):331, 2020.

[HGD16] Judy Hoffman, Saurabh Gupta, and Trevor Darrell. Learning with side information through modality hallucination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 826–834, 2016.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IPOS13] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[JLLT18] Saumya Jetley, Nicholas A Lord, Namhoon Lee, and Philip HS Torr. Learn to pay attention. *arXiv preprint arXiv:1804.02391*, 2018.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[LCY18] Chenxu Luo, Xiao Chu, and Alan Yuille. Orinet: A fully convolutional network for 3d human pose estimation. *arXiv preprint arXiv:1811.04989*, 2018.

[LHL$^+$17] Chunhui Liu, Yueyu Hu, Yanghao Li, Sijie Song, and Jiaying Liu. Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding. *arXiv preprint arXiv:1703.07475*, 2017.

[LPBSV15] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015.

[LPV$^+$20] Timm Linder, Kilian Y Pfeiffer, Narunas Vaskevicius, Robert Schirmer, and Kai O Arras. Accurate detection and 3d localization of humans using a novel yolo-based rgb-d fusion approach and synthetic training data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1000–1006. IEEE, 2020.

[LRS$^+$18] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 85–100, 2018.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[LSP+19] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C Kot. Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2684–2701, 2019.

[MGVCO18] Angel Martínez-González, Michael Villamizar, Olivier Canévet, and Jean-Marc Odobez. Real-time convolutional networks for depth-based human pose estimation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 41–47. IEEE, 2018.

[MHRL17] Julieta Martinez, Rayat Hossain, Javier Romero, and James J Little. A simple yet effective baseline for 3d human pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2640–2649, 2017.

[MNA+17] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[MSM+19] Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Mohamed Elgharib, Pascal Fua, Hans-Peter Seidel, Helge Rhodin, Gerard Pons-Moll, and Christian Theobalt. Xnect: Real-time multi-person 3d human pose estimation with a single rgb camera. *arXiv preprint arXiv:1907.00837*, 2019.

[MSM+20] Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Mohamed Elgharib, Pascal Fua, Hans-Peter Seidel, Helge Rhodin, Gerard Pons-Moll, and Christian Theobalt. Xnect: Real-time multi-person 3d motion capture with a single rgb camera. *ACM Transactions on Graphics (TOG)*, 39(4):82–1, 2020.

[NHMP19] Aiden Nibali, Zhen He, Stuart Morgan, and Luke Prendergast. 3d human pose estimation with 2d marginal heatmaps. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1477–1485. IEEE, 2019.

[OVBG19] Tanguy Ophoff, Kristof Van Beeck, and Toon Goedemé. Exploring rgb+ depth fusion for real-time object detection. *Sensors*, 19(4):866, 2019.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[PZDD17] Georgios Pavlakos, Xiaowei Zhou, Konstantinos G Derpanis, and Kostas Daniilidis. Coarse-to-fine volumetric prediction for single-image 3d human pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7025–7034, 2017.

[PZK+17] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin Murphy. Towards accurate multi-person pose estimation in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4903–4911, 2017.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[RKE20] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. Sign language recognition: A deep survey. *Expert Systems with Applications*, page 113794, 2020.

[SFEG19] Thomas Schnürer, Stefan Fuchs, Markus Eisenbach, and Horst-Michael Groß. Real-time 3d pose estimation from single depth images. In *VISIGRAPP (5: VISAPP)*, pages 716–724, 2019.

[SGG16] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

[SLAL20] István Sárándi, Timm Linder, Kai Oliver Arras, and Bastian Leibe. Metrabs: Metric-scale truncation-robust heatmaps for absolute 3d human pose estimation. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 3(1):16–30, 2020.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[SXW$^+$18] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 529–545, 2018.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[VI15] Vladimir Vapnik and Rauf Izmailov. Learning using privileged information: similarity control and knowledge transfer. *J. Mach. Learn. Res.*, 16(1):2023–2049, 2015.

[VL20] Márton Véges and András Lőrincz. Multi-person absolute 3d human pose estimation with weak depth supervision. In *International Conference on Artificial Neural Networks*, pages 258–270. Springer, 2020.

[WGGH18] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[WRKS16] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[XLZ$^+$17] Yang Xing, Chen Lv, Zhaozhong Zhang, Huaji Wang, Xiaoxiang Na, Dongpu Cao, Efstathios Velenis, and Fei-Yue Wang. Identification and analysis of driver postures for in-vehicle driving activities and secondary tasks recognition. *IEEE Transactions on Computational Social Systems*, 5(1):95–108, 2017.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[XWW18] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.

[ZHS$^+$17] Xingyi Zhou, Qixing Huang, Xiao Sun, Xiangyang Xue, and Yichen Wei. Towards 3d human pose estimation in the wild: a weakly-supervised approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 398–407, 2017.

[ZWD+18] Christian Zimmermann, Tim Welschehold, Christian Dornhege, Wolfram Burgard, and Thomas Brox. 3d human pose estimation in rgbd images for robotic task learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1986–1992. IEEE, 2018.