Human Pose Estimation on LSP Dataset

Giacomo Virginio
giacomo.virginio@studenti.unipd.it

Mikhail Kolobov
mikhail.kolobov@studenti.unipd.it

Abstract

This report presents an investigation into various model architectures for human pose estimation applied to the LSP dataset[1], building upon the work of Xiao et al.[2]. We modified and evaluated different baseline models, aiming to identify the impacts of these modifications on model performance. Our approach involved altering existing architectures, testing new configurations, and comparing obtained results. The primary focus was to determine how these changes affect accuracy. Our findings contribute to a better understanding of the trade-offs between model complexity and performance in pose estimation. The report includes a comparative analysis of the models tested, offering insights for future research in this area. All code and results from our experiments are made available[1] *for further academic scrutiny and development.*

1. Introduction

This research examines the field of human pose estimation, a critical area in computer vision with wide-ranging applications. Our work is inspired by the study of Xiao et al.[2], which proposed simple yet effective baseline methods for pose estimation. We focus on experimenting with these baseline models, modifying them to suit constrained computational resources while evaluating their performance.

Our primary approach involved adapting a baseline model, particularly the pretrained ResNet[3], through various modifications. These included unfreezing a limited number of layers and resizing input images, aimed at managing computational demands. However, these modifications resulted in a trade-off between computational efficiency and accuracy.

We further experimented with other lightweight baseline architectures like EfficientNet[4] and ShuffleNet[5]. Each exhibited distinct performance characteristics, with ShuffleNet showing a tendency to overfit.

[1]https://github.com/JackdiQuadri97/HumanPoseEstimationLSP

We also addressed a specific issue in the original algorithm we found during the training phase, related to the loss function interacting with the heatmaps, enhancing the model's learning capability.

The experiments were conducted using the Leeds Sports Pose (LSP) dataset[1], focusing on single-person pose estimation. Unlike the original study, we did not employ extensive data augmentation or bounding box predictions due to our limited computational resources. Furthermore, LSP doesn't include ground truth for bounding boxes, nor data about the visibility of the joints, so we didn't include those in our code.

In summary, our work presents a practical exploration of pose estimation models under computational constraints, offering insights into the trade-offs and performance dynamics of various architectural modifications.

2. Related Work

The foundational paper by Xiao et al.[2] that underpins our study makes significant strides in human pose estimation and tracking. It simplifies the complex architectures typically seen in this field, offering a more accessible approach for analysis and comparison. This work utilizes a backbone network, specifically ResNet, with added deconvolutional layers for heatmap generation, achieving state-of-the-art results on challenging benchmarks like COCO and PoseTrack.

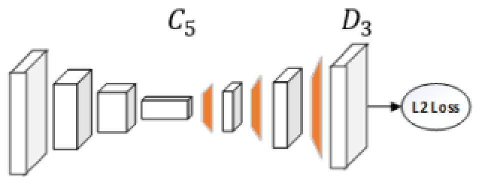

Figure 1. Model architecture from Xiao et al.[2]

Xiao et al.'s[2] work stands out for its simplicity and effectiveness in an area often dominated by more intricate models. They compare their baseline method with prominent architectures like Hourglass[6] and CPN[7], noting their method's comparative ease and efficiency. Their approach differs in generating high-resolution feature maps and their use of upsampling combined with convolutional parameters in deconvolutional layers, a simpler alternative to the methods used in other advanced models.

This paper sets a new benchmark in pose estimation and tracking, inspiring our project to explore variations of these baseline models under different computational constraints. Our goal is to understand the trade-offs involved in modifying these efficient and straightforward baseline models in terms of computational resources and accuracy.

3. Dataset

For our project, we utilized the Leeds Sports Pose (LSP) dataset, a widely recognized dataset in the field of human pose estimation. This dataset comprises images of athletes engaged in various sports activities, providing a diverse range of human poses for analysis. The LSP dataset is particularly suited for single-person pose estimation, aligning with our project's focus.

The dataset includes thousands of annotated images (1000 for training and 1000 for testing), each labeled with key joint positions, making it an invaluable resource for training and testing pose estimation models. We worked with this dataset's full extent, leveraging its variety to assess our models' performance across a broad spectrum of human poses.

Given the variety of images in the LSP dataset, we undertook the necessary preprocessing steps. This involved resizing images to fit the input requirements of our models and normalizing the pixel values for consistency. We did not, however, employ extensive data augmentation techniques like those used in the original study by Xiao et al.[2], due to our computational limitations. This decision allowed us to evaluate the models' performance in a more constrained yet realistic scenario, typical of many real-world applications where computational resources are limited.

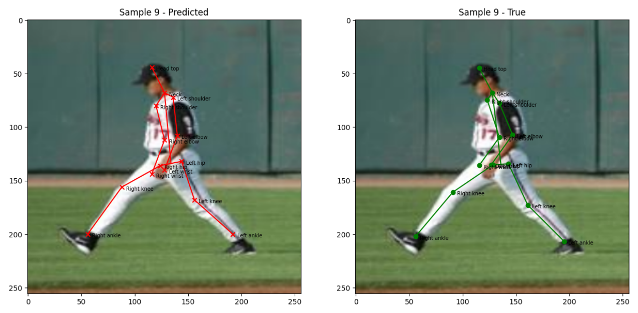

Figure 2. Example of joint prediction and joint ground truth

4. Method

Our approach to solving the challenges in human pose estimation centred around modifying and evaluating baseline models, primarily the architecture presented by Xiao et al.[2]. We chose this direction because of the simplicity and effectiveness of the baseline model, making it ideal for adaptation under computational constraints. We introduced two main changes to the baseline method proposed by Xiao et al.[2].

The first change regards the construction of the backbone component of the model: we decided to directly load the pretrained model as a backbone, as opposed to building the structure of the model from scratch and then using the pretrained weights. This approach has the advantage of allowing us to easily change the backbone model, without having to create the structure of any model we wanted to apply, essentially having an almost "plug-and-play" structure; however this approach also has some important limitations, in fact it isn't possible to use low-level features, but only the final output of the backbone, which prohibited us from testing by adding some modules that need lower level features (for example the WASP module proposed in UniPose by Artacho et al. [8]).

The second change regards instead the heatmap created starting from the ground truths, which is used in the loss function. During our initials test, in which we used the standard parameters proposed by Xiao et al.[2], the model was not learning the position of the joints despite the loss getting reduced at each epoch, by exploring the output heatmap produced by the model we discovered that the model was learning to minimize the loss function mainly by bringing all the values in the heatmap to 0.

To solve this issue we initially tested two different solutions. The first solution was to simply enlarge the Gaussian produced by the ground truth, by changing the *sigma* parameter, which indeed provided us with a model that was actually learning joint positions, however, the model incurred a lot of overfitting, in particular, it was very imprecise when joints appeared in an "uncommon" position (e.g. ankles not being at the bottom of the image), an effect which might have been solved through data augmentation. The second solution was to instead multiply the values of the Gaussian by a large factor, this method proved more effective than the first and didn't present the same issues, but it made the resulting outputs very different from a usual heatmap, with the graphical representations being hardly readable.

We, therefore, decided to use the second solution with a slight modification: we made the multiplication factor inversely dependent on the training epoch, this approach allowed the model to quickly learn a general position of the joints in the first epochs, while at the same time approaching the results offered by Xiao et al.[2] and showing a readable graphical representation 3; this approach however also has

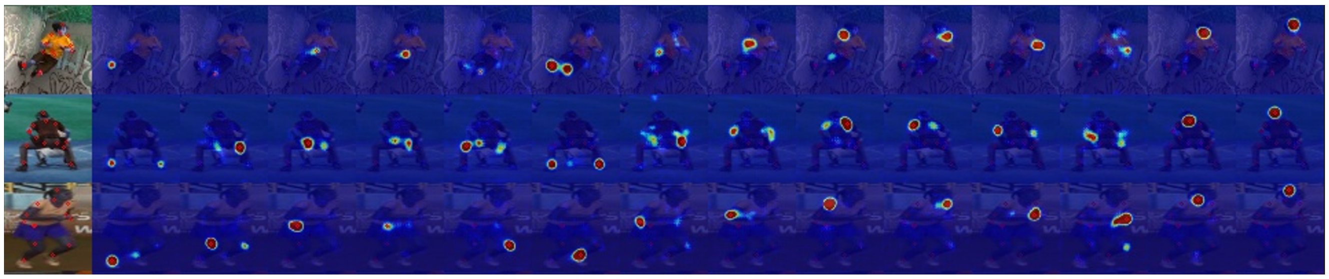
Figure 3. Example of output heatmaps from test dataset

its downside, in fact the target changes at each step in a way that naturally shrinks the loss, therefore it is not possible to compare loss at different epochs.

We used the same backbone model structure used by Xiao et al.[2], Resnet, however thanks to the changes made we were able to not only test different versions of ResNet, but also very different backbone structures like EfficientNet and ShuffleNet. The backbone models used were chosen because they respectively had a comparable number of parameters while obtaining a better performance in classification tasks, and a lower amount of parameters while obtaining a comparable performance in classification tasks[9].

It is important to mention that all the training and testing were performed using the free plan of Google Colab, which uses an NVIDIA T4 GPU, and limits GPU availability if a user uses resources for too much time[10]. Because of this, we decided to focus our analysis on models that are not too computationally expensive; as a reference, the baseline model takes roughly 4 hours to train for 20 epochs.

5. Experiments

Our experiments were planned to test the impact of various single changes to the model we chose as a baseline: a model that works on images of size 256x256, uses the ResNet50 backbone, then uses 3 deconvolution layers, with kernels of size 4 and 256 filters for each layer (hence producing heatmaps of size 64x64), employing a final convolution layers with a kernel of size 3, and that is trained while freezing all backbone layers except the last. Our ablation study involved all the specifics mentioned.

Table 1 shows the models tested and, for each, the backbone used, the resulting output heatmap size (which is determined by the input image size and number of deconvolution layers), the number of parameters and learnable parameters, and finally the PCK accuracy obtained at the final epoch of training and then on the testing. In the table, the data shown in bold have a non-negligible difference from the baseline.

Figure 4 shows the PCK accuracy obtained by the models during the training set at each epoch of training.

From the graph, we can infer several key points about the model performances. All models seem to improve rapidly within the first 5 epochs, indicating a swift initial learning phase. This is typical in training neural networks where large gains are often seen early on. However, after around 10 epochs, the rate of improvement in PCK slows down for most models, and they begin to plateau. This suggests that the models are approaching their maximum performance given the current architecture and hyperparameters.

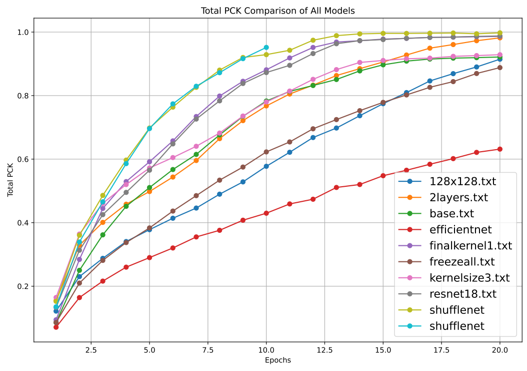

Figure 4. Models' accuracy on the train set by epoch

Table 2 shows for each model the Total PCK as well as the PCK on each joint for the test dataset, no model shows a clear difference in the accuracy in predicting left or right joints, therefore we paired the two measurements together and reported the average.

We will now go more in depth with each model tested and compare them with the baseline:

- **kernelsize3** uses kernels of size 3, compared to the kernels of size 4 used in the backbone, it results in a non-negligible smaller number of parameters and learnable parameters that make the model faster, while not affecting the model's effectiveness both in general and for each single joint, which is consistent with the results obtained by Xiao et al.[2] .

- **finalkernel1** uses a kernel of size 1 in the final step of the model, compared to the kernel of size 3 used

Model	Backbone	Heatmap size	Param.	Learnable param.	Train PCK	Test PCK
base	resnet50	64 x 64	34.0M	25.5M	92.2	57.1
kernelsize3	resnet50	64 x 64	**29.4M**	**20.9M**	92.9	57.0
finalkernel1	resnet50	64 x 64	34.0M	25.5M	**98.6**	**60.0**
128x128	resnet50	**32 x 32**	34.0M	25.5M	91.5	**43.6**
2layers	resnet50	**32 x 32**	33.0M	24.4M	**98.2**	**55.1**
freezeall	resnet50	64 x 64	34.0M	**10.5M**	**88.8**	**35.4**
resnet18	**resnet18**	64 x 64	**15.4M**	**12.6M**	**98.8**	56.9
efficientnet	**EfficientNet_V2_S**	64 x 64	**27.6M**	**7.7M**	**63.2**	**34.1**
shuffle	**ShuffleNet_V2_X2_0**	64 x 64	**15.9M**	**12.5M**	**99.8**	**37.3**
shuffle10	**ShuffleNet_V2_X2_0**	64 x 64	**15.9M**	**12.5M**	95.2	**36.5**

Table 1. Models tested

Model	Total PCK	Ankle	Knee	Hip	Wrist	Elbow	Shoulder	Neck	Head Top
base	57.1	46.8	**59.6**	58.9	40.1	49.3	61.7	84.1	82.6
kernelsize3	57.0	47.3	58.4	58.4	41.2	48.5	61.3	84.6	83.0
finalkernel1	**60.0**	**57.9**	**59.6**	**61.2**	**42.0**	**50.4**	**63.0**	**85.4**	**84.8**
128x128	43.6	47.8	40.7	45.7	25.3	33.0	46.2	69.3	63.6
2layers	55.1	54.0	53.6	55.6	40.0	43.7	57.3	82.5	81.2
freezeall	35.4	37.6	30.8	33.0	21.5	23.1	38.0	66.3	60.5
resnet18	56.9	56.7	57.0	60.8	35.0	44.6	62.6	83.2	80.4
efficientnet	34.1	33.5	31.7	33.0	19.3	24.0	36.4	61.9	59.2
shuffle	37.3	37.7	33.5	36.6	21.8	24.7	39.8	67.1	66.8
shuffle10	36.5	37.7	31.5	34.9	20.5	24.3	39.6	68.4	65.9

Table 2. Percentage of correct keypoints on the test set

by the baseline, the number of parameters and learnable parameters is almost the same, however, it learns faster and results in better performance on each joint, *finalkernel1* is the best-performing model we tested in our study.

- **128x128** changes the size of the image in input (256x256 in the base model), it has a big impact on the time it takes train and test each batch, while however not impacting much the size of the architecture, its training accuracy is comparable to the baseline but the test accuracy is much worse, it shows slower learning per epoch and it is still learning after 20 epochs, therefore, it might still have great improvement.

- **2layers** has only 2 layers in the deconvolution part of the model, compared to the 3 in our baseline, however, this doesn't impact much the number of parameters or learnable parameters, the training accuracy per epoch is very similar to the baseline but it shows less signs of slowing down in the latest epochs, the testing accuracy is also close to our baseline, but a difference of 2 percentage points is not negligible.

- **freezeall** uses the same architecture as the baseline but freezes also the last layer of the backbone, therefore resulting in a model with the same amount of parameters, but a vastly lower amount of learnable parameters. Similarly to the *128x128* model, it has a slow learning per epoch and it is still learning after 20 therefore it might still have a decent improvement, however, its results after 20 epochs are far worse than the baseline.

- **resnet18** changes the backbone model to ResNet18, resulting in a model with close to half the baseline's parameters and learnable parameters, its learning speed and training accuracies are very similar to *finalkernel1*, but doesn't show the same improvements on test results, which however are very similar to the ones obtained by the baseline model while being a much lighter model.

- **efficientnet** uses EfficientNet_V2_S as a backbone, the smallest EfficientNet_V2 model, it has fewer parameters and significantly fewer learnable parameters, the difference between the change in the number of parameters and learnable parameters is due to the structure of the backbone and the decision on the cut-off for freezing the backbone layers, the model is by far the slowest learner overall but the fact that it only reaches an accuracy of 63% means that the model still has a great potential to improve by learning for more epochs, which might improve its results on the test set, which are the worse we obtained in our tests; it would also be interesting to test different freezing cut-offs, allowing the model to have more learnable parameters, which probably is a limitation the model we tested had.

- **shuffle** uses ShuffleNet_V2_X2_0 as a backbone, the biggest ShuffleNet_V2 model, the model is similar in size to *resnet18*, it shows the fastest learning, which however brings it to a plateau with accuracies around 100% after only 15 epochs, the model shows an almost perfect training accuracy but has bad results on the test set.

- **shuffle10** uses the same architecture of *shuffle*, however, we stopped its training after 10 epochs (which is the only model we tested with a number of training epochs different from 20), as we wanted to understand if the low test accuracy shown by *shuffle* was due to overfitting, it doesn't however seem the case as the two models have very similar test accuracy.

The results, as illustrated by the performance curves, suggest that these variations have a considerable impact on performance. For instance, the "finalkernel1" model, which boasts the highest overall PCK at 60.0%, demonstrates the potential benefits of carefully calibrated architectural adjustments, as it excels in capturing the nuances of various joints in the human pose.

In general, models exhibit higher accuracy when predicting keypoints for the neck and head top. As for the other parts of the body, the accuracy of predicting joint positions declines as the anatomical distance from the central torso region increases, particularly for joints with greater ranges of motion such as the wrists. These results are consistent with results obtained on other datasets and by other architectures in pose estimation tasks [11].

Generally, the change in total performance compared to the base model reflects the performance of every single joint, with the only particular case being the ankle, which has far worse accuracy than the average only for the baseline and the kernelsize3 models.

6. Conclusion

To summarize, the interplay between model architecture, training strategy, and computational resources is crucial in the development of efficient and accurate human pose estimation models. The baseline model provided a strong foundation, yet the nuanced alterations demonstrated both the potential and the limitations inherent in optimizing pose estimation models within computational constraints.

The experiments that we have conducted suggest that the most efficient model seems to be a model that uses resnet18 as a backbone, the kernel size of deconvolution equal to 3 and the kernel size in the final layer equal to 1.

Moving forward, research should pivot towards enhancing computational methods to refine efficiency without compromising on accuracy. Further exploration into the adaptability of various architectures could provide a deeper understanding and improvements in the prediction of challenging joints, which we found are the most mobile ones. This project has laid the groundwork for future innovations in the field, emphasizing the need for a balanced approach to model architecture and training methodology in human pose estimation.

References

[1] S. Johnson and M. Everingham, "Clustered pose and nonlinear appearance models for human pose estimation," in *Proceedings of the British Machine Vision Conference*, pp. 12.1–12.11, BMVA Press, 2010. doi:10.5244/C.24.12.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[4] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," 2021.

[5] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," 2018.

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

[7] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, "Cascaded pyramid network for multi-person pose estimation," 2017.

[8] B. Artacho and A. Savakis, "Unipose: Unified human pose estimation in single images and videos," 2020.

[9] "Pytorch pretrained models." `https://pytorch.org/vision/stable/models.html`.

[10] "Google colab gpu usage limits." `https://research.google.com/colaboratory/faq.html#resource-limits`.

[11] "Mpii dataset results." `http://human-pose.mpi-inf.mpg.de/#results`.