3D Monocular Pose Estimation from Synthetic Data

Jonathan Kane

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2022

Abstract

3D pose estimation has been a very popular part of computer vision for a long time now. The main challenge these days is to find data that can be used to train a 3D pose estimation algorithm since it requires 3 dimensional data. Our project addresses this problem by using a synthetic dataset based on the Sungaya stick insect. The dataset used is artificially generated to provide photo realistic samples. These samples are used to train our 3D pose estimation pipeline which consists of 2 deep neural networks. The first of these networks detects the 2D pose of a stick insect and the second network detects the 3D pose of the stick insect based of the 2D pose data from the previous network. The project also investigates how well these networks trained on synthetic data and carries out experiments on both models to determine how well the models are able to perform when faced with real data. Using the results provided by our investigations we determine that this pipeline is a good step into creating a 3D pose estimator for a stick insect which is able to detect 3D pose only trained on synthetic data. The code used will be available here.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jonathan Kane*)

Acknowledgements

I would like to firstly thank my mum for introducing me to Computer Science and always eager to help out in any way she can. I would also like to thank my supervisor Oisin for his help and guidance throughout the project and always teaching me something new after every meeting. Finally, I would like to thank David Labonte and Fabian Plum from Imperial College London for providing me with their custom Sungaya dataset from their scAnt project, this project would not have been possible without them.

Table of Contents

Chapter 1

Introduction

1.1 Motivations

Pose estimation has been a very popular field of computer vision over the past two decades. The main focus of this research has been on humans where large real life datasets have been created and many methods of prediction have been researched, ranging from Decision Trees to Deep Neural Networks (DNNs). These methods have for the most part been evaluated on the same or similar datasets the most notable of these being Human3.6m [1] and MPII [2] where the subject is a human being. Predicting 3D Pose is useful for many applications, such as tracking a person, or for detecting gestures. In our case pose could also be used to track behaviour in animals such as in [3] where they use estimated pose to determine an animal's behaviour. This could be useful in animal behaviour analysis where a researcher has many hours of video of a subject and would like to search by specific behaviour.

The human pose estimation techniques today currently rely on the large size of these datasets to learn pose, and synthetic data is rarely used. Synthetic data in general is artificially created with the help of algorithms. The most notable paper that uses synthetic images to produce a human pose estimator is [4], where they create a pipeline that generates synthetic photo realistic images of humans. These images are used to train their pose estimator.

We aim to create a pipeline that uses synthetic images to train multiple neural networks that will as a result be able to predict its 3D Pose. In particular our dataset will be a species of stick insect called the Sungaya Inexpectata (also known as the Sunny Stick Insect). This synthetic dataset is photo realistic and has been given to us by a team in Imperial College London. The real size of the stick insect is around 5 cm long and quite small in comparison to a human. We showcase a real and synthetic image of the stick insect in Figure 1.1. This research allows us to work towards the goal of having a 3D keypoint estimator for a subject that has never been interacted with such as our stick insect that has very few real images online and zero 3D interpretations of the real insect.

Figure 1.1: (a) is an image of a real Sungaya Inexpectata (b) is an image of a synthetic image of the same species

1.2 Contributions

In this section we provide the contributions made by this project:

1. We used Pytorch an optimized tensor library that is used to implement deep neural networks. We created two models with this library:
   (a) The first model we implemented is the 2D keypoint estimator that takes an image as input and outputs 2D points. This model was based off of [5].
   (b) The second model we implemented is a 3D keypoint estimator using the lifting technique. This model takes 2D points and outputs 3D depth. The architecture we based this model of off was based on [6].
2. We also created a training and testing pipeline to allow us to easily create and test new models created on our dataset
3. We implemented different loss evaluation metrics into our pipeline, such as Masked Mean Squared Error that allow us to test these new models.
4. We also implemented the PCK (Percentage Correct Keypoint) accuracy metric (both 2D and 3D) into our pipeline that gives us our main unit of comparison to evaluate models.
5. For our synthetic dataset we integrated it with PyTorch and used these libraries to create different pre-processing techniques to improve performance.
6. We collated a small set of real images of real stick insects from the same genus. We also labelled the 2D keypoints for each image and then integrated it into Pytorch.

1.3 Structure of the Report

- **Chapter 1**: Introduces the project and its motivations.
- **Chapter 2**: Provides relevant background into pose estimation and its history. It also goes into more depth about the other datasets mention earlier in the chapter.
- **Chapter 3**: Discusses the method in which we carry out pose estimation and details in which we used to implement it.
- **Chapter 4**: Presents the results of experimentation with the trained implementations and evaluates them.
- **Chapter 5**: Wraps up the report summarising its outcome and mentions the work that could be done in the future.

Chapter 2

Background

2.1 Pose Estimation

Pose estimation is when a computer system is used to detect the pose of a human or object that it is trained for from either an image or video. Pose is the positioning of an object or thing (e.g. sitting, standing) typically represented by keypoints. Estimating pose is done by determining the keypoints in the image and inferring the pose from them. Examples of keypoints are in the context of a human is the right knee, left elbow and so on but is not strictly since pose estimation can be applied to animals. Pose estimation task can be grouped into two sets of approaches:

1. Bottom-up is where the machine learning algorithm estimates each of the keypoints by going for a pixel by pixel estimation to get a larger understanding of the pose. Essentially focusing on the composition of the smaller learned features to then merging them together to estimate a pose.

2. Top-down is when the full object is detected first and represented in a bounding box (this is a rectangle that surrounds a detected object). This detected object is then decomposed into smaller problems to find all the keypoints of the object which are then brought together to determine the pose.

Pose estimation when applied to videos could provide an estimator with more data allowing it to be more confident due to pose not changing much when thinking about a few frames of video. More sophisticated implementations are able to perform pose estimation on multiple objects. In Figure 2.1 we see that the green boxes surround each limb of the actor and red circles represent the keypoints at the key joints. In recent years the most favourable technology used for pose estimation, (giving cutting edge results) is to use specifically designed convolutional neural networks (CNNs). We discuss the history of what came before in the next section.

2.1.1 History

This section discusses the history of pose estimation and techniques from object recognition that influenced the development of the pose estimation branch of computer vision

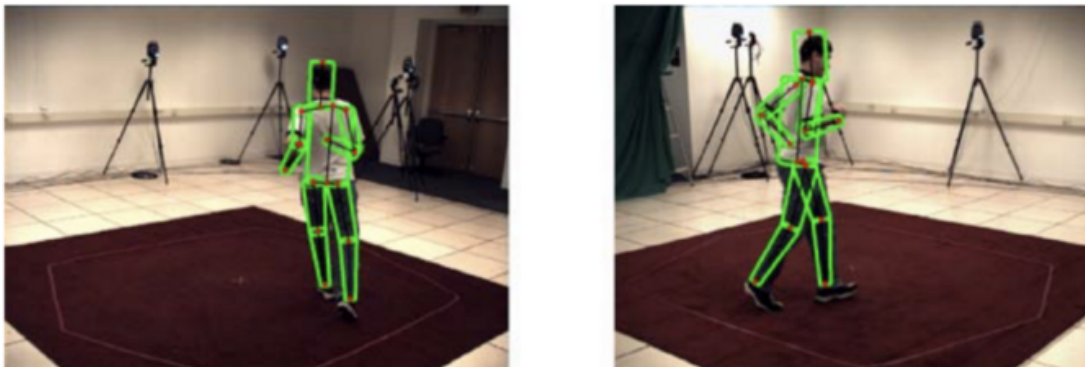


Figure 2.1: Example of 3D pose estimation. Image adapted from HumanEva II [7].

to the current use of Convolutional Neural Network's (CNNs).

Pictorial structures[8] is one of the earlier occurrences of a pose estimator. It used the concept of deformable objects and the relationships that the parts had with the object to determine a pose. This method is a more probabilistic method of determining the parts of an object. The problems that occurred were that they were not computationally efficient due to struggling to minimise the energy and the large number of parameters required.

Another influential part of the research was conducted by a team at Microsoft for the XBOX Kinect [9]. They used the decision tree concept to ask pose estimation questions like" what is the likelihood that this pixel in an image of a person is part of the right shoulder". Developing on this concept they developed a deep randomised decision forest classifier trained on synthetic data. The key advance over the pictorial structures was the added depth provided by the Kinect sensor. This allowed for better performance due to added dimension. The Kinect sensor caused this method to be less desirable due to needing the requirement of an extra hardware on top of a camera.

In the next section, we will introduce single frame camera (Monocular) pose estimators (2D and 3D). These methods remove the use of extra hardware and use the advancement in neural networks (NNs) to train a smarter model.

2.1.2 Application

Pose estimation has many real-world applications:

Physical health and fitness apps have had a rise in the past 18 months and more apps are using pose estimation as a way to have a personal trainer in the app for a fraction of the cost of a face to face trainer. An example of this is the HomeCourt[10] app that train specifically for basketball related exercises. This could then be used to provide an interactive experience by encouraging when it sees the user struggling and telling them when they have finished all their reps. Another part of this experience is to let the user know if the form of the exercise they are doing is wrong to minimise injury risk.

Physical therapy is another use of pose estimation that detects the user's body postures and can recommend exercises to help correct them. This is even more beneficial since it is in the comfort of their own home with a lower cost since a specialised physical therapist does not need to intervene.

The entertainment industry could benefit greatly from pose estimation since it could make the large motion capture stages obsolete. This would allow smaller lower budget films more access to CGI allowing for filmmakers the flexibility to create their vision. This argument could also apply to smaller game studios that would like to use motion capture. Motion capture to date has predominately been used by the larger studios.

The Robotics field could use pose estimation to control robots where the rigid movements can be replaced with pose estimation to give a more realistic response. This would give the robot the ability to adapt quickly to different environments and re-calibrate less than its old system.

2.2 2D Pose Estimation

2D pose estimation is the process of using a 2-dimensional image to determine keypoints on an object (e.g. human) and then using the detected keypoints to determine its pose. The challenge of 2D estimation is that the algorithm should be able to detect pose while being indifferent to environments, not just physical environments but the scale of the object from the viewpoint of the camera and cases where some keypoints could be obscured. An older method using CNNs was a top-down approach but if the object is never detected in the first place the keypoints can never be determined.

One of the current approaches used to determine keypoints is the "stacked hourglass". This approach to pose estimation uses both top-down and bottom-up methods to allow the model to use both the spatial and feature information to give its best estimation. The hourglass is a symmetric architecture that takes an image and pools(max pooling) it down to determine its feature information. The architecture then upsamples the low-resolution output from the pooling at each step adding the residual layers[11] from the pooling down stage. This approach allows for the spatial and feature information to be combined into the output. Please look at Figure 2.2 to see the operation of a single hourglass.

The stacked hourglass is when multiple hourglasses are done one after the other. The stacking of hourglasses allows the model to re-evaluate its prediction to provide a robust system. The stacked hourglass uses an immediate supervision process to allow a loss to be calculated to provide the next hourglass with data to evaluate on.

Another popular approach for keypoint estimation used is DeepLabCut[13] that builds on previous work from the DeeperCut[14] architecture. This architecture is an adaptation of the Residual network(ResNet)[11] where a sliding window-based body part detection is implemented. DeeperCut also uses immediate supervision to address the issue of vanishing gradients and create score-maps similar to how stacked hourglass creates heat-maps. The advantage that this method has over the hourglass method is

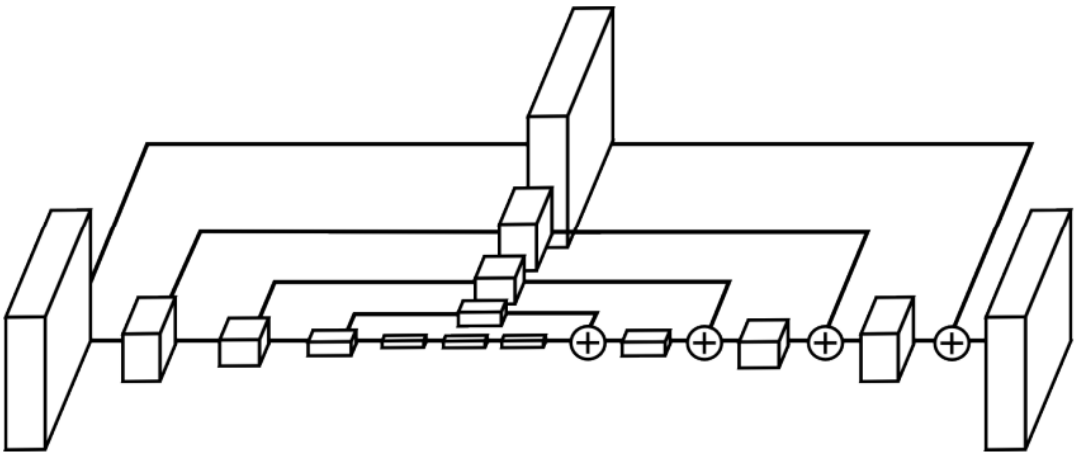


Figure 2.2: Single Hourglass architecture. Image adapted from [12].

that the architecture is designed to detect keypoints of multiple objects in a frame.

There are many architectures and approaches used for 2D pose estimation, and it is a very popular area due to the amount of fields it can be applied in. Another popular approaches used is Part Affinity Fields[15, 16] this provides accurate multi-person 2D pose estimation even for larger amounts of people in the image.

2.3 3D Pose Estimation

The process of 3D Pose estimation is to estimate the object's pose in 3-dimensions and represent that pose with a set of coordinates. In some cases, the inputs to this estimation may contain multiple camera viewpoints or a sensor that can perceive depth(XBOX Kinect, LiDAR). This process has two main sets of approaches:

1. End-to-end
2. Lifting

2.3.1 End To End

The end-to-end approach to 3D pose estimation is to input a 2D image into a CNN and the output of that deep neural network would be a set of 3-dimensional coordinates or in some cases volumetric heat maps.

One of the designs that carry out this task uses a combination of two methods joint point regression and joint point detection [17]. This detects if there is a keypoint in a window to be estimated then estimate the joint points relative to the root joint position.

Another end-to-end network design uses an auto-encoder to learn the constraints of the human body and then a CNN architecture is mapped to the dimensions of this auto-encoder. Then decoding layers are added after the auto-encoder to re-project the

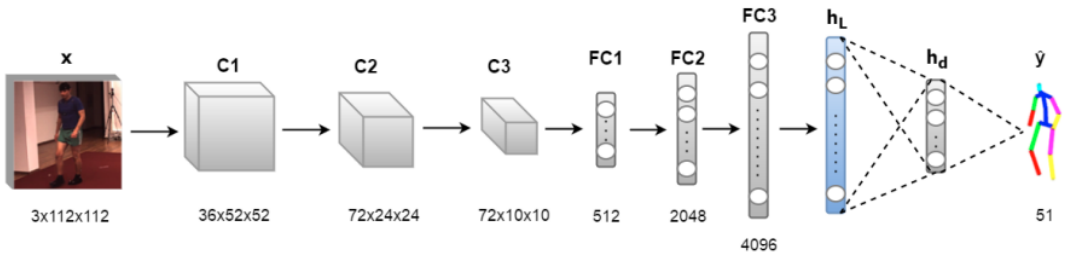


Figure 2.3: End-to-end Architecture. Image adapted from [18].

data back into the original space from the auto-encoder. This Figure 2.3 shows the functionality described.

2.3.2 Lifting

The lifting method is a more modern popular approach to 3D pose estimation. This approach is when we get estimated inputs from a 2D keypoint estimator something such as the 2D estimating hourglass architecture mentioned earlier and then use its outputs to bring them into 3-dimensions.

Another popular paper uses the same method of estimating the 2D keypoints for Humans and then lifting them in 3-Dimensions[6]. For their lifting, they designed a multi-layer neural network that used Batch Normalization[19], Dropout[20], Rectified Linear Units(RELUs)[21] and Residual Connections[11].

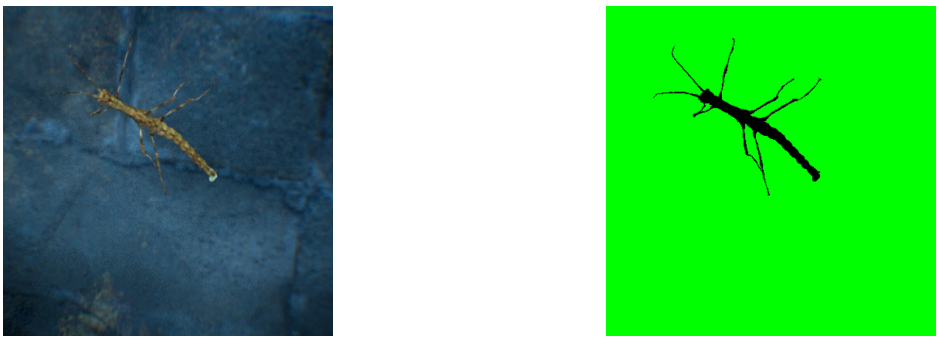
This method in general is favoured since it is computationally less expensive than the end-to-end and there are only a handful of 2D data points needed instead of the whole raw image. The only drawback with this method is that you require 2D Ground Truth to be provided as well as 3D GT.

2.4 Data Sets

There are many areas and types of datasets that have been created for pose estimation. The most basic type of dataset is a set of data that has an image of an object with labels providing the ground truth for the 2D positions of each keypoint. Most of the popular pose estimation datasets have ground truth labels for both 2D and 3D keypoints.

There is another type of 2D dataset that can also be used to aid in pose estimation is a more general object dataset. This dataset contains segmentation data to train the models initial weights, and after is training on a specific keypoint dataset afterwards. One of the datasets that could be used for this technique is the COCO(Common Objects in Context) [22] dataset. This contains more than 200,000 labelled images and 250,000 people with keypoint labels, so this dataset could also be used to train for 2D pose estimation.

The more difficult datasets to collate and acquire are the datasets used for 3D pose estimation. This is because of the large amount of time and resources required to be able to find the 3D coordinates. These datasets are only acquired in studio conditions with depth sensors or using realistic synthetic data.



(a) (b)

Figure 2.4: (a) is a render of a synthetic model of an insect while (b) is the segmentation of the same render

Datasets for 3D pose estimation have become more accessible and much larger in recent years. One of the largest datasets with 3D data is Human3.6m [23] that contains 3.6 million 3D poses with their images of 11 professional actors. They carry out 17 different scenarios. This dataset has been used to benchmark multiple human 3D pose estimation research in the past few years.
For this project we will be using a dataset that is still currently in development and has the aim of creating a synthetic dataset of arthropods. This is part of the scAnt project [24]. This project takes 3D scans of stored arthropods and allows a photo-realistic render to be made. The Figure 2.4 above is a render provided by the scAnt team and the segmentation data of the same render. Currently, the specific genus we will be using is a type of stick insect called the Sungaya.

Chapter 3

Method & Implementation

3.1 3D Keypoint Dataset

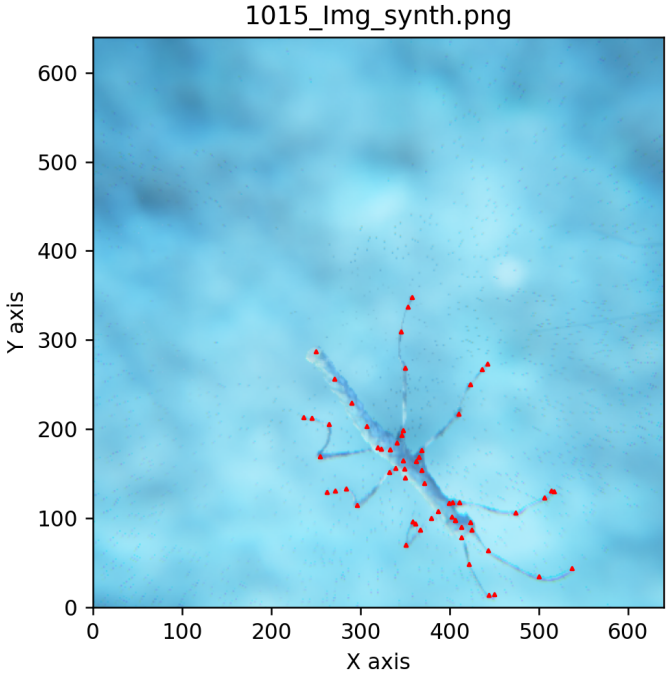
We were kindly given access to a dataset that was created by a team in Imperial College that uses a model of a Sungaya Inexpectata to create images in a rendering environment with photo-realistic backgrounds and object occlusion. The dataset we will be using in this project is a custom dataset created using scAnt[24]. scAnt is an open-source macroscopic camera that can create 3D models of any small subject at an affordable cost.

This dataset has $\approx$ 3700 samples that come with 2D and 3D Ground Truth keypoints. Ground truth in the context of a dataset is the true value of a data point. Each image has a resolution of 640x640px. This dataset also indicates which points have been occluded by the environments. The number of keypoints for a sample is 62 - much greater than most human pose datasets. Details about the virtual camera have also been given to us for each image, such as rotation matrix, translation matrix, and intrinsic matrix, allowing us to see the relationship the 2D keypoints have to the 3D keypoints. We were also given a bounding box that allowed a box to be made on the image containing the full stick insect. An example of an image of the stick insect with its 2D keypoints is shown in Figure 3.1a. The 3D equivalent of that stick insect is shown in Figure 3.1b. The Sungaya model used is based on the male gender of the Sungaya, which is roughly around 5-5.6 cm long. Its width is around 2 cm (including legs in a normal standing pose). Figure 3.1b also has all the main limbs to allow us to understand where each keypoint is for later sections.

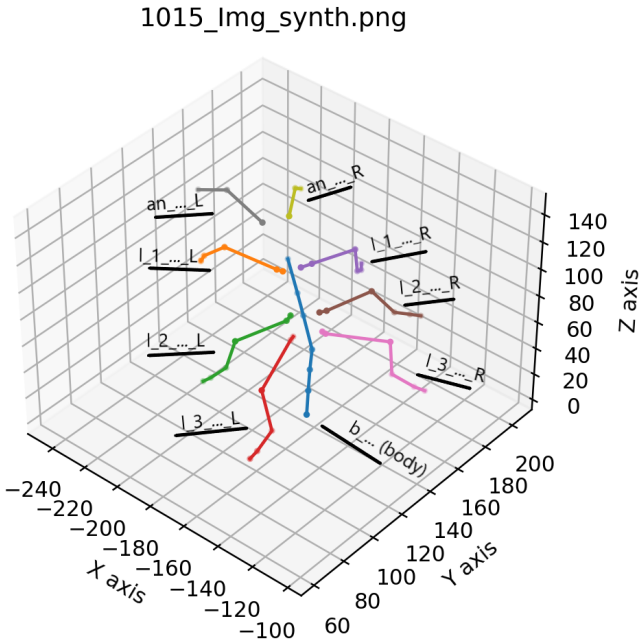
3.1.1 Data Analysis

Here we discuss a few data analytics techniques to determine some information about our dataset to check for details that could affect our neural network performance in the future. We used the following techniques.

The first technique was to determine if the limbs of the Sungaya were actually changing in 3D space. If these limbs were not changing in 3D space then the model would learn a specific bias to that pose and not recognize any different poses rendering the model



(a)



(b)

Figure 3.1: (a) is a render of a synthetic model of an insect with its 2D keypoints while (b) is the 3D representation of the same image and also has the limb names labelled.

useless. The method we used to determine if the 3D keypoints are varying from sample to sample was to use Procrustes Analysis. This method is able to ignore the translation and rotation of these points and is able to only consider the scale of the keypoints. We calculate this disparity that is a number between 0 and 1 that indicates how much the 3D keypoints are varying from sample to another sample. Disparity is calculated by using the equation below where data1 & data2 each are a standardized sample of 3D keypoints.

$$M^2 = \frac{1}{n}\sum_{i=1}^{n}(data1_i - data2_i)^2 \tag{3.1}$$

where n is the number of keypoints in the sample. The closer that disparity is to 0 is the more similar two datapoints are. The closer the disparity is to 1 is the more dissimilar two datapoints are. From our perspective we would like the disparity to be closer to 1, so we can train with a dataset that contains more variation allowing us to better generalize. We ran an experiment that determined the pairwise average disparity of the 3D keypoints to be ≈ 0.1099 and this value tells us that there is a difference in 3D keypoints between samples, but there are similarities. From the shape of the Sungaya we can estimate that this lower number could be attributed to the main body of the Sungaya that is not able to vary (in terms of pose) much, so the disparity on these points is low. As a result, dragging down the average from the other limbs that move more such as the legs.

The second technique we used was to determine the best point to use as the centre of the insect. We used the visible points as a reference to determine what point is the most visible in the dataset since we would not like to miss out on data that could be useful for training. We determined that the centre should be the third point on the main body. This was not the most visible point in the dataset, but it did not make sense to centralize the insect on a keypoint with less significance such as its leg.

3.1.2 Data Preprocessing

This section discusses the data preprocessing we carried out to prepare our dataset for training. We used the following techniques:

The first technique was to remove the outliers with the centre keypoint not visible in the dataset. To do this, we removed the samples with the centre keypoint that were not visible. Removing data without this keypoint reduces the number of images we had from 3778 to 3266. We then used this visible keypoint to centralise our 3D GT poses to the centre of the bug. Centralising was done by subtracting the centre keypoint from all the other keypoints. We did this because the 3D GT poses were not centralised initially, which could hinder our lifting models' performance. Doing this technique gives us a fully centralised dataset that can be used for training.

The following technique was used to prepare our images for our first network that takes an image and predicts the 2D keypoints. During our data analysis, we noticed that the bounding boxes given to us were incorrect - as a result, we cut off the insect's limbs, so we implemented our code that took the 2D GT and produced our bounding box. This bounding box was also used to crop the images to the size of the bounding

box. Cropping was done to reduce the amount of data we had to train on and force our network to learn to detect the keypoints in the image and not other irrelevant data. After this first process, we then scaled all the images to the fixed size of 264*264, reducing the complexity of training the first network and the amount of graphics memory required when training. Our 2D GT keypoints were scaled alongside these transformations to maintain the image's relationship and the 2D keypoints.

Another method we used to reduce the complexity of the processing required is to use a reduced number of keypoints but still keep the same shape but more generality. We did this by manually inspecting the images with the keypoints labelled to determine which keypoints had the most valuable data. The keypoints that matched these criteria were mostly joints that varied a lot or joints that connected to the main body. Points like the feet joints were so close together that, in this case, we treat one of them as a single point. After the reduction, the number of keypoints went down from 62 to 28.

We also applied the most common preprocessing method, standardization, that was applied to both 2D and 3D GT. This is done by calculating the standard deviation & mean over the whole dataset for each keypoint.

$$\sigma_i = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(data_i - \mu_i)^2} \tag{3.2}$$

$$\mu_i = \frac{1}{l}\sum_{j=1}^{l} data_i \tag{3.3}$$

$$data_i = \frac{data_i - \mu_i}{\sigma_i} \tag{3.4}$$

Where l is the number of samples in the set. This method keeps the relationship that all the points have but constrains the numbers between 0 and 1 to reduce the complexity of the data.

For computer vision tasks, it is very common to flip images horizontally and vertically. In our implementation, this was done randomly to allow us to experiment if the network was better at detecting the keypoints in the image or not. We implemented this technique with the help of the OpenCV package. We used the flip function to flip the image horizontally and vertically. Furthermore, we then used the same transformation to the 2D keypoints.

The last technique we used was to convert our 2D GT keypoints into heatmaps for each keypoint. This was done to accommodate for our ResNet adaptation that takes a list of heatmaps for each keypoint as input. We converted these keypoints into heatmaps. We used a Gaussian kernel to create the heatmaps. The equation we used to create the heatmaps was:

$$\mathbf{heatmap}_n = \frac{1}{\sigma_n^2}\exp\left(-\frac{(data_n - \mu_n)^2}{2\sigma_n^2}\right) \tag{3.5}$$

In our dataset, we defined σ_n^2 as 2 since this defines how large the kernel is. It would be interesting to see how training the network on a different size kernel for smaller size

keypoints affects the performance. These heatmaps were created with a resolution of 64x64 to keep the amount of data required to a minimum but enough to give sufficient data for training.

3.2 Keypoint Estimation

The pipeline created in this paper will consist of a 2D keypoint estimator based on [5] that feeds its output into a 3D lifting network adapted from [6] that then predicts a set of the 3D position of keypoints. The reason for picking this pipeline over an end-to-end model is that it can allow the 2D network to be trained on other samples that do not have 3D Ground Truth (GT). Using this pipeline allows the network to be trained even if there is less 3D GT data. Training the 2D network on more samples would make a more robust model predicting 2D keypoints. Another advantage of using this pipeline over an end-to-end system is that we are forcing the model to learn features about the positional data instead of trusting it will learn the correct distinctions. The 2D keypoint estimation model can be depicted below:

$$\hat{\mathbf{H}} = f(\mathbf{x}_j) \tag{3.6}$$

Where f is the 2D keypoint estimator and $\mathbf{x}_i$ an input image. The output $\hat{\mathbf{H}}$ is the predicted the set 2D heatmaps The 3D lifting network can be shown below:

$$\mathbf{y}_k = g(\mathbf{x}) \tag{3.7}$$

Where g is the 3D lifting network and $\mathbf{x}$ is a set of 2D coordinates for each keypoints. The output $\mathbf{y}$ is the predicted 3D keypoints. We can now define the full pipeline from the image to the 3D keypoints as:

$$\mathbf{y}_k = g(m(f(\mathbf{x}_j))) \tag{3.8}$$

Where the equation m is a function that calculates the argument max of the heatmaps per keypoint to create a set of 2D keypoints. This final equation represents the whole pipeline of being able to predict 3D poses.

3.2.1 2D Keypoint Estimation Architecture

The 2D keypoint estimator model [5] uses ResNet [11] as its backbone in conjunction with deconvolutional layer to produce 2D heat-maps for n keypoints. These heatmaps provide an area in which the keypoint is estimated to be. Specifically, we will be producing a 64x64 heatmap that will illustrate a confidence(or belief) map of where the network believes the keypoint to be. As we defined in the previous section, the heatmaps are Gaussian kernels with a fixed size. Heatmaps give the network a better representation of the keypoint location and allow us to see what the network learns if we inspect the heatmaps during training.

The ResNet has a prevalent and exciting technique that is used is where the network is pre-trained on a more general dataset such as COCO [18] and researchers fine-tune the weights with their own 3D pose data afterwords. We chose this architecture because it

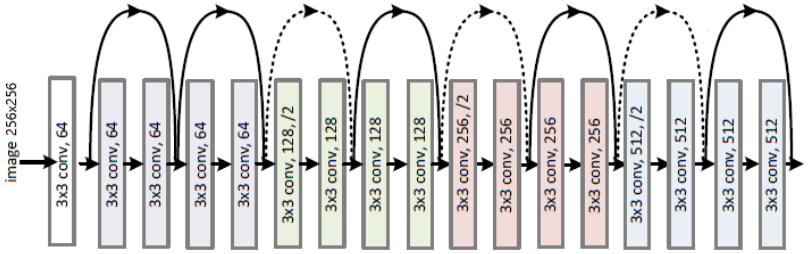


Figure 3.2: ResNet-18 Architecture until stage C_5. Image adapted from programmer-sought.com.

provides a well-performing keypoint estimator at a lower computational cost in GPU (Graphic Processing Unit) processing. If we were to have implemented a more expensive model such as in Convolutional Pose Machines [25], we would have had to reduce parameters such as the batch size. This reduction of parameters would have negated the benefit of using a more complex model and caused increased training and prediction times due to its complexity. Another element of running a ResNet as a backbone is that there are deeper configurations available of the architecture. These deeper architectures have more layers that could learn more fundamental features about the stick insect and could, in the future, return better results than the shallower architecture that we are opting to use.

The ResNet [11] backbone used provides a deep, robust network that is trusted for the task of deep image feature extraction and allows us to add deconvolutional layers to the end of the ResNet architecture that has learned useful feature maps. Then that output can go through the deconvolutional layers to learn the 2D heatmaps for each of the keypoints.

The specifics of the ResNet backbone that will be used is the ResNet-18, which contains 8 layers of 'Basic Blocks' that contain 2 convolutions. Each block varies in the input size and takes advantage of residual connections at the end of each block. Residual connections essentially take data from the previous network block and add it to the current block's output. This concept allows models to become deeper since this combats the Vanishing Gradient Problem (VGP). Their architecture also uses RELU (Rectified Linear Unit) as the activation function. This activation function allows faster computation since traditional activation functions are slower during training and this method also actively combats VGP.

The deconvolutional layers are added at the end of ResNet-18 architecture at a stage called C_5 this stage is at the end of Figure 3.2. The newly added layers are illustrated in Figure 3.3. These layers allow us to take advantage of the deep and effective nature of ResNet-18 and learn the features that we need to predict the 2D heatmaps from the image successfully. We can obtain the most confident spots by using the brightest point on the heatmaps for each keypoint.

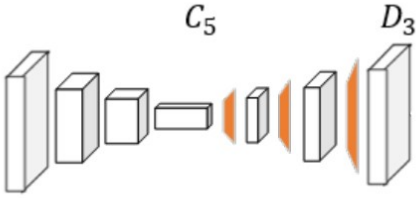


Figure 3.3: The ResNet-18 output feature map into deconvolutional layers. C_5 depicts the output from the network in Figure 3.2 and shows deconvolutional layers being used to produce the heatmap output. Image adapted from [5].

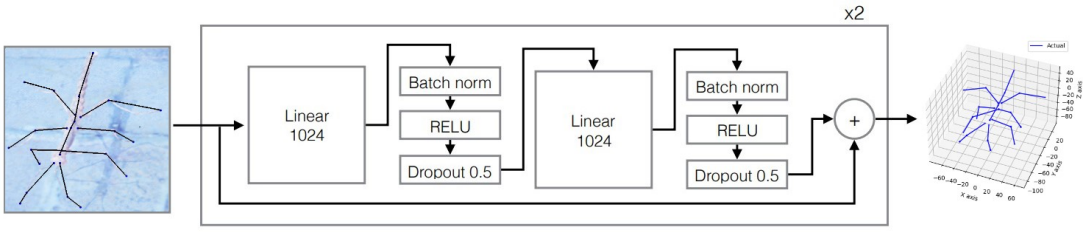


Figure 3.4: Our Lifting Architecture. Image adapted from [6].

3.2.2 3D Keypoint Estimation Architecture - Lifting

The 3D Keypoint Estimation architecture we thought was suitable for this project was the simple yet effective baseline for 3D human pose estimation [6]. They introduced a simpler architecture that, compared to others in the field, takes 2D points as input to the model and predicts their location in 3D world space. This architecture is a deep, multi-layer network that uses the latest advancements in DNN architecture design, specifically the introduction of Residual Connections [11], Dropout [20], RELUs [21] and Batch Normalization [19].

This architecture is depicted in Figure 3.4 and contains the main block iterated through twice. This block consists of two linear layers, batch normalization and the use of dropout. The linear layers carry out feature extraction from the 2D keypoints. Batch normalisation[19] is primarily used to allow the network to be more computationally efficient by keeping the weights of the network to a more workable range. Dropout layers are also used to cause the model to generalise the weights by dropping some neurons in the network by some probability to improve the model's generalization. Generalization occurs because the network cannot be dependent on specific neurons forcing the network to learn deeper features about the task in question. The activation function used by this network is the RELU. Like in the 2D keypoint estimation architecture is used for faster computation that would reduce training time compared to other activation functions.

The reason for choosing this simpler architecture is because considering its lower cost architecture and the results shown in [6] shows an impressive result. We want to recreate that same performance in this report. In future, this allows us to build upon this network

3.3 Training Details

Our implementations of both the adapted ResNet and the simple 3D lifting model were implemented using Python 3.8.2. The training was performed using a GPU, specifically an NVIDIA RTX 2060 Super. The main library we used for deep networks was PyTorch. The main library for image processing was OpenCV.

3.3.1 2D Keypoint estimation

We will now discuss the key elements we used to implement the 2D keypoint estimation network. As we discussed earlier, we used a ResNet-18 as our base network. In this implementation, we did not use a network with pre-trained weights. This network was only trained on the stick insect synthetic data we were provided. The network was trained on the following parameters:

- Learning rate: 0.0001
- Batch size: 32
- Epochs: 150
- Train/Validation/Test split: 70/10/20
- Image Resolution (Height/Width): 256/256

We determined these hyperparameters via experimentation and found that our model performed best in the validation set with a learning rate of 0.0001. The batch size was 32 because our GPU memory constrained us, which only had 8 GB VRAM, and anything higher than that caused us to run out of memory for 1 batch. We ran the network for 1 epochs and saved the network's weights after each epoch. Then we cherry-picked the better performing epoch manually analysing via validation loss to ensure we do not overfit the network. This network took around ≈ 4 hours to train.

The optimiser we used was the Adam optimiser. We used the learning rate of 0.0001 and the momentum of 0.9. We opted to not use weight decay, but we did use beta parameters, $\beta_1 = 0.5$ & $\beta_2 = 0.999$. These beta parameters are the hyperparameters that tune the exponential decay rate for the first &second moment estimates. These parameters are used to define how the learning rate decays over time. This technique allows the Adam optimiser to achieve good results over a shorter time. The Adam is known for its good convergence rate and robustness even under its default parameters. It would be interesting to investigate and tweak these parameters in the future to better tune the optimiser. However, a more suited optimiser for high tuning might be the Stochastic Gradient Descent Optimiser.

The metric we used to determine the loss at each epoch was called Masked Mean Squared Error (MSE). This masking was done to ignore the keypoints that were not

visible in the image. As a result, the losses computed by keypoints not visible would not contribute to the network's learning algorithm. Masked MSE is calculated as follows:

$$Masked\_MSE = \frac{1}{n}\sum_{i=1}^{n}(M_i(data_i - \hat{data_i}))^2 \tag{3.9}$$

Where n is the number of keypoints in the image, and M_i is the mask for each keypoint. The $data_i$ is the ground truth keypoint heatmap, and $\hat{data_i}$ is the predicted keypoint heatmap. We then mask the MSE by multiplying it by the mask, in our case, is an array of ones and zeros that correspond to a visible point or not.

Another vital element of the network was the accuracy metric. Which is used to verify the accuracy of the network. The metric we used was called Percentage Correct Keypoints (PCK). This metric is calculated as follows:

$$PCK = \frac{1}{n}\sum_{i=1}^{n}(|data_i - \hat{data_i}| < thr) \tag{3.10}$$

Where thr is defined as the threshold. If the difference is less than thr, the result is 1. Otherwise, the result is 0. PCK, as a result, calculates a percentage of the keypoints that were correctly predicted. In our implementation, before we could evaluate how well our model did, we had to calculate the argument max of the heatmaps to calculate the brightest point in the heatmap predictions and use these as the 2D KP. The accuracy uses a threshold value to determine what is considered a correct keypoint. The threshold value was set to 0.5. If the distance between the ground truth and the predicted keypoint is less than 0.5, it is considered a correct keypoint.

3.3.2 3D Keypoint estimation - Lifting

This section will discuss the critical elements used to implement our 3D keypoint estimation network. As we have mentioned at the start of the section, we will be carrying out a lifting implementation, so the inputs to the network are a list of 2D keypoints that are then "lifted" into the 3rd dimension. Our implementation is a slight variation on this idea where instead of predicting the full 3 dimensions, we only predict the Z coordinate - we also call this depth. We attempted to implement the model with the 3-dimensional output but failed because the accuracy was so low that we could not train the model. So given that we struggled to implement a 3D lifting model, we opted only to predict the 3D keypoint depth. This network's output is depth in world coordinates where the stick insect is centred (0,0,0). World coordinates are the coordinates that the camera is looking at, so the depth is defined as the distance from the camera to the stick insect joint (keypoint). The model is technically learning camera parameters and the camera intrinsics and tries to generalise them for every data point while also learning the depth's relationship with inputted 2D keypoints.

Knowing all these newer detail about our implementation we will now discuss the training details:

- Learning rate: 0.001
- Batch size: 128

- Epochs: 150
- Train/Validation/Test split: 70/10/20

The learning rate we have established was determined by testing different learning rates and finding the best one using the validation accuracy as a guide to determine suitable performance. This learning rate was used in combination with a learning rate scheduler. We used a lambda function with an extra two hyperparameters: gamma and decay step. The gamma hyperparameter is used to determine how quickly the learning rate decays. We choose a value of 0.96. This makes sense since we do not want to decay the learning rate too quickly because the optimiser might start learning too quickly at more refined detail, causing it to learn the incorrect attributes. The decay steps hyperparameter determines how often the learning rate decays. We choose a significant value of 100000 that allows the learning rate to decay at a slow rate. The value is calculated as follows at every epoch(i):

$$NEW\_LR = LR * \gamma * \frac{i}{decay\_step} \tag{3.11}$$

As a result, our learning rate would dynamically reduce over time.

The batch size was 128 because we were constrained by our GPU memory which was only 8 GB and anything higher than that caused us to run out of memory for a batch. We ran the network for 150 epochs and saved the network's weights after each epoch. Then again, we cherry-picked the better performing epoch analysing via validation loss & validation accuracy to ensure we do not overfit the network. This network took around ≈ 5 hours to train.

We used the same Adam optimiser for the 2D keypoint estimation model due to its robust nature and quick training to a reasonable accuracy. Again we did not use weight decay because we are using the Lambda learning rate scheduler in combination with Adam.

Another valuable thing we did at training time was to implement a max norm regulariser to the network. This regulariser is used to prevent the weights from becoming too large. This is done by dividing the weights by the max norm of the weights. As a result of this, the network weights are less than or equal to 1. The advantage of this is that it is shown to help the network generalise more efficiently and improve the validation set's accuracy.

The loss function we used here was fundamentally the same as the 2D keypoint estimation model. The reasoning for this is the same as before. The only difference is that we now are only calculating loss on depth instead of a heatmap applying the masking in the same way by multiplying the output squared error by the mask that is a list of keypoints visibility. This masking is done to ignore the keypoints that were not visible in the image. Calculating this results in us obtaining a masked loss, so it ignores the keypoints that were not visible in the image.

The final element of training the lifting model was the accuracy metric. This metric is a variation on PCK where we only calculate the accuracy of the depth. Threshold, again being the appropriate distance away from the GT. We used a different threshold value of 5.0 to determine what is considered a correct keypoint. The scale of 5.0 is circled in

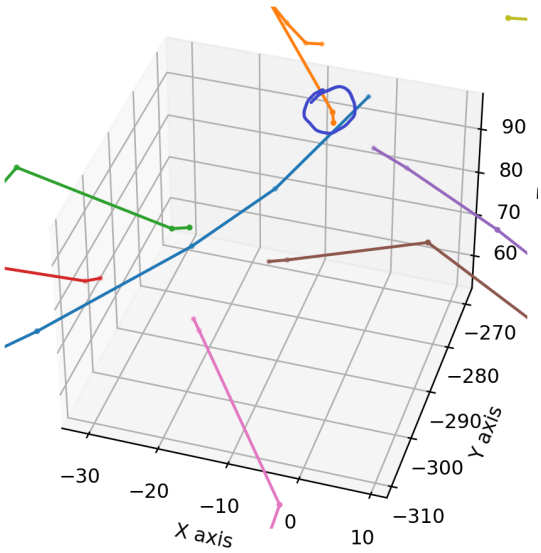


Figure 3.5: Scale of how large the threshold is in comparison to the rest of the insect

Figure 3.5 in comparison to the scale of the 3D stick insect. Conceptually this PCK is the same as before, but now we are only calculating the accuracy of the depth.

Chapter 4

Results & Evaluation

4.1 Training Results

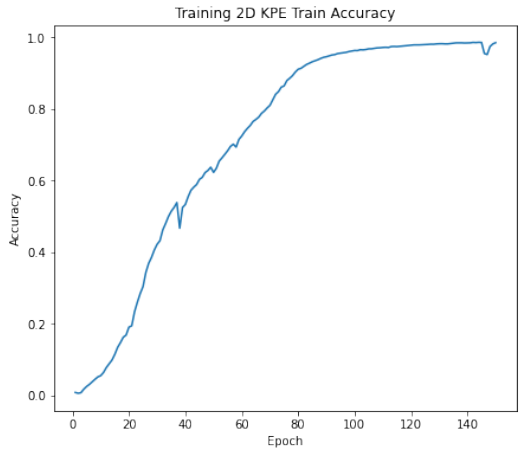
This section will discuss the results of training both the 2D keypoint model and the 3D lifting model.

4.1.1 2D Model Training

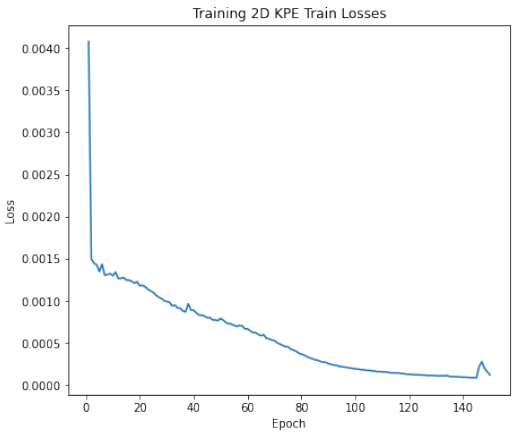
In Figure 4.1 we show the training and validation accuracy of the 2D network at each epoch, as well as the losses. This Figure gives us insight into how well the network trained over the 150 Epochs. We can see that the training loss gradually decreases while the accuracy increases. On the other hand, the Validation loss (log scale) was very erratic initially and then gradually increased, showing that the network was starting to learn the critical features of the image and was able to generalise. The accuracy on the validation set was also very low initially, with a few large spikes downward and then increased as the network was learning. It would be interesting to see if the model was still able to learn more if left on for longer, but due to time constraints, we could not. On parameters defined in the previous chapter, we found that the best performance of our network had a validation accuracy of 0.6016 at Epoch 128.

4.1.2 3D Model Training

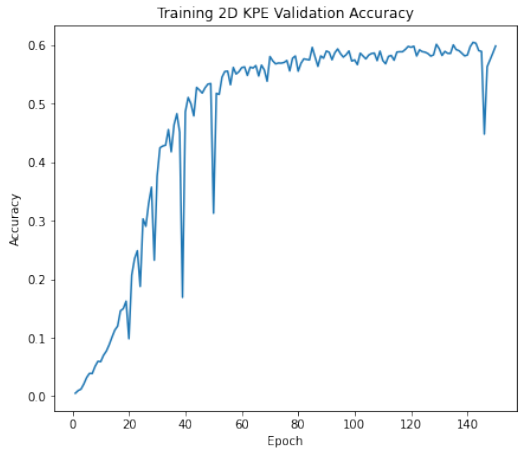
In Figure 4.2 we now show the training and validation accuracy of the 3D lifting model at every epoch and its losses. We can see that the training loss and validation loss are reducing with the validation accuracy. The validation accuracy during training had a few declines but was steadily increasing. This model also may have benefited from training for longer because there was never a huge drop off in the validation accuracy to let us know the model was overfitting. On the parameters defined in the chapter, we got a validation accuracy of 0.6516 at Epoch 128.



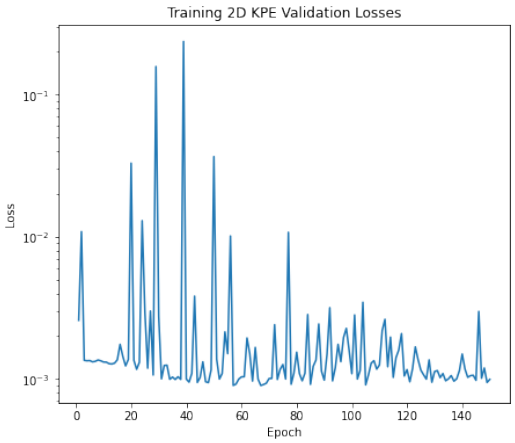
(a) Training Accuracy



(b) Training Loss

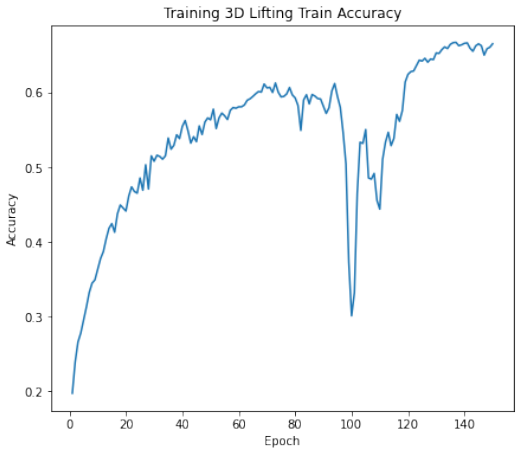


(c) Validation Accuracy

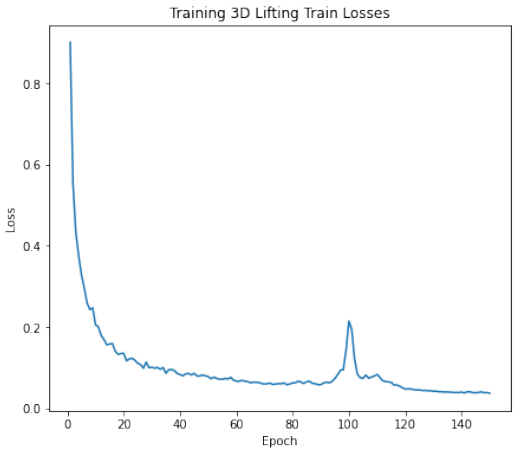


(d) Validation Loss

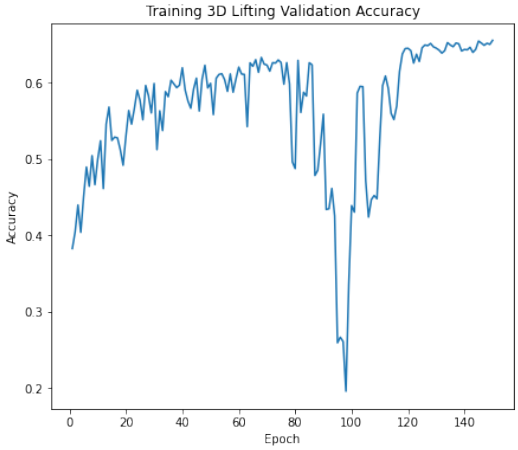
Figure 4.1: 2D keypoint estimation training and validation accuracy and loss.



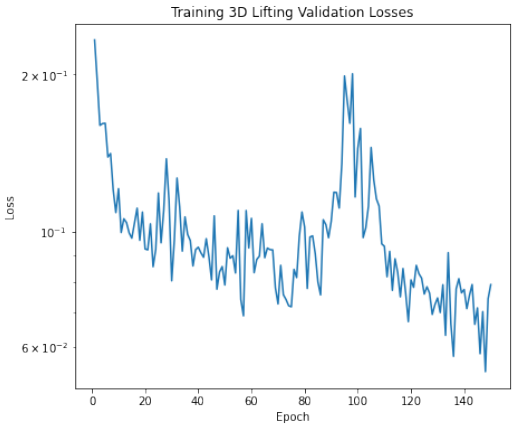
(a) Training Accuracy



(b) Training Loss



(c) Validation Accuracy



(d) Validation Loss

Figure 4.2: 3D keypoint estimation training and validation accuracy and loss

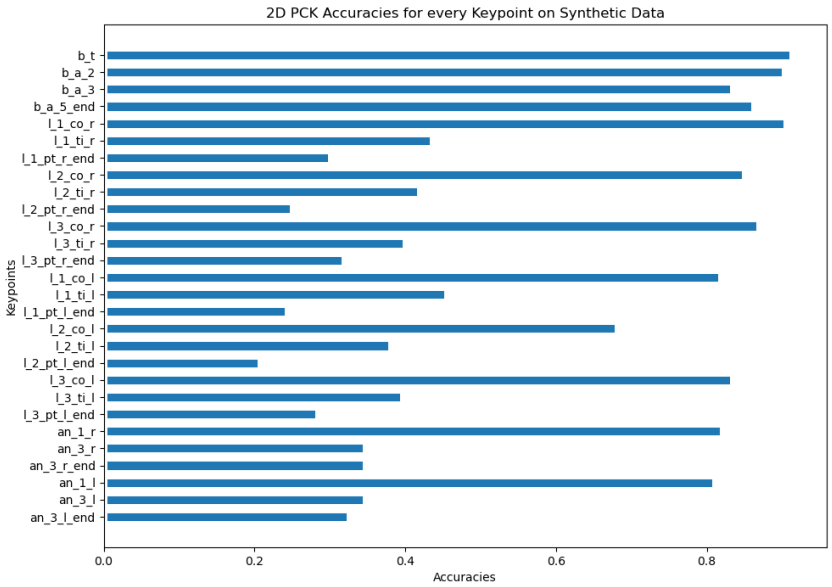


Figure 4.3: Accuracy of each keypoint of the 2D keypoint estimation model.

4.2 Synthetic Performance

The synthetic evaluation will contain three investigations. First is the 2D keypoint estimator, where the test input will be a set of synthetic images, and the output will be a set of 2D keypoints. Second, given the 2D ground truth, lift the 2D keypoints to 3D space. Third, combine these two models to predict the 3D GT from the image providing a 2 stage system. We will also evaluate these models and determine what else could be done to improve the accuracy of the models or different techniques that could be used.

4.2.1 2D Keypoint Estimation Investigation

We decided to use the highest performing epoch of the 2D keypoint estimation model on the validation set to test the network's performance on the synthetic data. Given this, we find that the PCK accuracy on the 20% test set is around 0.556, with a threshold value of only 0.5 pixels. This value of 0.5 might seem relatively small given the size of the image, but we compare the heatmap dimensions that are reduced to 64x64, so 0.5 pixels is not as small as we initially thought. This value is quite reasonable given that the network is relatively shallow compared to other networks and even other variations on ResNet architecture. The Masked MSE loss we achieved was 0.00026.

The first experiment we do is determine where our network fails to predict correctly. Instead of calculating an average over all the keypoint accuracies, we do this. We adapt our code to show the accuracy of each keypoint. Running with 20% of the synthetic dataset, we achieve these values that are portrayed below in Figure 4.3. We see that the model can predict the main body (b to b_end) of the bug well with accuracies higher than 0.8. The start of each limb that comes from the main body is labelled as _co_ for

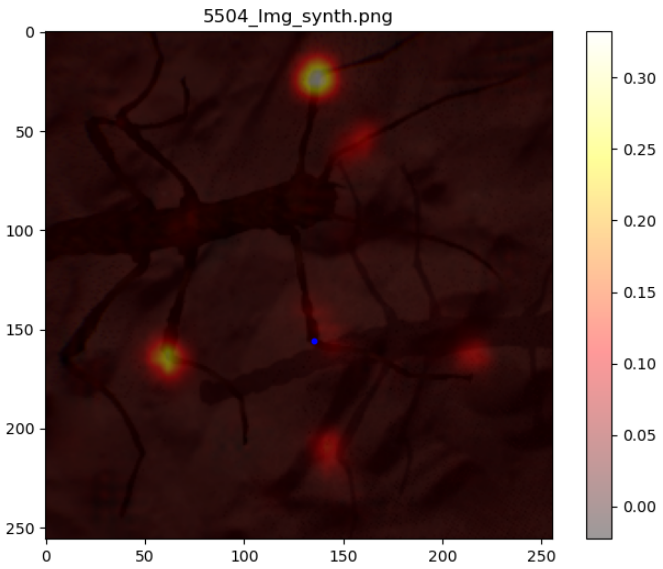


Figure 4.4: Heatmap of the l_1_ti_r keypoint from the 2D keypoint estimation model. Where the heatmaps are the estimates and the blue point is the actual keypoint.

main legs & an_1_... for the insect's antennas. These are well predicted with an accuracy of ≈ 0.8. The accuracy of the other limbs is lower than this. The more inaccurate limbs are the keypoints equivalent to the elbow and hands on a human. The accuracies of these keypoints vary from as low as 0.28 to 0.45. These are the remaining labels we have not mentioned in the graph.

Upon further inspection of these specific keypoints, we can see that there are two heatmaps where the network is confident in two locations that happen to be the limb on the opposite side of the bug. We use a challenging example see this in Figure 4.4 that heatmaps do show on the correct points but on the wrong limb. We use a more complex example because our dataset generally has tricky images to predict. The network seems not to be able to tell the difference. This behaviour of mismatching limbs is very consistent with the rest of the test set. We attempted to combat this by using random rotation in the training set, but that did not have a positive effect. The tricky thing about this dataset is that many reflective surfaces and shadows make it even harder for our model to predict the correct keypoints confidently. We could not avoid these since we could not distinguish between images with and without these more challenging environments.

We also experiment with varying thresholds of the PCK accuracy metric to see how the accuracy varies. As we expect, the higher the threshold, the higher the accuracy. We depict this experiment in Figure 4.5. This data shows that potentially the threshold is a bit low, and varying the threshold to something closer to one might be more representative of how the model performs on the test set. This change makes sense as when we manually inspect the heatmaps, in some cases are in the correct place but still in the correct place would be classified as incorrect. Statistically, this is backed up by

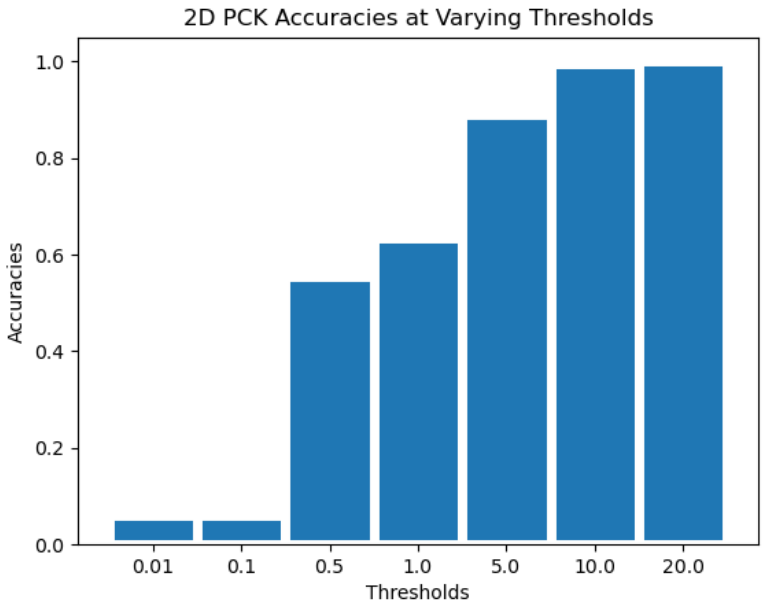


Figure 4.5: Accuracy of the 2D keypoint estimation model with varying thresholds(pixels in the heatmap).

our experiment since the increase from 0.5 to 1 is not large enough to be a significant difference but enough to start classifying correctly. A method we did not carry out could be to use this data to allow us to tune the threshold. For example, manually inspecting the image and deciding what distance is too far. As a first implementation, this network is a good step to working toward a better model, given more time to tune and refine the model's parameters

4.2.2 3D Lifting Investigation

In this section, we investigate the performance of our 3D lifting network on synthetic data. As we mentioned at the chapter's start, we are using the 3D model's highest performing epoch on the validation set to evaluate with our test set. We do this by inputting the synthetic GT 2D keypoints & comparing our model's output to the Z coordinate of each keypoint in the 3D GT. We find that the PCK accuracy on the test set is around 0.653, with a threshold value of 5.0. The Masked MSE loss we achieved on this test set was 0.0693.

Following the same procedure as before, we calculate the PCK accuracy for each keypoint. We have displayed this data in a plot in Figure 4.6. This plot shows us a very similar representation as Figure 4.3, where the keypoints are the main body or limbs connected to the body have very high accuracies but the further the limbs get the most the accuracy degrades. We can see that b_a_3 is close to an accuracy of 1.0. The network would learn this distinction since the stick bugs are all centralised on (0,0,0). The accuracy degradation could be that the network is in the correct position, but the threshold could be too low. We will plot in Figure 4.7 the actual and prediction of a model to allow us to investigate the issue. When looking at this Figure, the Z-axis we

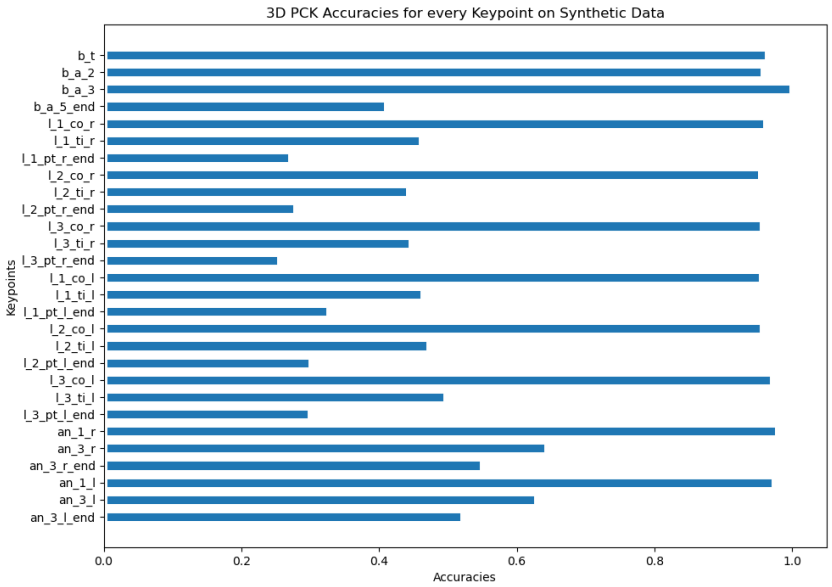


Figure 4.6: Accuracy of each keypoint of the 3D lifting model.

predict recognises the general relationship the end joints have, not the pinpoint accuracy that the current threshold of 5.0 has.

This information leads our next experiment into how the overall PCK accuracy changes given a varying threshold value. We plot the PCK accuracy of the model with varying thresholds in Figure 4.8. After a certain point, we can see that the accuracy jumps from a low 0.65 with a threshold of 5.0 to a high 0.85. A jump of 0.20 points leads us to believe that the accuracy of these keypoints is starting to be included as correct, causing the average to jump up, leading us to believe that the threshold is too low and will need to be tuned.

4.2.3 Two-Stage Investigation

This section will investigate the performance of our two-stage network on synthetic data. The two-stage implementation is just the two best performing models mentioned previously are combined. As a result, it allows us to input an image into the network and have it output the pose of the stick insect in 3D space. We evaluate the two stages at the end using the lifting model's PCK to determine the accuracy, evaluating the model's outcome. We find that the PCK accuracy on the test set is around 0.0529, with a threshold value of 5.0. We use the same PCK accuracy metric we used to evaluate the 3D lifting model in the 3D Lifting Investigation. So we only evaluate at the end of the pipeline by comparing the predicted 3D depth against the 3D GT depth. The MSE loss we achieved on this test set was 5.175. This outcome seems suspicious since these values are substantially lower than expected from the model, especially when their 2D and 3D parts are much more accurate, with PCK accuracies of 0.556 & 0.653, respectively. We would expect the use of both models in conjunction would have an

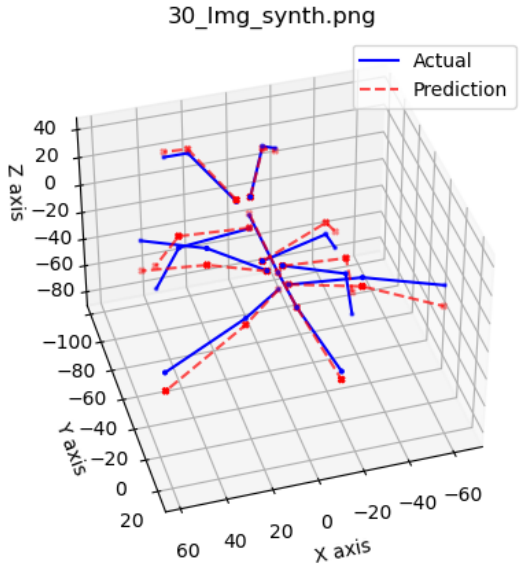


Figure 4.7: Actual and predicted 3D lifting keypoint of the model.

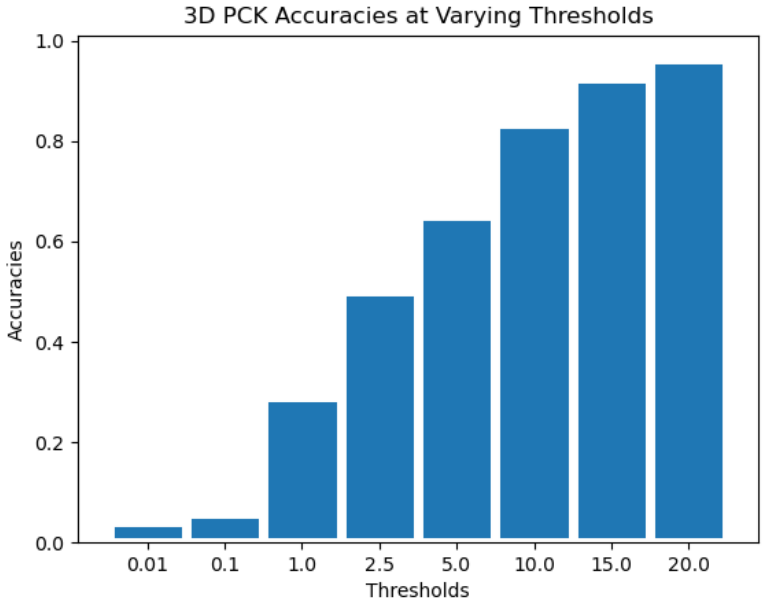
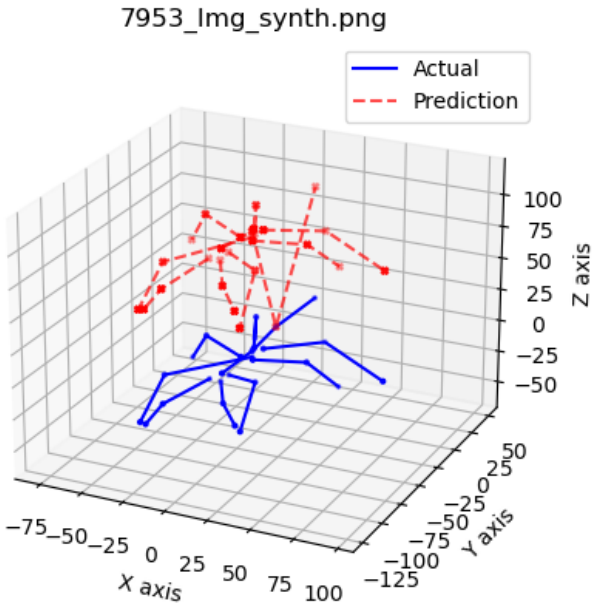


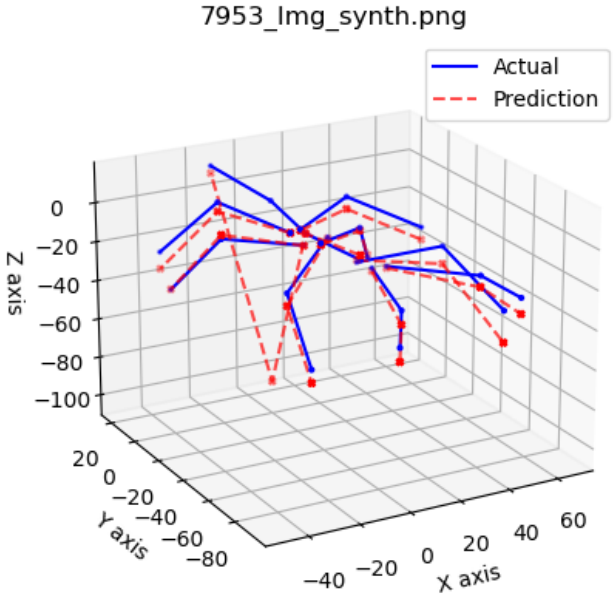
Figure 4.8: Accuracy of the 3D lifting model with varying thresholds.

accuracy of 0.45.

Given that the accuracy is so low, we decide to visualise what the network is the prediction depicted in Figure 4.9a. This prediction was based on an image from the test set to provide more insight. This plot compares the actual and predicted 3D pose of the stick insect. We can see that the pose of the rest of the insect is quite accurate at predicting the correct depth and learning the relationship the points have with each



(a)



(b)

Figure 4.9: (a) is the Actual and predicted 3D lifting keypoint of the model. and (b) is the Actual and predicted 3D lifting keypoint of the model. Where the depth aligned with 1st 2nd keypoint

other. The only issue is that the insect seems to be off centre. We can not centralise this specific sample since the model has learned that that point is always 0. So we realign the depth of this sample to match up with the second keypoint. We show this in Figure 4.9b. Disregarding this issue, we can see that the model can predict the correct depth of the stick insect. Now, this poses the question of why this behaviour occurs in the two-stage model but not in their separate investigations. We believe this might be because the Lifting model is not taking ground truth as input and has a slight variation in input due to the 2D model being used. This lifting model has learned the (0,0,0) is where the 3rd keypoint always but has not learned the relationship that point has with the rest of the keypoints, so when they are lifted, they are not centralised around 0 on the depth axis.

Furthermore, this plot leads us to believe that the lifting network is not outputting the centralised data, so we decide to realign in the same way as before by aligning with the 2nd keypoint. We manage to achieve a much more desirable result of 0.4935 PCK accuracy and the MSE loss of 0.5351. These results are much more along the lines of what we expected, given the results of the individual tests—proving that the two-stage network can output the correct pose of the stick insect.

Given the fixed model, we decided to investigate the per keypoint accuracy of this two-stage pipeline. The results of this are shown in 4.10. We continued to use the same threshold of 5.0 to allow us to make comparisons between the previous investigation on similar data. These results essentially amply the error on the less accurate keypoints from their evaluations. Again, we see that the further the keypoints are from the main body, the lower the accuracy. With the separate case where not the centralised limb has 0 percent accuracy, this was due to our alignment procedure. These results lead us to believe that more tuning is required.

We investigate this outcome by carrying out an experiment where the PCK threshold value is varied, and the accuracy is plotted in Figure 4.11. This plot portrays a similar conclusion as we did in the lifting model investigation. The results are similar and lead us to believe that the threshold value might need to be better tuned to provide an actual accuracy value.

4.3 Real-World Performance

Here, we investigate the performance of our model on real-world data to bring into context the work done to evaluate how well we do on real data. This allows us to see how well our synthetic data trained our model. We have compiled a list of around 13 real-life samples taken from the Heteropterygidae family. We use stick insects from the same family as the Sungaya, so they are not entirely different in terms of appearance. The reason for using the family is that the Sungaya stick bug has very few real-life observation pictures. For example, on inaturalist, we could only find 2 real images of the Sungaya stick bug. A caveat is that these stick insects vary in size, and their limbs differ. These differences can be more significant, with wider bodies and more thorny limbs that may confuse our network.

Using the Sungaya's family will be interesting to evaluate if the more thin stick insect

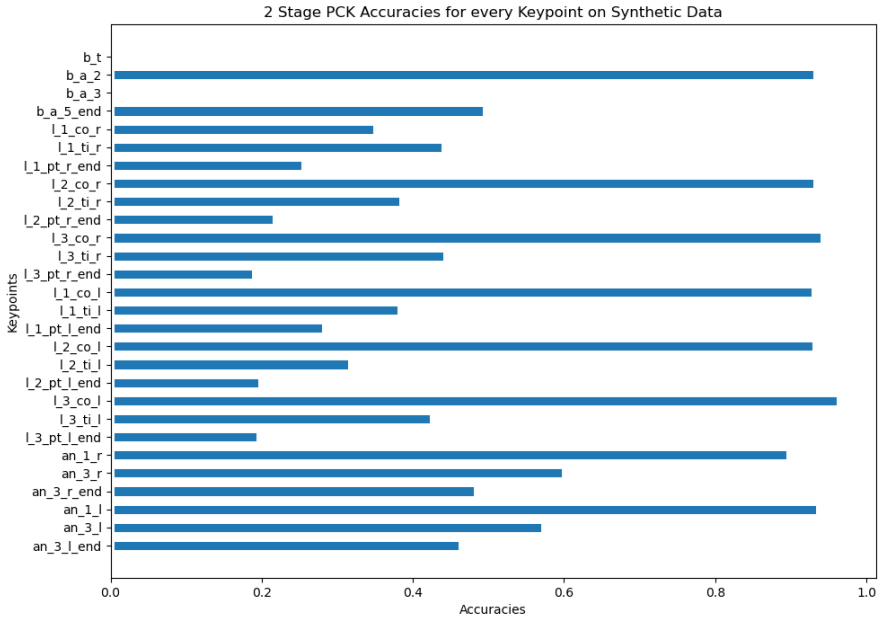


Figure 4.10: Accuracy of each keypoint of the two stage model.

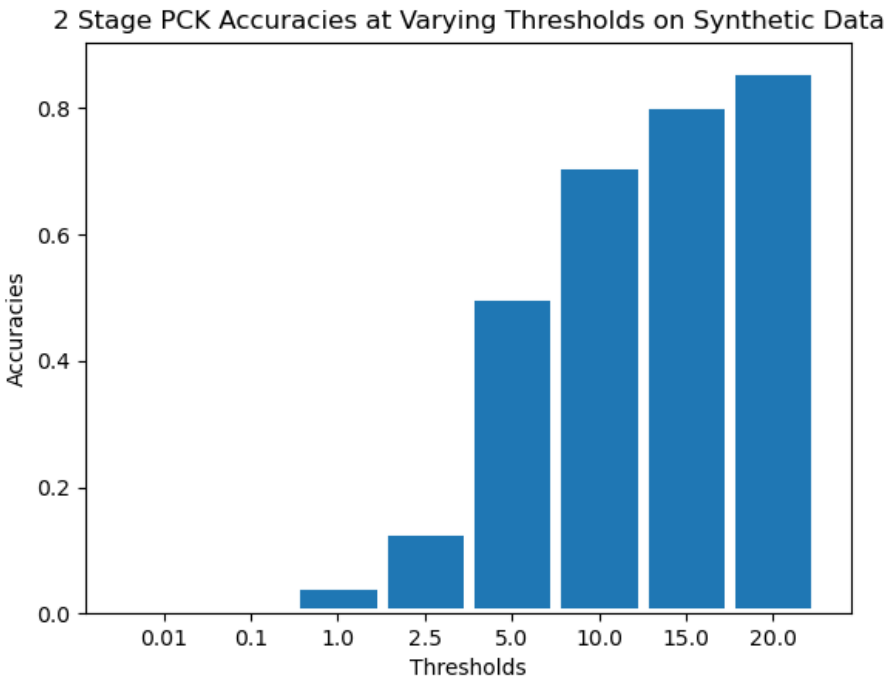


Figure 4.11: Accuracy of each keypoint of the two stage model with varying thresholds. With the alignment fix applied.

the Sungaya still allows for a different species of stick insect to be detected. This allows us to see if at all is the pipeline we have trained on synthetic data will be able to not only identify but track its limbs in 2D and then lift them into 3D to provide a 3D representation of the bug's pose. This real-world performance will be determined in two experiments. These will be 2D Keypoint Estimation Experiment and 3D Lifting Keypoint Experiment. The 2D experiments will be quantitative since we managed to label all the samples gathered manually. The remaining experiments will have to be judged qualitatively. We compare the sample image against the 3D model using the best judgement to determine if it is correct.

4.3.1 2D Keypoint Estimation Investigation

This investigation will allow us to evaluate the performance of our 2D keypoint estimator on real-life data. The model we will be using has been fully trained on 70% of the synthetic Sungaya Inexpectata data and is the best performing epoch we stated before based on validation accuracy. We will also carry out experimentation on this model to allow us to evaluate the model and highlight its upsides and downsides. We will then discuss what could be done differently to improve the model. The test set PCK accuracy reported initially as 0.1402 with a threshold of 0.5. We continue to use this small threshold to keep continuity when comparing results. It is also interesting to see the harshest evaluation since more data is shown about what is less accurate. The loss achieved by this model was 0.00153 using Masked MSE.

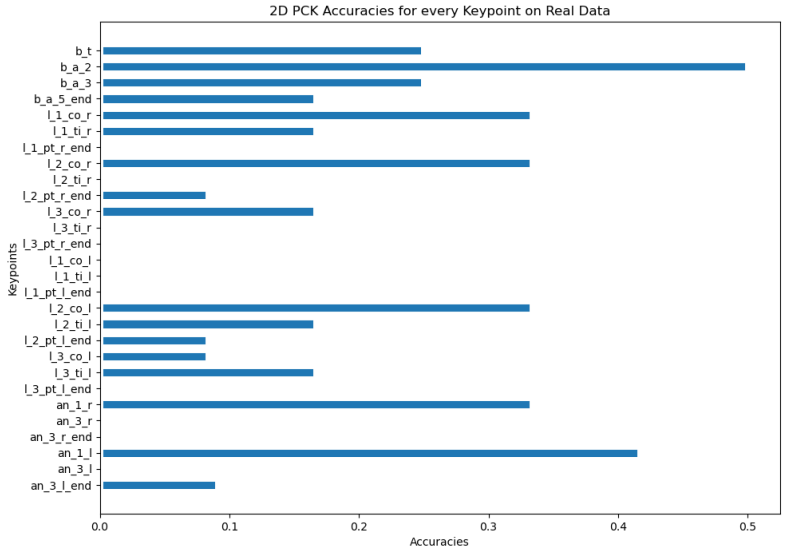


Figure 4.12: Accuracy of the 2D keypoint model on real life data.

The first experiment we carry out the real data is to determine how well the model can predict each 2D keypoints of the real-life insect. Like in previous sections, we plot a bar chart that shows each joint label with its accuracy. This plot is depicted in Figure 4.12. In this Figure, we can see the model is much more accurate at predicting the insect's

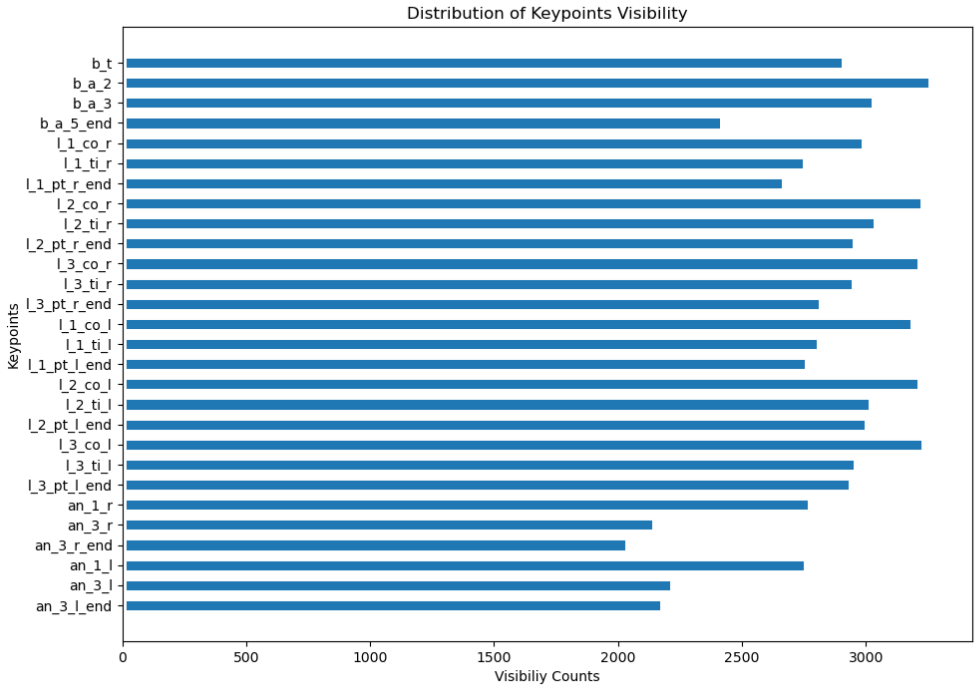
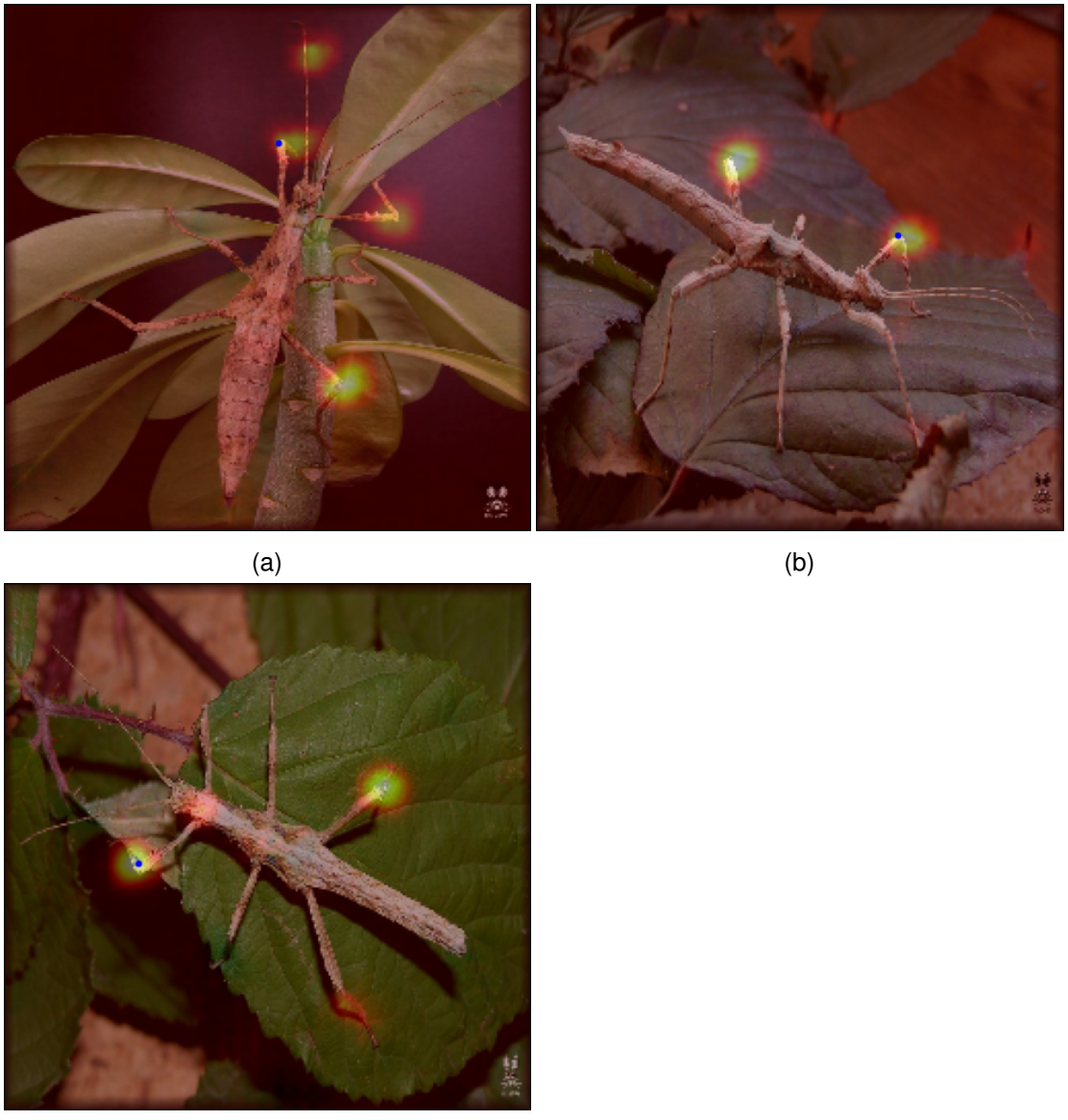


Figure 4.13: This is the distribution of visibility amongst all the keypoints in the Synthetic dataset

body. This varies from what happens in Figure 4.3 in the synthetic evaluation. Where not only are the body predictions accurate, but so are the joints that connect the legs to the body. The more peculiar behaviour is that there is 0 percent accuracy on the 1st left leg in the middle of the bar graph.

Upon further examination, we find that these leg keypoints are not well predicted in the real-life dataset. We investigate the visibility of this limb in the synthetic training set to see if this limb is being miss represented in training so not being able to be accurately predicted. After inspecting this, we find that these keypoints do slightly under-represent the synthetic training. We have collated this data in Figure 4.13. The leg visibility of the Sungaya stick insect is low in the training set and, due to their nature of being thin, makes it hard for our network to distinguish where it is.

We visualise one of the joints causing an issue to see if the model can predict the correct joint. This allows us to understand what the model is doing wrong. We see this image in Figure 4.14a. The image shows us that the model struggles to differentiate between limbs. The model is getting close to the correct since the brighter heatmaps are over the correct part of the leg, just not the right leg. We provide two extra Figures 4.14b & 4.14c where it depicts the same issues. Furthermore, we tried to combat this issue using our random flipping, but that does not seem to have worked. The next course of action would be to improve our model by using a deeper ResNet-50 network. That would hopefully learn these features deeper and be able to distinguish between limbs.



(a) (b)

(c)

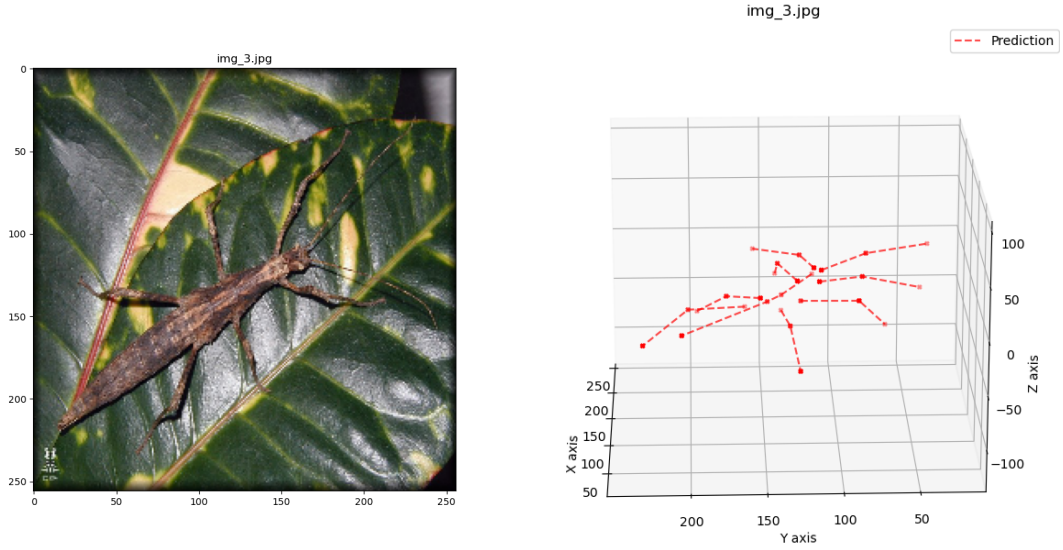
Figure 4.14: Multiple real life 2D predictions of same limb joint

When we compare these results to the synthetic equivalent, the accuracy is lower. Despite this, our current pipeline shows that it can estimate some real-life keypoints quite well, and others have better results to be desired. Now it poses the question on a larger dataset of this family, how well would a fine-tuned model using the ResNet-50 architecture pre-trained on COCO compare.

4.3.2 3D Lifting Keypoint Investigation

This investigation will be carried out using the real datasets 2D GT keypoints. However, we will not be able to provide any quantitative analysis such as PCK accuracy because we cannot label real image's 3D positions manually. So in this section, we will evaluate the 3D model qualitatively by comparing the images and the 3D prediction to determine how good or bad the Lifting network is performing.

The image we will be evaluating the predictions on is depicted in Figure 4.15a because this Figure contains all the keypoints and is most similar to how the synthetic data was generated. We will be evaluating its 3D prediction. This is not easily done in our case since we do not predict X and Y positions in our model. The method we use to visualise our prediction is by using the 2D GT X and Y and plot that with our depth prediction. This representation is known as the camera-centric representation. We visualise the image from Figure 4.14a in a 3D representation in Figure 4.15b.



(a) 2D real image of the insect.

(b) Real life 3D lifting, camera centric prediction.

Analysing the 3D prediction, we can see that the model can quite clearly predict the correct depth. This depicts the pose quite close to the image where the limbs vary in height based on the keypoint correctly. The only error we can see in this plot is that the plot is inverted, but due to an error carried out in the labelling process. If we focus on the depth of each limb, we can see that the 3D model is highly representative of the image's pose. The legs and antenna positions with respect to the body are accurate. The legs are low, precisely how the image depicts it. The lifting model on the real dataset has successfully shown excellent performance.

Chapter 5

Conclusions

5.1 Summary

In this project, we were successful in implementing a 3D pose estimation pipeline composed of a 2D keypoint estimator using a ResNet-18 backend and a 3D lifting network using a simplistic but more modern and effective model, that uses recent advances in computer vision and DNNs. We were able to replicate and adapt these models to work within this pipeline so that we could use our own custom synthetic dataset.

Due to the scarcity and difficulty of obtaining 3D GT data creating a well performing 3D pose estimator was a challenging task. We undertook this task by using a synthetically generated dataset that does not have any real life samples in it. We took this dataset and analysed it to determine how we could augment and change the data to make it suitable for training models in our pipeline. The augmentations we carried out were: image flipping, centralization and standardization. This provided the models with a much more diverse dataset to train on.

We trained these models with 70% of the synthetic dataset and used the remaining 30% for evaluating the epochs to determine the best epochs to use for our final pipeline. The training was successful showing a very constant growth for each model being able to learn by reducing the loss. As a result the validation accuracy of these models were showing significant promise.

Our implementations of these two models were tested and analysed to provide us with an understanding of how well our models performed. We first carried out experimentation on a portion of the synthetic data that was not used for training. We were able to evaluate the performance of both models separately being able to quantitatively measure how well they did, using our GT data in combination with PCK accuracy. As well as this we carried out qualitative analysis to showcase the results of the models on the synthetic dataset. These investigations showed a good performance in predicting the correct data. They also showed that the models have the potential to be tuned further to produce even better results.

We also created a two stage method that combines the two models and tested them on

the synthetic dataset. We carried out both quantitive and qualitative analysis of this method. Initially the results were not desirable but after some quantitive analysis we determined the output did not produce a centralised dataset. We re-ran the experiment and determined that this method was performing much better than in initially thought, allowing us to deeper understand the method's performance.

Finally, we compiled images from the Sungaya family to create a real life dataset. This was to allow us to evaluate our models on real life data since so far it has only interacted with synthetic data. We manually labelled these images with 2D GT to allow us to provide quantitative analysis, and then we were able to carry out experiments but with real life data. After carrying out these experimented we evaluated the performance of the models. This evaluation consisted of a quantitative and qualitative analysis on the 2D model, as well as a qualitative analysis of the 3D model. We were unable to carry out quantitative analysis on the 3D model since we were unable to label the 3D data. From visual inspection we saw that the pipeline was able to predict some data correctly.

The limitations of this pipeline were that it was not able to accurately evaluate on real life data very well, tending to get confused with the smaller limbs such as legs. Our current approach to the 3D lifting was to only predict the Z axis (depth) and use 3D GT to plot the 3D keypoints. It would have been much more preferable for our model to predict the full 3 dimensions instead.

Overall this project has managed to successfully train a 3d pose estimation pipeline on synthetic data. It has shown good performance on synthetic test data. In addition, when using real life test data we were able to predict some limbs to a good standard and others contain confusion with the smaller limbs.

5.2 Future Work

From our experimentation we have seen that training on synthetic data has promising performance since it produces a model that is able to predict some limbs confidently. An idea not really investigated in this project was to spent time tuning the models to find the best parameters. This would hopefully allow us even better results but given the more computation and time we would have attempted this.

Better yet, we could attempt to train a deeper version of the ResNet architecture. It is more capable of learning deeper features such as the thin legs of the Sungaya, which our ResNet-18 struggled to differentiate between. This again would require larger amounts of time and computation to train, but we could provide ourselves with pre-trained weights to even further improve the performance of this model. Our recommended choice of model to train with is the ResNet-50, but deeper networks do exist such as ResNet-101 and ResNet 152. Although these networks would take much longer to train and make predictions, given the increased depth of these model architectures.

Another idea we could explore is the use of video in predictions and use synthetic videos to train the models and evaluate them. This would further improve the capabilities of the pipeline and allow for more data to be given to the models that provides better predictions. This idea does come with some drawbacks such as how to deal with video

data, motion blur, the frequency in which the images are inputted into the network and how fast the network can predict.

Bibliography

[1] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.

[2] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pp. 3686–3693, 2014.

[3] T. D. Pereira, D. E. Aldarondo, L. Willmore, M. Kislin, S. S.-H. Wang, M. Murthy, and J. W. Shaevitz, "Fast animal pose estimation using deep neural networks," *Nature methods*, vol. 16, no. 1, pp. 117–125, 2019.

[4] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, "Learning from synthetic humans," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 109–117, 2017.

[5] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 466–481, 2018.

[6] J. Martinez, R. Hossain, J. Romero, and J. J. Little, "A simple yet effective baseline for 3d human pose estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2640–2649, 2017.

[7] L. Sigal, A. O. Balan, and M. J. Black, "Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion," *International journal of computer vision*, vol. 87, no. 1-2, p. 4, 2010.

[8] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International journal of computer vision*, vol. 61, no. 1, pp. 55–79, 2005.

[9] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *CVPR 2011*, pp. 1297–1304, Ieee, 2011.

[10] "Homecourt ai." https://www.homecourt.ai/. Accessed: 2021-11-01.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[12] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*, pp. 483–499, Springer, 2016.

[13] A. Mathis, P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis, and M. Bethge, "Deeplabcut: markerless pose estimation of user-defined body parts with deep learning," *Nature neuroscience*, vol. 21, no. 9, pp. 1281–1289, 2018.

[14] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele, "Deepercut: A deeper, stronger, and faster multi-person pose estimation model," in *European Conference on Computer Vision*, pp. 34–50, Springer, 2016.

[15] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7291–7299, 2017.

[16] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 172–186, 2019.

[17] S. Li and A. B. Chan, "3d human pose estimation from monocular images with deep convolutional neural network," in *Computer Vision – ACCV 2014* (D. Cremers, I. Reid, H. Saito, and M.-H. Yang, eds.), (Cham), pp. 332–347, Springer International Publishing, 2015.

[18] B. Tekin, I. Katircioglu, M. Salzmann, V. Lepetit, and P. Fua, "Structured prediction of 3d human pose with deep neural networks," *arXiv preprint arXiv:1605.05180*, 2016.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[22] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[23] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, pp. 1325–1339, jul 2014.

[24] F. Plum and D. Labonte, "scant—an open-source platform for the creation of 3d models of arthropods (and other small objects)," *PeerJ*, vol. 9, p. e11155, 2021.

[25] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4724–4732, 2016.