Western Norway University of Applied Sciences

Project report in DAT255 – Deep Learning Engineering

Bike Fitting Web Application using Deep Learning

25.04.2025

GitHub Repository: https://github.com/williamsaether/BikeFit

William Sæther (candidate number 669810)

Ole-Mathias Midtun Egeland (candidate number 669807)

I confirm that the work is self-prepared and that references/source references to all sources used in the work are provided, cf. Regulation relating to academic studies and examinations at the Western Norway University of Applied Sciences (HVL), §10.

Problem description

As two cyclists, we have spent countless hours trying to find the optimal position when riding our bikes. Achieving this requires repeated adjustment and testing on the bike, and sometimes it comes down to a few millimetres. This process is time-consuming and if you want help from a professional it can be very expensive. Therefore, we want to develop a deep-learning tool that helps find a proper starting position on the bike, saving both time and money.

While deep learning can't replace the professional fitting, it can provide a valuable guidance without having to spend money on the expensive services. There are existing solutions, such as **MyVeloFit**, an online video-based bike fitting service that is both innovative and effective. Another option is **APIIR**, which offers a similar service. However, both of these require users to record and submit videos – a process that is not always practical without an indoor trainer or bike stand. These are also paid services: MyVeloFit offers a basic free version with limited recommendations, but access to full analysis starts at $35 (MyVeloFit, 2023). APIIR charges €39.99 or more per fit (APIIR, no date). These are certainly more affordable than an in-person fitting, but they still present a barrier for some users.

Our project aims to offer an image-based alternative bike fit using deep learning. By predicting joint heatmaps from a single side-view image, we can extract key joint positions such as knee, ankle, hip, shoulder, and elbow. From these, we calculate relevant angles and provide feedback on posture and potential adjustments like saddle height or handlebar reach.

Deep learning is particularly suited for this task, as it enables robust pose estimation even in varied lightning conditions, camera angles, and varying appearances of both the user and the background. Traditional rule-based methods would likely fail in such a task. With the help of convolutional neural networks (CNNs), our model learns visual features that make keypoint detection possible without manual feature engineering.

Unlike existing services, our solution will be deployed in a lightweight web application. Users can upload a single image and get a quick assessment directly in the browser. For this research project, the service is naturally free, with a temporary limit of five recommendations per hour to prevent misuse. You can on the other hand calculate angles and compare to recommended ranges as much as you'd like. In a commercial version, the service would offer a freemium model where users get limited free assessments, with options to upgrade for more advanced features or higher usage limits.

By reducing the cost barrier and removing the need for specialized equipment like indoor trainers, we hope to make bike fitting accessible to a much broader audience – especially beginners with suboptimal setups.

Data

For this project we tested out two datasets; MPII Human Pose Dataset and COCO – Common Objects in Context. The MPII dataset includes around 25K images with over 40K annotated body joints (Andriluka *et al.*, 2014). It is widely used in pose estimation tasks and contains high-quality annotations with detailed joint labels. However, only a small portion of the dataset includes cyclists, which made it less targeted for our application. Despite this, MPII was a good starting point for our project.

The second dataset we explored was the COCO dataset, which is similar to the MPII dataset, but with over 330K images (250K of them with joints) compared to 25K (Lin *et al.*, 2015). However, most of the images does not include all 17 joints, which posed a significant challenge for training a model on this data.

One of the biggest challenges we encountered when it comes to reformatting was with the MPII dataset, which is distributed as a single '.mat' (MATLAB) file. This file includes annotations, activities and other information, and the structure was deeply nested and had a lot of inconsistent samples. Writing a parser to convert it into a usable format took a considerable amount of time and was not easy work. We also had to preprocess the data so that each sample was the same size. Both the MPII dataset and the COCO dataset included information about each annotation to centre the image around the subject using bounding boxes or coordinates with a scale. When doing these changes to the annotation/image it was also necessary to adjust the joints accordingly. We also had to generate a heatmap for each joint, which we used ChatGPT to generate a function for, as this math is not the simplest.

While the MPII dataset was our initial focus due to its popularity in pose estimation research and detailed annotations, we encountered several limitations after training some models on the data. First, although the quality of annotations was high, the dataset lacked sufficient representation of cyclists, which are central to our use case. Second, some poses were highly varied or ambiguous, and several samples had missing or occluded joints, which negatively affected training quality.

Despite the effort we put into formatting and parsing MPII, model performance on this dataset was underwhelming. Heatmap predictions were often inaccurate, especially for lower body joints such as knees and ankles – critical for calculating bike fitting angles. In hindsight, it could have been possible to mitigate some of the issues we were encountering, but we didn't see these possibilities at the time.

As a result, we shifted our focus to the COCO dataset. As stated earlier, many of the images have incomplete joint labels, meaning not all 17 joints were visible or annotated at all. Despite this, this dataset gave us better practical results. One reason might be that COCO has a lot more images and annotations compared to MPII, giving us a wider variety

of activities and perspectives. To work around COCO's missing joints, we filtered out samples that had fewer than 3 joints after we removed the 5 joints placed on the persons head as we didn't need these joints at all for the task at hand.

The switch to COCO resulted in noticeably improved model predictions. Visualizations showed that the models trained on COCO were more consistent in locating elbows, wrists, knees and ankles in particularly.

Input and Heatmap Resolutions

For the MPII dataset, both the input images and heatmaps were 128x128. This resolution was chosen to reduce training time and GPU memory usage during initial experiments. It provided a sufficient level of detail for learning joint locations, especially since the subjects in MPII are usually centred and relatively large within the image frame.

When switching to the COCO dataset, we increased the input resolution to 256x256 and generated 64x64 heatmaps. The reason behind this adjustment was because we wanted to see if a larger resolution would give better results, which it did. We also realized that there was no need for the heatmaps to be the same size as the images. This approach is standard in human pose estimation tasks, balancing computational efficiency with spatial precision. Generally the heatmap size is set to ¼ of the input image size as noted in recent research (Li *et al.*, 2023). Setting the heatmap size to the same as the image size wouldn't be a problem, but it requires a lot more memory doing it that way, which we don't have.

Data Augmentation and Data Handling

We applied different augmentation strategies for each dataset to try boosting the performance of the models trained on them:

- MPII: During initial training on MPII, no augmentation was applied. This was due to the low performance of the models trained on MPII in general, which led to trying out different ideas first. However, we did try as a last-ditch effort for the MPII dataset to apply data augmentation to only the subset of the images containing cyclists and applied; random rotation (± 15 degrees) and horizontal flipping. This improved the prediction accuracy when tested on cyclists.
- COCO: We noticed that the model often predicted some of the joint pairs (left and right wrist for example) at the same location on the image, which led us to believe that the dataset might be showing one side of the body far more than the other. Therefore, we applied random horizontal flipping during training to the entire dataset and relabelled the left and right joints. This simple but effective augmentation helped improve generalization to mirrored poses and reduced overfitting. The models trained on COCO were noticeably more stable in their predictions and joint pairs were clearly separated more often.

All training images and heatmaps were normalized to the [0,1] range directly before training. To efficiently manage the COCO dataset, we stored the preprocessed images and joint heatmaps in an HDF5 file. We then used a custom ‘keras.utils.Sequence’ generator to stream the data during training. This generator handled the on-the-fly augmentation mentioned above as well as normalizing the images and heatmaps, which allowed us to train models without loading the full dataset into memory and ensured consistency across epochs. For the MPII dataset we stored the entire dataset as NumPy arrays, as well as normalizing the images and heatmaps to the [0, 1] range.

Model Implementation

We explored a variety of custom and pretrained CNN architectures to tackle human pose estimation by predicting 2D joint positions from images. Our models focused on detecting key joints relevant to bike fitting, such as knees, hips, ankles, elbows, and shoulders. We also only cared about the side of the cyclist facing us, meaning we only really care about the left joints if the left side of the cyclist is facing the camera. Below is a summary of the models tested, including our observations and outcomes.

Models Considered

1. **Coordinate Regression Model**
   This model directly predicted (x, y) coordinates for each joint from a 128x128 MPII image. Despite being simple and fast to train, it performed poorly, with unstable predictions and large errors – especially for occluded or small joints. It lacked the spatial awareness that heatmap-based approaches offer and was ultimately discarded early in the project.
2. **Simple Heatmap Model**
   A basic convolutional model that outputs one heatmap per joint. It performed moderately well on MPII, particularly after we retrained it on the cyclist-only subset with augmentation. On COCO, this model gave decent results, correctly identifying many keypoints, but struggled with cyclist images. Common failure cases included left/right confusion and mixing up elbows with hips.
3. **Encoder-Decoder Model (U-Net style)**
   This model used skip connections and a downsampling-upsampling structure. On MPII, it produced strange checkerboard patterns in the predicted heatmaps, possibly due to upsampling artifacts or training instability. Given its poor results, we chose not to test it further on COCO.
4. **Stacked Hourglass Network**
   Based on a state-of-the-art architecture, this model was designed for robust multi-scale feature learning. However, due to its high computational requirements, it was impractical to train on our available GPU (a 3060 TI). Training steps seemed to take up to 20 minutes each, making it infeasible to evaluate or tune properly. We halted any further development of this model. Our implementation of the stacked hourglass network was based on code by Li (2020), available on GitHub.
5. **Pretrained ResNet-50 Backbone**
   This was the best-performing model on COCO. We used a ResNet-50 encoder pretrained on ImageNet followed by a series of transpose convolutional layers to generate heatmaps. This model gave excellent initial results as shown in figure 1, where these results were obtained after only 1 epoch using this model. The model clearly detected all major joints but got confused with the ankles. After later

improvements to the model the results improved by quite a bit as shown in figure 2, were the model correctly placed basically every joint. The predictions were spot on for the joints facing the camera. However, it showed signs of overfitting after epoch 9, likely due to relatively small subset of filtered COCO data.

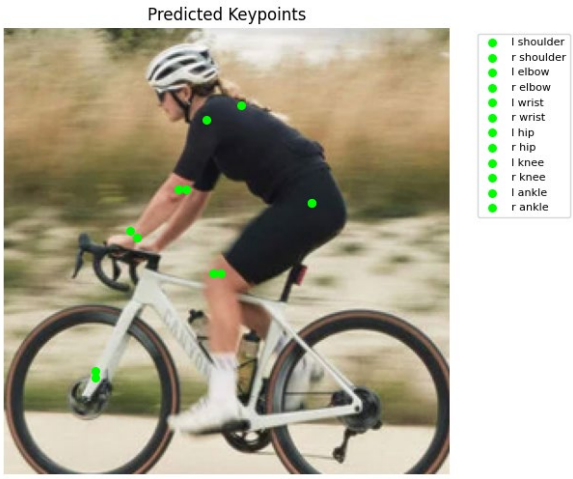

Figure 1: Image showing predicted joints for the first version of the pretrained ResNet-50 model trained after 1 epoch.

Figure 2: Image showing the latest ResNet-50 model predictions.

Theoretical Motivation

We ultimately focused on heatmap-based pose estimation because it offers better spatial awareness, especially when working with occluded joints. The ability to represent uncertainty as a distribution rather than a single point is a clear advantage compared to regression models. Lastly, alignment with modern pose estimation research such as HRNet and OpenPose. Regression models simply lack the robustness needed for complex or real-world images.

The different models we chose to focus on were chosen because they represent a balance between simplicity, performance and deployability. The simple model allowed us to experiment and test ideas without long training times, which made it ideal for early developments and experiments. On the other hand, the ResNet-based model leverages

deep hierarchical feature extraction. Both models support the core idea that learning spatial probability maps is more effective than predicting coordinates in the context of pose estimation.

Hyperparameters and Tuning

We chose to focus on two main models: the simple heatmap model and the pretrained ResNet-50 backbone model. For both models the list under is mostly the same with slight differences explained in each of them:

- Learning rate: 1e-4 or 0.001, also tested with 1e-5 and 1e-6, but these values led to very slow learning or stagnation. We therefore settled on 1e-4 with ReduceLROnPlateau.

- Batch size: Due to GPU limitations we couldn't test with more than 16 for MPII and 8 for COCO. We also didn't want to test with less as the COCO models were already using 8 hours for training (and decreasing this would increase training time)

- Epochs: Tested early generations of the simple model and the pretrained model at 100 epochs but later realized there were no real performance gains after epoch 40 according to the loss function. As mentioned above, the pretrained model seemed to overfit after epoch 9, but by using EarlyStopping it recovered the weights from that epoch.

- Callbacks: EarlyStopping monitoring validation loss with patience 8 as well as it will restore the best weights. ModelCheckpoint monitoring validation loss to backup models in case of errors like OOM or crashes. ReduceLROnPlateau monitoring validation loss with factor 0.5, patience 5 and with a minimum learning rate of 1e-6.

- Loss function: We never managed to get MSE or MSE-like functions to work as it only gave NaN values, which makes sense as most of the pixels in a heatmap are empty and MSE averages the squared differences, causing the loss to diverge during early training (loss exploded). For MPII we therefore used Huber which is less sensitive to outliers, but still penalizes minor errors like done in MSE (Alake, 2024). For COCO we couldn't use a predefined loss function, as the COCO data doesn't include the same amount of joint in each sample. The option was therefore to create custom loss functions for this purpose, where it only computes the loss for the visible joint heatmaps. These functions were generated by ChatGPT due to their complexity and can be found in our notebook 'exploring-coco.ipynb' on our public repository (Sæther and Egeland, 2025):

    - masked_huber_loss: A variation of the successful predefined Huber loss function which only computes loss where ground truth is non-zero.
    - masked_weighted_huber_loss: A custom variation of the masked_huber_loss function with per-joint weighting. Which allowed us to reduce the influence of missing or less relevant joints by assigning lower weights. We tested out different weights, like 81, 50, 20 and 10, and overall, it seemed like we were getting better results with the last one.

- Optimizer: We primarily used the Adam optimizer for all model training. This choice was made because this optimizer is well-suited for this task due to its adaptive learning rate mechanism, which helps accelerate convergence and handle sparse gradients (Ultralytics, 2025). The main motivation behind using the Adam optimizer though, is in fact that is the default optimizer in state-of-the-art models like the hourglass network seen in the model implemented by Li (2020).

- Regularization: To prevent overfitting and stabilize training, we relied primarily on batch normalization and early stopping rather than heavier regularization techniques like dropout, which didn't show a significant improvement during initial testing compared to other strategies. Batch normalization was applied after most convolutional layers to smooth the optimization and improve gradient flow, especially in deeper models.

- Dilation rate: In the simple heatmap model when working on MPII, we found that the model was confused and couldn't decide very well on certain heatmaps. A change that happened to do quite a lot was adding a dilation rate of 2 to some of the convolutional layers. By adding this, it will increase the receptive field, essentially allowing the model to capture more spatial context (Yu and Koltun, 2016). An example of this is joint 12 and 13 in figure 3 and figure 4 below. These joints correspond to the right and left shoulder, and when the model sees the entire context, it realizes where the joints should rather be. While dilation isn't a regularization technique per se, it can be considered a structural hyperparameter that improved joint localization without increasing model depth or complexity.

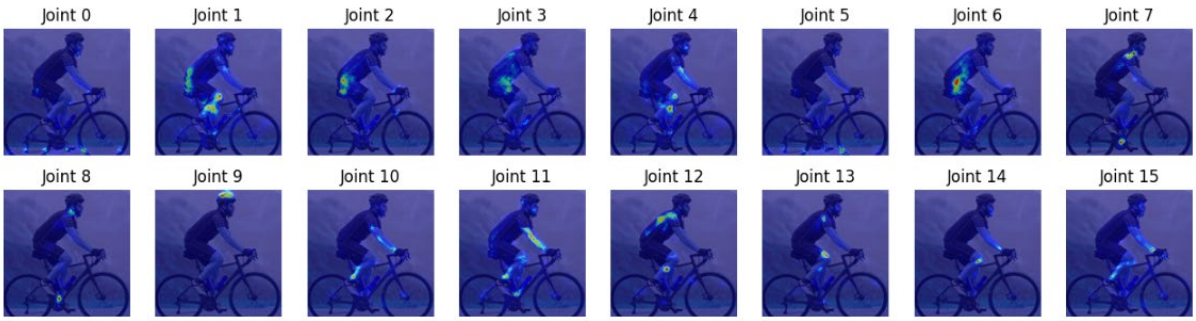

Figure 3: A collage of images with heatmaps from early stages of the simple heatmap model.

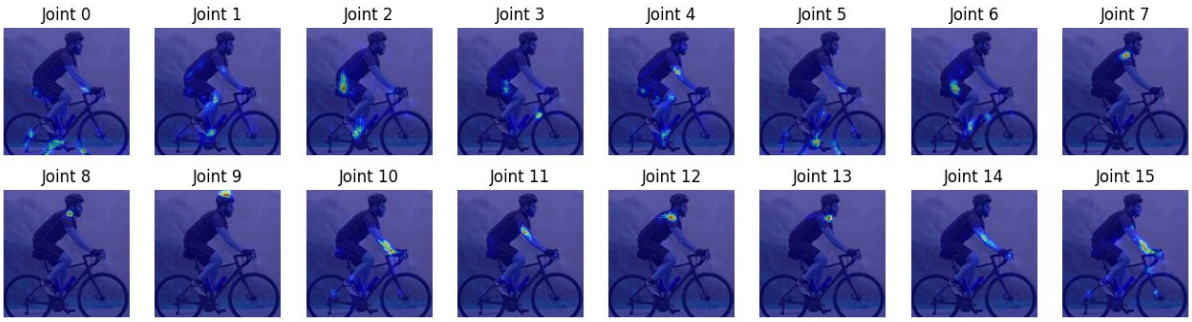

Figure 4: A collage of images with heatmaps from early stages of the simple heatmap model after dilation_rate=2 was added to some convolutional layers.

Training Constraints and Practical Limitations

Training these models was impacted by several constraints:

- GPU Limitations: Due to available hardware, we could not train models with large batch sizes (e.g. more than 16). This limited the depth of experimentation and increased training times for larger datasets. Additionally, the stacked hourglass model could not by fully trained because of its extreme computational cost.
- Long Training Times: Training on COCO models took up to 8 hours per session, and sometimes even longer depending on the total epochs and when EarlyStopping kicks in. This made it impractical to experiment with multiple configurations. Many models were trained overnight, and due to these long training times, we were limited in the number of epochs we could run and the number of models we could test.

We will delve further into these problems in the Evaluation chapter below.

Evaluation

To evaluate the performance of our models, we relied primarily on pose estimation-specific metrics, as well as qualitative analysis of the predictions. Our focus was on keypoint localization accuracy across different joints, which is essential for our use case in bike fitting.

It is important to note that the following results gathered from the models were all based on the COCO dataset with modifications like previously explained.

Metrics Used

We used the following metrics to evaluate our models:

- PCK@0.2 (Percentage of Correct Keypoints): Measures the proportion of keypoints predicted within 20% of a reference length from the ground truth.
- mPCK (mean PCK): Calculates the average PCK over multiple thresholds, offering a more comprehensive view of model performance across varying precision levels.
- AUC (Area Under the PCK Curve): Measures the area under the curve generated by plotting PCK across thresholds (e.g. from 0.05 to 0.5), which summarizes robustness across tolerances.
- Per-joint PCK: Allows for more detailed inspection of performance per joint, identifying which joints the model struggles with (e.g. hips, ankles).
- PCK per threshold: We also computed per-joint PCK values across thresholds to visualize performance consistency.

These metrics are commonly used to evaluate human pose estimation systems in academic and industry contexts (Barla, 2021; Samkari *et al.*, 2023).

Model Results

Simple Heatmap Model:

- PCK@0.2: 0.356
- Per-joint PCK@0.2: [0.589 0.579 0.385 0.389 0.356 0.365 0.279 0.284 0.276 0.269 0.253 0.244]
- mPCK: 0.358
- AUC: 0.366
- PCK per threshold: [0.103 0.251 0.322 0.356 0.378 0.397 0.416 0.434 0.453 0.473]

While this model is fast and light, it is not very accurate overall. Per-joint PCK showed weaker performance on the lower-body joints, which by matching the index from the list to table 1, showcasing that the model didn't do a great job predicting hips, knees or

ankles. However, considering the simplicity of the architecture and the limited training data, it still performed decently.

Table 1: Table for showing joint index and name for the modified COCO dataset.

Joint index	Joint name
0	Left shoulder
1	Right shoulder
2	Left elbow
3	Right elbow
4	Left wrist
5	Right wrist
6	Left hip
7	Right hip
8	Left knee
9	Right knee
10	Left ankle
11	Right ankle

We can understand more about the model if we plot the predicted heatmap for each joint, shown in figure 5. The model has a hard time deciding which location is correct for some of the heatmaps.

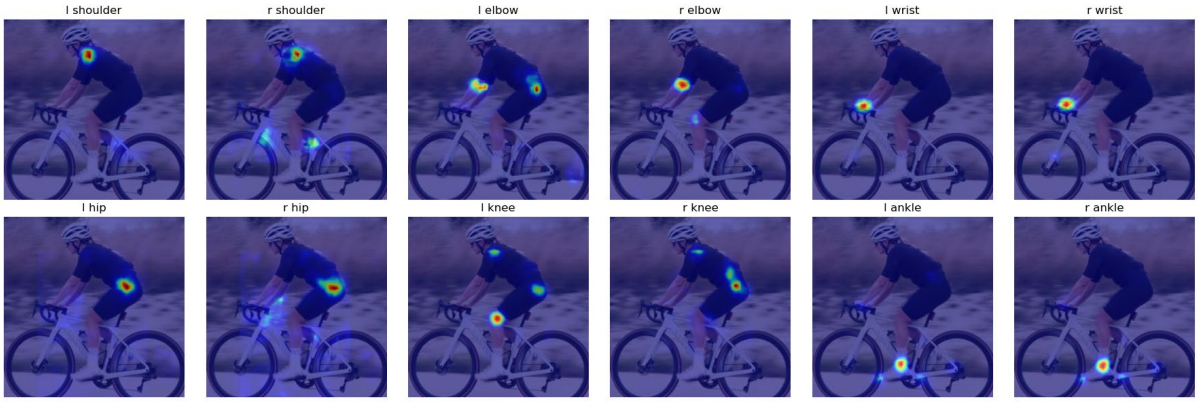

Figure 5: A collage of heatmaps overlayed on images. Heatmaps predicted by the latest 'Simple Heatmap' model.

Pretrained ResNet-50 model:

- PCK@0.2: 0.55
- Per-joint PCK@0.2: [0.791 0.783 0.622 0.606 0.521 0.524 0.571 0.569 0.449 0.436 0.37 0.361]
- mPCK: 0.529
- AUC: 0.543
- PCK per threshold: [0.16 0.386 0.499 0.55 0.579 0.598 0.612 0.625 0.634 0.643]

This model clearly outperformed our custom-built models in all evaluation metrics. It performed particularly well on joints such as shoulders and elbows, but as also seen in

the simple model it also struggles with the lower-body joints, especially the knees and ankles. This is likely due to the lack of cyclist-specific training data. The COCO dataset also includes far more upper-body joints than lower-body joints like shown in table 2.

Table 2: Joint-wise annotation counts for the original COCO dataset, showing an uneven distribution of joints.

Joint-wise annotation counts:
nose: 101435
l_eye: 88206
r_eye: 88806
l_ear: 76780
r_ear: 77785
l_shoulder: 130904
r_shoulder: 131177
l_elbow: 103588
r_elbow: 105227
l_wrist: 94592
r_wrist: 96898
l_hip: 113761
r_hip: 114067
l_knee: 86613
r_knee: 86857
l_ankle: 72740
r_ankle: 72847

To get a better understanding of how the model is predicting, figure 6 shows the same collage of pictures like figure 5 did for the simple model. Here the model manages to identify the location of every joint, especially if we are talking about the ones in our interest; left shoulder, left elbow, … left ankle. It shows some confusion when it comes to the joints on the right side of the body, with right wrist and right ankle being the standouts. Overall, we were overall pleased with the model's practical performance, despite its modest PCK score.

Figure 6: A collage of heatmaps overlayed on images. Heatmaps predicted by the latest 'Pretrained ResNet-50' model

Baseline Estimation

In terms of performance expectations, we didn't train a conventional academic baseline, like HRNet or OpenPose due to time and resource constraints. However, we set our own functional baseline by considering the performance of tools like MyVeloFit and other state-of-the-art pose estimators, which are known to perform reliably on side-view cyclist images.

Even though these systems weren't formally benchmarked in our environment, we had a practical sense of what "good" predictions should look like – both from testing MyVeloFit and from reviewing predictions in related literature, as well as using common sense. This became our target benchmark, and we continuously compared our overlay results and ultimately PCK scores against this standard to judge whether our models were reaching usable accuracy for a bike fit.

While this model is less formal than using a trained academic benchmark, it was highly relevant for our application and guided our model development effectively.

Interpretation Techniques

While we computed quantitative metrics such as PCK, mPCK, and AUC for final evaluation, most of the development and debugging was guided by visual interpretation. Specifically, we:

- Overlayed the predicted heatmaps on top of the original input images to visually assess whether the model was accurately locating the joints.
- Observed how clear and sharp the model responded to certain body parts for the side-view cyclist images, as well as for normal images of people walking for instance.

This identified common failure cases, such as left/right confusion, misclassified joints, and incorrect predictions in occluded or non-visible joints. Like mentioned in the subchapter Model Results and in figure 6, visual interpretation can be much more effective than looking at quantitative metrics.

Architectural Potential

While we observed that the pretrained ResNet-50 model clearly outperformed our custom-built architectures, we did not fully exploit its potential due to time and hardware limitations. The model began to overfit after approximately 9 epochs, like shown in figure 7 below. This indicates that more robust regularization could further improve performance.

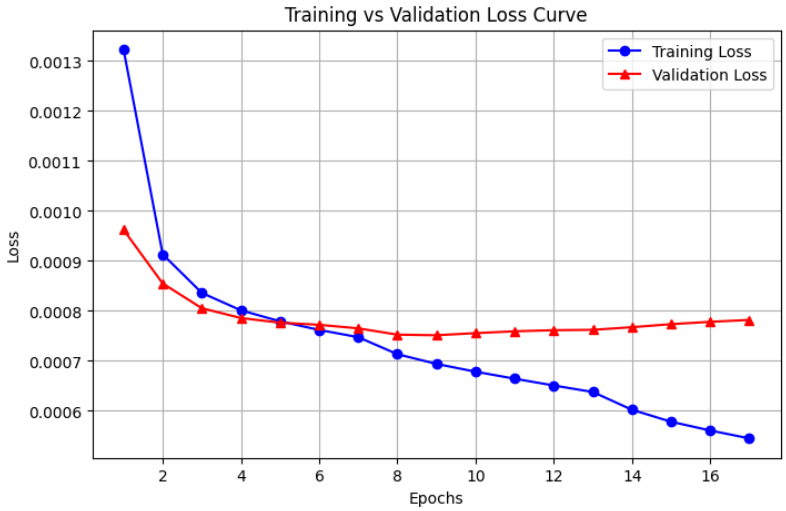

Figure 7: Training vs Validation loss curve for the latest 'Pretrained ResNet-50' model. Graph showcasing signs of model overfitting after 9 epochs.

For the simple model it never showed signs of overfitting during training. The training and validation curve seemed to follow each other for every epoch, hinting that the model might be almost fully explored and close to reaching its maximum potential.

Runtime and Deployment Considerations

We considered model size and runtime behaviour as part of our deployment strategy, especially since the models were going to be ran locally in the browser using TensorFlow.js, meaning they need to be fetched from a server. After optimization, which saves the model stripped down without an optimizer:

- The simple model was reduced from 15 MB to 5 MB
- The ResNet-50 model was reduced from 400 MB to 133 MB

The models were then converted to TensorFlow.js format, meaning the model was stored as a JSON file together with the weights as BIN files. This shrank the size by around 50% due to quantization and serialization optimizers.

This made them suitable for browser-based inference without the need for a dedicated backend. While the larger model took longer to load, particularly on slower networks, prediction time was fast for both models since inference runs directly in the user's browser. Because we predicted on individual images (not video streams), loading time has a bigger impact than runtime performance, and we found the models sufficiently responsive for interactive use. More on this in the coming chapter.

Deployment

Our goal from the beginning was to create a usable tool for cyclists, so deployment was an important part of the project. We developed and deployed a fully functional Next.js web application hosted on Vercel at bikefit.bysaether.com. The app runs entirely in the browser using TensorFlow.js, meaning no user images are uploaded to a server – improving both privacy and speed.

Framework and Model Serving

Like previously mentioned, we converted our Keras models to TensorFlow.js using the `tensorflowjs_converter` tool, enabling inference directly in the browser. To deliver these models efficiently, we store all model files in a Vercel blob publicly, which allows us to serve these models at scale with fast delivery. This removes the need for a separate server like AWS S3, although this would keep the model more private.

The app supports loading four different models, which users can choose from:

- The **Simple Heatmap Model** and the **Pretrained ResNet-50 Model**
- Two **MoveNet models**: *Lightning* (lightweight) and *Thunder* (more accurate)

Once a model is selected and downloaded from the public blob, it is temporarily cached in the app's runtime, minimizing load time if you wish to test out different models. Users can upload an image via a drag-and-drop file picker and are prompted to crop the image with a cropping tool. A built-in guide explains how to properly frame the side-view image for optimal results.

Inference and Output

After cropping, the selected model performs pose estimation on the image. The app outputs:

- An image with predicted joints and skeleton
- A list of calculated joint angles, like knee, hip, and shoulder angle.

The web app will compare the calculated joint angles to recommended ranges for road cyclists. The recommended ranges used are from MyVeloFit, which they updated a few years back in 2022, and are adapted to better accommodate more riders (MyVeloFit, 2022).

Additionally, users can enter their goal, either 'comfort' or 'performance' and known issues, for this to be sent along with detected angles to a GPT-4.0 model via the OpenAI API. The AI then returns feedback and recommendations for adjustments that can be made on the bike, making the experience both informative and interactive.

Monitoring and Maintenance

Currently, no server-side monitoring is required or available, as the app is entirely client-side as mentioned earlier. We can however monitor traffic on Vercel and API usage on the OpenAI portal. For maintaining the models, you can simply swap out the JSON and BIN files in the Vercel blob for updating them. Alternatively, you could update the frontend to allow more and different models to be incorporated.

Future Expansions and User Experience

The web app is per now a very basic application, meant for showcasing the project, and it does not have any real commercial value. Possible improvements for the future could include:

- UI and structure conversion for commercial usage
- User accounts for storing fits and tracking progress
- Model(s) could be hosted on a backend for keeping models private
- Optional video-based fitting
- Deeper integration with GPT models for better responses

Also, a model trained on a cyclist specific pose dataset would be one of the first changes and improvements to be made.

Conclusion

Overall, we are very satisfied with the outcome of this project. Our goal was to explore how deep learning could be used to support bike fitting through pose estimation – and we successfully built, trained, evaluated, and deployed multiple models, culminating in a fully functional web application. We also managed to incorporate a GPT4.0 model into the web application which was not a part of the project originally.

Metrics and Evaluation

In hindsight, the evaluation metrics we selected – particularly mPCK and per-joint PCK – were appropriate and informative for our use case. These metrics provided both quantitative insight into model accuracy and helped highlight joint-specific weaknesses. Combining this with a qualitative analysis of the heatmaps via the visual overlays, we were able to interpret the model's performance and identify weaknesses. It was made clear that our custom simple heatmap model could not localize occluded joints. Specifically joints close to other detailed parts of the image, like the handlebars, bike frame and wheels. Our ResNet-based model on the other hand manages to correctly identify each visible joint, as shown in figure 8 below to the left. Meanwhile in figure 9, the simple model misplaces the right ankle joint.

Figure 8: Showcasing the joints converted from the 'Pretrained ResNet-50' model output.
Figure 9: Showcasing the joints converted from the 'Simple Heatmap' model output.

Effectiveness of Deep Learning

We believe deep learning was a very suitable approach for this problem. Traditional computer vision methods would not have been capable of robustly detecting human joints in varied poses, image quality levels, and clothing types – especially without strict background or lighting conditions. While we acknowledge that deep learning does not replace professional bike fitting, our models showed promising potential to provide

meaningful and accessible feedback, especially when combined with the power of a GPT-4.0 analysis.

Challenges and Limitations

As stated earlier we have faced a series of technical challenges and limitations. For starters the COCO dataset, while its large, was not specifically tailored to cyclists. This led to suboptimal performance on key joints like knees and ankles, which are especially important in bike fitting. We have also faced hardware constraints, where we were not able to train computationally heavy models like the stacked hourglass model, as training took too long.

Final Reflection

This project provided valuable hands-on experience in real-world deep learning engineering – from designing models and managing training data to debugging runtime issues and deploying a web-based application. If we were to repeat or extend the project, we would likely:

- Spend more time on the MPII dataset or create a cyclist-specific dataset.
- Explore further regularization techniques to prevent overfitting
- Perform more hyperparameter tuning

Throughout the project, we observed that the models might be unsure of the exact positioning of each joint, but the area is often spot on – especially for joints that are clearly visible. While we would have liked to generate perfect pixel-level accuracy, all things considered; we are proud of what we accomplished – especially being able to deploy this into a fully functional web app. This project gave us a deeper appreciation of both the power and practical challenges of deep learning in real-world applications.

References

Alake, R. (2024) *Loss Functions in Machine Learning Explained*. Available at: https://www.datacamp.com/tutorial/loss-function-in-machine-learning (Accessed: 24 April 2025).

Andriluka, M. *et al.* (2014) '2D Human Pose Estimation: New Benchmark and State of the Art Analysis', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3686–3693. Available at: https://openaccess.thecvf.com/content_cvpr_2014/papers/Andriluka_2D_Human_Pose_2014_CVPR_paper.pdf.

APIIR (no date) *Get a Bike Fit in Minutes – APIIR*. Available at: https://www.apiir.tech/ (Accessed: 24 April 2025).

Barla, N. (2021) *Human Pose Estimation: Deep Learning Approach [2024 Guide]*. Available at: https://www.v7labs.com/blog/human-pose-estimation-guide (Accessed: 25 April 2025).

Li, E.Y. (2020) *deep-vision/Hourglass/tensorflow at master · ethanyanjiali/deep-vision*, *GitHub*. Available at: https://github.com/ethanyanjiali/deep-vision/tree/master/Hourglass/tensorflow (Accessed: 24 April 2025).

Li, S. *et al.* (2023) 'SSA Net: Small Scale-Aware Enhancement Network for Human Pose Estimation', *Sensors*, 23(17), p. 7299. Available at: https://doi.org/10.3390/s23177299.

Lin, T.-Y. *et al.* (2015) 'Microsoft COCO: Common Objects in Context'. arXiv. Available at: https://doi.org/10.48550/arXiv.1405.0312.

MyVeloFit (2022) 'New Bike Fit Ranges - MyVeloFit', 31 January. Available at: https://www.myvelofit.com/fit-academy/new-bike-fit-ranges/ (Accessed: 25 April 2025).

MyVeloFit (2023) 'Rider Pricing - MyVeloFit', 16 May. Available at: https://www.myvelofit.com/pricing/rider/ (Accessed: 24 April 2025).

Sæther, W. and Egeland, O.-M.M. (2025) 'exploring-coco.ipynb'. GitHub. Available at: https://github.com/williamsaether/BikeFit/blob/master/data/exploring-coco.ipynb (Accessed: 24 April 2025).

Samkari, E. *et al.* (2023) 'Human Pose Estimation Using Deep Learning: A Systematic Literature Review', *Machine Learning and Knowledge Extraction*, 5(4), pp. 1612–1659. Available at: https://doi.org/10.3390/make5040081.

Ultralytics (2025) *Adam Optimizer: Deep Learning | Ultralytics*. Available at: https://www.ultralytics.com/glossary/adam-optimizer (Accessed: 24 April 2025).

Yu, F. and Koltun, V. (2016) 'Multi-Scale Context Aggregation by Dilated Convolutions'. arXiv. Available at: https://doi.org/10.48550/arXiv.1511.07122.