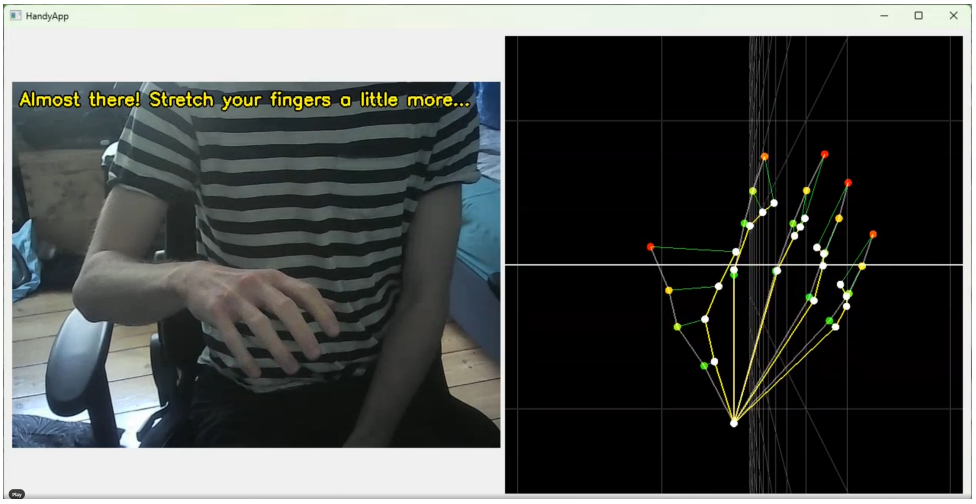
Optimizing Physical Hand Training Effectiveness Using Deep Learning and Unsupervised Learning



Author 1: Peter Asbæk Skøt (S185185)
Author 2: Jacob Asbæk Wolf (S236897)
Course: Applied Machine Learning and Big Data
Course ID: 62T22
Date: May 30, 2025

Equal contribution by both authors.

Contents

Optimizing Physical Hand Training Effectiveness Using Deep Learning and Unsupervised Learning

Peter Asbæk Skøt (DTU Student S185185)
Jacob Asbæk Wolf (DTU Student S236897)

June 10, 2025

Abstract

We implement a Convolutional Neural Network and a K-means model to assist disabled people prescribed with hand exercises by a therapist. Our software takes input from a teacher and provides a graphical view to the student to motivate and give feedback on inaccuracies in their performance. The software is inspired by teenagers diagnosed with Cerebral Palsy, who may suffer from cognitive disorders and find it difficult to remember instructions from a therapist or find it hard to find motivation. We have used open-source software to develop the software and made it work on a simple laptop with a web camera. Our work therefore aligns with the United Nations' Sustainable Development Goal 3: Good Health and Well-being [11]. Use of our work may be expanded to telemedicine, and to poorer regions of the world, as computers get faster and more globally accessible. We find that the software performs well on simple experiments we performed. To further validate the software for clinical use, a larger set of test runs should be performed and evaluated by trained medical professionals. The software we provide may qualify as a medical device and should be evaluated as such. We have not done this in our work, but we have made the software available for others to use and evaluate. The software is available on GitHub [12].

1 Introduction

Hand mobility is a critical component of many daily tasks, yet it is often compromised in individuals suffering from conditions such as Cerebral Palsy (CP) or recovering from physical trauma. In order to recover or strengthen hand mobility, physical therapy in the form of exercises is essential, which for many is either inaccessible or difficult to maintain motivation outside of clinical environments.

We aim to bridge the gap between these and improve motivation during hand exercises by developing a light-weight camera-based software that can run on a simple laptop with minimal setup. We utilize computer vision and clustering to provide patients with a visual interface, giving them live feedback on the performance on a given hand exercise.

Instead of relying entirely on manual evaluation, the software predicts hand landmarks using Deep Learning, and different hand postures are clustering with k-means, such that feedback can be given directly by the software without manual evaluation by an expert.

With this software, we aim to enable a low-cost, easily available rehabilitation and training for patients, aligning with the United Nations' Sustainable Development Goal 3: Ensure healthy lives and promote well-being for all at all ages [11]. Additionally, As speed and availability of computers increase, using the software for telemedicine purposes and expanding to poorer regions of the world becomes increasingly viable.

2 Problem Definition

We seek to apply machine learning to hand exercise training targeted for patients with hand mobility issues with the goal of making the training more effective and the patient more motivated. Hereunder:

1. Track hand positions from a web camera and apply supervised machine learning models to extract hand landmarks.
2. Use the extracted landmarks and unsupervised machine learning to rate the students hand positions against the teachers hand positions.
3. Apply supervised and unsupervised machine learning to a live stream of the students hand positions to provide feedback on how well the student is mimicking the teachers hand positions.

To put the problem definition and the investigation question into a software context, see the UML use case diagram in figure 1. It shows how the teacher first provides data on how each exercise is performed in the teacher system. The teacher then hands over the system to the student, who can now perform the exercises using the same technology.

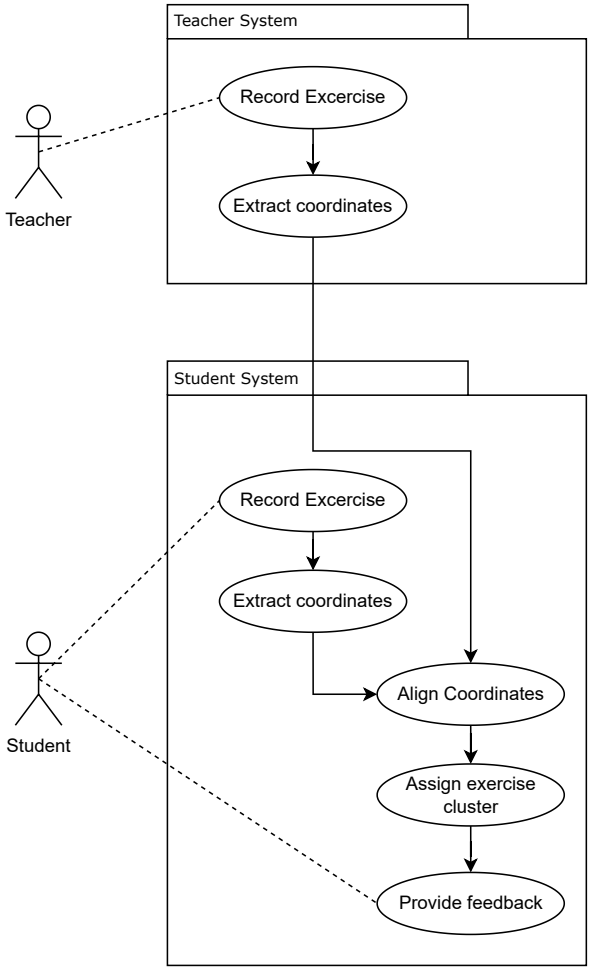


Figure 1: Use case UML diagram

3 Definitions

- **Landmark** A landmark is a specific point on the hand, such as the tip of the thumb or the wrist.
- **Supervised Machine Learning** For each observation of the predictor measurement(s) x_i, i=1,...,n there is an associated response measurement y_i. We wish to fit a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference). [2, p 25]]
- **Unsupervised Machine Learning** is a set of statistical tools intended for the setting in which we have only a set of features X_1,X_2,...,X_p measured on n observations. We are not interested in prediction, because we do not have an associated response variable Y. [2, p 503]
- **Convolutional Neural Network (CNN)** A CNN is a type of deep learning model that is particularly well-suited for image processing tasks. CNNs are designed to automatically learn spatial hierarchies of features from images, making them ideal for tasks such as image classification, object detection, and landmark extraction.
- **K-means** K-means is an unsupervised machine learning algorithm used for clustering data points into K distinct groups based on their features. It works by iteratively assigning data points to the nearest cluster centroid and updating the centroids until convergence.

4 Data Extraction and Data Processing

We are using data in different scenarios described in the following sections. They fall in three categories: 1: Using existing datasets to train a model on landmark recognition from pictures of hands, 2: Using a web camera to record teacher hand pictures in one correct and a few known incorrect positions in order to train the K-means model. 3: Using a live stream of student frames from the web camera along with landmark extraction and advice from k-means to advice the student. We will detail each of these in the subsections below.

4.1 Landmark calibration and normalization

HanCo provides landmarks in world coordinates in meters. To simplify the learning process and focus on learning position and not perspective, we convert from world to camera coordinates using:

$$\mathbf{X}^{\text{camera}} = \left(\mathbf{M} \cdot \begin{bmatrix} \mathbf{X}^{\text{world}} \\ 1 \end{bmatrix} \right)^T \tag{1}$$

Additionally, we normalize hand size and position. Given the 3D joint coordinates,

$$\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

we apply the following steps:

1. Centering at the wrist joint:
$$\mathbf{x}'_i = \mathbf{x}_i - \mathbf{x}_b \quad \forall i \tag{2}$$
2. Scaling using the wrist and middle finger base joint:
$$\text{scale} = \|\mathbf{x}'_s\|_2 \tag{3}$$

$$\tilde{\mathbf{x}}_i = \frac{\mathbf{x}'_i}{\text{scale}} \quad \forall i \tag{4}$$

Where:

- $\mathbf{x}_b$ is the position of the wrist joint.
- $\mathbf{x}_s$ is the position of the scale joint (middle finger base).
- $\tilde{\mathbf{x}}_i$ are the normalized joint coordinates.

This normalization process ensures that the model is invariant to hand size and position, allowing it to focus on the relative positions of the joints. The scaling factor is calculated as the distance between the wrist and middle finger base, allowing for a stable hand size and position across different hands.

4.2 Data structure

Before we go into the details of the data collection, we will describe the data structure we are using. The 3 main components of the datasets are the images, landmarks, and the calibration data.

- **Images** Images consist of 224 x 224 RGB images.
- **Landmarks** Landmark data is contained in a single json file per image id, each containing a 21 x 3 array of x, y, and z coordinates.
- **Calibration data** Calibration data is a json file containing: a: An intrinsic camera matrix K for each camera, b: An extrinsic camera matrix M for each camera

To structure the data, we utilize a torch Dataset class. Below in figure is the end of the __getitem__ method shown to illustrate the data structure.

```
{
    "image": image,
    "landmarks": joints.contiguous().view(-1), # Flattened
    "root": root, # position of the root/reference joint (wrist)
    "scale": scale, # scale used to normalize the hand size
    "calibration": (K, M),
    "hand_id": hand_id,
    "frame_id": frame_id,
    "camera_id": cam_id,
 }
```

4.3 Dataset for supervised training

First, we train a CNN model that learns normalized 3D landmark coordinates from hand images. We use the HanCo dataset [13], which is an extension of the FreiHAND dataset [14] originally consisting of 32560 training samples. The extended HanCo dataset introduces multiple different frames per hand along with a multiview from 8 different cameras, increasing the number to 860,304 images. HanCo also provides a dataset with randomized backgrounds, consisting 88.8% of the full dataset. This is the full dataset that we use to train our model. For target variables we use data provided with the HanCo dataset, which consists of x, y, z coordinates for 21 landmarks (wrist + 4 landmarks per finger), totalling 21 x 3 landmarks per image.

4.4 Dataset for unsupervised training

We extract landmarks with the purpose of comparing teacher and student landmarks while they perform an exercise. We start by examining the teacher's perception of what a good exercise looks like with a set of images taken by the teacher. We also ask the teacher to take a number of pictures of typical errors that the student could make. In this paper, we are working with one posture that is often difficult for CP patients: stretching the fingers. We give it the title "Finger stretch". CP Patients have a tendency to bend their fingers due to tension in their muscles. We therefore let the teacher record several similar pictures of each of the following scenarios:

- Good posture: The teacher's hand is lying flat on the table with fingers in rest position.
- Almost good posture: The teacher's fingers are slightly bent, but not too far from the flat stretch position.
- Bad posture: Fingers are fist-like

The recording setup has evolved over time, as we got more acquainted with the hardware. Initially, we stacked 2-4 coffee cups on top of each other with a web camera on top as shown in figure 2. This was useful and allowed for good lightning and gave the camera a proper angle to capture all the landmarks. However, we realized that this would be a complication for CP patients and families having such a setup at home. This setup was used to create the teacher pictures used to train the K-means model we will discuss later. We tested out just using the web camera sitting on top of the screen of many laptops, which worked reasonably well. In cases where the camera was too low to capture the landmarks, we could put a book halfway under the laptop to raise the camera. An impression of how it worked can be seen in figure 16.

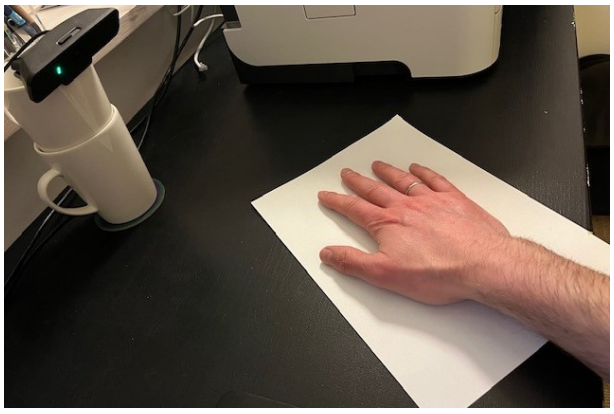


Figure 2: Initial setup for recording hand pictures.

This resulted in 1174 images with 370, 420 and 402 images of the three categories respectively in the same order. The images were captured using the OpenCV computer vision library. Similarly, we utilize OpenCV for live-streaming and providing feedback messages to the user, as covered later in this section.

Furthermore, we could have taken more postures and made a more detailed dataset for students to be able to choose between multiple exercises. But since our interest is to show how the software works, we have limited ourselves to this one exercise. The teacher has also supplemented each right and wrong posture with an advice or a supporting message which can be shown to the student during the exercise. The messages are invented by us, but are inspired by real life instructions from a teacher to a student. Again, the messages themselves are not the focus of this paper, but we have included them in the software to show how it works. The messages are shown in the upper left corner of the screen in figure 16.

4.5 Live data in the final software

The datasets we use to analyze the hand recognition and the exercises do not need the data as such, but can work with their respective training configurations on the images they are being presented for. In the final software, we are applying the two models to a live stream of frames from the web camera. This data flow is kept on the device and is shown directly to the user. No data is being stored.

4.6 Discussion of data collection

We have developed the inference software to work on a simple laptop with a web camera that is easy to set up and use. It is important to note that in order to use the software in a clinical setting, much more data, test scenarios and exercises prescribed by real professional therapists should be collected in a more structured fashion and have its performance evaluated by trained medical professionals. We have not done this in our work, but we have made the software proof-of-concept available for others to use and evaluate. In case we should have collected more data, it would have been important to list some target therapeutic areas and select exercises for these. It would be crucial to do a proper risk assessment to identify as many of the potential risks of errors as possible and address these with challenging datasets to prove where the limitations of our models covered in the below sections are.

5 Modeling

With the datasets explained and in place, we will now take a closer look at the two models we use for extraction of coordinates and advice.

5.1 Modeling CNN

According to [2, pp. 406], Convolutional Neural Networks (CNNs) are particularly well-suited for image processing tasks. We have chosen CNNs because they are well suited to automatically learn spatial hierarchies of features from images. We are training such a model to recognize landmarks from hands with the previously mentioned datasets. Zimmermann [14] finds that using a backbone of a ResNet50 model [5], pretrained on ImageNet produces a good cross-dataset generalization. ResNet is characterized by its residual/bottleneck block structure which implements CNN layers along with a shortcut connection:

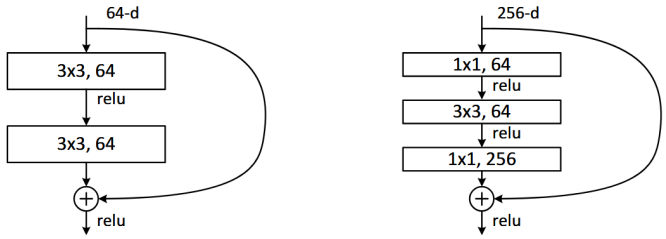


Figure 3: On the left, a building block for ResNet34 and below. On the right, a bottleneck connection used in ResNet50 and above [5].

For smaller sizes of ResNet, a building block with two convolutions with dimensionality 3 x 3 is used. For deeper networks, i.e. ResNet50 and above, the bottleneck layer is utilized to reduce the number of trainable parameters per block, allowing for deeper networks. In each block type, input is passed through

the set of convolutional layers and the result of the convolution is then summed with the input itself:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

Zimmermann [14] replace the fully connected layer in the model with the model head shown in the table below. We perform the same replacement of the fully connected layer.

Layer	**Dimensionality**
FC + ReLU	2048 x 2048
FC + ReLU	2048 x 2048
FC	2048 x 63

Table 1: Model head architecture with dimensionality of each layer.

5.1.1 Model Training

We trained the model in python, using the PyTorch library. Early training runs showed relatively poor results, such as bias and overfitting. To improve the learning and generalization of the model, careful selection of hyperparameters and data augmentation proved necessary. We will discuss these further in this section.

5.1.2 Data Augmentation

We utilize data augmentation extensively throughout training to increase the variance in the training data, helping the model's ability to generalize and reduce overfitting. By utilizing the randomized backgrounds provided by HanCo, some powerful augmentation is already provided, as the varying backgrounds helps the model learn the task at hand rather than certain backgrounds. In addition to this, we add several other sources of augmentation: color jitter, crop-resizing, random horizontal flipping and rotation. The two former are implemented directly in the training loop, allowing for GPU acceleration. The two latter require transformations to both images and landmarks, thus implementing them at the dataset level is more feasible. For rotation and flipping, the augmentation flow can be illustrated as follows:

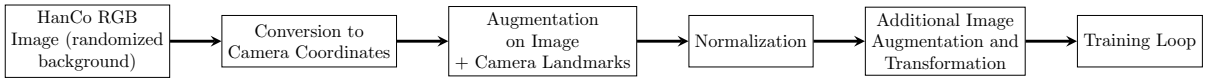


Figure 4: Data augmentation flowchart

We apply random horizontal flipping and rotation to the images, and apply similar transformations to the 3D landmarks. During training, we randomly rotate images by a random, uniformly distributed amount between -30 and +30 degrees, around the focal point of the lens, which is extracted from the intrinsic matrix K. In addition, we flip the images horizontally around the center with 50% probability. When rotating, we rotate the 3D landmarks by the same amount around the Z-axis at the origin point. Similarly, When flipping the image, we flip the landmarks across the YZ plane accordingly, as displayed by Figure 6.

In addition to augmentation performed at dataset level, we apply data augmentation only to the images within the training loop. We apply random color jitter to the images, randomly setting the brightness, contrast and saturation level to values between 80% and 120% of the original image. Additionally, we apply random crop-resizing, adjusting the scale and aspect ratios of the image. figure 7 visualizes the effects, setting the color jitter levels to 50% (for visual clarity) and crop-resizing at pixel coordinates (30, 30) with scale 0.8 and aspect ratio 0.75.

Finally, we normalize the images using the ImageNet normalization parameters [9].

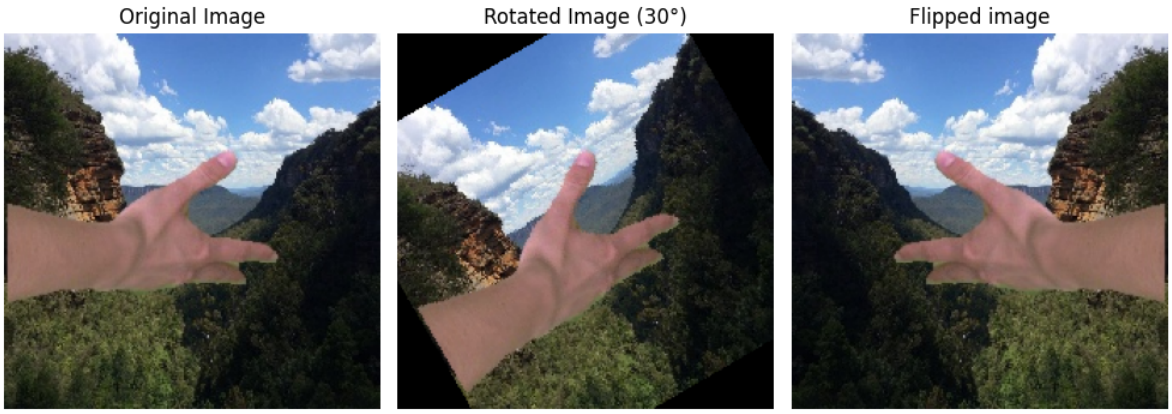


Figure 5: Image rotation and flipping.

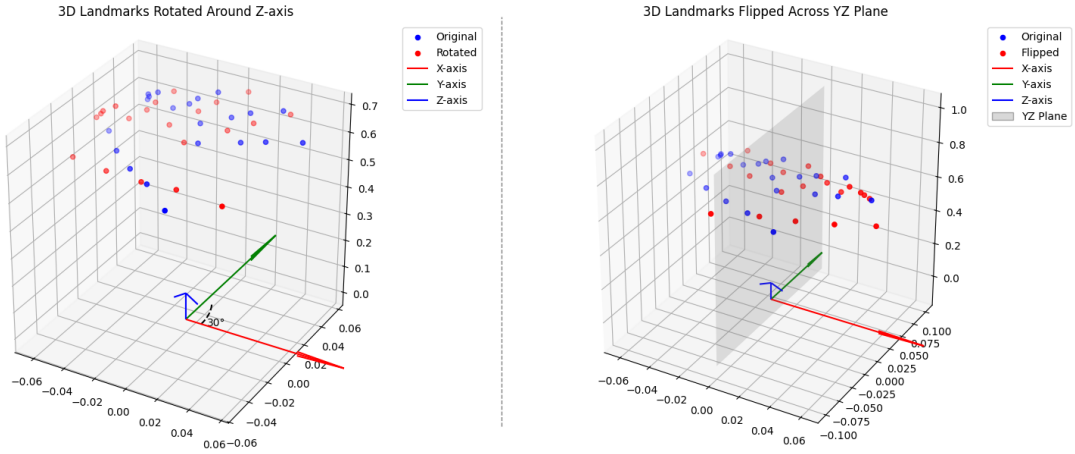


Figure 6: Landmark rotation and flipping.

Figure 7: Image color jitter and crop-resizing.

5.1.3 Hyperparameters and optimization

Some hyperparameters have been tested, including dropout and different loss functions (e.g. Huber loss), but selecting a relatively low learning rate and batch size has proven the most efficient. Again, good results were obtained by utilizing a similar setup to that of Zimmermann et al. [14] and Iqbal et al. [3], using an initial learning rate of 1e-05 and a batch size of 16. Zimmermann et al. [14] drop their learning

rate by an order of magnitude to 1e-06, after 150,000 iterations. Instead, we utilize a learning rate scheduler, dropping the learning rate similarly after the validation loss plateaus for 5 epochs, half of the early stopping trigger. Finally, we use the ADAM optimizer, adapting a per-parameter learning rate according to its gradient history. To date, it's one of the most widely used optimizers in deep learning.

5.1.4 Train, Validation and Test split

We split the full image dataset 80/10/10 into training, validation and test sets, respectively. The data is split randomly with a fixed seed of 42 for reproducibility. To ensure a proper, non overlapping split, we implement the following logic:

$$\begin{aligned} \text{test\_size} &= \lfloor \text{test\_split} \times N \rfloor \\ \text{val\_size} &= \lfloor \text{val\_split} \times N \rfloor \\ \text{train\_size} &= N - \text{test\_size} - \text{val\_size} \end{aligned}$$

For each dataset, a data loader is constructed, allowing us to fetch images from disk batch by batch, as they are needed. To increase the speed when training, we utilize pin memory for faster transfers from CPU to GPU. Additionally, we use persistent workers, avoiding that processes/workers are released between each epoch, speeding up the start of each epoch significantly.

5.1.5 Model checkpoints and early stopping

Throughout the training loop, the validation error is checked, and a checkpoint is stored if it has improved compared to previously the best epoch. The latest best model is also saved to disk along with the following checkpoint statistic:

```
{"val_loss": 0.002715722719865723, "epoch": 101, "plateau": 0}
```

This ensures that the best seen model is always available, and also allows resuming training at a later stage. In addition to checkpoints, we implement early stopping, which stops the training process if the model has failed to improve for 10 epochs. These two strategies help avoid overfitting, save compute resources and are useful in hyperparameter tuning experiments where some configurations might not converge well.

5.1.6 Loss and Training Loop

As we are solving a regression problem, we use Mean Squared Error (MSE) as our loss function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

Our training loop can then be summarized by the following four steps, applied per batch:

(1) Forward pass: $\hat{y} = f(x;\theta)$

(2) Compute loss: $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

(3) Backward pass: $\nabla_\theta \mathcal{L}$

(4) Update parameters: $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \mathcal{L}$

5.1.7 Discussion of model

We found that implementing substantial augmentation described in the sections above helped reduce overfitting. While it increased the time required to train the model, it helped with generalization, reducing the gap between training and validation error. Additionally, augmented models showed more stability during live inference.

This model gives a good starting point and prototype for live exercise feedback, yet there are several areas that can improve its usefulness and robustness. Improving data collection and fine tuning the current model on newly acquired image and landmark data would likely strengthen the model significantly. Doing so would require a reliable tool for labeling new images with landmark data. Here, we could benefit from semi-supervised learning, utilizing existing models, e.g. Mediapipe [1] or MANO [10], to automatically label new images taken for our specific purpose. To acquire more precise labels, manual labeling could also be beneficial, but is more expensive in time and resources.

We reflect further fine tuning and data collection in the perspectives section at the end of the paper.

5.2 Modeling K-means

The second model we are using is K-Means [2, pp. 521]. We use this model to provide students with live feedback on the exercises. To prepare the training of the model, we first collect the teachers instructions in different picture recording sessions. Some sessions were right and and some deliberately wrong as described in section 4.4. This means that we created some clusters of landmark data points where we already knew what clusters we expect the model to find. We will explain each step in more detail in the following sections.

5.2.1 Model Training

To cluster our 1174 self-collected images from the 6 sessions, we utilize a simple KMeans clustering algorithm, provided via `scikit-learn`, with default arguments.

We train the KMeans model on the predictions on the collected images, using the supervised CNN model. Before passing the landmarks to KMeans, we define a hand-centric local coordinate system and transform the 3D landmarks as outlined below.

Let $\mathbf{P}$ be the hand landmark matrix:

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \dots & \mathbf{p}_{20} \end{bmatrix}^\top \in \mathbb{R}^{21\times 3}$$

Where:

- $\mathbf{p}_0$: wrist
- $\mathbf{p}_5$: index base
- $\mathbf{p}_{17}$: pinky base

Then:

X-axis (wrist → index base):

$$\hat{\mathbf{x}} = \frac{\mathbf{p}_5 - \mathbf{p}_0}{\|\mathbf{p}_5 - \mathbf{p}_0\|}$$

Z-axis (normal to palm plane):

$$\hat{\mathbf{z}} = \frac{(\mathbf{p}_5 - \mathbf{p}_0) \times (\mathbf{p}_{17} - \mathbf{p}_0)}{\|(\mathbf{p}_5 - \mathbf{p}_0) \times (\mathbf{p}_{17} - \mathbf{p}_0)\|}$$

Y-axis (completes right-handed system):

$$\hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}}$$

We then transform each landmark using the following 3 steps:

1. **Center the points at the wrist:**

$$\mathbf{P}_{\text{centered}} = \mathbf{P} - \mathbf{p}_0$$

2. **Construct rotation matrix:**

$$\mathbf{R} = [\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}] \in \mathbb{R}^{3\times 3} \tag{6}$$

3. **Rotate into the local frame:**

$$\mathbf{P}_{\text{local}} = \mathbf{P}_{\text{centered}} \cdot \mathbf{R} \tag{7}$$

This alignment allows us to evaluate hands in a consistent, rotation robust manner. If rotation is relevant for a given exercise, rotation can be returned, for one or several axes, before the input is passed to k-means. The full process is shown in figure 8.

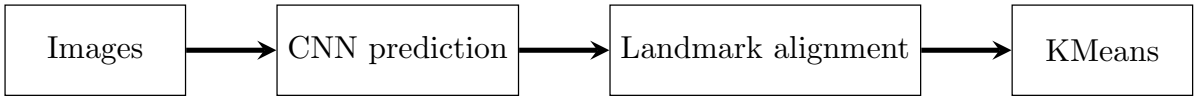


Figure 8: K-means data preparation process.

5.2.2 Selection of Clusters

For the selection of clusters, we already expected 3 clusters, as we had let the teacher record prictures 3 different sessions with 3 different hand positions. But to do this in a more scientific way, we used the elbow method similar to the scree method [2, pp. 514]. The elbow method shows the inertia versus the number of clusters. The inertia is defined as $\sum_{i=1}^{n}\sum_{j=1}^{k}||x_i - c_j||^2$, where x_i is the data point, c_j is the centroid of cluster j and k is the number of clusters. The elbow method shows that the inertia decreases with increasing number of clusters, but at some point it stops decreasing significantly. This is the elbow point, and we can use this to select the number of clusters. Consistent with our expectations, given the input data's 3 states, we find 3 clusters as the elbow point of the model, as shown in figure 9.

5.2.3 Cluster Evaluation

It is promising that the elbow method and our intuition is aligned, but to we want to verify that the centroids are representing the data in a meaningful way. We could risk that the model has found some uninteresting clusters. To evaluate the model, we transform the data to a more intuitive format than the 63 variables we have used before. To reduce the number of variables, we calculate the euclidean distance from the wrist landmark to each of the landmarks. This reduces the number of variables to 20 distances. We can do this for both data points and centroids, so we can plot them together as shown in figure 10. The centroids are shown in red, and the measures are shown in blue. The clusters 0-2 are shown side by side. We get the impression of movement of the hand from left to right, as the hand closes just as we did when creating the dataset in section 4.4.

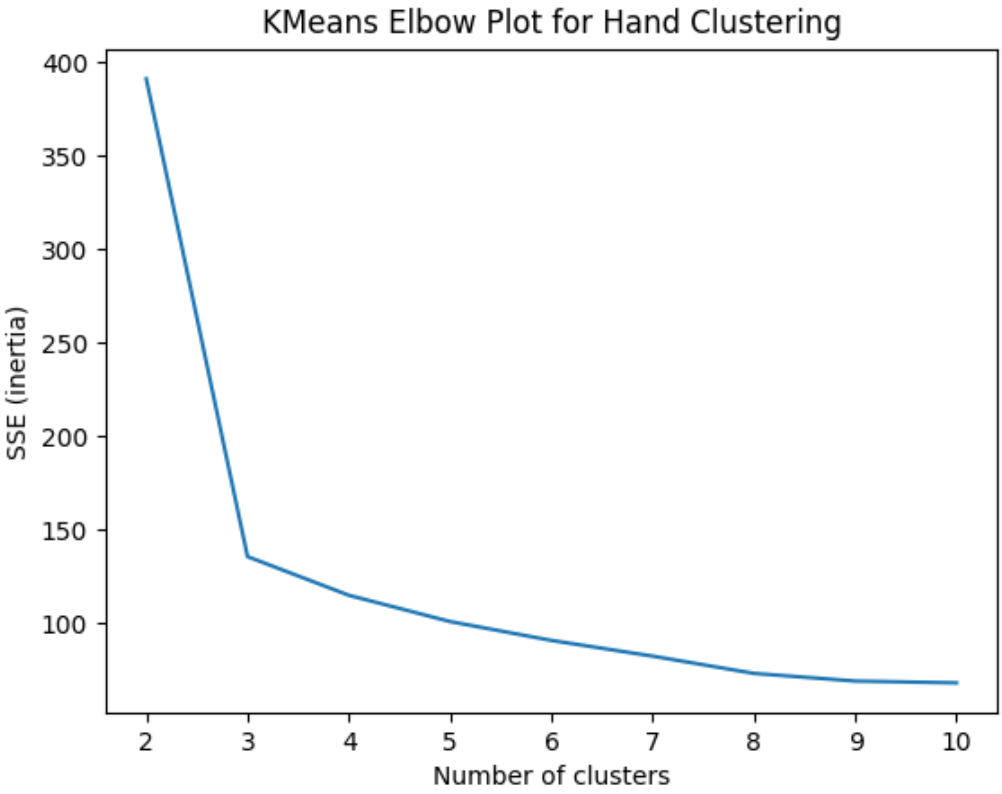


Figure 9: Elbow method showing the inertia versus the number of clusters.

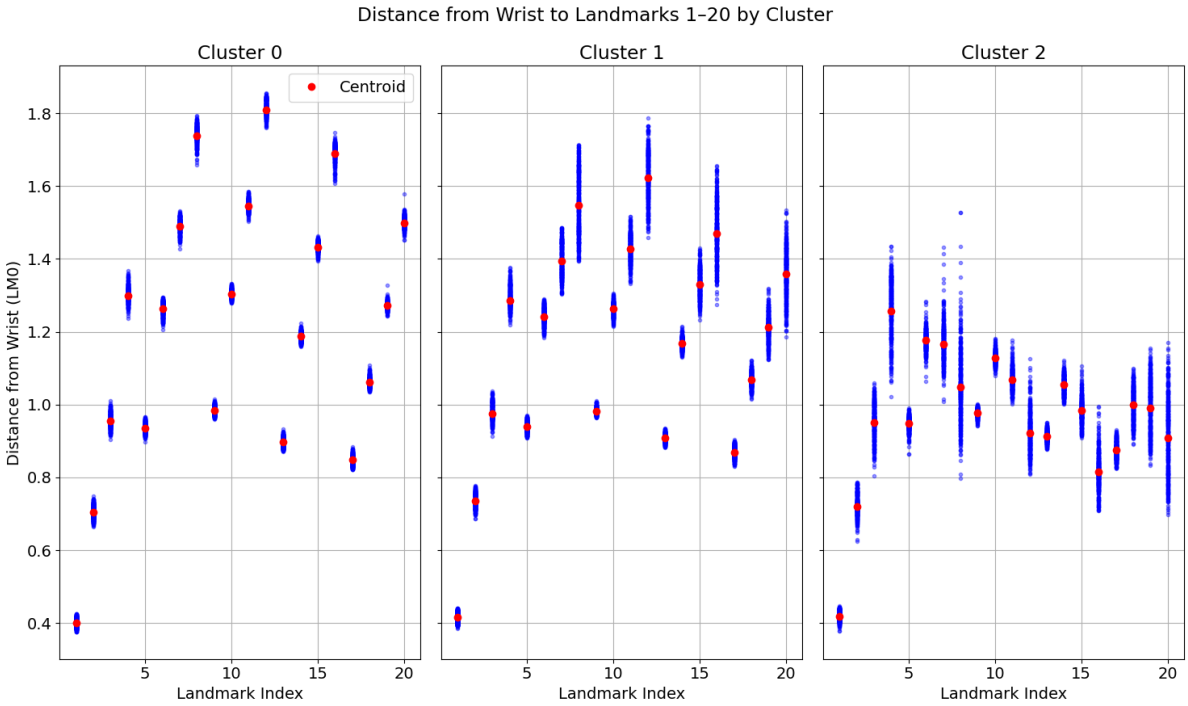


Figure 10: K-means clustering of the dataset. The centroids are shown in red, and the measures are shown in blue. The clusters 0-2 are shown side by side.

5.2.4 Discussion of model

We can see that the model is able to find the clusters we expect. It is also clear that the centroid for each session is aligning nicely with the data points. Exactly how well was not shown, but we will return to this in the Performance evaluation section 6 below. We are aware that the scenario we are testing is relatively simple with only 3 clusters. Complex hand exercises may very well have more clusters, and we have not tested this. A software solution that could incorporate more clusters has not been implemented. Given the potential clinical use of the software, this is a potential complex area to make error free, as clusters may fail. this may lead to a need to create more hand-held central training sets with proper analysis, before these are made available to patients. Our own dataset is limited in diversity (e.g., hand size, lighting, heatly/patient, skin color). Future work should include synthetic or hand photo model variation to improve generalization.

6 Performance Evaluation

6.1 Coordinate Extraction Performance

In our final CNN training loop, we train the model for a total of 111 epochs. The best model is achieved at epoch 101, where the training stops at 111, triggered by early stopping. The learning rate is dropped by an order of magnitude, from 1e-05 to 1e-06, automatically at epoch 89, after the validation loss initially plateaus for 5 epochs.

The training curve shown in figure 11 illustrates a stable training loop, with training and validation loss following each other well. A small additional drop is seen at epoch 89 due to the change in the learning rate, showing the effectiveness of the learning rate scheduler.

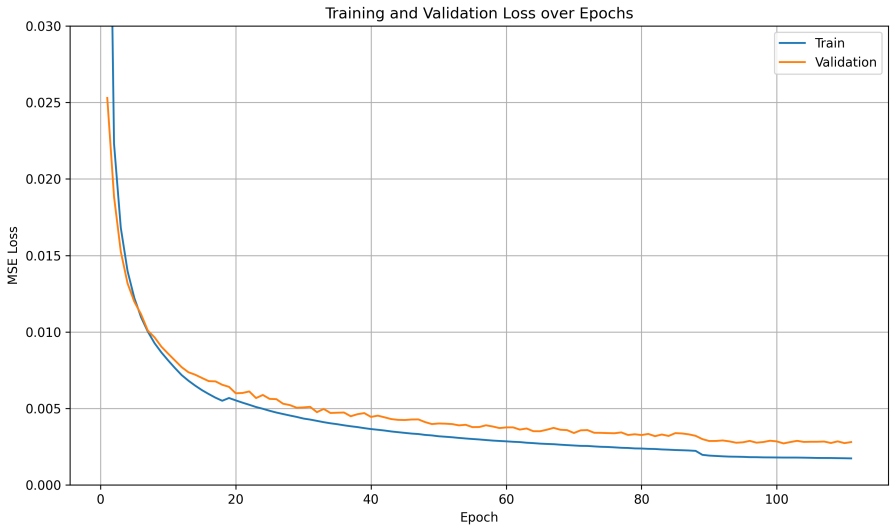


Figure 11: Training and validation loss per epoch.

The following statistics are achieved for the best model achieved at epoch 101:

Dataset	MSE Loss	R^2
Train	0.00172	0.985
Val	0.00272	0.982
Test	0.00271	0.982

Table 2: Performance metrics for the best CNN model: Mean Squared Error (MSE) loss and R^2 for each dataset split.

For each individual landmark (excluding the wrist), we achieve the following R^2 statistics in figure 12.

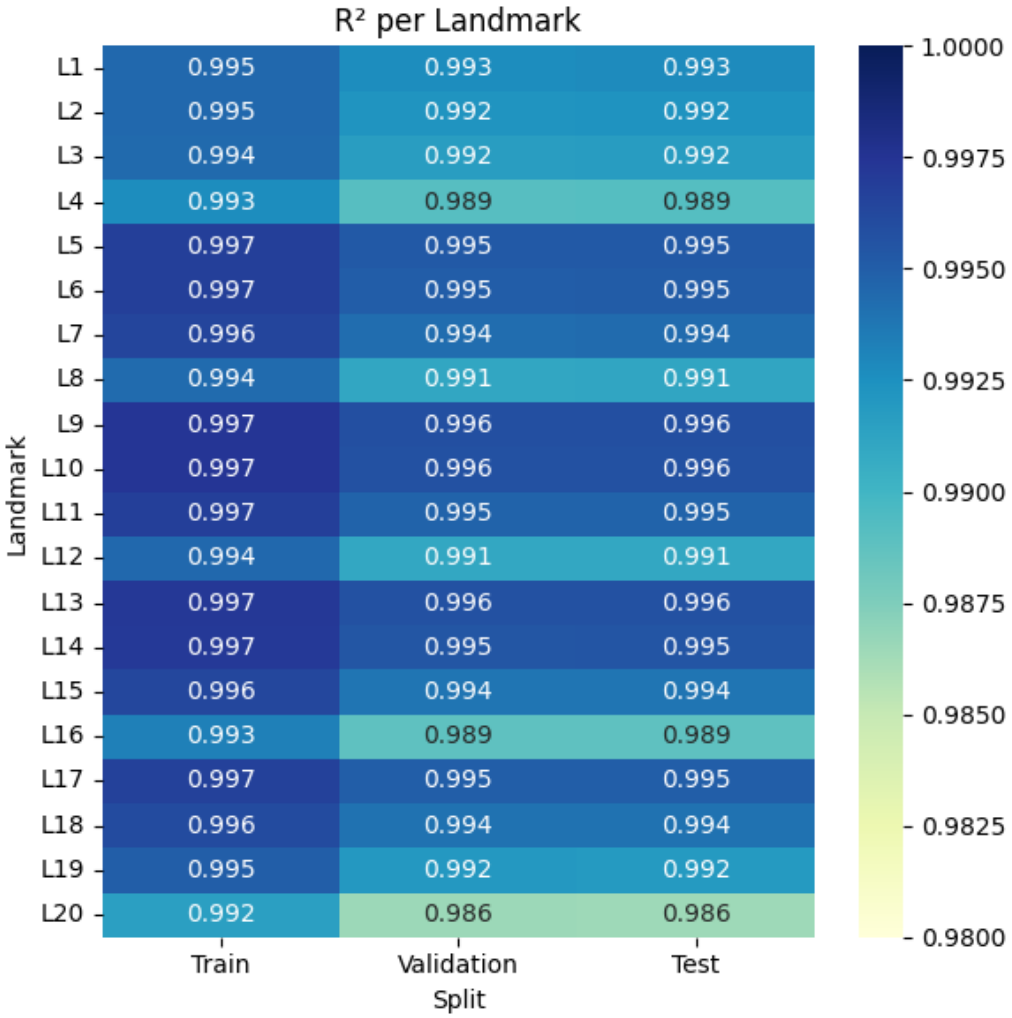


Figure 12: R^2 per landmark.

Each landmark shows a slightly higher value than that of the overall model. This is due to the wrist skewing the overall statistic slightly, due to its near 0 distance values. One solution to combating this, would be to train the model only on the 20 landmarks that are relative to the wrist (origin).

However, the overall within-dataset performance is still very clear, with R^2 values around 0.99 per landmark.

Figure 13 illustrates the model's performance visually by comparing the actual landmarks to the predicted on an arbitrary image from the test set. It is clear that the predictions are visually close to those of the actual landmarks, aligning with the high R^2 values.

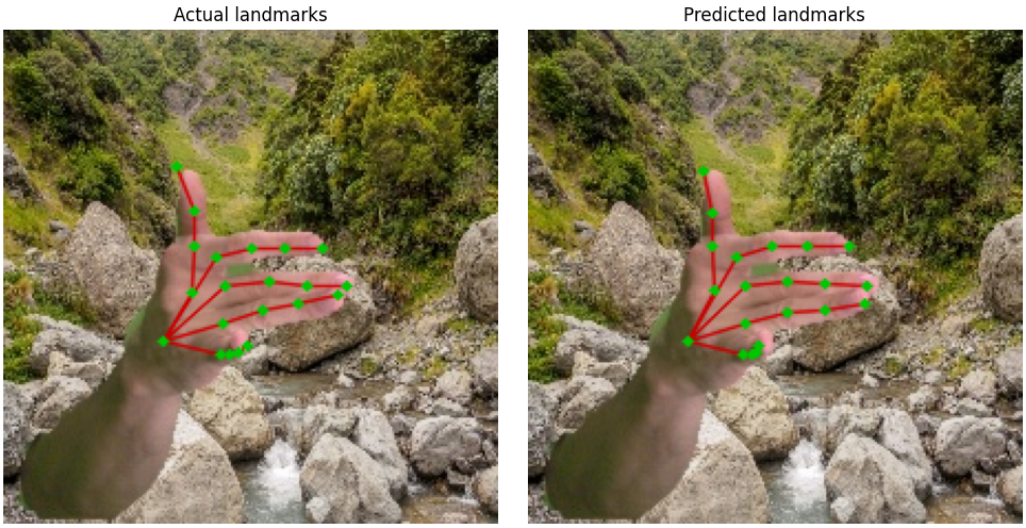


Figure 13: Actual vs. predicted landmarks on a test image.

6.2 Advisor Performance

Early iterations of the k-means model showed some instabilities during live inference, and we needed a way to properly assess the performance of the clusters. The question naturally arises, whether the model is able to give correct advice to the student when using other teacher training datasets.

We will show how important each landmark's distance to the wrist is in the clustering of the dataset. Here we use an approach inspired by PVE (Proportion of Variance Explained), commonly used for Principal Component Analysis (PCA) [2, pp. 510] to show how important each parameter (wrist-landmark distance) is in determining the cluster. For simplicity, we use the name PVE as well as our approach is conceptually very similar.

We compute PVE similarly to R^2, where the sum of squares are computed for each landmark and cluster using the cluster centroids.

Formally:

Let $X \in \mathbb{R}^{n \times d}$ be the matrix of wrist-relative distances, where $d = 20$. Let the global mean be:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

For each cluster $k \in \{0, 1, 2\}$, let C_k be the set of sample indices in that cluster and $c^{(k)} \in \mathbb{R}^d$ be the cluster centroid in 20D space. The total sum of squares (TSS) and residual sum of squares (RSS) for cluster k and feature j are:

$$\mathrm{TSS}_{k,j} = \sum_{i \in C_k} (x_{ij} - \mu_j)^2$$

$$\mathrm{RSS}_{k,j} = \sum_{i \in C_k} \left(x_{ij} - c_j^{(k)} \right)^2$$

Then the proportion of variance explained (PVE) is:

$$\mathrm{PVE}_{k,j} = \begin{cases} 1 - \frac{\mathrm{RSS}_{k,j}}{\mathrm{TSS}_{k,j}}, & \text{if } \mathrm{TSS}_{k,j} \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Intuition would suggest that the most distant parameters are the most important, as they are being moved the most when the hand is closed. The PVE plot is shown in figure 14.

Looking at the PVE plot, we see that the most distant landmarks are indeed the most important in determining the cluster. The plot clearly shows areas of high proportion of variance explained, particularly from the index to the pinky. There are 3 moving landmarks on each finger in the movements we investigate. This aligns well with the nature of the exercise and indicates that the clusters are well-defined and explain the different exercise states well.

We do notice that for cluster 1, i.e. the slightly bent fingers, the cluster's variance explainability is not quite as strong as for the fully stretched and closed hands, for most landmarks. During live inference we also found that the middle cluster was sometimes not captured as well as the two others. One solution to improving this posture would be to collect more input images, as well as fine tune the CNN model to improve its performance. We discuss this further in the perspectives at the end of the paper.

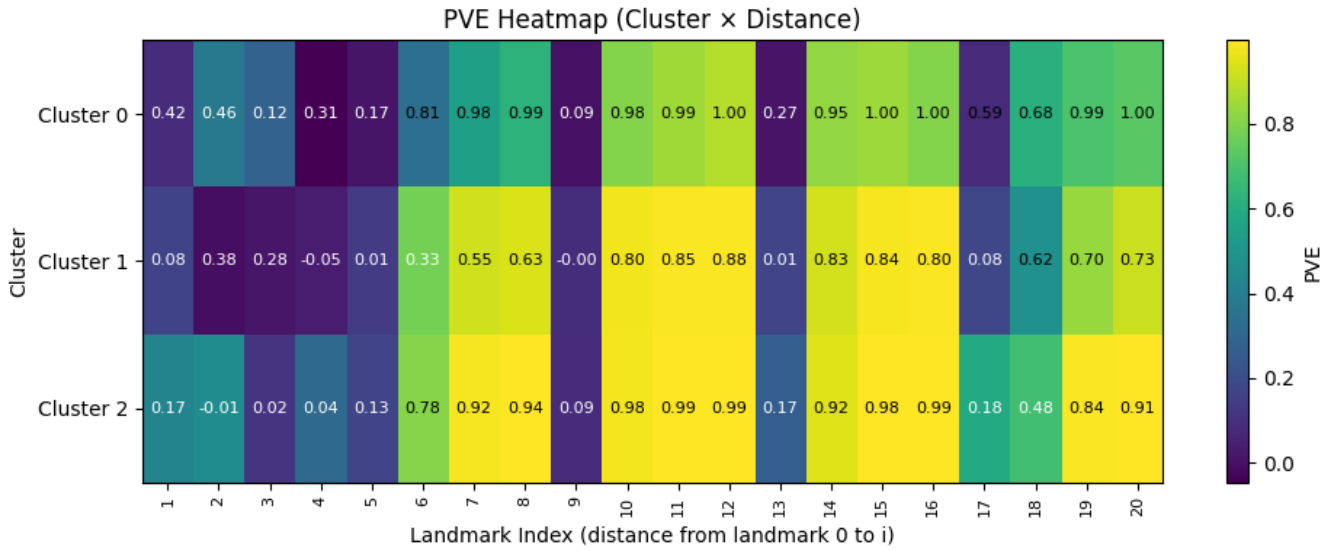


Figure 14: PVE plot showing the importance of each parameter in determining the cluster.

6.3 Hardware performance

To find the best performing way to train our model, we have analyzed hardware and model performance.

To assess raw hardware performance, we benchmarked training time using a fixed subset of the HanCo RGB dataset consisting of 100,000 RGB images with associated 3D hand landmark annotations. Each image is paired with 21 hand joints, each with (x, y, z) coordinates, resulting in a 63-dimensional regression target.

We used a MobileNetV2 model pretrained on ImageNet and replaced its classification head with a linear layer outputting 63 values. Images were resized to 224×224 and normalized using ImageNet statistics. Training was run for 3 epochs with a batch size of 64.

To isolate hardware effects, we kept the model, data, and training settings fixed while varying two parameters:

- The execution device (CPU or GPU)
- The number of threads used in PyTorch via torch.set.num.threads(N)

In our performance test, a given configuration may be run n times. Across those runs, mean (μ) and standard deviation (σ) are computed. Figure 15 displays two single runs using a on the data subset,

using a Tesla M60 to an NVIDIA GeForce RTX 4070Ti SUPER. Despite the Tesla GPU being specifically developed for computational tasks, the RTX outperforms it significantly, as it is a much newer model. Having more cores, as well as tensor cores, while also supporting mixed precision training, significantly speeds up training for the NVIDIA GPU.

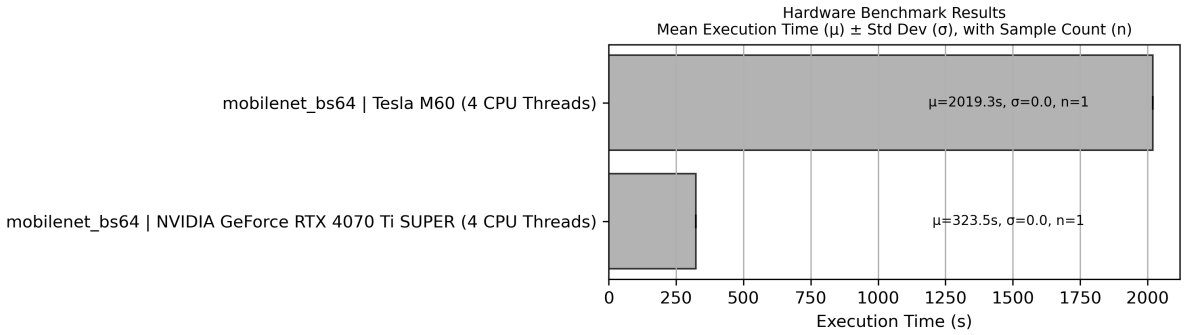


Figure 15: Different CPU/GPU configurations and their performance against a predefined CNN test described further in the text

7 Results

Bringing it all together, we implement the UML use case diagram in figure 1 in a small app, showing how the software works, as shown in figure 16.

On the right hand side of the app, a plot with two aligned hands is displayed:

1. A fixed hand which is the resulting prediction of the reference/teacher image
2. A live plot of the landmark predictions of the student's hand, captured live from the webcam.

The distance between the two hands' landmarks are visualized with a green line for each landmark. Additionally, each landmark point on the teacher hand follows a red-yellow-green gradient, moving from red towards green the closer the student landmark predictions are to the teacher hand.

On the left hand side of the app, is a webcam feed which provides the input for the ResNet50 model, producing the landmark predictions. Additionally, the clusters from the k-means model have been translated to meaningful messages. During the live inference, landmark predictions are passed to the k-means model, and a message is returned based on which cluster is assigned. As displayed by figure 16, a positive message is returned when the student ends up in cluster 0, corresponding to the stretched hand. Similarly, a message prompting the student to stretch their fingers, or stretch them slightly more, is given for clusters 2 (near-open hand) and 1 (closed hand), respectively.

Overall, the software aims to provide a useful and visually pleasing exercise session for the user, to increase motivation when performing an exercise.

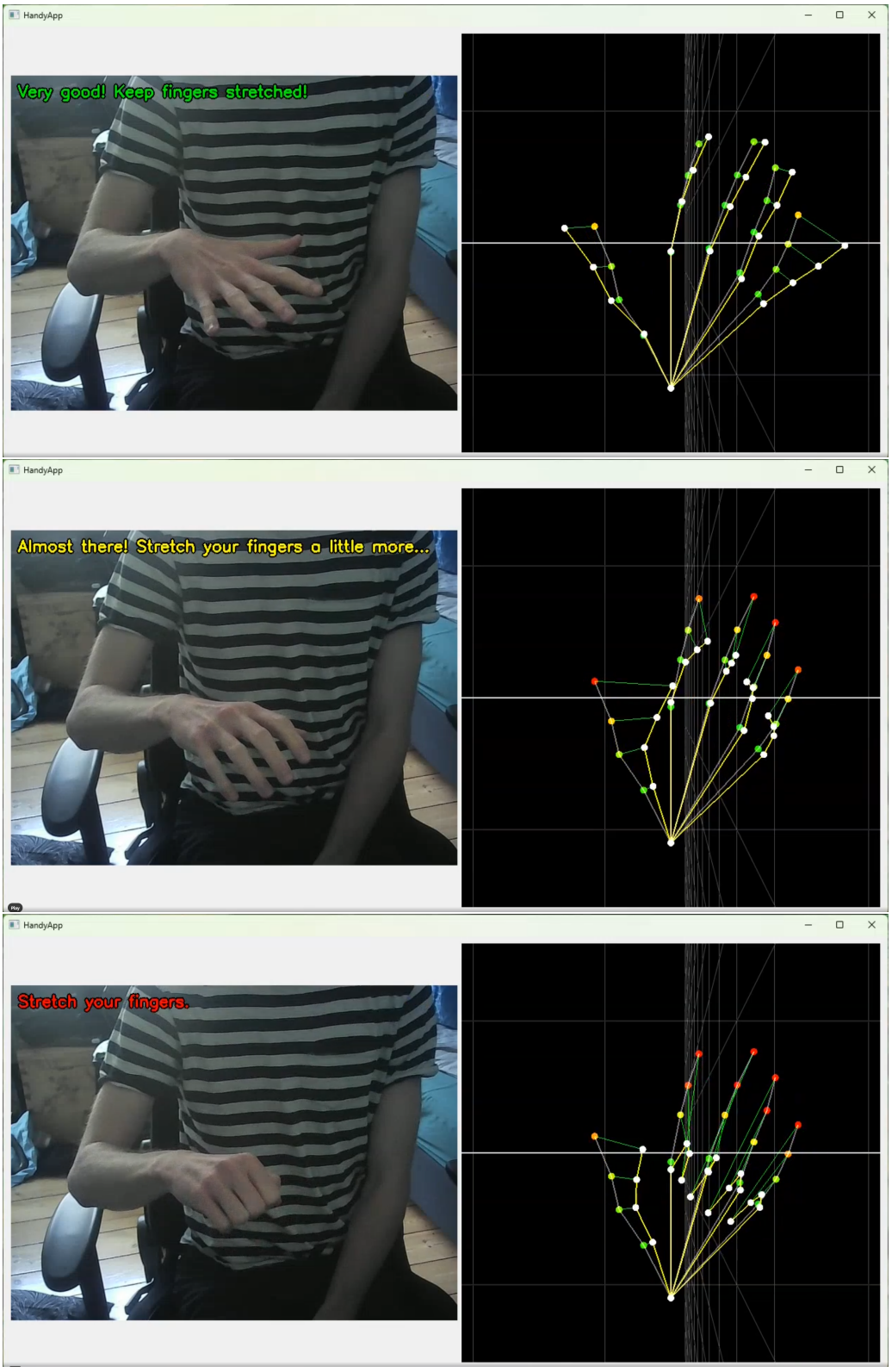


Figure 16: A demonstration of the final software, showing the feedback to the student. The left hand is the students hand, and the right hand is the skeleton hand of landmarks extracted live from the hand on the left. Note the k-means feedback in the upper left corner showing, good, almost good and bad.

8 Discussion and Conclusion

We have shown how we can use machine learning to help CP patients with their exercises. We have trained a CNN model to extract landmarks from images of hands, and a k-means model to cluster the landmark predictions and give feedback to the student. The models are able to give good feedback to the student, and the software is easy to use.

By applying extensive data augmentation, we improve model performance and generalization, giving a within-dataset R^2 value of 98.2% on the test set along with a healthy loss curve and a small gap between training and validation loss. We apply k-means along with the elbow method to select a suitable number of clusters for patient feedback on a finger stretch exercise. The elbow point aligns well with our input data and expectations about exercise states, finding three clusters. We evaluate the k-means clustering by visually examining the distance from the wrist to each of the other landmarks, clearly showing how the clusters move from a stretched to a closed hand. Additionally, we confirm the clusters' explainable value with a PVE computation, particularly highlighting that the clustering explains the variance well around the middle to the edge of the fingers.

The model is built on open source software, and can be used on a simple laptop with a web camera. This opens up for the possibility of using the software at home, and with more advanced exercises in the future, provide therapists with a tool to provide more effective therapy to CP patients and other patients who are in need of rehabilitation. The software has been tested on a small dataset, and we have shown that excellent results can be achieved. The software is not yet ready for production, but it is a good prototype with some promising statistical results.

We have analyzed a single, simple exercise in this paper. It would be natural to extend the experiment to include more exercises prescribed by real life therapists. This would allow the software to provide feedback on a larger set of exercises, and provide more value to the patients. The software could also be extended to include more advanced exercises, such as individual finger movements or rotational hand movements. New exercises would likely require their own specific k-means model, such that clustering and inference is done in the right context.

The software has not been tested on real patients yet, and it remains to be seen if it provides benefits to real patients. This would require a larger dataset and more testing with a larger group of patients including a control group and compliance documentation required by authorities, such as the US Food and Drug Administration (FDA) or the European Medicines Agency (EMA). Other regulations, such as the EU AI Act may also apply, because the software is processing patient information, which is considered sensitive personal data according to EU GDPR. This will likely require more complex threat modeling, testing and documentation than we have done in this paper. Examples of possible proper solutions are suggested by [7].

9 Perspectives

We already discussed future improvements in section 8, but many potential improvements not covered by our models and dataset could be explored.

According to [8] acquiring well-curated datasets for CP poses a significant challenge. With a tool like ours, it could be extended to allow data collection for scientific use. Many patients have an interest in sharing their data to improve therapies. Collecting such datasets could not only improve our software, but could enable other scientific areas to improve as well. To collect a more diverse set of images, users could voluntarily contribute with their image data to help fine tune models. This would involve a voluntary upload to a database with a TLS encrypted connection and strong certificate based authentication. This should be handled with caution, however. Potentially, this process could be facilitated by a therapist,

approving that the user holds the right abilities to provide solid, useful feedback before the data is collected or used for further development.

Other hand pose models, such as Mediapipe, utilize bounding box detection [1], which significantly helps performance of the landmark detection model. Building off existing object detection models, such as YOLO [4], might help the prediction significantly by first locating the bounding box of the hand, limiting the input to our ResNet50 model to just the hand's bounding box. Using this approach will also help transform the live feed into images that are similar in structure to the HanCo dataset, likely improving the overall performance and live inference.

Finally, model speed is also critical when dealing with live inference, especially since the software must run smoothly on a simple laptop. ResNet50 is a relatively heavy model, and running it live may be a challenge on some devices. Sacrificing some model performance for a smoother experience by replacing ResNet50 with a lighter model, such as MobileNetV2 [6] as was used in our hardware performance test, might be worthwhile. Combined with fast bounding box prediction and fine tuning, using a lighter model will likely provide an overall better experience for the user than using a heavier backbone. This could also open up the possibility of running the software on mobile devices.

References

[1] Andrey Vakunov Andrei Tkachenka George Sung Chuo-Ling Chang Matthias Grundmann Fan Zhang, Valentin Bazarevsky. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009. Accessed: 2025-02-16. URL: `https://hastie.su.domains/ISLP/ISLP_website.pdf`.

[3] Umar Iqbal, Pavlo Molchanov, Thomas Breuel, Juergen Gall, and Jan Kautz. Hand pose estimation via latent 2.5d heatmap regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 118–134, 2018. URL: `https://arxiv.org/abs/1804.09534`.

[4] Glenn Jocher and Ultralytics. YOLOv5. `https://github.com/ultralytics/yolov5`, 2020.

[5] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[6] Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen Mark Sandler, Andrew Howard. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2019.

[7] A. Mohammed, B. Mohammed, and C. Mohammed. A survey on machine learning in cybersecurity: A comprehensive review. *arXiv preprint arXiv:2405.10295*, 2024.

[8] A. Nahar, S. Paul, and M. J. Saikia. A systematic review on machine learning approaches in cerebral palsy research. *PeerJ*, 2024.

[9] PyTorch Team. Resnet - pytorch hub, 2024. Accessed: 2025-05-29. URL: `https://pytorch.org/hub/pytorch_vision_resnet/`.

[10] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6), November 2017.

[11] United Nations. Sustainable development goal 3: Good health and well-being, 2015. Accessed: 2025-02-16. URL: `https://www.un.org/sustainabledevelopment/health/`.

[12] Jacob Asbæk Wolf and Peter Asbæk Skøt. Hand training motion tracking, 2025. Accessed: 2025-05-28. URL: `https://github.com/PSkot/hand-training-motion-tracking`.

[13] Christian Zimmermann, Max Argus, and Thomas Brox. Contrastive representation learning for hand shape estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. URL: `https://lmb.informatik.uni-freiburg.de/projects/hanco/`.

[14] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russell, Max Argus, and Thomas Brox. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. URL: `https://lmb.informatik.uni-freiburg.de/projects/freihand/`.