A High Performance Model for Recognition of Static Letters in Libras

Roberto S. M. Tomchak[1], Andrieli L. Gonçalves[1], Eloisa Nielsen[1], Rafael R. Kluge[1], Elen Pasqualli Gesser[1], Luiz Carlos P. Albini[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba, PR – Brazil

{rsmt23, epg21, albini}@inf.ufpr.br

{andrieligoncalves, eloisanielsen, rafaelkluge}@ufpr.br

Abstract. *Brazilian Sign Language (Libras) is the primary visual language in Brazil, recognized as an official language by law since 2002. It uses hand gestures and facial/body expressions to allow communication with deaf people, therefore making it a social inclusion instrument. However, there is a lack of tools for teaching it, leading to a low percentage of the population actually knowing how to communicate using Libras. This paper introduces an instance of such tools, by presenting a model that can recognize the static letters of the Portuguese alphabet in Libras. The model was implemented using traditional Machine Learning algorithms, paired with the library MediaPipe. We develop an extremely lightweight model which achieves accuracy higher than 96% and recognizes gestures in less than 0.2 ms, making it a high performance and low latency application. These characteristics allow the proposed model to be embedded in user-friendly software, including mobile applications.*

Resumo. *Língua Brasileira de Sinais (Libras) é a principal língua visual do Brasil, reconhecida como língua oficial por lei desde 2002. Ela utiliza gestos com as mãos e expressões faciais/corporais para permitir a comunicação com pessoas surdas, tornando-se, portanto, um instrumento de inclusão social. No entanto, há uma escassez de ferramentas para o ensino dessa língua, o que resulta em uma baixa porcentagem da população que realmente sabe se comunicar usando Libras. Este artigo introduz uma dessas ferramentas, apresentando um modelo que reconhece as letras estáticas do alfabeto português em Libras. O modelo foi implementado utilizando algoritmos tradicionais de Aprendizado de Máquina, em conjunto com a biblioteca MediaPipe. Desenvolvemos um modelo extremamente leve que atinge acurácia superior a 96% e reconhece gestos em menos de 0,2 ms, tornando-se uma aplicação de alto desempenho e baixa latência. Essas características tornam o modelo proposto adequado para integração em softwares user-friendly, incluindo aquelas voltadas para dispositivos móveis.*

1. Introduction

The Brazilian Sign Language (Libras) is an official visual language according to Brazilian law [Brasil 2002]. However, very few people actually know how to communicate using it, and an even smaller number of schools offer a Libras class. Therefore, there is a demand

for accessible education methods in Libras. One way to achieve such goal is through free and user-friendly automatic tools. An instance of such tools can be a system that is able to recognize gestures in Libras performed by a user in front of a camera and provide feedback on their execution. For example, if a user tried to gesture the letter S, but ended up making the letter A (which are similar in Libras), they would be warned by a message on the screen stating that the symbols are mixed up. On the other hand, if the user made the correct symbol, but did it awkwardly, the system would warn them about it and suggest to them how to improve it.

This paper reports the development of a model which can recognize hand gestures in Libras and classify them according to which symbol it seems to represent. Since identifying Libras gestures is a complex task that cannot be easily made through a traditional algorithm, our solution includes the use of a Machine Learning (ML) model that can be taught using data, collected from various volunteers, of the possible hand placements. Using ML is beneficial because the model can learn the complexity of the data by itself, without the help of a specialist in the area or a complex code that would be hard to maintain and might not even guarantee good results [Géron 2022]. Since we aim to use this model as an educational tool, it is essential that the model is lightweight, so it can easily run in various devices, and makes predictions quickly, to avoid a slow user interface. We believe these requisites are mandatory in creating an accessible educational tool.

We demonstrate that, by using state of the art MediaPipe library with traditional ML techniques, we can create a system that is not only good at classifying symbols in Libras, but is also fast and light regarding memory usage, making it useful for user-friendly applications. Moreover, we showed that this model can easily be embedded into a user-friendly extreme lightweight application with more than 96% accuracy, less than 0.2ms recognition time and less than 375MB memory consumption. These characteristics allow the proposed model to be embedded in user-friendly mobile applications as modern mobile devices have RAMs in the order of gigabytes.

This paper is organized as follows. In Section 2, we present related works to ours. In Sections 3, we explain how the MediaPipe library works, and why it is useful in our task. Sections 4 and 5 describe how the dataset was created. In Section 6 we describe the preprocessing we did to the data. In Sections 7 and 8 we explain how we trained the model and introduce the weighted accuracy metric. Finally, in Section 9 we present our results and conclusions in Section 10.

2. Libras Recognition Related Work

Many recent works have tackled the Libras recognition problem, using different types of models and datasets. The most common approach is to use Deep Neural Networks (DNN). In [Santos et al. 2022], the authors use a YOLOv5 architecture [Redmon et al. 2016], which consists of 7.2 million parameters. The model was trained with the Libras Computer Vision Project, which consists of 8,455 images distributed in 40 different classes. Their model achieves an accuracy of 90.67%. In [Caiafa et al. 2020], the authors also train a model in the same 40 classes of symbols in Libras, achieving a higher accuracy of 99.54%. However, to obtain this result, they use a VGG19 architecture [Simonyan and Zisserman 2015], which uses 144 million parameters, 20 times more than the YOLOv5 used in [Santos et al. 2022]. They also combine two different

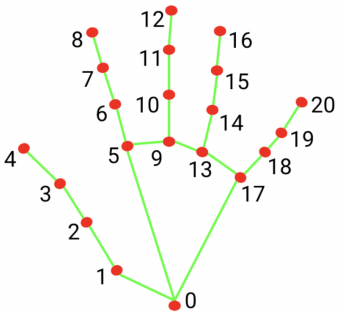
databases, with a total of 21,800 images. Other works follow a similar methodology [Furtado and De Oliveira 2021], achieving high accuracy but at the cost of using very large models.

More recent approaches use an DNN model only to extract relevant features of the hand, and then use these features to train a Libras recognition model. This is called skeleton-based recognition, and has been used in other sign language systems [Li et al. 2025, Trabelsi et al. 2025]. The works of [de Morais et al. 2025] and [Benevides et al. 2024] use MediaPipe Hands [Vakunov et al. 2020] as a feature extractor, achieving accuracy of 98.01% and 81%, respectively. The first uses a Extra trees model, while the second uses a SVM. This approach leads to the training of a much smaller model compared to DNNs, allowing for high prediction power and easier training.

All of these works manage to achieve high accuracy using sets of basic symbols in Libras. However, none of them mention costs in using the model, such as model size and energy required to train it. Many of them use DNN models that are very large and require big datasets, and thus cannot be easily embedded in practical applications. In this work, we continue the research in the Libras recognition task, but our goal is a lightweight model with low latency that still manages to achieve a competitive accuracy.

3. MediaPipe Library

MediaPipe is an open source project developed by Google which provides tools for Machine Learning and Artificial Intelligence. In our project, we use its sublibrary MediaPipe Hands, which is able to recognize hands in a given image and extract useful information from it. In particular, we use the 21 three-dimensional points it provides (see Figure 1) and the hand orientation (whether it is the left or the right hand). A plausible assumption is that these values are equivalent to the gesture in the image. The MediaPipe library offers enough information to know the position and rotation of each finger, which are the primary factors to predict the gesture. Other visual information, like the thickness of the finger, is not relevant when communicating with Libras. Therefore, we can discard the rest of the image, and use only these 64 values (21 points in 3D space + 1 Boolean value representing if it's the left or right hand) to predict the gesture in a given image.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 1. Reference Points of Mediapipe

The library provides three main advantages. First, it significantly reduces the dataset file size; the image files have 71 bytes each in JPG format, and the data points have 0.57 bytes, on average, using pickle file format. This represents a 99.20% reduction in space usage. Second, instead of our model having to find the symbol by analysing

the entire image — a complex computer vision task — it only needs to consider the 64-dimensional input vector. This is a much easier task, which leads to a simpler model with higher accuracy, making it easier to embed it in a more sophisticated software. Finally, given that only the information about the points of the hand and its orientation are stored, it makes the process of collecting data safer for volunteers by protecting their identity, even in a public repository.

4. Data Collection

For this project, we define the possible symbols as the letters of the alphabet that do not require hand movement in Libras. The ones that require movement are K, H, J, X and Z. Therefore, there are 21 symbols in total. The choice of excluding those letters was made to simplify the model, since detecting movement would require video analysis, and our focus on this step was to detect the symbols though one image only. The data was collected by the group "PET Computação UFPR", whose members worked on this project. The group shared a Google forms through their social media for volunteers to participate in the project. All the data collection process was explained to the volunteers, who authorized the group to maintain the data collected in a public repository. Each file represents the data from one volunteer, whose name was anonymized using MD5 hashing. A member of PET demonstrated how to make the symbols though the collection process, to guarantee the volunteer was doing the correct symbol.

5. Dataset

During collection, each volunteer did 20 images per gesture, 10 with the left hand and 10 with the right one, giving a total of $21 \times 20 = 420$ images per volunteer. In total, there were 14 volunteers, resulting in a total of 5,880 images. The final dataset had 5,801 observations since the MediaPipe library failed to recognize the hand in a few of them. However, this represents a reduction of only 1.34% of the data. Figure 2 shows the percentage of each symbol in the dataset. The class that deviates the most from the average is the letter Q, which represents 4.53% of the dataset, while the average is 4.76%. Since that is a small deviation, the dataset is considered balanced.

In order to train and evaluate the ML models, the dataset was split into a train set and a test set. The ratio used is 80/20, meaning 20% of the original dataset was set apart for testing. The data is divided in such a way that the people used in training are disjoint from the ones used in testing. Therefore, if an example from a person is used for training, none of the examples in testing are from this person, and vice versa. This is a fair analysis of the performance of the model, since when using it in an application, it is expected that the model will have to predict the symbol of gestures made from people that do not appear during training. After the split, the train set has 4.556 examples from 11 different people, and the test set has 1.245 examples from 3 different people.

Table 1 shows the notation used in this paper. The terms label and symbol have the same meaning here, representing the class of the respective hand gesture. Each $\mathbf{x}$ is an array with 64 features, those being the 21 three-dimensional points from MediaPipe and the hand orientation. Equation 1 shows how the array $\mathbf{x}$ is organized, where (x_i, y_i, z_i) are the coordinates of the i-th point, and is_left is 1 if the gesture is made using the left hand and 0 otherwise. After preprocessing the data, this array can and will change. We keep the same notation, and let context clarify exactly how $\mathbf{x}$ is organized.

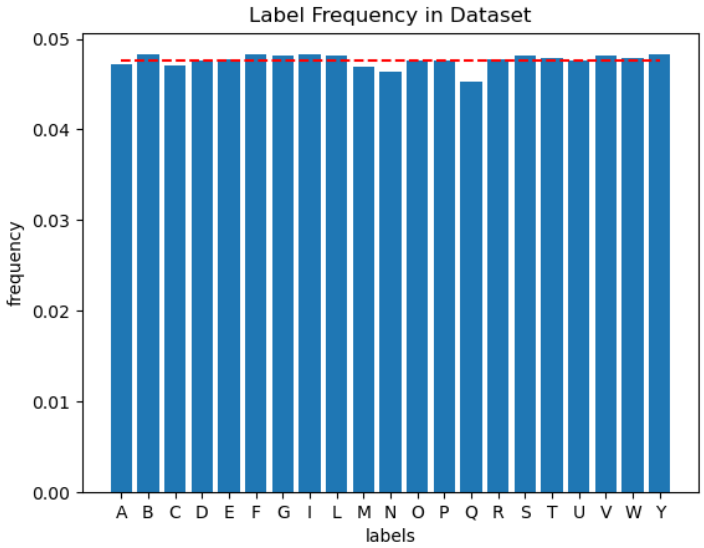


Figure 2. Distribution of Observations per Symbol

Table 1. Table of Notation

Symbol	Description
$\mathbf{x}$	Features of an observation
s	Label (Symbol) of an observation
$\mathbf{X}$	Matrix of features (each row is an observation)
$\mathbf{s}$	Array of labels

$$\mathbf{x} = (x_0, y_0, z_0, x_1, y_2, z_2, ..., x_{20}, y_{20}, z_{20}, is\_left) \quad (1)$$

6. Preprocessing

Even though ML models can learn from raw data, doing so means the model will have to learn all patterns by itself. If a certain pattern is clear, we can do some preprocessing to the data to simplify the problem, which can help the model learn and increase its performance [Bishop and Bishop 2019]. Also, these transformations to the data are usually computationally inexpensive when compared to the computations made by the ML model, so they do not add any overhead. The following transformations were applied to the input data $\mathbf{X}$. Recall that, initially, each array $\mathbf{x}$ is organized as described in Equation 1. These transformations were applied sequentially to the data, however, in Section 9 we also consider what happens if we only apply a subset of them.

6.1. Centralize

Consider Figure 3, showing two different images of the letter B sign in Libras. Even though the hand gesture is about the same in both images, their coordinates are very different since they are in distinct places of the image. As the exact location of the hand should not impact the label, it makes sense to have all hand gestures be in the same

place. Thus, we can centralize all hand images by making the wrist, which is marked as the point 0 in MediaPipe, be the origin of the axis. This can be done by subtracting the point 0 from all other points. Therefore, for each (x_i, y_i, z_i), the new coordinate is $(x_i - x0, y_i - y0, z_i - z0)$. This normalization ensures that the coordinates remain consistent regardless of the location of the symbol within the image – whether at the center or in a corner. Consequently, the model will no longer have to learn this pattern, leading to better predictions. After this transformation, $\mathbf{x}$ is a 61-dimensional array, since the point 0 is now always $(0, 0, 0)$, and therefore can be discarded from the array. Equation 2 describes the new array $\mathbf{x}$.

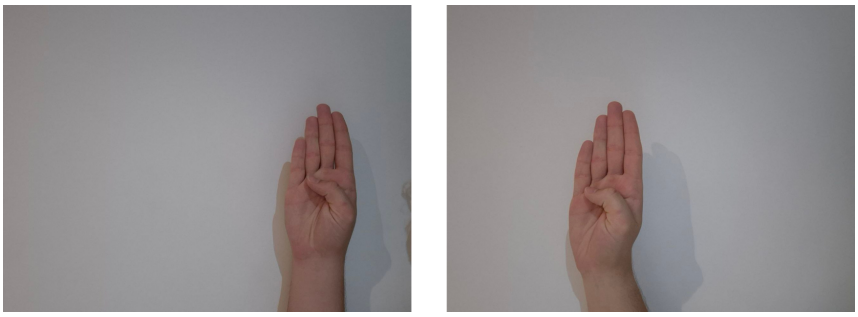


Figure 3. Two Different Images of the Gesture "B"

$$\mathbf{x} = (x_1, y_1, z_1, x_2, y_2, z_2, ..., x_{20}, y_{20}, z_{20}, is\_left) \tag{2}$$

6.2. Fix Orientation

Consider Figure 4, showing the letter S sign in Libras using opposite hands. Notice that the symbols are almost identical. In fact, what sets them apart is solely a reflection across the y-axis. Given that, we choose to convert all right-hand gestures into left-hand gestures, which don't need to be modified, by mirroring the points about the y-axis. Since the previous transformation fixed the hands to the origin, this can be done by simply negating the x value across all points. Therefore, if is_left = 0, each coordinate (x_i, y_i, z_i) becomes $(-x_i, y_i, z_i)$. This way, the model does not have to learn separate patterns for the left and right hand, and can generalize better. After that, the hand orientation is no longer required and can be removed from $\mathbf{x}$, which is now described by Equation 3. Notice $\mathbf{x}$ now has 60 features.

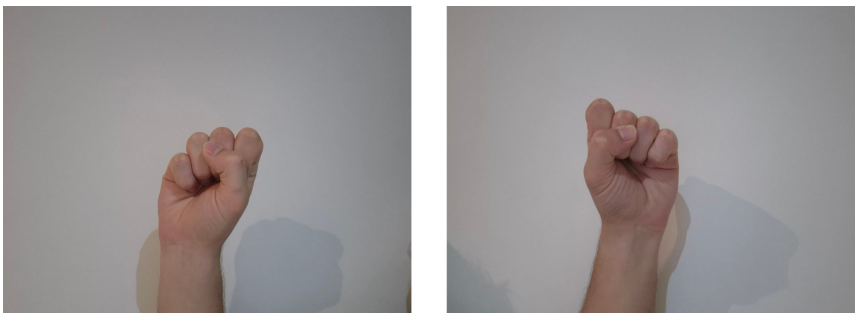


Figure 4. Two Different Images of the Gesture "S"

$$\mathbf{x} = (x_1, y_1, z_2, x_2, y_2, z_2, ..., x_{20}, y_{20}, z_{20}) \tag{3}$$

6.3. Remove z dimension

The z-axis of the coordinates represents the depth of the point in relation to point 0. It is not clear if this information helps the model predict the label, since none of the used gestures require rotating the hand. Features that have little to no correlation can negatively impact the performance of the model since they introduce noise into the dataset [Bishop and Bishop 2019]. Also, the time complexity of the algorithms used to create the models usually increases with the number of features, so the less features, the faster it is at predicting labels and training [Géron 2022]. Given that, we consider two types of tests with the models: one with the Z dimension and another without it. Models that use this transformation will have each coordinate (x_i, y_i, z_i) turn into (x_i, y_i). Equation 4 describes the array $\mathbf{x}$ after this transformation. Notice this brings down the number of features from 60 to 40, which is a big difference. We show in the following sections that some models benefit from this transformation.

$$\mathbf{x} = (x_1, y_1, x_2, y_2, ..., x_{20}, y_{20}) \tag{4}$$

7. Metrics

To determine the quality of the models, various numerical metrics are applied. The main metric of this project is weighted accuracy, a metric created specifically for our problem. It is defined as the weighted average recall of each label, where the weights are the frequency of each letter in the Portuguese language [Quaresma and Pinho 2007]. Equation 5 shows the calculation for weighted accuracy, where S is the set of all labels in our task, $recall(s)$ is the recall metric of label s, and $freq(s)$ is the frequency of the label s in the Portuguese language. This metric represents our problem better than default accuracy, since it penalizes models that are bad at recognizing the most recurring letters. As a way to visualize that, consider two models: model 1, which is very accurate at recognizing the letter A, but very inaccurate with the letter W; and model 2, which is the opposite: very good at handling the letter W, but bad with the letter A. Any other letter both models are equally good at recognizing. In this scenario, these two models have about the same default accuracy, but model 1 is probably better in practice, since the letter A appears more often than the letter W in words in portuguese. With weighted accuracy, the model 1 will be prioritized over model 2, since the letter A receives a higher weight than the letter W.

$$Acc_w = \sum_{s \in S} recall(s) \times freq(s) \tag{5}$$

8. Training

There are many ML algorithms that are very successful in classification tasks. The following ones are used in this project: KNN, SVM and Random Forest. K-Nearest Neighbors (KNN) [Géron 2022] is a simple algorithm for classification, which determines the label by analysing the label of the observations of the train set that are closest to the given

input. This algorithm has some geometric meaning built into it, which can be very interesting for our problem, since our data represents points in some space. Support Vector Machine (SVM) [Géron 2022] is a good classification model for small and medium size datasets, which is our case, due to its complexity. This algorithm defines a boundary that separates the data according to their labels, so just like KNN, it has some geometric meaning. Finally, Random Forest (RF) [Géron 2022] is one of the most popular machine learning algorithms, and generally performs well in most tasks with tabular data [Grinsztajn et al. 2022]. This algorithm is an ensemble of many Decision Trees, which determines the label based on some binary questions over the input data.

It is important to point out that only traditional ML algorithms are used here, and not neural networks. Even though neural networks can learn very complex functions, they have the drawback of needing many parameters to learn these types of relationships, which leads to slower models that require a lot of memory [Bishop and Bishop 2019]. This can not only make training very slow, but also make it harder to create a light application with the model embedded into it. Also, having a big amount of parameters relative to the size of the dataset can lead to overfitting, causing a drop in performance in the test set [Bishop and Bishop 2019]. For these reasons, neural networks are left out as possible architectures for our model.

In order to achieve the best model, each of the algorithms is set apart and its main hyperparameters are fine tuned using the train set. All algorithms are applied using the Scikit-learn library. To compare different choices of hyperparameters, the Grid Search technique is used. The best three combinations of hyperparameters of each algorithm is saved, to be compared with the others in the test set. Since there are three choices of algorithms, this results in 9 final models.

During training, only the weighted accuracy is considered to filter out the best models, since it is our main metric. Also, the performance of the models are analyzed using k-fold cross validation of the train set, with $k = 5$ [Géron 2022]. By not using the test set, we avoid overfitting the models [Bishop and Bishop 2019], and the small value of k has a small impact on the total training time.

9. Results

To guarantee constant results, all models are trained and tested in the same environment. For this work, we use a machine with 11th Gen Intel® Core™ i7-11390H @ 3.40GHz × 8 processor, with 16 GiB of memory, running the Ubuntu 22.04.4 LTS operation system. This architecture can be found in a current generation notebook, so these results should be easily reproduced in practical applications of our models.

Table 2 shows the performance of the final models in the test set. We can see that all models have good weighted accuracy and accuracy, but the SVM models dominate over the other algorithms. SVM 1 and SVM 2 have the best weighted accuracy and accuracy, respectively. However, RF 3, while not being as accurate or light as the other models, has lower latency, which can be interesting in some applications. Random Forest models tend to have low latency since they are very parallelizable, but also need big file sizes to keep the information of all trees, which explains these results The KNN models have the lowest accuracy scores and highest latency, so it seems they are not good for the Libras recognition task. We choose SVM 1 as our main model, since weighted accuracy

is our main metric, and it has low latency and is lightweight in terms of file size.

To understand how the transformations in Section 6 affect the performance of the model, we also train and test SVM 1 with all possible combinations of transformations. This means we test the model with no transformations, with only one of them, with two of them and all combined. Notice using the centralize and fix orientation transformations is equivalent to SVM 1 in Table 2. We measure weighted accuracy and latency, divided in transformation (trans.) latency and prediction (pred.) latency, to see how relevant the computational time of the transformations were. The results can be seen in Table 3. Even though in the train set using the Z dimension gave the best results, it turns out SVM 1 has slightly better results when using the remove z transformation in the test set. We can also see that the transformations have no relevant impact in the latency. In fact, using more transformations tends to reduce the total latency, since they reduce the amount of features the model needs to consider. The centralize transformations seems to be the most important one, since only the models using it manage to achieve a weighted accuracy above 90%. This is likely due to the fact that learning to detect the same gesture in different positions in an image is a difficult task, since the coordinates of the points can change drastically.

Table 2. Performance of the Best Models in the Test Set

Model	Remove z?	W. Acc. (%)	Acc. (%)	Latency (ms)	File Size (MB)
KNN 1	Yes	90.19	88.59	4.70	1.43
KNN 2	Yes	90.13	88.67	4.82	1.43
KNN 3	Yes	89.98	88.35	4.38	1.43
RF 1	Yes	88.62	85.78	0.11	38.93
RF 2	Yes	88.52	85.78	0.03	19.48
RF 3	Yes	89.07	85.86	**0.02**	19.45
SVM 1	No	**96.48**	94.54	0.17	1.13
SVM 2	No	96.31	**94.62**	0.14	**0.99**
SVM 3	No	96.19	94.54	0.16	**0.99**

Figure 5 shows the confusion matrix for SVM 1 in the test set. The diagonal is very bright, while all other cells are very dark, which shows the model in general makes very few mistakes. The highest number outside the diagonal is 10, where the model predicts T when it should have been F. These letters are similar in Libras, so it's expected for this to be one of the weak points of the model. Surprisingly, the model does not make the mistake in the other way around. This could show a bias in the model to predict T when it is split between T and F. Another mistake the model makes is mix R and U, which are also similar in Libras. The model also predicts T when it was a Y in 7 examples, which are not really similar letters in Libras.

To check how the model would perform in an user-friendly application, we developed a simple interface, shown in Figure 6. The user makes a gesture, and the application passes it through our pipeline (MediaPipe, Transformations, SVM 1). The predicted symbol is then shown in the interface, close to the hand. The application demonstrates the extreme lightness of the developed model achieving more the 96% accuracy with a recognition time smaller than 0.2 ms. Furthermore, the memory consumption of the ap-

Table 3. Impact of Transformations on Performance and Latency

Transformation	W. Acc. (%)	Trans. Lat. (ms)	Pred. Lat. (ms)	Total Lat. (ms)
None	86.30	0.00	0.74	0.74
Centralize	93.55	0.00	0.49	0.49
Fix orientation	86.25	0.00	0.57	0.57
Remove Z	88.07	0.00	0.61	0.61
Centralize and Fix orientation	96.48	0.00	0.32	0.32
Centralize and Remove Z	94.54	0.00	0.38	0.38
Fix orientation and Remove Z	87.57	0.00	0.65	0.66
All	**96.79**	0.00	0.29	**0.29**

plication is 374.58 MB, including the whole pipeline. Since most modern mobile devices have RAMs in the order of gigabytes, this application may run in most mobile devices.

10. Conclusions

Our work shows that the Machine Learning approach is a viable tool for creating a model to recognize a subset of the gestures in Libras. Using the MediaPipe library within traditional ML algorithm, we achieved a high performance model, not only in terms of accuracy, but also in terms of latency and model size.

We showed that this model can easily be embedded into a user-friendly an extreme lightweight application achieving more the 96% accuracy with a recognition time smaller than 0.2 ms. Furthermore, the memory consumption of the application is 374.58 MB, including the whole pipeline. These characteristics allow the proposed model to be embedded in user-friendly mobile applications as modern mobile devices have RAMs in the order of gigabytes.

Currently, the biggest limitation of the proposed model is the recognition of only 21 out of the 26 letters of the Portuguese alphabet. Future work includes the addition of the remaining letters, which require dynamic symbols, so it can recognize the whole alphabet. This is a much harder task, since it requires analyzing a sequence of images, instead of just one. This comes with several challenges, including the different amounts of frames required for a person to do the symbol. For example, considering a frame rate of 30 FPS, if a person makes the symbol in one second, it needs to analyze 30 frames; however, if someone who is a beginner and tends to do it slower might do it in two seconds, requiring 60 frames. Another interesting addition are the digits 0-9. These digits also complement the application well, since they appear commonly in conversations, and are all static in Libras, so they are not as complicated as the remaining letters. Together

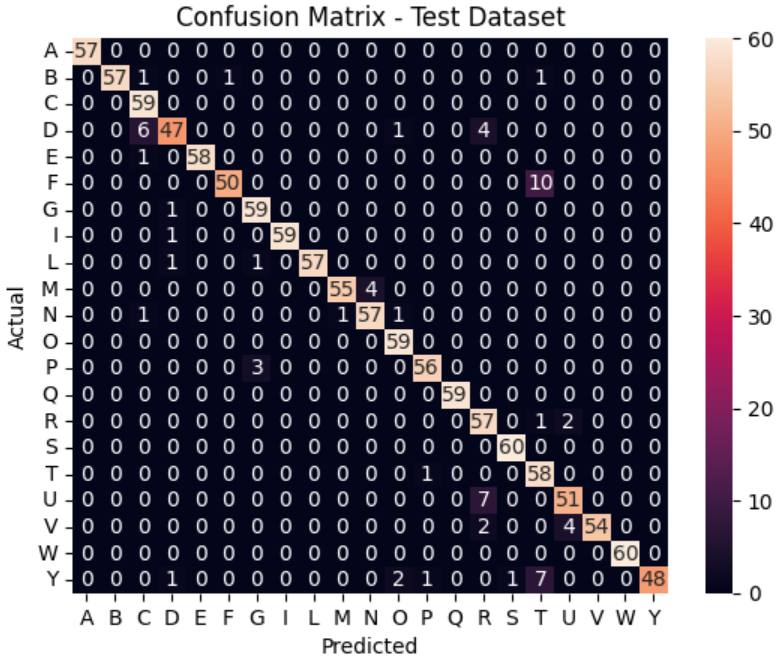


Figure 5. Confusion Matrix of SVM 1

with the digits, other mathematical symbols, like the operational symbols (+, -, /, . . .) are good options. However, most of them either are dynamic, requiring video analysis, or use more than one hand, requiring multiple hands to be analyzed at once.

Acknowledgements

This work was made possible thanks to the members of PET Computação UFPR and the volunteers who helped create the Libras dataset used for our models. This work was supported by the FNDE organization.

References

Benevides, M. P., Xavier, K. R. S. L., Silva, A. P. S., Junior, O. R., Villarta, C. J. B., Richetto, M. R. S., and de Moura, R. A. (2024). Sign talk assistive technology: real-

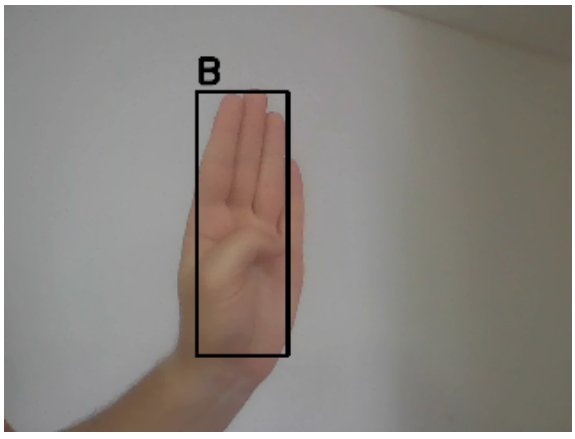


Figure 6. Basic User-friendly Application

time recognition of the libras typical alphabet using artificial intelligence. *Revista de Gestão Social e Ambiental*, 18(12):1–13.

Bishop, C. M. and Bishop, H. (2019). *Deep Learning: Foundations and Concepts*. Springer Nature.

Brasil (2002). Lei nº 10.436, de 24 de abril de 2002. Dispõe sobre a Língua Brasileira de Sinais - Libras. Diário Oficial da União, Brasília, 25 abr. 2002. Acessado em: 17/03/2025.

Caiafa, E. G., Fonseca, F. F., Lima, A. A., Araujo, G. M., and da Silva, E. A. (2020). Aprendizado profundo no reconhecimento de sinais estáticos de libras. In *Proc. 38th Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, pages 1–5.

de Morais, L. M., Almeida, W. M., and Rego, R. C. (2025). Machine learning approaches for efficient recognition of brazilian sign language. In *Simpósio Brasileiro de Sistemas de Informação (SBSI)*, pages 49–56. SBC.

Furtado, S. L. and De Oliveira, J. (2021). Computer vision and neural networks for libras recognition. In *Workshop de Visão Computacional (WVC)*, pages 137–142. SBC.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.".

Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520.

Li, Y., Chen, X., Li, H., Pu, X., Jin, P., and Ren, Y. (2025). Vsnet: Focusing on the linguistic characteristics of sign language. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24320–24330.

Quaresma, P. and Pinho, A. (2007). Análise de frequências da língua portuguesa. In *Livro de Actas da Conferência Ibero-Americana InterTIC*, pages 267–272.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Santos, M. F. O. d. M. et al. (2022). Classificação de libras em imagens através de redes neurais convolucionais.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, San Diego.

Trabelsi, L., Harrouch, H., Mohamed, S., Jebali, M., and Sharma, A. (2025). Advancements and challenges in vision-based sign language recognition: A comprehensive review. *Information Fusion*, page 103626.

Vakunov, A., Chang, C.-L., Zhang, F., Sung, G., Grundmann, M., and Bazarevsky, V. (2020). Mediapipe hands: On-device real-time hand tracking. In *Workshop on Computer Vision for AR/VR*, volume 2, page 5.