DHBW
Duale Hochschule
Baden-Württemberg

Real time gesture recognition using deep neural networks

Max Kiefer, Laurin Tarta, Gregor Boschmann
Department of Computer Science
DHBW Stuttgart
7301175, 9354426, 2259116

Abstract

The ability to accurately recognize and classify gestures has the potential to revolutionize the way we communicate and interact with technology, particularly for individuals with hearing impairments or other disabilities that make verbal communication difficult. In this project, we explore the use of a neural network for gesture recognition with the future end-goal of enabling the real-time translation of sign language. We use a live video feed as input and train the neural network on a dataset of annotated hand gestures. Our results demonstrate the effectiveness of this approach, with an accuracy of 92% on our test set. Our work represents a step forward in the development of sign language translation systems using machine learning and has the potential to facilitate more effective communication between individuals with and without hearing impairments.

1 Introduction

Gesture recognition is an important and rapidly growing area of research within the field of machine learning, with the potential to revolutionize the way we communicate and interact with technology. In this project, we focus on the recognition of individual gestures as a first step towards the real-time translation of sign language using a neural network approach.

The ability to accurately recognize and classify gestures has the potential to facilitate more effective communication between individuals with and without hearing impairments, and to create more accessible educational and entertainment content for the deaf and hard of hearing community. Sign language, in particular, is a rich and expressive form of communication that is used by millions of people worldwide, yet it remains largely inaccessible to those who do not know it.

While complete sign language translation is a complex and challenging task, the recognition of individual gestures is a crucial component and a necessary starting point. By accurately recognizing individual gestures, it becomes possible to build systems that can recognize and interpret longer sequences of gestures, eventually leading to the translation of entire conversations in sign language.

In this project, we use a hand tracking solution that utilizes an ML pipeline consisting of three models working together.

The first model is a palm detector that operates on a full input image and locates palms via an oriented hand bounding box. This model is responsible for identifying and localizing the hands within the image, providing a region of interest for the second model to operate on.

The second model is a hand landmark model that operates on the cropped hand bounding box provided by the palm detector and returns 21 2.5D landmarks. These landmarks provide detailed information

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

about the shape and posture of the hand, allowing for the tracking of hand gestures and movements with high precision.

In addition to the palm detector and hand landmark model, our gesture recognition system includes a third model that takes the landmarks as input and detects the gesture being made. This model is responsible for classifying the gestures based on the input features provided by the hand landmark model, and it plays a crucial role in the overall performance of the system.

By combining these three models, our hand tracking solution is able to operate efficiently and accurately, providing real-time tracking of hand movements and gestures. We believe that this approach has the potential to enable more intuitive and natural forms of human-computer interaction, and we look forward to exploring its potential applications in the future.

Our results demonstrate the effectiveness of neural network approaches for the recognition of individual gestures, and suggest that this approach has the potential to be extended to more advanced sign language translation systems in the future. By using a live video feed as input, our system is able to operate in real-time, enabling the translation of sign language as it is being used. Overall, our work represents a significant step forward in the development of sign language translation systems using machine learning.

2 Related work

There have been numerous previous attempts to address the problem of gesture recognition using machine learning, and a wide range of approaches have been proposed. Some common approaches include the use of traditional machine learning algorithms such as decision trees and support vector machines, as well as more modern techniques such as convolutional neural networks [1]. A large portion of previous work requires specialized hardware, e.g. depth sensors like Microsoft Kinect [2]. However, we follow a lighter approach, which does not require extra hardware and works in real time even on less powerful devices. Many of these approaches have achieved good results on various gesture recognition tasks, and there is ongoing research to improve the accuracy and efficiency of these methods. Some clever and good approaches have used multi-modal data, such as combining visual and audio information, to improve performance [3].

Overall, the state-of-the-art in gesture recognition is highly dependent on the specific task and dataset being used. For some tasks, such as the recognition of static hand gestures, the performance of machine learning algorithms is already quite good and approaches the performance of humans. For other tasks, such as the recognition of dynamic hand gestures or sign language, there is still room for improvement and ongoing research.

3 Dataset and Features

In this machine learning paper, we used a dataset consisting of 256705 samples. Due to the time and computational constraints, we limited the number of samples used for the majority of development in this project to 40000. The resolution of the images in the dataset is 1920×1080. For testing the performance of the model, we took a variety of pictures with the goal of improving the performance. We captured images with different angles, distances, backgrounds, and lighting setups to create diversity in the test images. The final model can then more easily map the coordinates on the hands and detect the gestures in real life situations. We especially used pictures that are particularly challenging in order to increase our accuracy as much as reasonably possible. This process was quite time-consuming, and so we also included some images from the standard dataset. The test-set consists of 1057 images. To make sure that these images were not used in the training, we removed them from the training-set. The idea was to evaluate the model's ability to generalize to new and unseen data. This helped us to evaluate the robustness and the real-world performance of the model. The data used for training and testing the model was sourced from https://github.com/hukenovs/hagrid.

Plotting the hand landmarks in 3D is just one way of visualizing the data and understanding the structure of the hand landmarks. As you can see in figure 1, the wanted features get extracted and the other attributes do not get analyzed. The plotting of the landmarks itself is not necessarily required for the gesture recognition process. It was done since plotting the data can help you understand the distribution and patterns in the data, and to identify any potential issues or anomalies. In terms of

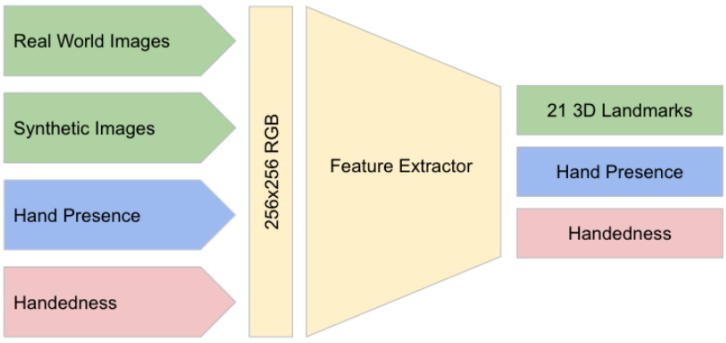

Figure 1: Multiple pictures get extracted and only chosen features are used [4]

the gesture recognition process, the hand landmarks would be used as input features to a machine learning model, rather than being directly used for recognition.

The model learns to identify patterns in the hand landmarks that correspond to different gestures, and use these patterns to make predictions on live data. One hand consists of 21 hand landmarks, each of which is represented by coordinates in 3D space (x, y, z). This works faster and is easier than actually absorbing the information from hands themselves. By mapping coordinates on the hands and then predicting the gesture the problem gets simplified and the learning problem decreases. The result of the coordinate mapping can be seen in figure 2.

In order to preprocess the data, two functions are used in this machine learning project. The process_image_with_hands function takes a file path to an image and returns the coordinates of the hand landmarks detected in the image. It does this using the mp_hands module, which is a library for detecting hand landmarks in images. The function flips the image around the y-axis for correct handedness output, converts the image from BGR to RGB, and then processes the image using the mp_hands module to detect hand landmarks. If no hand landmarks are detected, the function returns None. If hand landmarks are detected, the function iterates over them and converts them to a NumPy array of coordinates before returning the array. The plot_hand_landmarks function takes a NumPy

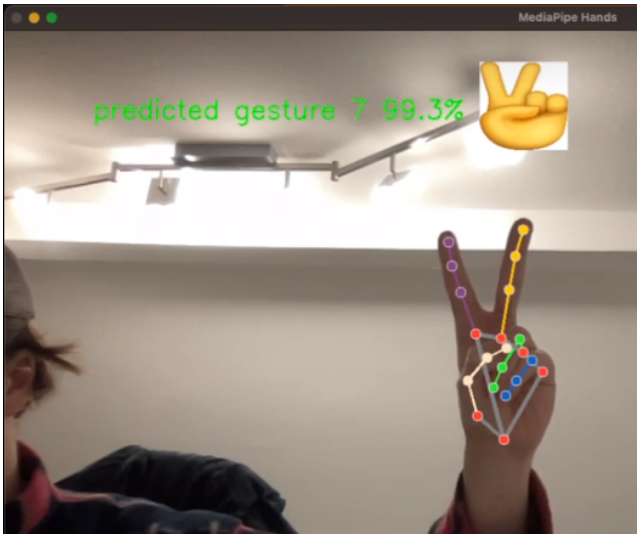

Figure 2: Coordinates mapped on hand and displayed

array of hand landmark coordinates and plots them in 3D. It also plots the different finger joints and

the connections between them. The function creates a figure, adds a subplot to it, and then plots the points and lines representing the hand landmarks. It sets the title of the plot and displays it to the user.

4 Methods

We utilized a Bayesian Hyperparameter Optimizer (BHO) to identify the most suitable set of hyperparameters. BHO is a technique that uses Bayesian optimization to find the optimal set of hyperparameters for a machine learning model. The optimization process is based on a probabilistic model that represents the uncertainty about the performance of the model given different hyperparameter settings. BHO iteratively selects the next set of hyperparameters to evaluate based on the current model, using an acquisition function that balances exploration and exploitation [5]. To evaluate the performance of the model we used cross-validation. Cross-validation is a technique used to evaluate the performance of a machine learning model. In this technique, the data is divided into a training set, used to train the model, and a validation set, used to evaluate its performance. To implement cross-validation in this project, we used a 66% split between the training and validation sets, meaning that 66% of the data was used for training, and 34% was used for validation. The aim is to find the best set of hyperparameters that maximize the performance of the model on the validation dataset while minimizing the number of evaluations. BHO is especially useful when the number of evaluations is limited or when the search space is large and complex, as it can be more efficient and robust than grid or random search.

The batch size, epochs, hidden layers, neurons per layer and existence of dropout and batch normalization are all hyperparameters that affect the performance of a machine learning model. BHO was used to experimentally determine the optimal set of hyperparameters.

The batch size determines the number of samples that are processed before the model's weights are updated. Larger batch sizes can lead to faster training times, but may also result in worse generalization to new data. We tested batch sizes of 32, 64, 128, 256, 512, and 1024 to see how they impacted the model's performance.

The number of epochs specifies the number of times the model is trained on the entire dataset. More epochs can lead to better model performance, but also increase the risk of overfitting. We tested epochs values of 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 to determine the optimal value for our model.

The number of hidden layers and the number of neurons per layer can also impact the model's performance. Deep neural networks with more hidden layers and neurons may be able to learn more complex relationships in the data, but also have a higher risk of overfitting. We tested hidden layer configurations of 1, 2, and 3 hidden layers, with neurons per layer values of 16, 32, 64, 128, and 256.

We also evaluated the use of batch normalization and dropout as regularization techniques to reduce this overfitting problem. Batch normalization normalizes the activations of each layer, which can improve the training of deep networks. Dropout randomly sets a fraction of the input units to zero during training, which can also help prevent overfitting. We tested both batch normalization and dropout, with both enabled and disabled.

We have tested our models accuracy with the three different activation functions for 20000 and 40000 test samples. Both tests ended in the same result showing that the ReLU activation function has the highest accuracy as seen in figure 3.

The activation function used in our model was the rectified linear unit (ReLU) function. There are several reasons why the rectified linear unit (ReLU) activation function may be preferred over the hyperbolic tangent (tanh) or sigmoid activation functions for a gesture recognition machine learning model. The ReLU activation function is a simple, non-linear function that can be implemented efficiently. It has only one parameter (the threshold at which the activation is set to zero), which makes it easy to tune and use. It is also faster to compute than the tanh or sigmoid functions, which is beneficial when training large models or working with large datasets. The ReLU activation function has been shown to work well in many deep learning models and can improve the model's generalization performance. This means that the model is more likely to perform well on new, unseen data.

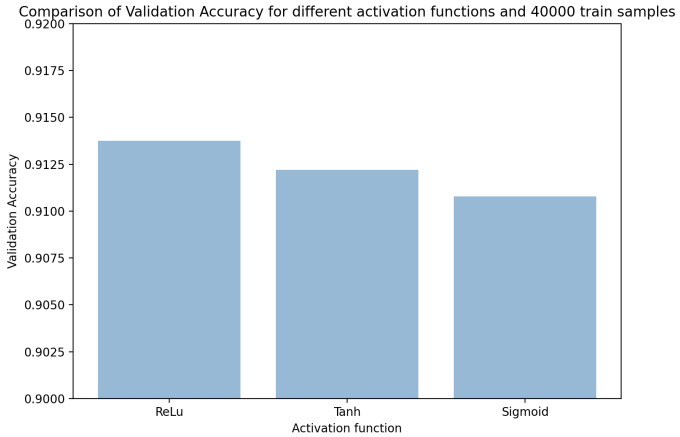

Figure 3: Test accuracy based on different activation functions

We used a dropout rate of 0.2, which means that 20% of the input units are randomly set to zero during training. This helps to prevent overfitting by reducing the dependence of the model on any particular feature.

Finally, we used the Adam optimizer to train our model. Adam is an adaptive learning rate optimization algorithm that has been widely used in deep learning. It adjusts the learning rate of the model based on the gradient of the loss function, which can help the model converge to a good solution more quickly.

The loss function used in our model was categorical cross-entropy, which is the most common choice for classification tasks with multiple classes. It measures the difference between the predicted class probabilities and the true class probabilities. The categorical cross-entropy loss is computed by taking the negative logarithm of the predicted probability for the correct class. This can be formalized as follows:

$$L = -\sum_{c=1}^{C} y_c \log(p_c)$$

where y_c is the true distribution (vector, which has a 1 for the correct class and 0's for all other classes.) and p_c is the predicted probability for class c. C is the number of classes. The negative logarithm of the predicted probability for the correct class is taken to penalize the predictions that are further away from the true class.

In summary, we carefully chose a range of values for the hyperparameters of our gesture recognition model in order to find the best combination for performance. We also employed regularization techniques such as batch normalization and dropout to reduce overfitting, and used the Adam optimizer and categorical crossentropy loss function to train the model effectively.

5 Experiments/Results/Discussion

After the optimization process with BHO, the following best hyperparameters were found: batch size=256, number of epochs=80, number of hidden layers=2, number of neurons in layer1=64, number of neurons in layer2=128, batch normalization was activated and dropout was activated. The resulting model architecture is shown in figure 6.

We trained the model based on those hyperparameters with all 256705 samples and validated the performance on the test set.The accuracy and loss of the final model over the amount of epochs can be seen in figure 4. The results show that the model performs well with a train accuracy of 91.59% and a test accuracy of 92.05%. This means that the model is able to generalize well to the unseen data, and it is achieving a high accuracy on the test set. The model reaches a macro average precision of 93%, recall of 92% and f1-score of 92%. The corresponding confusion matrix is shown in figure 5

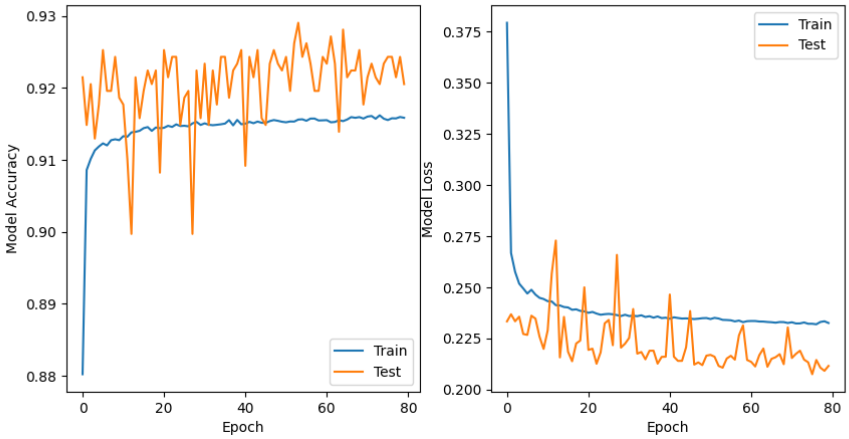

Figure 4: Performance of final model

We also experimented with different activation functions. After evaluating multiple activation functions such as sigmoid, tanh, and ReLU, it was found that the best results were achieved with the ReLU activation function. This suggests that ReLU is particularly suited for this task and dataset. Overall, the results suggest that the model is performing well and it can be considered as a good candidate for deployment.

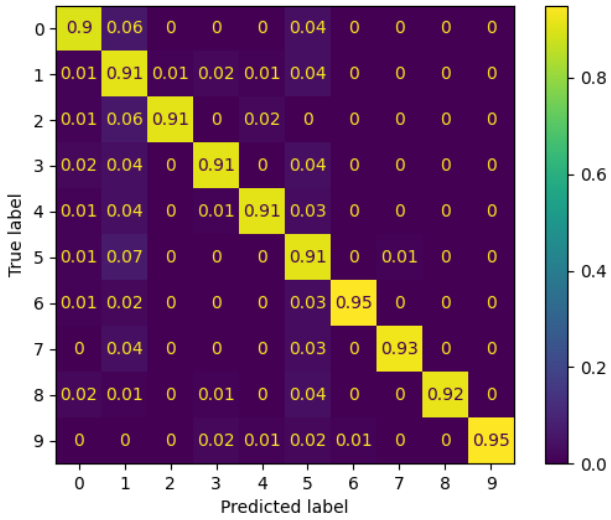

Figure 5: Confusion matrix of final model on test set

In order to create the user interface and to display the corresponding emoji for each recognized gesture, the project used OpenCV library. OpenCV is an open-source computer vision library that is widely used for image and video processing. It provides a wide range of tools for image processing, including functions for image capturing, filtering, and manipulation.

In this project, OpenCV was used to process the video feed of the user's hand gesture, detect and track the hand, and recognize the gesture. The recognized gesture was then matched with an emoji which was displayed on the screen. This way, users can communicate with others through the gestures in a user-friendly and intuitive way. The result of the User Interface can be seen in figure 2.

6 Conclusion/Future Work

In conclusion, our machine learning model for hand gesture recognition achieved high accuracy on a large dataset of hand images. By carefully tuning the hyperparameters and using regularization techniques, we were able to reduce overfitting and improve the model's generalization performance. The Adam optimizer and categorical crossentropy loss function were effective in training the model.

There are many potential applications for this hand gesture recognition model in the future. For example, it could be used to control a user interface or a robotic device using hand gestures, or to assist in sign language translation. Additionally, incorporating temporal information or other feature representations, such as hand poses or shapes, could improve the model's performance on more complex gestures. Testing the model on real-world data and in real-world scenarios will be important in order to fully evaluate its performance and identify any challenges or limitations.

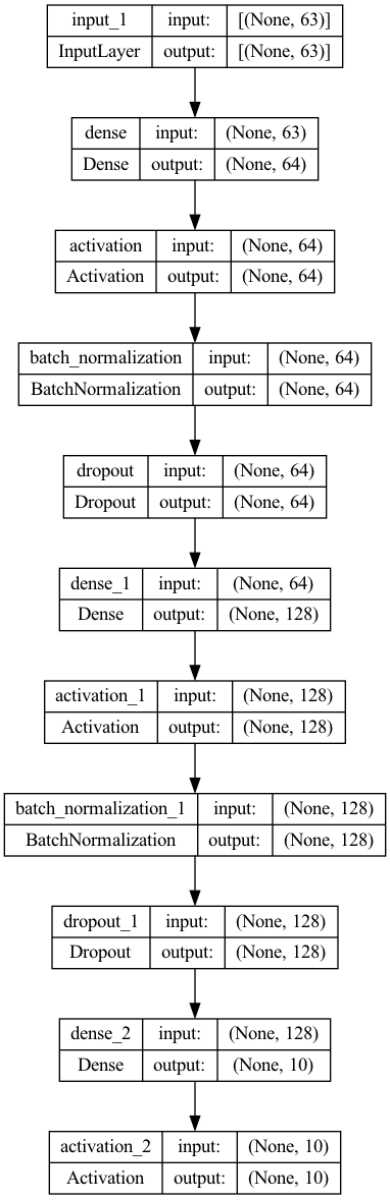

Figure 6: Model architecture of final model

7 References

[1] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, 2012, pp. 411-417, doi: 10.1109/ROMAN.2012.6343787.

[2] Iason Oikonomidis, Nikolaos Kyriazis & Antonis A. Argyros (2011) Efficient Model-based 3D Tracking of Hand Articulations using Kinect.

[3] Vassilis Pitsikalis, Athanasios Katsamanis, Stavros Theodorakis & Petros Maragos (2017) Multimodal Gesture Recognition via Multiple Hypotheses Rescoring *The Springer Series on Challenges in Machine Learning*

[4] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C. L., Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking. arXiv preprint arXiv:2006.10214.

[5] Snoek, J., Larochelle, H., Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.