Real Time Implementation of a Machine Learning Model Sign Language Recognition System Using Human Pose Estimation

Kristian Yair González Márquez

A Thesis Submitted in Partial Fulfilment

of the requirements for the

Degree of

Master of Science in Data Analytics



September 2023

Supervisor: David McQuaid

**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | Capstone Project |
| **Assessment Title:** | Real Time Implementation of a Machine Learning Model Sign Language Recognition System Using Human Pose Estimation |
| **Lecturer Name:** | David McQuaid |
| **Student Full Name:** | Kristian Yair González Márquez |
| **Student Number:** | 2022421 |
| **Assessment Due Date:** | 24/09/2023 |
| **Date of Submission:** | 24/09/2023 |

**Declaration**

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

# ABSTRACT

This study focused on American Sign Language Recognition. It starts with a thorough literature review that led to the combination of MediaPipe and two algorithms based on the MMPose framework with different Neural Network architectures composed of CNN, GRU, and LSTM layers.

Experimentation was the research methodology using stratified sampling, 30 groups combining NN architectures and HPE algorithms were identified to be tuned using Keras Tuner Hyperband to achieve the highest validation F1 Score. Human joint coordinates were extracted from the fifty glosses with more samples available in the Microsoft ASL Citizen dataset stored in the form of NumPy arrays for training and validation. After extensive model tuning, the best performing model achieved 86.3% validation F1 Score with four hidden layers composed of 2 1D-CNN, 1 GRU, and a Fully Connected layer fed by MediaPipe with the hands positions information, this combination was consistent for the three HPE algorithms the best performer.

To validate the model's capacity to classify unseen data, five videos per sign were recorded by the researcher simulating real life situations with different light conditions and camera angles, the developed NN reached 81.9% F1 score in the unseen data even when overfitting was present in the model. The experimentation ends by evaluating its inference speed finding that the model combined with MediaPipe can predict between 25 to 30 FPS which is a satisfactory speed achieved to consider a real time application in the future, however, it is pointed out that this is a theoretical exercise as no actual Sign Language users were involved and further research for improvement and evaluation need to be done for its implementation in real life.

# ACKNOWLEDGEMENTS

This project is a collaborative effort in which many people participated by reading, opining, correcting, being patient, giving hope, and accompanying me in times of crisis and joy.

Thank you to my parents, grandparents, and brothers for their love, affection, and understanding throughout my life, and for unconditionally understanding my absences and bad moments, even at the distance I know they are always supporting me.

Thanks to Alejandra Linares for agreeing to join me on this adventure of travelling to an unknown country to study for a master's degree.

I thank CCT College for encouraging me all the time to do my best in each project until the culmination of my degree, each of the lecturers who contributed to my professional development that have shown great patience and willingness to teach me, sharing their knowledge and experiences, also is important to mention David McQuaid for having trust in me, for his patience and invaluable guidance on this project and during the whole degree programme.

**INDEX**

[ABSTRACT I](#_Toc146442153)

[ACKNOWLEDGEMENTS II](#_Toc146442154)

[FIGURES INDEX IV](#_Toc146442155)

[TABLES INDEX V](#_Toc146442156)

[LIST OF ABBREVIATIONS VII](#_Toc146442157)

[CHAPTER 1 INTRODUCTION 1](#_Toc146442158)

[1.1 Background of Study 1](#_Toc146442159)

[1.2 Objectives 1](#_Toc146442160)

[1.3 Scope of Study and Limitations 2](#_Toc146442161)

[CHAPTER 2 LITERATURE REVIEW 3](#_Toc146442162)

[2.1 Sign Language 4](#_Toc146442163)

[2.2 Object detection 6](#_Toc146442164)

[2.3 Human Pose Estimation 7](#_Toc146442165)

[2.4 Deep Neural Networks 9](#_Toc146442166)

[2.5 Activation Functions 12](#_Toc146442167)

[2.6 Sign Language Recognition Systems 19](#_Toc146442168)

[2.7 Challenges 22](#_Toc146442169)

[2.8 Conclusion 23](#_Toc146442170)

[CHAPTER 3 METHODOLOGY 24](#_Toc146442171)

[3.1 Ethical Considerations 24](#_Toc146442172)

[3.2 Sampling and Primary Research Method 25](#_Toc146442173)

[3.3 Dataset Description 26](#_Toc146442174)

[3.4 Exploratory Data Analysis 27](#_Toc146442175)

[3.5 Keypoint Collection 32](#_Toc146442176)

[3.6 Neural Network Design 34](#_Toc146442177)

[CHAPTER 4 RESULTS 38](#_Toc146442178)

[4.1 MediaPipe 38](#_Toc146442179)

[4.2 RTMPose 50](#_Toc146442180)

[4.3 CSPNext + UDP 56](#_Toc146442181)

[4.4 Real Time Implementation 64](#_Toc146442182)

[CHAPTER 5 CONCLUSIONS AND FUTURE WORK 74](#_Toc146442183)

[REFERENCES 78](#_Toc146442184)

[APPENDIX A. TRIALS SUMMARY MEDIAPIPE 86](#_Toc146442185)

[APPENDIX B. TRIALS SUMMARY RTMPOSE 90](#_Toc146442186)

[APPENDIX C. TRIALS SUMMARY CSPNEXT 94](#_Toc146442187)

[APPENDIX D. BEST MODELS PERFORMANCE COMPARISON BETWEEN VALIDATION AND CUSTOM TEST DATA 98](#_Toc146442188)

[APPENDIX E. KERAS TUNER HYPERBAND ALGORITHM CODE CONSTRUCTION 104](#_Toc146442189)

FIGURES INDEX

Figure 2.1 Tanh Activation Function Representation 13

Figure 2.2 ReLU Activation Function Representation 14

Figure 2.3 Leaky ReLU Activation Function Representation 15

Figure 2.4 ELU Activation Function Representation 16

Figure 2.5 GELU Activation Function Representation 17

Figure 2.6 SELU Activation Function Representation 18

Figure 2.7 Mish Activation Function Representation 19

Figure 2.8 Machine Translation methods distribution between 2004 and 2022. Taken from (De Coster et al., 2023) 21

Figure 3.1 Microsoft ASL Citizen Sample, image taken from the official Microsoft ASL Citizen repository 27

Figure 3.2 Frequency Data Distribution in Training Dataframe Top 50 Words 28

Figure 3.3 Frames per Video in Training and Testing Dataframes Top 50 Words 30

Figure 3.4 Frequency Data Distribution in Final Training Dataframe 31

Figure 3.5 Frames per Video in Final Training Dataframe 32

Figure 4.1 2 1D-CNN + 1 LSTM Layers Model Architecture 40

Figure 4.2 2 1D-CNN + 1 LSTM Layers Training History 41

Figure 4.3 3 1D-CNN Layers Model Architecture 42

Figure 4.4 3 1D-CNN Layers Training History 43

Figure 4.5 2 1D-CNN + 1 GRU Layers Trial 721 Model Architecture 45

Figure 4.6 2 1D-CNN + 1 GRU Layers Trial 721 Training History 46

Figure 4.7 3 1D-CNN Trial 715 Model Architecture 47

Figure 4.8 3 1D-CNN Layers Trial 715 Training History 48

Figure 4.9 2 1D-CNN + 1 GRU Model Architecture 52

Figure 4.10 2 1D-CNN + 1 GRU Layers Training History 53

Figure 4.11 2 1D-CNN + 1 LSTM Model Architecture 54

Figure 4.12 2 1D-CNN + 1 LSTM Layers Training History 55

Figure 4.13 2 1D-CNN + 1 GRU Model Architecture 58

Figure 4.14 2 1D-CNN + 1 GRU Layers Training History 59

Figure 4.15 2 1D-CNN + 1 LSTM Model Architecture 60

Figure 4.16 2 1D-CNN + 1 LSTM Layers Training History 61

Figure 4.17 2 1D-CNN + 1 GRU Trial 698 Model Architecture 63

Figure 4.18 2 1D-CNN + 1 GRU Trial 698 Training History 64

Figure 4.19 Custom Sign Videos Sample 65

Figure 4.20 MediaPipe 2 1D-CNN + 1 GRU Layers Trial 721 Model Performance Comparison between Validation and Custom Test Data 66

Figure 4.21 Real Time Detection Interface with MediaPipe 68

Figure 4.22 Real Time Detection Interface with RTMPose / CSPNext 68

Figure 4.23 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU + GPU computer 70

Figure 4.24 RTMPose Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU + GPU computer 73

TABLES INDEX

Table 3.1 Top 50 Glosses with more videos in training ASL Citizen Dataset 28

Table 3.2 Descriptive Statistics of Training and Testing Datasets combined 29

Table 3.3 Descriptive Statistics of Training and Testing Datasets combined after Filtering 31

Table 3.4 Descriptive Statistics of Validation Dataset after Filtering 31

Table 3.5 Models tuned with Keras Tuner 35

Table 4.1 MediaPipe Best Models After Hyperparameter Tuning 38

Table 4.2 MediaPipe Best Models After Training 500 Epochs 39

Table 4.3 MediaPipe Additional Models Best Epoch Checkpoint 43

Table 4.4 MediaPipe Additional Models After Hyperparameter Tuning 44

Table 4.5 MediaPipe Additional Models After MediaPipe Training 500 Epochs 44

Table 4.6 Hyperparameters Comparison Between 3 1D-CNN Top Performer Models 49

Table 4.7 Hyperparameters Comparison Between 2 CNN-RNN Top Performer Models 49

Table 4.8 RTMPose Best Models After Hyperparameter Tuning 50

Table 4.9 RTMPose Best Models After Training 500 Epochs 51

Table 4.10 RTMPose Additional Models Best Epoch Checkpoint 55

Table 4.11 RTMPose Additional Models After Hyperparameter Tuning 56

Table 4.12 RTMPose Additional Models After Training 500 Epochs 56

Table 4.13 CSPNext Best Models After Hyperparameter Tuning 56

Table 4.14 CSPNext Best Models After Training 500 Epochs 57

Table 4.15 CSPNext Additional Models Best Epoch Checkpoint 62

Table 4.16 CSPNext Additional Models After Hyperparameter Tuning 62

Table 4.17 CSPNext Additional Models After Training 500 Epochs 62

Table 4.18 Custom Sign Videos Dataframe Sample 65

Table 4.19 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU + GPU computer 69

Table 4.20 MediaPipe Inference Time 1 1D-CNN + 1 GRU Layers Model in CPU + GPU computer 69

Table 4.21 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU computer 71

Table 4.22 MediaPipe Inference Time 1 1D-CNN + 1 GRU Layers Model in CPU computer 71

Table 4.23 MMPose Framework Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU+GPU computer 72

Table 4.24 MMPose Framework Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU computer 72

# LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| ANN | Artificial Neural Network |
| ASL | American Sign Language |
| CNN | Convolutional Neural Network |
| GRU | Grated Recurrent Unit |
| HPE | Human Pose Estimation |
| LSTM | Long Short-Term Memory |
| NN | Neural Network |
| RNN | Recurrent Neural Network |
| RTMPose | Real Time Multi-Person Pose Estimation |
| SL | Sign Language |
| SLR | Sign Language Recognition |
| YOLO | You Only Look Once |

# INTRODUCTION

* 1. Background of Study

Machine Learning algorithms for Signal Language Recognition have been receiving more interest in recent years because they will provide more opportunities and improve communication in case of success to the deaf and hard of hearing population. Machine learning and deep learning algorithms can be trained on labelled datasets of sign language gestures, allowing them to learn the patterns and variations in hand movements, gestures and signs.

Human Interpreters are used as a way for traditional sign language translation, but this is time consuming, expensive, and there is not always someone available to carry it out. Because of that, finding an accurate and efficient machine learning algorithm has started to be a big point of interest, also it is essential to mention that there are some systems that require to have a device attached to help with the interpretation, however, this is expensive and it is again in the same situation as Human Interpreters, they are not always available.

Developing this system considering only a camera as input, capturing the human joints position prior to the classification is an exciting approach that has been spoken for a few years now, and working on this the aim is to make a meaningful contribution to the data analytics area while helping in the inclusion of people in the deaf, mute, and hard of hearing community. Furthermore, this area of study comprehends a diverse skill set, including computer vision, object recognition, data pre-processing, Machine Learning, and Deep Learning which are explored in this project.

* 1. Objectives
* Determine the impact of different factors on the performance of Sign Language recognition models, such as lighting conditions, camera angles, and variations in hand gestures, and develop techniques to mitigate these factors.
* Implement a machine learning model testing different techniques to recognise and translate American Sign Language.
* Evaluate the usability and effectiveness of the interface in real world settings considering the speed and accuracy of the classification when using commodity hardware.
  1. Scope of Study and Limitations

During the project different areas of data analytics are touched including object segmentation, computer vision, neural networks, and human pose estimation. However, in this project, existing Human Pose Estimation techniques are applied for the human joints coordinates collection to train and feed the developed neural networks, this is an essential consideration as the accuracy of the algorithm is also dependent on the quality of information that it receives.

The available hardware is a laptop composed of an Intel Core i5-10210U CPU with 4 cores @ 1.60GHz, an Nvidia GeForce MX250 GPU (2GB GDDR5 version), and 32 GB DDR4 RAM running Ubuntu 23.04, CUDA 11.8, and cuDNN. With the objective of not sticking with just one keypoint collection technique, different algorithms were run and tried selecting three of them based on accuracy and the hardware ability to run them, these algorithms are below:

* MediaPipe 0.10.2
* RTMPose running with MMPose framework version 1.1.0
* CSPNext + UDP running with MMPose framework version 1.1.0

MediaPipe runs successfully in the CPU, for the two running with the MMPose framework, it is important to mention that the default object detector was changed to YOLOX-Tiny to assist the GPU in running them as the inference time in the CPU was too slow. Other approaches like OpenPose/AlphaPose were also tested, but the inference speed was extremely slow. YOLOV8 unfortunately, its human pose estimation module is just trained to recognise 17 keypoints excluding the hands, because of this, it was attempted to train the model in a custom dataset to recognise the whole body but the time and hardware limitations made this possibility not feasible.

Another restriction is time, which is limited because it is a project with a deadline, preventing a more in depth research, influencing some decisions like the development and testing of more models.

# LITERATURE REVIEW

Millions of deaf and people with difficulties hearing use sign language to communicate between each other and also with other people generating a communication gap with individuals who are not fluent in this complicated expressive language. That is why it is important to research the possibility of developing a sign language recognition system as this has the potential to shorten the communication barrier by automatically translating sign language to spoken or written language which can be done using computer vision techniques capturing and analysing the visual features of sign language, passing it through a machine learning algorithm to classify it and translate the signs.

Sign language recognition systems have started to be a special research subject in recent years, the current state of research is treated in this literature review to identify what can be done in this project differently in order to have the most accurate and efficient model possible taking in consideration that the final goal of this research project is the evaluation and viability of the project in a real time situation.

The chosen topics for this literature review were selected carefully, considering their importance as background knowledge beginning with a summary of Sign Language providing an essential understanding of this unique form of communication to have an overview of it before jumping into the creation of models without a clear direction on how they should work.

Next, the literature review encompasses object detection which is a crucial concept to explore Human Pose Estimation as once the person and their body parts are accurately detected, the model can focus solely on analysing and interpreting those specific elements effectively disregarding the background information present in the image, giving place to explore Deep Learning as is the component behind Human Pose Estimation algorithms and Sign Language classification finalizing with a quick review of challenges that researchers have faced in the past.

* 1. Sign Language

(WHO, 2023) stated that 5% of the global population, or 430 million people, have a certain degree of hearing loss and in 2050 2.5 billion individuals would present this condition. Sign language is a well defined system of movements in the hands indicating meanings which are used regularly by deaf and hard of hearing individuals to interact with others in everyday life (Konstantinidis et al., 2018) that allows them to communicate between each other and with the rest of people in a non-verbal way, often it is challenging to communicate with others as they cannot speak or listen and that is where sign languages come in useful because allow people to communicate without using spoken language (Sharvani Srivastava, 2021b). Sign language is a formal system consisting of hand movements, signs, body posture, facial expression, and eye gaze to express meaning through the head, face, arms, hand, torso, and fingers to represent words. (Cheok et al., 2019). Gestures can be divided into different types including conversational, controlling, manipulative, and communicative gestures (Ying Wu and Huang, 1999) which have their own vocabulary and grammar completely different from spoken and written languages, for transmitting information spoken languages employ the oratory abilities to create sounds that are mapped against certain words and grammatical combinations (Sahoo et al., 2014).

As research since the 1960s and 1970s demonstrated that linguistic processes were not limited merely to the spoken modality, the field of sign language has grown exponentially (Stokoe, 2005), (Klima and Bellugi, 1979), over the course of four decades, studies of sign language have made substantial contributions to our knowledge of language, thought, and social interaction. Sign language is used in different regions of the world, like spoken languages, and several sign languages are used all over the world, like the Japanese, British, Indian, Arabic, and American sign language (Zeshan, 2006), being the last one mentioned the most common visual language used by the deaf community in North America which is the focus of several of these investigations (Hauser et al., 2016).

ASL, as is the case with other natural languages, has certain linguistic features that distinguish it from English (Liddell, 2003), some have speculated that ASL developed more than 200 years ago as a result of the blending of indigenous sign languages with French Sign Language (LSF, or Langue des Signes Française) (Cagle, 2010). Some features of LSF and the original local sign languages have made their way into modern American Sign Language which is a rich, complex, and mature language. The current versions of ASL and LSF are two different languages. Although they share some common indicators, neither set of users understands the other’s signals (NIDCD, 2021), (Hosain et al., 2020) noted that about 6000 hand gestures are used to represent common words in American Sign Language while fingerspelling is used to represent less common words and proper nouns.

American Sign Language is one of the most influential, as a result of their introduction in newly emerging educational systems, and some sign languages have a high percentage of (old) ASL (-influenced) lexicon (Kusters, 2020). Not just in situations where the use of a national sign language was not yet institutionalized ASL was imported, the status of ASL and/or the accessibility of resources in ASL are likely significant factors in these shifts. Teachers in some deaf schools initially started out using a local or national sign language before switching to ASL (Kusters, 2021).

Signs can be written down in textual form using Glosses, glossing means choosing an appropriate English word for signs in order to write them down, although it is not the same as translation, it is conceptually very similar. A gloss on a signed story can be a series of English words written in small capital letters that correspond to the signs in an American Sign Language story (Othman and Jemni, 2012), for example, children who are fluent in American Sign Language and have a need to go through the process of learning to read English can benefit from using ASL gloss, which can be thought as the "elusive" intermediary system (Supalla et al., 2017), this SL gloss annotation form is crucial to help machines understanding and processing the sign language (Johnston, 2008).

* 1. Object detection

(Brownlee, 2019) refers to Object Detection as a set of computer vision tasks related to each other that involves identifying objects in digital images. Basically, tasks can be categorized as two major tasks, image classification, image classification (class prediction of an object in a given image) and object localization (locating one or more objects within a given image).

Object detection models can be classified basically in two categories, two stage and one stage detectors. The two stage detectors locate first regions with objects and then classify them, while the one stage detector feeds the whole image to the algorithm in one single step locating the regions of interest and classifying all the objects simultaneously (Zhao et al., 2019).

This process is used in many sectors like Healthcare, Transportation, Robotics, Fashion, Home, and Tourism (Wang et al., 2022) playing a crucial role in many computer vision tasks for analysing and understanding images or video, its versatility makes it very useful for a wide range of applications, and classification algorithms benefit from object detection as it enables them to concentrate on the relevant information, in Sign Language Recognition for example, once the hands are detected it can ignore the rest of the scene to produce more precise interpretations of the signs being made, improving not only accuracy but also accelerating the training process as the model does not need as much data as many other situations (Sunmok Kim, 2018).

Object detection goes beyond SLR systems, it is critical for human pose estimation because it helps to localise the body joints (Amadi and Agam, 2023) avoiding the problem of dealing with large variations across an image by focusing on joint detection within bounding box regions, save a lot of time and effort by targeting the objects of interest and discarding all the other data (Wang et al., 2020).

Nowadays, one of the most popular object detection algorithms is YOLO which stands for “You Only Look Once” and was first released by (Redmon et al., 2016), since its creation, as it is open source, different people and organizations like Meituan and Ultralytics have been involved in its continuous improvement and development until the latest release of YOLOv8 (Terven and Cordova-Esparza, 2023)

Generally speaking, object detection gives the algorithms contextual information for making decisions based on what they are seeing in a picture and SLR systems get a lot out of object detection because it allows them to more accurately understand and interpret visual data increasing classification accuracy and boosting training efficiency.

* 1. Human Pose Estimation

Human Pose estimation is finding human figures in pictures and videos and figuring out which joints (keypoints) are present (Moryossef et al., 2021). It is essential to allow machines to see and understand people and their interactions (Cao et al., 2021). HPE is widely increasingly used for many applications like sports instruction, limb rehabilitation training, augmented reality, and intelligent security (Meng and Gao, 2021). It can also be used for action recognition, the idea behind this is to extract body joint locations (keypoints) and then use this as input to another algorithm/neural network to select visual features in space and time to perform the action classification (Luvizon et al., 2018).

Deep Learning based on top down and bottom up pose estimation are the two categories into which human estimation pose algorithms are divided (Gui and Luo, 2022).

Traditional top down human pose estimation detects people in an image or video and draws a bounding box around them using object detection, and then a pose estimator extracts body keypoints from the bounding box. Bottom up is the opposite to top down as it draws keypoints on the image and then uses part affinity maps to map it to different people in the image (Gojariya et al., 2021), the most significant benefits of utilising this second method are the lightweight network and the accelerated processing times against top down methods that have computational cost and runtime proportional to the number of people detected (Martinez et al., 2017).

OpenPose was the first real time multiperson system to jointly detect human body, hand, facial, and foot keypoints. Developed by researchers from Carnegie Mellon University and maintained by Yaadhav Raaj and Ginés Hidalgo and was the first known method to do a whole body pose estimation consisting of three different blocks, body-foot detection, hand detection, and face detection which combined output a total of 135 keypoints divided as follows: 25 body keypoints, 70 face keypoints, and 40 hand keypoints . Openpose's CNN architecture is multi-stage and is considered a bottom up model because a CNN analyses the picture, producing a set of feature maps that are sent into the first stage. The first stage generates a collection of PAFs (Part Affinity Fields) by repeatedly concatenating the prediction with the original image features to get improved predictions. The second stage predicts confidence maps using the same iterative approach as the first stage. PAFs assist in part association, while confidence maps assist in part detection. Each stage is composed of numerous convolution blocks that are generated by three 3x3 convolutional kernels simultaneously (Badiola-Bengoa and Mendez-Zorrilla, 2021).

Nowadays, there exist multiple pose estimation techniques, since OpenPose was launched in 2016 lots of other methods have been developed. This literature review will review the actual state of the art. The first one to be reviewed is MediaPipe, a framework developed by Google for creating pipelines that carry out inference over any type of sensory data building a perception pipeline. A perception pipeline processes data from the real world using sensors and algorithms to create digital representations that can be analysed and understood by a computer system, face detection and segmentation, hands detection, pose detection, and any kind of object recognition are examples of perception pipelines (Lugaresi et al., 2019). For tasks like object detection, face detection, hand tracking, and pose estimation, it offers a wide variety of pre-built models which are trained on sizeable, varied datasets that are relevant to the task at hand like the COCO dataset which contains over 200,000 images of people in various poses and activities. They serve as the skeleton of nodes, edges, or landmarks, tracking important points on various body parts and each coordinate point is normalized in three dimensions (Halder and Tayade, 2021).

For this project four MediaPipe models listed below are of particular interest.

* **Hand Landmark Model:** locates 21 hand-knuckle coordinates as keypoints within the identified hand regions. About 30K real-world images and several rendered synthetic hand models imposed over different backgrounds served as the model’s training data. A palm detection model and a hand landmarks detection model are included in the hand landmarker model package. First, the palm identification model locates hands inside the input image, and then the hand landmarks recognition model recognises specific hand landmarks on the palm detection model's cropped hand image.
* **Face Mesh Model:** Similar to the hand landmark model it detects face landmarks and facial expressions in selfie-like images and videos producing a total of 468 3D keypoints. It performs the task using two deep neural network models, a face location detector that operates on the entire image and a 3D face landmark model that operates on those locations and uses regression to predict the 3D surface.
* **Pose Landmark Model:** locates human body landmarks in images using machine learning, and they can process either a single image or an ongoing stream of images. The model produces 33 body pose landmarks in image coordinates and three-dimensional (x, y, z) world coordinates.
* **Holistic Landmark Model:** It is no more than the combination of the three models described above allowing the analysis of full body gestures, poses, and actions, giving a total output of 543 landmarks (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand) in real time.

More MediaPipe documentation is available at the official Google’s Developer MediaPipe site (Google, 2023).

Another method of particular interest is Real Time Multi-Person Pose Estimation (RTMPose), which is one of the most recent developments in human pose estimation released in 2023. It is integrated into the MMPose opensource toolkit, which allows for an easy deployment of the model. The model is trained on the COCO-Wholebody dataset giving the model the ability to recognise 133 dense landmarks with 68 on the face, 42 on the hands, and 23 on the body and feet (Jin et al., 2020). RTMPose uses a top down approach, which tackles the usual slow inference problem associated with this technique by working in conjunction with high speed object detection models, using CSPNext as the backbone for the model, and executing human detection every certain number of frames instead of executing the detection in every single frame, in general, RTMPose has demonstrated a good balance of speed and accuracy achieving better results in terms of accuracy/inference speed balance for whole-body pose estimation than other algorithms like OpenPose (Jiang et al., 2023).

* 1. Deep Neural Networks

Artificial Neural Networks (ANNs) often known as Neural Networks are innovative systems and computational approaches for machine learning, knowledge demonstration, and eventually the application of acquired information to maximize the output responses of complex systems (Chen et al., 2019). Artificial neural networks are built in the same way as the human brain which has billions of neuron nodes connected between each other, each neuron has a cell body that processes information by transporting it to and from the brain (van Gerven and Bohte, 2017).

The system comprises many highly linked processing components known as neurons (Walczak and Cerpa, 2003), which work together to solve problems and communicate information via synapses (electromagnetic connections in the human brain). The neurons are interconnected and organized into layers. The data is received by the input layer, and the final outcome is generated by the output layer with one or more secret layers placed between the two of them (Dastres and Soori, 2021).

Deep Neural Networks are Neural Networks that including the input and output layers have more than three layers of neurons. These layered representations are using models referred to as "Neural Networks," which are arranged into literal layers and stacked one on top of the other (Schmidhuber, 2015).

Different approaches had been used in previous works to do the SL classification, two of which are Convolutional Neural Networks and Recurrent Neural Networks.

The CNN model is an important Neural Network component used widely, but not exclusively, for image recognition and classification during sign and face detection or recognition (Pathak and Maheshwari, 2019). CNN models are composed of neurons with weights and biases. Specific neurons receive input data in response to actions, and weighted sums take over, activating certain functions and producing certain outputs. The CNN models are frequently used in multi-channel images (Dhulipala, 2022). CNNs as ANNs are inspired by human nature, CNNs emerged from the study of human’s visual cortex and thanks to the advancements in computation and data availability in the last years, these Deep Neural Networks have achieved an amazing performance on image recognition tasks, the most basic architectures stack a few Convolutional Layers, Pooling Layers, and so on making the image smaller and smaller between layers, finally passing through a common neural network which layers are the fully connected layers, these ones are the ones in charge of generating the output (Géron, 2019), the convolutional layer determines the output of neurons connected to important local regions, the pooling layer reduces the number of parameters by performing down sampling along spatial dimensionality, and the fully connected layers perform the same tasks as any standard Artificial Neural Network, producing class scores from the activations to be used for classification (O’Shea and Nash, 2015).

The most popular CNNs are 2D-CNNs used principally for images, but there are other CNNs as well such as 1D-CNN. 2D-CNNs were designed to operate exclusively in 2D data and therefore they work well with images being the input of a 2D-CNN (width, height, depth), and with its success it was tried on other data types for engineering applications composed only of one dimension, however, its application to 1D data was not as good as other traditional machine learning algorithms because the data was necessarily needed to be transformed to 2D data which is not a straightforward process always depending on the application, this problem was present for 1D data until 1D-CNN appearance showing a lower computational complexity allowing 1D-CNNs to be trained in standard computer configurations with no need of special hardware. As 2D-CNNs are mostly used for computer vision/images, 1D-CNNs are mostly used for sequential or time series data. In 2D-CNNs, the kernel travels two dimensions which are the width and height of the image, and the third dimension, the depth, is always the same as the number of channels of the input image. In 1D-CNNs, the kernel slides along one dimension only, the kernel width which is adjusted can be customized, while the height of the kernel is fixed to the number of points per timestep (Kiranyaz et al., 2021).

Long short-term memory (LSTM) is a type of RNN architecture that retains values at random intervals. They are employed in the classification, processing, and prediction of time series with known time lags but unknown durations. The LSTM is referred to as the cell state, and its recursive nature is represented by a looping arrow. As a result, the data from the previous interval is saved in the cell state. A remember vector located beneath it adjusts the cell state, while the input modification gates adjust it. The gates additionally instruct the network on how to save, forget, remember, focus on, and output data. The cell and hidden states are used to collect data for processing in the following state, which attempts to solve the gradient vanishing problem (Yamak and P. K. G, 2019) allowing complex and artificial long time lag tasks to be solved (Hochreiter and Schmidhuber, 1997), additional to the LSTM units, there is another approach introduced a couple of years ago by (Cho et al., 2014) which name is Gated Recurrent Unit (GRU) consisting of an Update Gate and a Reset Gate, being the Update gate an assistant to the model in determining how much past knowledge (from earlier time steps) should be passed on to the future, and the Reset Gate helps the model to decide how much past information has to be forgotten (Chung et al., 2014), LSTM and GRU can also be combined to achieve higher accuracies and this has been demonstrated by (Kothadiya et al., 2022) where six different architectures were tested LSTM, GRU, LSTM-LSTM, GRU-GRU, LSTM-GRU, and GRU-LSTM being LSTM-GRU the one achieving the highest accuracy predicting Indian Sign Language video frames over 11 signs.

* 1. Activation Functions

Activation functions, also known as transfer functions, are mathematical functions that determine whether a neuron is activated or not adding the ANNs to model nonlinear problems (Wiley, 2016), even when there is no evidence of having something similar in the human brain, it is the way to activate artificial neurons as biological neurons are activated (Montesinos López et al., 2022). There exist a lot of different activation functions that have been developed being the sigmoid function the first one to be used, and has been replaced over the years. As there are plenty of activation functions nowadays, the ones discussed in this literature review are some of the most popular and that will be implemented in the neural network design process of this project.

A lot of different activation functions have been developed being the sigmoid function the first one to be used, and has been replaced over the years. As there are plenty of activation functions nowadays, the ones discussed in this literature review are some of the most popular and will be implemented in this project’s neural network design process.

**SoftMax**: The SoftMax activation function is used for multi-label classification usually in the output layer of the network returning the probability distribution over output classes summing between all classes 1, a strong prediction will be a vector close to 1, leaving the rest close to 0, a weak prediction will have multiple possible categories with similar probabilities between each other (Montesinos López et al., 2022).

**Tanh**: Tanh goes from -1 to 1, having the advantage against the sigmoid function to deal better with negative values (Patterson and Gibson, 2017) and is represented the same as the trigonometric function. Tanh is extensively used nowadays in RNNs such as LSTMs/GRUs as gates. Tanh mathematical function is represented as:

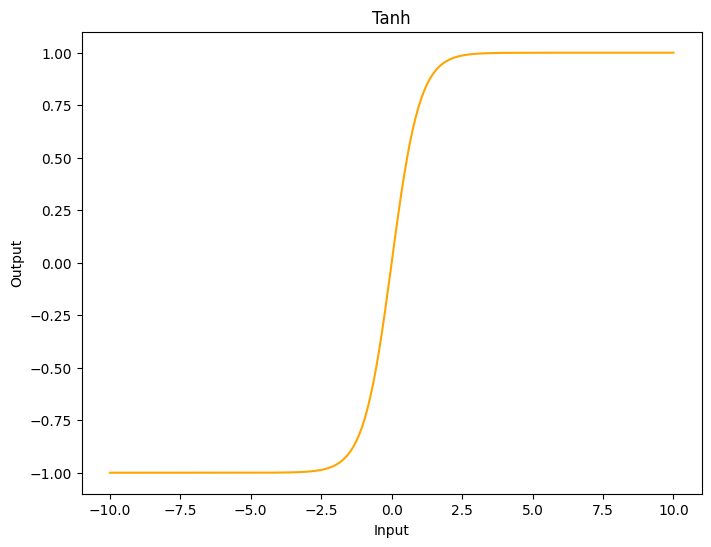


Figure 2.1 Tanh Activation Function Representation

**Rectified Linear Unit:** commonly known as ReLU, it is the go to activation function nowadays being one of the keys to Deep Learning success in recent years as they work in different situations, ReLU only activates if the input is above a certain number. When the input is less than zero, the output is zero, however, when the input is above a particular threshold, it has a linear relationship with the dependent variable and it is represented as (Patterson and Gibson, 2017):

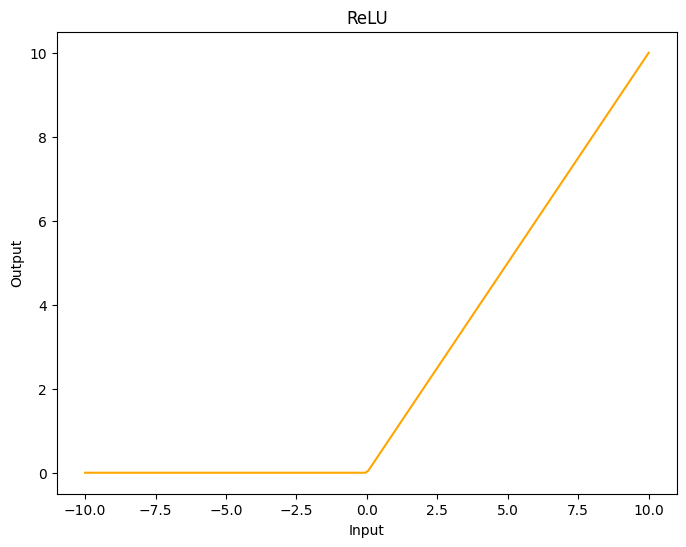


Figure 2.2 ReLU Activation Function Representation

Even when ReLU is the state of the art activation function and has been demonstrated to train better than sigmoid/tanh activation functions. It has its downsides, the most important one is because the gradient of a ReLU is either zero or a constant, it is not easy to control the vanishing exploding gradient issue and this is known as “dying ReLU” (Montesinos López et al., 2022) which has been tried to be solved with several ReLU variants, some of them will be touched in this project.

**Leaky ReLU:** LeakyReLU is a ReLU variant that attempts to solve the dying ReLU problem by producing none zero output for negative input with a nonzero slope for negative values predefined between 0 and 1 enabling part of the negative feature information to be retained (Zhang et al., 2017), Leaky ReLU is represented as:

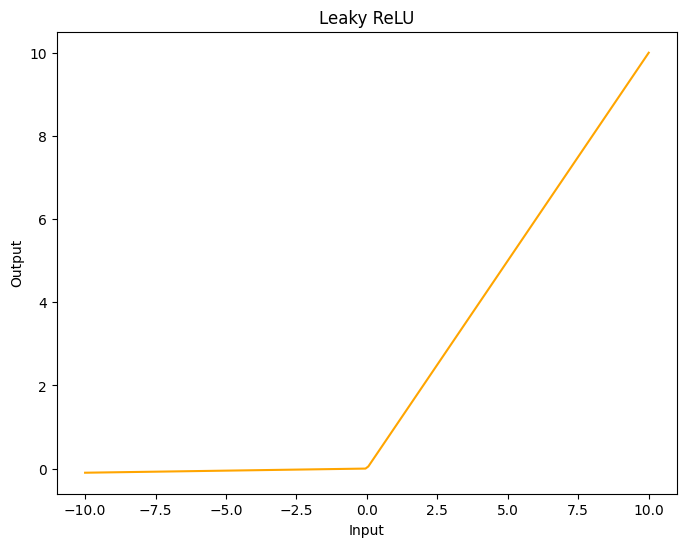


Figure 2.3 Leaky ReLU Activation Function Representation

**Exponential Linear Units:** ELUs, in difference to ReLU have negative values which push the mean closer to zero enabling a faster learning. It is proven to produce better results than ReLU as the nonzero coefficients applied for negative values also help the network to mitigate the “dying ReLU” problem. The only problem with ELU is that it requires more computation resources as it uses an exponential function making it slower than ReLU which is compensated when training by the faster learning, but in real time, it will always be slower than ReLU and other of its variants(Clevert et al., 2015). ELU is represented as

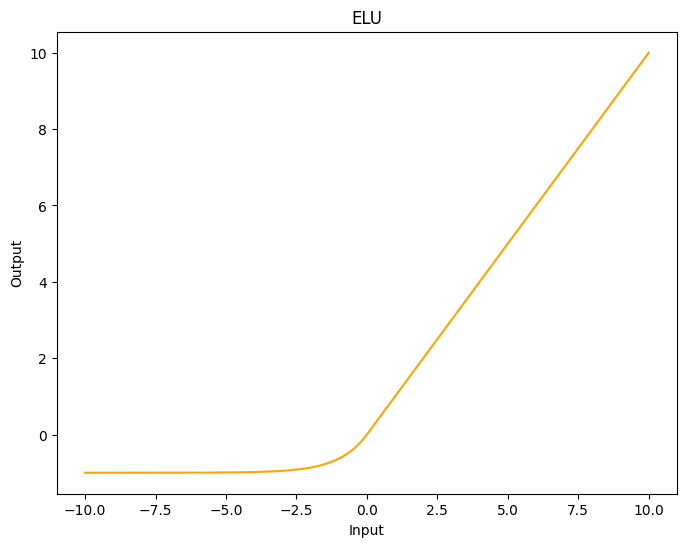


Figure 2.4 ELU Activation Function Representation

**Gaussian Error Linear Units:** GELU is designed based on the idea of combining regularization with RELU, it does an entirely different job to weight its inputs in comparison to RELU instead of weighting the input based on its sign, it weights it based on the value, this is achieved by multiplying the input with the cumulative distribution function of the normal distribution at this input. As GELU is a computationally intensive function because of the CDF calculation, there is an approximation alternative that has only a slight variation but is faster (Hendrycks and Gimpel, 2023). The two representations of GELU are:

Or its approximation

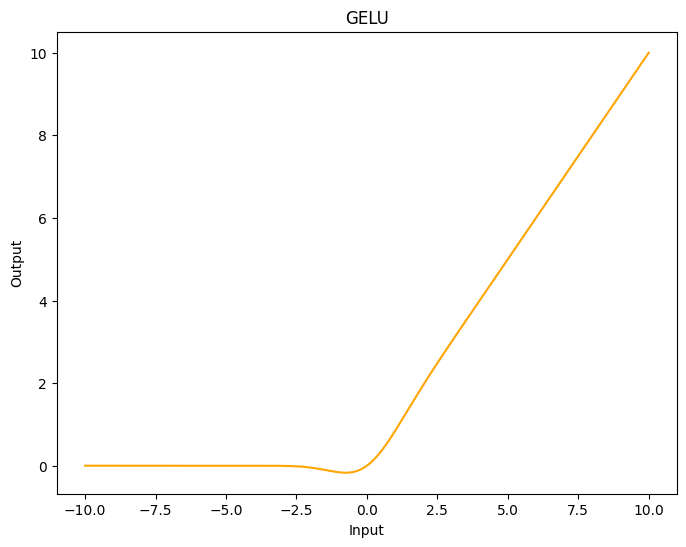


Figure 2.5 GELU Activation Function Representation

**Scaled Exponential Linear Units:** SELU is a variant of ELU attempting to bring fully connected neural networks again to the game, bringing the concept of Self-Normalized Neural Networks (SNNs). SNNs are only formed with dense layers using the SELU activation function, causing the network to self-normalize, which means that during the training, the output of each layer will have a mean of 0 and a standard deviation of 1 addressing in this way the vanishing/exploding gradients problem. However, in order to this happen, certain conditions need to be met, and such conditions are listed below:

* The input features must be standardized (mean 0 and standard deviation 1).
* Weights for every hidden layer have to be initialized with LeCun normal initialization.
* The network’s architecture must be sequential.
* SELU activation with parameters λ ≈ 1.0507 and α ≈ 1.6733.
* Regularization with alpha-dropout, which is α and is the dropout technique also proposed with SNNs that randomly sets inputs to α value.

If these conditions are met, self-normalization will occur and is most likely that SELU will outperform other activation functions, if these conditions are not met, self-normalization is not guaranteed, which means it will not necessarily outperform other activation functions (Klambauer et al., 2017). Even when the problem in this project is not being addressed with fully connected neural networks, there are studies showing that SELU has improved the performance on CNN-LSTM networks successfully (Phankokkruad and Wacharawichanant, 2019) (Huang et al., 2020).

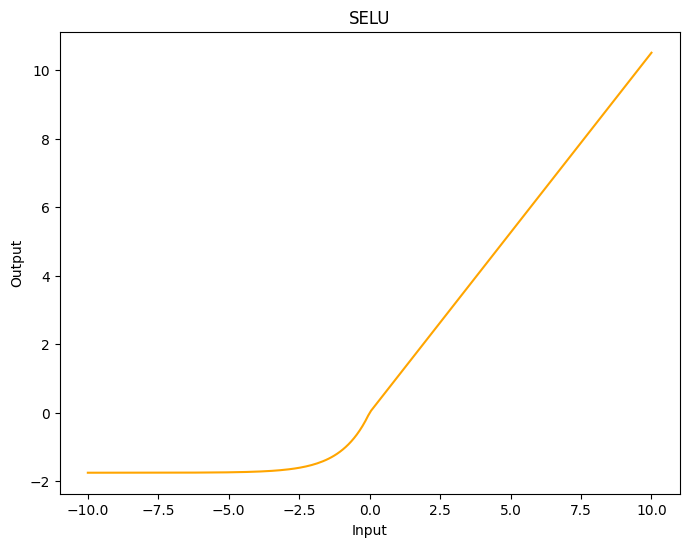


Figure 2.6 SELU Activation Function Representation

**Mish:** Mish is an activation function inspired by the activation function ‘Swish’ (Ramachandran et al., 2018) which was a function discovered using a novel automatic search technique and was tested in experiments designed for ReLU, just replacing ReLU with Swish increased the performance of the models, and Mish was found while studying the characteristics of Swish that makes it more effective than others. Mish trials were done in computer vision tasks with various standard architectures comparing its results against LeakyReLU, ReLU, and Swish which were just outperformed by Swish in 2 tests. Mish is a smooth, continuous, self-regularised, non-monotonic activation function mathematically represented as (Misra, 2020):

Or

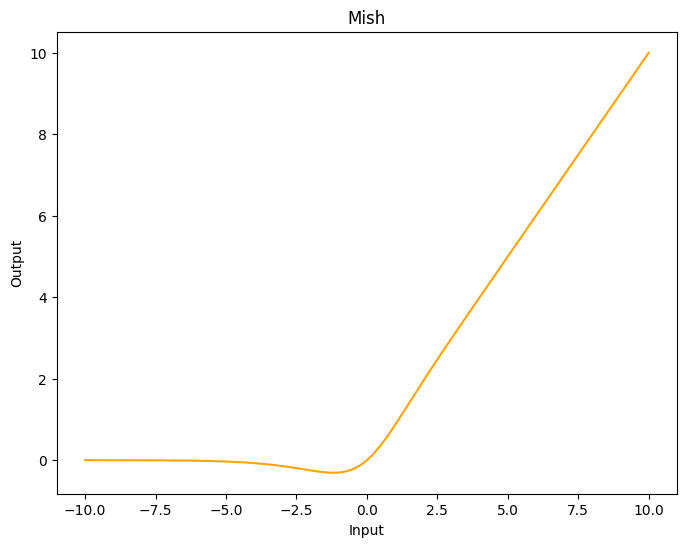


Figure 2.7 Mish Activation Function Representation

* 1. Sign Language Recognition Systems

Recent years have seen remarkable development in AI technologies related to Natural Spoken Languages however, sign languages have not seen similar progress especially in terms of word and sentence recognition. Even though sign language recognition technology has seen some progress over the years, particularly impulsed by Deep Learning, more advanced recognition models have emerged but are still not at the same level as spoken languages (Selvaraj et al., 2022).

Numerous authors have created effective methods for data collection and classification which can be divided into two categories based on the data acquisition method (Sarma and Bhuyan, 2021), the first ones to be reviewed are Direct measurement methods, these methods use devices such as motion data gloves, motion capturing systems, or sensors (Kudrinko et al., 2021) that can convert the signal, in this case Sign Language, into an electric or audible signal acting as a bridge for communication. However, there are some issues with sensor data acquisition like noise, poor human manipulation, and a faulty ground connection, these issues without mentioning that it makes the system more expensive because it requires specialized devices and training to use them adequately (Elakkiya, 2020). The extracted motion data can be used to track fingers, hands, and other body parts accurately, leading to the development of robust SLR methodologies (Sharvani Srivastava, 2021a), one of the most recognised devices of this kind is the Microsoft Kinect sensor, which captures an RGB image and a depth map thanks to its infrared led, infrared sensor and RGB camera integration in one single device (Zhang, 2012). One example of success using the Kinect for American Sign Language recognition was presented by (Cao Dong et al., 2015) where a Random Forest Classifier is fed with the joint angles to classify 24 static signs reaching a 90% of accuracy.

The second methods to be reviewed are the vision based SLR systems, which can work with a device as simple as a laptop webcam or phone camera getting RGB images to extract spatial and temporal information without the need to attach sensors to humans physically. These vision based systems have recently gained popularity despite being prone to reliability issues in the past due to background noise, colours, and lighting variations in the real world (Kin Yun Lum, 2020). Nevertheless, with the latest developments in vision and machine learning, image classification has considerably improved making the vision based Signal Language Recognition algorithms improve as well. SLR is not a new problem in computer vision as researchers have used diverse kinds of classifier algorithms over the last two decades, which can be roughly divided into linear classifiers, neural networks, and Bayesian networks (Anagha.G, 2022).

(De Coster et al., 2023) did recently an interesting article analysing state of the art in Sign Language Recognition systems over the last few years, the below table shows a selection of 57 research papers dedicated to SLR and their evolution. From 2004 to 2018, all of them were using Statistical Machine Translation, but since 2019 all of them have started to use Neural Machine Translation, except for one article (Luqman and Mahmoud, 2020), where they used Rule based Machine Translation.

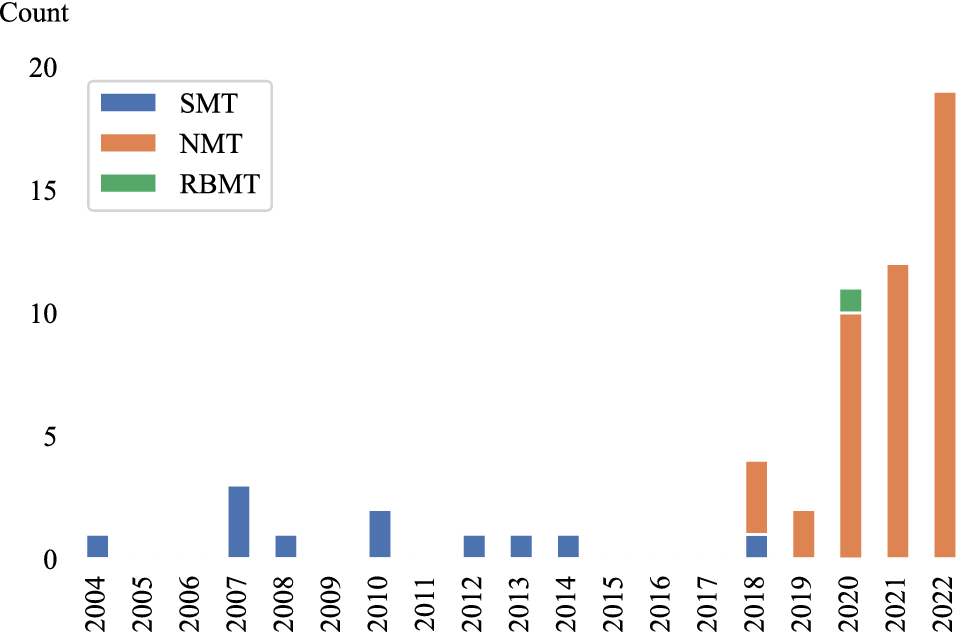


Figure 2.8 Machine Translation methods distribution between 2004 and 2022. Taken from (De Coster et al., 2023)

Another point to highlight is how the research for sign language has increased since 2018 from only 10 before since 2004, to 47 in the last years, corresponding with Neural Machine Translation systems trends.

Since OpenPose launched in 2016 different algorithms for pose estimation have been widely used to solve action recognition problems due to its simplicity and high accuracy. OpenPose and similar frameworks feed the vision based systems with extra information about coordinates that the direct measurement systems have but with any camera, converting this technique into a fundamental tool to study human behaviour, particularly in recognizing actions. When there are a lot of occlusions or severe deformations, using the body features can help improve recognition accuracy (Rastgoo et al., 2021), this technique of human keypoints extraction allowed (Ko et al., 2018) to develop a novel Korean Sign Language recognition system that reached 89.5% accuracy feeding the keypoints to a neural network in 100 complete sentences that can be used in emergencies, another approach using keypoint extraction also was done recently by (Kim and Baek, 2023) where its model executes four principal tasks: extraction of keypoints, keypoint normalization, and skip sampling (SASS) to achieve the Sign Language Translation, this was done removing the lower body keypoints and face keypoints to end with a 55 keypoints extract which is after normalised before feeding the algorithm. after creating a system that detects finger spelling using MediaPipe Hand solution to extract hand only joints and OpenPose for a system able to recognise complete words extracting the full body keypoints concluded as future work, it might be interesting to try a combination between OpenPose and MediaPipe to improve the results being OpenPose the responsible for detecting the body position and MediaPipe Hands module to extract the hand gestures keypoints.

* 1. Challenges

Sign Language Recognition systems face the same challenges as every computer vision problem, the environment light, change in views and movement speed are factors that may influence the prediction accuracy as they cause the gesture to seem different in 2D space (Cheok et al., 2019). High computational cost is also an important point to mention because even when Graphic Process Units in the last years have tried to address this problem (Thompson et al., 2020), achieving high speed video real time processing while maintaining accuracy is a complex challenge which involves optimising algorithms, hardware acceleration, and efficient memory usage.

(Subramanian et al., 2022) stated that finding a prototype that acquires the sign gesture and its corresponding text is the primary difficulty in creating a sign language recognition system because even when different techniques are available, the challenges of hand tracking, occlusion of hand movements, high computational cost, feature selection and lower learning efficiency still exist.

(Singh, 2022) noted in three points important facts that make sign language recognition systems challenging to be deployed nowadays, these points are “Limited number of datasets available”, “Domain restricted data” and “Lack of variety in datasets”, so data is a big challenge in this task, the author points that the few datasets available do not have a wide variety of signers (10-20 average), lots of them are collected in the same environment, and have limited vocabulary making the model not able to generalise in unseen data after training is done.

Also, (Attar et al., 2022) highlighted the lack of standardization and availability of proper linguistic and grammar rules as an important obstacle to developing effective sign language translation systems, forcing researchers to use data driven translation approaches in Sign Language Translation systems.

* 1. Conclusion

Two kinds of Neural Networks with the potential to be applied to this task after extracting the pose keypoints were identified, being CNNs and RNNs the selected architectures to work in this project thanks to the ability of these networks to learn and extract features from sequential data.

Object identification and segmentation techniques are essential for sign language recognition systems, these strategies aid in the identification and location of relevant items in the video stream, such as hands and body parts and further, estimating the signer pose is another component of sign language recognition which consists on finding and keeping track of the positions of key body landmarks within images or videos of the subject in question. Frameworks such as MediaPipe, OpenPose, and YOLOv8 are examples of some of the most popular options for pose estimation, however there are plenty of other methods to extract keypoints nowadays.

Direct measurement methods and vision based approaches are the two primary classifications that can be applied to get information about sign language recognition systems. When it comes to capturing motion data, direct measurement methods involve the utilization of specialized hardware in the form of data gloves or motion capture systems while vision based approaches rely on computer vision techniques and use RGB images obtained from cameras, this literature review has helped identify that the project will be done in a vision based method as it is more suitable in a real time situation due to non-additional equipment being required other than a camera and a device to run the algorithm.

# METHODOLOGY

* 1. Ethical Considerations

It is essential nowadays to consider ethics in every Artificial Intelligence project that will be developed as this is evolving rapidly and changing our lifestyles in positive and negative ways. (Green, 2018) made an important statement to take in consideration when researching this science area, “AI, as the externalization of human intelligence, offers us in amplified form everything that humanity already is, both good and evil”, also one of the concerns the author points as the most important when dealing in this kind of application is privacy, and privacy when a vulnerable population is involved has to be more critically evaluated and taken in consideration to develop these kind of applications.

Another concern is that although machine translation systems have significantly improved in recent years they are sometimes not trustworthy enough for use in areas where lexical and conceptual precision are critical, such as in sectors dealing with cultural expression and literature or the medical field. AI-based Machine Translation is anticipated to be developed principally for the primary world languages, especially English because these languages have access to the massive datasets necessary for the technology's success. This is harmful to the preservation of different languages, because of this it is always important to note that AI interpretations, at this stage, cannot substitute human interpreters and should be used with caution and in non-critical situations.

Having these challenges in consideration, even when the privacy concern can be easily addressed in this project as the final data extracted and fed to the algorithm are the joint positions in a NumPy array, the decision to use available data on the internet was taken to completely avoid falling in any kind of ethical problem from this side.

Regarding the other point touched, it is crucial to keep in mind that the project developed in this piece of work is a theoretical exercise, pointing to expand the field for further research and evaluation prior to its implementation to a real life situation, in this project no deaf or hard of hearing people was involved in its development, which is essential to involve, if real life implementation is the main point for a project of this nature.

* 1. Sampling and Primary Research Method

The populations of interest in this project are potential users of sign language translation and direct American Sign Language users which means people who interacts with the mute, deaf and hard hearing community, who sometimes suffer a gap of communication because there is not many people out of the deaf community that knows how to interpret sign language and an interpreter is not always available, as previously identified in this chapter, collecting data directly from the deaf and hard of hearing community raises ethical considerations and requires special attention to ensure the protection of their rights and privacy, given the vulnerable nature of this population using an existing dataset publicly available on the internet to train and validate the different models fed by different HPE algorithms rather than getting data collected directly from individuals within the community was preferred.

Each combination of the identified data acquisition technique (MediaPipe, RTMPose, CSPNext+UDP) and Neural Network architecture (CNN, CNN-GRU, CNN-LSTM) was implemented and tested for the experimentation phase. Stratified sampling was the method used with groups identified having 10 different combinations of network architectures and three different Human Pose Estimation algorithms for a total of 30 different subgroups, which are then optimized with Keras tuner and Hyperband algorithm. The Hyperband tuning algorithm combines random sampling at the beginning with successive halving to efficiently explore the hyperparameter space for each neural network combination assigning more resources (time) to try the most promising combinations (Li et al., 2018).

When the experimentation process was executed, the Training F1 Score, Training Loss, Validation F1 Score, and Validation Loss were collected for the ten best models of each combination to have a better overview and make decisions based on this data with the goal of identifying the optimal combination that yields the highest accuracy applying the principle of Concomitant Variation.

In each step of the experimentation, temporal sequence was present, the temporal sequence in this sign language recognition system will be related to the accuracy, as it will change accordingly while different techniques of human pose extraction and architectures are tested and tuned. The theoretical support provided by the literature review was the guidance while the experimentation process was conducted.

* 1. Dataset Description

The data that has been used along the project to train the different models and testing is a portion of the just released “ASL Citizen” by (Desai et al., 2023) , it is a conjunction work of Microsoft Research, University of Washington, Boston University, and University of Maryland in an effort to support machine learning models for Sign Language Recognition, it is composed of 83,399 videos for 2,731 different signs performed by 52 signers, 35 signers for training, 6 for validation, 11 for testing. It is important to mention that these videos were collected by people in everyday settings recording themselves with a webcam available, this is a great advantage compared to other previous datasets where the videos are recorded by professionals in laboratories or classrooms with particular characteristics as gives the dataset a rich variety of conditions that allows the algorithm training to be more robust, a paid seed signer recorded the videos to follow, and the contributors submitted their own version of the sign.

The videos in the dataset were already cleaned by the research team removing the ones where the YOLOV3 algorithm did not detect a person, and also blurred the background in videos where additional people was also detected to protect their privacy. Also, filters were applied to remove blank videos, videos not containing people, and videos without signing as described, however, Microsoft highlights in the dataset repository that these filters were basic and may not have caught all of these situations giving place to some videos with different regional variations or completely erroneous videos for a sign.

The given dataset includes 3 different CSV files, each containing the videos for training, testing, and validation, the features contained are as follows:

* **Participant ID:** Signer identifier.
* **Video File:** Name of the file.
* **Gloss:** The actual meaning of the file based on a previously released dataset ASL-LEX (Caselli et al., 2017).
* **ASL-LEX Code:** Gloss identifier in the ASL-LEX dataset.

Please refer to the creator dataset repository [ASL Citizen](https://www.microsoft.com/en-us/research/project/asl-citizen/) for more information about the dataset.



Figure 3.1 Microsoft ASL Citizen Sample, image taken from the official Microsoft ASL Citizen repository

* 1. Exploratory Data Analysis

Taking in consideration the amount of time and resources available for this project, only fifty words of the complete ASL Citizen catalogue are considered, and the decision on words to be used was based on the fifty Glosses with more samples on the training dataset which can be found in Table 3.1.

| **Gloss** | |
| --- | --- |
| AXE1 | GUESS1 |
| BACKPACK1 | HALLOWEEN1 |
| BASKETBALL1 | HOSPITAL1 |
| BEE1 | HURDLE/TRIP1 |
| BELT1 | JEWELRY |
| BITE1 | KNIGHT1 |
| BREAKFAST1 | LOCK1 |
| CANCER1 | LUNCH1 |
| CHRISTMAS1 | MAPLE |
| CONFUSED1 | MEAT1 |
| DARK1 | MECHANIC1 |
| DEAF1 | MICROSCOPE1 |
| DECIDE1 | MOVIE1 |
| DEMAND1 | NOON1 |
| DEVELOP1 | PARTY1 |
| DINNER1 | PATIENT2 |
| DOG1 | RAZOR2 |
| DOWNSIZE1 | RIVER1 |
| DRAG1 | ROCKINGCHAIR1 |
| EAT1 | SHAVE1 |
| EDIT1 | SINK |
| ELEVATOR1 | SQUEEZE |
| FINE1 | THEY1 |
| FLOAT1 | TWINS1 |
| FOREIGNER1 | WHATFOR1 |

Table 3.1 Top 50 Glosses with more videos in training ASL Citizen Dataset

Once this filter was applied, the frequency distribution on the training dataset ended with 950 videos in total for 50 words, having a minimum of 18 videos and a maximum of 24 videos per word, falling 75% of them in the range of 18–20 samples.

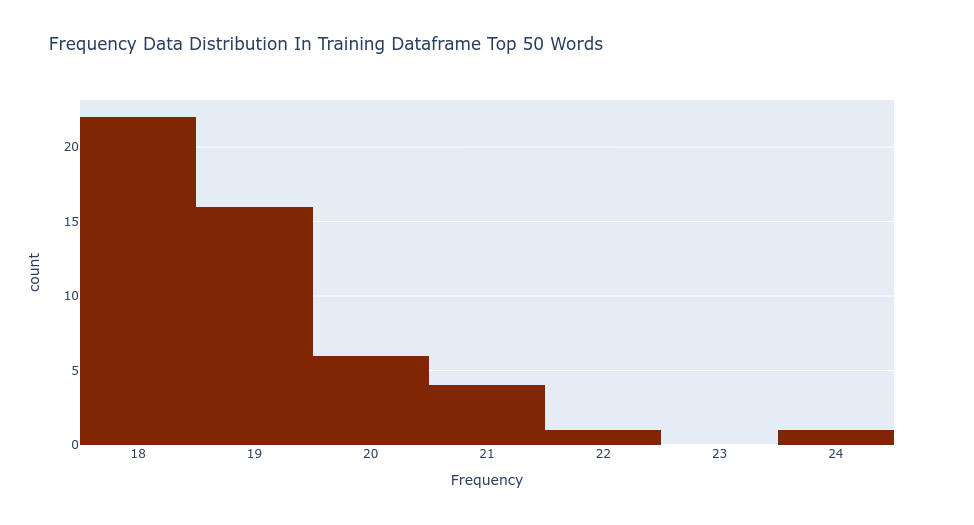


Figure 3.2 Frequency Data Distribution in Training Dataframe Top 50 Words

These same Glosses were collected from the validation and test dataset ending with a total of 191 and 729 videos respectively. Considering that the end goal of this research is to train a neural network and the nature of the human pose estimation techniques used, some extra pieces of information not contained in the dataframes were gathered and analysed using the OpenCV library, such information becomes more important while the process of this work progresses, below is the list of columns added to the dataframes including a short description:

* **Frames:** Total number of the individual images that compose the video.
* **FPS:** Rate at which consecutive frames were recorded.
* **Length:** Total duration in seconds.
* **Width:** Number of pixels from the left edge to the right edge of the frame.
* **Height:** Number of pixels from the top edge to the bottom edge of the frame.

After this information was obtained, the below statistics describe the combination of training and testing dataframes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Frequency** | **Frames** | **FPS** | **Length** | **Width** | **Height** |
| count | 1679 | 1679 | 1679 | 1679 | 1679 | 1679 |
| mean | 33.840977 | 82.300774 | 29.069089 | 2.846337 | 660.011912 | 483.752233 |
| std | 2.876121 | 40.07867 | 2.653846 | 1.365843 | 77.504276 | 14.532052 |
| min | 27 | 10 | 11 | 0 | 640 | 480 |
| 25% | 33 | 59 | 29 | 2 | 640 | 480 |
| 50% | 34 | 75 | 30 | 3 | 640 | 480 |
| 75% | 35 | 94 | 30 | 3 | 640 | 480 |
| max | 41 | 540 | 31 | 18 | 960 | 540 |

Table 3.2 Descriptive Statistics of Training and Testing Datasets combined

From Table 3.2, it can be seen how the videos were captured with different devices varying the FPS rate, which was expected from the start as Microsoft described the dataset was collected individually by the signers themselves without involving any kind of support other than the seed signer who indicated how the signs should be performed, which is helpful in general for the training. However, a few more steps were required to be performed prior to starting collecting the keypoints data as the neural network input layer needs to have a specific input shape and there are videos with different lengths, meaning different quantities of frames.

The idea behind the keypoint collection is to have an array per video containing the keypoints per frame obtained from the Human Pose Estimation algorithm, because of this all the videos were filtered considering the ones lasting more than 1 second, with an FPS rate equal/greater than 29 and containing maximum 130 frames, this decision was made with the idea that the final arrays will be of 130 x number of keypoints, meaning that all the videos lasting less than 130 i.e. 120, the missing 10 keypoint arrays will be padded with zeroes. 130 is the number chosen as this is the exact point between the third quantile and the upper fence, ending with videos lasting more than a second and a maximum of approximately 4 seconds which is enough to capture the movement sequences to perform any sign, there are videos like “4256498922014438-SHAVE.mp4” lasting more than that where the signer performs a diversity of signs for the same action which will provoke the input layer having to be larger, arrays for training bigger on memory containing zeroes only and confusing the algorithm.

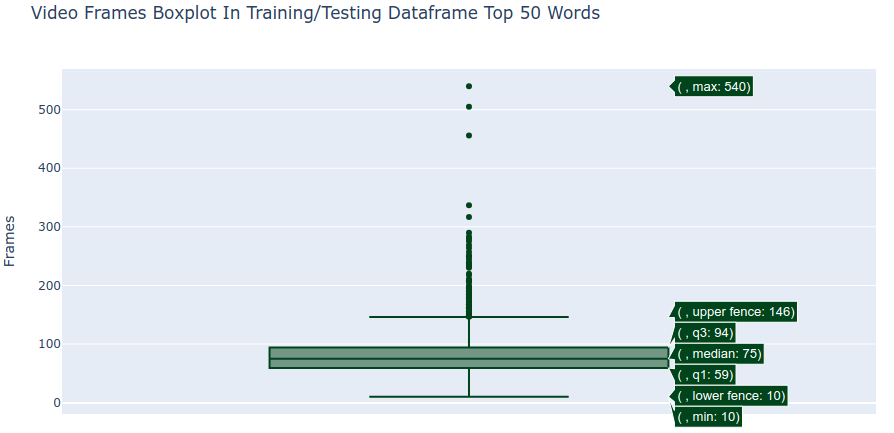


Figure 3.3 Frames per Video in Training and Testing Dataframes Top 50 Words

After the filtering had been applied, for training there were Glosses with just 13 videos per Gloss which was considered too low to train the network, because of this, the decision to join the training and testing datasets was taken ending with 1,338 videos, having a frequency distribution from 20 to 32 samples per Gloss having a minimum of 45 frames and a maximum of 130 frames per observation, the validation dataset was used applying the same rules ending with 161 videos having 2 to 5 samples per Gloss.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Frequency** | **Frames** | **FPS** | **Length** | **Width** | **Height** |
| count | 1338 | 1338 | 1338 | 1338 | 1338 | 1338 |
| mean | 27.203288 | 76.569507 | 29.689088 | 2.603886 | 660.328849 | 483.811659 |
| std | 3.306202 | 19.827114 | 0.49881 | 0.671049 | 78.080254 | 14.640048 |
| min | 20 | 45 | 29 | 2 | 640 | 480 |
| 25% | 25 | 60 | 29 | 2 | 640 | 480 |
| 50% | 28 | 75 | 30 | 2 | 640 | 480 |
| 75% | 30 | 89 | 30 | 3 | 640 | 480 |
| max | 32 | 130 | 31 | 4 | 960 | 540 |

Table 3.3 Descriptive Statistics of Training and Testing Datasets combined after Filtering

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Frequency** | **Frames** | **FPS** | **Length** | **Width** | **Height** |
| count | 162 | 162 | 162 | 162 | 162 | 162 |
| mean | 3.506173 | 78.203704 | 29.691358 | 2.623457 | 640 | 480 |
| std | 0.907162 | 23.787397 | 0.463365 | 0.772311 | 0 | 0 |
| min | 2 | 45 | 29 | 2 | 640 | 480 |
| 25% | 3 | 59.25 | 29 | 2 | 640 | 480 |
| 50% | 4 | 71.5 | 30 | 2 | 640 | 480 |
| 75% | 4 | 92 | 30 | 3 | 640 | 480 |
| max | 5 | 130 | 30 | 4 | 640 | 480 |

Table 3.4 Descriptive Statistics of Validation Dataset after Filtering

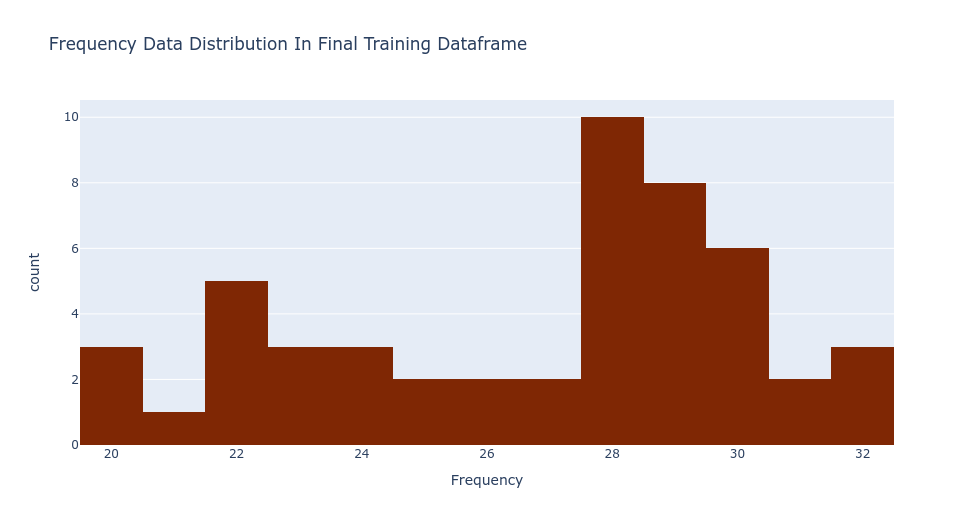


Figure 3.4 Frequency Data Distribution in Final Training Dataframe

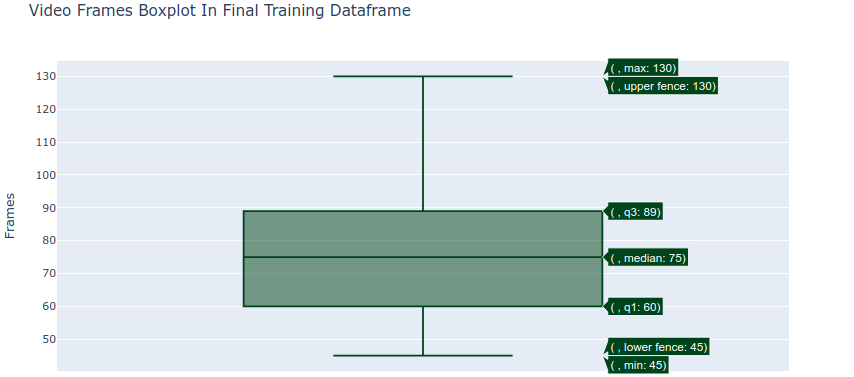


Figure 3.5 Frames per Video in Final Training Dataframe

* 1. Keypoint Collection

For this section, two main frameworks were used, MediaPipe and MMPose. MediaPipe, developed by Google is an open-source framework that allows to quickly implement tasks like object detection, face detection, hand tracking, and pose estimation with pre-built models being the Holistic Model the one used on this project, the hands detection model was also tested, however it was discarded as the model at the moment this project was developed, the MediaPipe hands instead of having a property identifying the left hand and right hand separately as in the Holistic model, it just creates an array with the keypoints visualized in the image, even though the right/left hands are identified, these keypoints are not always arranged in the same way pointing to additional challenges and noise for the algorithm to learn.

This being said, the Holistic model outputs 543 landmarks in total, 33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand. In the project only the 42 hand keypoints are considered, as using the whole given landmarks reaches a total of 1,662 keypoints (543 x 3 coordinates) increasing model complexity and noise induced by the proposed zero padding to standardize the inputs to 130 frames, so 42 hand keypoints are flattened to get a total of 126 points. Whenever a hand is not detected, all the points for the hand are automatically filled with zeroes ensuring that each frame has consistently 126 points. A loop running through each of the videos collects an array containing an array of 126 x number of frames per video, however this does not ensure the homogeneous shape required to feed a neural network and as mentioned previously pre-padding was the technique decided to feel the missing frames in videos shorter than 130, pre-padding was chosen over other techniques like post-padding or truncating because as mentioned by (Dwarampudi and Reddy, 2019), truncating drives to information loss and post-padding performs worse than pre-padding on LSTMs due to LSTMs nature to remember, so as in a human brain, it is easier to remember recent information than older information, ending finally with a three dimension NumPy array composed of (#videos, 130, 126), this process was also done flipping the frames 180 degrees prior to the keypoint collection using this as a data augmentation technique as overfitting was identified in multiple occasions while designing the neural networks.

On the side of MMPose, is a Pytorch based pose estimation framework, part of the OpenMMLab project which allows quick deployment of algorithms to complete tasks like human pose estimation, animal pose estimation, face recognition, and fashion detection, the section involving the wholebody pose estimation algorithms is the one this project is focused on, specifically, CSPNext+UDP algorithm and RTMPose algorithm, CSPNext+UDP was decided to be tried as CSPNext is the backbone for RTMPose and when tested in the used hardware for the project, it works with a good accuracy and inference speed balance.

Both of these algorithms working under MMPose Inference API produce a list containing arrays with the image, 133 2D keypoints (17 for body, 6 for feet, 68 for face and 42 for hands), keypoints scores from 0 to 1, showing the prediction confidence for the 133 keypoints, and the classification box-score. In the project, the useful extracted information was the 133 keypoints with their respective scores, the scores are helpful as these algorithms always give the 133 keypoints, so even if there is not i.e. a hand in the picture, the algorithm will still predict the keypoint but the confidence score will be zero, for the rest of the process. When the inference API outputs the keypoint coordinates x and y, it does it according to the coordinates of pixels depending upon the size of the image frame, it outputs differently from a video with 640 x 480 resolution than a video 960 x 540 resolution, because of this, as previously identified 75% of the videos have a 640 x 480 resolution and all the videos, previous to be fed to the algorithm, are resized using OpenCV to standardize the keypoint coordinates outcome. The rest of the process was the same as MediaPipe for each of the two algorithms flattening the keypoints and scores, and then filling an array with zeroes from right to left to do the pre-padding ending with a three dimension NumPy array composed of (#videos, 130, 399), done as well flipping the frames 180 degrees prior to the keypoint collection for completing the data augmentation.

Finally, before passing the keypoints information for training, the keypoints collected were scaled for each of the three algorithms with the Scikit-Learn MaxAbsScaler, MaxAbsScaler was chosen among other scalers like MinMaxScaler or StandardScaler because the data is sparse and sparse data must not be centred, MaxAbsScaler scales each feature by dividing it with the largest maximum absolute value in each feature leaving transforming the data to a [-1,1] range and its formula is:

* 1. Neural Network Design

When designing the neural network, three different kinds of layers were considered based on previous works identified in the literature review section, 1D-CNN, LSTM, and GRU layers as each of them has the ability to learn from temporal sequences, in this process, Keras tuner in conjunction with Hyperband was used to adjust the hyperparameters of the network to find the optimal network configuration. Hyperband was chosen against other optimization algorithms because it is not as computationally expensive as i.e. Bayesian Optimization, due to its design to allocate more resources to the most promising models, random sampling at the beginning and stop the bad performing combinations in an early stage, allowing to try numerous combinations in less time which is an excellent point in favour considering the time limitation present on this project.

The models tried for each human pose estimation algorithm are presented in the table below.

| **Model** |
| --- |
| 1 1D-CNN Layer |
| 2 1D-CNN Layers |
| 3 1D-CNN Layers |
| 1 LSTM Layer |
| 2 LSTM Layers |
| 3 LSTM Layers |
| 1 1D-CNN + 1 GRU Layers |
| 2 1D-CNN + 1 GRU Layers |
| 1 1D-CNN + 1 LSTM Layers |
| 2 1D-CNN + 1 LSTM Layers |

Table 3. Models tuned with Keras Tuner

Even though the number of layers can be tuned with Keras Tuner, trying models adding layer by layer was preferred because it reduces the search space and gives the option to try the best model in each situation, as well as the option to select a smaller model if the inference time is too slow when evaluating the real time scenario, each model has been tuned to get the best validation F1 score possible, two executions per trial, early stop set to 30 epochs patience, and a maximum of 300 epochs resulting on 725 trials that the algorithm runs to find the best combination.

Validation F1 score was chosen as the metric to maximize over validation accuracy due to the class imbalance present in the data used to train and validate the models, specifically, the macro averaged F1 Score to ensure that all classes are equally important regardless of the number of samples.

Each of the best models chosen by Hyperband was further trained until 500 epochs without any early stop callback restriction, with the weights that produce the best results in the validation data being saved to ensure squeezing the best out of each.

The models tried were designed with 1, 2, and 3 CNN/LSTM/GRU hidden layers and just 1 fully connected hidden layer given several studies showing that networks with more than three hidden layers are not optimal in terms of time complexity (Uzair and Jamil, 2020) and this is a crucial factor to consider in the evaluation on real time commodity hardware implementation. By any means, the maximum number of four hidden layers was tried with the purpose of balancing the relation between the cost of the extra layer and the gained accuracy. 2 Dropout layers are also present being tuned with values from 0 to 0.9, if the chosen value by Hyperband is 0, for the network it is the same as if the Dropout layer did not exist.

The fully connected hidden layer present in every model was tuned with neurons ranging between 70 and 120, and the activation function, which selection is explored later. The tuned hyperparameters for the CNN Layers were the activation function, number of filters, searching values ranging from 50 to 200, 50 to maintain the relationship suggested in various literature stating that the number of neurons should be a number between the total inputs and outputs, 200 as maximum, because it was observed that after 200, the training time increase was already significant (being more notable in methods with input sizes of 130,399). Kernel size is tested with sizes between 2 to 20 at different scales depending on the model tried, and MaxPooling in the same range as the kernel size with the exception in the models composed by 3 1D-CNN Layers where the MaxPooling size was fixed to 2 because the combinations of sub-samplings in lots of sizes combinations resulted in errors due to the nature of MaxPooling causing to have a smaller input size than the number of down samples desired.

The LSTM/GRU layers were designed in the same range as the CNN layers for the number of neurons, adding L2 regularization to help reduce overfitting where it is present regularization was added to LSTM/GRU layers due to their trend to overfit data easily. The activation function is left with the default ‘Tanh’ as this is the activation function required by CUDA/CUDNN in this kind of network layer.

NADAM optimizer was chosen over ADAM because even though ADAM is a go to optimizer nowadays, the author of this integration of ADAM with Nesterov Momentum conducted experiments showing how NADAM can achieve better results in less time than ADAM in many situations , and as many combinations of algorithms are trained and tuned in this project, having that training increased speed just with this variation is a good benefit, this optimizer was also preferred over others as it can achieve good results without having to tune additional hyperparameters increasing the search space.

For activation functions to be tested, for the MediaPipe algorithm, the hyperparameter search was done using ‘ReLU’, ‘LeakyReLU’, ‘GELU’, ‘ELU’, and ‘SELU’. ‘MISH’ function was not included because it is not implemented in TensorFlow, leaving it out of the scope at the beginning, however, after finding a way to include it in the training, trials substituting the activation function in the best model found were done to compensate the situation, it was found that ‘MISH’, as expected, leads to improve some of the models accuracies. For the other two algorithms based on the MMPose framework, the activation function selection was done with ‘ReLU’, ‘GELU’, ‘ELU’, ‘SELU’, and ‘MISH’, ‘LeakyReLU’, was removed as it was observed from MediaPipe that it could not outperform the others in any model. Further reference about the hyperparameter tuning construction can be found in the Appendix section.

# RESULTS

As mentioned in the previous chapter, 30 different Human Pose Estimation Algorithm and Neural Network combinations were tested, and on each of them, 725 trials with different combinations of hyperparameters were carried out in an effort to evaluate as many combinations as possible before selecting a specific model. In this chapter, a summary of the trials is presented. All the best models are further trained in each combination up to 500 epochs, and also, while collecting the data to summarise these trials, some models were detected with potential based on their best epoch validation loss and F1 Score, these models were also trained further.

* 1. MediaPipe

After the hyperparameter tuning, the models fed by MediaPipe are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| 1 1D-CNN Layer | 74.78% | 1.4759 | 97.63% | 0.0941 | SELU |
| 2 1D-CNN Layers | 83.33% | 1.0351 | 99.18% | 0.0387 | GELU |
| 3 1D-CNN Layers | 82.30% | 1.2494 | 100.00% | 0.0017 | GELU |
| 1 LSTM Layer | 76.93% | 1.4636 | 97.90% | 0.1349 | SELU |
| 2 LSTM Layers | 80.74% | 1.3282 | 99.40% | 0.1341 | SELU |
| 3 LSTM Layers | 81.56% | 1.2911 | 99.85% | 0.1742 | ELU |
| 1 1D-CNN + 1 GRU Layers | 83.23% | 0.8656 | 98.76% | 0.1007 | SELU |
| 2 1D-CNN + 1 GRU Layers | 83.44% | 0.9564 | 100.00% | 0.0282 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 83.97% | 0.9505 | 99.81% | 0.0480 | SELU |
| 2 1D-CNN + 1 LSTM Layers | 83.22% | 0.8976 | 99.85% | 0.1012 | ELU |

Table 4.1 MediaPipe Best Models After Hyperparameter Tuning

As previously mentioned, each of the models that scored the best validation F1 Score was further trained up to 500 epochs, it is essential to note that each of the models was trained first with the same hyperparameters proposed by Hyperband and then, it was also trained replacing the activation function to ‘MISH’ (talking about the non RNN layers like GRU or LSTM). For the models where ‘SELU’ was the chosen activation function, they were also trained following the rules for self-normalization identified in the literature, however, in any case, following these rules improved the performance. This could be due to one of the rules for self-normalization being that the network needs to be a fully connected network which is not the case, and also, the specified architectures were designed to have Dropout layers that are not compatible with the rules as well, so the process for these models was using Lecunnormal kernel initializer and removed the Dropout layers, but no success was obtained.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| 1 1D-CNN Layer | 77.97% | 2.3795 | 99.30% | 0.0252 | SELU |
| 2 1D-CNN Layers | 83.63% | 1.2460 | 99.60% | 0.0209 | GELU |
| 3 1D-CNN Layers | 85.12% | 1.7271 | 100.00% | 4.53E-05 | MISH |
| 1 LSTM Layer | 79.34% | 1.6643 | 100.00% | 0.0547 | MISH |
| 2 LSTM Layers | 83.10% | 1.2800 | 99.59% | 0.0938 | SELU |
| 3 LSTM Layers | 81.72% | 1.5337 | 99.92% | 0.1599 | ELU |
| 1 1D-CNN + 1 GRU Layers | 84.45% | 1.1434 | 99.85% | 0.0536 | MISH |
| 2 1D-CNN + 1 GRU Layers | 84.68% | 0.9879 | 100.00% | 0.0132 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 84.59% | 0.9023 | 99.89% | 0.0334 | MISH |
| 2 1D-CNN + 1 LSTM Layers | 85.99% | 1.1932 | 100.00% | 0.0313 | MISH |

Table 4.2 MediaPipe Best Models After Training 500 Epochs

From Table 4.2, the below key points are important to highlight:

* Even though there is a sign of overfitting with the training score already being close to 100%, additional training without early stop restrictions always helped to improve the validation F1 Score.
* It is not the same case for the validation loss, in some instances the validation loss increased, raising another flag of overfitting presence in the models.
* ‘MISH’ assisted five of the ten models to increase their performance by just changing the activation function. For this point it is important to remember that further training was completed with the original activation function and replacing it with ‘MISH’.
* The best two models use four hidden layers, which was expected per (Uzair and Jamil, 2020) point, the performance of this needs to be evaluated when executing the real time implementation.

The models with 2 1D-CNN + 1 LSTM layers and 3 1D-CNN layers were the top performers, having both ‘MISH’ as activation function.

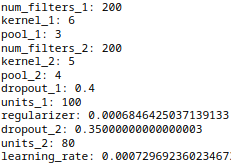
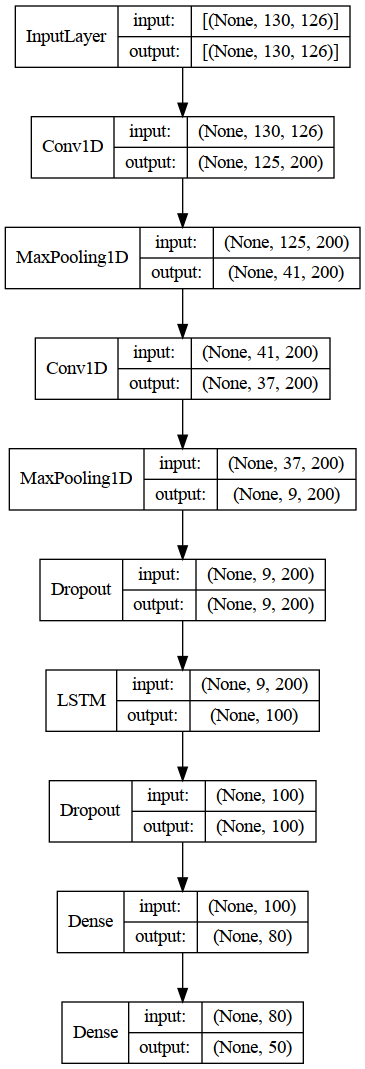


Figure 4.1 2 1D-CNN + 1 LSTM Layers Model Architecture

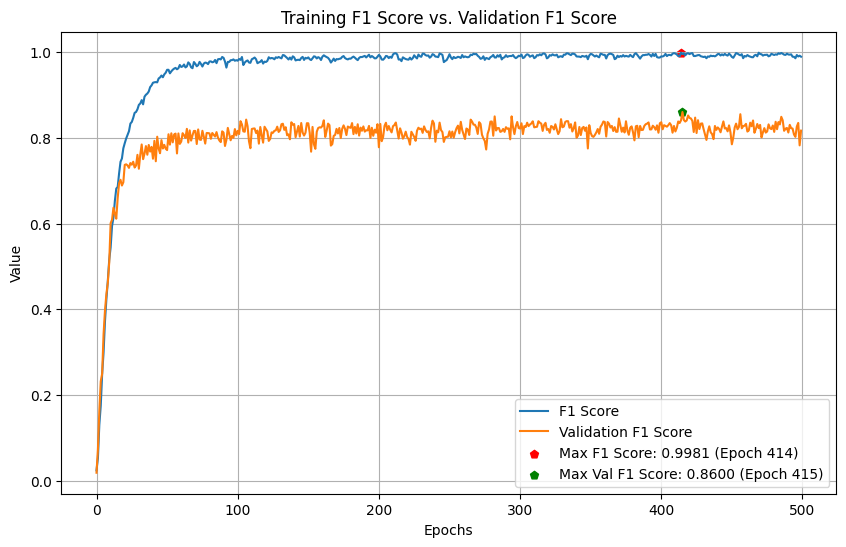
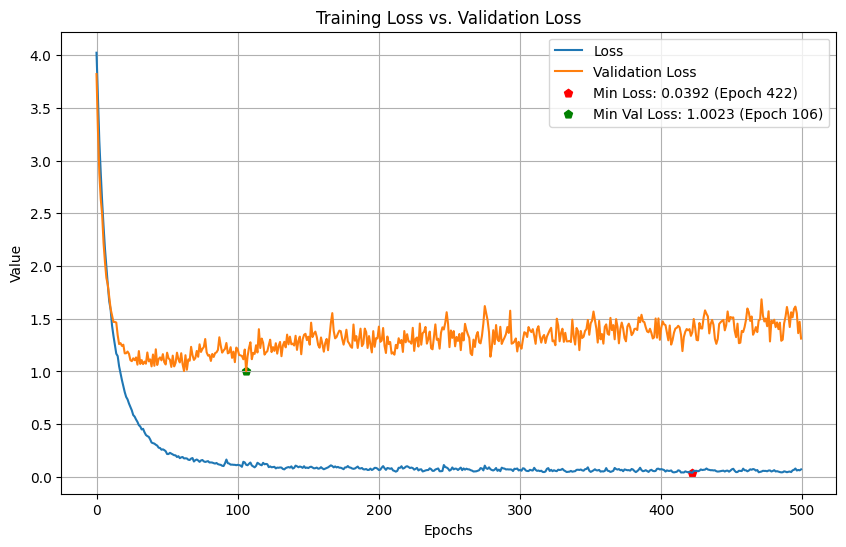


Figure 4.2 2 1D-CNN + 1 LSTM Layers Training History

From the graphs above, it can be seen that the model with 2 1D-CNN + 1 LSTM Layers is overfitting and reached its maximum capacity around 100 epochs where the validation loss is at its minimum, from there the model Validation F1 Score continues increasing steadily, but in the same way the Validation Loss does, the model checkpoint is at epoch 415 where it reached the 85.99% and 1.1932 Validation Loss.

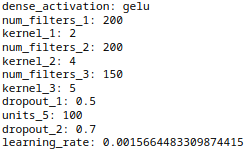
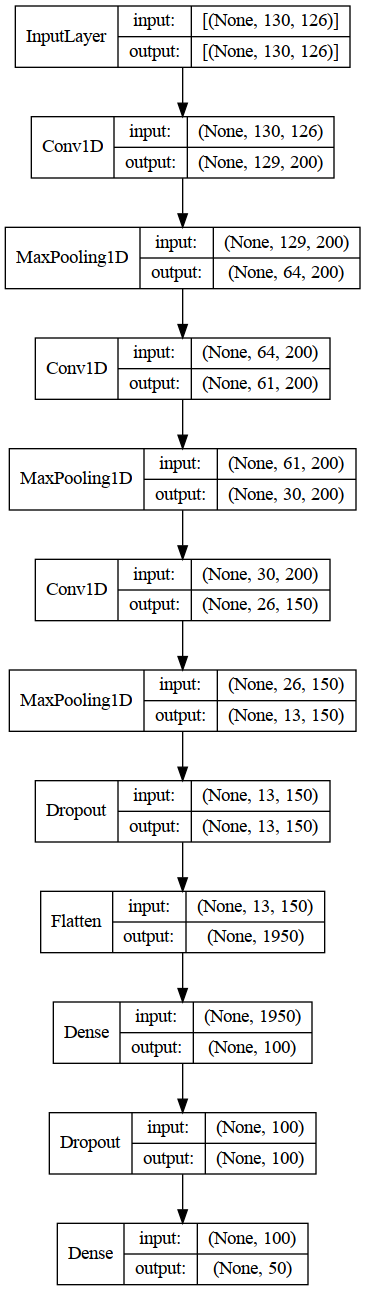


Figure 4.3 3 1D-CNN Layers Model Architecture

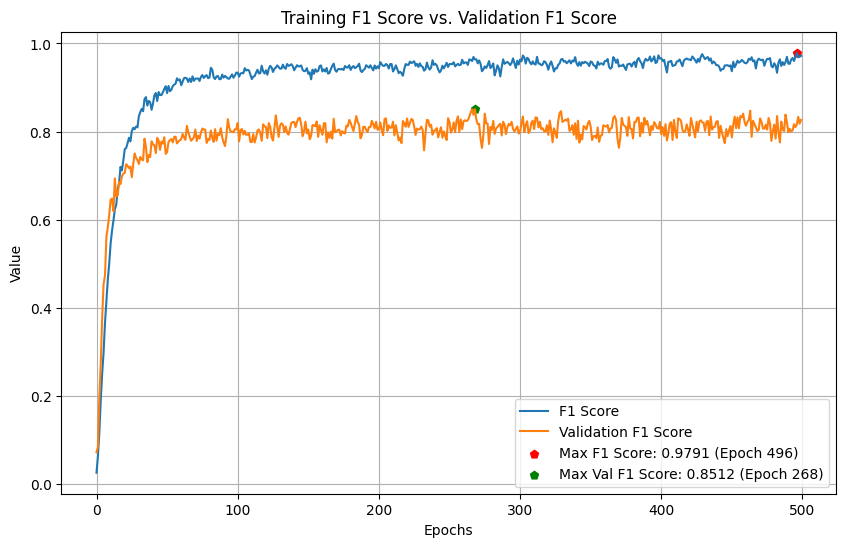
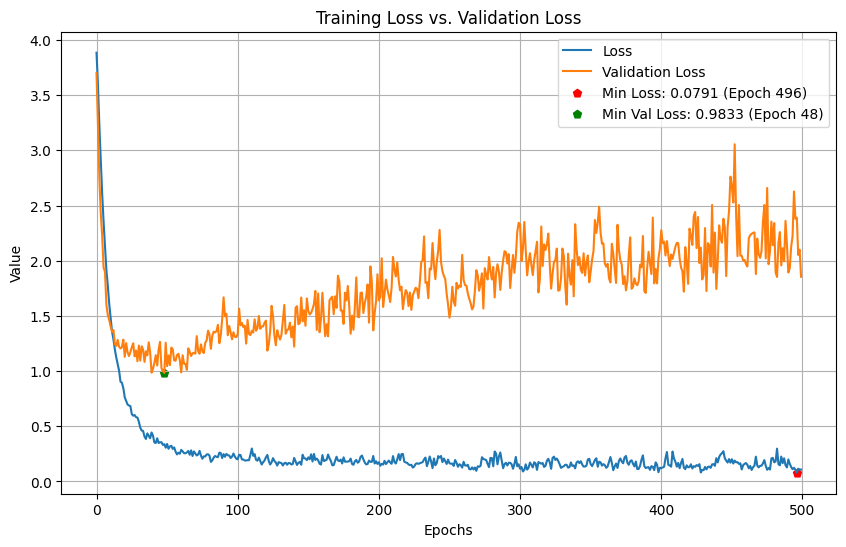


Figure 4.4 3 1D-CNN Layers Training History

In the case of the model with 3 1D-CNN Layers, the overfitting is even more evident with the Validation Loss just constantly increasing from epoch 48, and while there exists an increase in the Validation F1 Score and is the score more interest is put on this project, the validation loss has almost duplicated from its best point in epoch 268 where the model snapshot is taken.

Due to this situation, the decision to further explore models was taken and this was done by capturing the training statistics of the ten best model configurations per architecture, giving place to some models worth trying. In this chapter, just a few relevant tables are shown, but the rest can be found in the Appendix section.

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 664 | 80.10% | 1.0440 | 83.96% | 0.5008 |
| 3 1D-CNN Layers | 715 | 79.98% | 1.2299 | 85.89% | 0.4373 |
| 3 1D-CNN Layers | 718 | 79.45% | 1.0102 | 77.47% | 0.7096 |
| 1 1D-CNN + 1 GRU Layer | 665 | 80.15% | 1.0597 | 87.38% | 0.4970 |
| 2 1D-CNN + 1 GRU Layers | 721 | 82.28% | 1.0919 | 88.08% | 0.4878 |
| 2 1D-CNN + 1 GRU Layers | 700 | 80.19% | 1.2403 | 85.81% | 0.7634 |
| 1 1D-CNN + 1 LSTM Layers | 722 | 80.39% | 0.8521 | 82.31% | 0.6514 |

Table 4.3 MediaPipe Additional Models Best Epoch Checkpoint

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss[[1]](#footnote-2)** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 664 | 80.35% | 0.9967 | 99.24% | 0.0394 | GELU |
| 3 1D-CNN Layers | 715 | 81.06% | 1.2981 | 99.31% | 0.0293 | ELU |
| 3 1D-CNN Layers | 718 | 79.77% | 1.0140 | 94.29% | 0.2306 | ELU |
| 1 1D-CNN + 1 GRU Layer | 665 | 80.15% | 1.0999 | 99.20% | 0.1624 | ELU |
| 2 1D-CNN + 1 GRU Layers | 721 | 82.74% | 1.0158 | 98.76% | 0.1352 | LeakyReLU |
| 2 1D-CNN + 1 GRU Layers | 700 | 81.39% | 1.2823 | 98.91% | 0.3963 | LeakyReLU |
| 1 1D-CNN + 1 LSTM Layers | 722 | 81.95% | 0.8031 | 96.36% | 0.1995 | SELU |

Table 4.4 MediaPipe Additional Models After Hyperparameter Tuning

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 664 | 85.51% | 1.3073 | 99.89% | 0.0031 | MISH |
| 3 1D-CNN Layers | 715 | 86.03% | 1.0245 | 99.89% | 0.0038 | MISH |
| 3 1D-CNN Layers | 718 | 82.80% | 1.0711 | 99.24% | 0.0457 | MISH |
| 1 1D-CNN + 1 GRU Layer | 665 | 82.43% | 1.2409 | 99.92% | 0.1306 | ELU |
| 2 1D-CNN + 1 GRU Layers | 721 | 86.32% | 1.2372 | 100.00% | 0.1029 | MISH |
| 2 1D-CNN + 1 GRU Layers | 700 | 82.92% | 1.4969 | 99.15% | 0.4481 | LeakyReLU |
| 1 1D-CNN + 1 LSTM Layers | 722 | 84.41% | 1.1209 | 99.78% | 0.0362 | SELU |

Table 4.5 MediaPipe Additional Models After MediaPipe Training 500 Epochs

Key points from tables 4.3 to 4.5:

* Seven new models with potential were trained, the criteria to select which ones among the extra 90 models available were the ones where the F1 Score and validation F1 score differ by less than 10% from each other, and it was near 80%.
* Models with the ‘MISH’ activation function again in four of seven models outperformed their similar with the original function.
* Overfitting seems to be present in these models as well due to the validation F1 Score and validation loss just consistently improved in one of the models after training 500 Epochs “3 1D-CNN Trial 715”.
* Two of these models outperformed the previous ones reaching 86% validation F1 Score.

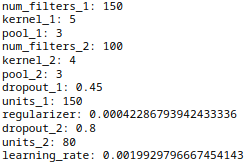
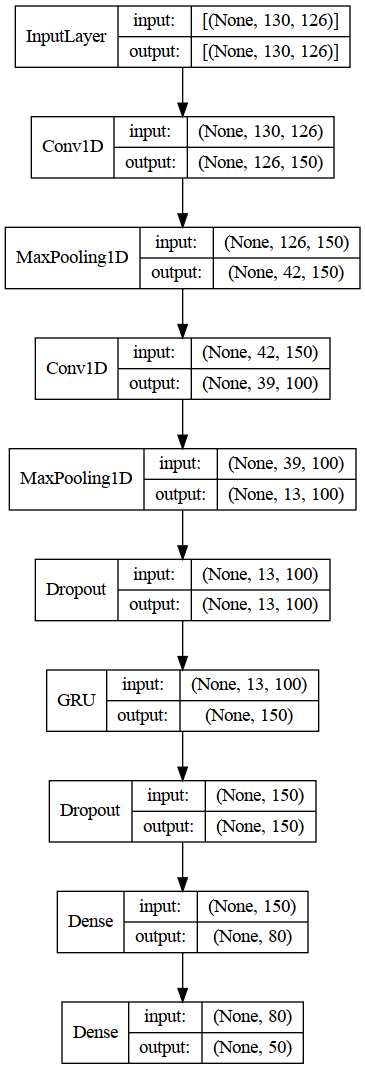


Figure 4.5 2 1D-CNN + 1 GRU Layers Trial 721 Model Architecture

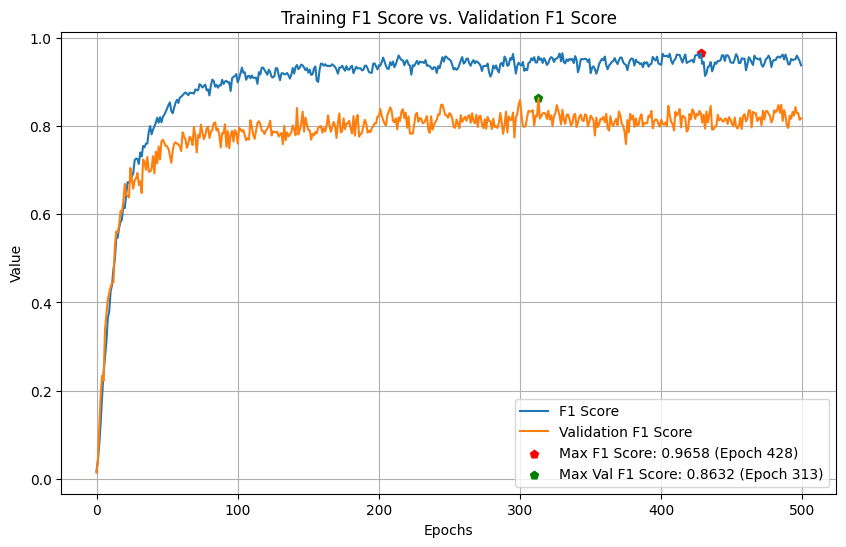
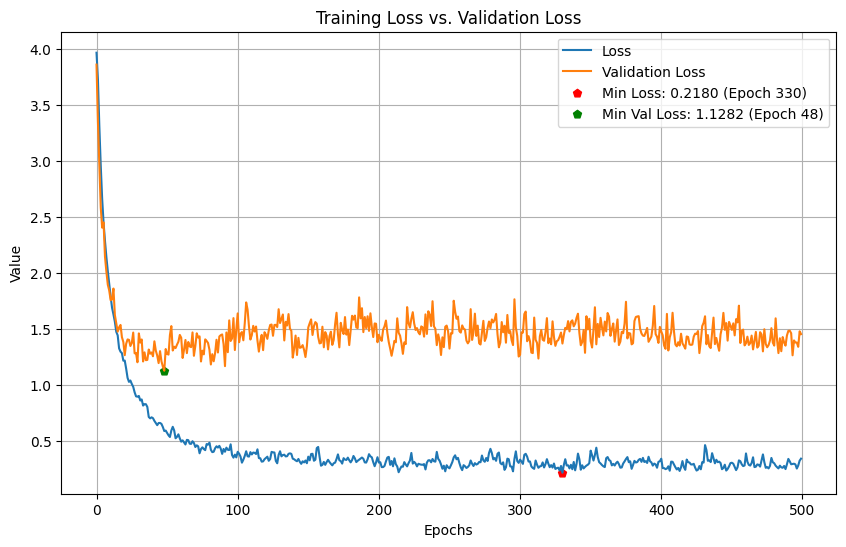


Figure 4.6 2 1D-CNN + 1 GRU Layers Trial 721 Training History

The model described in Figures 4.5 and 4.6 has improved over the previous models while there still exists the possibility of overfitting as the validation loss touched its minimum point at epoch 48, the increase is not as dramatic as in the previous models, also the validation F1 Score continued increasing until epoch 313 where it reached its maximum value of 86.32% against a validation loss increase of 1.2372, is not ideal, but it is the model that got the highest validation F1 Score and the lowest signs of overfitting, showing the best balance between the validation and training metrics, by epoch 313 where the weights were saved, 0.26210 and 95.70% were the training loss and accuracy respectively.

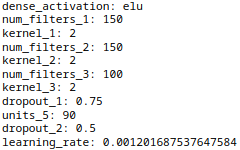
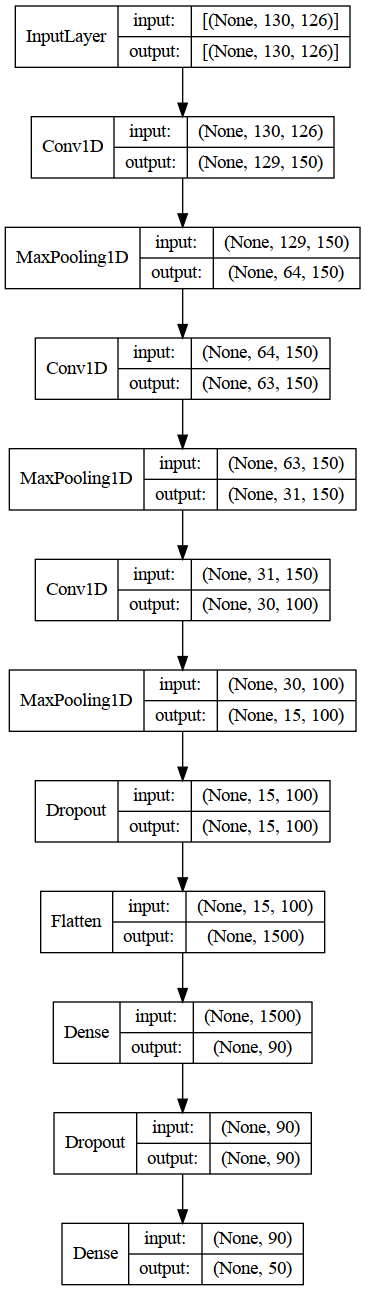


Figure 4.7 3 1D-CNN Trial 715 Model Architecture

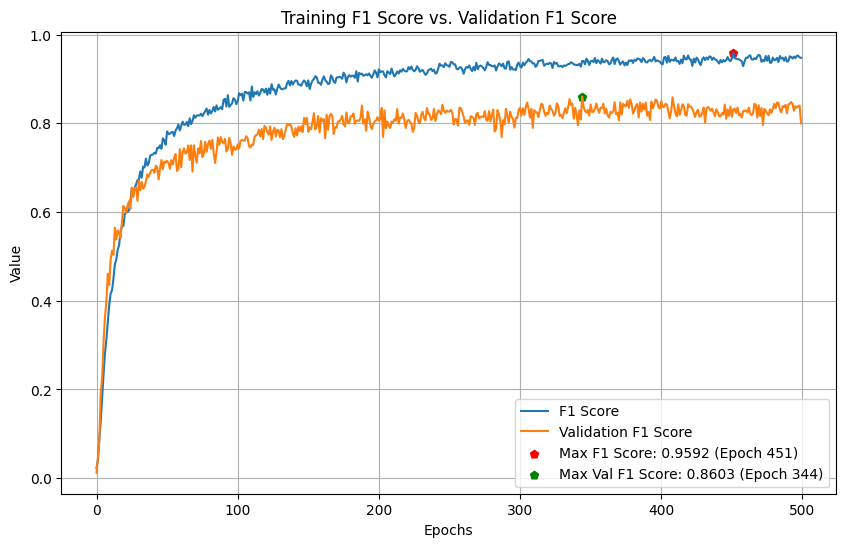


Figure 4.8 3 1D-CNN Layers Trial 715 Training History

The model with 3 1D-CNN Layers Trial 715 showed to be a good performer as well, where overfitting is less evident than the other model with 3 1D-CNN Layers, and it also has a good balance like the 2 1D-CNN + 1 GRU Layers Trial 721 model having the validation F1 Score a steady increase in apparently the same proportion as the training F1 Score which is a good sign, however, overfitting cannot be discarded entirely as the validation loss also increases from epoch 123 equally proportional to the training time, however, in epoch 344 where the checkpoint was made, the validation F1 Score is 86.03% and the validation loss is 1.0245 which is a lower value than the 2 1D-CNN + 1 GRU trial 721 model and is not too far from its minimum value during training which was .9646.

Overall, there are three models that, even where the scenario is not ideal and overfitting seems to be not completely dissipated, have shown good F1 Scores in the validation data, the validation loss is not the best, but as previously mentioned, the number of samples is imbalanced which can make small errors in minority classes to increase the loss value. Also, analysing the hyperparameters, the last two are less complex models with more robust regularization, which makes sense on the signals of reduced overfitting.

| **Hyperparameter** | **3 1D-CNN** | **3 1D-CNN Trial 715** |
| --- | --- | --- |
| Filters 1 | 200 | 150 |
| Kernel 1 | 2 | 2 |
| Filters 2 | 200 | 150 |
| Kernel 2 | 4 | 2 |
| Filters 3 | 150 | 100 |
| Kernel 3 | 5 | 2 |
| Dropout 1 | 0.50 | 0.75 |
| Dense Units | 100 | 90 |
| Dropout 2 | 0.70 | 0.50 |
| Learning Rate | 0.001566 | 0.001201 |

Table 4.6 Hyperparameters Comparison Between 3 1D-CNN Top Performer Models

| **Hyperparameter** | **2 1D-CNN + 1 LSTM** | **2 1D-CNN + 1 GRU Trial 721** |
| --- | --- | --- |
| Filters 1 | 200 | 150 |
| Kernel 1 | 6 | 5 |
| Pool 1 | 3 | 3 |
| Filters 2 | 200 | 100 |
| Kernel 2 | 5 | 4 |
| Pool 2 | 4 | 3 |
| Dropout 1 | 0.40 | 0.45 |
| LSTM/GRU Units | 100 | 150 |
| Regularizer | 0.000684 | 0.000422 |
| Dropout 2 | 0.35 | 0.80 |
| Dense Units | 80 | 80 |
| Learning Rate | 0.000729 | 0.001992 |

Table 4.7 Hyperparameters Comparison Between 2 CNN-RNN Top Performer Models

* 1. RTMPose

After the hyperparameter tuning, the models fed by MMPose/RTMPose are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| 1 1D-CNN Layer | 53.53% | 2.1379 | 99.31% | 0.0905 | ELU |
| 2 1D-CNN Layers | 69.54% | 1.2194 | 92.74% | 0.431 | MISH |
| 3 1D-CNN Layers | 66.03% | 1.3640 | 90.51% | 0.4578 | GELU |
| 1 LSTM Layer | 59.01% | 2.2577 | 98.90% | 0.1129 | ELU |
| 2 LSTM Layers | 67.02% | 2.3504 | 100.00% | 0.2579 | GELU |
| 3 LSTM Layers | 70.73% | 1.9141 | 99.77% | 0.2002 | MISH |
| 1 1D-CNN + 1 GRU Layers | 75.56% | 1.1833 | 99.28% | 0.1752 | ELU |
| 2 1D-CNN + 1 GRU Layers | 76.42% | 1.0145 | 93.16% | 0.2685 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 69.36% | 1.2940 | 95.63% | 0.3471 | ELU |
| 2 1D-CNN + 1 LSTM Layers | 76.25% | 1.3475 | 100.00% | 0.1221 | MISH |

Table 4.8 RTMPose Best Models After Hyperparameter Tuning

For RTMPose, the same approach as in MediaPipe was taken, each of the models was further trained up to 500 epochs, with the only difference that since ‘MISH’ was included since the beginning in the hyperparameter tuning, no further training changing the function was made in all models with the only exception of two of the top performers where ‘MISH’ was not the activation function but no success was obtained in this case replacing it. Also, Lecunnormal initialization for ‘SELU’ activation models was omitted since no improvement was observed in MediaPipe.

| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| --- | --- | --- | --- | --- | --- |
| 1 1D-CNN Layer | 55.10% | 2.4444 | 100.00% | 0.5510 | ELU |
| 2 1D-CNN Layers | 72.72% | 1.1180 | 99.83% | 0.0722 | MISH |
| 3 1D-CNN Layers | 70.11% | 1.5131 | 98.66% | 0.1096 | GELU |
| 1 LSTM Layer | 60.95% | 2.5357 | 99.96% | 0.0613 | ELU |
| 2 LSTM Layers | 68.15% | 2.2822 | 100.00% | 0.1515 | GELU |
| 3 LSTM Layers | 69.32% | 2.1100 | 99.54% | 0.1956 | MISH |
| 1 1D-CNN + 1 GRU Layers | 76.78% | 1.1852 | 99.36% | 0.1627 | ELU |
| 2 1D-CNN + 1 GRU Layers | 79.12% | 0.9192 | 99.01% | 0.0864 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 71.09% | 1.4886 | 99.88% | 0.0613 | ELU |
| 2 1D-CNN + 1 LSTM Layers | 77.45% | 1.2420 | 99.93% | 0.1019 | MISH |

Table 4.9 RTMPose Best Models After Training 500 Epochs

From Table 4.9 the below key points are important to highlight:

* Validation F1 Score did not improve in all the cases, in the case of the model with 3 LSTM Layers could not reach the same score as the score obtained directly by Hyperband, probably caused by different random seeds used by the algorithm and TensorFlow.
* Validation Loss was reduced in just four of the ten algorithms.
* ‘MISH’ only outperformed in three models this time, with ‘ELU’ being the most frequent activation function.
* The top performer models are again using four hidden layers.

The best models in RTMPose were 2 1D-CNN + 1 GRU and 2 1D-CNN + 1 LSTM Layers using ‘SELU’ as activation function and ‘MISH’ respectively, both combining CNN + RNN layers.

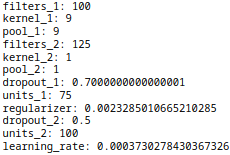
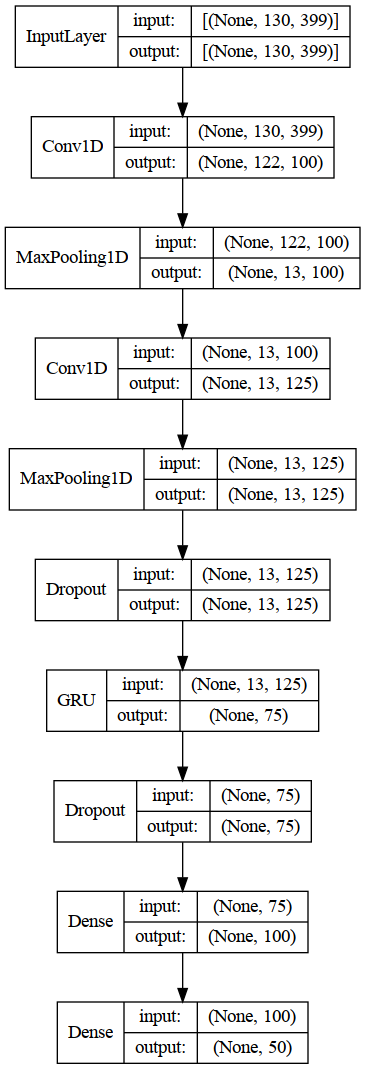


Figure 4.9 2 1D-CNN + 1 GRU Model Architecture

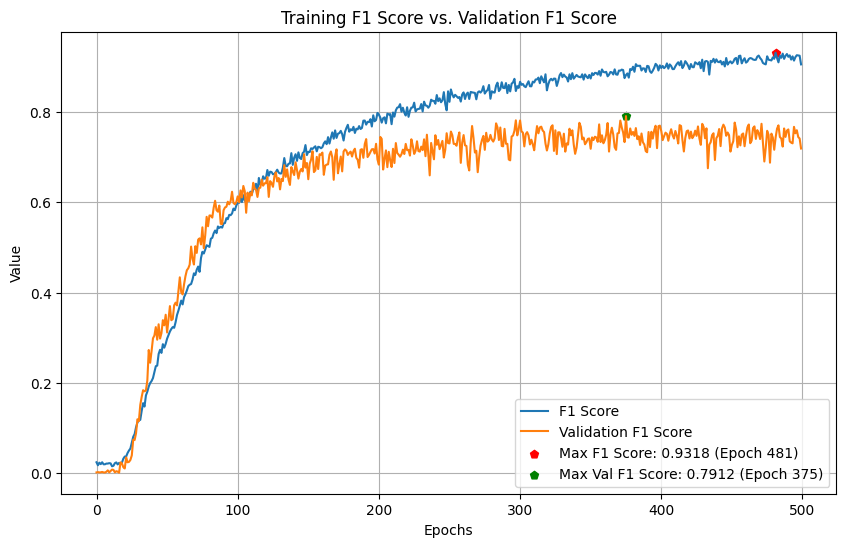
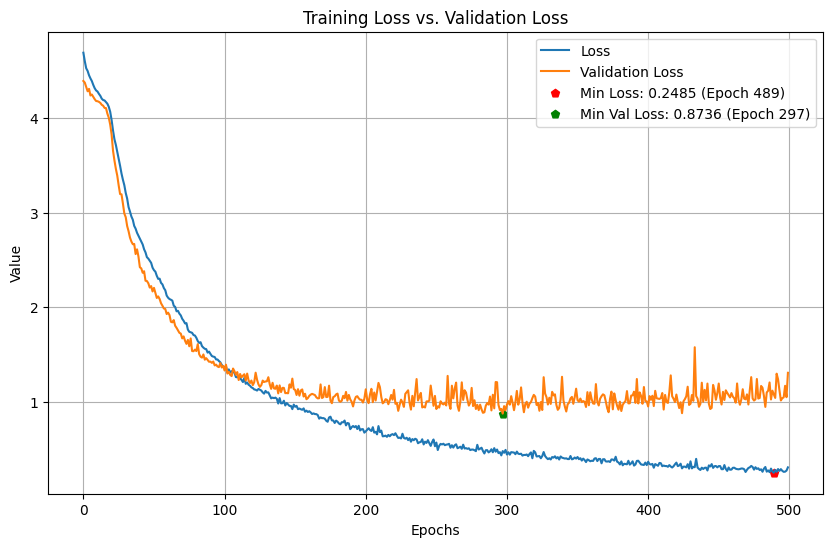


Figure 4.10 2 1D-CNN + 1 GRU Layers Training History

From the graphs in Figure 4.10, the model with 2 1D-CNN + 1 GRU Layers is overfitting but is not as severe as other situations, from epoch 200 the validation loss starts to stabilise but continues decreasing slowly until epoch 297 where it reaches its lower value, for the validation F1 Score, it continues increasing until epoch 375 where the checkpoint was saved, and the validation loss has not increased that much for this epoch going from .8736 minimum point to .9192 and the training F1 Score is 88.22% at this point.

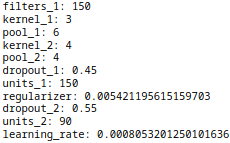
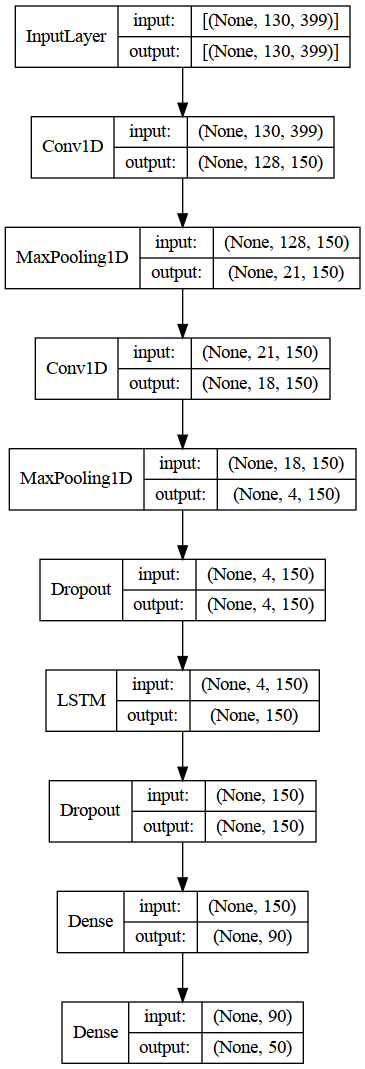


Figure 4.11 2 1D-CNN + 1 LSTM Model Architecture

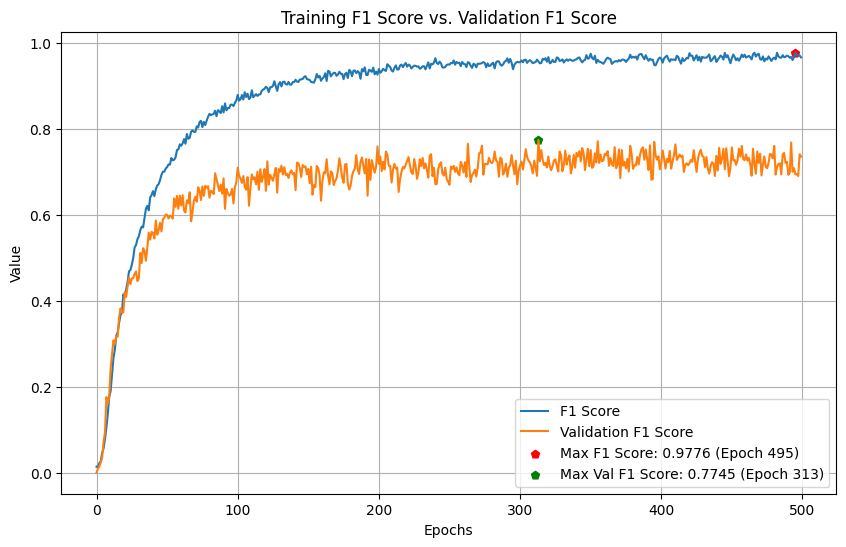
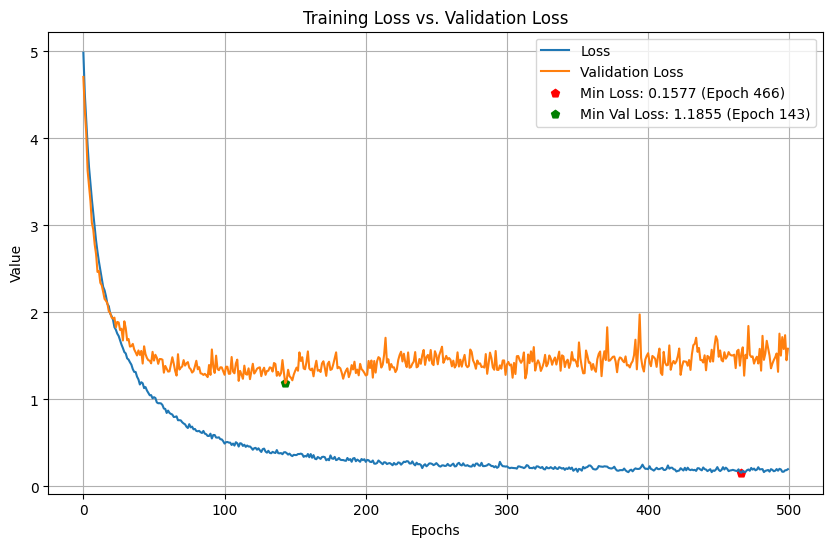


Figure 4.12 2 1D-CNN + 1 LSTM Layers Training History

While the overfitting in the previous model was not extremely severe, in the model shown above this is more evident with the metrics starting to diverge since an early stage of the training, by the moment the model reached its maximum validation F1 Score, the model has reached already 96% on training data the loss also has these signs of overfitting as its minimum point was touched in epoch 143, and from there, it just increases reaching values close to 2 around epoch 400.

Any of the models fed by RTMPose could make it to 80% and the tuner suffered to find promising models with this Human Pose Estimation algorithm having some that could not even reach the 60% mark. An important consideration to take is that this one also considers face and pose keypoints which adds complexity to the problem. As in MediaPipe, some additional models were considered based on their metrics, but in this case, any of them could outperform the previous models previously identified by Hyperband.

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 718 | 64.97% | 1.4593 | 60.86% | 1.2088 |
| 1 1D-CNN + 1 GRU Layer | 593 | 69.95% | 1.3452 | 75.52% | 0.8362 |
| 1 1D-CNN + 1 GRU Layer | 722 | 69.34% | 1.2266 | 66.09% | 1.0993 |
| 1 1D-CNN + 1 LSTM Layers | 717 | 65.02% | 1.4289 | 69.70% | 1.1101 |
| 2 1D-CNN + 1 LSTM Layers | 715 | 72.02% | 1.2437 | 78.02% | 0.7161 |

Table 4.10 RTMPose Additional Models Best Epoch Checkpoint

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss[[2]](#footnote-3)** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 718 | 65.77% | 1.4396 | 88.64% | 0.4830 | ELU |
| 1 1D-CNN + 1 GRU Layer | 593 | 70.39% | 1.3141 | 93.73% | 0.3788 | ELU |
| 1 1D-CNN + 1 GRU Layer | 722 | 69.61% | 1.1756 | 84.69% | 0.5961 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 717 | 69.79% | 1.2795 | 92.58% | 0.3942 | SELU |
| 2 1D-CNN + 1 LSTM Layers | 715 | 72.04% | 1.2463 | 87.61% | 0.4285 | GELU |

Table 4.11 RTMPose Additional Models After Hyperparameter Tuning

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 3 1D-CNN Layers | 718 | 70.68% | 1.1519 | 96.80% | 0.1865 | ELU |
| 1 1D-CNN + 1 GRU Layer | 593 | 68.00% | 1.5675 | 99.96% | 0.1200 | ELU |
| 1 1D-CNN + 1 GRU Layer | 722 | 75.61% | 0.9942 | 98.30% | 0.1065 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 717 | 69.03% | 1.4317 | 99.83% | 0.1189 | SELU |
| 2 1D-CNN + 1 LSTM Layers | 715 | 73.06% | 1.3636 | 98.49% | 0.0999 | GELU |

Table 4.12 RTMPose Additional Models After Training 500 Epochs

* 1. CSPNext + UDP

After the hyperparameter tuning, the models fed by MMPose/CSPNext are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| 1 1D-CNN Layer | 44.43% | 2.2785 | 94.91% | 0.4082 | SELU |
| 2 1D-CNN Layers | 63.97% | 1.3178 | 96.93% | 0.2394 | MISH |
| 3 1D-CNN Layers | 66.48% | 1.4420 | 99.75% | 0.0543 | MISH |
| 1 LSTM Layer | 60.88% | 1.9230 | 92.03% | 0.2982 | GELU |
| 2 LSTM Layers | 65.56% | 2.1651 | 99.20% | 0.4180 | GELU |
| 3 LSTM Layers | 69.98% | 1.9652 | 99.36% | 0.2280 | MISH |
| 1 1D-CNN + 1 GRU Layers | 72.47% | 1.1993 | 98.18% | 0.2019 | ELU |
| 2 1D-CNN + 1 GRU Layers | 76.82% | 1.0408 | 94.20% | 0.2381 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 69.08% | 1.3643 | 90.52% | 0.4791 | SELU |
| 2 1D-CNN + 1 LSTM Layers | 71.79% | 1.4066 | 100.00% | 0.0779 | MISH |

Table 4.13 CSPNext Best Models After Hyperparameter Tuning

The same process as in RTMPose was conducted, each of the models was further trained up to 500 epochs, also including ‘MISH’ since the beginning in the hyperparameter tuning. Also, no further training changing the function was made in all models with the only exception of two of the top performers where ‘MISH’ was not the activation function, but no success was obtained either.

| **Model** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| --- | --- | --- | --- | --- | --- |
| 1 1D-CNN Layer | 42.03% | 2.7388 | 99.96% | 0.0302 | SELU |
| 2 1D-CNN Layers | 68.15% | 1.3521 | 99.96% | 0.0371 | MISH |
| 3 1D-CNN Layers | 70.78% | 1.5809 | 100.00% | 0.0030 | MISH |
| 1 LSTM Layer | 63.67% | 2.4111 | 99.00% | 0.0956 | GELU |
| 2 LSTM Layers | 71.44% | 1.8845 | 99.96% | 0.2470 | GELU |
| 3 LSTM Layers | 65.29% | 1.9338 | 94.86% | 0.3059 | MISH |
| 1 1D-CNN + 1 GRU Layers | 73.23% | 1.4189 | 100.00% | 0.0487 | ELU |
| 2 1D-CNN + 1 GRU Layers | 78.20% | 1.0642 | 97.71% | 0.1385 | SELU |
| 1 1D-CNN + 1 LSTM Layers | 71.60% | 1.3088 | 98.53% | 0.1758 | SELU |
| 2 1D-CNN + 1 LSTM Layers | 73.81% | 1.8662 | 100.00% | 0.0280 | MISH |

Table 4.14 CSPNext Best Models After Training 500 Epochs

From Table 4.14, the below key points are important to highlight:

* Validation F1 Score did not improve in all the cases. In the case of the model with 1 1D-CNN Layer, it could not reach the same score as the score obtained directly by Hyperband because of the random seed in weights initialization present in RTMPose.
* Validation Loss was reduced in just three of the ten algorithms this time.
* ‘MISH’ outperformed the rest of activation functions with four models, followed by ‘SELU’.
* The top performer models are using four hidden layers for the third time.

The best models in CSPNext were 2 1D-CNN + 1 GRU and 2 1D-CNN + 1 LSTM Layers using ‘SELU’ as activation function and ‘MISH’ respectively, both combining CNN + RNN layers, the top performer for CSPNext, is exactly the same model than the best in RTMPose.

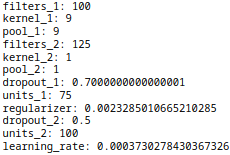
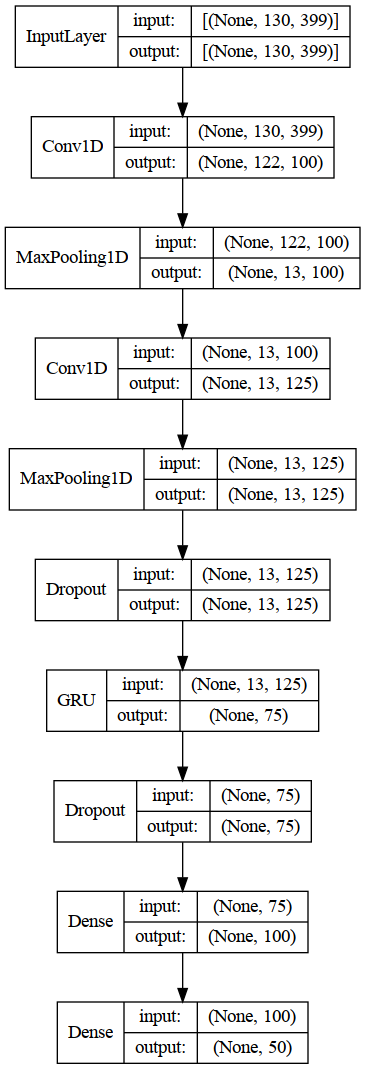


Figure 4.13 2 1D-CNN + 1 GRU Model Architecture

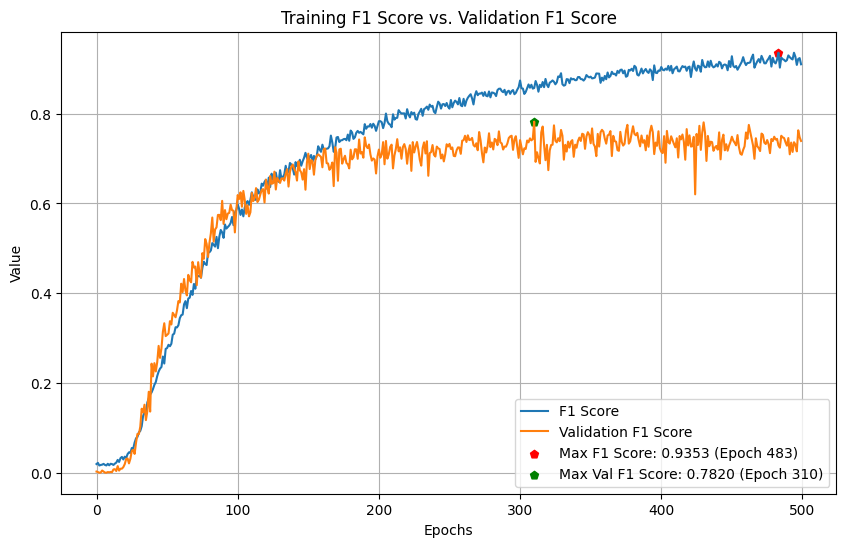
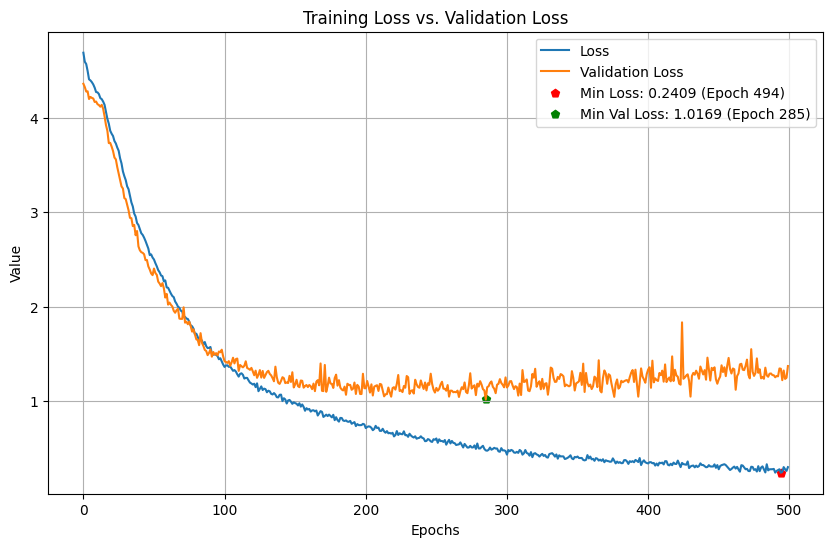


Figure 4.14 2 1D-CNN + 1 GRU Layers Training History

The model with 2 1D-CNN + 1 GRU Layers is overfitting, presenting a similar behaviour to the model in RTMPose. From epoch 200 the validation loss starts to stabilise decreasing gradually until epoch 285. The validation F1 Score increases until epoch 310 with a 78.20% peak value, at which point the validation loss has not increased significantly from its lowest point reaching 1.0642, and the training F1 Score is 85.67% at this point.

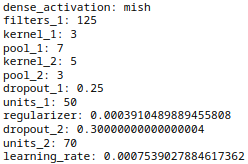
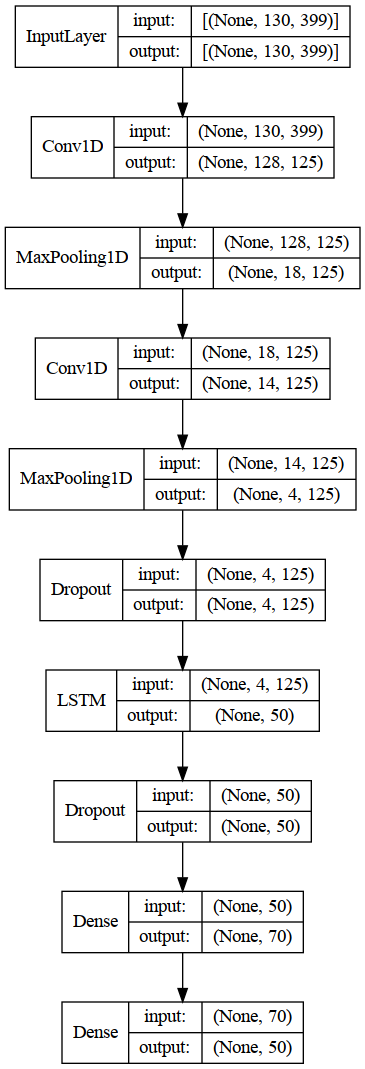


Figure 4.15 2 1D-CNN + 1 LSTM Model Architecture

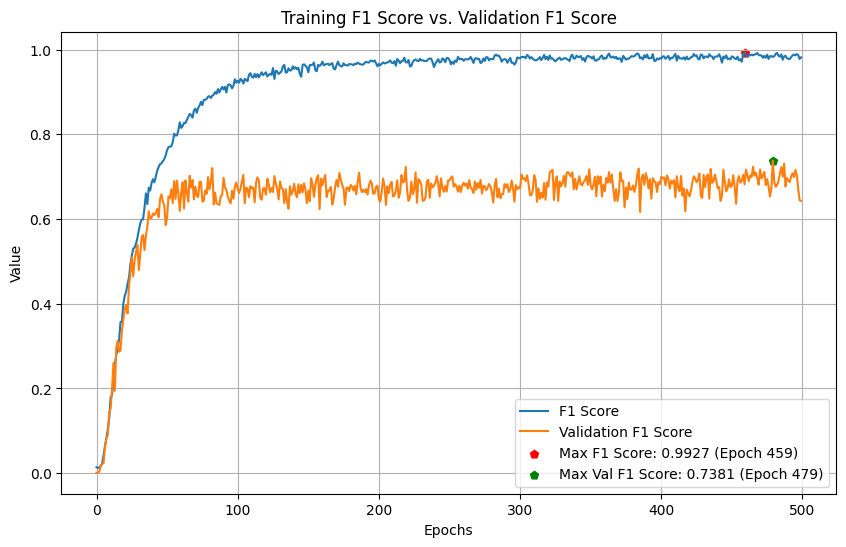
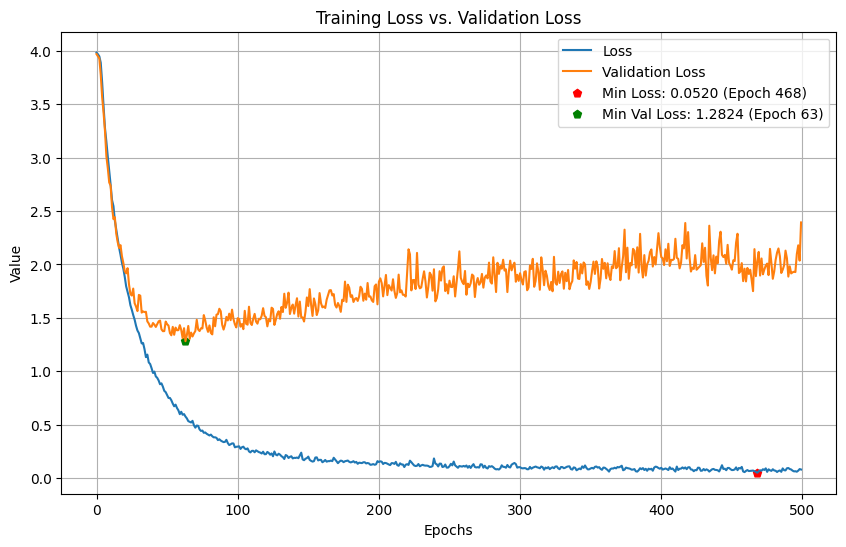


Figure 4.16 2 1D-CNN + 1 LSTM Layers Training History

From the models presented, the model shown in Figure 4.16 is the one with the most apparent overfitting signals among all, having a validation loss decrease until epoch 63. After this epoch, the validation loss just increases constantly, reaching values of nearly 2.5 which is almost the double of its minimum point, by the moment the maximum validation F1 score was achieved, the validation loss was 1.8662 already being around 1.5 times the minimum value.

CSPNext fed models were the ones with the lowest validation F1 Score having the 1D-CNN model with a score that could not even make it to the 45% mark. Three models were considered for further training with another 2 1D-CNN + 1 GRU model surpassing the 2 1D-CNN + 1 LSTM obtaining a slightly higher F1 Score and with a lower loss.

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- | --- |
| 1 1D-CNN + 1 GRU Layer | 717 | 67.45% | 1.3248 | 75.38% | 0.7744 |
| 1 1D-CNN + 1 GRU Layer | 722 | 66.15% | 1.2593 | 65.65% | 1.1391 |
| 2 1D-CNN + 1 GRU Layers | 698 | 70.06% | 1.2535 | 73.20% | 0.9557 |

Table 4.15 CSPNext Additional Models Best Epoch Checkpoint

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss[[3]](#footnote-4)** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 1D-CNN + 1 GRU Layer | 717 | 67.47% | 1.3133 | 88.01% | 0.4298 | ELU |
| 1 1D-CNN + 1 GRU Layer | 722 | 67.88% | 1.2218 | 86.92% | 0.4927 | SELU |
| 2 1D-CNN + 1 GRU Layers | 698 | 71.51% | 1.2167 | 85.07% | 0.6389 | MISH |

Table 4.16 CSPNext Additional Models After Hyperparameter Tuning

| **Model** | **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** | **Activation Function** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 1D-CNN + 1 GRU Layer | 717 | 62.89% | 2.2838 | 100.00% | 0.0551 | ELU |
| 1 1D-CNN + 1 GRU Layer | 722 | 73.67% | 1.0799 | 97.34% | 0.1594 | SELU |
| 2 1D-CNN + 1 GRU Layers | 698 | 73.87% | 1.3137 | 99.85% | 0.1068 | MISH |

Table 4.17 CSPNext Additional Models After Training 500 Epochs

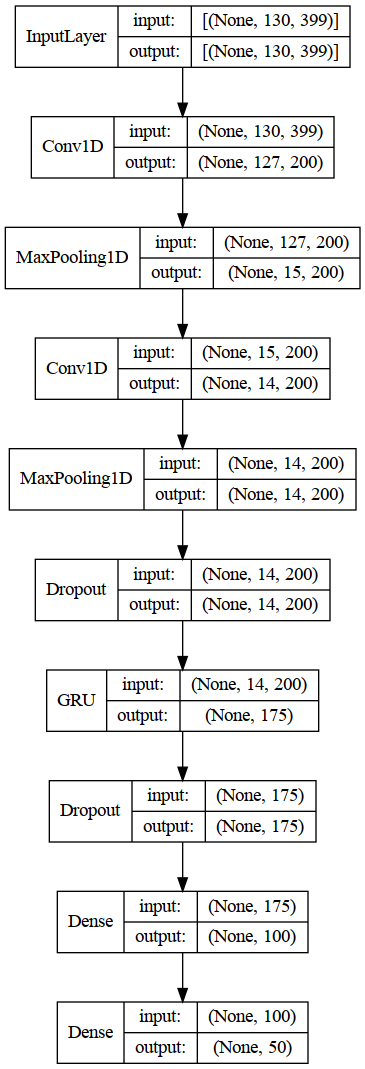
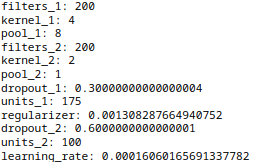
 

Figure 4.17 2 1D-CNN + 1 GRU Trial 698 Model Architecture

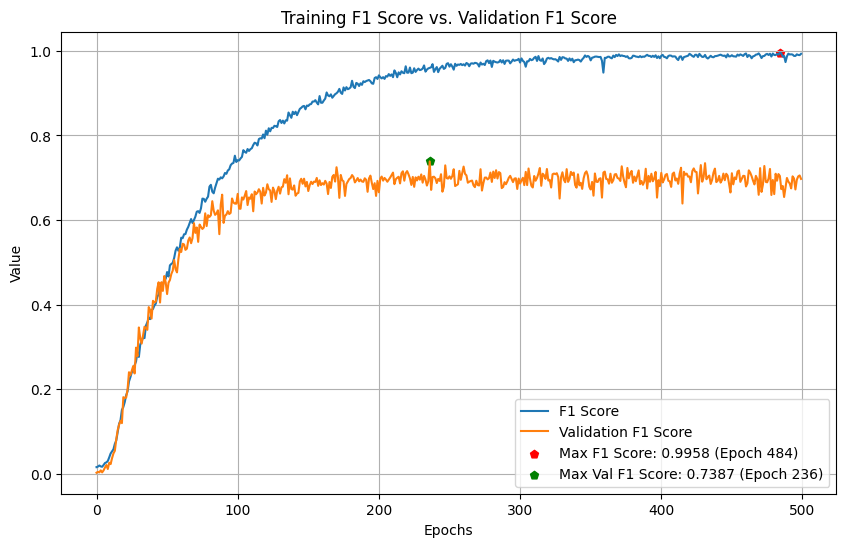
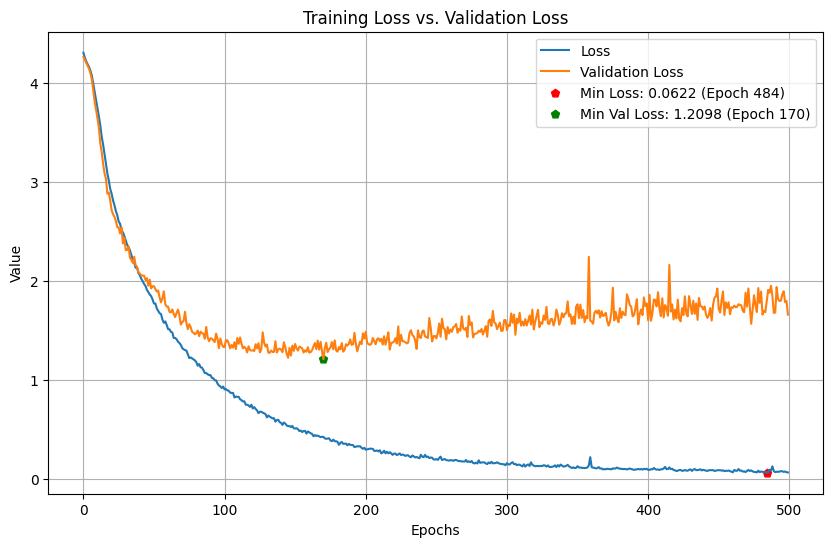


Figure 4.18 2 1D-CNN + 1 GRU Trial 698 Training History

* 1. Real Time Implementation

With the objective of testing the models in a real time environment, five videos per Gloss were recorded giving a total of 250, while these videos were recorded, intentionally they were recorded varying the distance of the subject from the camera, illumination, angles and the dominant hand involved. While no additional techniques for data augmentation were treated in the training process of the models other than flipping the frame before feeding the HPE algorithm, the ASL citizen dataset itself is carrying most of the task as the only subject of the 52 participants who were in a controlled environment was the seed signer, the rest of contributors submitted their versions which translates in different environment conditions.

Important to mention is that the researcher is not a signer, so when recording, it was systematically done by watching the seed signer video corresponding to the Gloss and recording afterwards, attempting to do the sign as accurately as possible. The videos recorded were collected in MP4 format, 130 frames at 24 FPS. These were stored in a folder per Gloss to facilitate later the dataframe creation. After the videos were collected, they were processed with PySpark in a dataframe containing the paths and the rest of the information contained in the others was created.



Table 4.18 Custom Sign Videos Dataframe Sample

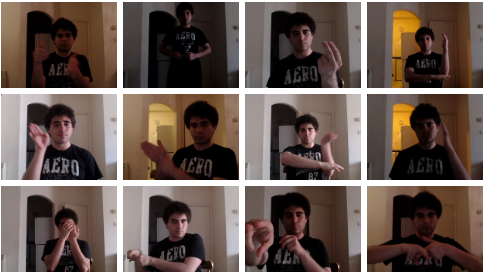


Figure 4.19 Custom Sign Videos Sample

Once the dataframe was created, the keypoints with each of the HPE algorithms were collected as previously done in a NumPy array to do a quick evaluation of these unseen videos using the best two models for each HPE algorithm resulting in the best algorithm fed with MediaPipe the one that could generalise better the unseen data, below in figure 4.20 a comparison of the performance with this model on the validation and custom test data is shown, the comparison for other five models can be found in the Appendix section.

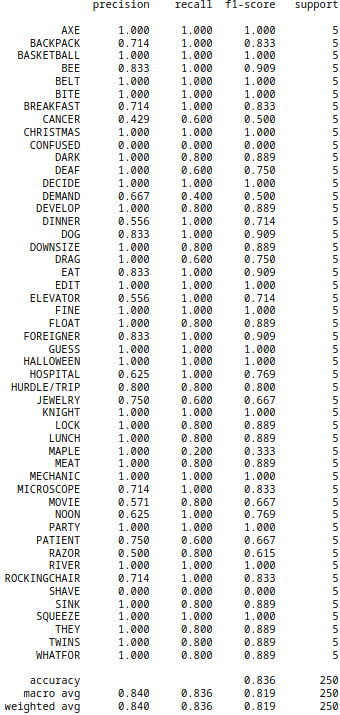
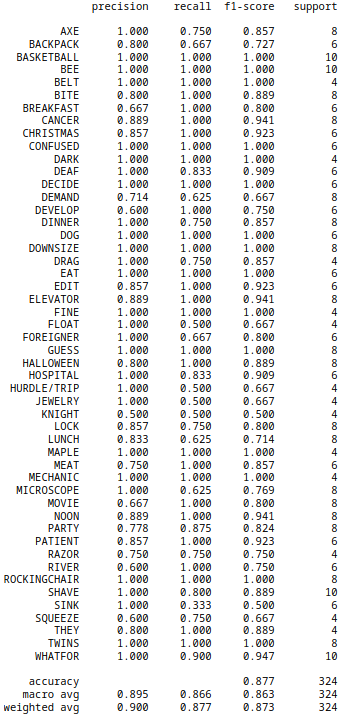


Figure 4.20 MediaPipe 2 1D-CNN + 1 GRU Layers Trial 721 Model Performance Comparison between Validation and Custom Test Data

The model although it does not have a performance as accurate as in the validation data, the model still has good performance with completely unseen data, also, the subject involved is not a signer, possibly adding more noise to the test data. Surprisingly, the model could not classify correctly some of the videos that showed no problems in the validation data, like Maple, where only one out of the five videos was correctly predicted in the custom data. In contrast, for the validation data, it accurately predicted each instance of Maple, or Shave which could not classify any of the videos correctly. On the other side of the coin, there are some of them, like Knight, which classified correctly the five videos in the test set, while in the validation dataset, it could only predict two of the four videos.

An interface was also developed to either predict feed from the camera directly or from a video file for the model to be able to predict accurately, as a scaler was previously used, this same scaler fitted with the training data was saved and is always loaded to scale any input before feeding the neural network to predict. The real time prediction logic works in summary as below:

1. Initialize a NumPy array with zeroes to store each frame keypoints.
2. Initialize an array to store the predictions made.
3. Start capturing the video and process the frame through the HPE algorithm.
4. Inverse the scaling in the data (this is because each time the array is fed to the Neural Network, it needs to be scaled).
5. Append the recent frame keypoints to the NumPy array and delete the last one (ensuring we always feed the algorithm with 130 frames).
6. Scale the NumPy array.
7. Check if a half second has passed, and feed the algorithm to predict. If not, show the frame.
8. Extract the prediction and if it has passed the threshold set to 0.8, but can be adjusted, append it to the predictions array and display.
9. Calculate the frame inference time and display it.
10. Continue the loop until it is stopped.



Figure 4.21 Real Time Detection Interface with MediaPipe



Figure 4.22 Real Time Detection Interface with RTMPose / CSPNext

This interface includes the last prediction made by the algorithm and the Inference time. The name of the Gloss is shown in Figures 4.21 and Figure 4.22 as is included in 5 demo videos that were prepared to take randomly ten videos of the custom sign videos dataset and concatenate them together to constantly feed the algorithm simulating in this way as closely as possible the inference in real time. To evaluate the inference speed of the whole process since it captures the first frame to the moment it makes a prediction, some tests were made with the same five videos randomly generated putting down the average inference speed with different complexities of the MediaPipe HPE algorithm and also with three and four layers neural networks.

| **Trial** | **Mean Inference time (s)**  **complexity = 0** | **Mean Inference time (s) complexity = 1** | **Mean Inference time (s) complexity = 2** |
| --- | --- | --- | --- |
| 1 | 0.08 | 0.09 | 0.20 |
| 2 | 0.08 | 0.09 | 0.19 |
| 3 | 0.08 | 0.08 | 0.19 |
| 4 | 0.08 | 0.08 | 0.20 |
| 5 | 0.08 | 0.09 | 0.19 |

Table 4.19 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU + GPU computer

| **Trial** | **Average Inference time (s)**  **complexity = 0** | **Average Inference time (s) complexity = 1** | **Average Inference time (s) complexity = 2** |
| --- | --- | --- | --- |
| 1 | 0.09 | 0.09 | 0.19 |
| 2 | 0.09 | 0.09 | 0.20 |
| 3 | 0.08 | 0.09 | 0.20 |
| 4 | 0.09 | 0.09 | 0.20 |
| 5 | 0.08 | 0.09 | 0.21 |

Table 4.20 MediaPipe Inference Time 1 1D-CNN + 1 GRU Layers Model in CPU + GPU computer

Predicting every half second rather than every captured frame leads to a faster inference speed by up to half the time. The change from one frame to the next one may not be representative for the algorithm that is looking for the sequence of change in joint positions along the time space. Also, any information is lost as the frame positions are stored the whole time, it is just the frequency that the algorithm is fed. This difference in inference speed can be clearly noted in the graphs below that capture the inference time along a video of five seconds duration while predicting every frame keeps the inference time constantly above .1 seconds, reaching peaks above .2, predicting each half second helps to keep this inference time below .1 with rises above this mark every time the Neural Network does the classification, but never reaching .2 seconds.

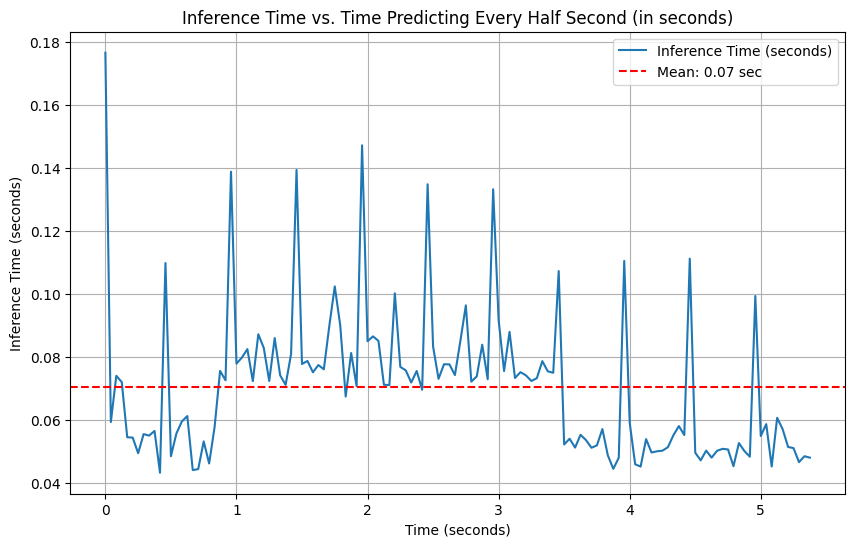


Figure 4.23 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU + GPU computer

The same tests were performed in a computer without a GPU equipped with an Intel Core i5-1135G7 CPU 4 cores @ 2.40GHz, and 8 GB DDR4 RAM running Windows 11, and in general, this hardware was able to handle faster inference speeds than the previous one, which is not surprising given that it is a faster CPU processor and MediaPipe runs directly on the CPU. MediaPipe is inferring all the time contrary to the Neural Network, which only infers twice per second, and this is why, with the models and logic developed in this project, having a faster CPU than a GPU becomes more critical.

| **Trial** | **Mean Inference time (s)**  **complexity = 0** | **Mean Inference time (s) complexity = 1** | **Mean Inference time (s) complexity = 2** |
| --- | --- | --- | --- |
| 1 | 0.03 | 0.03 | 0.07 |
| 2 | 0.03 | 0.03 | 0.07 |
| 3 | 0.04 | 0.04 | 0.07 |
| 4 | 0.03 | 0.04 | 0.07 |
| 5 | 0.04 | 0.04 | 0.07 |

Table 4.21 MediaPipe Inference Time 2 1D-CNN + 1 GRU Layers Trial 721 Model in CPU computer

| **Trial** | **Mean Inference time (s)**  **complexity = 0** | **Mean Inference time (s) complexity = 1** | **Mean Inference time (s) complexity = 2** |
| --- | --- | --- | --- |
| 1 | 0.03 | 0.03 | 0.07 |
| 2 | 0.03 | 0.03 | 0.07 |
| 3 | 0.04 | 0.04 | 0.07 |
| 4 | 0.03 | 0.04 | 0.07 |
| 5 | 0.04 | 0.04 | 0.08 |

Table 4.22 MediaPipe Inference Time 1 1D-CNN + 1 GRU Layers Model in CPU computer

Furthermore, these tests were performed on the RTMPose and CSPNext algorithms, which revealed that the inference speed is slower than MediaPipe even when these two algorithms are both running directly in the GPU. While the inference speed of both algorithms is close to the heaviest MediaPipe model in the hardware equipped with GPU, the decrease in the CPU only computer is dramatic, showing drops of up to 3.5 times between running the models in a CPU and GPU. Also, the same situation with the inference time decrease was observed in these two algorithms when predicting every half second instead of predicting every frame.

| **Model** | **2 1D-CNN + 1 GRU** | | **1 1D-CNN + 1 GRU** | |
| --- | --- | --- | --- | --- |
| **Trial** | **Mean Inference time (s)**  **RTMPose** | **Mean Inference time (s) CSPNext** | **Mean Inference time (s)**  **RTMPose** | **Mean Inference time (s) CSPNext** |
| 1 | 0.13 | 0.13 | 0.14 | 0.13 |
| 2 | 0.13 | 0.12 | 0.14 | 0.11 |
| 3 | 0.13 | 0.12 | 0.14 | 0.13 |
| 4 | 0.14 | 0.12 | 0.14 | 0.13 |
| 5 | 0.14 | 0.12 | 0.15 | 0.13 |

Table 4.23 MMPose Framework Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU+GPU computer

| **Model** | **2 1D-CNN + 1 GRU** | | **1 1D-CNN + 1 GRU** | |
| --- | --- | --- | --- | --- |
| **Trial** | **Mean Inference time (s)**  **RTMPose** | **Mean Inference time (s) CSPNext** | **Mean Inference time (s)**  **RTMPose** | **Mean Inference time (s) CSPNext** |
| 1 | 0.44 | 0.32 | 0.43 | 0.29 |
| 2 | 0.45 | 0.30 | 0.43 | 0.29 |
| 3 | 0.44 | 0.29 | 0.49 | 0.29 |
| 4 | 0.46 | 0.30 | 0.46 | 0.29 |
| 5 | 0.44 | 0.30 | 0.45 | 0.29 |

Table 4.24 MMPose Framework Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU computer

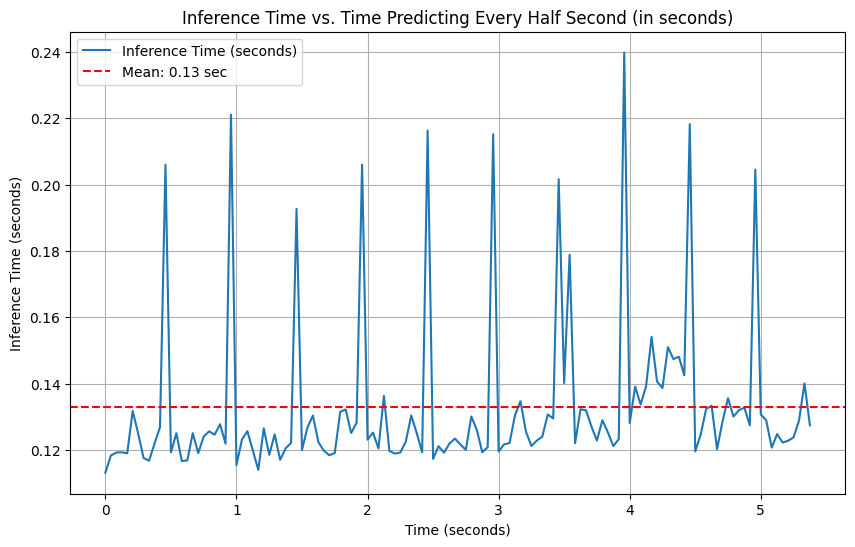


Figure 4.24 RTMPose Inference Time 2 1D-CNN + 1 GRU Layers Model in CPU + GPU computer

# CONCLUSIONS AND FUTURE WORK

This piece of work began with a comprehensive literature review involving different topics related to Sign Language Recognition which concluded with the decision to evaluate combinations of CNN, GRU, and LSTM layers with MediaPipe, RTMPose, and CSPNext HPE algorithms to complete this task.

Once this was determined, a methodology was developed to achieve the main objective of designing this system, as it was identified that the artefact's final users are a vulnerable group, raising potential ethical issues, the decision was made to use a publicly available dataset instead of directly involving deaf or hard of hearing people, converting the thesis into a theoretical piece of work that provides a basis for a real life implementation, it cannot be directly implemented without further evaluation and research as it is essential to involve this people to implement a system of this nature in real life, also, there is no intention in this project to replace human interpreters, as it was previously stated AI interpretations, at this stage, cannot substitute human interpreters.

The primary research was conducted through experimentation using Stratified Sampling identifying 30 subgroups composed of NN and HPE algorithm combinations, posterior to this, random sampling combined with successive halving is performed by Keras Tuner Hyperband algorithm in each combination to find the highest F1 Score in the validation data. The data used to train and validate the data is a NumPy array including the human joints coordinates extracted from the fifty glosses/words with more samples available from the recently released “Microsoft ASL Citizen” dataset, which is a rich dataset including 52 signers in their own environment. This rich dataset was key for this research success as training the NN with recordings captured from different cameras, with a variety of backgrounds, lightning conditions, video lengths, etcetera, gave the algorithm robustness, that even when the NN does not directly handle these situations and it falls more in the HPE side, it affects the quality of the keypoints estimation indirectly affecting the classification accuracy. Flipping the frames before the keypoint extraction was a measure considered as data augmentation to help reduce overfitting and also to have a more robust algorithm that can detect signs no matter the hand this one is performed with.

After tuning each of the 30 different models, each of these was further trained up to 500 epochs saving the weights that achieved the best validation F1 Score. It was found that, with a few exceptions, most of the models achieved higher scores after further training. In addition, the ten best hyperparameter combinations for each of these models were captured in tables that can be referred to in the Appendix section to identify potential combinations. After this exercise, two of these models outperformed the previous ones in MediaPipe, and one model outperformed the second best performer for the CSPNext algorithm. The model with the highest validation F1 Score (86.3%) was the 2 1D-CNN + 1 GRU Trial 721 model using MediaPipe as the HPE algorithm.

Following that, because the test data provided in the Microsoft ASL Citizen dataset had already been used for training purposes, 250 videos were recorded following the seed signer to validate the model's performance replicating the sign as accurately as possible showing a performance not as good as in the validation data but still getting an 81.9% F1 Score, an important consideration on this is that the researcher is not a signer, potentially adding errors in the videos used. To finalize, an evaluation of the inference speed was carried out to check the usability of the system in real time showing how the selected model can infer up to 25-30 FPS in a CPU only equipment, in this case, the CPU only hardware showed better inference speeds as most of this inference time is spent by MediaPipe because the NN is inferring just twice per second and MediaPipe is running on the CPU, so having hardware with a faster CPU becomes more critical than having a GPU available. However, is not the same case for the MMPose based HPE algorithms that run on the GPU, these models when implemented in the CPU only hardware presented speed decreases of up to 3.5 times between running the models in a CPU and a GPU.

Overall, the models that had the best performance are composed of four hidden layers, particularly the ones combining 2 1D-CNN layers, 1 GRU layer, and the fully connected layer. During the project development, it was identified that studies proved that, while four hidden layers can lead to generalising better, the tradeoff between gained accuracy and time is not worth it against having three layers only, but it was not the case in this project where there was a gain in accuracy and the inference time is not heavily affected. Again, one of the reasons why this phenomenon is not present is because the NN is just inferring twice per second.

Even when one of the models was able to generalise unseen data, overfitting is present in all of them and is just less severe in some of them, but it was not possible to completely dissipate it. Regularization techniques like weight decay and dropout were applied as well as data augmentation, but none of these completely worked out, suggesting that more data would be required to achieve better results, however, not necessarily more data would be needed, instead, there are some points while handling the data that might be done differently to reduce the complexity of the models.

In first place, recapitulating sections 3.4 and 3.5, it was found that the videos have different lengths and were recorded at different frame rates resulting in a wide range of frames per video from 10 to 540. Because a Neural Network needs to be fed with same length inputs, the strategy was to drop the videos above 130 frames and apply prepadding to the videos with less than 130 frames standardising the data in this way. This approach allowed for a reduction in the complexity of the data ending with inputs of 130 x 126/399 instead of 540 x 126/399, resulting in a reduction in input layer complexity. However, when the real time recordings were done, it was discovered that 130 frames is not necessary for most of the signs, to illustrate, if 130 frames were recorded at a 30 fps rate it means 4 seconds length approximate, and in these 4 seconds, a large portion of the video is just the person without performing any sign, implying that unnecessary information is being fed to the NN, and also, when inferring, in 4 seconds it is possible that the algorithm is being fed with two signs information, because of this, an approach based on fewer frames per video surely would decrease complexity and help in the overfitting reduction. An alternative to the approach used in this study could be the one proposed directly by Microsoft in the dataset paper, in which the videos were standardised to 64 frames, skipping frames for longer videos and padding shorter videos, with this approach, it is possible to use the entire dataset and reduce complexity even further.

Proof that reducing complexity would work is in the MMPose framework fed algorithms where it was decided to take into consideration face and pose information as there were not as many points as in MediaPipe 1662 vs. 399 MMPose which ended being 126 vs 399 after not considering the rest of information in MediaPipe. What was observed is that the MediaPipe fed algorithms were able to generalise the data better than the other two, ranging from 77% to 86% validation F1 score, while the others fell in a wide range starting from 44% and any of these could get to the 80s.

Causality was not present in this project as each of the combinations presented ended up having many different variables, meaning non-spurious association was not possible to demonstrate however, while there are some drawbacks in the presented model, there is evidence of a path that could be followed with further research to improve it and potentially implement Deep Learning for real life situations in the near future, the model was able to predict most of the words in different conditions with unseen data correctly. Also, the inference speed achieved is between 25-30 FPS using inexpensive day to day hardware not even requiring a GPU to achieve this performance.

To finalize, from this research, below are the main points that should be considered in future research to shorten the existing communication gap using Deep Learning:

* Reduce data complexity as much as possible to reduce overfitting probabilities.
* Validate the results with the actual Sign Language users.
* Keep privacy in mind to avoid ethical issues, and this can be easily addressed following this path, as even if data is directly collected from deaf and hard of hearing people, what is needed to train the algorithm are the keypoint change sequences, meaning, numbers stored as a NumPy array are needed, no videos or images have to be stored.

REFERENCES

APPENDIX A. TRIALS SUMMARY MEDIAPIPE

1 LSTM Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 427 | 76.23% | 1.4726 | 94.47% | 0.2527 |
| 426 | 76.05% | 1.6431 | 96.99% | 0.2331 |
| 671 | 75.30% | 1.5751 | 99.02% | 0.1502 |
| 598 | 74.29% | 1.8760 | 97.61% | 0.1290 |
| 670 | 74.24% | 1.6395 | 97.30% | 0.1812 |
| 701 | 73.72% | 1.9176 | 95.61% | 0.2890 |
| 422 | 73.53% | 1.6120 | 91.62% | 0.3946 |
| 425 | 73.43% | 1.5491 | 88.03% | 0.4543 |
| 599 | 73.38% | 1.6251 | 96.41% | 0.2109 |
| 668 | 72.67% | 1.4932 | 94.40% | 0.2685 |

2 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 426 | 80.02% | 1.3646 | 98.29% | 0.1679 |
| 724 | 78.64% | 1.6006 | 97.31% | 0.2052 |
| 714 | 78.63% | 1.3749 | 99.62% | 0.1376 |
| 715 | 77.88% | 1.4511 | 92.94% | 0.4241 |
| 708 | 77.57% | 1.3461 | 98.61% | 0.1664 |
| 670 | 77.53% | 1.5035 | 93.18% | 0.3852 |
| 598 | 77.22% | 1.4168 | 98.16% | 0.1077 |
| 422 | 76.73% | 1.3174 | 95.13% | 0.2727 |
| 427 | 76.70% | 1.6322 | 98.39% | 0.1896 |
| 593 | 75.87% | 1.2397 | 93.33% | 0.2553 |

3 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 599 | 80.78% | 1.2805 | 96.02% | 0.3136 |
| 598 | 80.17% | 1.3462 | 97.55% | 0.2371 |
| 670 | 80.05% | 1.3289 | 97.50% | 0.3046 |
| 671 | 79.25% | 1.4106 | 99.80% | 0.2249 |
| 664 | 78.46% | 1.3822 | 94.56% | 0.4111 |
| 426 | 78.28% | 1.4909 | 95.76% | 0.2616 |
| 665 | 77.86% | 1.3605 | 97.10% | 0.3618 |
| 593 | 77.75% | 1.2499 | 93.71% | 0.3854 |
| 701 | 77.58% | 1.6472 | 95.11% | 0.3409 |
| 427 | 77.55% | 1.3995 | 98.95% | 0.1685 |

1 1D-CNN Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 74.64% | 1.4745 | 71.49% | 0.9494 |
| 713 | 72.43% | 1.4363 | 71.40% | 0.9529 |
| 715 | 72.30% | 1.2529 | 77.82% | 0.7167 |
| 598 | 72.26% | 1.9606 | 77.97% | 0.8768 |
| 597 | 71.55% | 1.8177 | 72.64% | 0.9967 |
| 599 | 71.38% | 1.7168 | 93.38% | 0.2065 |
| 700 | 70.92% | 1.6697 | 73.79% | 0.9016 |
| 594 | 70.80% | 1.6597 | 88.93% | 0.3494 |
| 716 | 70.06% | 1.3670 | 82.98% | 0.5322 |
| 701 | 69.82% | 1.8321 | 83.40% | 0.5436 |

2 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 670 | 82.98% | 1.0391 | 82.55% | 0.5493 |
| 715 | 78.57% | 1.3614 | 92.02% | 0.2292 |
| 426 | 78.18% | 1.6864 | 97.93% | 0.0657 |
| 598 | 77.97% | 1.3790 | 93.59% | 0.1895 |
| 594 | 77.69% | 1.3843 | 90.80% | 0.2776 |
| 666 | 77.42% | 1.0564 | 73.93% | 0.7868 |
| 427 | 76.67% | 3.0072 | 91.58% | 0.4114 |
| 599 | 76.60% | 1.6545 | 98.07% | 0.0679 |
| 700 | 76.48% | 1.1857 | 71.56% | 0.9038 |
| 664 | 76.43% | 1.3846 | 84.76% | 0.4878 |

3 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 598 | 82.27% | 1.2285 | 93.65% | 0.1991 |
| 427 | 80.89% | 1.4881 | 94.34% | 0.1903 |
| 599 | 80.74% | 1.4952 | 95.18% | 0.1521 |
| 593 | 80.30% | 1.0723 | 90.57% | 0.2995 |
| 664 | 80.10% | 1.0440 | 83.96% | 0.5008 |
| 715 | 79.98% | 1.2299 | 85.89% | 0.4373 |
| 426 | 79.62% | 1.4685 | 94.89% | 0.1729 |
| 594 | 79.57% | 1.1810 | 91.88% | 0.2533 |
| 718 | 79.45% | 1.0102 | 77.47% | 0.7096 |
| 597 | 79.42% | 1.2837 | 89.20% | 0.3296 |

1 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 722 | 83.17% | 0.8498 | 91.50% | 0.3166 |
| 670 | 81.04% | 1.1566 | 91.27% | 0.3742 |
| 598 | 80.87% | 1.1411 | 99.47% | 0.1014 |
| 701 | 80.78% | 1.2853 | 89.33% | 0.5841 |
| 426 | 80.72% | 1.3163 | 99.12% | 0.0790 |
| 671 | 80.15% | 1.2259 | 97.30% | 0.1875 |
| 665 | 80.15% | 1.0597 | 87.38% | 0.4970 |
| 664 | 80.03% | 1.1673 | 93.93% | 0.3010 |
| 595 | 80.01% | 1.1122 | 98.40% | 0.1576 |
| 700 | 79.99% | 1.1793 | 91.60% | 0.3440 |

2 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 82.30% | 0.9308 | 99.40% | 0.0601 |
| 721 | 82.28% | 1.0919 | 88.08% | 0.4878 |
| 598 | 82.15% | 1.1865 | 98.81% | 0.0861 |
| 670 | 81.64% | 1.2477 | 91.69% | 0.3901 |
| 599 | 81.63% | 1.2085 | 97.23% | 0.2115 |
| 427 | 81.32% | 1.1254 | 99.44% | 0.0713 |
| 594 | 80.59% | 1.1861 | 98.31% | 0.1214 |
| 595 | 80.57% | 1.2379 | 96.05% | 0.2626 |
| 706 | 80.22% | 0.9205 | 99.40% | 0.0792 |
| 700 | 80.19% | 1.2403 | 85.81% | 0.7634 |

1 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 700 | 83.78% | 0.9429 | 95.33% | 0.1929 |
| 694 | 82.92% | 0.8163 | 88.21% | 0.4149 |
| 714 | 82.30% | 0.9762 | 93.90% | 0.2443 |
| 426 | 81.73% | 1.1918 | 99.64% | 0.0722 |
| 670 | 81.09% | 1.2809 | 97.67% | 0.1605 |
| 423 | 80.72% | 1.1410 | 98.75% | 0.0992 |
| 671 | 80.57% | 1.0712 | 92.33% | 0.3890 |
| 598 | 80.41% | 1.1399 | 97.22% | 0.2074 |
| 722 | 80.39% | 0.8521 | 82.31% | 0.6514 |
| 593 | 80.37% | 1.1583 | 94.97% | 0.2966 |

2 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 593 | 82.77% | 0.9535 | 96.55% | 0.1907 |
| 599 | 82.57% | 1.0351 | 96.61% | 0.2603 |
| 706 | 82.55% | 0.8075 | 98.04% | 0.1226 |
| 598 | 82.34% | 0.9435 | 97.12% | 0.1710 |
| 714 | 82.32% | 0.8595 | 99.46% | 0.0671 |
| 427 | 81.56% | 1.0077 | 98.61% | 0.1536 |
| 594 | 81.51% | 1.1163 | 96.34% | 0.2726 |
| 670 | 81.44% | 1.0003 | 97.74% | 0.1240 |
| 423 | 81.07% | 1.2748 | 91.38% | 0.4625 |
| 426 | 80.83% | 1.3837 | 93.30% | 0.4173 |

APPENDIX B. TRIALS SUMMARY RTMPOSE

1 LSTM Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 427 | 58.07% | 2.2090 | 92.37% | 0.2973 |
| 700 | 56.70% | 2.0271 | 89.95% | 0.4034 |
| 598 | 56.41% | 2.6100 | 98.05% | 0.1797 |
| 670 | 53.88% | 2.3122 | 88.93% | 0.4303 |
| 701 | 52.91% | 2.4122 | 85.50% | 0.5138 |
| 426 | 49.51% | 2.8727 | 93.84% | 0.7640 |
| 671 | 49.40% | 2.5169 | 87.39% | 0.4978 |
| 593 | 48.35% | 2.0380 | 73.53% | 0.9249 |
| 693 | 47.97% | 1.9221 | 67.09% | 1.0658 |
| 422 | 47.37% | 2.6059 | 84.30% | 1.0828 |

2 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 670 | 65.83% | 2.3422 | 99.64% | 0.3206 |
| 724 | 65.03% | 1.8988 | 90.96% | 0.3908 |
| 671 | 65.02% | 1.8851 | 98.43% | 0.2665 |
| 599 | 64.92% | 2.3053 | 99.46% | 0.2175 |
| 598 | 64.32% | 2.0088 | 96.70% | 0.2182 |
| 700 | 63.88% | 2.0156 | 90.52% | 0.4781 |
| 426 | 62.93% | 1.9776 | 95.39% | 0.2932 |
| 714 | 62.22% | 2.0254 | 98.87% | 0.1670 |
| 693 | 61.35% | 1.6966 | 75.19% | 0.9378 |
| 715 | 61.29% | 1.9195 | 83.73% | 0.6369 |

3 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 67.85% | 1.8173 | 94.27% | 0.4106 |
| 598 | 67.63% | 1.7513 | 91.29% | 0.5171 |
| 715 | 67.10% | 2.0108 | 97.33% | 0.2760 |
| 720 | 66.88% | 2.0948 | 98.47% | 0.2556 |
| 716 | 66.83% | 1.8276 | 96.00% | 0.2450 |
| 426 | 66.57% | 2.0229 | 97.93% | 0.5747 |
| 700 | 66.30% | 1.8700 | 97.38% | 0.2493 |
| 702 | 65.31% | 1.8210 | 90.82% | 0.4178 |
| 670 | 64.60% | 2.0320 | 93.13% | 0.4958 |
| 722 | 64.50% | 2.4959 | 99.29% | 0.7031 |

1 1D-CNN Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 670 | 53.34% | 2.1390 | 79.06% | 0.6157 |
| 598 | 49.33% | 1.9740 | 49.79% | 1.5673 |
| 599 | 47.88% | 2.8703 | 98.45% | 0.0751 |
| 593 | 46.41% | 2.0345 | 45.78% | 1.7036 |
| 595 | 44.76% | 2.6315 | 92.69% | 0.2862 |
| 718 | 43.41% | 2.6100 | 77.21% | 0.7038 |
| 426 | 43.40% | 2.2178 | 58.45% | 1.3466 |
| 723 | 42.83% | 2.3698 | 87.00% | 0.4970 |
| 664 | 42.21% | 2.3054 | 56.20% | 1.3943 |
| 671 | 41.75% | 2.3458 | 62.86% | 1.2200 |

2 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 69.54% | 1.2192 | 71.29% | 0.9039 |
| 723 | 67.25% | 1.3849 | 91.54% | 0.2517 |
| 598 | 66.08% | 1.6426 | 86.27% | 0.4106 |
| 671 | 65.21% | 1.6458 | 91.05% | 0.2771 |
| 599 | 63.06% | 1.5364 | 65.63% | 1.0526 |
| 593 | 62.46% | 1.6252 | 81.55% | 0.5456 |
| 701 | 61.59% | 1.8102 | 84.32% | 0.4618 |
| 670 | 60.21% | 1.4481 | 92.55% | 0.2692 |
| 715 | 58.29% | 2.1663 | 94.36% | 0.1929 |
| 722 | 57.42% | 1.6195 | 89.96% | 0.3855 |

3 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 65.68% | 1.3736 | 59.27% | 1.2365 |
| 671 | 65.33% | 1.4575 | 81.48% | 0.5479 |
| 718 | 64.97% | 1.4593 | 60.86% | 1.2088 |
| 670 | 64.09% | 1.5704 | 68.94% | 0.9295 |
| 598 | 64.05% | 1.6292 | 81.79% | 0.5574 |
| 702 | 61.99% | 1.3944 | 76.70% | 0.7153 |
| 700 | 60.87% | 1.7215 | 91.60% | 0.2716 |
| 715 | 60.74% | 1.8083 | 84.18% | 0.4962 |
| 724 | 60.45% | 1.5068 | 78.22% | 0.6632 |
| 716 | 60.33% | 1.4581 | 74.67% | 0.7782 |

1 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 75.56% | 1.1832 | 92.86% | 0.3378 |
| 598 | 71.11% | 1.2589 | 79.01% | 0.7226 |
| 593 | 69.95% | 1.3452 | 75.52% | 0.8362 |
| 722 | 69.34% | 1.2266 | 66.09% | 1.0993 |
| 599 | 68.32% | 1.2195 | 76.59% | 0.7525 |
| 427 | 68.08% | 1.4681 | 95.51% | 0.2883 |
| 671 | 68.06% | 1.3289 | 80.05% | 0.7098 |
| 595 | 67.91% | 1.2180 | 83.95% | 0.7305 |
| 664 | 66.91% | 1.3396 | 75.25% | 0.8397 |
| 717 | 66.48% | 1.3561 | 77.61% | 0.8060 |

2 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 719 | 74.21% | 1.0943 | 83.70% | 0.5548 |
| 715 | 73.99% | 1.4242 | 93.50% | 0.3012 |
| 714 | 73.19% | 1.2316 | 99.44% | 0.0642 |
| 700 | 72.89% | 1.3883 | 99.83% | 0.0626 |
| 671 | 72.69% | 1.4512 | 97.59% | 0.1746 |
| 670 | 72.33% | 1.2387 | 90.54% | 0.3859 |
| 665 | 70.96% | 1.2466 | 84.74% | 0.5735 |
| 693 | 70.74% | 1.3113 | 99.19% | 0.1940 |
| 664 | 70.52% | 1.4066 | 93.84% | 0.3000 |
| 706 | 69.94% | 1.2613 | 98.07% | 0.1614 |

1 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 69.22% | 1.2940 | 86.21% | 0.5737 |
| 426 | 68.46% | 1.2849 | 89.62% | 0.3780 |
| 670 | 66.54% | 1.3541 | 87.00% | 0.4981 |
| 717 | 65.02% | 1.4289 | 69.70% | 1.1101 |
| 664 | 64.60% | 1.3835 | 77.95% | 0.7569 |
| 427 | 63.81% | 1.6323 | 99.69% | 0.0790 |
| 425 | 63.00% | 1.3551 | 78.77% | 0.7382 |
| 700 | 63.00% | 1.6576 | 87.26% | 0.4783 |
| 422 | 62.91% | 1.6154 | 99.39% | 0.0914 |
| 711 | 62.74% | 1.5288 | 71.43% | 1.0694 |

2 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 427 | 74.92% | 1.3330 | 92.63% | 0.3423 |
| 426 | 74.70% | 1.2958 | 96.60% | 0.1961 |
| 425 | 72.16% | 1.2700 | 90.29% | 0.3986 |
| 670 | 72.08% | 1.4947 | 96.74% | 0.1534 |
| 715 | 72.02% | 1.2437 | 78.02% | 0.7161 |
| 700 | 71.28% | 1.6248 | 95.98% | 0.2071 |
| 424 | 70.60% | 1.2888 | 84.64% | 0.5892 |
| 665 | 69.85% | 1.3485 | 87.50% | 0.4538 |
| 717 | 69.36% | 1.5452 | 81.38% | 0.6076 |
| 693 | 68.48% | 1.6669 | 93.65% | 0.2728 |

APPENDIX C. TRIALS SUMMARY CSPNEXT

1 LSTM Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 671 | 60.36% | 1.9250 | 84.59% | 0.5152 |
| 426 | 58.96% | 1.8635 | 89.94% | 0.3659 |
| 598 | 57.37% | 1.7027 | 80.83% | 0.6489 |
| 670 | 55.81% | 2.0738 | 84.99% | 0.5282 |
| 422 | 54.92% | 1.6675 | 74.97% | 0.7701 |
| 593 | 54.91% | 1.7342 | 71.64% | 0.9309 |
| 599 | 54.06% | 2.1388 | 84.91% | 0.7300 |
| 700 | 53.64% | 1.8923 | 81.00% | 0.6443 |
| 701 | 51.77% | 2.2170 | 78.04% | 0.7148 |
| 594 | 50.37% | 2.0683 | 73.96% | 1.0666 |

2 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 665 | 65.21% | 2.0418 | 95.53% | 0.5626 |
| 695 | 63.84% | 1.9330 | 84.39% | 0.6500 |
| 593 | 63.65% | 2.3980 | 99.12% | 0.0727 |
| 659 | 62.53% | 1.8449 | 81.75% | 1.0534 |
| 422 | 62.26% | 1.9068 | 91.16% | 0.6235 |
| 710 | 61.70% | 1.7135 | 76.32% | 0.8452 |
| 594 | 61.33% | 2.0085 | 95.65% | 0.4075 |
| 696 | 61.01% | 1.7053 | 84.58% | 0.6617 |
| 719 | 60.98% | 1.8969 | 83.79% | 0.6442 |
| 588 | 60.53% | 2.0453 | 93.66% | 0.2466 |

3 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 599 | 69.10% | 1.9730 | 98.02% | 0.3087 |
| 598 | 68.96% | 1.8250 | 94.25% | 0.6013 |
| 427 | 68.31% | 1.8454 | 96.49% | 0.3503 |
| 593 | 68.18% | 1.6776 | 89.87% | 0.7776 |
| 715 | 68.15% | 1.7692 | 92.46% | 0.3416 |
| 671 | 67.50% | 2.1964 | 93.42% | 1.0861 |
| 426 | 66.98% | 1.9899 | 99.67% | 0.3341 |
| 720 | 66.14% | 1.9317 | 96.60% | 0.3233 |
| 422 | 64.94% | 1.9627 | 96.80% | 0.4868 |
| 700 | 64.70% | 1.8790 | 97.07% | 0.2656 |

1 1D-CNN Layer Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 670 | 44.10% | 2.2801 | 60.04% | 1.2857 |
| 671 | 43.14% | 2.1467 | 47.18% | 1.6854 |
| 599 | 42.85% | 3.7958 | 85.63% | 0.4353 |
| 714 | 42.72% | 2.5894 | 66.15% | 1.0595 |
| 427 | 42.13% | 2.7739 | 72.05% | 0.8461 |
| 422 | 41.66% | 2.6757 | 71.68% | 0.8590 |
| 723 | 41.63% | 3.4951 | 95.17% | 0.1771 |
| 667 | 41.37% | 2.2866 | 45.53% | 1.8139 |
| 669 | 40.71% | 2.1679 | 37.75% | 2.0278 |
| 594 | 40.43% | 3.6289 | 94.78% | 0.1693 |

2 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 62.57% | 1.370344281 | 74.57% | 0.818958 |
| 671 | 62.07% | 1.476919 | 85.40% | 0.443908 |
| 723 | 61.92% | 1.533741 | 88.69% | 0.348799 |
| 598 | 61.04% | 1.561541438 | 66.29% | 1.063895 |
| 701 | 59.44% | 1.767323434 | 94.45% | 0.198786 |
| 599 | 58.52% | 2.202119 | 81.61% | 0.557080 |
| 593 | 57.90% | 1.594255507 | 54.77% | 1.439982 |
| 670 | 56.77% | 2.371407 | 87.84% | 0.358847 |
| 664 | 55.62% | 2.221631 | 83.49% | 0.510939 |
| 426 | 54.91% | 2.390743494 | 85.80% | 0.412875 |

3 1D-CNN Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 700 | 66.42% | 1.4440 | 80.85% | 0.5876 |
| 671 | 66.38% | 1.4807 | 86.90% | 0.3900 |
| 714 | 64.41% | 1.2867 | 59.00% | 1.2146 |
| 715 | 64.13% | 1.3125 | 75.73% | 0.7530 |
| 670 | 61.38% | 1.7711 | 90.54% | 0.3007 |
| 718 | 61.26% | 1.4335 | 58.82% | 1.2501 |
| 693 | 59.99% | 1.4681 | 68.19% | 0.9542 |
| 701 | 59.90% | 1.7258 | 93.37% | 0.2162 |
| 716 | 58.92% | 1.7241 | 82.68% | 0.5171 |
| 724 | 58.85% | 1.5474 | 72.99% | 0.8483 |

1 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 714 | 72.47% | 1.1993 | 91.42% | 0.3800 |
| 700 | 69.89% | 1.2430 | 89.43% | 0.4341 |
| 693 | 68.01% | 1.2499 | 81.35% | 0.6568 |
| 717 | 67.45% | 1.3248 | 75.38% | 0.7744 |
| 715 | 66.44% | 1.3467 | 82.67% | 0.5827 |
| 722 | 66.15% | 1.2593 | 65.65% | 1.1391 |
| 670 | 65.51% | 1.3760 | 78.27% | 0.7786 |
| 598 | 65.05% | 1.2931 | 68.56% | 0.9819 |
| 426 | 64.33% | 1.5668 | 99.92% | 0.0503 |
| 423 | 63.67% | 1.5363 | 99.74% | 0.0701 |

2 1D-CNN + 1 GRU Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 705 | 76.22% | 1.0074 | 78.58% | 0.7147 |
| 700 | 75.26% | 1.1675 | 92.04% | 0.3720 |
| 584 | 74.13% | 1.1974 | 94.93% | 0.2087 |
| 656 | 72.57% | 1.5480 | 96.04% | 0.2122 |
| 579 | 70.57% | 1.1204 | 85.61% | 0.4948 |
| 650 | 70.34% | 1.4158 | 92.45% | 0.3405 |
| 686 | 70.14% | 1.4599 | 99.85% | 0.0648 |
| 698 | 70.06% | 1.2535 | 73.20% | 0.9557 |
| 657 | 69.43% | 1.4631 | 93.93% | 0.2744 |
| 701 | 68.23% | 1.4373 | 99.12% | 0.0857 |

1 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 717 | 68.83% | 1.3645 | 73.53% | 0.9164 |
| 714 | 67.40% | 1.4019 | 93.14% | 0.3281 |
| 670 | 66.32% | 1.4046 | 85.76% | 0.5254 |
| 426 | 65.21% | 1.3820 | 90.61% | 0.4332 |
| 598 | 64.17% | 1.4012 | 68.45% | 0.9462 |
| 664 | 63.90% | 1.3715 | 76.69% | 0.7873 |
| 595 | 61.90% | 1.4571 | 64.11% | 1.0984 |
| 599 | 60.97% | 1.5377 | 71.67% | 0.9539 |
| 594 | 60.89% | 1.4916 | 61.63% | 1.2638 |
| 715 | 60.67% | 1.4546 | 77.96% | 0.7426 |

2 1D-CNN + 1 LSTM Layers Model

| **Trial #** | **Best Val\_F1 Score** | **Val\_Loss** | **Training F1 Score** | **Training Loss** |
| --- | --- | --- | --- | --- |
| 670 | 71.71% | 1.4377 | 95.16% | 0.2205 |
| 666 | 71.18% | 1.3316 | 89.78% | 0.3774 |
| 426 | 69.82% | 1.2672 | 94.61% | 0.2829 |
| 427 | 69.35% | 1.2668 | 93.61% | 0.3056 |
| 423 | 69.07% | 1.2334 | 89.28% | 0.4258 |
| 598 | 67.82% | 1.6566 | 93.23% | 0.2725 |
| 714 | 67.38% | 1.3111 | 86.40% | 0.4599 |
| 424 | 67.35% | 1.2432 | 92.04% | 0.3769 |
| 700 | 67.28% | 1.7021 | 92.15% | 0.3752 |
| 593 | 67.07% | 1.3910 | 80.89% | 0.6571 |

APPENDIX D. BEST MODELS PERFORMANCE COMPARISON BETWEEN VALIDATION AND CUSTOM TEST DATA

MediaPipe 2 1D-CNN + 1 GRU Layers Trial 721

|  |  |
| --- | --- |
| Validation Data | Custom Data |

MediaPipe 2 1D-CNN + 1 LSTM

|  |  |
| --- | --- |
| Validation Data | Custom Data |

RTMPose 2 1D-CNN + 1 GRU

|  |  |
| --- | --- |
| Validation Data | Custom Data |

RTMPose 2 1D-CNN + 1 LSTM

|  |  |
| --- | --- |
| Validation Data | Custom Data |

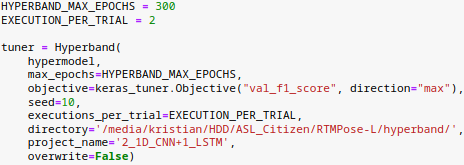
CSPNext 2 1D-CNN + 1 GRU

|  |  |
| --- | --- |
| Validation Data | Custom Data |

CSPNext 2 1D-CNN + 1 GRU Trial 698

|  |  |
| --- | --- |
| Validation Data | Custom Data |

APPENDIX E. KERAS TUNER HYPERBAND ALGORITHM CODE CONSTRUCTION



1. Training Loss between Table 4.4 and Table 4.5 differ as Table 4.4 is the checkpoint saved by Keras Tuner, and Table 4.5 is the result of model evaluation, even when the batch size was adjusted to be the same size, results differ from each other. [↑](#footnote-ref-2)
2. Training metrics between Table 4.10 and Table 4.11 differ as Table 4.10 is the checkpoint saved by Keras Tuner, and Table 4.11 is the result of model evaluation, even when the batch size was adjusted to be the same size, results differ from each other. [↑](#footnote-ref-3)
3. Training metrics between Table 4.15 and Table 4.16 differ as Table 4.15 is the checkpoint saved by Keras Tuner, and Table 4.16 is the result of model evaluation, even when the batch size was adjusted to be the same size, results differ from each other. [↑](#footnote-ref-4)