MDPI

Article

Neural Networks applied to Virtual Environment with Human Motion Detection

José Machado [1,‡], Tiago Jesus [2,‡], Guilherme Viveiros[4,‡], Luis Macedo[4,‡], Guilherme Andrade[4,‡] and Regina Sousa [3,‡]

[1] ALGORITMI Research Center, School of Engineering, University of Minho, Gualtar Campus, 4710 – 057 Braga, Portugal;jmac@di.uminho.pt;
[2] ALGORITMI Research Center, School of Engineering, University of Minho, Gualtar Campus, 4710 – 057 Braga, Portugal; tiago.jesus@algoritmi.uminho.pt;
[3] ALGORITMI Research Center, School of Engineering, University of Minho, Gualtar Campus, 4710 – 057 Braga, Portugal; regina.sousa@algoritmi.uminho.pt;
[4] School of Engineering, Informatics Department, University of Minho, Gualtar Campus, 4710 – 057 Braga, Portugal;
* Correspondence: jmac@di.uminho.pt;
‡ These authors contributed equally to this work.

Version September 3, 2020 submitted to Sensors

Abstract: Nowadays, all applications are developed with the user's comfort in mind. Regardless of the sector of activity where an application is framed, it should be as simple as possible so that it is easily accepted by the users. With the evolution of technology, simplicity evolved and became intrinsically connected with the automation of tasks. While simplicity was "a click away" a few years ago, it has become "a gesture away" nowadays, eliminating the need for physical interaction between humans and machines. This work presents a pipeline of artificial neural networks that allow the detection of movement in various environments. One of the main objectives is to detect in real-time which gestures are made by the user in order to perform a particular operation in, for instance, the user's computer.

Keywords: Gestual Commands; Computer Vision; Machine Learning; Deep Learning; Neural Networks; Virtual Environment; Desktop Task Simulator

1. Introduction

1.1. Contextualization

Regardless of the objective of an application, it should be as simple as possible so that it is easily accepted and by the users. As technology progressed, the concept of simplicity evolved and became intrinsically connected with the automation of tasks. While "a click away" was considered as simple a few years ago, nowadays "a gesture away" would be simpler, making it possible to do more with less effort and without the need for physical interaction with a machine. With that in mind, this paper introduces an environment with a new strategy with the potential to change the way some operations are made in a computer, whether through hotkeys, mouse, or some other external device. All of the before mentioned can be replaced by gesture interactions. The usefulness of the provided virtual environment depends entirely on the user. However, there are several cases when this method can be useful:

- When cooking, if the user is following a recipe on the laptop, having dirty hands requires extra work (cleaning / drying hands). Using gestures would allow the user to perform some actions on the laptop without getting it dirty.
- The user may be using external monitors and the computer may be far away.
- If the user doesn't have an external device like a mouse to help them with the tasks shown here and they didn't know how to do them otherwise.

1.2. Objectives

Taking into account all the evidence mentioned in the previous section, several objectives were outlined according to a logical sequence of work.

The main objective of building the described environment is the development of a neural network pipeline that allows the detection of movement, in various environments. Furthermore, it is intended that the same pipeline determines which operation the user intends to perform according to the movement he has made. Besides these two main objectives it is intended to capture in real time the user's movement, also processing in real time all those frames and still perform the operation previously determined in the most varied operating systems.

1.3. Core Concepts - Deep Learning - Deep Artificial Neural Networks

Deep Learning (DL) is an emerging term that characterizes a branch of Machine Learning (ML),one of the subfields of Artificial Intelligence (AI) [1][2], with the objective of training computers to perform tasks in a similar way to humans [3][17,18]. This sub field of ML is dedicated to algorithms with the ability to model high-level abstractions, such as data from images, text or even sound. DL's main structural and functional inspiration is the human brain. The associated algorithms are called neural networks [4][5].

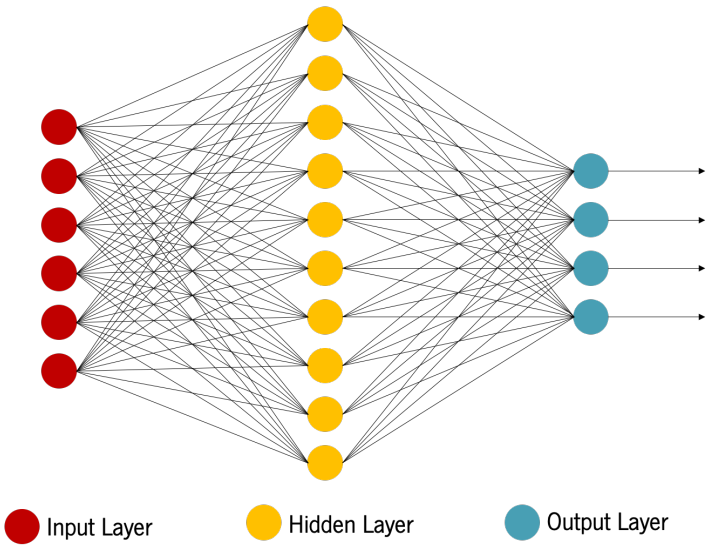

Figure 1. Simple Neural Network.Adapted from [6].

It is common knowledge that neural networks are very powerful algorithms used in various problems. These DL algorithms not only far surpass other kinds of ML algorithms, but also rival the accuracy achieved by humans [2]. This is because a neuronal network can be generically described as a set of nonlinear functions with the objective of modeling, as we can see in figure 1, a type function Y $\approx$ g(X).

In terms of functionality, the input layer is used to receive input data from the network. This layer does not perform any kind of processing, it only receives and stores the input vector. The number of

neurons in this layer corresponds to the input vector. The output layer is used to store the responses obtained by the network. The number of neurons in this layer corresponds to the output vector. The hidden layers are what give the complexity to this process. There is no method or formula to determine the correct number for the hidden layers and neurons [21].

Being this a problem of supervised learning, the input and output layers are provided to the network with the purpose of modeling a function g, which is represented by the weights in the middle layer. As a rule, the greater the complexity and extent of the problem the more intermediate layers are contained in a neuronal network, hence the origin of the name deep in DL, deep neuronal networks [16].

In addition to the type of architecture presented here, there are others, such as convolutional networks [9] and recurring networks [13][20], each appropriate for a given task.

Convolutional networks were part of the solution developed because of their excellent framework for image classification. These neural networks can preserve the relationships between the pixels of a image by learning the appropriate features using small squares, known as filters in the input, as can be seen in figure 2. Moreover, since a lot of data sets and complex problems were worked on, this proved to be the best choice [18].

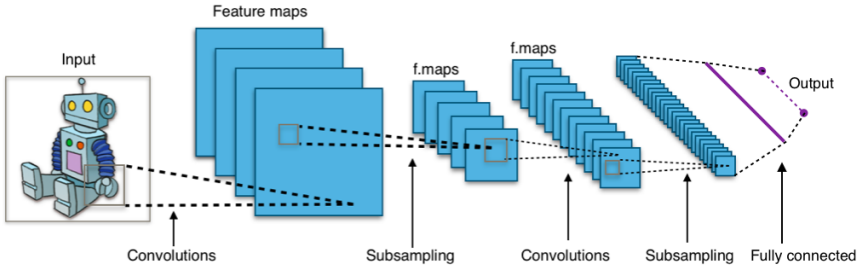

Figure 2. Convolutional neural networks.

The convolutions work as filters that perceive small squares and progress throughout the image capturing the most striking features. More specifically, for a 64x64x6 image and a filter that covers a 5x5 area of the image with 4 jumps (called a stride), the filter will pass through the entire image, through each of the channels, forming at the end a feature map or activation map of 56x56x1.The depth of a convolution's output is equal to the amount of filters applied. The deeper the convolution layers, the more detailed the traces identified with the activation map.

Therefore, there are 5 basic terms to describe a neuronal network [4] [7]:

- Size: Number of knots;
- Width: Number of nodes of a layer;
- Depth: Number of layers;
- Capacity: Functions that can be learned by a network;
- Architecture: Arrangement between the layers and the nodes of a network.

2. Materials and Methods

2.1. Gesture Detection and Recognition

In this section the detection of movements through the pipeline of neural networks is described.

The developed pipeline has some similarities with personal assistants such as Alexa or Siri, in which the detection of a specific word occurs initially. For example "Hey Siri" triggers the Siri assistant and only then the following text is processed. Likewise, the developed environment is triggered when it detects a hand with a certain confidence and then processes the given movement, which may or may not be a known gesture to be executed by the operating system.

2.1.1. Trigger Hand

At an early stage in the process two proposals were taken into account. The first was based on the creation of a dataset and the second on the search for a dataset appropriate to the task at hand.

Taking into account the fact that the group is small in relation to the amount of data needed for a deep learning algorithm, it was decided to look for a dataset.

The approach that was taken for the project did not ease the search for a dataset since the trigger hand is not a binary ranking problem, but a bounding box problem. This means that the algorithm does not only detect whether the hand is present, but it also detects its position.

Firstly, it should be made clear that, although the approach adopted increases the complexity of the problem, it is advantageous in that it is intended to continue the project by adding several operations where it is possible to interact with the operating system through a gesture. For example, to scroll continuously, to click on a certain operation on a web page, to open a certain program, among many other actions. All of the above require spatial coordinates, which can only be provided by the proposed method.

2.1.2. Dataset

Taking into account everything already described in this paper, the dataset chosen had to be the most similar to the environment in which this system will operate, in this case users who are in front of a computer and where the quality of the camera, brightness and adjacent noise can vary innumerably.

The first experiment was carried out with a set of data provided by Oxford, called the Hands Dataset: http://www.robots.ox.ac.uk/ vgg/data/hands. However, the result was not what was desired and so it turned to researching an alternative set. Given the scarcity of research and availability carried out so far in this area, movement recognition for desktop (without the use of sensors), a dataset not very appropriate to the terms described above as we can see in figure 3, but where the results obtained are satisfactory, was used to show the efficiency of this prototype, Egohands Dataset: http://vision.soic.indiana.edu/projects/egohands.

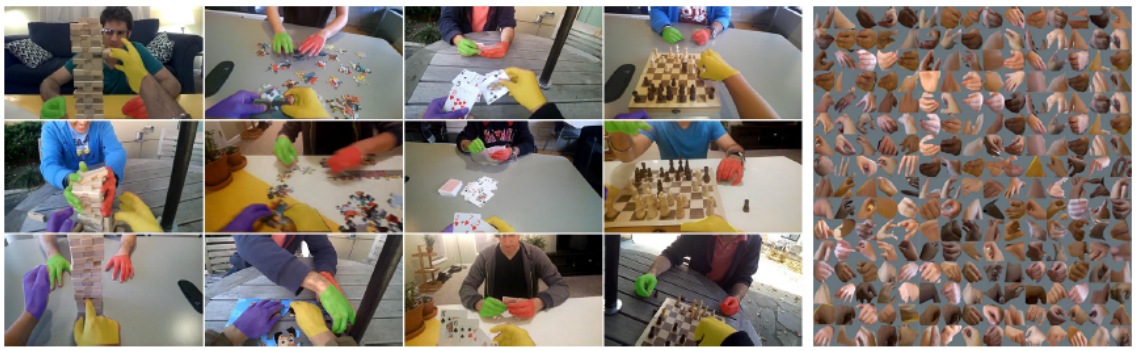

Figure 3. Egohands Dataset.

This dataset fitted quite well for several reasons. Besides containing a large amount of data (15,053 hands with bounding boxes), it contains high quality images and the possibility to distinguish between the hands of the user and adjacent people, such as the distinction between left and right hands, which for the intended environment is fundamental. In addition, it provides a wide range of distinct spaces where they do the most diverse activities, thus providing a more complex and robust dataset.

Regarding the preparation of the dataset, it was decided to use Tensorflow as a library for all methods used in the hand detection process. Having the dataset ready to use, a Python script was developed for the transformation of the set into tfrecords, which is what Tensorflow uses to train the models.

It was then decided to use 80% of the data as a training set, 10% for validation and the remaining 10% for testing, thus creating 3 tfrecords.

2.1.3. Model Used

Followed by the preparation of the dataset is the model's theorem which will be applied to it. For this process neuronal networks weighted through a process called Transfer Learning were used.

The first step to understand this process is to know that this concept is not new specific to deep learning. There is a stark difference between the traditional approach of building and training machine learning models, and using a methodology following transfer learning principles[17].

Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task.

This means that we can use an existing model, which have already been trained in a similar area or situation (in this case image detection) and modify the last layers to our benefit.

Since neural networks can contain millions of parameters and the training process can take days, weeks or even months, Transfer Learning proves to be a very interesting solution to help reduce the time to hours. In addition, the choice of model must take into account the performance of the environment in question. In other words, going back to Siri's example, being this a good analogy, the model is always waiting for the trigger word, "Hey Siri", and it is processing what it receives at all times. The model described here has the same concept, all captured frames are analyzed for potential triggers.

Having said that, we looked for a model already trained in this area and that is as fast as possible. Fortunately Tensorflow already offers some models, and it was decided to use the ssd_-mobilenet_v1_coco model as a starting point since it is one of the fastest current models for this sort of task [11].

2.1.4. Training environment

Model training can be done locally, on local GPUs that would take some time, or in the cloud. For the sake of time optimization, it was chosen to use the cloud. By reference only, the model training on a MacBook Air (Intel i5 2.6GHZ, 8GB RAM), ran at a maximum speed of 8 seconds with CPU and 3 seconds with GPU. It would take about 17 days to run approximately 200,000 epochs on the Mac, mentioned above, compared to 3 to 4 hours on the cloud. Therefore, GoogleColab was used as a platform. It's not the best choice but it was chosen because it's free software and varies the GPU in use so you can not tell which GPU is used.

As all this model and successive API is provided by Tensorflow, Tensorboard was used in the visualization of the models, being the most significant parameter the loss of the same in the test dataset. Given the use of the Tensorflow Object Detection platform API for image detection, the training graphics were not made available, being which will only exemplify the loss of the model in the test set.

The model was left running in 200,000 epochs and the result can be seen in the following figure 4 shows a 0.5 smooth.

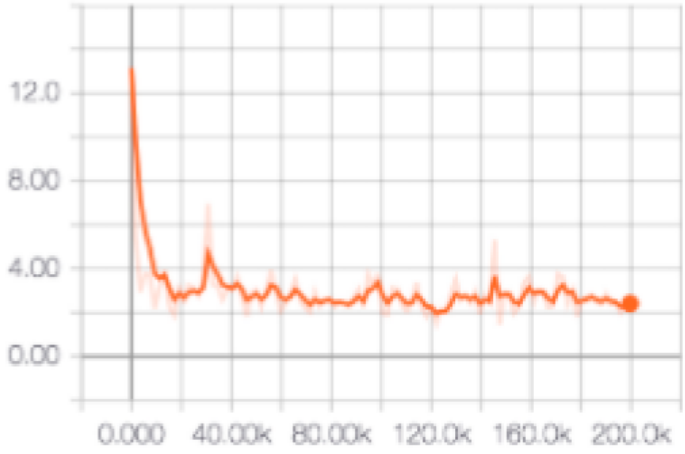

Figure 4. Model Loss.

After the training is complete, an inference graph (among other files) is created and then exported to all employees so that they can perform the detection of the hand.

After the model was completed, it was adapted to capture frames in real time and the result can be seen in the following figures.

Figure 5. Hand detection model.

2.1.5. Movement Detection

A vital part of the whole project is the recognition of the operation to be performed, demonstrated and performed by a user in real time. This is the only way this project differs from the others that only detect whether or not the hand is in the camera frame, without performing any action after that.

The recognition of the operation to be performed was made by a trained neural network, incorporated into the system.

The first step for the construction of this network was, once again, the search for the data set, previously processed and treated, that could be used. After some searching and consultation, the 20BN-JESTER: https://20bn.com/datasets/jester., shown in figure 6, was found. This dataset has a large collection of video excerpts from several individuals to perform predefined gestures, where each excerpt translates into a set of frames.

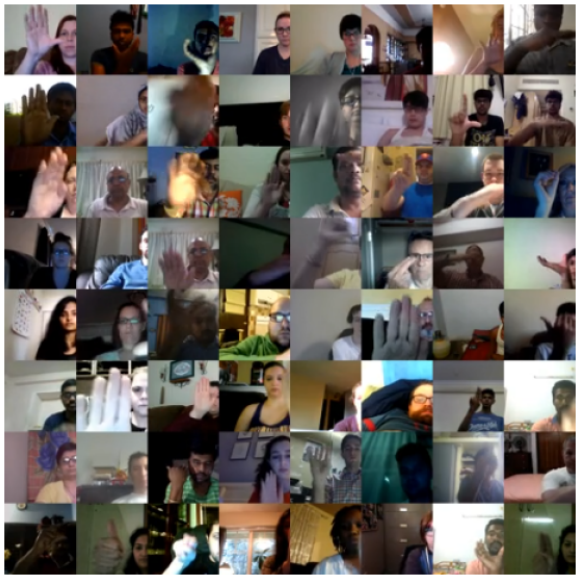

Figure 6. Frame samples of 20BN-JESTER.

Each excerpt would be composed on average of 36 frames, the result of processing the original video from which 12 frames per second were extracted. This implies that any gesture has a 3 second time window to be executed, although most frames are transitive movements for the beginning or end of the gesture, the gesture itself has a relatively small presence, in terms of the number of frames as can be seen in figure 7.

Figure 7. Frame sequence of the zooming gesture.

As in the previous image, the gestures to be recognized are also found in this collection. For this reason we have chosen to use this dataset as a basis. Then, from the whole dataset, the clips corresponding to four different classes were used: swiping left, swiping down, zooming in with full hand and doing other things, this last class is applied when none of the gestures are recognized. To complete the dataset some personal videos were created in the image of the existing ones.

The personal videos were processed in the same way, regarding the number of frames (12 per second), however a feature was added that allowed to increase exponentially the size of the data set: each gesture executed in the video would have 4 seconds of transition actions between them, which allows that for the same instance of the same gesture, it can be presented somewhere at the beginning, middle or end of the 3 seconds established by 20BN-JESTER. A sliding window style script has been developed which, having marked the first and last exact frame corresponding, respectively, to the beginning and end of the gesture, makes a sequential sampling that puts this exact frame in the possible positions throughout the 3-second video time. This has become quite useful because, besides increasing the number of data, in this case being a process of data augmentation [10], a gesture can theoretically be detected correctly if placed in any position of the clip.

198 ## 2.2. Data Processing

199 Given the enormous size of the dataset, resulting from the joining of the 20BN-JESTER and the
200 manually created dataset, which as a whole occupied among the 20GB, Out-of-core Training was used
201 [12][21].
202 Out-of-core Training is a method used when it is practically impossible to train the models in the
203 entire dataset, since there is not enough memory in the machine, among other cases, and this form,
204 allows the data to be available in a sequential way at each step [12].
205 That said, a customized generator was created, also with the keras API in order to send X in X
206 gestures to the memory, where X represents the batch-size predefined.
207 As previously mentioned, 20BN-JESTER contains gestures that can be lower or higher than 36
208 frames, so it was decided to set the maximum frame limit to 36, corresponding to 3 seconds, and if any
209 gesture contained less than that, a masking was performed, if it was higher it was cut to 36 frames.
210 Besides that, all frames are reduced to (100,250) pixels.

211 ### 2.2.1. Model Used

212 So it was necessary to think about the creation and design of the architecture to use for the
213 convolutional network. In this process the possibility of using pre-trained models was considered,
214 but an original model was chosen so that it could contain specific characteristics of the original data.
215 The designed architecture was inspired by an existing C3D[11], which uses several 3D convolutional
216 layers, pooling and dense to remove spatial and temporal attributes from each image. The original
217 architecture presents considerable complexity, which led to the decision to use a relatively simpler
218 approach to the problem in question. The architecture of the model can be seen in figure 8.
219 It has always been a priority to use an architecture with as few parameters as possible, since it was
220 intended to be used in real time. The simplification from the original model was based primarily on the
221 reduction of units in each layer, maintaining proportionality between the new numbers. It is important
222 to note that the model presented was not the original, the reduction of units was gradual. What was
223 sought was a decrease in the complexity of the model without compromising the precision values
224 pointed out by the model, values that remained constant until the order of eight hundred thousand
225 parameters.

Layer (type)	Output Shape	Param #
conv1 (Conv3D)	(None, 18, 38, 50, 16)	1312
pool1 (MaxPooling3D)	(None, 18, 19, 25, 16)	0
batch_normalization_13 (Batc	(None, 18, 19, 25, 16)	64
dropout_13 (Dropout)	(None, 18, 19, 25, 16)	0
conv2 (Conv3D)	(None, 18, 19, 25, 32)	13856
pool2 (MaxPooling3D)	(None, 9, 9, 12, 32)	0
batch_normalization_14 (Batc	(None, 9, 9, 12, 32)	128
dropout_14 (Dropout)	(None, 9, 9, 12, 32)	0
conv3a (Conv3D)	(None, 9, 9, 12, 64)	55360
conv3b (Conv3D)	(None, 9, 9, 12, 64)	110656
pool3 (MaxPooling3D)	(None, 4, 4, 6, 64)	0
batch_normalization_15 (Batc	(None, 4, 4, 6, 64)	256
dropout_15 (Dropout)	(None, 4, 4, 6, 64)	0
conv4a (Conv3D)	(None, 4, 4, 6, 128)	221312
conv4b (Conv3D)	(None, 4, 4, 6, 128)	442496
pool4 (MaxPooling3D)	(None, 2, 2, 3, 128)	0
batch_normalization_16 (Batc	(None, 2, 2, 3, 128)	512
dropout_16 (Dropout)	(None, 2, 2, 3, 128)	0
zeropad5 (ZeroPadding3D)	(None, 2, 3, 4, 128)	0
pool5 (MaxPooling3D)	(None, 1, 1, 2, 128)	0
flatten_4 (Flatten)	(None, 256)	0
fc8 (Dense)	(None, 4)	1028

Total params: 846,980
Trainable params: 846,500
Non-trainable params: 480

Figure 8. Model Architecture.

2.2.2. Platform used and model training

The machine used was supplied by the University of Minho, Department of Informatics (DI), which by reference, was a Nvidia Quadro P6000 GPU, with 24 GB of memory and a Ram size of 62.8 GB, which allowed the tuning, in a few days, of what would be the final model.

After passing the dataset to the respective server, in a few hours it was possible to obtain results and analyze in order to make potential changes to the model.

For the training of the model, the Keras platform was used, characterized by its high level API's that allow the production of models at a high pace.

The demonstrated architecture has been trained with the optimizer SGD with momentum and with a learning rate decay. It's trained for 50 epochs, in which callbacks would be used to save only the best models in terms of validation set accuracy.

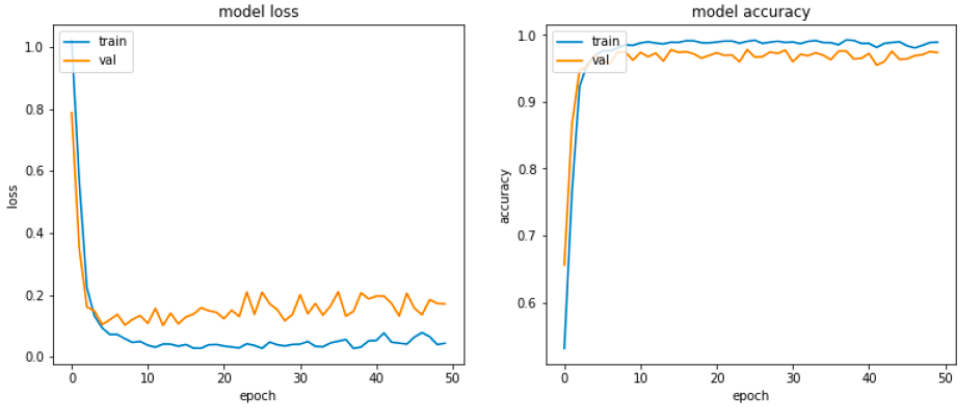

Figure 9. Accuracy and Model Loss.

When observing the learning curves, presented in figure 9, it is clear that, quickly (in relation to the total number of epochs), satisfactory accuracy values are achieved within the first dozens of iterations to the dataset. The callback used to store only the best models last recorded the model of epoch number 13, which indicates that from the second dozen epochs the continuation of the training is no longer efficient, and the final model reaches an accuracy in the validation set of 96% and in the training set of 98% for gesture detection.

2.3. Real-time Movement Capture

In order to identify the gesture made by users, it is necessary to capture them in real time. This capture is made thanks to the OpenCV library, which allows access to any camera connected to the computer.

Using OpenCV means taking the maximum number of frames that a given computer can take per second, also known as fps. In order to counter this act predefined by the library, because we only want 12 frames per second in order to maintain consistency with the dataset used, the following architecture was structured:

Figure 10. Frame allocation.

Each square represented in figure 11 represents 1 second and as can be seen, the first square is the past, the second the present and the third the future, in this way you get a time array with the user's real-time frames.

In this case the first square contains the past X frames, the second one contains the X current frames and the third square the X frames that are still to come in the next second, where X is dependent on the number of fps of the computer where processing takes place. The advantage of this type of architecture is that the number of frames desired can be changed at any time, in this case only 12 of those to which the computer you can pull out in a second. For this we use dynamic strides in the choice of successive frames.

Furthermore, the trigger word model fits perfectly in the following architecture, because when the hand is detected, automatically the user is in the second square, the previous 12 frames are already present and the user only has 1 second to perform the gesture, which after several tests was considered

sufficient. For all intents and purposes these frames can also be modified, that is, it is possible to have more time to perform the gesture, however, it increases the inconsistency before the dataset used here.

Being already clarified the part of capture and allocation of frames in real time, it is specified the modeling used in the arrays described above.

Threads (form of a process to divide itself into two or more tasks that can be performed competitively) and locks (variables associated to data that determine if operations can be performed) are used to organize the access to resources, in this case the temporal array used to allocate the frames.

In total there are 3 Threads:

- Main Thread, which tries to create the other Threads and executes the main process, in this case, initialize the hand and gesture detection model;
- Capture Thread. This Thread accesses the camera of the user's laptop and captures images, which sends to the frame allocation Thread.
- Frame allocation thread, which allocates them in the respective array and pre-processes them to be in accordance with the data used in the model training.

The capture Thread, after capturing an image, signals the main Thread to verify if the captured image contains a hand, if not the process continues, in this case the Thread continues to collect frames, if it contains, the Thread of allocation of frames waits for the following frames that are still to come from the second that is missing and then the main Thread uses the model to detect if there is any gesture contained in the 36 frames.

3. Execution of the operation

With the network trained and ready to stop hands and classify the gestures, all that remains is to execute the process that the gesture represents (minimize, zoom in or swipe left). For this it was decided to simulate hotkeys.

The simulation is done using packages in Python that allow you to simulate pressing of keyboard keys. Therefore, all that remains is to find out which hotkeys to carry out the processes related to the gestures in the different operating systems. As for example, in Windows 10 to switch to the right desktop (swipe left) the key combination is used: WIN + CTRL + right arrow.

All the tests made in order to test the hotkeys in MacOS, were executed in the MacOS Sierra version.

It's important to mention that different hotkeys to the implemented ones can be added easily, changing the source code, due to its modality.

4. Conclusions and Optimizations

Decisions to build a prototype were made throughout this work. These decisions had a practical and effective demonstration of the goal in mind, that is, understanding and remotely executing commands issued by a human entity without the help of equipment.

The use of an existing dataset as a conceptual basis for the definition of the gesture enables a better degree of normalization in relation to a personalized data set, since this is better structured in terms of quantity and quality of the samples. However, this measure is inherently a limitation because it limits the training process to the reality imposed by the data that was not defined by the authors.

As mentioned above, a set of data has been added to the existing dataset. The aim is to create a completely original data set with the desired features such as the speed of the gesture and the frame window in which the gesture is carried out (start, middle or end). Note that the 20BN-JESTER should be used for this purpose with some manipulation of the existing frames through data sampling strategies that have been designed and already idealized.

The implementation of the pipeline itself, which was developed from scratch, has extremely adaptable properties for various machine types in terms of performance and frames per second, which have a major influence on the designed system. The pipeline was built on the basis of dynamic arrays

that complement each other and support the creation of a necessary background for the model created. There is a lot of room for improvement in terms of threading, efficiency and flexibility. These aspects of improvement focus primarily on capturing and handling images that are different for each machine. It is necessary to put together the various aspects that contribute to the portability and viability of what could be a commonly used application.

Python's choice for implementation was based on the ease of use and simplicity of the language, mainly in the area of Machine Learning. The simplification as well as the compatibility with various required platforms (such as Tensorflow, Keras, OpenCV) made Python the obvious choice for a first creation. However, the simultaneous use in relation to this language is not the most accurate as it is an interpreter. With this in mind, we intend to use optimized compilers like Numba or a transition to C ++.

The implementation of only a small set of gestures is a very important feature as it allows for more efficient maintenance and optimization / debugging to find flaws in the pipeline. However, when developing and improving it is essential to expand the vocabulary associated with the commands that can be interpreted by the model. This is a measure that needs to be developed along with the personalized data to allow better immersion in what would be a motion driven desktop.

Using a more appropriate dataset for hand recognition or its construction is an important step in improving the environment because the dataset used does not have all the features of real cases, making system performance impractical in certain cases.

Regarding the execution of the process regarding the performed gesture, one possible optimization is to switch from using the simulation (simulate the event of pressing the buttons) to the use of direct calls to the operating system, i.e. instead of using key simulations directly execute the desired process. This optimization would be achieved with the programming language C ++ and would be what connects the application with the operating system.

Author Contributions: Each of the authors made substantial contributions to the conception of the article, approving pleasantly the submitted version.

Conceptualization, R.S., T.J., G.V., L.M., G.A., and J.M.; Methodology, R.S., T.J., G.V., L.M., G.A., and J.M.; Software, G.V., L.M. and G.A.; Validation, R.S, T.J. and J.M.; Formal analysis, R.S. and T.J.; Investigation, R.S., T.J., G.V., L.M., G.A., and J.M.; Resources, J.M.; Data Curation, R.S., T.J. and J.M.; Writing–original draft preparation, R.S. and T.J.; Writing–review and editing, R.S.; Visualization, R.S., T.J.; Supervision, J.M.; Project Administration, J.M.; Funding Acquisition, J.M.

Funding: "This research as an external funding"

Acknowledgments: This research has been supported by FCT - Fundação para a Ciência e Tecnologia whithin the R&D Units Project Scope: UIDB/00319/2020

Conflicts of Interest: ""The authors declare no conflict of interest.""

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
DL	Deep Learning
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine Learning

References

1. Chollet, F. (2018). Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek. MITP-Verlags GmbH & Co. KG.
2. Buduma, N.,& Locascio, N. (2017). Fundamentals of deep learning: Designing next-generation machine intelligence algorithms. " O'Reilly Media, Inc.".
3. Bengio, Y., Goodfellow, I., & Courville, A. (2017). Deep learning (Vol. 1). Massachusetts, USA:: MIT press.
4. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

5. Alpaydin, E. (2020). Introduction to machine learning. MIT press.
6. Samani, N., Gohari-Moghadam, M., and Safavi, A. (2007). A simple neural network model for the determination of aquifer parameters. Journal of Hydrology, 340(1-2):1–11.
7. Da Silva, I. N., Spatti, D. H., and Flauzino, R. A. (2010). Redes neurais artificiais para engenharia e ciências aplicadas curso prático. São Paulo: Artliber
8. Chung, H., Iorga, M., Voas, J., & Lee, S. (2017). Alexa, can I trust you?. Computer, 50(9), 100-104.
9. Jin, K. H., McCann, M. T., Froustey, E., & Unser, M. (2017). Deep convolutional neural network for inverse problems in imaging. IEEE Transactions on Image Processing, 26(9), 4509-4522.
10. Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.
11. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.
12. Sjardin, B., Massaron, L., & Boschetti, A. (2016). Large Scale Machine Learning with Python. Packt Publishing Ltd.
13. Mandic, D., & Chambers, J. (2001). Recurrent neural networks for prediction: learning algorithms, architectures and stability. Wiley.
14. Kaplan, A., & Haenlein, M. (2019). Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. Business Horizons, 62(1), 15-25.
15. Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the IEEE international conference on computer vision (pp. 4489-4497).
16. Ferreira, D., Silva, S., Abelha, A., Machado, J. (2020). Recommendation System Using Autoencoders. Applied Sciences, 10(16), 5510.
17. Aqra, I., Abdul Ghani, N., Maple, C., Machado, J., Sohrabi Safa, N. (2019). Incremental Algorithm for Association Rule Mining under Dynamic Threshold. Applied Sciences, 9(24), 5398.
18. Neto, C., Brito, M., Lopes, V., Peixoto, H., Abelha, A., Machado, J. (2019). Application of data mining for the prediction of mortality and occurrence of complications for gastric cancer patients. Entropy, 21(12), 1163.
19. Brito, C., Esteves, M., Peixoto, H., Abelha, A., Machado, J. (2019). A data mining approach to classify serum creatinine values in patients undergoing continuous ambulatory peritoneal dialysis. Wireless Networks, 1-9.
20. Pereira, S., Portela, F., Santos, M. F., Machado, J., Abelha, A. (2015). Predicting type of delivery by identification of obstetric risk factors through data mining. Procedia Computer Science, 64, 601-609.
21. Neves, J., Vicente, H., Esteves, M., Ferraz, F., Abelha, A., Machado, J., ... Sampaio, L. (2018). A deep-big data approach to health care in the AI age. Mobile Networks and Applications, 23(4), 1123-1128.