ASL to text

1st Orfeo Tërkuçi
Faculty of Science
University of Antwerp
Antwerp, Belgium
orfeo.terkuci@student.uantwerpen.be

2nd Jason Liu
Faculty of Science
University of Antwerp
Antwerp, Belgium
jason.liu@student.uantwerpen.be

3rd Anass Hamzaoui
Faculty of Science
University of Antwerp
Antwerp, Belgium
anass.hamzaoui@student.uantwerpen.be

***Abstract*—In this paper we discuss about our process of creating a model that can translate American Sign Language (ASL) to text. We will discuss the data we used, the model we created and the results we obtained. We will also discuss the challenges we faced and the future work that can be done to improve our model.**

I. Introduction

ASL (American sign language) to text conversion is a very important task, as it can help people with hearing disabilities to communicate with people who do not know sign language. While technology has advanced a lot in the last years, there are still many challenges in this field. According to the World Health Organization, Over 6.1% of the world's population - or 466 million people - require rehabilitation to address their disabling hearing loss (including 34 million children). It is estimated that by 2050 over 700 million people - or 1 in every 10 people - will have disabling hearing loss. [1]. This is why it is important to develop technology that can help people with hearing disabilities to communicate with people who do not know sign language. In this paper we will discuss our process of creating a model that can translate American Sign Language (ASL) and the challenges we faced to make a closer step to achieve this goal.

II. Dataset

Let's first discuss the data we need to use and the data we used to train our model.

A. Data

Data is the most important part of any machine learning model. What data we use to train our model will determine how well our model will perform. There were many things we had to consider when choosing the data.

First we looked into custom data, in case we want to make our own dataset. Making our own dataset would have been a good idea, as we could have made the dataset as big and diverse as we wanted. The model would be closer to our biases and would fit better to our needs, but the one big disadvantage of making our own dataset, is that it would have taken a lot of time to make a proper dataset.

We tried making our own dataset, and made a small working code that could do this automatically, but we quickly realized that it would take too much time considering that one letter per person takes $20 \sim 30$ minutes each.

We then looked in to different existing datasets, and came across MNIST sign language dataset [12]. We considered using this dataset, because MNIST is very well known among the machine learning community regarding image classification of letters [14]. The dataset consists of 28×28 images of the American Sign Language alphabet, and has 24 different classes. Some examples listed below: Letter J and Z are not

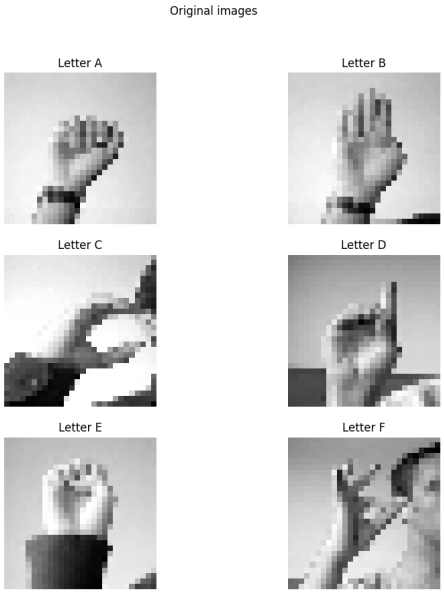

Fig. 1. Six ASL alphabet

included in the dataset, as they require motion.

One problem we later faced is that the dataset was not as big as we wanted it to be, we thought of combining this dataset with another dataset, but this would have not been ideal (covered in a later section).

B. Data preprocessing

The dataset was already preprocessed, so we did not have to do any preprocessing on the dataset. We did provide some preprocessing in the code, this is provided in case the dataset is not preprocessed or in case we want to make our own dataset. The preprocessing we provide is as follows:

- We first convert the images to grayscale, as we only need the intensity of the image.
- We then resize the images to 28×28, as the images are not all the same size.
- We then normalize the images, as the pixel values are between 0 and 255 (in our case remapped between 0 and 1).

These are the basic preprocessing steps that are needed, you could add some other preprocessing steps if you want to add blur or noise, etc. Accounting that we focus the image first to the hand before we resize. We also do some normalization and standardization of the data, as this will help the model to learn better. We also make use of a mean image, this is the

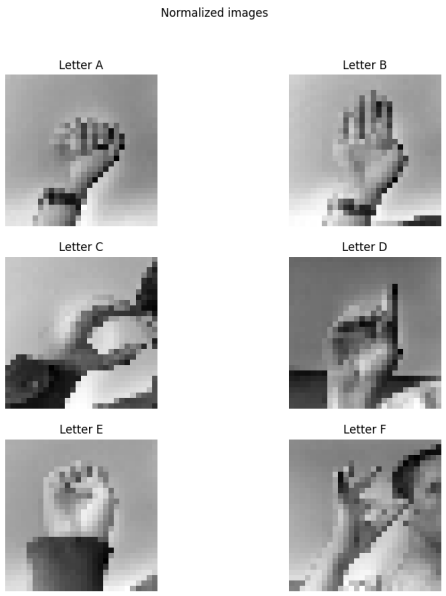

Fig. 2. Normalized images

average of all the images in the dataset. We use this image to substract from the images, this will help the model to only focus on the important parts of the image.

This is the preprocessing for the first method, the second method is a bit different, as we need to track the landmarks of the hand first. Every hand has 21 landmarks, each landmark signifies a certain point on the hand. Each landmark has a certain x, y and z coordinate, where x is the width, y is the height and z is the depth.

C. Data formats

The dataset itself is in the form of images, and the labels are in the form of integers. After preprocessing the images, we convert the images to an array of pixel values and store this in a csv file.

The landmarks are stored in a csv file, where each row is a different image and each column is a different landmark. In total there are 21 landmarks on each hand.

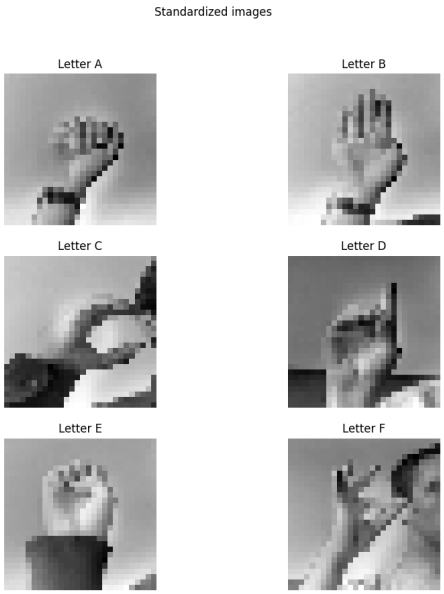

Fig. 3. Standardized images

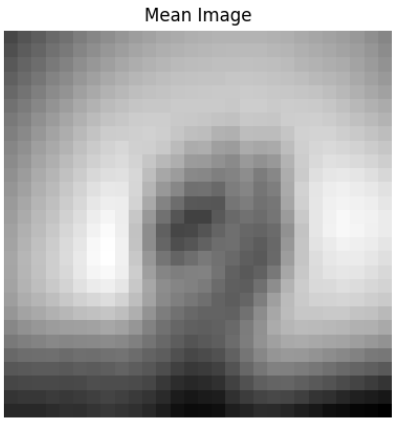

Fig. 4. Mean image

D. Data Augmentation

Data augmentation is a technique used to increase the size of the dataset by adding slightly modified copies of the data. Note that this could cause problems if not done correctly, the augmentation should still represent realisitic data.

E. Data distribution

We distribute our dataset in a 60% training set, 30% validation set and 10% test set. This causes our dataset to be smaller than we would have liked.

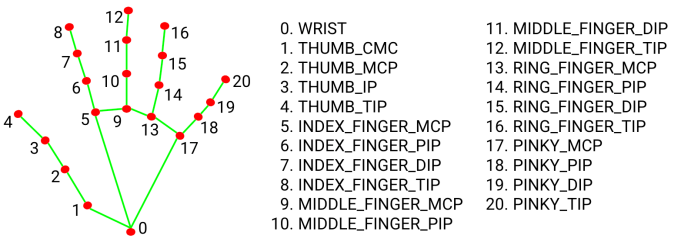

Fig. 5. Landmarks

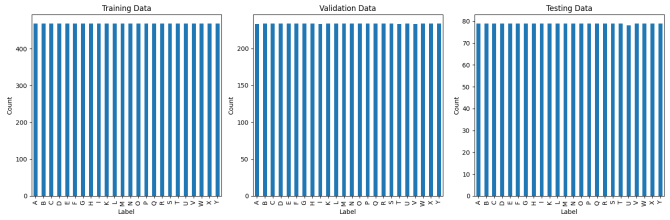

Fig. 6. Dataset distribution

III. Modelling

The most important part of our project is the model we created. Let's discuss how and what we did to create the model.

A. Layers

Before we dive in to the model we created, let's first discuss the different layers we used in our model.

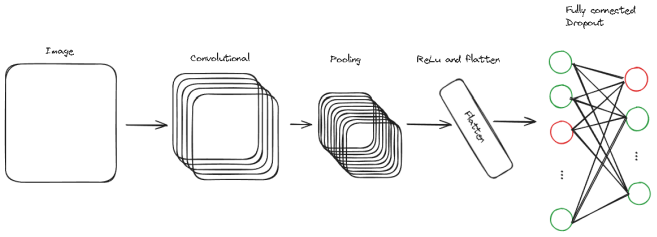

Fig. 7. Network

1) Convolutional layers: This is the most important layer in a standard CNN model. The purpose of a convolutional layer in a Convolutional Neural Network (CNN) is to extract features from the input data. This layer performs a mathematical operation called convolution, which involves sliding a filter (also known as a kernel) over the input data to produce a feature map. [17]

The most common type of convolution that is used is the 2D convolution layer and is usually abbreviated as conv2D. A filter or a kernel in a conv2D layer "slides" over the 2D input data, performing an elementwise multiplication. As a result, it will be summing up the results into a single output pixel. The kernel will perform the same operation for every location it slides over, transforming a 2D matrix of features into a different 2D matrix of features. [18]

2) Non Linearity (Activation) layers: Next, we have the activation layer. An activation layer is a crucial component that introduces non-linearity into the network. This is important because convolution operations are linear, and without non-linearity, the network would not be able to model complex data distributions effectively.

3) Pooling layers: Pooling layers are used to reduce the spatial dimensions of the input volume. They divide the input data into small regions, called pooling windows or receptive fields, and perform an aggregation operation, such as taking the maximum or average value, within each window. This aggregation reduces the size of the feature maps, resulting in a compressed representation of the input data. [20]

4) Fully connected layers: A fully connected layer refers to a neural network in which each neuron applies a linear transformation to the input vector through a weights matrix. As a result, all possible connections layer-to-layer are present, meaning every input of the input vector influences every output of the output vector. [21]

5) Dropout layers: Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of the input units to zero during training. This method makes the model more robust and less likely to memorize the training data.

6) Flatten layers: The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. It acts as a bridge between the convolutional/pooling layers, which extract spatial features, and the fully connected layers, which perform classification or regression tasks. [23]

7) Softmax layers: The Softmax function essentially converts the raw output scores (logits) of the neural network into probabilities. This allows the network to output a probability distribution over the classes, making it easier to interpret the model's predictions and make decisions based on the class probabilities. In a CNN, the Softmax function is typically applied to the output of the last fully connected layer of the network, which is then used to make predictions about the input data belonging to different classes. [24]

8) Batch normalization layers: Batch Normalization is a normalization technique that can be applied at the layer level. Put simply, it normalizes "the inputs to each layer to a learnt representation likely close to (μ = 0.0, σ = 1.0). As a consequence, all the layer inputs are normalized, and significant outliers are less likely to impact the training process in a negative way. And if they do, their impact will be much lower than without using Batch Normalization. [30] This is done when the data is not normalized, and the model is not converging well. For instance if the batches are different from each other, causing the weights of the model to shift from eachother, making the training process longer.

B. Pixel

The first method we used to train our model is by using the pixels of the image. Every image is made up of pixels, and these pixels have values. Like earlier discussed, this value is

between 0 and 255, we normalize this by ranging it between 0 and 1. The data is then fed to the model, which learns the patterns in the data.

C. Landmarks

The second method we used to train our model is by using the landmarks of the hand. Every hand has 21 landmarks, each landmark signifies a certain point on the hand. This is based on x and y coordinates, where x is the width and y is the height. Instead learning what the image looks like, it will now train based on where the hand is.

D. Training and Validation

Next, we will discuss how we trained our model. Training is similar to the training of any other model. We define some hyperparameters, such as the learning rate, the number of epochs, the batch size, etc. More on that later. We then feed the training data to the model and let it train. This training process is done in batches, where the model is fed a batch of data, computes the loss, and updates the weights. During every epoch, we have two passes through the data, one forward pass and one backward pass. The purpose of the forward pass

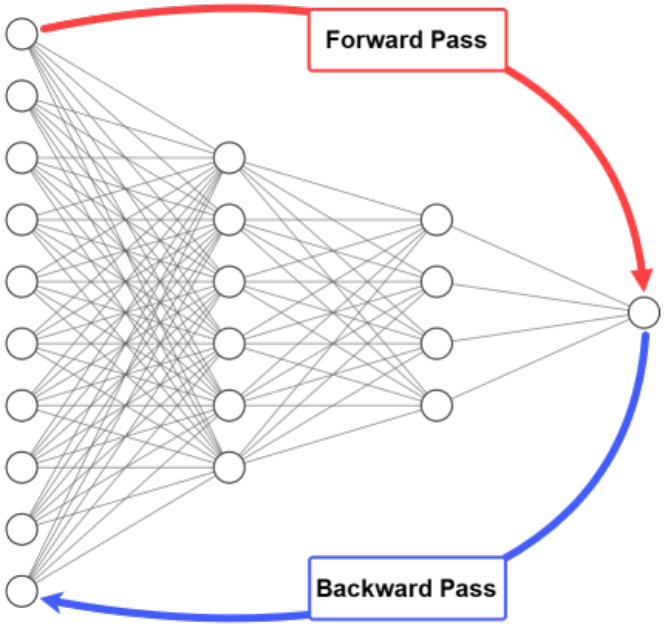

Fig. 8. Forward and backward pass

is to propagate input data through the network and produce predictions or outputs based on the model's learned parameters (weights and biases). [26] In the backward pass, the flow is reversed so that we start by propagating the error to the output layer until reaching the input layer passing through the hidden layers. [27] After these two passes, we have completed one epoch for training.

We also have a validation set, which is used to evaluate the model's performance on unseen data. Note that in our case, the validation set is a subset of the original training set that we split up in training and validation.

After these two steps are done, we evaluate the model based on the performances of training and validation. The training can be interupted if the model is not improving anymore. We do this by using a technique called early stopping. Early stopping works as follows: We monitor the validation loss and trainig loss during training. At each epoch, we check if the validation loss has decreased compared to the training loss. In case the validation loss has not decreased for a certain number of epochs, we stop the training process. The amount of epochs we wait before stopping is a hyperparameter we call patience. Why do we use early stopping? Early stopping is a regularization technique that helps prevent overfitting by stopping the training process before the model starts to memorize the training data. [28]

1) Tuning: We also have some hyperparameters that we can tune. Earlier we already mentioned some hyperparameters, but what are hyperparameters? They are like settings you choose before teaching a neural network to do a task. They control things like how many layers the network has, how quickly it learns, and how it adjusts its internal values. [29] Some hyperparameters we can tune are:

- Learning rate
- Number of epochs
- Batch size
- etc.

These hyperparameters can be tuned based on how the model is performing after training. In case the learning takes too long and the model is not improving, we can increase the learning rate for instance.

Hyperparameters are important because they can have a significant impact on the performance of the model. In practice, hyperparameters are usually set based on trial and error, where different values are tested to see which ones produce the best results on a validation set.

IV. Hand tracking

Hand tracking is one of the most important parts of our project. This combines the visual image of the hand and the model we created.

We make use of OpenCV [31] and MediaPipe [33] to track the hand. We start by capturing the video feed from the camera, and then we use the hand tracking model from MediaPipe to track the hand. OpenCV helps us to process the image before we do anything with it.

We also make use of MediaPipe to help us track/calculate the landmarks of the hand.

V. GUI

So, let's talk about the implemented GUI.

A. Customtkinter

Customtkinter is a open-source library that provides a set of custom widgets for Tkinter. It is a wrapper around Tkinter that provides a set of custom widgets that are not available in Tkinter. We used this library to create the GUI for our project because it is easy to use and provides us with a small interface that easily displays our implemented features.

B. Features

The GUI has a few features that showcase our project. The main features are:

- Loading a implemented model (CNN).
- Handtracking using your webcam.

The Handtracking feature uses the MediaPipe and OpenCV library to track the hand and display the landmarks on the screen, More about the hand tracking is discussed in the hand tracking section. If you want to know more about the models you can also read the corresponding section.

VI. Results

So, let's talk about the results we obtained.

A. CNN

We made two models, one makes use of pixel values while the other makes use of landmarks. Note that this is based on a reduced dataset, as we only have 6 letters. This is because of the hardware limitations, as we can't train effectively on a bigger dataset.

1) Pixel: The pixel model didn't perform as well as we hoped, because it makes use of the pixel values, it needs a lot of data to perform well. The model reaches an accuracy of 0.4 during training and stops learning quickly afterwards. We tried to fix this by adding more data, by augmentation, but this didn't help.

2) Landmarks: Landmarks performed better than the pixel model, as it only needs 21 landmarks to predict the letter. Not only that, the landmarks train based on position, so enviroment doesn't matter. The model during testing reaches an performance of 0.6667, whith a dataset of only 6 letters.

B. Graphs

We made a few graphs to show the performance of the models. We can see how the training evolves over time. The

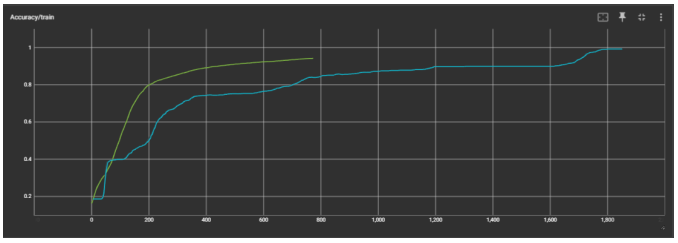

Fig. 9. pixel vs landmarks

green line represents the pixel model, while the blue line represents the landmarks model.

We can also look at the evolution of all models that we trained. As you can see the figure 10, we had struggles with many models, and eventually only one model out of many performed well.

VII. Issues

There were several issues we encountered during the project. We will discuss the most important ones in this section.

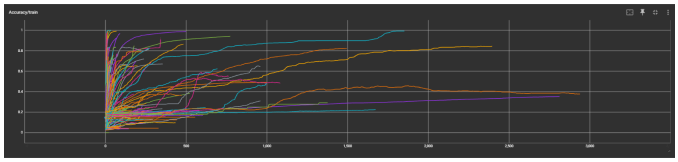

Fig. 10. accuracy graph for all models

A. Data

The first issue we encountered was the data we used to train our model.

1) Data Collection: The issue we faced was that data found online is not always trustworthy. One of the dataset we used before MNIST, was one that we found on kagle, stating that it was a dataset of the American Sign Language alphabet. After looking further into the dataset, we quickly found out that the dataset did not only consist of the American Sign Language alphabet, but also of other alphabets. This was a big issue, as the model would not be able to distinguish between the different alphabets.

2) Data mapping: Another issue is that the dataset were always mapped to the full alphabet, including the letters J and Z. This creates extra number of classes that our model needs to account of, leading to less accuaracy. We fixed this by remapping the classes to the alphabet excluding J and Z.

3) Data creation: Why not create our own data? As mentioned before we tried to create our own data, but this was not as easy as we thought. One of the problems is that quality of images can have an impact on the model. For instance if the obtained dataset is not in the same quality/format as the webcam we use to test the model, the model will not be able to predict the letters correctly or accurately. Not only that, creation of a dataset or combining datasets can also result in a dataset that is not balanced. This means that the model will be biased towards a certain enviroment.

4) Data preprocessing: One more issue we had, that we realised with landmarks, is that the data had been cropped too much and would not be recongnized as a hand. To solve this issue, we just disabled cropping for landmarking.

B. Model

Making a good model is not easy, and we encountered several issues when creating the model.

1) Overfitting: One issue we encountered was that the model was overfitting. The model would quickly overfit causing our model to only remember certain letters, for instance only A for all hand signs. Fixing this issue was not easy, as we had to try different things to prevent overfitting.

2) Layers: Next issue, deciding the layers and how many layers our model should have is a difficult task. You have to account for the complexity of the model, the number of classes, the size of the dataset, etc. Our model was originally overcomplex and would train longer than needed. At one point it was not complex enough and would not be able to distinguish between the different letters. Complexity can also

decrease the training time, if the layers are chosen correctly. So, overall choosing the right layers and the right number of layers is a difficult task. Another issue that can occur with complex models, is that the model would stop too early and not learn anything.

C. Training

Training the model was also a difficult task.

1) Hyperparameters: As mentioned before, there are a few hyperparameters that we used when training the model: the learning rate, the batch size, the number of epochs, min delta, and patience. Finding the right values for these hyperparameters was a fiddly task that required a lot of trial and error. If the learning rate was too low, the model would not learn anything and would essentially flatline right at the start. If the learning rate was too high, the model would learn too quickly and the predictions would be all over the place or result in overfitting. When tested, it would sometimes predict the same letter, no matter what the input was. Other times, it would predict a different letter every time. The same problems occured with the batch size. If it was too low, the model would not have enough data to learn in each epoch and it would end up the same as if the learning rate was too low.

We started with a learning rate of $1e^{-3}$, but that was extremely high, so then we reduced it, tested different values and ended up training our model mostly with a learning rate in the range of $1e^{-6} - 1e^{-8}$. For the batch size, we started with a batch size of 32, but most of the time we decreased it to 16, as it gave us better results.

The number of epochs was initially 20! but it quickly became apparent that this was not enough. Most of the time we ended up training the model with a maximum of 1000 epochs and even later on with 10k to 100k epochs. Min delta and patience we did not change around much. Sometimes a higher min delta would give us better results, because the difference between the loss of the validation set and the training set was too big. The min delta was set to 0.0001 for most of the training and the patience was set to 2 (in the end it was raised to 4).

As mentioned previously, the dataset that we used was not big. This caused the model to overfit quickly. We tried to fix this by training the model in a reduced dataset of the first 6 letters of the alphabet, but this did not help much. It gave better results than the full 24-letter dataset, but it was still not detecting the letters correctly in "real life" situations.

VIII. Conclusion

In this paper we discussed about our process of creating a model that can translate American Sign Language (ASL) to text. We discussed the data we used, the model we created and the results we obtained. We also discussed the challenges we faced and the future work that can be done to improve our model.

We created two models, one that makes use of pixel values and one that makes use of landmarks. The pixel model didn't perform as well as we hoped, because it makes use of the pixel values, it needs a lot of data to perform well. The model reaches an accuracy of 0.4 during training and stops learning quickly afterwards. We tried to fix this by adding more data, by augmentation, but this didn't help. The landmarks model performed better than the pixel model, as it only needs 21 landmarks to predict the letter. Not only that, the landmarks train based on position, so enviroment doesn't matter. The model during testing reaches an performance of 0.6667, whith a dataset of only 6 letters.

We also created a GUI that showcases our project. The GUI has a few features that showcase our project. The main features are:

- Loading a implemented model (CNN).
- Handtracking using your webcam.

The Handtracking feature uses the MediaPipe and OpenCV library to track the hand and display the landmarks on the screen.

Hand tracking is one of the most important parts of our project. This combines the visual image of the hand and the model we created. We make use of OpenCV and MediaPipe to track the hand. We start by capturing the video feed from the camera, and then we use the hand tracking model from MediaPipe to track the hand. OpenCV helps us to process the image before we do anything with it. We also make use of MediaPipe to help us track/calculate the landmarks of the hand.

In the future we would like to improve our model by training on a bigger dataset. We would also like to improve the GUI by adding more features and improving the hand tracking. We would also like to improve the model by adding more letters to the dataset.

References

[1] Deafness and hearing loss, World Health Organization, 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss#~:\protect\protect\unhbox\voidb@x\hbox{=}Over%205%25%20of%20the%20world%27s,will%20have%20disabling%20hearing%20loss.

[2] A. Singh, "Building a Hand Tracking System using OpenCV" Analytics Vidhya, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/07/building-a-hand-tracking-system-using-opencv/

[3] S. Som, "Image Classification: American Sign Language using PyTorch" Medium, 2021. [Online]. Available: https://medium.com/@sachinsom507/image-classification-american-sign-language-using-pytorch-7bef9d7e8a25

[4] S. Som, "Sign Language Recognition with Advanced Computer Vision" Towards Data Science, 2021. [Online]. Available: https://towardsdatascience.com/sign-language-recognition-with-advanced-computer-vision-7b74f20f3442

[5] Y. Bing, "CNN Facial Landmark" GitHub, 2021. [Online]. Available: https://github.com/yinguobing/cnn-facial-landmark

[6] How can I get the return value of a function passed to multiprocessing.Process?, Stack Overflow, 2012. [Online]. Available: https://stackoverflow.com/questions/10415028/how-can-i-get-the-return-value-of-a-function-passed-to-multiprocessing-process

[7] Real Time Sign Language Detection with Tensorflow Object Detection and Python — Deep Learning SSD [Online] Available: https://www.youtube.com/watch?v=pDXdlXlaCco

[8] Sign language detection with Python and Scikit Learn — Landmark detection — Computer vision tutorial [Online] Available: https://www.youtube.com/watch?v=MJCSjXepaAM

[9] J. Pu, et al., “Sign language recognition using 3D convolutional neural networks” Nature Communications, 2021. [Online]. Available: https://www.nature.com/articles/s41467-021-25637-w
[10] S. K. Singh, et al., “Sign language recognition using deep learning: A review” Scientific Reports, 2023. [Online]. Available: https://www.nature.com/articles/s41598-023-43852-x#:~:text=In%20recent%20years%2C%20utilizing%20deep,recognition%20in%20the%20data%20image.
[11] C. Kent, “Writing Training a Vision Transformer for ASL Letters” Medium, 2021. [Online]. Available: https://medium.com/@crsnkntn/writing-training-a-vision-transformer-for-asl-letters-d870c4d3aeb7
[12] “Sign Language and Static Gesture Recognition using Sklearn” GitHub, 2021. [Online]. Available: https://github.com/mon95/Sign-Language-and-Static-gesture-recognition-using-sklearn/blob/master/Dataset.zip
[13] “Sign Language Recognition using CNN” GitHub, 2021. [Online]. Available: https://github.com/gchilingaryan/Sign-Language
[14] “MNIST database” Wikipedia, 2021. [Online]. Available: https://en.wikipedia.org/wiki/MNIST_database#:~:text=The%20MNIST%20database%20(Modified%20National,the%20field%20of%20machine%20learning.
[15] “Convolutional Neural Networks” Wikipedia, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network
[16] “Convolutional Neural Networks” Towards Data Science, 2021. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-cnns-and-layer-types-9bbd0a05e1a8
[17] “Convolutional Layer” Science Direct, 2021. [Online]. Available: https://www.sciencedirect.com/topics/engineering/convolutional-layer#:~:text=A%20convolutional%20layer%20is%20the,and%20creates%20an%20activation%20map.
[18] “Convolutional Layer” Databricks, 2021. [Online]. Available: https://www.databricks.com/glossary/convolutional-layer
[19] “Non Linearity (Activation) layers” TUM, 2021. [Online]. Available: https://collab.dvb.bayern/display/TUMlfdv/Layers+of+a+Convolutional+Neural+Network#:~:text=A%20non%2Dlinearity%20layer%20in,activation%20map%20as%20its%20output.
[20] “Pooling layers” Dremio, 2021. [Online]. Available: https://www.dremio.com/wiki/pooling-layers/#:~:text=What%20is%20Pooling%20Layers%3F,retaining%20the%20most%20important%20information.
[21] “Fully connected layers” Built In, 2021. [Online]. Available: https://builtin.com/machine-learning/fully-connected-layer
[22] “Dropout layers” Keras, 2021. [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/#:~:text=Dropout%20class&text=The%20Dropout%20layer%20randomly%20sets,over%20all%20inputs%20is%20unchanged.
[23] “Flatten layers” GeeksforGeeks, 2021. [Online]. Available: https://www.geeksforgeeks.org/what-is-a-neural-network-flatten-layer/
[24] “Softmax layers” Machine Learning Mastery, 2021. [Online]. Available: https://machinelearningmastery.com/softmax-activation-function-with-python/
[25] “Softmax layers” Quora, 2021. [Online]. Available: https://www.quora.com/What-is-Softmax-in-CNN
[26] “Training” DataCamp, 2021. [Online]. Available: https://campus.datacamp.com/courses/introduction-to-deep-learning-with-pytorch/training-our-first-neural-network-with-pytorch?ex=1#:~:text=The%20purpose%20of%20the%20forward,training%20and%20generating%20new%20predictions.
[27] “Training” Neptune.ai, 2021. [Online]. Available: https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide#:~:text=In%20the%20backward%20pass%2C%20the,backward%20propagation%2C%20or%20simple%20backpropagation.
[28] “Early stopping” Educative, 2021. [Online]. Available: https://www.educative.io/answers/what-is-early-stopping#:~:text=Early%20stopping%20is%20a%20technique,validation%20error%20starts%20to%20increase.
[29] “Neural Network Hyperparameters” Analytics Vidhya, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/#:~:text=Conclusion-,Neural%20Network%20Hyperparameters%20(Deep%20Learning),it%20adjusts%20its%20internal%20values.
[30] “Batch normalization layers” Medium, 2021. [Online]. Available: https://medium.com/@francescofranco_39234/batch-normalization-with-pytorch-20cdc205377f
[31] “OpenCV” OpenCV, 2021. [Online]. Available: https://opencv.org/
[32] “PyTorch” PyTorch, 2021. [Online]. Available: https://pytorch.org/
[33] “MediaPipe” MediaPipe, 2021. [Online]. Available: https://mediapipe.dev/