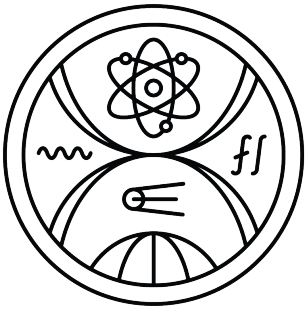
Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

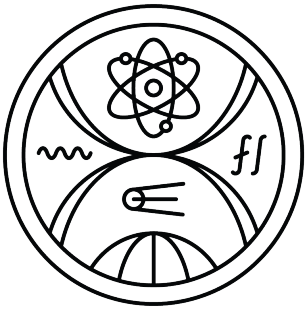


Structured Point Clouds for Machine Learning Processing of Human Body Data

Master's Thesis

2024
Bc. Erik Róbert Ján Jakubovský

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics



Structured Point Clouds for Machine Learning Processing of Human Body Data

Master's Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	Mgr. Dana Škorvánková

Bratislava, 2024
Bc. Erik Róbert Ján Jakubovský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Erik Róbert Ján Jakubovský
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Structured Point Clouds for Machine Learning Processing of Human Body Data
Štruktúrované mračná bodov zobrazujúce ľudské telo pre metódy strojového učenia

Anotácia: 3D dátové štruktúry predstavujú efektívny dátový typ ako vstupné dáta pre metódy strojového učenia. Štandardné neorganizované mračná bodov poskytujú reálne 3D súradnice dátových bodov, avšak chýba im priestorová topológia a informácie o lokálnych susedstvách bodov. Operácie ako konvolúcia nemôžu byť na tento typ dát priamo použité, preto sa vyžadujú odlišné spôsoby spracovania v neurónových sieťach. Zavedenie mriežkovej štruktúry do mračien bodov otvára nové možnosti v oblasti spracovania a prináša nové výzvy vo viacerých oblastiach, vrátane úloh zameraných na analýzu ľudského tela a pohybu.

Cieľ: Cieľom tejto práce je preskúmať výhody resp. nevýhody mračien bodov štruktúrovaných do mriežky ako vstupných dát pre úlohy zamerané na analýzu ľudského tela. Použijú sa metódy strojového učenia, keďže analytické prístupy boli v tejto oblasti prekonané.

Kľúčové slová: štruktúrované mračná bodov, neurónové siete, ľudské telo

Vedúci: Mgr. Dana Škorvánková
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 19.09.2022

Dátum schválenia: 20.09.2022

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Erik Róbert Ján Jakubovský
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Structured Point Clouds for Machine Learning Processing of Human Body Data

Annotation: 3D data structures present an effective input data type for machine learning methods. Standard unorganized point clouds provide real 3D coordinates of data points, however they lack the spatial topology and information on local neighborhoods. Operations like convolution cannot be directly used, therefore it requires different ways of processing in neural networks. Introducing the grid structure to the point clouds opens new possibilities for processing, and brings new challenges to various areas, such as human body and motion-related tasks.

Aim: The aim of the thesis is to explore the benefits and, possibly, the drawbacks of the grid-structured point clouds as an input data type for human body-related tasks. Use machine learning methods, as analytical approaches have been outperformed in these tasks.

Keywords: structured point clouds, neural networks, human body

Supervisor:	Mgr. Dana Škorvánková
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	19.09.2022
Approved:	20.09.2022

prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I am grateful to my supervisor, Mgr. Dana Škorvánková, for her invaluable guidance and assistance throughout the completion of this thesis. Additionally, I extend my heartfelt thanks to my friends and family for their unwavering support.

Abstrakt

Porozumenie ľudskému správaniu si vyžaduje presný odhad pózy človeka jednak v čase ale aj priestore. Odhad póz hrá kľúčovú úlohu pri zjednodušovaní priamej interakcie medzi človekom a strojom. Zatiaľ čo mnohé existujúce metódy odhadu póz využívajú hlboké neurónové siete, ktoré sú trénované na obrazových dátach, takejto forme obrazových dát často chýba podstatná hĺbková informácia, ktorá je nevyhnutná pre presný odhad pózy v 3D priestore. Na prekonanie tohto nedostatku boli navrhnuté iné dáta ako sú napríklad neštruktúrované mračná bodov. Avšak klasické konvolučné neurónové siete nie sú priamo aplikovateľné pri spracovaní neštruktúrovaných mračien bodov, okrem iného neštruktúrované mračná bodov nenesú žiadnu informáciu o susednosti jednotlivých bodov. Jedným sľubným riešením sú štruktúrované mračná bodov, ktoré vzniknú usporiadaním bodov do mriežky. V tejto práci sa zaoberáme skúmaním výhod a potenciálnych nevýhod štruktúrovaných mračien bodov pri odhade 3D pózy človeka.

Kľúčové slová: odhad 3D pózy človeka, štruktúrované mračná bodov, hlboké učenie

Abstract

Understanding human behavior requires precise estimation of human poses in both temporal and spatial dimensions. Pose estimation plays a pivotal role in facilitating direct human-machine interactions. While numerous existing methods for pose estimation rely on deep neural networks trained on image data, such data format often lack essential depth information crucial for accurate 3D pose estimation. To overcome this limitation, alternative media such as unstructured point clouds have been proposed. However, conventional convolutional neural networks are not directly applicable to point cloud data, since unstructured point clouds lack vital neighbouring information among points. One promising solution is the utilisation of structured point clouds, achieved by organising points into a structured 2D grid. In this thesis, we delve into investigating the advantages and potential drawbacks of employing structured point clouds for 3D human pose estimation.

Keywords: 3D human pose estimation, organised point clouds, deep learning

Contents

List of Figures

List of Tables

Introduction

In today's rapidly evolving fields of computer vision and artificial intelligence, there is a growing interest in automated processing of human behaviour. A critical aspect of this processing is the estimation of human body poses, which enables computers to recognise and interpret human body position and movements in space and time.

Accurate pose estimation is particularly crucial in domains like autonomous driving, where understanding the positions and movements of pedestrians, cyclists, and other road users is essential for safety. Moreover, pose estimation plays a vital role in human-machine interaction, enhancing natural communication between humans and machines. Gesture recognition, an integral part of this interaction, greatly benefits from precise pose estimation. In entertainment industries like film, television, and gaming, accurate pose estimation is indispensable for creating realistic animations through motion capture. Virtual reality experiences are also enhanced when human movements are accurately tracked and reproduced in virtual environments. Behavioural analysis relies on pose estimation to study and understand human actions and interactions in various contexts. Additionally, surveillance applications benefit from effective pose estimation, enabling the identification and tracking of individuals in crowded or complex environments.

It's evident that accurate pose estimation has broad applications across different domains. However, achieving precision in human body pose estimation is inherently challenging owing to several factors. Firstly due to the highly deformable nature of the human body and the diversity among individuals in terms of body shapes, skin colors, and clothing. Additionally, occlusions, whether from body parts or external objects, further complicate the task. While many existing methods can predict 2D poses, interpreting these poses can be challenging due to perspective projection. Therefore, 3D human body pose estimation is preferred, but it presents its own challenges, especially when depth information is lacking in image data.

It's evident that accurate pose estimation has broad applications across different domains. However, achieving precision in human body pose estimation is inherently challenging owing to several factors. Firstly, the human body is remarkably deformable, and individuals exhibit significant variability in terms of body shapes, skin tones, and attire. Additionally, occlusions, whether from body parts or external objects, further

complicate the task. Addressing these challenges necessitates the development of robust algorithms capable of handling the complex and dynamic nature of human body poses across varying contexts. Moreover, integrating advanced techniques for handling occlusions and leveraging additional cues, such as contextual information and temporal dynamics, can further enhance the accuracy and reliability of pose estimation systems.

This thesis aims to explore the potential of using point clouds augmented with topological information to enhance the accuracy and reliability of human pose estimation. The structure of this thesis is as follows: The first chapter provides a detailed exploration of key concepts in deep learning, pose estimation, and point clouds. The subsequent chapter reviews relevant publications in deep learning and pose estimation, offering a comprehensive summary. The third chapter discusses the dataset used and proposes tailored methods for the task at hand. The fourth chapter delves into implementation details. Finally, the last chapter evaluates the conducted experiments and draws conclusions from the findings.

Chapter 1

Problem Overview

In this section, we introduce key concepts relevant to this thesis. Initially, we define pose estimation, elucidating its significance and applications. Subsequently, we delve into the concept of point clouds, exploring their representation of spatial data and their relevance to our research. Finally, we distill essential machine learning principles, particularly focusing on deep learning theory, as it underpins various computer vision tasks, including pose estimation.

1.1 Pose Estimation

Pose estimation involves the computer vision task of determining the spatial position and orientation of objects or entities within an image or sequence of images. The term "pose" typically encompasses both the location (position) and the orientation (rotation) of an object. This field finds applications in various domains, including robotics, augmented reality, virtual reality, and human-computer interaction.

Human body pose can be represented in a variety of ways. It may be depicted as a simple contour of a human figure or by a volumetric model. One common structure for representing bodies is undirected graphs, where each vertex represents a point of articulation and stores the position of the joint in space.

There are two main types of pose estimation:

- 2D Pose Estimation: This involves determining the position and orientation of objects in a two-dimensional space, typically within a single image or frame. Common applications include tracking the position body in human pose estimation, facial feature tracking, or object localization in images.

- 3D Pose Estimation: Extending pose estimation to three-dimensional space provides information about the object's position and orientation in the 3D world. Applications include robotics, where the pose of objects or the robot itself needs

to be accurately determined, and augmented reality, where virtual objects need to be precisely placed in the real world.

Various methods can be employed for pose estimation, ranging from traditional computer vision techniques to more advanced deep learning approaches. Deep learning, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated significant success in 3D pose estimation tasks [20, 17, 18, 11].

Two approaches are commonly used for solving pose estimation:

- Top-down methods: These involve first detecting humans in an image and then estimating the pose for each person. A drawback of this method is that humans in images might overlap, leading to duplicated or completely incorrect estimated poses.

- Bottom-up methods: In this approach, each joint is detected, classified, and then organized to form a skeleton.

Pose estimation, whether in 2D or 3D, presents several challenges due to the complexity and variability of real-world scenarios. Some of the main problems and challenges in pose estimation include:

- Ambiguity: Certain poses or scenarios may present multiple valid interpretations of the same visual cues, leading to ambiguity in determining the true pose.

- Occlusion: Objects or body parts may be partially or completely occluded in the image, making it challenging to accurately estimate their poses.

- Variability in Appearance: Different lighting conditions, backgrounds, and viewpoints can significantly impact the appearance of objects or body parts, making it difficult for pose estimation models to generalise across diverse settings.

- Scale and Size Variation: Variability in the size of objects or individuals can affect the scale of body parts, making it challenging to maintain precision in the estimation of poses on different scales.

- Complexity of Articulated Structures: For human pose estimation, the articulated nature of the human body introduces complexity. Interactions and dependencies between joints and body parts need to be accurately captured, especially in dynamic scenarios.

- Real-time Processing: In applications like robotics, augmented reality, or gaming, real-time pose estimation is often required. Achieving low-latency processing while maintaining accuracy is a constant challenge.

- Limited Training Data: Annotated training data for pose estimation, especially in 3D, can be scarce and challenging to obtain. Limited data may lead to overfitting and hinder the generalisation of models to new environments.

- Computational Complexity: Advanced pose estimation methods, particularly deep learning approaches, can be computationally demanding. Efficient algorithms and hardware are required to ensure real-time performance, especially in resource-constrained environments.

- Accuracy in 3D Pose Estimation: Estimating the three-dimensional pose of objects or humans is inherently more complex than 2D pose estimation. Accurately determining depth information from images or sensors is a challenging problem.

Addressing these challenges often involves a combination of advanced algorithmic techniques, enhanced training data quality, and access to powerful computing resources. Ongoing research in computer vision and machine learning continues to advance the state-of-the-art in pose estimation methods, making them more robust and applicable in diverse real-world scenarios.

Human body pose estimation has a wide range of applications across various domains, and its importance stems from its ability to understand and interpret the spatial configuration of the human body. Some key applications and the significance of human body pose estimation include:

- Sports Analysis: In sports, pose estimation is used to analyze the biomechanics of athletes during training or competition. It provides insights into body movements, joint angles, and overall performance.

- Exercise Monitoring: Pose estimation is employed in fitness applications to monitor and guide users during exercise routines. It ensures proper form and helps prevent injuries.

- Virtual Fitting Rooms: Pose estimation enables the virtual try-on of clothing and accessories in online shopping applications. Customers can see how items fit and look on their bodies.

- Motion Capture: In gaming and animation, human pose estimation is used for motion capture, allowing characters to mimic the movements of real individuals. This enhances realism in virtual environments.

- Immersive Experiences: Pose estimation contributes to AR and VR applications by accurately placing virtual objects in the real world and enhancing interactions in immersive environments.

- Human-Computer Interaction (HCI): Pose estimation is crucial for recognizing and interpreting hand gestures, facial expressions, and body movements in HCI applications. It enables natural and intuitive interactions with devices.
- Healthcare and Rehabilitation: Pose estimation is used in healthcare for monitoring rehabilitation exercises. It assists therapists in assessing patient movements and progress.
- Person Tracking: Pose estimation aids in person tracking for security and surveillance purposes. It is used to monitor and analyze human movements in crowded areas.
- Workplace Safety: In industrial settings, pose estimation can be applied to assess the ergonomic aspects of workstations and identify potential risks to worker health and safety.
- Driver Assistance Systems: Pose estimation is utilized in driver monitoring systems to track the driver's posture, head position, and gaze direction. It enhances safety by alerting drivers to signs of drowsiness or distraction.
- Accessibility Technology: Pose estimation is integrated into assistive technologies to help individuals with disabilities control devices using body movements. It enables hands-free interaction.

The importance of human body pose estimation lies in its capacity to extract rich information about the spatial arrangement of body parts. This information, when accurately obtained, enables a diverse set of applications that enhance human-machine interactions, provide valuable insights in various fields, and contribute to the development of innovative technologies for both leisure and practical purposes.

1.2 Organised Point Clouds

Point clouds, commonly generated through photogrammetry software or 3D scanners, comprise finite sets of points representing object surfaces in space. Each point is spatially registered and may also include color information. These collections of points, often termed unstructured or unorganized point clouds, serve diverse purposes across machine learning, including segmentation and classification, facilitated by specialized deep learning architectures tailored for their processing. However, a limitation of point clouds is that, without preprocessing, they do not inherently convey explicit information about neighboring regions. One potential solution to address this limitation is to organize the points into grids, similar to RGB images. By arranging the points into 2D

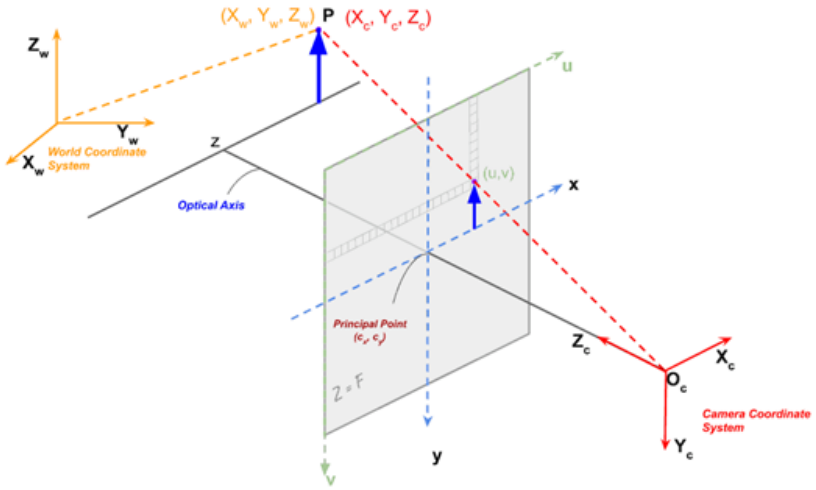


Figure 1.1: Figure represents how pinhole camera model projects points. Image taken from [4]

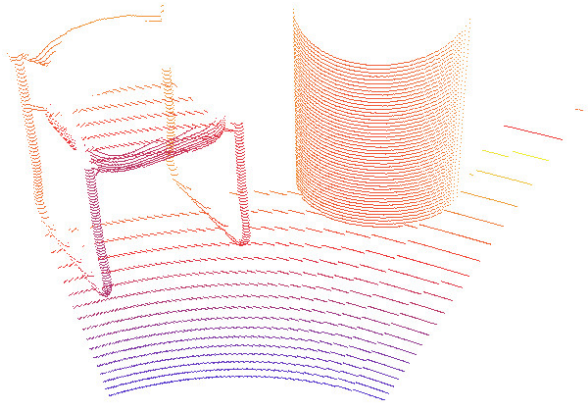


Figure 1.2: Figure depicting a point cloud. Image taken from [5]

grids, we can leverage deep learning models developed for image processing tasks. The process of organizing these points into a 2D grid can be elucidated using the pinhole camera model:

$$w\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = PX = [KR|t]X \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{pmatrix} \begin{pmatrix} X_x \\ X_y \\ X_z \\ 1 \end{pmatrix}$$

Where f_x, f_y, x_0, y_0, s are the intrinsic parameters of camera and model the behavior of optical lenses in real world camera. The curvature of lenses can cause distortion in projected image however the curvature can be often omitted. f_x, f_y are called the focal length and are usually equal, they directly model the distance of image plane from camera centre. (x_0, y_0) is the principal point, used to offset the projected 2D position so that the point (0, 0) aligns with the top-left corner of the image. They are usually

set as half of the image's width and height, respectively, while s represents the skew of the lens and is usually set to 0. The parameters corresponding to the camera's position and tilt in 3D space are known as the extrinsic parameters of the camera. Typically, the rotation matrix R is set as the identity matrix, and the translation vector t is set as zero. However, if we want to relate points between different cameras or coordinate spaces, both the rotation and translation need to be computed. Despite being called a rotation matrix, R performs a transformation from one coordinate system to another—specifically, from the world coordinate system to the camera coordinate system.

The world coordinate system can be arbitrarily selected, but for the camera coordinate system, we aim to position the origin of the camera system at the camera's location. The camera's view vector, or direction vector, is parallel to the z-axis. The y-axis and x-axis are perpendicular to each other, with the x-axis aligned with the camera's side vector and the y-axis aligned with the camera's up vector. The dot product of a unit vector and a vector corresponding to a point in 3D yields a scalar representing the position where the point is projected along the unit vector. With this understanding, we can compute the rotation matrix as follows:

$$\vec{f} = normalise(\vec{p} - \vec{e})$$

$$\vec{u} = (0, 1, 0)$$

$$\vec{r} = normalise(\vec{v} \times \vec{u})$$

$$\vec{u'} = \vec{r} \times \vec{f}$$

$$R = \begin{pmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 \\ \vec{u'}_1 & \vec{u'}_2 & \vec{u'}_3 \\ -\vec{f}_1 & -\vec{f}_2 & -\vec{f}_3 \end{pmatrix}$$

Here $\vec{p}$ is point the camera is focusing on and $\vec{e}$ is the "eye" or camera position, $\vec{f}, \vec{u}, \vec{r}$ are the camera's forward, up, and right unit vectors, respectively. Normalise is a function which computes the normalised unit vector as:

$$normalise(\vec{v}) = \frac{\vec{v}}{\|\vec{v}\|}$$

Where $\|\vec{v}\|$ is the length of vector $\vec{v}$. Since the camera is not necessarily centered at the origin of the world coordinate system, we must offset the projected point by the vector $\vec{t}$. This vector can be computed by projecting the camera position using the rotation matrix as:

$$\vec{t} = -R\vec{e}$$

Using homogeneous coordinate system, the pinhole camera model can be summarised as:

$$w\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = [KR|t]X = \begin{pmatrix} f_x & s & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 & t_x \\ \vec{u'}_1 & \vec{u'}_2 & \vec{u'}_3 & t_y \\ -\vec{f}_1 & -\vec{f}_2 & -\vec{f}_3 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_x \\ X_y \\ X_z \\ 1 \end{pmatrix}$$

where u and v are image coordinates. Since images consist of grids of fixed-size pixel elements, we are only interested in discrete values of u and v . Integers u and v provide information on how the pixel in the image relates to a point in 3D coordinate space. Typically, in the case of color images, we store only the discrete color value of points in 3D. However, additional information such as the world coordinates of point X or even its depth w can also be stored.

1.3 Machine Learning

The foundation of machine learning lies in statistical algorithms, which possess the capability to address non-trivial tasks. Machine learning, as a subfield of artificial intelligence, revolves around algorithms that can learn from data. According to Tom M. Mitchell, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." [16] Although this definition may seem somewhat abstract, Ian Goodfellow offers insight into interpreting tasks, experience, and performance intuitively. [10]

Typically, when a machine is assigned to solve a particular problem, we provide step-by-step instructions that ultimately lead to a solution. However, what if the task is challenging to define or to be solved through simple steps? For instance, while sorting a sequence of numbers may be straightforward for a machine, tasks inherently human, such as finger-counting from photographs, pose greater complexity. This problem consists of multiple sub-problems, each difficult to solve. Initially, the machine must learn to detect hands, then distinguish between different gestures, and finally output a corresponding number. Even if an algorithm for hand detection could be devised, would it adequately identify occluded hands or those of varying skin tones? What if the hand is captured from an unusual angle? Would such an algorithm generalise well? These challenges are addressed and potentially overcome through machine learning. Rather than prescribing step-by-step instructions, the machine predicts solutions based on previous experiences.

The aforementioned task is termed classification, where the machine predicts a label for a given input—for instance, determining which number a hand represents in a photograph. In the context of image processing, one might seek an algorithm to assign

semantic labels to each pixel of an input image; this task is known as segmentation. An concrete example of segmentation in practice is discerning pathological tissue from healthy tissue in medical images. Another task tackled by machine learning algorithms is regression, where the algorithm predicts a real number for a given input, such as estimating waist size from a human photograph. Numerous tasks, including language translation, sequence prediction, denoising, upscaling, and many others, are currently solved by machine learning algorithms.

Now that we understand the concept of tasks, we need to assess the performance of the machine in executing these tasks. Let's consider a dataset comprising of pairs of examples and their corresponding expected outputs:

$$D = (x_i, y_i)$$

Our machine learning algorithm predicts:

$$\hat{y_i} = f(x_i)$$

In regression tasks, the most commonly used loss functions are the L_1 loss:

$$L_1 = \frac{\sum_{i=1}^{n} |\hat{y_i} - y_i|}{n}$$

or the L_2 loss:

$$L_2 = \frac{\sum_{i=1}^{n} (\hat{y_i} - y_i)^2}{n}$$

Since the loss is squared, if the difference between prediction and target is significant, the errors can become quite large. Therefore, the L1 loss is more robust to outliers. Both L_1 and L_2 loss are defined in a manner that even if the mean difference between prediction and target is zero, the losses might not be.

In classification tasks, instead of predicting a number, we predict the probability that the input corresponds to each distinguishable class j:

$$\hat{y_i}^j = f(x_i)^j$$

Then, the cross-entropy loss CE is defined as:

$$CE = \frac{\sum_{i=1}^{n} \sum_{j=1}^{c} -y_i^j log(\hat{y_i}^j)}{n}$$

These loss functions are utilized during the training of machine learning algorithms. However, they may not directly indicate the performance of the algorithm. To evaluate performance more intuitively, metrics such as accuracy can be employed, which inform us about the proportion of examples correctly labeled. In regression tasks, the Euclidean distance may be used to measure how far apart predictions and ground-truth measures are.

In practice, one might assume that training our algorithm on all available data would yield better results. However, this approach fails to provide insights into the algorithm's performance on new, unseen data. Our goal is to train the algorithm and then apply it to solve tasks on unseen data. To ensure that the algorithm generalizes well to unseen data, we partition our dataset into two distinct sets: the training set and the validation set. The algorithm is trained on the training data, and its performance is evaluated on the validation set. Importantly, the algorithm does not learn from the validation set. Many machine learning algorithms consist of trainable weights, which are adjusted during training, and hyperparameters, which are predetermined before training. The overall performance of the algorithm may vary depending on the selection of hyperparameters. It is evident that choosing hyperparameters based on the performance on the validation set essentially involves manually training the algorithm on the validation data. However, this approach does not accurately reflect how well the algorithm will perform on unseen data. To address this issue, a third partition of the dataset, known as the testing dataset, is introduced. The algorithm is trained on the training dataset, hyperparameters are selected based on the validation dataset, and once the algorithm converges, it is evaluated on the testing dataset. It is crucial to emphasize that we must refrain from attempting to improve performance based on the testing dataset, as doing so would undermine the purpose of estimating performance on unseen data. In the previous section, we made the assumption that every data point in the dataset is labeled, enabling the machine learning algorithm to learn through supervision. Depending on how the machine interacts with the dataset, we categorize learning algorithms into supervised and unsupervised processes. In unsupervised learning, the data points lack associated labels. In this type of learning, the machine attempts to discern some probability distribution of the random vector x or uncover interesting underlying properties of the dataset. In reinforcement learning, the dataset size is not fixed, and the machine learns through interactions with the environment.

1.3.1 Deep Learning

Deep learning encompasses a suite of algorithms within machine learning that are based on artificial neural network. The term "deep" stems from the fact that they consist of multiple blocks, which are stacked consecutively. Each layer progressively extracts higher-level features. In image processing, for instance, the network initially grasps low-level features like edges, then progresses to discern shapes and blobs at lower levels, culminating in the comprehension of high-level concepts such as digits. Artificial neurons serve as the fundamental units, composed of trainable parameters. The subsequent section explains the most common building blocks of neural networks.

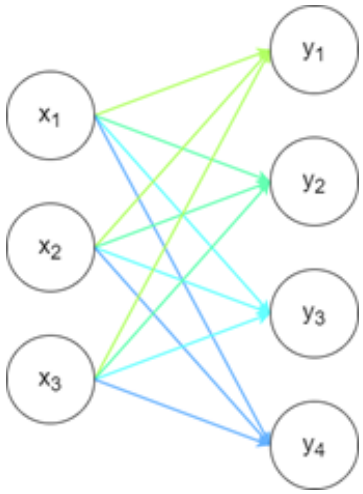


Figure 1.3: Scheme of a fully connected layer.

1.3.2 Multi-Layer Perceptron

The perceptron serves as a fundamental element in many deep learning methods, especially those geperforming regression or classification tasks. Perceptron is essentially simplified model of neuron. In classical neuron input signal is received by dendrites which determine the overall contribution of input signal in excitation of neuron. If the excitation exceeds a certain threshold the neuron will fire action potential across axon, which might potentially excite different neuron via synaptic terminal. Similarly, in a perceptron, input signals are represented as a vector, with each value weighted and summed by the perceptron to produce the final output. Through the application of a sign activation function, the perceptron can categorize input signals into two distinct categories. The weights of the perceptron effectively define a hyperplane—or in the case of a 1-D signal, a line—that facilitates binary classification based on whether the signal lies geometrically "above" or "below" this hyperplane, or if the input signal is sufficiently strong to cause the neuron to “fire”. Mathematically it can be summed as:
:

$$y = f(\vec{w}\vec{x} + b) = f\left(\left(\sum_{i=1}^{n} w_i x_i\right) + b\right)$$

Here, b represents the bias term, $\vec{w}$ denotes the perceptron weights vector, $\vec{x}$ signifies the input signal, and f represents the activation function. This formula can be written more compactly if we include the bias term in weight vector and simply add 1 into input vector corresponding to position of bias in weight vector:

$$y = f(\prime\vec{w}\prime\vec{x}) = f\left(\left(\sum_{i=1}^{n-1} \prime w_i \prime x_i\right) + \prime w_n 1\right)$$

However, this definition of the neuron poses a significant drawback, as it can only classify signals that can be separated linearly by a hyperplane—that is, they are linearly

separable. To address this limitation, multiple perceptrons are introduced to generate a feature vector, where each value represents the result of applying a single perceptron. This can be expressed mathematically as:

$$\vec{y} = f(W\vec{x})$$

Here, W represents the weight matrix, where each row corresponds to a single perceptron, and $\vec{y}$ denotes the output feature vector. These perceptrons form a fully connected layer, which, when stacked, form a multi-layer perceptron capable of discerning nonlinearly separable signals. Non-linear activation functions are essentials for proper deep neural networks, since without the non-linearity the multi-layer perceptrons collapses to simple fully connected layer:

$$\vec{y} = B(A\vec{x}) = C\vec{x}$$

Various activation functions are employed, each with its own set of advantages and drawbacks. Among them, rectified linear units (ReLUs) stand out as the most widely used activation functions in the hidden layers of neural networks:

$$f(x) = max(x, 0)$$

ReLUs offer straightforward computation; however, they are not zero-centered, which can pose challenges in deep learning, and their derivative is zero for negative values, leading to optimization issues. Several alternatives to ReLUs exist, each addressing these concerns with its own trade-offs. For the output layer, activation functions such as softmax, tanh, sigmoid, or linearity are utilized, depending on the nature of the task. When the objective revolves around predicting probabilities, softmax or sigmoid activations are preferred, as they confine outputs to the range of zero to one. In regression tasks, tanh or linear activation functions may be employed.

Now that we have gained some intuition about what machine learning is and how neural networks work, we are left to wonder how to select proper parameters/weights in the network. This process is performed iteratively by an algorithm called stochastic gradient descent (SGD). Overall, deep-learning algorithms are based on the assumption that we already have some deep neural network. This allows us to build a computational graph beforehand, which becomes useful in gradient computation. Using our loss functions, we evaluate the performance of the proposed neural network. With the help of our loss function, we attempt to find the minimum of the loss function with respect to each weight. Since each layer is known and defined as a function in the computation graph, we can perform first-order derivation using the chain rule to compute the gradient with respect to each parameter. The gradient essentially tells us which direction we need to travel to reach a local minimum of the loss function.

The challenge with stochastic gradient descent is the fact that we are unlikely to find a global optimum. Nowadays, vanilla stochastic gradient descent is no longer employed, and many different optimization methods are used, with the most popular being the Adam optimizer. However, every single method is based around gradient computation.

1.3.3 Convolutional Neural Networks

In the realm of computer vision, which primarily revolves around images, a critical question arises: How can deep learning be effectively applied to tasks like image classification or segmentation? One might initially consider a simplistic approach of flattening 2D images into 1D vectors and employing a multilayer perceptron. However, it becomes apparent that this method is impractical. For instance, with images of dimensions 512 pixels by 512 pixels, the flattened image results in a 262,144-dimensional vector. Let's say we want to generate feature vector consisting of 1024 values, the weight matrix alone would comprise of 268,435,456 entries. This translates to one gigabyte of RAM memory, assuming each entry can be stored in 4 bytes. Furthermore, such an approach requires fixed-size images, and even a slight shift in the image could lead to erroneous classifications by the multilayer perceptron. A more memory-efficient and effective method for image processing is convolution. Rather than learning weights per pixel, convolution involves learning weights within a sliding window, known as a kernel. Mathematically, convolution is a operation which produces new function by composing two functions f and g, which are both real valued.

$$(f * g)(t) = \int f(a)g(t-a)\,da$$

However, as images are finite discrete structures represented as a two-dimensional grid of pixels, the convolution is simplified to:

$$(f * g)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Here, K represents the kernel, I denotes the 2D image, m and n are the kernel indices, and i and j are the 2D coordinates in the image. By employing handcrafted kernels in image processing, tasks such as denoising or edge extraction can be achieved. However, the true power of convolutional networks lies in their ability to automatically learn these kernels and extract the most relevant features. Convolution in image processing can be viewed as generating feature maps, where each value is produced by applying a weight matrix to a specific image region. This operation is memory efficient, as the kernel is smaller than the input image, permits parameter sharing, is robust to certain image transformations (e.g., shifting of the image), and allows images to be of variable size and shape. As the kernel can only be applied to regions in the image,

the resulting feature map is downsized, this can be solved by applying padding to the input image. Zero-padding, where zeros are added to the borders of the image, is the most commonly used padding technique in deep learning, ensuring maintenance of the original resolution. The final resolution along a given axis of the feature map can be computed using the formula:

$$out = \left\lfloor \frac{in + 2p - 5k}{s} \right\rfloor + 1$$

Where p represents padding, k denotes the size of the kernel along a specific axis, in indicates the input size, s is the stride (i.e., the number of pixels the kernel moves each step), and $\lfloor x \rfloor$ denotes the floor operation. Since input images essentially contain RGB values, the convolution must also be performed along the depth of the image. This entails applying a different kernel along each input channel separately and summing the results:

$$(I * K)(i, j) = \sum_c \sum_m \sum_n I(i - m, j - n) K(m, n, c)$$

In deep learning, generating a single feature map is often insufficient, therefore multiple feature maps are generated using multiple 3D kernels. The number of parameters required by a single convolutional layer is $h \times w \times c \times d$, where h and w represent the kernel dimensions, c denotes the number of features of the input image, and d indicates the number of channels of the generated feature map. Many deep learning architectures are split into encoder and decoder parts. The idea is that for given task the input images contain redundant or unimportant information, therefore the encoder part tries to extract relevant features and decoder part upsamples these features to generate heatmaps, segmentations, and so forth. Another commonly employed method for gradually generating feature maps of lower resolutions involves pooling layers. Similar to convolutions, pooling is applied over image regions of fixed size and spaced by a stride value. Instead of applying a kernel with trainable weights, the feature in each region is aggregated using a specific function. The most prevalent functions used are average pooling or max-pooling. In average pooling, the mean is computed for the values in the region, while in max-pooling, the region is represented by the largest value within it. Similar to convolution, pooling operations are invariant to small translations of the input image.

CNN architectures can vary in depth and complexity, with deeper networks generally having a greater capacity to learn intricate patterns. Popular CNN architectures include LeNet-5, AlexNet, VGGNet, GoogLeNet (Inception), ResNet, and many more.

One interesting type of convolution is deformable convolution. While ordinary convolutions can only sample from discrete positions in input feature maps, deformable convolutions, on the contrary, can adjust the sampling by incorporating learnable off-

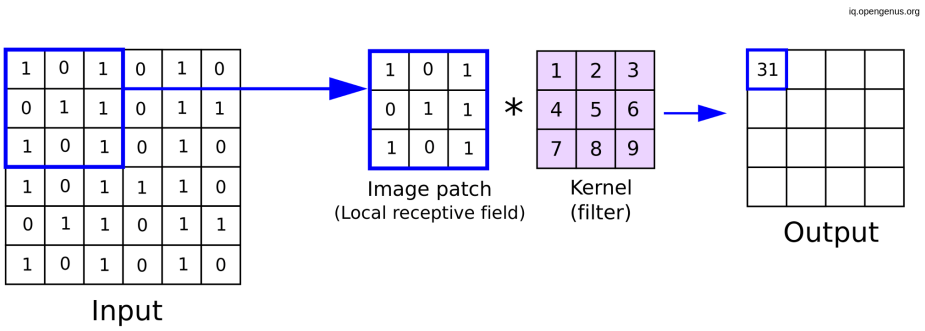


Figure 1.4: Depiction of how convolution operates, image taken from [2].

sets. If convolution can be represented as:

$$y(p_0) = \sum_{p_n \in R} w(p_n) x(p_0 + p_n)$$

Where p_0 denotes the position in the output feature map and p_n represents all positions in the kernel, then by introducing trainable offset Δp_n, we obtain deformable convolution as:

$$y(p_0) = \sum_{p_n \in R} w(p_n) x(p_0 + p_n + \Delta p_n)$$

To access features at non-discrete positions, bilinear interpolation is utilized, or more precisely, a bilinear interpolation kernel is employed. The function $x(p)$ can then be expressed as:

$$x(p) = \sum_{q} G(q, p) x(q)$$

Where G represents the bilinear interpolation kernel, p indicates the non-discrete position, and q corresponds to neighboring discrete positions in the feature map. The bilinear interpolation kernel $G(q, p)$ is defined as:

$$G(q, p) = g(q_x, p_x) g(q_y, p_y)$$

$$g(a, b) = max(0, 1 - |a - b|)$$

It is evident from $g(a, b)$ that pixels farther than one pixel away are evaluated as zero. To generate the offset field, a convolutional layer is used. The resulting feature map predicts x and y offsets for each channel separately. One noteworthy application of deformable convolution is in the method [26], which presents a novel approach for object detection.

As previously discussed, the inclusion of pooling and potentially convolutional layers in the network architecture can lead to a reduction in the resolution of the outputted feature maps compared to the input image. However, in scenarios where it's necessary to predict heatmaps or segmentation maps with the same resolution as the input images, it becomes imperative to enhance the resolution of the predicted feature maps.

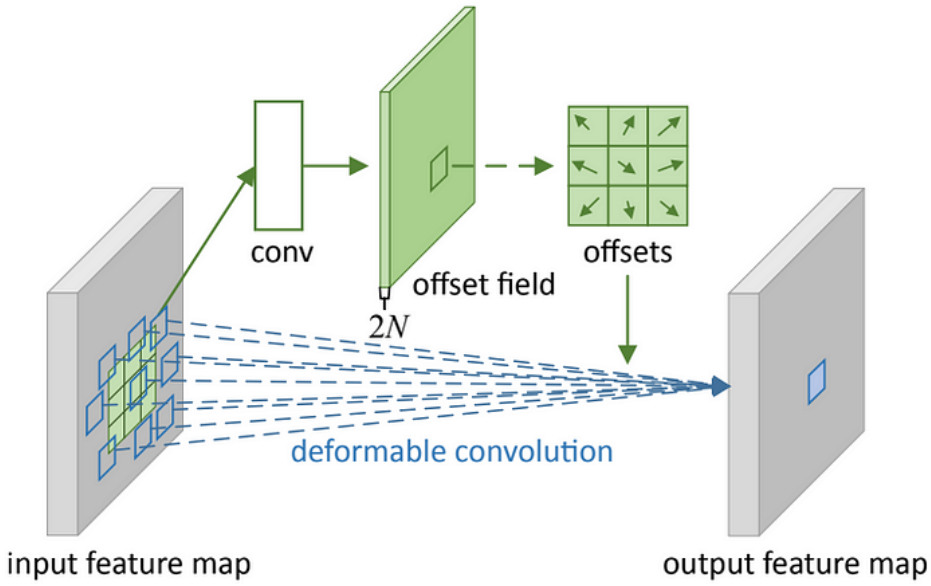


Figure 1.5: Scheme of how deformable convolution operates, image taken from [3].

One straightforward approach for achieving this is through unpooling. Unpooling essentially aims to reverse the effect of pooling operations. If pooling were to be followed by unpooling, the resulting feature map would ideally match the original input. There are three primary methods of unpooling: nearest neighbor, bed of nails, and max unpooling:

- Nearest neighbor involves copying the value of the input into each corresponding region of the output.
- Bed of nails simply duplicates input values into the top-left corners of the corresponding region while filling the other positions with zeros.
- In max unpooling, the spatial positions of the maximum values selected during the corresponding max pooling operation are stored. Subsequently, during unpooling, these stored positions are utilized instead of simply copying values into the upper-left corner, as in the bed of nails method.

One other method of increasing the spatial dimensions is interpolation, there exist many interpolation methods, with the most direct one being bilinear interpolation. Bilinear interpolation is an extension of linear interpolation. In case we want to compute new value x, between two know data point, we can linearly interpolate as:

$$y(x) = y_0 + (x - x_0)\frac{(y_1 - y_0)}{(x_1 - x_0)}$$

In case of bilinear interpolation, we know four data points and interpolate between them. We first interpolate in x-axis:

$$f(x, y_0) = \frac{(x_1 - x)}{(x_1 - x_0)} f(x_0, y_0) + \frac{(x - x_0)}{(x_1 - x_0)} f(x_1, y_0)$$

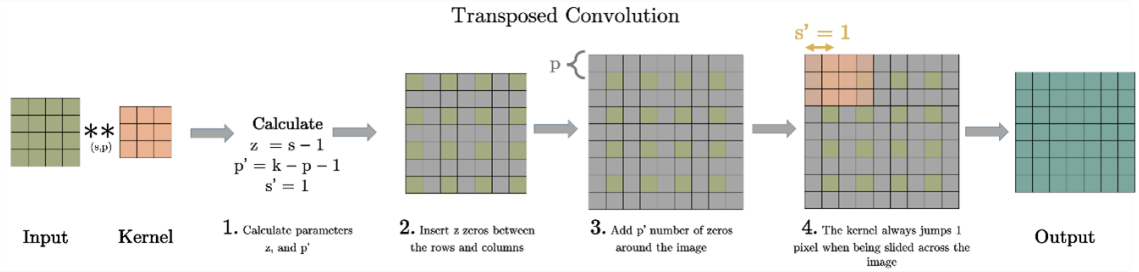


Figure 1.6: Transpose convolutions effectively upsample the input feature map. Initially, new parameters z are computed to determine the number of zeros to be inserted between features. Subsequently, padding p' is calculated to pad the features with zeros. Finally, the feature map is convolved by a kernel with a stride of 1 to generate the final upscaled feature map. Image was taken from [6].

$$f(x, y_1) = \frac{(x_1 - x)}{(x_1 - x_0)} f(x_0, y_1) + \frac{(x - x_0)}{(x_1 - x_0)} f(x_1, y_1)$$

Then we interpolate intermediate values in y-axis:

$$f(x, y) = \frac{(y_1 - y)}{(y_1 - y_0)} f(x, y_0) + \frac{(y - y_0)}{(y_1 - y_0)} f(x, y_1)$$

However, a drawback of both unpooling and interpolation methods is their lack of trainable parameters, which restricts the learning capabilities. This limitation is addressed by transpose convolutions, sometimes incorrectly referred to as deconvolutions. Similar to standard convolutions, transpose convolutions are characterized by parameters such as stride, padding, and kernel size. Padding and stride correspond to values, which had to be used to generate the input feature map by convolution. The size of output feature map along single dimension is computed as:

$$out = (in - 1)s + k - 2p$$

For a visual representation of how transpose convolution upsamples the input feature map, refer to the figure provided.

Chapter 2

Related Work

In this chapter, we delve into the methodologies employed for classification tasks, often serving as backbones for various applications. Additionally, we explore techniques utilized in both 2D and 3D pose estimation, with some leveraging temporal information to enhance accuracy and robustness.

2.1 Classification Networks in Computer Vision

Multitude of existing frameworks for human body parsing use similar backbones, used previously for image classification. Generally speaking these networks are used to generate some global feature, which are used in later stages to predict final pose.

In the realm of deep learning for image processing, the challenge arises from the inherent variability in image resolutions. To effectively capture information from both smaller and larger objects within images, a strategic choice of kernel sizes becomes imperative. Smaller kernels are adept at extracting features from diminutive objects, while larger kernels are more suited for comprehensively covering substantial portions of an image. While the augmentation of a deep-learning model's depth and size is essential for enhancing performance, it brings along inherent drawbacks such as heightened computational complexity and potential training instability. GoogLeNet, a pioneering convolutional neural network architecture, ingeniously addresses these challenges. At its core, the innovation lies in the utilization of inception modules. These modules employ one-by-one convolutions to initially reduce input dimensions and then process the input at various scales using kernels of different sizes. The results of these operations are concatenated and fed into subsequent layers, enabling the network to capture intricate features at multiple scales. This not only optimizes the model's efficacy in handling objects of different sizes but also significantly reduces the number of operations, leading to a more expedient training process. In addition to mitigating computational complexities, GoogLeNet tackles the challenge of vanishing or exploding

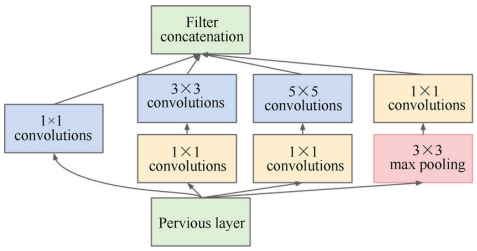


Figure 2.1: Scheme of an inception module. Image taken from [21].

gradients in deep networks. This is achieved through the incorporation of loss calculations from both the final layer and intermediate layers, suitably scaled down. By preventing the degradation of gradients, GoogleNet ensures robust training and effective feature extraction throughout its relatively deep architecture. The culmination of these innovations propelled GoogLeNet to victory in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2014, solidifying its status as a pioneering solution in the field of deep learning for image classification tasks [21].

The fundamental aspect of numerous deep learning models lies in their depth, as deeper models can progressively extract higher-level features from input images. While the intuitive expectation is that stacking multiple layers should enhance a model's accuracy by allowing it to learn intricate patterns, empirical evidence has revealed a counter-intuitive phenomenon. Simply adding layers to a model, even when it is already proficient in solving a given problem, does not guarantee improved accuracy; in fact, it often leads to degradation in training accuracy. To address this unexpected challenge, the concept of identity mapping is introduced. If we assume that a model is adept at solving a problem, adding new layers should ideally allow the model to learn an identity mapping to preserve its problem-solving ability. However, the conventional approach of stacking layers often hampers training accuracy. This limitation is resolved by reformulating the problem: instead of explicitly learning the identity mapping $f(x)$, the model is designed to push the weights of redundant layers to zero. The output Y is then expressed as the sum of $f(x, W)$ and the input x. In cases where the dimensions of f and x do not align, a linear projection is employed to ensure compatibility:

$$Y = f(x, W_1) + W_2 x$$

These constructs are referred to as residual connections, forming the main idea of the ResNet architecture, which reached the top spot in the ILSVRC 2015 classification competition. Beyond their role in preserving model accuracy, residual connections offer an additional advantage by mitigating the issues associated with vanishing/exploding gradients during training. Presently, many state-of-the-art architectures incorporate ResNets as a backbones for extracting features from images, underlining their enduring impact on the landscape of deep learning [12].

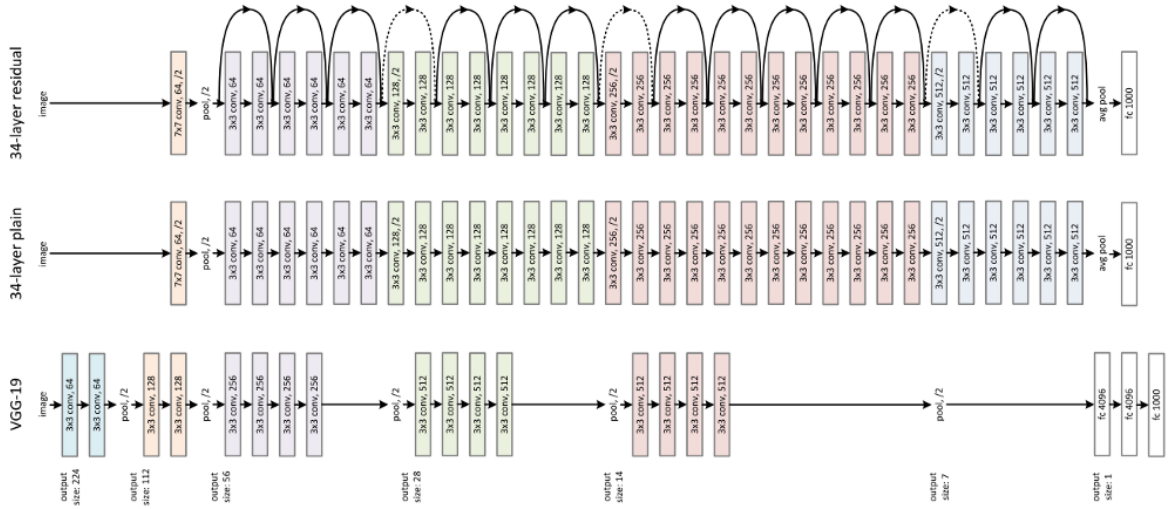


Figure 2.2: Comparison of ResNet with VGG-19. Image taken from [12].

A notable advancement in the realm of processing unstructured point clouds for classification and segmentation tasks is PointNet. Unlike earlier approaches that either voxelized point sets or reprojected them into 2D images, PointNet stands out as one of the pioneering networks capable of directly handling point sets. Its architectural design revolves around the unique properties inherent in point sets, where each point is represented by arrays and maintains arbitrary order, necessitating an architecture that disregards this order. The overarching challenge addressed by PointNet lies in ensuring robustness to transformations of point sets while capturing local relationships among nearby points. This is achieved through the incorporation of shared multilayer perceptrons, max-pooling operations, and T-nets that transform point features into canonical space. A key aspect is the utilisation of max-pooling along each feature vector, enabling the network to extract global features that remain invariant to permutations. However, one limitation of the original PointNet architecture is its efficiency in capturing local features of points. This limitation spurred the development of PointNet++, an enhancement that employs sampling techniques to extract regions of points and applies PointNet on subsets of the input point cloud. By strategically addressing this limitation, PointNet++ further refines the capacity to capture both global and local features, contributing to the ongoing evolution of point cloud processing methodologies. [19]

2.2 Human Pose Estimation Methods

Since there is a high abundance of images, most of the methods currently estimate 3D pose from RGB images. Due to lack of depth estimating absolute pose proves to be difficult and many approaches estimate both 2D heat-maps and depths.

Numerous studies have delved into the realm of 2D pose estimation, progressively

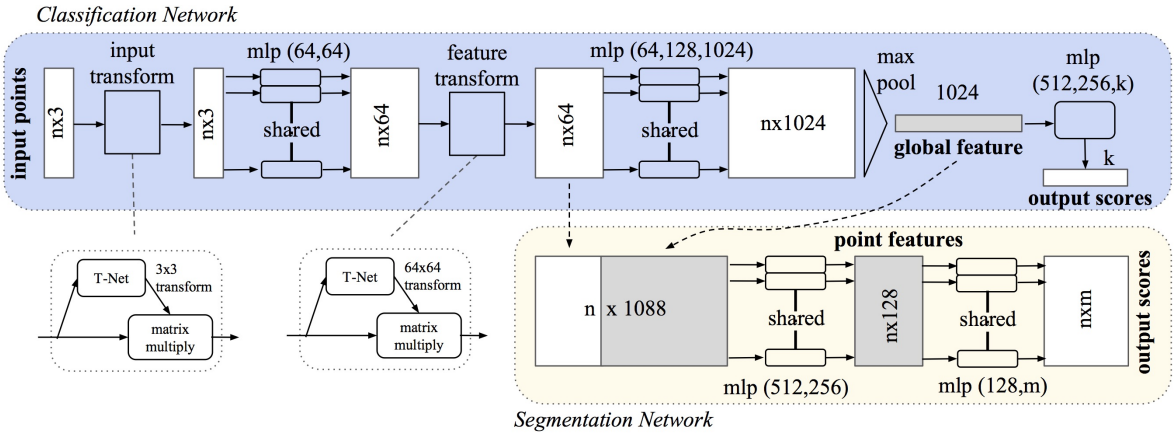


Figure 2.3: The proposed PointNet architecture for classification and segmentation. Image taken from [19].

introducing more intricate methods. The lingering curiosity revolves around the potential effectiveness of a simpler model for 2D pose estimation. This inquiry finds its resolution in the work titled "Simple Baselines for Human Pose Estimation and Tracking," where a straightforward yet highly efficient framework is proposed for pose tracking and 2D pose estimation. The pose estimation network in this approach employs ResNet as its backbone, integrating three transpose convolutions to generate heat maps. For person detection, the framework leverages the optical flow field to propagate joint positions from previous frames, enabling the prediction of new bounding boxes. A notable advantage lies in the framework's ability to generate bounding boxes even for occluded individuals. To accurately label bounding boxes, the system utilizes object keypoint similarity to establish connections between IDs from preceding frames. The framework employs both R-FCN and information from previous bounding boxes to estimate and predict new bounding boxes effectively[23].

3D Human Pose Estimation from Monocular Images with Deep Convolutional Neural Network marked a pioneering venture into deep learning methods for 3D pose estimation by Li et al., who introduced a model alongside two distinct training strategies. The proposed network adeptly addresses two primary tasks: body joint regression and joint detection. In the first strategy, the framework undergoes pretraining for detection followed by regression, while the second strategy involves simultaneous learning of both tasks. The objective of the joint regression task is to predict the displacement vector of each joint from its parent joint. This formulation proves advantageous as it enables the model to grasp limb lengths, distances between joints, and aids in handling occlusion of body parts. For the joint point detection task, the input image is partitioned into windows, and the network endeavors to detect joints within each window. The overall architecture is elegantly straightforward, comprising multiple convolutional layers followed by max-pooling, which generates features for subsequent regression. Both

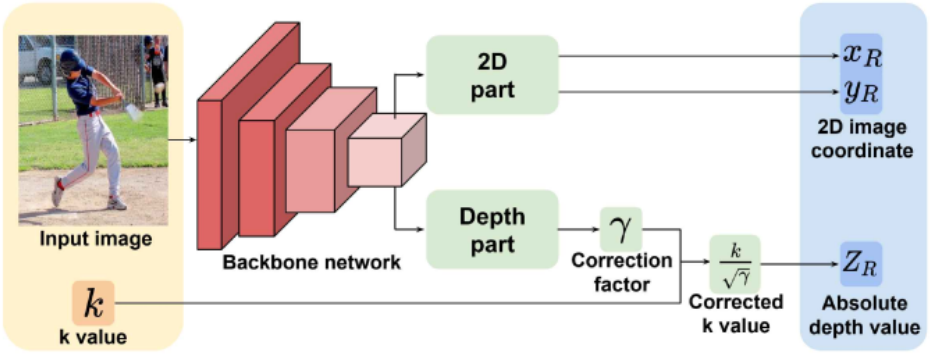


Figure 2.4: Network architecture of the RootNet. Image taken from [17].

regression and detection tasks are addressed using fully-connected layers. In the case of the detection network, the output is reshaped into 10 by 10 images for each joint, while for regression, the network outputs three coordinates for each joint. Evaluating the model's performance on the Human3.6 dataset, the Mean Per Joint Position Error (MPJPE) ranged from 78 to 189 mm, reflecting the efficacy of the proposed approach in 3D pose estimation [13].

An innovative approach to multi-person 3D pose estimation involves the prediction of joint 2D heat-maps with depths, subsequently reprojecting them to calculate the absolute pose of each individual. This intricate network comprises of three key components: DetectNet, RootNet, and PoseNet. DetectNet employs Mask R-CNN to predict bounding boxes within images. In its initial stage, global features of the image are computed, followed by the proposal of candidate bounding boxes in the second stage, and, in the third stage, the classification of bounding box features to determine whether they contain a human or not. RootNet is responsible for estimating the absolute position of the root joint. Utilizing camera intrinsics and a pinhole camera model, the network predicts the 2D coordinates of the root, incorporating a correction factor that signifies the correlation between the predicted bounding box and image depth. The root position is then determined through reprojection. Similarly, RootNet employs ResNet as the backbone, followed by transpose convolutions that generate a 2D hea tmap of the root. For the estimation of the correction factor, backbone features are pooled and convolved. PoseNet, like RootNet, utilizes ResNet as its foundation, followed by three transpose convolutions and a 1x1 convolution to predict 3D heatmaps of root-relative joints. With the predicted root depth and relative root joints, the 3D position of joints can be estimated using reprojection. This comprehensive approach integrates sophisticated techniques across DetectNet, RootNet, and PoseNet to achieve accurate multi-person 3D pose estimation [17].

The objective of 3D human pose estimation is to precisely determine the positions of joints and vertices. Given the computational infeasibility of computing 2D heat

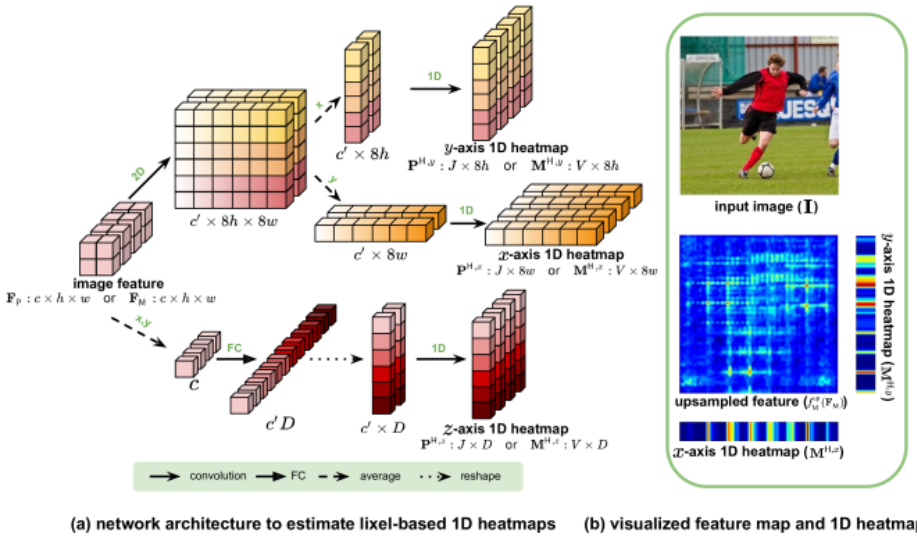


Figure 2.5: Network architecture to predict lixel-based 1D heatmaps. Image taken from [18].

maps and depth for each vertex in a mesh, I2L-MeshNet introduces the concept of "lixels," derived from the fusion of lines and pixels, akin to voxels. The proposed architecture, I2L-MeshNet, comprises two essential components: PoseNet for 3D pose estimation and MeshNet for mesh reconstruction. PoseNet utilizes ResNet to generate global image features, which are subsequently upscaled three times using transpose convolution. These upsampled features undergo axis-wise averaging to generate 1D lixel-based heatmaps for both the x and y coordinates. The z-axis heat map is derived from averaging the global features. In parallel, MeshNet, mirroring PoseNet, takes as input image features from the first layer of PoseNet and a 3D Gaussian heat map representing the predicted pose. MeshNet, also leveraging ResNet, predicts the positions of lixels. With the known intrinsic and extrinsic parameters of the camera, the 3D positions are back-projected, yielding the final positions of vertices. This innovative approach encapsulated in I2L-MeshNet integrates PoseNet and MeshNet seamlessly to tackle the intricacies of 3D human pose estimation and mesh reconstruction [18].

Due to occlusion or self-occlusion deep-learning models may over-fit, for this vary reason we can relax the network by uncertainty as shown in the publication: Single image based 3D human pose estimation via uncertainty learning. This approach leverages the ResNet architecture to extract comprehensive global features from the input image, subsequently employing transpose convolutions to upscale these features and generate 2D heat maps for each joint. To extend this to a 3D heat map, 1D convolutions are applied to augment heat map depth, while spatial information is preserved through skip connections, reminiscent of the U-Net architecture. Given the non-differentiability of the max function, a soft-argmax operation is employed to compute 3D key-point coordinates. This operation scales each discrete point in space based on the corresponding

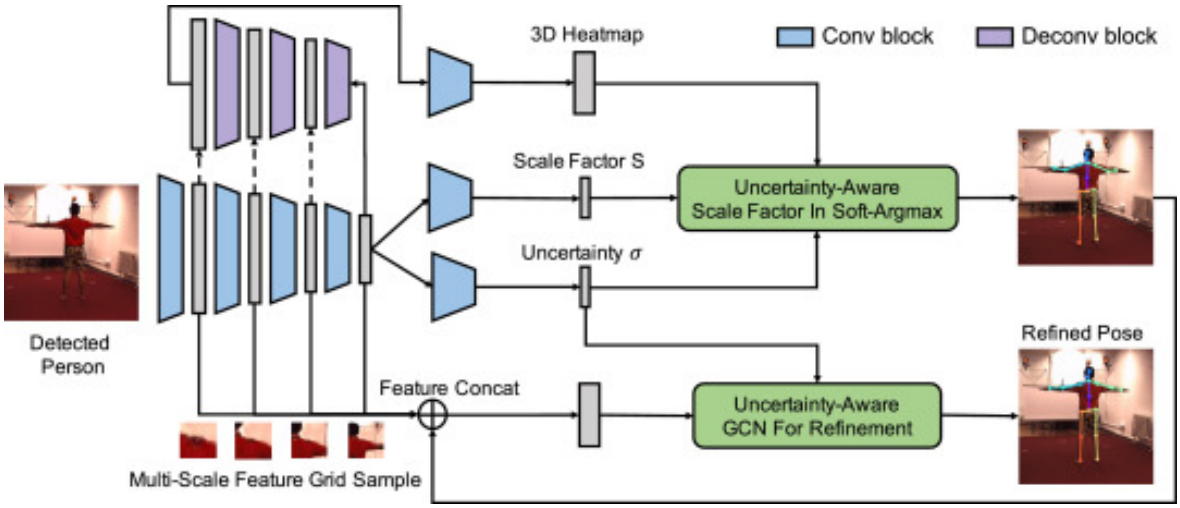


Figure 2.6: The proposed uncertainty-based 3D human pose estimation framework. Image taken from [11].

value in the 3D heat map, and the weighted average yields the final point position. Addressing challenges related to self-occlusion, the model incorporates the ability to predict uncertainty for each joint estimation. The uncertainty is modeled using a Laplace distribution, parameterised by mean and variance. The mean represents the difference between the ground truth and prediction, while the variance corresponds to the uncertainty. These two terms are harmonized in the loss optimization process. Additionally, the predicted uncertainty influences the scaling of the arg-softmax operation; higher uncertainty prompts a behavior akin to averaging, while lower uncertainty leads to argmax-like behavior. Recognizing potential self-occlusion in certain joints, a graph convolutional network is introduced to learn inter-joint dependencies. Each node in this network represents a body joint and is weighted by learned weights and associated uncertainty. On the Human 3.6M dataset, the proposed method demonstrates noteworthy performance with an average Mean Per Joint Position Error (MPJPE) of 46.9 and 66.7 for Protocol 2. This underscores the model's effectiveness in 3D human pose estimation, particularly in scenarios involving uncertainty and self-occlusion [11].

Due to boom cause by transformers in natural language processing, many computer vision scientist wonder if transformers can be used in field of computer vision. Overall it seems they might have their use especially in sequence modelling. In case of pose estimation transformer encoder base approaches already exist in abundance. The main building blocks of transformers are the self-attention mechanism and feed-forward networks. The self-attention mechanism operates as follows:

$$K = xW^K$$

$$Q = xW^Q$$

$$V = xW^V$$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here d_k represents the dimension of the key and query vectors. In the original paper it is proposed that using multiple linear projections are beneficial. The projections are processed by multi-head attention mechanism in following way:

$$Multihead(Q, K, V) = concat(head_1, ..., head_h)W^O$$

$$head_i = attention(QW_i^Q, KW_i^K, VW_i^V)$$

The feed-forward networks in transformers consist of two fully-connected layers separated by a rectified linear unit (ReLU) activation function.

VitPose revolutionizes 2D pose estimation by harnessing the power of visual transformers. In this innovative architecture, input images are strategically partitioned into 16 patches, each undergoing a unique embedding process that incorporates positional information. Leveraging the inherent permutation invariance of transformers, this method ensures robust feature extraction. The core of VitPose lies in its application of multiple multi-head self-attention layers and feed-forward layers, working collaboratively to generate high-level image features. When it comes to decoding global features, two distinct approaches are introduced. The first employs a conventional transpose convolution, a tried-and-true method for predicting heat maps. The second approach takes a different route, utilizing simple bi-linear interpolation followed by a convolutional layer, offering an alternative perspective for heat map generation. While VitPose boasts impressive capabilities, it's worth noting that its training process can be time-consuming. To address this, a strategic workaround involves utilizing a pre-trained model for subsequent stages of 2D pose processing. This not only streamlines training time but also capitalizes on the knowledge and patterns learned from diverse datasets [24].

HSTFormer emerges as a groundbreaking 2D-to-3D lifting methodology centered around transformer-based architectures. At its core, the primary innovation lies in the incorporation of four transformer blocks meticulously designed to process joint sequences at varying scopes. The overarching architecture seamlessly integrates with a conventional off-the-shelf 2D pose estimation network, embedding the resulting 2D positions for further refinement through the transformer encoder blocks. The sequence of joints undergoes a comprehensive transformation through the stacked encoder blocks, each contributing uniquely to the model's understanding of spatial and temporal relationships. The process unfolds with the Spatial Transformer Encoder (STE), devoted to mastering the spatial intricacies among different joints within a fixed frame. Subsequently, the Joint Temporal Transformer Encoder (JTTE) hones in on the temporal

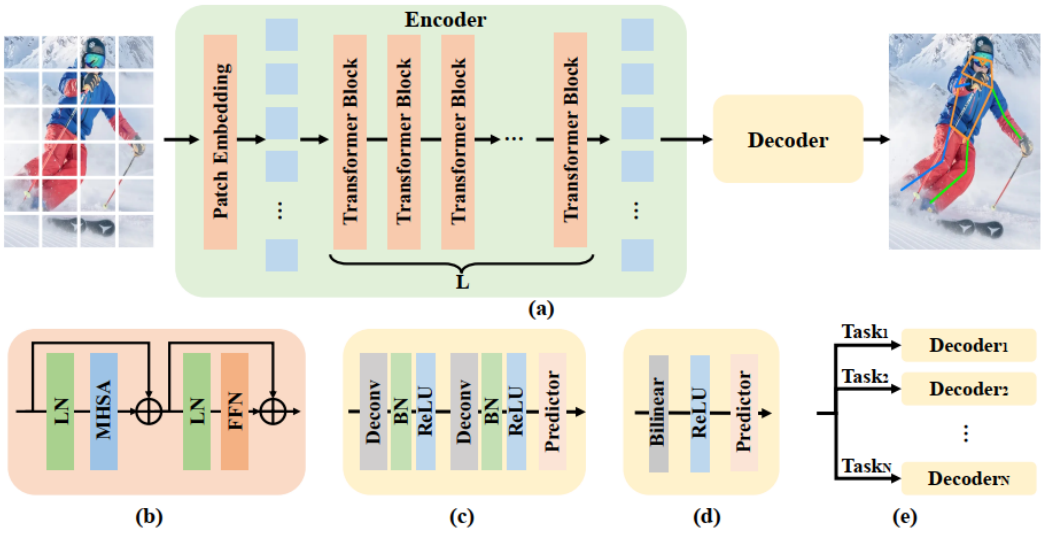


Figure 2: (a) The framework of ViTPose. (b) The transformer block. (c) The classic decoder. (d) The simple decoder. (e) The decoders for multiple datasets.

Figure 2.7: The proposed VitPose architecture. Image from [24].

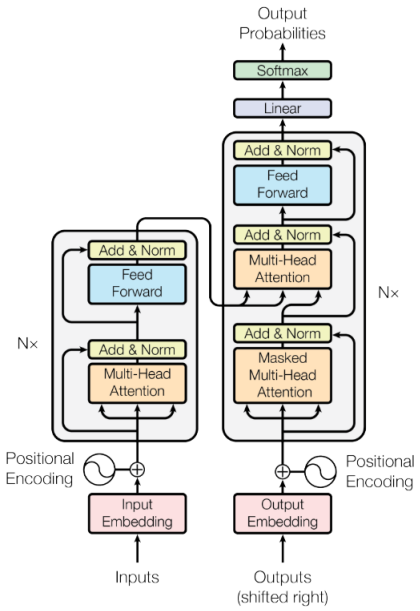


Figure 2.8: The transformer architecture taken from [22]

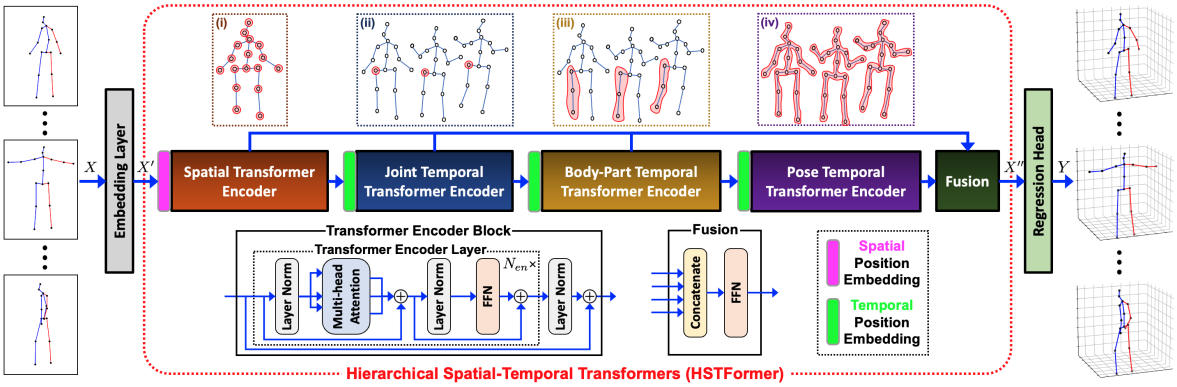


Figure 2.9: Overview of HSTFormer for 3D pose estimation. Image from [20].

dynamics of each joint across successive frames. As the sequence progresses, the input is partitioned into distinct body pose regions, paving the way for the Body-part Temporal Transformer (BTTE) to delve into the spatial and temporal relationships inherent to each specific body pose region. The last last transformer used is Pose Temporal Transformer Encoder (PTTE), a pivotal component that captures the intricate dependencies of each pose joint across the entirety of the frame sequence, unraveling the global pose-level temporal correlations. In the final stage of proposed framework, the outputs from these transformer blocks are concatenated and regressed in order to prediction the final 3D joint positions. Overall, the model performed on Human 3.6M with average MPJPE of 42,7 under protocol 1 and 33,7 (P-MPJPE) under protocol 2. HSTFormer, with its intricate transformer-based architecture, stands as a testament to the continual evolution of 2D-to-3D pose lifting techniques, offering a robust and accurate solution for diverse applications in computer vision [20].

One of the few robust frameworks which use point clouds for estimating 3D pose of both hand and human is PointLSTM. This research addresses the challenge of effectively employing Long Short-Term Memory (LSTM) networks in the context of unstructured point clouds, particularly when dealing with sequences where the points are not aligned. LSTMs, being recurrent neural networks, possess the unique ability to forget previous inputs, thus mitigating the vanishing/exploding gradient problem commonly encountered in traditional recurrent networks. LSTMs consist of four gates, summarized as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1})$$

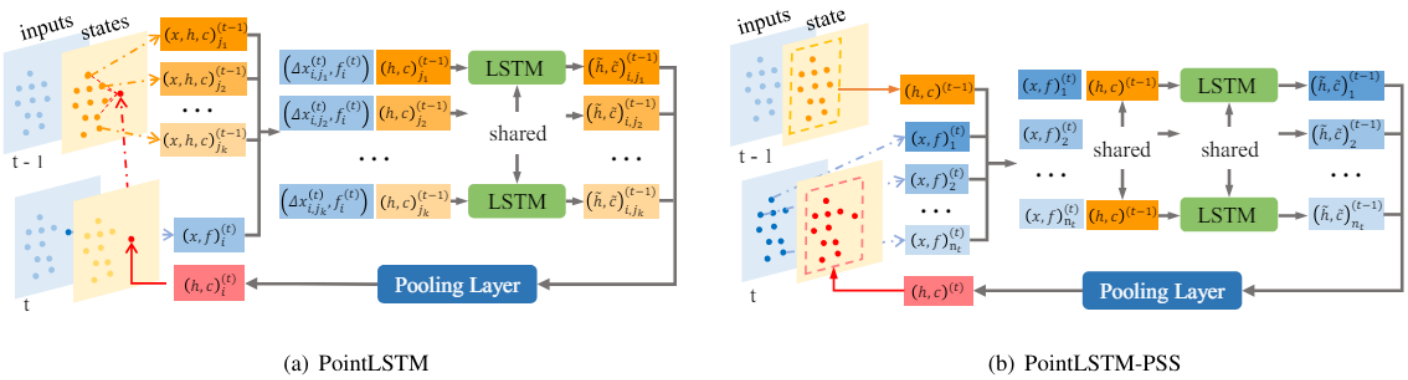
$$i_t = \sigma(W_i x_t + U_i h_{t-1})$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1})$$

$$\tilde{c_t} = tanh(W_c x_t + U_c h_{t-1})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t}$$

$$h_t = o_t \odot tanh(c_t)$$



(a) PointLSTM (b) PointLSTM-PSS

Figure 2.10: Overview of PointLSTM and PointLSTM-PSS. Image from [15].

Here, σ represents the logistic activation function, tanh denotes the hyperbolic tangent, $\odot$ indicates the elementwise multiplication (Hadamard product). Unless the network predicts a zero forget gate, LSTM doesn't guarantee the prevention of vanishing or exploding gradients. However, in practice, LSTMs are easier to train than Elman networks. To adapt LSTMs to unstructured point clouds, the approach involves assigning individual hidden and cell states to each point, creating point-independent states. In subsequent frames, for each point within the selected neighborhood, new hidden and cell states are calculated. The updated states are then aggregated, and the maximum values are adopted as the new cell and hidden states for the designated point. Recognizing the potential computational cost of this method, an alternative technique, termed point-shared states, is introduced. In this variant, cell and hidden states are shared among points, reducing computational overhead. Using the MSR dataset, which comprises point clouds capturing human activities, the proposed method demonstrates compelling results. Specifically, the approach achieves the highest accuracy, reaching approximately 92.29% for 3D pose estimation when the module is employed in the later steps of the network. This innovation not only addresses the alignment challenges in unstructured point clouds but also showcases the versatility of LSTM networks in enhancing accuracy in capturing human poses across dynamic actions[15].

Chapter 3

Proposed Solution

In the following chapter we introduce a potential solution for 3D pose estimation. Due to initially limited knowledge about camera parameters we first tried regressing poses since this approach only needs input data and ground truth skeletons. Later we introduce promising approach based on joint heat maps predictions and joints depth prediction, which can be performed only when camera parameters are known.

3.1 Dataset

The AMASS Dataset unifies multiple motion capture datasets. [14] Each captured skeleton in this dataset shares the same number of joints, eliminating the need for further skeleton preprocessing. Instead of standard RGB images, AMASS offers a pipeline for generating synthetic parameterized meshes of human bodies. These motion capture data are accompanied by SMPL parameters, ensuring consistency across the generated meshes. One potential drawback of this dataset is that the generated meshes only vary in body shape and gender. The appearances of the generated meshes remain uniform, lacking diversity in clothing, skin color, or hairstyle. By using the pinhole camera model, we can project points on the mesh surface into a 2D image grid.

3.2 3D Pose Regression

Due to limited knowledge about the dataset, the most straightforward approach for 3D pose estimation was to directly regress the joint positions. However, this approach is generally not ideal. Many state-of-the-art methods employ the so-called 2D lifting approach, which involves predicting 2D heatmaps and then projecting these 2D joints into 3D space. Since the camera parameters were unknown, we attempted to solve this task through regression. We drew inspiration from [13], which proposed a network that tackled the joint task of joint regression and detection. However, as we were unable

to preprocess the data in the same manner as proposed in the paper, we solely performed the regression task. Unfortunately, this approach bore no fruit. This could be attributed to the simpler architecture employed. The network predicted default poses that did not visually resemble the ground truth skeletons. We then attempted to utilize a U-Net architecture, which addressed the joint task of segmenting the human body and regressing joint positions. However, the network only succeeded in segmentation, failing to solve the joint regression task. Despite experimenting with more complex architectures and different preprocessing steps, we ultimately had to resort to generating a new dataset with known camera parameters.

Given that many pose estimation approaches leverage ResNets as their backbone, we opted to assess its efficacy in directly regressing joint positions. Since we lacked knowledge of camera intrinsics and extrinsics, this approach seemed to be our only viable option. Training deep neural networks can be challenging. Ideally, if a network can successfully perform a specific task, adding extra layers should either enhance accuracy or maintain it. However, in practice, deep neural networks struggle to learn identity mapping and often perform worse with deeper architectures, unless additional techniques are applied. The concept of residual learning, introduced by He et al. [12] addressed this issue by reformulating the problem, instead of learning identity the networks learns to predict zeros, the feed-forward layers were reformulated followingly:

$$F(x) + x$$

Where x are the features predicted by previous layer and $F(x)$ are new features generated by feed forward neural network. This formulation facilitated the training of deeper networks by making gradient flow smoother and mitigating the problem of vanishing gradients. The paper proposed several ResNet models for object classification, ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152, each denoting the number of layers in the model. Since its introduction, Deep Residual Learning for Image Recognition has become one of the most cited papers and continues to serve as a backbone for various tasks up to date. ResNets equipped with transpose convolutions have proven effective in tasks like segmentation and heat-map prediction. Given the widespread use of ResNet in pose estimation solutions, we decided to tackle the joint regression task by adapting the classification network. If the baseline model can accurately estimate poses, additional modules could further enhance accuracy by leveraging the sequential nature of images, predicted poses, or multiple views to refine joint position predictions in 3D space.

Since the performance of ResNet did not meet our expectations, we tried a different network namely GoogLeNet. The main building block of GoogLeNet are Inception modules. Inception modules are capable of capturing features at different scales while also addressing the problem of large number of parameters. Inputs are downsampled

by 1 by 1 convolution to reduce the number of features which are processed with kernels of different size. To improve flow of gradient GoogLeNet uses auxiliary classifiers at different depths which are used in loss computation and backpropagation. Inception blocks were also used by [8] to regress human body measurements. This was the main motivation of employing GoogLeNet for pose regression.

To assess the efficacy of structured data in pose regression, we conducted a comparison between ResNet, Inception-based architectures and PointNet, a method designed for processing of unstructured point clouds. Given that our dataset comprises structured points, converting them to unstructured point clouds for PointNet was straightforward. PointNet represented a breakthrough, as it could directly process unstructured point clouds, unlike the methods prevalent at the time. By avoiding voxelization or the use of image collections, PointNet enabled classification and segmentation without computationally expensive preprocessing or 3D convolutions.

The authors of PointNet addressed three key challenges inherent to the data format. First, point sets are ordered collections, necessitating network invariance to all possible permutations of points. Second, points in close geometric proximity form meaningful subsets, requiring the network to capture local structures and their interactions. Third, while point sets may undergo transformations, they still depict the same object or scene, demanding network invariance to rotations and translations of point sets. The proposed solution effectively tackles these challenges. The first challenge is mitigated by employing a shared multilayer perceptron (MLP) that transforms each input point into a distinct feature vector. This shared MLP essentially acts as a one-dimensional convolution applied along the length of the sequence. After computing the feature vectors, they are aggregated via max pooling into a single feature vector, capturing the global context of the point cloud. Each feature i is the global feature corresponding to the maximum feature among all feature vectors at position i. To encompass both global and local knowledge, the global feature is duplicated and concatenated to each point's feature vector. Addressing the last challenge, smaller PointNet-like networks called T-Nets predict transformation matrices for a given input, supposedly transforming the input features into a canonical space. For point cloud classification, the global context vector is utilized to predict the final label. Conversely, when segmenting unstructured point clouds, the global context is concatenated to each feature vector, and these are classified to generate per-point labels.

Building upon our prior work [7], where we demonstrated the feasibility of segmenting human bodies, we attempted to adapt the classification network to directly regress 3D poses. Instead of predicting C classes the network predicts $J \times 3$ real values representing the predicted position of joint.

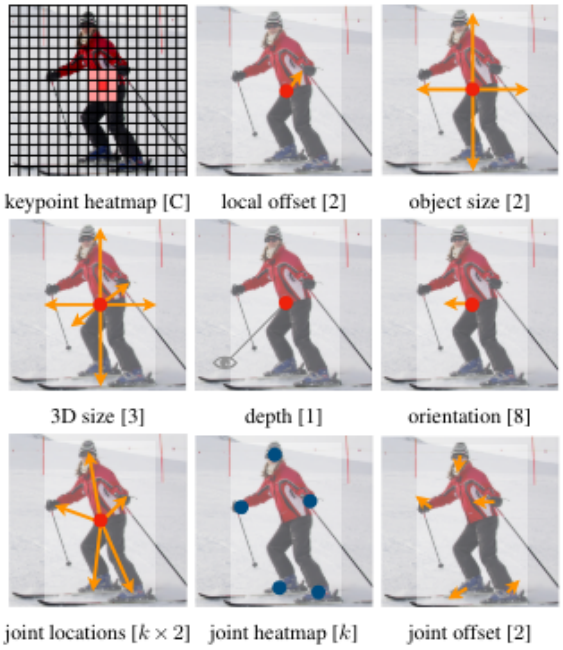


Figure 3.1: Each row represents different output for different task. Top for object detection, middle for 3D object detection, last for 2D pose estimation. Image taken from [22]

3.3 2D to 3D Lifting Approach

At the time of the publication of the Object as Points paper, many object detectors identified objects as axis-aligned bounding boxes in images. The prevailing approach among the best performing object detectors involved two main steps: proposing regions potentially containing objects and then classifying images cropped around these proposed regions as either background or containing the detected object. However, these methods were computationally costly and inefficient.

CenterNet addressed this issue by detecting objects as center points of bounding boxes and regressing their properties. It boasted the best accuracy-to-speed trade-off compared to FasterRCNN, RetinaNet, and YOLOv3. CenterNet was employed to tackle three types of tasks: object detection with 2D axis-aligned bounding boxes, object detection with 3D oriented bounding boxes, and 2D bottom-up human body pose estimation.

Given our focus on single-person 3D pose estimation, which is ensured by the input data, we combined the first two tasks. Each joint of the skeleton was considered a separate object, with a separate heatmap predicted for each joint. Subsequently, we regressed local offsets using two separate offset maps. Finally, for each detected point, we regressed its depth, mirroring the approach used in 3D object detection.

The primary rationale behind performing 2D joint detection lies in the fact that

knowing the 2D position and depth of a detected joint allows us to reproject the point back into 3D space using the pinhole camera model. Recall the pinhole camera model:

$$w \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = [KR|t]X \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{pmatrix} \begin{pmatrix} X_x \\ X_y \\ X_z \\ 1 \end{pmatrix}$$

This can be rewritten as:

$$w \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K(RX + t)$$

$$K^{-1} \begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = K^{-1}K(RX + t) = RX + t$$

$$K^{-1} \begin{pmatrix} wu \\ wv \\ w \end{pmatrix} - t = RX$$

$$R^{-1} \left(K^{-1} \begin{pmatrix} wu \\ wv \\ w \end{pmatrix} - t \right) = R^{-1}RX = X$$

If we know the camera intrinsic and extrinsic parameters, we can compute their inverses (the matrices are regular).

To safe parameters in transpose convolutions the predicted heat-maps have smaller spatial dimensions than the input image. If the predicted heat-map is R times smaller, than the predicted 2D position can be computed as:

$$\tilde{p} = \left\lfloor \frac{p}{R} \right\rfloor$$

Therefore:

$$R(\tilde{p} + \Delta p) = p$$

Where $\tilde{p}$ is the predicted position in heat-map, R is the factor Δp is the offset and p is the position in original image. These offsets are regressed at the position of predicted joints. Similarly, the depth of point is regressed at the position of predicted heat-map. Each heat-map is predicted by a different head. The position with the highest confidence in each joint heat-map is used to retrieve the offset and depth in both the offset map and depth map.

Chapter 4

Implementation

In this chapter we introduce technologies, which have been used for implementing solutions. Next we describe how was both the unstructured point clouds and the ground truth data prepared. Lastly we describe our modifications in existing pose estimation methods.

4.1 Technologies Used

Due to its user-friendly nature, Python has emerged as one of the most widely used programming languages. However, this ease of use comes with its own set of drawbacks. Python, being an interpreted language, tends to be slower compared to compiled languages like C. Additionally, the presence of the Global Interpreter Lock (GIL) restricts true parallel programming in native Python. Nevertheless, Python can be extended with C to mitigate these limitations, making it particularly popular in data science circles due to its extensive libraries. Hence, we opted to use Python for this thesis.

Given Python's inherent sluggishness in handling large-scale data manipulation, specialised libraries have been developed to address this issue. NumPy, for instance, is a library specifically designed for fast array operations. It achieves significant speed gains by leveraging low-level data types and executing operations in highly optimized, low-level code. Apart from array operations, NumPy also offers numerous numerical methods, which we utilised extensively for data preprocessing. However, it's worth noting that NumPy lacks the ability to harness the computational power of GPUs, making it unsuitable for deep learning tasks.

For deep learning applications in Python, PyTorch stands out as one of the most popular frameworks. Central to PyTorch are tensors, which not only store data but also record the operations performed on them—crucial for automatic gradient computation. Moreover, PyTorches manipulation with tensors can be accelerated by GPUs through the CUDA API. As a result, every deep learning model in this thesis was implemented

using the PyTorch framework.

The training and testing of deep learning models were conducted using a combination of our personal resources and the computational server Neptune at the Faculty of Mathematics, Physics, and Informatics of Comenius University.

Personal computer:

- **GPU:** NVIDIA GeForce RTX 3060 Ti; 8 GB
- **CPU:** 13th Gen Intel(R) Core(TM) i7-13700KF 3.40 GHz (16 cores, 24 threads)
- **RAM:** 32 GB
- **SSD:** 1 TB

Neptune server:

- **GPU:** NVIDIA GeForce RTX 3060; 12 GB
- **CPU:** AMD Ryzen 9 5900X (12 cores, 24 threads)
- **RAM:** 64 GB
- **SSD:** 1 TB

In our work we used mainly CMU, EKUT, Eyes Japan and ACCAD skeletons from AMASS dataset.[14] 3D synthetic human meshes for task of pose estimation were generated using SMPL model. Both female and male frames were used for deep learning. For each mesh structured point cloud was generated simulating the point clouds generated using structured light. There are four images per pose which were generated from 4 different views. ACCAD, SMU, EKUT and eyes japan consisted of 252, 940, 349, 26 sequences respectively. Which totaled in 6541, 40611, 6317, 3120 frames per view. The dataset was split randomly into 3:1:1 ration for training, validation and testing. It should be noted that in case the point was not detected for some pixel a zero vector was used to represent missing point. Before preprocessing the points are in global coordinate system where the coordinate system represents real-life coordinate system measured in mms.

4.2 Data Generation

Structured point clouds for pose estimation were generated from meshes generated by the AMASS pipeline, which aggregates motion capture data from a multitude of datasets, including: ACCAD, BMLhandball, BMLmovi, BMLrub, CMU, CNRS, DFaust, DanceDB, EKUT, EyesJapanDataset, GRAB, HDM05, HUMAN4D, HumanEva,

KIT, MoSh, PosePrior, SFU, SOMA, SSM, TCDHands, TotalCapture, Transitions, WEIZMANN. For each dataset AMASS provides unified skeletons, with consistent joint counts, and SMPL model parameters, influencing body shape and soft tissue dynamics. The pipeline employed by AMASS is called MoSH++, which fits SMPL+H body models.

Due to constraints in time and memory resources, we focused on motion capture data from only a subset of datasets, namely CMU, ACCAD, EKUT, and EyesJapanDataset. Furthermore, rather than generating meshes for every captured skeleton, we selected only two skeletons per second for processing. In addition to meshes the pipeline also generates new skeletons corresponding to parameterised body shape.

Using these meshes the structured point clouds were generated via ScanShoot program. This program was provided to me by the company Skeletex. The program simulates the behaviour of 3D scanner but instead of using real data in uses meshes in obj format. Using predefined extrinsic and intrinsic parameters of camera the point on top of the mesh surface were projected to grid. This behaviour can be modelled by simple pinhole camera model.

To generate structured point clouds from these meshes, we utilized the ScanShoot program provided by Skeletex. This program mimics the functionality of a 3D scanner, using meshes in obj format instead of real data. By applying predefined extrinsic and intrinsic camera parameters, the program projected points on top of the mesh surface were projected to grid. This behaviour can be modelled by simple pinhole camera model.

The camera intrinsics were constant and set as: $f_x = f_y = 400, c_x = c_y = 512/2, s = 0$. Each camera was manually positioned to focus on point $(0, 0, 900)$, with each camera having one of the following positions: $(0, 2300, 2000)$, $(-2300, 0, 2000)$, $(0, -2300, 2000)$, $(2300, 0, 2000)$. For each mesh in every sequence, RGB images and structured point clouds were generated separately. RGB images were stored in PNG format, while structured point clouds were saved in a custom binary format. The binary files began with four bytes indicating the number of scans s are stored in binary file (four, one for each view), followed by eight bytes representing the width w and height h of the grid, respectively. Subsequently, $s \times h \times w$ triples were listed. Each triple is represented by twelve bytes with every four bytes representing the x, y and z position respectively. If i is the index, starting from zero, the image coordinates u, v were computed as follows:

$$(j, u, v) = \left(\left\lfloor \frac{i}{h \times w} \right\rfloor, \left\lfloor \frac{i}{h} \right\rfloor, i \,(mod\, w) \right)$$

The $\lfloor x \rfloor$ operation represents the floor function, rounding down to the nearest integer. Since not every image coordinate (u, v) necessarily corresponds to a registered point,

a placeholder value is required. In this case, a zero vector serves as the placeholder for unregistered points. However, due to the presence of many unregistered points in unstructured point clouds, the data format becomes memory inefficient, potentially occupying hundreds of gigabytes of memory. Additionally, reading binary files byte by byte can be slow. To address these inefficiencies, the point clouds were converted into NumPy arrays, a format easily interpretable by many deep learning frameworks in Python. These arrays were then stored as compressed NumPy archives, along with images and all other data necessary for experiments. For pose estimation purposes, all joint positions were stored however for the experiments the joints corresponding to finger position of hands were omitted, therefore only twenty-two key points were used for experiments. Given that most GPUs support only thirty-two bit floats, the doubles were converted to floats to ensure compatibility. Each point cloud and image in the sequence were stored individually for each camera. Additionally, the generated skeletons were converted to NumPy archives and are shared among views within the sequence.

Deep neural networks generally converge faster when the input data are normalized. Therefore, different values used in feature scaling, along with camera parameters, number of frames, and sequences, were stored in a metadata JSON file. Given that many existing solutions are based on 2D joint prediction via heat-map prediction, ground truth heat-maps needed to be generated.

Since we have knowledge of the camera parameters, we can project the 3D joints into 2D coordinates on images. To reduce the number of parameters in up-convolutions, the predicted heatmaps are usually of lower resolution. Thus, the projected (u, v) coordinates were scaled accordingly. For example, if the generated heat-map is four times smaller than the input image, the projected joint positions were divided by four.

For each joint in each sequence and view, a 2D heat-map was generated as follows: using a 1D Gaussian kernel of size eleven and sigma two, we generated a 2D kernel as the outer product of two 1D kernels. This resulted in a 2D kernel of size eleven by eleven pixels, positioned with the maximum value centered at coordinate (u, v) where the joint's 3D position was projected. All other non-overlapping values were set to zero. These heat-maps essentially supervise the network to be confident in predictions centered around the joints' projected 2D positions. The higher the value in the heat-map, the more likely the position corresponds to the ground truth 2D position.

As the [26] approach requires the pixel with the highest confidence to be 1.0, the confidence scores were divided by the largest value in the 2D Gaussian kernel. Along with the heat maps, we also generated the joint depth maps and offset maps for the Object as Points approach.

For performance comparison between images and structured point clouds, RGB images were captured as well. The dimensions of both the images and structured point

clouds were $512 \times 212 \times 3$ pixels. The joint heat maps, offset maps, and joint depth maps were of shapes $128 \times 128 \times 22$, $128 \times 128 \times 2$, and 128×128, respectively.

As we sought to evaluate the performance of certain deep learning approaches designed for unstructured point clouds, we also generated such point clouds. Converting structured point clouds into unstructured ones is straightforward; we simply flattened the points captured in the grid into a 1D array of points, omitting the zero points.

Many deep learning frameworks utilize multidimensional arrays for mini-batch training. However, using batches of any size larger than one with unstructured point clouds poses challenges because each sample in the batch occupies blocks of memory of different sizes. As a result, many deep learning frameworks cannot operate on such batches, necessitating some form of sampling.

One form of sampling which can be easily implemented is to select fixed number of random points for each data sample. However, random sampling does not guarantee that the selected points will be evenly distributed in space. Another problem is that some regions, which are closer to camera, are over-represented by points, there is higher density of points in some regions.

A more reliable approach to sampling is known as farthest point sampling [9]. This algorithm is a relatively efficient greedy method that involves the following steps:

1. Initialize an empty collection of selected points and a collection of remaining points.

2. Add a random point from the list of remaining points to the selected list and remove it from the pool of potential candidates.

3. For each remaining point, determine the closest point among those already selected and store their distance.

4. Based on these distances, select a new point that is farthest away, add it to the selected list, and remove it from the candidate pool.

5. Repeat the last two steps until the desired number of selected points is reached.

Concatenated unstructured point clouds from multiple views were sampled via this method to produce point clouds consisting of evenly distributed points. We undersampled the point clouds to obtain 2048 points for each frame in each sequence.

In one experiment, we attempted segmentation. To generate ground truth labels each point was classified by the nearest skeleton joint. Points that were zero vectors or had a y-coordinate lower than a certain threshold were classified as background. The thresholding was applied to filter out points representing the floor.

The dataset was divided into three categories: training, validation, and testing. Sequences whose sequence ID was divisible by five were selected for validation, those

with a remainder of one were chosen for testing, and the rest were used for training. This partitioning allowed us to split the dataset into a 1:1:3 ratio, meaning that approximately 60% of frames were used for training, while 20% each were allocated for validation and testing. Since each sequence consists of a varying number of frames, these distributions are approximate.

4.3 Regression of 3D Pose

In the subsequent section, we outline our adjustments to existing networks designed for classification tasks. In our deep learning experiments, we utilized both official and unofficial implementations, each of which required minor modifications. Our training and testing procedures were conducted using custom scripts developed by us.

4.3.1 ResNet

In our experiments, we utilized two different implementations of ResNets.

Firstly, we employed an existing third-party reimplementation in PyTorch, which we slightly modified. Instead of using ReLU, we opted for LeakyReLU as the activation function between layers. Additionally, we replaced the first max-pooling layer in the neck with average pooling to better preserve joint locations. Instead of employing global average pooling, we flattened the predicted feature map of the last convolutional block and applied two fully connected layers with batch normalization to predict the final joint positions. For training, we selected the ResNet-18 model, which was trained on structured point clouds in the world coordinate system. We utilized the AMSGrad optimizer with a learning rate of 5×10^4, $\beta_1 = 0.9$, and $\beta_2 = 0.999$, training the model for 100 epochs using cosine annealing rate. The datasets used for training, validation, and testing included ACCAD, CMU, EKUT, and Eyes Japan sequences, totaling 1567 sequences comprising 56589 frames per view. Mean squared error was employed as the loss function, with a batch size of 32. The network was trained to predict joint positions relative to their parent joints, with the root joint situated at the origin of the world coordinate system. The relative joint position R_i was computed as

$$R_i = J_i - J_{P(i)}$$

where J_i represents the joint position and $J_{P(i)}$ denotes the position of the parent joint. This method, as adopted in [23], allows the network to learn to predict distances between joints and sharing some information among joints. Both the structured point clouds and skeletons were zero-centered and scaled using min-max normalization. The metric used for evaluation was MPJPE.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Figure 4.1: Overview of different ResNet architectures. Image taken from [12].

In the second experiment, we employed the official PyTorch implementation, specifically ResNet-34 pretrained on the ImageNet dataset. This time, we used newly generated point clouds with known camera parameters. By utilizing the extrinsic camera matrix, we transformed the structured point clouds and skeletons into the camera coordinate system. We only utilized the CMU dataset with newly captured mocap data, comprising 2079 sequences totaling 52938 frames captured from four different views. The batch size for this experiment was 16. The network was trained to minimize the mean squared error between predicted skeletons and scaled ground truth skeletons in camera space, using the Adam optimizer with a learning rate of 5^{10-4} and predefined betas. The network underwent fine-tuning for 25 epochs, and the learning rate was decreased using the cosine annealing learning rate. To compute the MPJPE metric, the skeletons were reprojected back into the world coordinate space.

4.3.2 GoogleNet

In the next section, we discuss our adaptations to a network originally designed for Anthropometric Body Measurements Estimation, which we re-implemented in PyTorch. Instead of regressing body measurements, this network was repurposed to regress relative joint positions, as described in the previous section. The input for the joint regression task consisted of structured point clouds in the world coordinate system. The feature maps from the last layer were flattened and regressed using a multi-layer perceptron (MLP) with one hidden layer. Batch normalisation was not applied, and the rectified linear unit (ReLU) function served as the non-linear activation function. Both the input and skeleton data were zero-centered and scaled. The network was trained using mean squared error (MSE) loss minimization with the AMSGrad optimizer, utilizing a learning rate of 5×10^{-4} with betas consistent with those used in previous experiments. Training proceeded for 100 epochs with cosine annealing learning rate scheduling. No auxiliary classifiers were employed in this experiment.

In the second experiment, we employed an official pretrained PyTorch reimplementation designed for classification tasks, which we modified for joint regression. The predicted feature maps from the last pooling layer were flattened and regressed using two fully connected layers, separated by batch normalization and ReLU activation functions. The input for the network consisted of structured camera point clouds in the camera coordinate system. Training was conducted for 25 epochs using the Adam optimizer with cosine annealing learning rate scheduling, following a similar approach to the ResNet experiment. Auxiliary classifiers were not utilized in this instance. The dataset underwent zero-centering and scaling using min-max normalization methods.

4.3.3 PointNet

For our experiment, we utilized a previously developed implementation originally intended for segmentation tasks, as detailed in [7]. We adapted this implementation to perform classification, and further modified it to serve as a regression model. Notably, we removed dropout layers from the architecture, as they have been shown to hinder regression capabilities. The network was trained to minimize the combined loss function:

$$L = L_{pose} + wL_{reg}$$

Where L_{pose} represents the L_2 norm between predicted joint positions $\hat{p_j^i}$ and ground truth positions p_j^i, defined as:

$$L_{pose} = \frac{1}{NJ}\sum_{i=1}^{N}\sum_{j=1}^{J}\|p_j^i - \hat{p_j^i}\|_2^2$$

Additionally, L_{reg} serves as a regularization term, ensuring that the predicted feature alignment matrix A generated by second T-Net is close to orthogonal matrix:

$$L_{reg} = \|I - AA^T\|_F^2$$

Here, I represents the identity matrix, $\|A\|_F^2$ is the square of Frobenius norm of matrix X, w was set to 10^{-3}. The network underwent training for one hundred epochs using the Adam optimizer, with a learning rate of 10^{-3} and $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Following the original authors' recommendation, the learning rate was halved every twenty epochs. The mean per joint position error (MPJPE) served as the evaluation metric.

To generate unstructured point clouds from structured ones, we employed farthest point sampling. The unstructured point clouds were derived from four different views, utilizing a total of 940 sequences provided by my thesis supervisor. We experimented with various augmentation techniques, including randomly sampling a fixed number of points, adding random noise from a normal distribution with a standard deviation of

Figure 3: GoogLeNet network with all the bells and whistles

Figure 4.2: Architecture of GoogLeNet used. Image from [21].

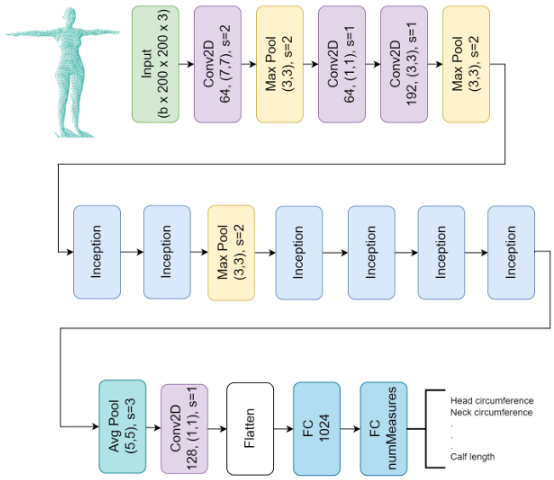


Figure 4.3: Inception based model originally used for body measures regression. Image from [8].

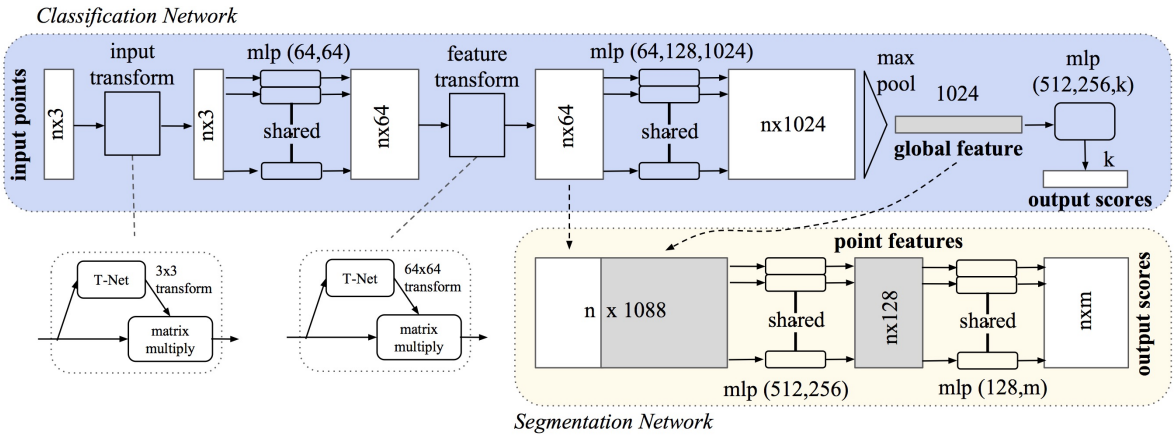


Figure 4.4: The proposed PointNet architecture for classification and segmentation. Image taken from [19].

five, and applying random rotations around the x, y, and z axes. For vertical z-axis rotations, degrees were randomly chosen from the interval -180 to 180, while for the other axes, angles were selected from -30 to 30 degrees. For training, validation, and testing purposes, we utilized 40,611 frames from the 940 sequences generated from the CMU dataset.

4.4 Heat-map Centred Approach

In the following section we provide information about our modification to DLA-34 network originally used for object detection. After generating dataset with known camera parameters, we were able to generate ground truth data required by Object as

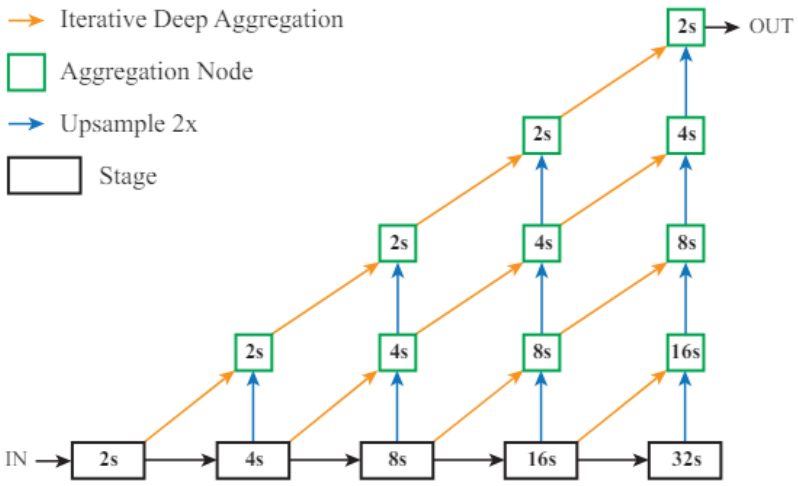


Figure 4.5: Scheme of deep layer aggregation architecture proposed in [25] used in objects as points approach [26]. Image from [25].

Points [26] approach.

4.4.1 CenterNet

In our experiments, we utilized the officially implemented CenterNet by the authors of Objects as Points [26] paper. However, we encountered compatibility issues due to the age of the implementation and its exclusive support for Linux operating systems. Consequently, modifications were necessary to ensure compatibility with Windows systems. Specifically, deformable convolutions, which are incompatible with Windows, had to be replaced with Windows-friendly alternatives. Additionally, since we use custom dataset format the implemented networks had to be integrated into our own training scripts.

Furthermore, given that our proposed solution is combination of two different tasks, modifications were required in the network architecture. We experimented with both structured point clouds and RGB images. The encoder network generated low-dimensional feature maps, which were subsequently upscaled using transpose convolutions. The original convolutions performed during upscaling were replaced by 3×3 deformable convolutions. These upscaled feature maps served as the input for three distinct heads, each tailored for different tasks.

Each network head comprised two convolutional layers separated by ReLU activation functions, ending with sigmoid activation. Specifically, one head predicted twenty-two 2D key point heatmaps, while another predicted two feature maps, one for x-offset and one for y-offset. The final head predicted a single depth map. However, instead of using a sigmoid activation function, it employed an inverse sigmoidal transformation,

as proposed in the original paper:

$$\frac{1}{\sigma(\hat{d}_k)} - 1$$

Here, $\hat{d}_k$ represents the value in the feature map at joint k's position. The objective of the network is to minimize the loss L, which is computed as follows:

$$L = L_k + L_{off} + L_{dep}$$

$$L_k = -\frac{1}{N} \sum_{xyc} \begin{cases} (1 - \hat{Y}_{xyc})^{\alpha} log(\hat{Y}_{xyc}) \; if \; Y_{xyc} = 1 \\ (1 - Y_{xyc})^{\beta} (\hat{Y}_{xyc})^{\alpha} log(1 - \hat{Y}_{xyc}) \; else \end{cases}$$

Here L_k represents the focal loss, $\hat{Y}_{xyc}$denotes the predicted probability that joint c is located at position (x, y), while Y_{xyc} signifies the ground truth probability of joint c being located at position (x, y). The hyperparameters α and β were experimentally chosen as $\alpha = 4$ and $beta = 2$.

$$L_{off} = \frac{1}{N} \sum_{p} \left| O_{\tilde{p}} - \left(\frac{p}{R} - \tilde{p} \right) \right|$$

L_{off} denotes the L_1 norm between the predicted offset and ground truth offset. $O_{\tilde{p}}$ represents the predicted offset, p is the real 2D position of the joint, R is the stride (in our case $R = 4$) and $\tilde{p}$ is the discreet joint position in downsampled ground truth heat map.

$$L_{dep} = \frac{1}{N} \sum_{k=1}^{N} \left| \frac{1}{\sigma(\hat{d}_k)} - 1 - d_k \right|$$

L_{dep} signifies the L_1 loss between the predicted depth of joint k and its ground truth depth d_k, both measured in meters. The values $\hat{d}_k$ and $O_{\tilde{p}}$ are selected from corresponding maps using the maximum position at each joint heat map. However, these positions are most likely incorrect at the start of training and the other maps cannot be properly trained. Therefore, the ground truth positions are used instead. During inference, the positions are selected normally.

For 3D human pose estimation, we opted to utilize the DLA-34 network due to its robust performance in object detection tasks and efficient inference speed. Our training process involved training the network on unstructured point clouds in the camera space coordinate system, alongside RGB images for comparative analysis. Both data formats underwent preprocessing steps including zero-centering and scaling via min-max normalization. Additionally, we calculated the mean as the global average among each frame for both data formats, along with the maximum and minimum RGB values and points. No further data augmentation techniques were applied.

The networks were trained for twenty-four epochs using the Adam optimizer with a learning rate of 10^{-3} and $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate was halved every

3 epochs to facilitate stable convergence. A batch size of sixteen was employed during training, and both networks were trained on a single NVidia RTX 3060 Ti GPU.

To evaluate the performance of the trained networks, we utilized the Mean Per Joint Position Error (MPJPE) metric for 3D skeletons. These skeletons were generated using known camera parameters. The world 3D coordinates of joint j were computed as follows:

$$\begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix} = R^{-1}\left(K^{-1}\hat{d}_j \begin{pmatrix} s(\hat{u}_j + \Delta\hat{u}_j) \\ s(\hat{v}_j + \Delta\hat{v}_j) \\ 1 \end{pmatrix} - t \right)$$

Here, $(\hat{u}_j, \hat{v}_j)$ represents the 2D position in the predicted heat map j with the maximum value. The values $(\Delta\hat{u}_j, \Delta\hat{v}_j)$ were extracted from the predicted offset map at the position $(\hat{u}_j, \hat{v}_j)$, similarly $\hat{d}_j$ was obtained from the predicted depth map at the position $(\hat{u}_j, \hat{v}_j)$. The input data were 512×512 pixels, whereas the predicted heat maps were 128×128 pixels, this means the stride s was four.

It should be noted that in some cases it might happen that the predicted 2D positions of joints overlap. If this is the case the overlapping joints will share both the offsets and depth, which unfortunately causes that the reprojected joints share the same 3D position in space, which is not possible and therefore makes for incorrect joint estimation.

It's worth noting that in some cases, the predicted 2D positions of joints may overlap. In such instances, the overlapping joints will share both the offsets and depth, given that both offset maps and depth maps are shared among joints. Consequently, this can lead to certain reprojected points sharing the same 3D position in space. This scenario is not feasible and therefore results in inaccurate joint estimation.

Chapter 5

Results

In the upcoming chapter, we delve into the outcomes yielded by our trained models and conduct a thorough evaluation on the testing dataset. We also delve into the Metric of Mean Per Joint Position Error (MPJPE) and provide insights into its computation process.

5.1 Used Metrics

The Mean Per Joint Position Error, often abbreviated as MPJPE, stands as one of the primary evaluation metrics in 3D human body pose estimation. Typically measured in millimeters, it is computed as follows:

$$MPJPE = \frac{1}{NJ} \sum_{i=1}^{N} \sum_{j=1}^{J} \|p_j^i - \hat{p}_j^i\|_2$$

Which is the average Euclidean distance between predicted joint position p_j^i and the ground truth position $\hat{p}_j^i$. N denotes the number of poses evaluated, and J denotes the number of joints per pose. A lower MPJPE score indicates higher accuracy in joint estimation.

However, the MPJPE metric has its own set of limitations. For instance, if the predicted skeleton closely resembles the ground truth skeleton but is spatially displaced, the error may be large in proportion to the displacement. One strategy to address this challenge involves selecting a root joint and subtracting its coordinates from those of each joint. This effectively centers the skeleton with the root joint positioned at the origin of the coordinate system, thereby providing a more consistent reference point for comparison. Another limitation arises when the predicted skeleton matches the ground truth but is subject to rotation. In such cases, the error may significantly increase relative to the degree of rotation, impacting the overall evaluation of the pose estimation accuracy.

Table 5.1: Results of models trained with structured point clouds in world coordinate system. For ResNet-18 and Inception network structured point clouds were used, in case of PoinNet we randomly sampled points to produce unstructured point clouds.

Model	Average MPJPE [mm]
ResNet-18	**186.314**
Inception	193.73
PointNet	192.682

5.2 Experiments

As was outlined in previous chapters the dataset had to be generated multiple times. Originally we were provided with structured point clouds with ground truth skeletons. The problem of this dataset was that we did not know the camera parameters so we could only employ regression methods for pose estimation. Moreover the structured point clouds were captured from random camera positions, meaning the camera extrinsics were different for each frame. The first iteration of dataset consisted of Eyes Japan, EKUT, ACCAD and CMU sequences. This dataset was used for multiple experiments however we report the most successful ones in table 5.1 . Namely modified ResNet-18, inception network and PointNet.

Following adjustments to the program responsible for generating structured point clouds from meshes, we gained the capability to manually set camera parameters. This data was mostly used for PointNet regressions. We used only the CMU sequences. To generate unorganised point clouds we employed the farthest point sampling methods after joining multiple views. The meshes, which were provided to us had one drawback. During generation of meshes using SMPL model, the parameters were changing between frames, resulting in meshes changing body shape during captured sequence. The results are presented in Table 5.2. We assessed the model with the best validation metric, which was trained using unorganized point clouds augmented with random noise from a normal distribution. The average MPJPE of this model during testing was 169.715 mm.

As the body shapes consistently varied over time, even within a single person across sequence frames, we resorted to generating meshes using the pipeline provided by AMASS [14]. Parameters supplied by AMASS remained uniform for each individual throughout the sequence, ensuring consistency in body shapes over time. We used only the CMU sequences, since there was abundance of data. While the dynamic nature of body shapes could be perceived as a form of augmentation, maintaining consistency was crucial for potential utilization of temporal information to enhance

Table 5.2: Different combination of augmentations and the best validation metric. The results were gained by training PointNet on unstructured point clouds gained by FPS algorithm.

Augmentation	Best Validation MPJPE [mm]	Epoch
None	178.9	35
Random Sampling	164.8	66
Random Rotation	276	4
Normal Noise	**163**	**40**
Random Sampling + Rotation	235.9	7
Random Sampling + Noise	174.4	50
Random Rotation + Noise	307.4	5
Random Sampling + Rotation + Noise	255.1	7

Table 5.3: Models were training using structured point clouds in camera coordinate system. We regressed absolute joint positions, except for one case where we tried relative joint estimation as in [23]

Model	Average MPJPE [mm]
ResNet-34	**206.533**
GoogleNet (Relative Pose)	217.767
GoogleNet (Absolute Pose)	214.555

.

pose prediction accuracy. However, due to time constraints, we were unable to fully leverage any temporal information. In table 5.3 we report the metric on models trained with structured point clouds in camera coordinate system. In table 5.4 we report are best performing models, which utilised the camera parameters to reproject predicted 2D joint positions along with depths back to 3D. These models were trained with organised point clouds in camera coordinate system and images.

As can be observed from 5.2 and 5.3 while the mean squared error between regressed joints and ground truth joints had tendency to minimise, the validation loss was fluctuating. This might indicate that the networks, which perform joint regression fail to generalise. Moreover every single regression model predicted body shapes, which either did not resemble human pose, or were some kind of T-pose, centred at origin. 5.1 shows us the best results achieved by regression.

Since the regression based methods failed to accurately estimate the pose, we analyse the performance of DLA-34 architecture trained with structured point clouds in camera coordinate system. First we compute MPJPE per frame to order predictions

Table 5.4: Comparison of results gained by DLA-34 network. Employing the "Objects as Points" methodology [26], we trained our models using both images and structured point clouds in camera coordinates.

Model	Average MPJPE [mm]
DLA-34 (Point Clouds)	**44.939**
DLA-34 (Images)	62.394

Table 5.5: The following table summarises our results.

Model	Average MPJPE [mm]
GoogLeNet (Relative Pose, structured camera pt. clouds)	217.767
GoogLeNet (Absolute Pose, structured camera pt. clouds)	214.555
Inception Net (root-aligned Pose, structured pt. clouds)	214.555
ResNet-34 (Absolute Pose, structured camera pt. clouds)	206.533
PointNet (root-aligned Pose, Random pt. clouds)	192.682
ResNet-18 (root-aligned Pose, structured pt. clouds)	186.314
PointNet (Absolute Pose, FPS pt. clouds)	169.715
DLA-34 (Absolute Pose, images)	62.394
DLA-34 (Absolute Pose, structured camera pt. clouds)	**44.939**

Table 5.6: The average MPJPE achieved on CMU Panoptic Dataset of some deep learning models [1].

Model	Average MPJPE [mm]
ROMP (ResNet-50)	127.60
Learnable Triangulation of Human Pose	13.70
VTP	17.62
AdaFuse	13.55
TesseTrack Multi-view (5 views)	**7.3**

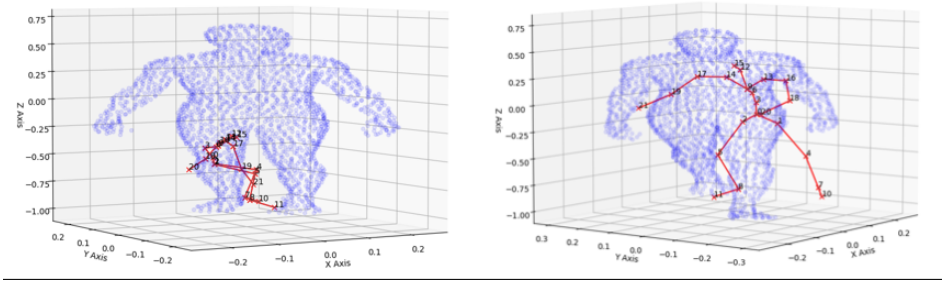


Figure 5.1: Inference of our best regression model, PointNet.

Figure 5.2: MSE while training Networks for joint regression.

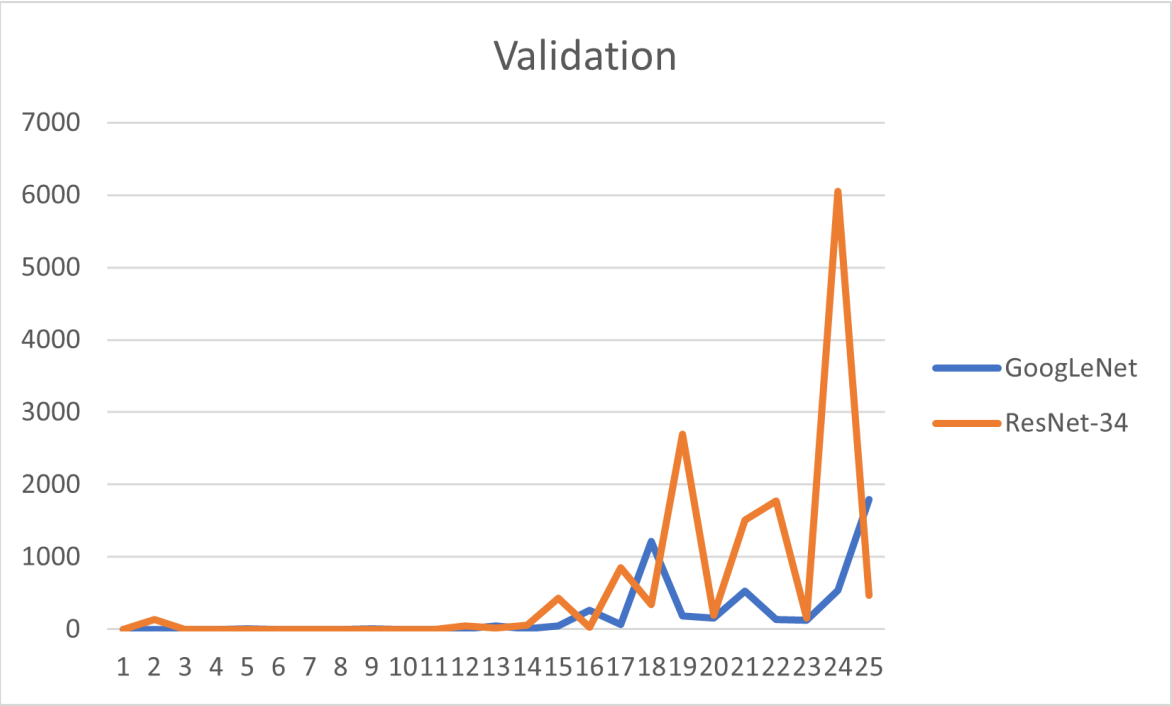


Figure 5.3: MSE during validation, this diverging behavior was quite common for regressions.

from best to worst. The best three, worst three and three predictions around median MPJPE are visualised in 5.4. In 5.5 and 5.6 we display the relative frequency of predictions with MPJPE lower or equal to the given threshold. Histogram 5.8 informs us about the prediction error and its standard deviation, the lower the deviation the more robust the predictions are.

Part of the CMU panoptic dataset comprises motion capture data depicting a break dancer's performance, featuring unique and unconventional movements like handstands. The pretrained model struggles significantly in predicting these poses, as evident in Figure 5.4. It appears that the model assumes joint positions akin to an upright posture, leading to inaccuracies in predicting the break dancer's movements.

Despite an average MPJPE of approximately 45 mm, analysis of Figure 5.5 reveals that around 54% of joints exhibit an MPJPE of 35 mm or less. Furthermore, approximately 90% of predicted joints have an MPJPE of 75 mm or lower, although some predictions show an MPJPE as high as 880 mm.

Figures 5.7 and 5.8 highlight that the most reliable predictions are for joints near the spine. This could be attributed to the majority of captured data being in upright positions, resulting in minimal variability in spine movement. Conversely, the least accurate predictions pertain to the positions of hands, followed by elbow joints, feet, heels, and knees.

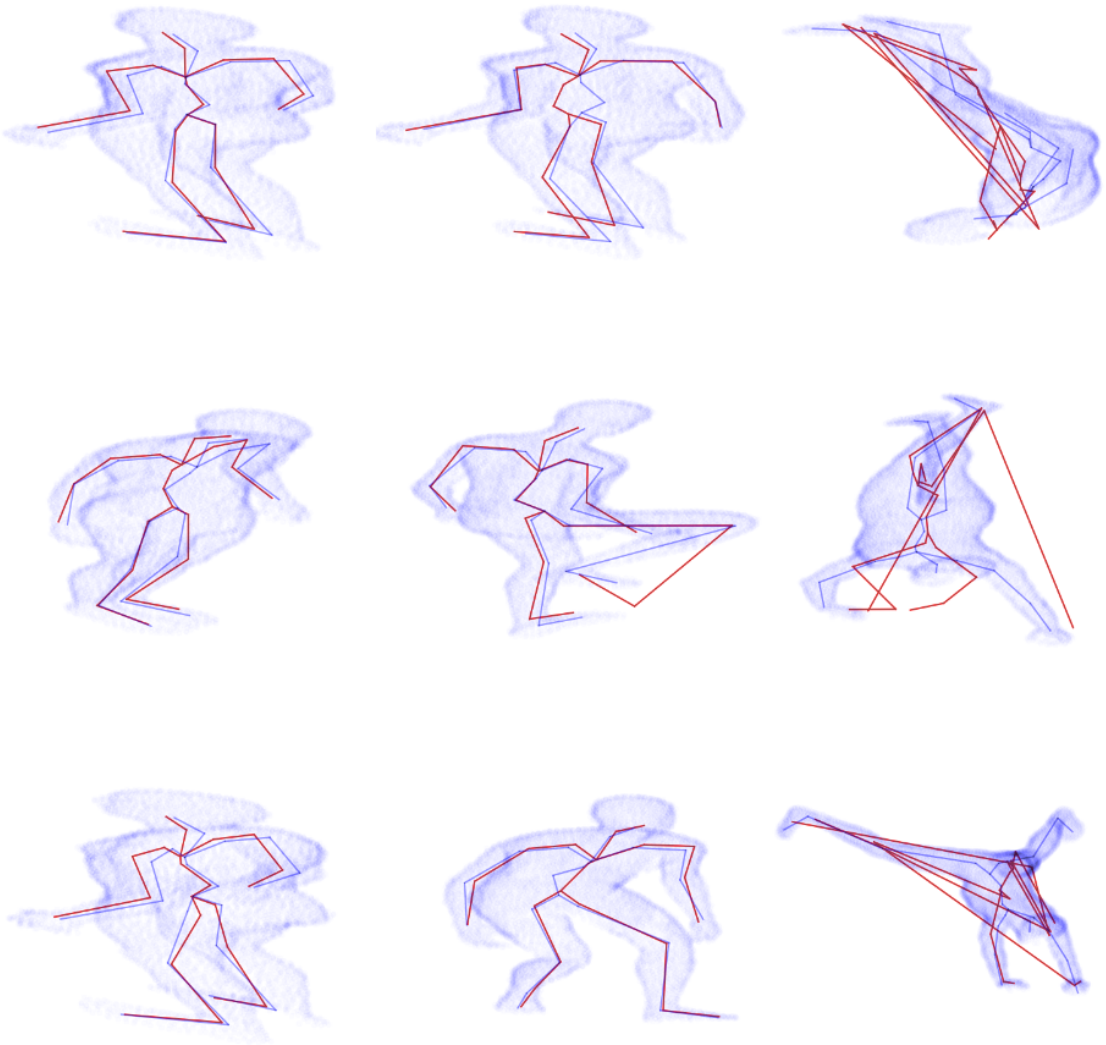


Figure 5.4: Examples of best and worst predictions of DLA-34 trained with organised point clouds. Each column corresponds to predictions with similar MPJPE, first column are the best predictions, middle column are average predictions and last column are the worst predictions. Blue skeletons are the ground truth poses, red represent the predictions on testing subset.

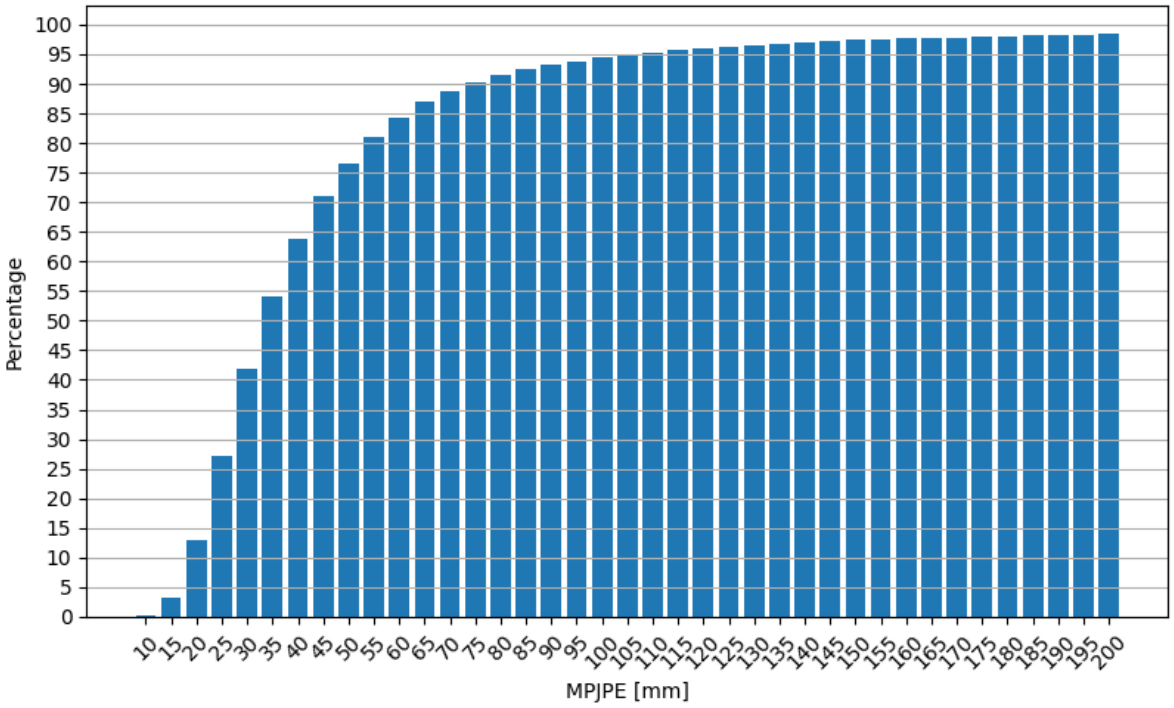


Figure 5.5: Each bar correspond to percentage of predictions with MPJPE lower or equal to value on the x-axis. Predictions were performed by DLA-34 on testing subset.

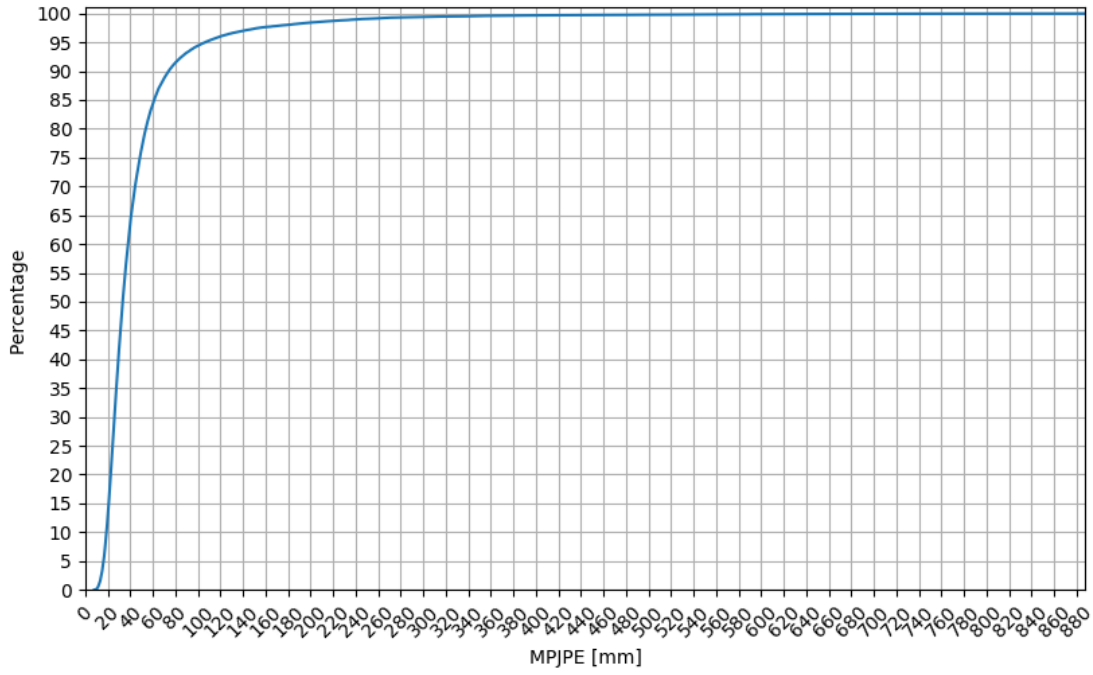


Figure 5.6: Similarly as in previous histogram, the graph depicts predictions with MPJPE lower or equal to the value on x-axis.

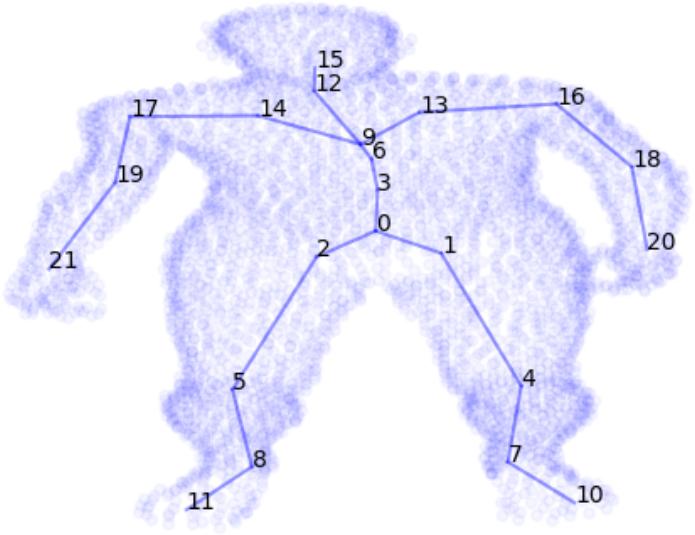


Figure 5.7: Point cloud representing body shape with annotated skeleton. Each joint is labeled with its identification number.

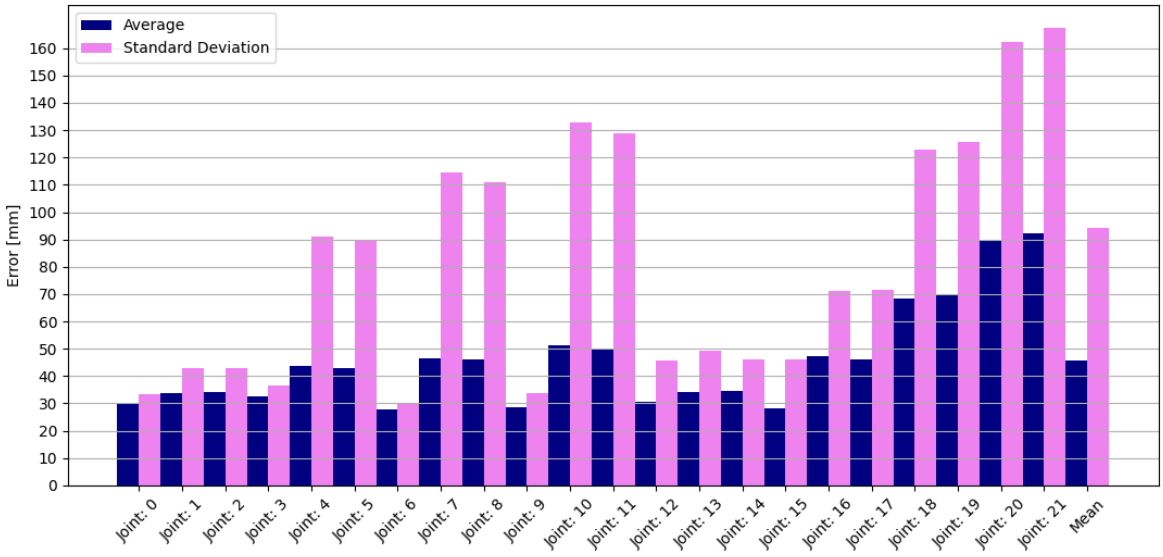


Figure 5.8: The following histogram depicts the average prediction error and its standard deviation for each joint separately. The last columns is the MPJPE and its deviation among testing examples. Error was computed as the Euclidean distance between ground truth position and predicted position of joint.

Conclusion

The primary objective of this thesis was to evaluate the advantages and limitations of structured point clouds in tasks related to human body analysis, particularly in 3D pose estimation.

Given the inherent characteristics of our dataset, our initial exploration was limited to regression tasks. Through experimentation with various networks for 3D pose regression, we reached a significant conclusion: naive regression of point clouds proved to be ineffective. Surprisingly, our findings revealed minimal disparity in results between organized and unorganized point clouds. We demonstrated that the incorporation of structural information for joint regression offered marginal benefits, as these approaches were consistently surpassed by methods tailored for unorganized point clouds. However, despite these efforts, the outcomes remained subpar. This could be attributed to the high variance in input data, which neither model adequately captured, or it may suggest that accurately estimating 22 joint positions is an inherently challenging task.

Furthermore, our study underscored the potential significance of 2D pose estimation approaches, even within the context of structured point clouds. We introduced a modified CenterNet-based architecture for 3D pose estimation, revealing that the integration of additional 3D information yielded improvements over RGB images in pose estimation accuracy.

Regrettably, due to time constraints, we were unable to fully explore the intricacies of our proposed dataset, particularly its sequential motion capture nature and the utilization of multiple views for a single ground truth skeleton. Although we attempted to implement the AdaFuse model, which utilizes predicted 2D skeletons from various views to triangulate the final 3D position, our training efforts were incomplete due to time limitations. To enhance the accuracy of predicted skeletons, future iterations could incorporate techniques such as temporal convolutions or modified transformers to ensure consistent predictions over time. Additionally, while our focus was primarily on single-person pose estimation, there is considerable potential in exploring the efficacy of 3D multi-person pose estimation from structured point clouds.

Another significant constraint we faced was hardware limitations. With the increasing complexity of deep learning models and the reliance on large-scale architectures trained across multiple high-end GPUs, our access to only one or two commercial

GPUs posed a significant obstacle. Overcoming this limitation would require access to more powerful computing resources to fully explore the potential of sophisticated models in our research endeavors. Despite these constraints, our study provides valuable insights into the challenges and opportunities inherent in leveraging structured point clouds for human body analysis tasks.

Bibliography

[1] 3d human pose estimation on panoptic. https://paperswithcode.com/sota/3d-human-pose-estimation-on-cmu-panoptic. Accessed: 2024-05-10.

[2] Convolution. https://iq.opengenus.org/content/images/2023/01/2023_01_20_0te_Kleki-min.png. Accessed: 2023-12-08.

[3] Deformable convolution image. https://miro.medium.com/v2/resize:fit:4800/format:webp/1*xwzmJH01WwcSTsZv3B7OfA.png. Accessed: 2024-05-06.

[4] Geometry of image formation. https://learnopencv.com/geometry-of-image-formation/. Accessed: 2024-05-10.

[5] Point clouds. https://manual.coppeliarobotics.com/en/pointClouds.htm.

[6] Transpose convolution image. https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11. Accessed: 2024-05-06.

[7] Deep learning-based human body segmentation on 3d body scans, 2022. Bachelor Thesis. Comenius University in Bratislava.

[8] Inception network for anthropometric body measurements estimation from structured point clouds, 2022.

[9] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Zeevi. The farthest point strategy for progressive image sampling. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 6:1305–15, 02 1997.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Chuchu Han, Xin Yu, Changxin Gao, Nong Sang, and Yi Yang. Single image based 3d human pose estimation via uncertainty learning. *Pattern Recognition*, 132:108934, 2022.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Sijin Li and Antoni Chan. 3d human pose estimation from monocular images with deep convolutional neural network. volume 9004, pages 332–347, 11 2014.

[14] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, October 2019.

[15] Yuecong Min, Yanxiao Zhang, Xiujuan Chai, and Xilin Chen. An efficient pointlstm for point clouds based gesture recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5760–5769, 2020.

[16] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math; (March 1, 1997), 1997. `https://www.cs.cmu.edu/~tom/files/MachineLearningTomMitchell.pdf`.

[17] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. Camera distance-aware top-down approach for 3d multi-person pose estimation from a single rgb image, 2019.

[18] Gyeongsik Moon and Kyoung Mu Lee. I2l-meshnet: Image-to-lixel prediction network for accurate 3d human pose and mesh estimation from a single rgb image, 2020.

[19] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[20] Xiaoye Qian, Youbao Tang, Ning Zhang, Mei Han, Jing Xiao, Ming-Chun Huang, and Ruei-Sung Lin. Hstformer: Hierarchical spatial-temporal transformers for 3d human pose estimation, 2023.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[23] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking, 2018.

[24] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. Vitpose: Simple vision transformer baselines for human pose estimation, 2022.

[25] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation, 2019.

[26] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *CoRR*, abs/1904.07850, 2019.