Deep Learning for Real-Time Hand Joint Tracking

Adithya Palle

Northeastern University
palle.a@northeastern.edu

Abstract

Given the proliferation of computers and the emerging demand for contactless computer control in the realm of alternate reality, our objective is to utilize real-time hand gesture tracking as a means to replace conventional input devices. We developed an AI agent capable of interpreting hand pose from a camera feed whose outputs can be translated into computer commands. Using the HanCo dataset, we trained a 3-level U-net architecture by regressing "heatmaps" derived from 21 key hand joint coordinates with an Intersection over Union (IoU) loss. This model achieved swift inference (12.51 ms on CPU, 3.16 ms on GPU) and high accuracy (4.2% Average Keypoint Error). The model's balance between simplicity and efficiency outperforms both simpler and more complex architectures. Experimental results also highlight the IoU loss's superiority over mean squared error for heatmap regression. Beyond immediate applications in real-time gesture tracking, this technology holds promise for contactless computer interfaces, paving the way for more convenient and intuitive human-computer interaction.

Introduction

Amidst the proliferation of computers and screens in the modern world, the reliance on conventional control interfaces like keyboards, mice, or touchscreens has become increasingly cumbersome. In certain domains such as alternate or virtual reality, the necessity for contact input poses a challenge as it disrupts the immersive experience. Consequently, the issue of employing machine vision to track hand gestures and eliminate the need for physical input has garnered significant attention. While one approach involves directly learning gestures from hand images, requiring specific labeled data for each gesture, a more robust strategy is to train a model to predict the hand pose graph from a given frame, enabling subsequent extraction of gestures through algorithmic processes.

The central challenge lies in developing a high-performing model capable of translating these predicted poses into precise computer commands, encompassing mouse movements, clicks, and essential keyboard inputs. For instance, the envisioned model could discern the cursor's position based on the right-hand index finger, recognize a touch between the index finger and thumb as a left-click, or evoke a virtual touch keyboard upon detecting a fully open palm. This paper specifically focuses on the development of a machine learning (ML) model to derive hand poses from images. The model's predicted pose graph can then be further processed to derive predefined commands and instructions in future work. One simple example would be associating the index finger's node position with the thumb's node to indicate a left-click, but such implementations are beyond the scope of this paper. This technological innovation holds great promise for the realm of contactless computers and screens, offering a more convenient and immersive approach to computer control.

Background

In the pursuit of establishing an intuitive and contactless interface between users and computers through hand gesture recognition, it is imperative to contextualize this effort within the broader landscape of computer vision and the evolution of deep learning models. The historical trajectory of Convolutional Neural Networks (CNNs) serves as a foundational narrative, showcasing the pivotal role these models have played in advancing the capabilities of computer vision systems, especially in tasks related to image analysis and pattern recognition.

The inception of CNNs traces back to the late 20th century, with luminaries such as Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner laying the groundwork in the 1990s. LeCun's LeNet-5, designed for handwritten digit recognition, represented a pioneering effort by demonstrating the effectiveness of convolutional layers in extracting hierarchical features from images(LeCun et al. 1998). However, it wasn't until the 2010s, buoyed by enhanced computational power and the availability of extensive labeled datasets, that CNNs experienced a transformative resurgence.

In the field of artificial intelligence and machine learning, Convolutional Neural Networks (CNNs) have become a cornerstone, particularly within the domain of computer vision. A CNN is a specialized type of deep neural network designed to process and extract hierarchical features from visual data. Unlike traditional neural networks, CNNs employ convolutional layers, enabling them to automatically learn spatial hierarchies of features from input images.

The fundamental operation in a CNN is the convolution operation, denoted as $*$, which involves applying a filter (also known as a kernel) to the input image. Mathematically, the convolution operation is expressed as:

$$(I*K)(i,j) = \sum_{m}^{M}\sum_{n}^{N} I(m,n)\cdot K(i-\frac{M}{2}+m, j-\frac{N}{2}+n) \tag{1}$$

Here, $(I*K)(i,j)$ represents the result of the convolution at position (i,j) for a $M \times N$ kernel K being applied to image I. The convolution operation captures local patterns in the input, allowing the network to identify features like edges, corners, and textures.

Additionally, CNNs often include pooling layers, typically max pooling, to downsample the spatial dimensions of the feature maps. The max pooling operation is defined as:

$$MP(I,i,j) = \max_{m\in[m-p,m+p],n\in[n-p,n+p]} I(i\times p+m, j\times p+n)$$

Here, p represents the pool size and the operation takes the maximum element of each region encompassed by a single pool and sets the pixel in the resulting image to that value.

Through the use of convolutional and pooling layers, CNNs excel at capturing local patterns in input data, progressively building a hierarchy of features. This hierarchical feature learning is crucial for recognizing complex patterns and structures within images, resembling the hierarchical nature of visual information processing in the human brain. The success of CNNs in computer vision applications has established them as a powerful tool for tasks such as image classification, object detection, and image segmentation.

Related Work

Recent advancements in the field of pose estimation have seen the emergence of several notable augmentations of the traditional CNN, each contributing unique methodologies to address the challenges posed by this intricate task.

Pfister et al. proposed a Convolutional Neural Network architecture tailored for human pose estimation in videos with multiple available frames(Pfister, Charles, and Zisserman 2015). The key contributions of their work include a deeper network for regressing heatmaps, spatial fusion layers for implicit spatial modeling, utilization of optical flow to align heatmap predictions from neighboring frames, and a parametric pooling layer to combine aligned heatmaps into a pooled confidence map. This architecture outperforms alternative models, including those using optical flow solely at input layers, directly regressing joint coordinates, and predicting heatmaps without spatial fusion. The results showcase its superiority over existing approaches, particularly on challenging datasets like Poses in the Wild and FLIC.

Newell et al. introduced a novel convolutional network architecture named the "stacked hourglass" network for human pose estimation(Newell, Yang, and Deng 2016). This architecture processes features across all scales and consolidates them to capture diverse spatial relationships associated with the body. The use of repeated bottom-up, top-down processing, along with intermediate supervision, proved critical in enhancing network performance. The architecture achieves state-of-the-art results on benchmark datasets such as FLIC and MPII, surpassing recent methods in the field.

Chernytska addressed the growing demand for 3D hand pose estimation, particularly in the context of VR/AR applications, utilizing single RGB cameras (Chernytska 2019). The work tackled challenges associated with algorithmic complexity and the absence of comprehensive datasets. Convolutional Neural Networks were employed, demonstrating that the direct heatmaps method is optimal for 2D pose estimation, while vector representation excels in 3D pose estimation. The thesis highlighted the significance of data augmentations, even in synthetic datasets, to improve performance on real data. Furthermore, it showcased the feasibility of training neural networks on large-scale synthetic datasets and fine-tuning them on smaller, partly labeled real datasets. The models exhibited the ability to predict 3D keypoint locations for simple poses, even in the absence of real 3D labels.

Methods

While prior research has explored the application of deep-learning-based computer vision for pose estimation, these studies often overlook a crucial aspect—real-time analysis of pose. In this study, I aim to build upon existing work by approaching the problem with differing architectures and employing new data-based methods, all designed to model hand pose keypoints. The primary objective is to enable real-time analysis, addressing a gap in the current literature.

Dataset

The dataset utilized for our modeling, named HanCo, comprises an extensive collection of over 850,000 labeled RGB images capturing diverse hand poses from varying camera angles. To enhance realism, background randomization has been applied using segmentation masks(see Figure 1). The dataset includes intrinsic and camera matrices, along with X, Y, and Z positions, providing comprehensive information for each image. For the modeling process, approximately 56,000 images have been selected for modeling, with 6,000 images reserved for validation, and 2,000 for final testing. This dataset configuration ensures a robust and well-validated foundation for the development and evaluation of our hand pose estimation model.

Data Preprocessing

Pfister et al. and Chernytska et al. highlight the efficacy of heatmap regression over direct keypoint coordinate learning. Heatmap regression involves transforming a single (x,y) coordinate pair into an $n \times n$ heatmap, where only a localized, smooth region around (x,y) exhibits non-zero values. This technique streamlines the learning process by offering rich gradient information to models and demonstrating greater tolerance for erroneous predictions during gradient descent.

The model processes a 128x128x3 RGB image, converting it into a 128x128x21 image that represents a heatmap for each joint. These heatmaps are generated using the Heatmap Generation algorithm (Algorithm 1), which derives them from the (x,y) points. Subsequently, the model reverts the

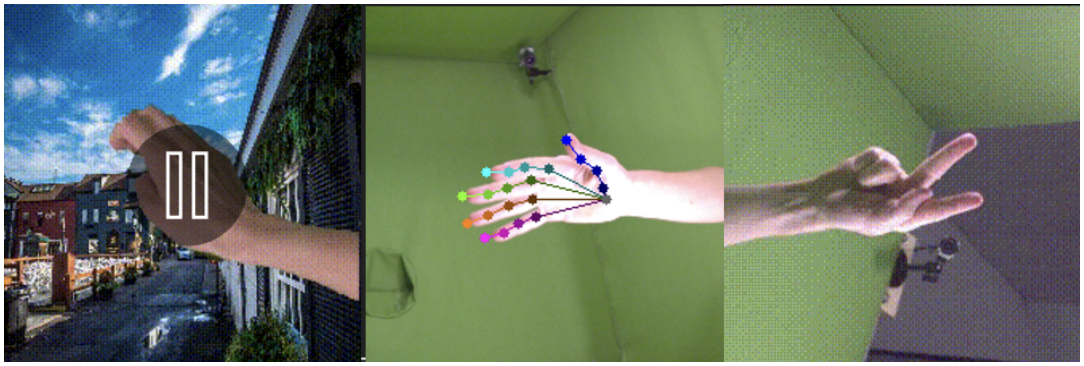

Figure 1: Sample Images from HanCo Dataset(University of Freiburg 2021). The leftmost example shows the result of background randomization, the middle panel displays the keypoint labels, and the right panel displays the raw RGB image.

Algorithm 1: Heatmap Generation

Input: 2D keypoint tensor K, image size vector S
Output: Tensor of heatmaps

Let w, h be image width, height from S.
Initialize H as an empty vector.
for i from 0 to $K.s(0) - 1$ **do**
    Extract X, Y from K at index i.
    Create M as a $w \times h$ matrix with zeros
    $M(Y, X) \rightarrow 1.0$.
    Apply Gaussian Blur ($\sigma = 51$) to M.
    Normalize M by dividing each pixel by its max value.
    Append M to H.
end for
return H.

Algorithm 2: Keypoint Extraction from Heatmap

Input: Heatmaps tensor H, device D
Output: Tensor of keypoints

// Calculate sums of each heatmap
$S \leftarrow H.s(1, 2)$
$H \leftarrow H/S.v(S.s(0), 1, 1)$ // Standardize heatmaps to sum to 1
$R \leftarrow H.s(1)$ // Row sums of shape (21, 128)
$C \leftarrow H.s(2)$ // Column sums of shape (21, 128)
$R \leftarrow R \cdot \{0, 1, 2....127, 128\}$ // dot product row sums by range
$C \leftarrow C * \cdot\{0, 1, 2....127, 128\}$ // dot product row sums by range
return $cat(\{R, C\}, 1)$

heatmap to an (x, y) coordinate by employing the Keypoint Extraction(Algorithm 2) algorithm, which involves averaging the heatmap.

In terms of preprocessing the X data (the input RGB images), we simply resize the images to be $128x128$ for faster inference. Then, we standardize each of the RGB channels of this image with the channel mean and standard deviation for all the images in the training set.

In addition to these techniques, we enhance the dataset's diversity and, consequently, the model's generalizability by incorporating background randomization into the RGB images. The dataset includes images with both randomized backgrounds and those without, and we utilize both variations in both training and testing phases. This strategy aims to expose the model to a broader range of scenarios, fostering robust performance across different visual contexts.

Training

Deep Learning is employed to train a Convolutional Neural Network model for predicting joint positions. This involves utilizing stochastic gradient descent to iteratively adjust the weights of a neural network, mapping input data to an output.

Given the substantial volume of data in the Hanco dataset, training on all images in each epoch is impractical due to computational and time constraints. As a solution, we opted to work with a random sample of 56,000 images from the dataset—48,000 for training, 6,000 for validation, and 2,000 for testing. We employed randomized minibatch gradient descent, a variant of gradient descent that updates model weights iteratively across multiple batches within each epoch.

To streamline the training process, we set a batch size of 48 and constrained the number of batches the optimizer sees in each epoch to 50. This limits the processing to $48 \times 50 = 2400$ images per epoch, randomly selected from the larger subset of 48,000 training images. This approach ensures exposure to the majority of the diverse set of images during training while avoiding having to process all 48,000 every epoch. While the upper limit for training epochs was set at 1,000, we implemented early stopping by monitoring the validation loss. If the validation loss did not improve in the last 10 epochs every epoch, every training was halted. The learning rate was initially set to 0.1 but was halved if the training loss had not improved for 20 epochs.

The training was conducted on Northeastern University's Discovery Cluster, utilizing an NVIDIA Tesla V100 SXM2 32GB GPU and Intel Xeon Gold 6132 2.6 GHz CPUs.

Modeling

Since the dimensions of the input image and output heatmap are identical, we use a U-net convolutional neural network which will encode the input image and then decode it back into the heatmap while employing residual connections to remember information about the input image during the decoding process. U-net has excelled in segmentation tasks(Ronneberger 2017) and has also been extended for hand pose estimation(Chernytska 2019) with promising results. In short, U-net employs "convolutional blocks", which are sequences of convolutional layers and activation and normalization layers, to first encode the input image into a rich feature space and then decode this feature map by concatenating activation from the decoding step. A detailed visualization can be seen in Figure 2.

A combination of C++ and Python to write all the code for this project. C++ was used for early modeling and data preprocessing efforts, later iterations employed Python code due to convenience and compatibility with certain open-source tools. The backend framework used for building and training all models was PyTorch, for which the C++ frontend, LibTorch, was used.

Metrics

We experiment with both Mean Squared Error(MSE) and Intersection over Union (IoU) loss functions to train the model. MSE loss simply finds the average square error between the cells of the predicted heatmap and the expected heatmap, while IoU takes the intersection of the predicted join heat region with the actual join heat region and divides it by the union area of the two regions. Since higher IoU indicates that the predicted joint regions have a strong overlap with the expected joint regions, we want the optimizer to maximize, hence the IoU loss is $1 - IoU$. Equations 2 and 3 show the mathematical formulations for the MSE and IoU losses respectively defined for the loss per $n \times m$ heatmap.

$$L_{MSE}(y, y^*) = \frac{1}{nm} \sum_{i,j}^{n,m} (y_{ij} - y^*_{ij})^2 \tag{2}$$

$$IoU(y, y^*) = \frac{\sum_{i,j}^{n,m} (y_{ij} y^*_{ij})}{\sum_{i,j}^{n,m} (y^2_{ij} + (y^*_{ij})^2 - y^*_{ij} y^*_{ij})} \tag{3}$$

$$L_{IoU}(y, y^*) = 1 - IoU(y, y^*)$$

Another metric we use for analysis is the average error between predicted (x,y) keypoint coordinates and actual (x,y) keypoint coordinates. This will be computed by getting the average Euclidean distance between predicted (extracted from Algorithm 2) and expected keypoints for each keypoint, averaging these values, and then dividing the difference in the averages for the expected and actual by the image size (128). This metric will be referred to as the average keypoint error (AKE). Previous experiments in 2D keypoint estimation(see Chernystka et al.) have been able to reduce error slightly below 5%, so we aimed to achieve even better performance on a more general dataset in HanCo. The inference speed of the model will also be timed for analysis of feasibility of deployment in real-time video settings.

Project Summary

To explicitly define the problem, we have an input space X of RGB images represented as $3 \times 128 \times 128$ tensors and an output space Y of 2D keypoint coordinate locations. Each Y sample is a 2×21 tensor representing x and y coordinates for 21 indexed key points on the hand (see Figure 3). We indirectly learn a mapping from $X \rightarrow Y$ by first converting the coordinate key points into heatmaps(Algorithm 1), learning those heatmaps by applying gradient descent on U-Net with the HanCo dataset, and "averaging" the heatmaps (Algorithm 2) to extract the (x, y) predictions for each joint.

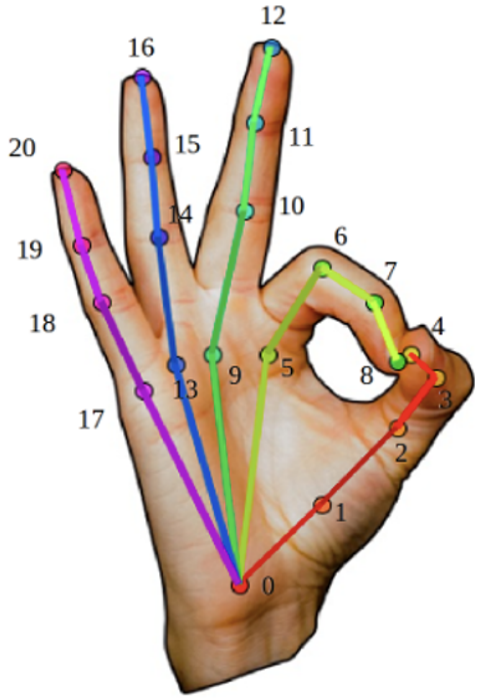

Figure 3: Indexed hand keypoints used for modeling.

.

Experiments

Varying Losses

First, we explore the impact of different loss functions on the model performance. We consider Mean Squared Error (MSE) and Intersection over Union (IoU) as the two different loss functions.

MSE Loss Initial attempts to utilize Mean Squared Error (MSE) loss for training keypoint tracking models proved unsuccessful. Despite rapid convergence (Figure 4), achieving a low MSE loss did not ensure accurate keypoint regression, as seen in Figure 5. The U-net model, with a validation MSE below 0.001, did not yield precise keypoint regression, highlighting the inadequacy of MSE for comparing joint heatmaps. This limitation is evident in the test predictions, with an average keypoint error (AKE) of 45.6%,

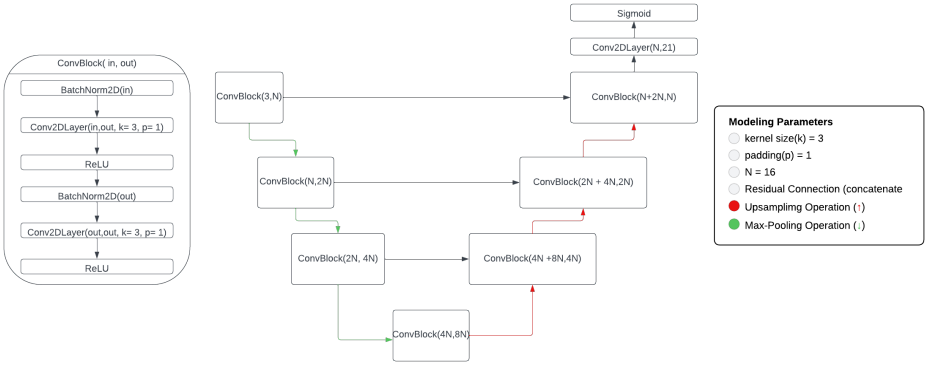

Figure 2: U-net architecture visualization including the definition of Convolutional Block (ConvBlock)

likely attributed to MSE's focus on individual pixels rather than comprehensive joint regions.

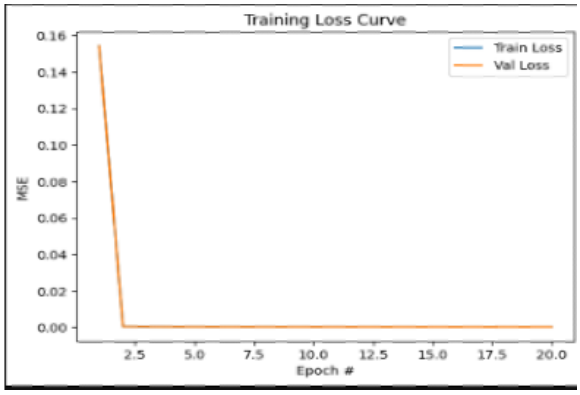

Figure 4: MSE Loss curve for 3-level U-Net

.

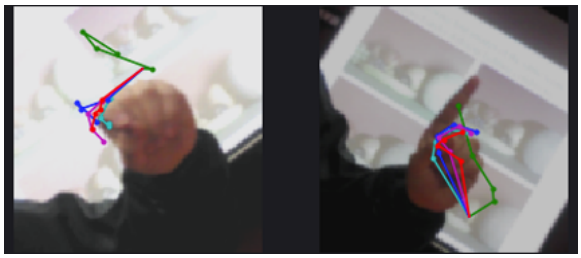

Figure 5: Plotted keypoints predicted of U-net trained with MSE loss

.

IoU Loss The IoU loss demonstrated superior performance compared to the MSE loss in this modeling task. Although the IoU loss took significantly longer to converge and did not reach zero, the predicted keypoint locations (Figure 10) were considerably more accurate than those obtained with the MSE loss. Figures 6, 7, and 8 illustrate the convergence of the IoU loss for different architectures (further differences will be discussed in the next section). The loss convergence is notably smoother than that observed for MSE (Figure 4), and the model continues to learn even in the later stages of the training process. The downward trajectory of both curves indicates that additional training data may further enhance the model's ability to achieve better loss, at least in training. Subsequent discussions on loss in this paper will specifically refer to the IoU loss, as MSE experiments were discarded due to unsatisfactory initial results.

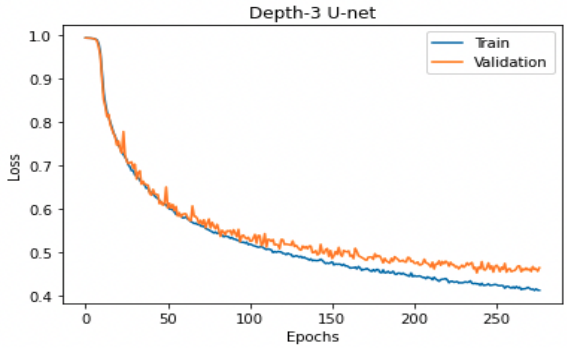

Figure 6: IoU Loss curve for 3-level U-net

.

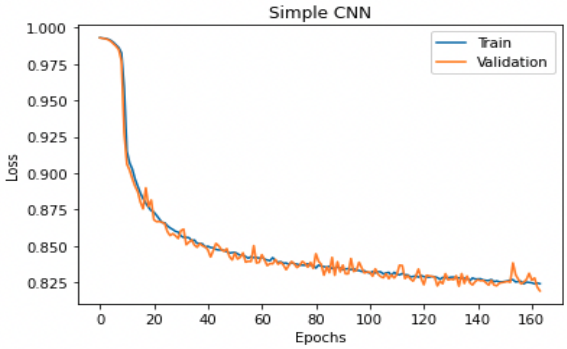

Figure 7: IoU Loss curve for simple CNN

.

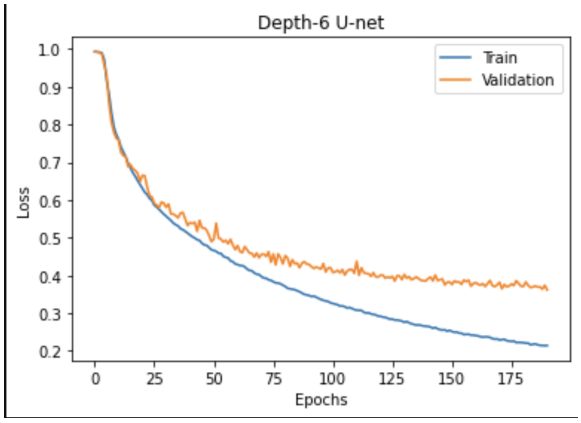

Figure 8: IoU Loss curve for 6-level U-net

.

Varying Architectures

We investigated the influence of different model architectures on the overall performance. We compare the following architectures which are listed in order of increasing complexity: a simple CNN, a 3-level U-Net, and a 6-level U-Net.

Simple CNN The simple CNN architecture comprises two Convolutional Blocks, as defined in Figure 2, applied consecutively. In the initial block, the 3-channel image is transformed into a 21-channel activation stack, and the subsequent block further processes this stack to generate another 21-channel activation stack. This final stack is then input into a convolutional layer, and a sigmoid function is applied to produce 21 heatmaps. The convergence of IoU loss for the simple CNN is depicted in Figure 7, indicating that the model struggles to achieve an IoU lower than 0.8 before the validation loss plateaus.

Despite the high loss, the model exhibits an average keypoint error (AKE) of 11.6%, and the predictions display discernible patterns rather than being entirely random (Figure 12). Occasionally, the model can capture the shape or structure of the hand; however, it encounters challenges in accurately modeling the hand's size. The predicted keypoint graphs often appear small and densely clustered, reflecting difficulties in precisely estimating the hand's proportions.

3-level U-Net The architecture used in this experiment is exactly that seen in Figure 2. The loss curve, as illustrated in Figure 6, underscores the model's proficiency in generating higher Intersection over Union (IoU) heatmaps compared to the simple CNN. Although the test loss for this model (0.469) markedly outperforms that of the simple CNN, the discernible gap between the training and validation curves in Figure 6 suggests a subtle overfitting tendency, unlike the simple CNN, which does not exhibit this behavior. This potential overfitting could be addressed by a strategic reduction in the stack depth (N) across the network, thereby curbing the overall parameter count.

Furthermore, the test AKE was a very impressive 4.2%, slightly exceeding performance in experiments conducted in (Chernytska 2019). Examining the predicted key points in Figure 10, it becomes evident that this approach significantly outperforms the simpler model in accurately delineating the key points. In the majority of predictions, the hand's structural representation is precise, although minor discrepancies persist in the exact joint locations.

For our specific use case, centered around gesture tracking, the precision in predicting the hand's structural configuration holds greater significance than achieving pinpoint accuracy in individual (x, y) positions for each keypoint. Because the relative positions of keypoints are reasonably accurate, this model's performance remains sufficient for our intended application.

6-Level U-Net The 3-level U-Net depicted in Figure 2 can be generalized to an N-level U-Net. For each block in the encoding (downwards) half, excluding the first block, it is evident that the input channel parameter for the convolutional block at level i is $2^{i-1} * N$, where i is the index of the level (starting at index 0), and the output channel is simply $2^i * N$. In the decoding (upwards) half, there are $(2^i + 2^{i+1}) * N$ input channels and 2^i output channels. With these observations, we can create a general N-level U-Net, where each level's ConvBlock is defined with the parameters above. For this experiment, we use the 6-level version.

Due to time constraints on GPU nodes on Northeastern's Discovery cluster, the model's loss did not fully converge as it exceeded the 8-hour time limit after 190 epochs. Regardless, the checkpointed version of the model surpassed the simple CNN and 3-level U-Net in IoU and AKE significantly. The model was able to converge to a training IoU loss of 0.220, while the test IoU loss was 0.363, indicating overfitting. The AKE on the test data was 3.3% for this model, a significant improvement over the 3-Level U-Net and the results of (Chernytska 2019). Comparing Figures 10 and 11, it is evident that the 6-Level U-Net is more accurate in predicting keypoint locations, even for occluded hand images. In terms of predictive ability, this architecture is superior to the other two, and likely has room for improvement given that training was cut short.

Inference Speed Analysis

With varying architectural complexity comes varying inference speed. For our use case, we want to be able to apply these models to each frame of a video feed to derive hand structure for gesture extraction in real-time without significant delay. For the architecture described previously, the simple CNN takes on average 7.5 ms per image, the 3-level U-net takes 12.51 ms, and the 6-level U-net takes 97.18 ms. The above times include time for shrinking and standardizing the input image. A 60 FPS camera feed running on a separate process from the model would feed 1 frame every 16.67 milliseconds to the model, so the simple CNN and 3-level U-net architecture would be able to process each frame without missing the next on average. If we wanted to process the model in the same thread as the camera feed, this would introduce a delay between reading frames due to the forward pass of the model, resulting in a reduction of frame reading speed. For a 60 FPS video, the simple CNN would reduce the FPS to 41.32 FPS, and the frame rates would be reduced to 34.28 FPS and 8.78 FPS for the 3-level and 6-level U-nets

respectively.

The inference speeds are further reduced if the model is run on the GPU, becoming 1.45 ms for the simple CNN, 3.16 ms for the 3-level U-net, and 5.51 ms for the 6-level U-net. This means all models would be able to process a frame parallel from the camera feed without missing subsequent frames. In terms of FPS reduction, these models on the GPU would take the 60 FPS feed to 54.96 FPS for the simple CNN, 50.44 FPS for the 3-level U-Net, and 45.09 FPS for the 6-level U-Net. Clearly, the 6-Level U-net is only usable for real-time processing on a device with a GPU.

Overall, the 3-level U-net stands out as the ideal candidate for real-time gesture tracking due to its reasonably fast inference speed and sufficient accuracy.

Conclusion

We have identified a convolutional neural network architecture that excels in providing swift inference and highly accurate hand pose estimation, making it well-suited for real-time gesture-tracking applications. The chosen model adopts a 3-level U-net architecture and was meticulously trained on a 60,000-sample subset of the HanCo dataset. This dataset employs background randomization to diversify the images by presenting hands in various backgrounds. While the HanCo dataset provides (x, y) coordinate positions, we opted to transform these coordinates into smooth joint "heatmaps" that the model learns to regress. The model's parameters were optimized using mini-batch gradient descent with an Intersection over Union (IoU) loss. Remarkably, our model achieves a satisfactory predictive accuracy (4.2% Average Keypoint Error) coupled with impressive inference speeds (12.51 ms on CPU and 3.16 ms on GPU), making it ideal for real-time gesture tracking applications. The U-net model strikes a balance, outperforming simpler models like the basic CNN (which achieves only 11.6% Average Keypoint Error) in predictive ability and surpassing more complex architectures like the 6-level U-net in terms of inference speed. This demonstrates the efficacy of U-net's residual connections and encoder-decoder architecture for the nuanced task of pose estimation. Additionally, while exploring mean squared error as an alternative loss function for heatmap regression, we discovered that models trained on this loss exhibited significantly worse Average Keypoint Error, despite achieving impressively low MSE values. This finding underscores the superiority of IoU as a loss function for regressing heatmaps in the context of pose estimation. The experimental results are summarized in Figure 9.

For future work in this domain, expanding the dataset during model training could enhance the generalizability of the model. The current models were trained on a subset of the HanCo dataset, and exploring the use of additional data may further improve performance. An intriguing avenue beyond the scope of this paper involves leveraging machine learning to derive gestures directly from the hand pose graph. With the current approach, gesture detections would require hard-coding for specific hand pose graph configurations. However, by employing machine learning on the keypoint locations predicted by our model, it may be possible to predict gestures such as mouse-clicks or keyboard activations, provided a robust dataset is available for training.

Additional Information

The code for this project and its previous versions can be found at https://github.com/4di03/PalmPilot.

Acknowledgments

I am grateful to Dr. Christoper Amato for leading the Foundations of Artificial Intelligence course for which this project was conducted. I would also like to thank Northeastern University for allowing me to use the Discovery Cluster to attain the computing resources necessary for this project.

Figure 9: Model Performance Metrics

Model Type	Test IoU Loss	Average Error	CPU Runtime (ms)	GPU Runtime (ms)
Simple CNN	0.823	11.6%	7.53	1.45
3-level U-Net	0.469	4.2%	12.51	3.16
6-Level U-Net	0.363	3.3%	97.18	5.51

References

Chernytska, O. 2019. *3D Hand Pose Estimation from Single RGB Camera*. Master's thesis, (Ukrainian Catholic University).

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 2278–2324. AAAI Press.

Newell, A.; Yang, K.; and Deng, J. 2016. Stacked Hourglass Networks for Human Pose Estimation. *arXiv preprint arXiv:1603.06937*.

Pfister, T.; Charles, J.; and Zisserman, A. 2015. Flowing Convnets for Human Pose Estimation in Videos. In *2015 IEEE International Conference on Computer Vision (ICCV)*.

Ronneberger, O. 2017. Invited talk: U-net convolutional networks for biomedical image segmentation. In *Informatik aktuell*, 3–3.

University of Freiburg, L. f. M. L. . D. M., Institute of Computer Science. 2021. HanCo Dataset. https://lmb.informatik.uni-freiburg.de/resources/datasets/HanCo.en.html.

Appendix

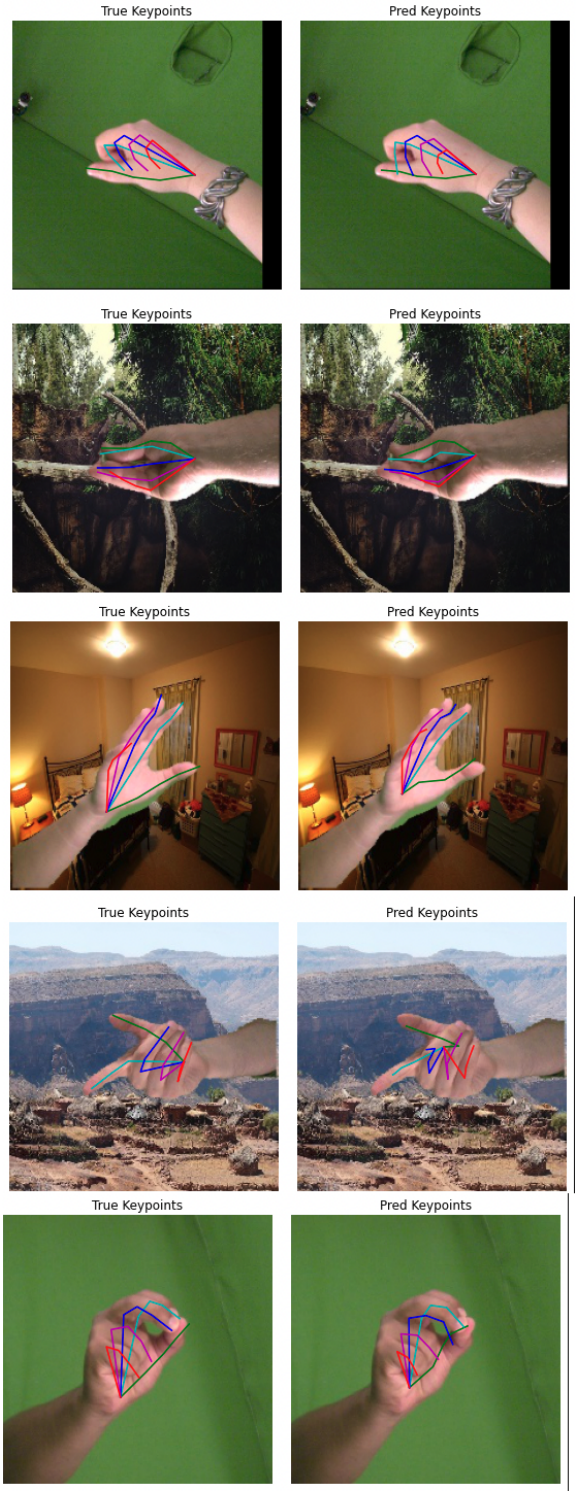

Figure 10: Keypoint predictions for 3-level U-net

.

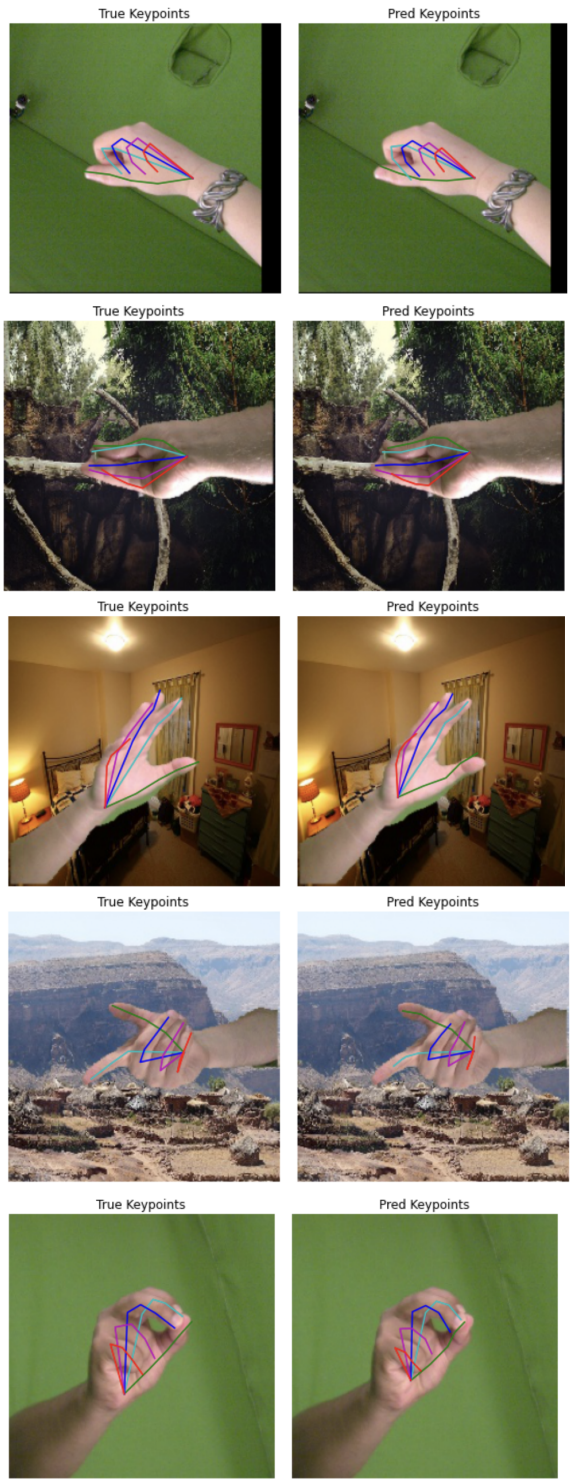

Figure 11: Keypoint predictions for 6-level U-net

.

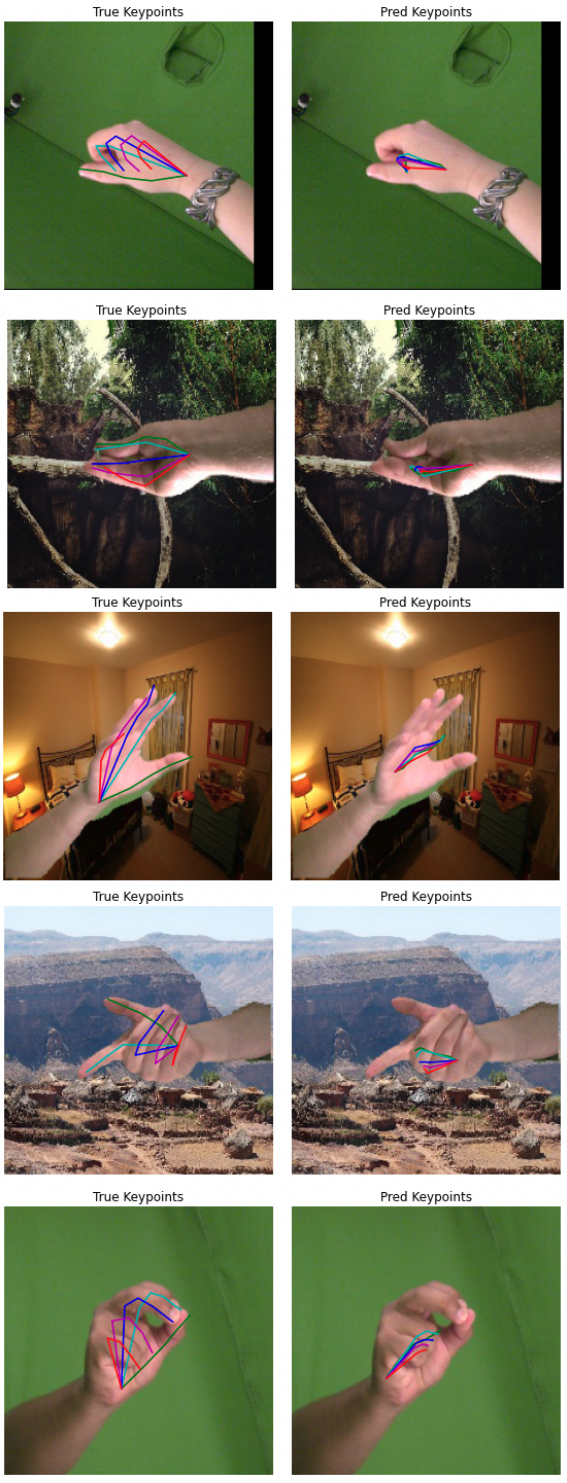

Figure 12: Keypoint predictions for simple CNN

.