An Efficient Encoder-Decoder Architecture for 3D Hand Pose Estimation

Flavio Schneider
scflavio@student.ethz.ch

Soel Micheletti
msoel@student.ethz.ch

Elrich Groenewald
egroenewald@student.ethz.ch

ABSTRACT

In this paper we present a new lightweight method for estimating 3D hand-keypoints from a monocular image. Our approach uses an efficient-net encoder and a novel decoder architecture that exploits separable convolutions to substantially reduce the number of parameters. Our decoder is trained end-to-end to produce 1D heatmaps that, unlike 2 or 3 dimensional heatmaps, produces outputs that scale linearly with the heatmap size and key-point dimensionality.

1 INTRODUCTION

Hands are one of the main ways that we use to interact with the world and, potentially, they could act as interface between the physical and the virtual world. A sufficiently good estimation of the hand pose would be very useful for many applications in multiple areas such as augmented reality and human computer interaction. For example, it would allow getting rid of hardware such as haptic gloves and input pads, leading to a more natural interaction with the machine and hence improving user experience. Another noteworthy application is the work of Konstantinidis et al. (see [5]), where they proposed an application for sign language recognition, enhancing accessibility in the deaf community. For these reasons, 3D hand pose estimation is a crucial problem in computer vision. Despite being well-studied, it is far from being completely solved, and the ongoing research is exploring diverse directions. In the next section, we will give an overview of the relevant literature for our work, but note that we have considered only a small subset of the plethora of possibilities that have been studied. We refer to [1] for a more exhaustive overview.

In this paper we will work on predicting the hands' configuration (21 points in 3D, each point for a specific joint) from the monocular RGB images using FreiHAND dataset (containing around 100 thousand training samples). This task is challenging for multiple reasons. Firstly, a human hand has more than 20 degrees of freedom. Therefore, a lot of parameters are needed in order to properly model its complexity. Since keeping the number of parameters under control was a priority for us, many of our choices have been done to satisfy this constraint. Secondly, predicting 3D points from a 2D image is an inherently ill-posed task. Even in the best case scenario where there's no occlusion of the hand, depth information of the key-points might be ambiguous. If we extend this problem to any image, e.g. occluded fingers due to different hand rotations and/or hand-object interaction, the problem becomes even harder. It's clear that we, as humans, have a strong prior of hands in general. We know how our hand works, and we can infer reasonably well how a hand is placed, even in case of occlusion. As we will explain in Section 2, a good model should hence be able to learn a prior on hand position, either explicitly or implicitly.

Here we present our deep-learning approach to solve the task. The main contribution of our work is the design of a novel decoder architecture, and we show how to properly combine it with other components from the literature in order to efficiently obtain satisfying predictions. Our implementation uses a lightweight model trained end-to-end. This choice is justified by multiple considerations. Firstly, a model with a small number of parameters usually trains faster. This is a desirable property when searching for an optimal architecture, especially when computing power and time are limited, as in our case. Secondly, one of the main applications of 3D hand-pose estimation is real-time control of virtual objects, which is only possible if the model is lightweight enough to run on consumer hardware such as small laptops or smartphones. Lastly, end-to-end models are much simpler to train and are usually more accurate than complex models trained for the same amount of time.

On a high level, our approach can be summarized as follows:

(1) We performed multiple techniques of data augmentation (rotation, translation, scaling, etc.). This helps to achieve the desirable equivariance property and makes the training data more diverse.
(2) The data is passed on a pre-trained encoder (efficientnet-b4, [10]) in order to get a meaningful representation of the input images. Note that we didn't freeze the weights of the encoder, so they can still slightly change during training.
(3) Inspired by [7], the decoder is trained to learn 63 1D heatmaps: one for each dimension of each point to be predicted. The 1D heatmaps are then mapped to the final 3D representation of the points using the soft-argmax function (a differentiable version of argmax). To learn the 1D heatmaps, we propose an innovative

decoder that combines 2D transposed convolutions, average pooling and 1D convolutions.
(4) The whole architecture (including the weights of the pre-trained encoder) is trained to minimize the mean-squared-error between the output and the true points.

2 RELATED WORK

The approaches for 3D hand pose estimation can be classified in two macro-categories: parametric mesh-based (that regress parameters of a hand model such as MANO, see [9]) and keypoint-based (that regress hand keypoints directly). A crucial distinction is that the former learns a prior of the hand parametrized by some lower-dimensional vector (e.g. [8]). However, there's no clear distinction between models that learn a prior and models trained end-to-end to predict the 3D points. A model trained end-to-end has a reasonable understanding of how the hand might be structured and placed, even with occlusion. Hence, it's very likely that an implicit prior is learned even when training end-to-end.

In this paper, we focus on keypoint architectures. Hence, we are going to restrict to the presentation of some important models in this category. As mentioned in [7], 3D mesh estimation breaks the spatial relationship among pixels in the input and cannot model the uncertainty of the prediction. Motivated by this observation, we focused on models that utilize heatmaps as the target representation. Each value of a heatmap represents the likelihood of the existence of the joint at the corresponding pixel of the input image. The choice of predicting heatmaps have been thoroughly studied in the literature. Here we present two lines of research in this direction: predicting 2D or 1D heatmaps. Note that one could directly learn 3D heatmaps, but we did not consider this option because it would have required a huge amount of parameters and memory.

2D heatmaps to 3D points are models that first estimate 2D heatmaps of the 2D hand-joints on the original image, and then lift this representation to 3D points. The first part might be implemented in different ways, either with an encoder/decoder architecture (e.g. UNet or Hourglass) with an encoder pretrained on Imagenet (e.g. ResNet, HRNet, DeepLab, etc.) with the goal of estimating a tensor of size $k \times s \times s$ from a single RGB image (shape $3 \times h \times w$), where s is the heatmap size and k the number of key-points.

1D heatmaps are models that, instead of learning 2D or 3D heatmaps, learn three 1D heatmaps for each joint. This makes the training and the prediction faster. We mention **I2L-MeshNet** (see [7]) because we follow a very similar approach. They propose an architecture consisting of two main components: PoseNet and MeshNet. PoseNet uses an encoder/decoder architecture to estimate 1D heatmaps of joints from the input image. The output of PoseNet is then fed to MeshNet to predict the 3D mesh. In our case we only need the keypoint prediction and hence worked to improve the PoseNet architecture.

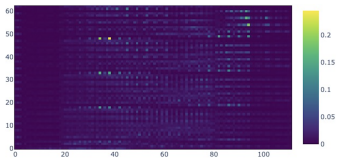

Figure 1: 63 1D heatmaps. Each heatmap represents one dimension of a particular joint. Note that each heatmap is similar to a Gaussian distribution, where the mean is the value of a coordinate of the keypoints to be predicted, and the variance quantifies the uncertainty.

3 METHOD

In this section, we illustrate our architecture. We trained the model for 145 epochs (around 120 hours) with batch size 32 on a single GPU to minimize the mean squared error. We used Adam optimizer and MultiStepLR to schedule an appropriate learning rate, starting at 10^{-4}, then lowering it to 10^{-5} after 100 epochs, and finally to 10^{-6} after 140 epochs.

3.1 Overview

Given the hardware constraints for the project (1 GPU training at a time), we focused not only on performance, but also on keeping our model small. This is one of the reasons why we train our model to learn 1D heatmaps, similar to the one represented in Figure 1. The following subsections present in detail the steps of the pipeline: the encoder, the decoder, details about the data augmentation techniques, and a trick we used in the inference phase.

3.2 Encoder

In the effort to keep our model as lightweight as possible, we used EfficientNet-B4 [10] as the encoder. We used an open source implementation [6] of EfficientNet with weights pretrained on ImageNet. The encoder, similarly to ResNet, lowers the resolution of the image while increasing the number of channel at each step. It takes as input an image of size $224 \times 224 \times 3$ and outputs a tensor of size $7 \times 7 \times 448$, which is then fed to the decoder.

3.3 Decoder

This is the novel part of our architecture. Given the output from the last layer of the encoder, its job is to predict the 63 final 1D heatmaps. The process of how this is achieved is shown in Figure 2. On the left we have the output of the encoder and on the right the prediction for the keypoints.

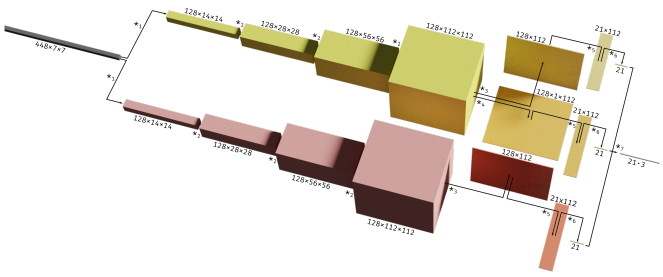

Figure 2: Decoder diagram.

In order to properly interpret the figure we need to list the operations we used:

(1) `ConvTranspose2d(k=3, s=2, p=1, sep = False)`, where k denotes the kernel size, s the stride and p the padding.
(2) `ConvTranspose2d(k=3, s=2, p=1, sep = True)`, similar as in point 1. However, we use a separable convolution, which means that we use a "matrix factorization" approach in order to keep the number of parameters substantially lower.
(3) `Mean(dim=2)`.
(4) `Mean(dim=3)`.
(5) `Conv1d(k=1, s=1, p=0)`.
(6) `SoftArgmax`, a differentiable version of argmax.
(7) `Concatenation`

The choice of parameters in the transposed convolutions makes sure that the resolution of the tensor is doubled at each step (similar to the UNet decoder). We double the resolution until we get to 112×112, which will determine the heatmap size. The yellow part of the decoder handles the xy-axis, while the red part of the decoder handles the z-axis. We found that using a single decoding block for xyz as in [7] produced results with low accuracy on the z-axis, hence we decided to split the depth prediction to a separate block. The xy feature block is then averaged on both the second and third dimension (the first dimension is the number of channels), while the z feature block is averaged only on the second dimension. Taking this mean has the same effect as average pooling, where the goal is to obtain tensors of size $C \times 112$ ($C = 128$) on which we can apply 1D convolutions to get 23 1-dimensional heatmaps, one for each axis. Applying soft-argmax then gives 3 vectors of dimension 21, which are concatenated to obtain a final output of shape 21×3.

We have the choice of a few hyperparameters in this decoder, the first is C which we choose to be 128, increasing this number would increase the number of kernels applied at each transposed convolution, and hence the number of parameters. We found 128 to be a good balance between accuracy and model complexity. The second hyperparameter is N which we choose to be 4, increasing this would double the resolution for each increment and hence produce a larger heatmap (note that on the diagram N is the number of blocks). The relationship with the heatmaps size is $H = 7 \cdot 2^N$.

3.4 Data Augmentation

We applied data augmentation during training to help with generalization by increasing diversity in the training data. For the final submission, we used random rotations of up to 180 degrees in either direction. We also tried flips, color jitter, scale and translation, but all models with those augmentations performed worse.

The augmentations are applied directly to the input images. To apply the same augmentations to the corresponding 3D point labels, the 3D points are first mapped to 2D image points, using the camera matrix. The augmentations are applied to the 2D image points, after which the points are mapped back to 3D camera coordinates, with their original z values.

Data augmentation was useful not only in the training phase, but also for inference. In an attempt to obtain more robust test predictions, we averaged the predictions over different augmented versions of each input image. Specifically, for each image we predicted the points for 10 different rotations, which were then rotated back with the corresponding inverse rotation and averaged.

This technique improved our final performance for each model we applied it to. For our submitted model, it improves performance from 6.72 to 6.24 in the public leaderboard. We found that more rotations did not substantially improve the accuracy.

4 EVALUATION

As already stated, the main idea of our model is to combine a pre-trained encoder with the idea of [7] of predicting 1D heatmaps in order to limit the number of parameters. We developed a novel decoder that exploits separable convolutions to further increase the number of parameters. We would like to justify the choice of the encoder (which, again, has an influence on the number of parameters) and spend a few words on the importance of data augmentation.

Before choosing EfficientNet-B4, we tested multiple encoders. We experienced that EfficientNet achieves better performance than ResNet (see [4]) and, within the class of EfficientNet encoders, EfficientNet-B4 is well positioned in the trade-off between performance and number of parameters. Our final model has $19.5M$ parameters which is acceptable with our computing resources. We mention that EfficientNet-B0 had a worse performance (8.03 on the public leaderboard

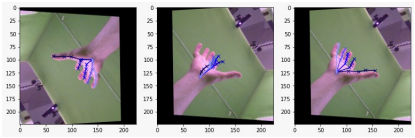

Figure 3: Visualization of the impact of data augmentation. On the left we have the prediction of the original (not rotated) image. The images in the middle and on the right show the predictions obtained by the same model on the rotated image. If trained without data augmentation (middle) the model is not equivariant. Data augmentation (right) improves the predictions and makes the model equivariant to rotation. Note that the predictions on the left and on the right are not the same, this is why we average different rotations at inference.

after 24 hours of training, without rotation averaging during inference), but only $5.4M$ parameters. This might be more appropriate for real-time applications.

During the development of our architecture, it turned out that data augmentation is a crucial ingredient to make the model *equivariant* (i.e. keypoints are rotated with the image). Figure 3 illustrates the positive impact of data augmentation on our predictions: in the middle the predictions of the model trained without data augmentation are shown, on the right we observe that the predictions obtained by the same model trained with data augmentation are much more accurate. Note that the prediction on the right image are a rotated version of the predictions on the original image on the left, which implies that the model trained with data augmentation is equivariant.

5 DISCUSSION

As often happens with deep learning projects, we tried multiple promising approaches that didn't work as expected. Here, we mention three of them.

The first failed method is about a potential approach for regularization. We observe that the 1D distributions represented by the heatmaps in Figure 1 seem a reasonable approximation of a Gaussian distribution. However, in order to constrain the model to learn this even better, we experimented with a regularization technique. We tried to include in the loss function a term designed to minimize the Jensen-Shannon divergence between the heatmaps and a Gaussian distribution with variance of two. However, this did not improve the performance of the model. An explanation for this could be that the heatmaps learned without regularization already resemble a Gaussian distribution. Moreover, this confirms a trend that we often observed during this project: simpler models tend to perform better. This could be due to our limited computational resources and training time we considered (between 24 and 100 hours on a single GPU).

Second, we tried to extract two different representations from the encoder (the first one from an intermediate channel, the second one from the final layer). The intuition on why this could have been useful is that deeper layers, despite being able to learn a larger field of view, might lose relevant detailed information that can be recovered by considering the representation in intermediate layers. However, our experiments showed that this is not always the case, hence we decided to consider only the deepest representation computed by the encoder.

Finally, we mention our failed approach with convolutional graph neural networks (GNN). In fact, graphs are an appropriate tool to model human hand problems because they can be used to learn spatial relationships between different joints. Inspired by [2], we tried an architecture composed by a pre-trained encoder, a convolutional GNN and an adaptive graph U-Net ([3]). Given an input image, the encoder returned a tensor of size 2048. This tensor (together with a 2D initial estimate for the keypoint) was then fed to a GNN with 21 nodes (one for each joint). The GNN learns how to use adjacency information to modify the 2D coordinates of the keypoints conditioned on the output of the encoder and the initial prediction of the 2D coordinates. Finally, a Graph U-Net architecture converts the 2D keypoints to the final 3D output using a series of graph convolutions, poolings, and unpoolings. Similar to the classical U-Net architecture, the dimension of the graph is first gradually reduced and then increased to the original size of 21 nodes. The choice of first learning 2D heatmaps and then mapping them to the final output via the U-Net neural network is justified by [2], where they showed that this is more accurate than combining both steps in the convolutional GNN. In [2] they mention that their model is well-suited for objects that are of similar size or shape to those seen in the dataset during training, but it might not generalize well to all categories of object shapes. This could be a possible reason for why this approach failed in our case, but we are not sure whether there are other limitations that make it inaccurate.

6 CONCLUSION

We presented an encoder-decoder architecture for 3D hand pose estimation from 3D monocular RGB images. Using a pre-trained encoder, a novel decoder to efficiently predict 1D heatmaps and data augmentation techniques both in training and inference phase, we developed a new lightweight model with $19.5 \cdot 10^6$ parameters that achieves satisfying performance (6.29 in the public leaderboard) with reasonable computing resources.

REFERENCES

[1] Theocharis Chatzis, Andreas Stergioulas, Dimitrios Konstantinidis, Kosmas Dimitropoulos, and Petros Daras. 2020. A Comprehensive Study on Deep Learning-Based 3D Hand Pose Estimation Methods. *Applied Sciences* 10, 19 (2020), 6850.

[2] Bardia Doosti, Shujon Naha, Majid Mirbagheri, and David J Crandall. 2020. Hope-net: A graph-based model for hand-object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6608–6617.

[3] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *international conference on machine learning*. PMLR, 2083–2092.

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

[5] Dimitrios Konstantinidis, Kosmas Dimitropoulos, and Petros Daras. 2018. SIGN LANGUAGE RECOGNITION BASED ON HAND AND BODY SKELETAL DATA. In *2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*. 1–4. https://doi.org/10.1109/3DTV.2018.8478467

[6] Luke Melas-Kyriazi. 2020. EfficientNet-PyTorch. https://github.com/lukemelas/EfficientNet-PyTorch.

[7] Gyeongsik Moon and Kyoung Mu Lee. 2020. I2L-MeshNet: Image-to-lixel prediction network for accurate 3D human pose and mesh estimation from a single RGB image. *arXiv preprint arXiv:2008.03713* (2020).

[8] Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. 2015. Hands deep in deep learning for hand pose estimation. *arXiv preprint arXiv:1502.06807* (2015).

[9] Javier Romero, Dimitrios Tzionas, and Michael J Black. 2017. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics (ToG)* 36, 6 (2017), 1–17.

[10] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.