ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

FÁBIO BRANDOLT BALDISSERA

A LIGHT IMPLEMENTATION OF A 3D CONVOLUTIONAL NEURAL NETWORK FOR ONLINE GESTURE CLASSIFICATION

Porto Alegre
2019

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULTY OF ENGINEERING
GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

A LIGHT IMPLEMENTATION OF A 3D CONVOLUTIONAL NEURAL NETWORK FOR ONLINE GESTURE CLASSIFICATION

FÁBIO BRANDOLT BALDISSERA

Dissertation presented as partial requirement for obtaining the degree of Master in Electrical Engineering at Pontifícia Universidade Católica do Rio Grande do Sul.

Advisor: Prof. Dr. Fabian Luis Vargas

Porto Alegre
2019

Ficha Catalográfica

B177i Baldissera, Fábio Brandolt

Uma implementação leve de uma rede neural de convolução 3D para detecção online de gestos / Fábio Brandolt Baldissera . – 2019.

100.

Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica, PUCRS.

Orientador: Prof. Dr. Fabian Luis Vargas.

1. Reconhecimento de Gestos. 2. Classificação online. 3. 3DCNN. I. Vargas, Fabian Luis. II. Título.

Dedico este trabalho a meus pais, minha família e a todos meus amigos que me acompanham nessa jornada.

“The best way to predict the future is to invent it.”

(Alan Kay)

ACKNOWLEDGMENTS

Gostaria de agradecer primeiramente a minha mãe, que sempre me apoiou em todas as etapas dessa trajetória. Agradecer aos meus primos Vinícius, Suzana e Vitória por estar sempre presentes na minha vida, dispostos a me ajudar. E agradecer a todos meus amigos de Santa Maria e Porto Alegre que ajudaram a manter o bom humor durante toda a jornada.

Um agradecimento especial ao meu orientador, Fabian Vargas, por aceitar me orientar neste tema que era uma incognita para ambos de nós no início da jornada.

UMA IMPLEMENTAÇÃO LEVE DE UMA REDE NEURAL DE CONVOLUÇÃO 3D PARA DETECÇÃO ONLINE DE GESTOS

RESUMO

Com os avanços de técnicas de aprendizado de máquinas e o aumento da capacidade computacional disponível, redes neurais artificiais (ANNs) representam o estado-da-arte na tarefa de classificação de imagem, e mais recentemente na classificação de vídeos. A possibilidade do reconhecimento de gestos através de imagens de vídeo permite uma interface homem-máquina mais natural, maior imersão ao interagir com equipamentos de realidade virtual e pode até nos levar, em um futuro breve, à transcrição automática de linguagem de sinais. No entanto, as técnicas utilizadas para classificação de vídeo possuem um alto custo computacional, se tornando proibitivas para o uso em hardware mais simples. Esta dissertação busca estudar e analisar a aplicabilidade de técnicas de classificação de gestos contínua para sistemas embarcados. Este objetivo é atingido através da proposição de um modelo de rede neural baseado em redes de convolução 2D e 3D, capaz de realizar reconhecimento de gestos de forma online, isto é, gerando uma predição de classe para o vídeo concomitantemente com a obtenção dos quadros são obtidos, de uma forma preditiva, sem ter acesso a todos os quadros do vídeo. O modelo proposto foi testado em três diferentes bancos de dados de gestos presentes na literatura. Os resultados obtidos expandem o estado-da-arte por apresentar uma técnica de leve implementação que ainda apresenta uma acurácia alta suficiente para a aplicação em sistemas embarcados.

Palavras Chave: Reconhecimento de Gestos, Classificação online, 3DCNN.

A LIGHT IMPLEMENTATION OF A 3D CONVOLUTIONAL NEURAL NETWORK FOR ONLINE GESTURE CLASSIFICATION

ABSTRACT

With the advancement of machine learning techniques and the increased accessibility to computing power, Artificial Neural Networks (ANNs) have achieved state-of-the-art results in image classification and, most recently, in video classification. The possibility of gesture recognition from a video source enables a more natural non-contact human-machine interaction, immersion when interacting in virtual reality environments and can even lead to sign language translation in the near future. However, the techniques utilized in video classification are usually computationally expensive, being prohibitive to conventional hardware. This work aims to study and analyze the applicability of continuous online gesture recognition techniques for embedded systems. This goal is achieved by proposing a new model based on 2D and 3D CNNs able to perform online gesture recognition, i.e. yielding a label while the video frames are still being processed, in a predictive manner, before having access to future frames of the video. This technique is of paramount interest to applications in which the video is being acquired concomitantly to the classification process and the issuing of the labels has a strict deadline. The proposed model was tested against three representative gesture datasets found in the literature. The obtained results suggest the proposed technique improves the state-of-the-art by yielding a quick gesture recognition process while presenting a high accuracy, which is fundamental for the applicability of embedded systems.

Keywords: Gesture Recognition, Online Classification, 3DCNN.

List of Figures

Figure 1.1 – "We describe a system that can reconstruct the pose of the hand from a single image of the hand wearing a multi-colored glove. We demonstrate our system as a user-input device for desktop virtual reality applications." Source: [46] 15
Figure 2.1 – Basic structure of a neural network 20
Figure 2.2 – Block diagram of a basic NN neuron 20
Figure 2.3 – Some examples of activation functions and its graphs 21
Figure 2.4 – Examples of overfitting and under-fitting for a two-class classificator 25
Figure 2.5 – "Zoomed In" repeated structure of Inception Network 27
Figure 2.6 – Image matrix and 2D Kernel example 29
Figure 2.7 – Demonstration of a convolution between a 5x5 image and a 3x3 kernel 30
Figure 2.8 – Demonstration of application for some 3x3 kernels 31
Figure 2.9 – Structure of ResNet-18 32
Figure 2.10 – An unrolled recurrent neural network 33
Figure 2.11 – Representation of an LSTM cell, inside a LSTM network 34
Figure 2.12 – Different parts of a LSTM cell: a) forget gate; b) input gate; c) state path; d) output gate 34
Figure 2.13 – VideoLSTM network structure 37
Figure 2.14 – A 3D CNN architecture for human action recognition. 38
Figure 2.15 – A demonstration of OpenPose capabilities of tracking the human body. 39
Figure 2.16 – A demonstration of the optical frame (right) resulted from the two overlapped original frames (left). 40
Figure 2.17 – Two-stream architecture for video classification. 41
Figure 4.1 – Sample of a gesture from Jester dataset (12 out of 26 images shown, for this specific sample) 48

Figure 4.2 – Sample of a gesture from nvGesture dataset (handpicked frames for demonstration purposes) 50
Figure 4.3 – Sample of a gesture from EgoGesture dataset (handpicked frames for demonstration purposes) 51
Figure 4.4 – High-level block diagram for models RT_RGB-8F and RT_RGB-16F 52
Figure 4.5 – Representation of data shape throughout the 2D CNN in RT_RGB-8F 54
Figure 4.6 – Representation of data shape throughout the 3D CNN in RT_RGB-8F 55
Figure 4.7 – Representation of data shape throughout the 2D CNN in RT_RGB-16F 57
Figure 4.8 – Representation of data shape throughout the 3D CNN in RT_RGB-16F 58
Figure 4.9 – Representation of data shape throughout the 3D CNN in RT_RGB-16F 59
Figure 4.10 – Data Augmentation for the Jester Dataset. a) Orignal Image, b) "Randomly" cropped image and c) "Randomly" rotated Image 61
Figure 4.11 – Observation of accuracy and train loss during training of a model 63
Figure 4.12 – Predictions for a moving window of frames of a sample from the nvGesture dataset 66
Figure 4.13 – Predictions for a moving window of frames of a sample from the nvGesture dataset 66
Figure 4.14 – Averaged signal (pink) of the probability of a gesture (blue) surpassing the threshold (dotted line) and yield a label (vertical line) 67
Figure 4.15 – Flow diagram for the proposed approach 69
Figure 5.1 – Accuracy of online operation for the nvGesture dataset. Each line represents a different number of averaged frames 78
Figure 5.2 – Number of frames in which the classification occurred before the deadline of online operation for the nvGesture dataset. Each line represents a different number of averaged frames 79
Figure 5.3 – Online accuracy for the RT3D_8F model on the EgoGesture dataset. Each line represents a different number of averaged frames 80
Figure 5.4 – Online accuracy for the RT3D_16F model on the EgoGesture dataset. Each line represents a different number of averaged frames 81

Figure 5.5 – Online delay for the RT3D_8F model on the EgoGesture dataset. Results averaged for a given confidence threshold 81
Figure 5.6 – Online delay for the RT3D_16F model on the EgoGesture dataset. Results averaged for a given confidence threshold 82
Figure 5.7 – Number of computations per second performed on each model for different batch sizes. 83
Figure 5.8 – VRAM allocated during the tests for each model and batch size. 84

List of Tables

Table 2.1 – Related techniques 42
Table 4.1 – Classes in Jester dataset and its number of samples 47
Table 4.2 – Summary of the innovations proposed by our approach when compared to similar techniques 70
Table 5.1 – Scores obtained in Jester's validation set 71
Table 5.2 – Score on Jester's test set, as in the official leaderboard 72
Table 5.3 – Scores obtained in nvGesture's validation set - Isolated gesture only 73
Table 5.4 – Scores obtained in nvGesture's validation set - Full sample 74
Table 5.5 – Scores obtained in nvGesture's validation set - Gesture only 74
Table 5.6 – Scores obtained in EgoGesture's validation set - Segmented gestures 75
Table 5.7 – Best combinations of Cth for each *N* 79
Table 5.8 – Best combinations of Cth for each *N* 83
Table 5.9 – Number of computations per second performed on each model for different batch sizes 84
Table 5.10 –VRAM allocated during the tests for each model and batch size 85
Table 5.11 –Number of computations per second performed on each model for different batch sizes, compared to its unoptimized version 85

LIST OF ACRONYMS

RT – Real-time

NN – Neural Network

LSTM – Long Short-Term Memory

CNN – Convolutional Neural Network

2DCNN – 2D Convolutional Neural Network

3DCNN – 3D Convolutional Neural Network

OP – Optical Flow

KNN – K-Nearest Neighbors

PCA – Principal Component Analysis

SVM – Support Vector Machine

HMM – Hidden Markov Model

VRAM – Video Random Access Memory

RELU – Rectified Linear Unit

Contents

1 INTRODUCTION **14**

2 BACKGROUND **19**

2.1 REVIEW OF NEURAL NETWORKS 19

2.1.1 TRAINING NEURAL NETWORKS 22

2.1.2 NEURAL NETWORK CONCEPTS 24

2.1.3 NEURAL NETWORK STRUCTURES 27

2.2 STATE-OF-THE-ART IN NEURAL NETWORKS FOR VIDEO CLASSIFICATION . 35

2.2.1 CONVOLUTIONAL NETWORKS + LSTM NETWORKS 36

2.2.2 3D CONVOLUTIONAL NETWORKS 38

2.2.3 HANDCRAFTED FEATURES 39

2.2.4 TECHNIQUES COMPARISON 41

3 PROJECT SPECIFICATION **44**

3.1 FUNCTIONAL-RELATED REQUIREMENTS 44

3.2 QUALITY-RELATED REQUIREMENTS 45

4 IMPLEMENTATION **46**

4.1 DATASETS 46

4.1.1 THE JESTER DATASET 47

4.1.2 THE NVGESTURE DATASET 48

4.1.3 THE EGOGESTURE DATASET 50

4.2 NETWORK STRUCTURE 51

4.2.1 MODELS RT_RGB-8F AND RT_RGB-16F 52

4.2.2 MODEL RT_FLOW 58

4.3 NETWORK TRAINING 59
4.3.1 DATA AUGMENTATION 60
4.3.2 TRAINING PARAMETERS 61
4.4 MODIFICATIONS TOWARDS ONLINE OPERATION 63
4.4.1 DATA REUSE 64
4.4.2 ONLINE OPERATION 65
4.4.3 SUMMARY OF THE INNOVATIONS PRESENTED 68

5 VALIDATION **71**
5.1 OFFLINE VALIDATION 71
5.1.1 JESTER DATASET VALIDATION 71
5.1.2 NVGESTURE DATASET VALIDATION 73
5.1.3 EGOGESTURE DATASET VALIDATION 74
5.2 ONLINE VALIDATION 75
5.2.1 CALCULATING ACCURACY FOR ONLINE OPERATION 76
5.2.2 NVGESTURE DATASET 77
5.2.3 EGOGESTURE DATASET 79
5.3 PERFORMANCE VALIDATION 82
5.4 DISCUSSION 86
5.5 THREATS TO VALIDITY 88

6 CONCLUSION **89**

REFERENCES **91**

APPENDIX A – Jester Dataset Samples **97**

ATTACHMENT A – EgoGesture classes of hand gestures **100**

1. INTRODUCTION

Ever since the creation of the computer, researchers and the industry have been working on more natural ways of interaction with machines. Outside from the common mouse and keyboard, some mainstream technology that is already present in nowadays commercial products include: speech recognition (e.g. Google Assistant, Alexa from Amazon, Siri from Apple, etc), face recognition (e.g. Android and iPhone unlock mechanisms), full-body movement recognition (e.g. Kinect from Microsoft), etc. Gesture recognition is present in some technologies, such as the remote triggering of a camera shutter, but less present than the prior mentioned. The introduction of gestures as a way to interact with machines can have several utilities ranging from controlling household equipment remotely, to being used to detect human intention in a self-driving car, for example. One of the characteristics that possibly made speech recognition more widely available and more present in everyday applications is the lessened complexity (in terms of the amount of data needed to represent it) of it, when comparing to gesture detection through a sequence of images, for example. Sound is less computationally expensive to process than video and removing noise and distractions from the original captured signal (or even obtain a clear voice over a silent ambient) can be easier than isolating the parts of a gesture in a camera outside of a controlled environment. Besides being an alternative to voice commands (in ambient with loud noises, speech recognition is hindered, for example), gesture recognition adds a new layer of what machines can understand from humans, therefore, enhancing our capability of communication and easing intention understanding from computers.

Gesture recognition techniques can be divided into two major groups: contact-based and vision-based [33]. The former uses additional equipment, that will physically interact with the subject performing gestures. Some examples include: wearing gloves or suits equipped with accelerometers, submitting the subject to ultrasound waves and holding a remote controller (such as used in the Wii game console). Figure 1.1 shows an example of a contact-based technique for hand tracking. Although the capture of the movement is vision-based (a camera) a special glove is required to improve the capability of the system to detect the exact hand position. Vision-based gesture recognition uses cameras to record the subject performing gestures, although in some cases it may still require the wear of reflective material to enable body detection and tracking. Contact based gesture detection is often considered intrusive and is less well-received due to the necessity to use extra equipment.

Vision-based recognition, on the other hand, due to the simplicity for the end-user, got more attention and development in the past years [33].

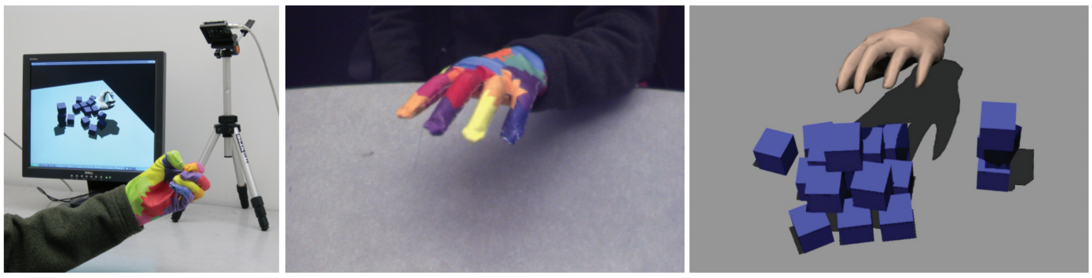


Figure 1.1 – "We describe a system that can reconstruct the pose of the hand from a single image of the hand wearing a multi-colored glove. We demonstrate our system as a user-input device for desktop virtual reality applications." Source: [46]

In addition to how the information of the gesture is captured, there is also a big distinction when it comes to how data are interpreted. According to [33], which compiled a study from many gesture recognition papers from 2005 to 2012, the majority of the vision-based approaches include some pre-processing of the captured image that is then used as features for a machine-learning algorithm to classify it as a certain gesture. Some of the reason why the pre-processing state is used in these works are:

- Isolate the hand from the background of the image;
- Compensate for the varying illumination conditions;
- Compensate for motion blur;
- Track hand movement;
- Identify and ignore other moving objects in the background.

This process of identifying only the meaningful information from the input data (the hands), while disregarding background information, is called detection, and can be achieved using a variety of approaches: based on skin color, based on the shape of the hand, based on pixel values (textures), based on motion, etc.

Another aspect, this time not always present, is tracking, i.e. the frame-to-frame correspondence for the segment of an image in which the hands are contained. Some implementations of tracking are as follows: template-based (what is considered a hand in the current frame is based on the hand detected from the previous frame), optimal estimation (a framework provided by the Kalman filter) and particle filtering (many particles are used to represent where the hands and fingers are).

The final step in gesture recognition is the recognition itself, i.e., using the given features extracted from the original data to predict what gesture is being performed. The recognition can be static (a still image can represent the gesture) or dynamic (gestures in which there is movement from the user's hand). Although it can be done, theoretically, by manually writing rules based on observed features, the majority of published techniques use some sort of learning algorithm to perform the recognition. Some techniques used in the literature are:

- K-means;
- K-nearest neighbor;
- Mean shift clustering;
- Support vector machine;
- Hidden Markov model;
- Dynamic time warping;
- Finite state machine;
- Neural Networks.

The recent highlight from these techniques in the past years has been the Neural Networks (NN) and its various topologies [23]. Due to the increase of available computing power, the enhancement of topologies such as the CNNs (Convolutional Neural Networks), and availability of richer databases with larger quantities of data to learn from, a sub-field of machine learning called "Deep Learning" was developed. When it comes to more data and computing power available, more complex models of NNs started to outperform other machine learning techniques [40], by being able to make better use of the additional data when utilizing more complex models. In addition to that, some topologies of neural networks, CNNs for instance, can produce its features, relinquishing the necessity of detection and tracking as separate steps (for classification only purposes). Many applications of CNNs shows its capabilities of recognizing a certain subject in the middle of a more complex scene, thus detecting and isolating hands from the rest of the environment is not necessary for most applications. Some implementations of CNNs will have only a normalization of the pixel values as the needed preparation before inserting the data in the network, in other words, the only information given to the network is the image itself. The features for detection can be learned during the training process (concomitant with the recognition portion of the network or not), without the need of human-created filters or other techniques in the process.

Neural networks implementations have been used to achieve excellent results with image recognition tasks [41]. The task of using neural networks for video classification (e.g. dynamic gestures) is a natural extension from this task and can make use of the same tools for identifying visual parameters on video. Video classification solutions are, nowadays, possible using the strong background of image recognition techniques and extending that with techniques that allows to the network to learn temporal aspects of the provided data, such as LSTM (Long short-term memory) networks, 3D convolutional networks (3DCNNs) or through the use of handcrafted features that extract motion information from the static frames, such as Optical Flow [19].

It is worth noting the definition of modes of operation for classification: offline classification and online classification. The former has all frames of the video in advance, the gesture recognition occurs after the video was captured, e.g. performing gesture recognition from a database. Since there are no hard deadlines for the processing of the data, the usual objective with this kind of operation is to achieve the highest accuracy possible. On the other hand, online classification refers to the operation in which the gesture recognition is being performed for every incoming frame, i.e. it does not have access to future frames and, in the case of real-time applications, it has a hard-deadline to yield the prediction, to keep up with the next incoming frames. This mode of operation can be used in real-time applications, requiring additional attention for computing power restrictions, or can be a simulated online operation, which is not being acquired in real-time, but it is used to check how accurate the model is when performing in this manner.

While the advancements in the usage of video footage for classification tasks have been heavily enhanced lately, most of the studies focus on offline classification, leaving out the considerations needed to apply such techniques in real-time applications through online classification. Designing a model for offline classification of video allows the design to dismiss some restrictions that would take part in a real-time environment, such as time to yield a prediction (overall complexity of the network), RAM and VRAM memory usage and dealing with a moving window of incoming frames. Making the model light on resources needed to run it, allows the technology to be used in conjunction with less expensive hardware, and thus be applied more broadly and in cheaper hardware.

This work will focus on the task of online dynamic gesture classification considering which topologies of NNs allow for a light implementation (low on computing power needed to run it) to use with online classification, using several hand-gestures datasets to train and validate the proposed model. Particularly, a study of which neural network structures can be better used to perform in online operation, the accuracy achievable by a lighter implementation when compared to state-of-the-art techniques, the corresponded comparison between accuracy and complexity and a study of the effectiveness of adapting trained models from one gesture dataset to another (transfer learning).

This work is developed as an embedded software problem. The neural networks proposed are implemented in python using the PyTorch library, which offers optimized code for describing the most common network structures. The main code of the networks is provided at https://github.com/fabiopk/RT_GestureRecognition. While the development aims at embedded systems, the networks were trained and run in desktop graded hardware. Implementing and testing the proposed networks in an actual embedded system (such as a Raspberry Pi or a smartphone) is part of future work.

2. BACKGROUND

The development of Neural Network is evolving rapidly since its invention in the past century, and most recent work has been highlighting this technique as one of the most common and effective inside the machine learning area. The advancements in hardware performance and richer databases allowed the supervised learning branch of neural networks to achieve a new baseline of what machine learning techniques can achieve in areas inside computer vision [45], natural language processing [49], and general classification problems.

In this section we will first explain how neural networks work and review some of the most relevant structures utilized in computer vision, focusing only on neural networks for supervised learning. After we will comment on different topologies that are state-of-the-art for video and/or gesture recognition. The reference for the review part of this chapter is based on the book [10].

2.1 Review of Neural Networks

A Neural Network (NN) is a technique of machine learning, first theorized in 1943 by the neurophysiologist Warren McCulloch and mathematician Walter Pitts, only tested in computers by the 1950s, due to computer hardware limitations of the time. Neural networks are inspired by the human brain, specifically on how neurons transmit, emit and block signals. A neuron, therefore, is the name given to the basic structure of a neural network, and it is responsible for combining many input "signals" from its input to make an output signal. A general image representing a neural network is shown in Figure 2.1. A neural network can have many layers, each one containing many neurons, in which the first layer of the network will contain neurons that receive the input of the system (shown in yellow) and will output an intermediate result that is passed to the next layer of neurons (blue neurons). The last layer of neurons will output the final output of the system (salmon-colored neuron). The layers between the input layer and the output layer, which are represented in blue and green in Figure 2.1, are called hidden layers. This kind of configuration in which all neurons of a given layer are connected to the neurons of adjacent layers is often called "fully connected".

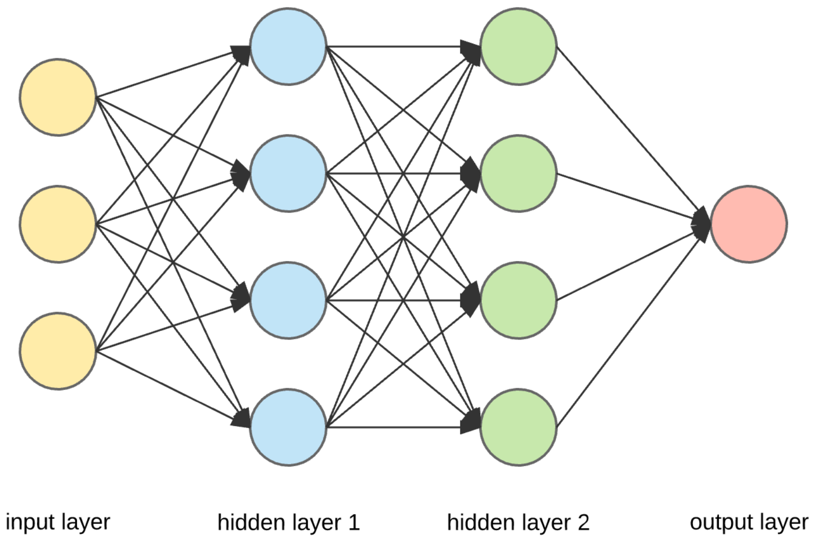


Figure 2.1 – Basic structure of a neural network
Source: [3]

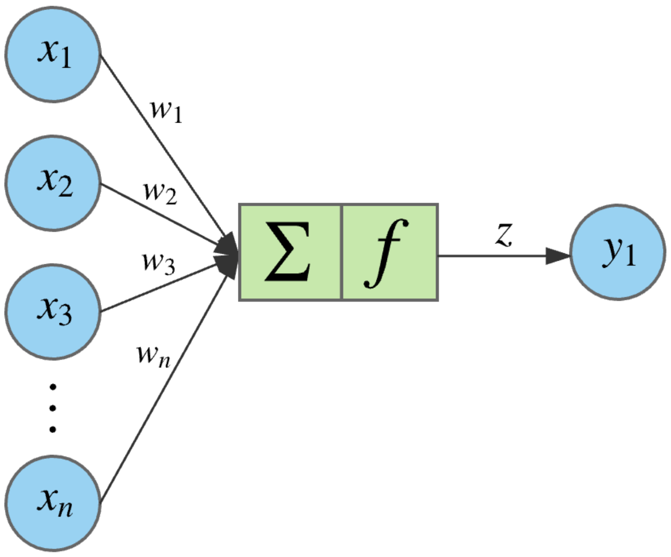


Figure 2.2 – Block diagram of a basic NN neuron
Source: [3]

The computation inside a neural network is a linear combination of all of its inputs passing through a non-linear "activation function". A representation of this is process is shown in Figure 2.2, and its output z is detailed in the equation (adapted from [10]):

$$\hat{z} = f(\hat{b} + \sum_{i=1}^{n} x_i w_i) = f(\hat{b} + \hat{x}.\hat{w}) \tag{2.1}$$

where x and w are the inputs of a neuron and the weights in vector form, f is the activation function and b is the bias. This input of each node from the previous layers is multiplied by a weight

ω_j which can be a positive or negative value, reinforcing or inhibiting a certain signal. The bias, sometimes represented as an input of unitary value, is almost always present to allow the neuron to fit to models that do not have an output zero when all inputs are zero (i.e. it allows the output of the neuron to "move" in the y-axis). There are many commonly used activation functions, each one with different applications. Some common examples are:

1. Binary step;
2. Rectified linear unit (ReLU);
3. Leaky rectified linear unit (Leaky ReLU);
4. Sigmoid;
5. TanH.

Some examples of activation functions are shown in Figure 2.3. ReLU, for example, keeps the original values of the signal if it is positive, or set it to zero if it is negative. The sigmoid function is useful for transforming a signal into another from the interval of zero to one. Different activation functions might have different uses depending on the need of the creator of the network.

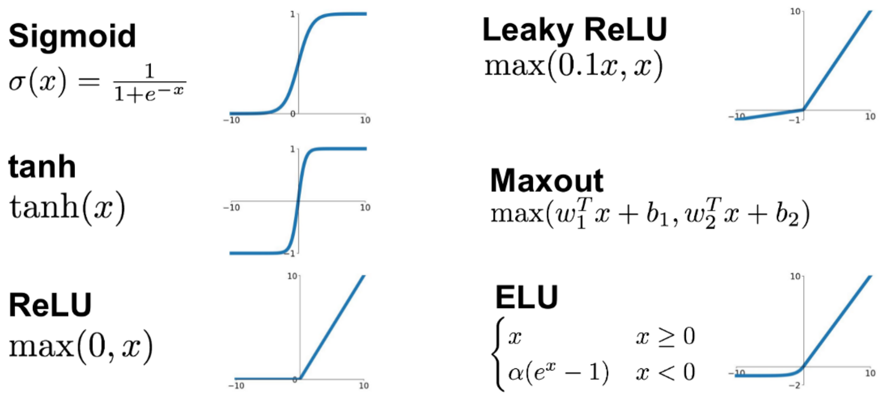


Figure 2.3 – Some examples of activation functions and its graphs
Source: [35]

Activation functions, depending on its type, help the model (i.e. the neural network structure) to solve non-linear problems, prevent values from overflowing (e.g. a $TanH$ function has an output range of $(-1, 1)$) and transform general numbers into probability values (e.g a $Sigmoid$ function transform values from the range $(-\infty, \infty)$ to the range $(0, 1)$).

The weights are part of the parameters of the neural network (the values that are changed when training a network). In supervised learning, the network modifies these values by training on a set of labeled data. The next section will elaborate on how neural networks are trained.

2.1.1 Training Neural Networks

The process of training a network refers to fitting that network to a certain dataset, that is, changing the values of the parameters of the network to yield an optimal accuracy between the inputs and labels of a certain dataset. The most common way of training neural networks is called backpropagation and consists of minimizing a defined loss function.

The loss function ($J(y, \hat{y})$) is defined based on what the network is trying to achieve. In regression problems (e.g. estimating the price of a house) a common loss function would be the Mean Square Error (MSE) of the predicted values and the true labels. For classification problems (e.g. deciding if a credit card is fraudulent or not) a common loss function would be Cross Entropy. The loss function can also help to compensate uneven class distribution and penalize high weights as a manner of preventing overfitting during the training process (overfitting will be commented in detail in a later section).

The backpropagation algorithm starts by inserting the input data into the network (either by feeding batched or the whole dataset), calculating all the values in the hidden layers until and output layer, this process is called "forward". With the predictions from the output layer ($\hat{y}$) and the true label of the data (y) it is possible to calculate the loss function. A loss function for a classification problem using cross-entropy is shown in the following equation (adapted from [10]):

$$J(y, \hat{y}) = -(y.log(\hat{y}) + (1 - y).log(1 - \hat{y})) \tag{2.2}$$

The next step in backpropagation is calculating the gradient of the weights in order to minimize the loss function. Since neural networks are deterministic, and all of its computational steps can be derivable, it is possible to calculate the gradient of weights that will reduce the loss function mathematically. With this gradient, small adjustments are made to the weights of the network by subtracting the current weight of the network by gradient calculated multiplied by a α factor called the learning rate. The following equation (adapted from [10]):

$$\hat{\theta_x} := \hat{\theta_x} - \alpha \frac{\delta}{\delta \theta_x} J(\hat{\theta_x}) \tag{2.3}$$

describe the updated value for the weights (θ_x) of the network on each step of the training process. This terms is a small number (commonly 10^{-2} to 10^{-5}) and prevents the network from making large adjustments at each step. Since the training is done iteratively, small steps towards the optimal point help the network to converge to an optimal set of weights, while a big learning rate might lead to bigger and bigger "adjustments" in the weights, setting it further from its point of local or global minimum of the loss function.

During the training process, all the available data are divided into three groups: train set, validation set, and test set. The NN is trained using the train set, either by forwarding the whole set through the network at once. Whenever the network is shown all the samples from a dataset an epoch has passed. Commonly, in the cases where the whole train set is too big to be fed to the network at once, or by design choice, the data are presented to the network in batches, which are a subset of the original set. Similarly, once all batches of a given set are used for training, an epoch has passed. In the case of using batches for training, the Gradient Descent technique is applied in the same manner, but it receives a different name, Stochastic Gradient Descent (SDG) since each step is not going necessarily in the gradient of the entire dataset anymore.

The validation set is used for evaluating the performance of the model during training. Since the data was trained using the test set, the network performance should be evaluated based on novel examples to check if it is learning useful and generic features of the dataset. When experimenting with different architectures of a neural network (e.g. different layers, different loss functions, activation functions) the data from the train set and validation set can be shuffled and redistributed among the two. This is helpful especially in smaller datasets in which there might be some bias for some classes for the train or validation set.

The test set contains data that should only be used for a final evaluation of the network, containing data never seen by the network before and never tested before. Even though the model is never trained in the validation dataset, some bias still exists for it since the experimentation done with the validation dataset will opt for models that have a good performance on it.

2.1.2 Neural Network Concepts

So far, we have discussed basic structures of simple neural network structures. This section will anticipate some of the possible concepts related to neural networks that will be mentioned further in the text.

Overfitting and Underfitting

Possibly the most common problem to be avoided in neural networks is overfitting, and it is the phenomenon of a network "adjusting" its response too close to the train set, instead of classifying based on general features that can be generalized for other samples. Overfitting a NN will cause it to have high accuracy on the training dataset but a low accuracy on the validation dataset. In extreme cases the neural network will behave as a Truth Table, knowing what are all the possible inputs and "memorizing" the correct answer for each item. If the network is able to memorize all possible inputs and its responses, there is no need to generalize (from the point-of-view of the optimization algorithm) and thus prediction on new data is impaired.

Some common causes of overfitting are:

1. Too few training samples;
2. Too many parameters available for the network;
3. Presence of bias in the training set;
4. Having too many features;

Having too few samples in the training set hinders the ability of the neural network to learn the truly useful parameters for a given dataset. For example, in gesture recognition, showing to the network only a few samples of a given gesture will hamper the understanding of the network that the movement of the hand is the deciding factor to classify a certain sample. Instead, the network might rely on other similarities that are not intuitive for humans, such as the texture of walls, clothes and skin, the color of the background, the relative position of the camera, and so on. Although it might seem a simple task for a human to understand that the hand gesture is the focus, for the network, there might be many other (possibly simpler) factors that help to differentiate one gesture from another. Increasing the number of samples in the training dataset reinforces common characteristics among the samples of a certain class, inducing the network to learn useful general features to minimize its loss function.

In addition to that, having a network with too many parameters allows the network to "memorize" specific characteristics of each sample to minimize its loss function. An analogy that might help to explain the problem of having too many parameters is the following: Let’s suppose we have a piece of paper to take notes to a math test about prime numbers. If the paper is big, we might be tempted to write all the prime numbers possible in the piece of paper, and check whether or not a number is prime by checking our list in the paper. However, if the piece of paper was small this strategy would not work well, it would be best to write the concept of what a prime number is. There is no rule of what is considered an ideal number of parameters of a network, most of it is based on trial and error and experience from other similar problems.

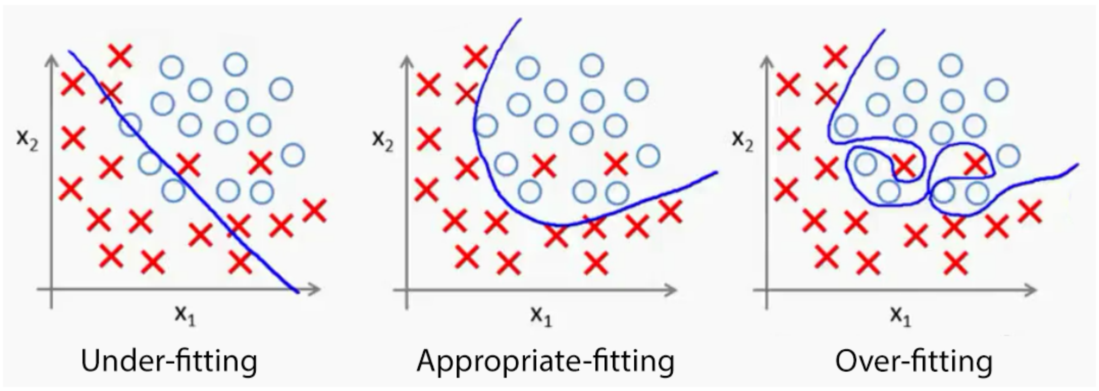


Figure 2.4 – Examples of overfitting and under-fitting for a two-class classificator
Adapted from: [2]

On the other hand, underfitting is the terminology used when the output predictions of a network do not represent well the train set, neither the validation set. It can happen when the network is not trained for enough epochs, there are too few parameters to represent the test set adequately or that the network is not adequate to learn the features needed for a given dataset. Figure 2.4 shows an example of a classification NN which tries to separate two classes (red crosses and blue circles) based on two features (represented by the x and y-axis). The blue line represents the border between where the network will predict as a red cross or a blue circle. In the first image, the line separates too poorly both classes, leaving many samples in the wrong side of the border. This might be the case that this model lacked the complexity to predict a border different than a line, for example. The right-most image represents an example of overfitting, the border do whatever is possible to get every sample right, resulting in a counter-intuitive shape. The red crosses in the middle of the blue circles might be outliers, and in that case, trying to force them to the right class might result in poor accuracy in the validation and test sets. The example in the middle, while getting a couple of predictions wrong, seems intuitively a more natural solution for this problem, which can only be evaluated based on testing outside of the test set.

Vanishing Gradient

A common occurrence of deep neural networks (ones with many layers) is the incapability of training properly due to a problem called "vanishing gradient". With the addition of many layers in a NN, the gradients of the loss function, during backpropagation, approach zero, making little to no modification on early layers, thus hindering training. This is especially the case when using activation functions that "squish" the signal, such as $Sigmoid$ and $Tanh$.

Some solutions to the vanishing gradient problem when implementing deep neural networks are:

1. Using activation functions that do not squish the tensors (e.g. $ReLU$);
2. Add batch normalization layers in between layers;
3. Make a direct path from earlier layers to the output of the network;

A "zoomed in" (a piece of a much larger network) version of the Google's network, Inception, is shown in Figure 2.5. This is a deep network used for general image classification. It is possible to see paths that skip some layers in the network every so often. This makes easier for the network to train through backpropagation of the earlier layers since there is a path passing through fewer layers.

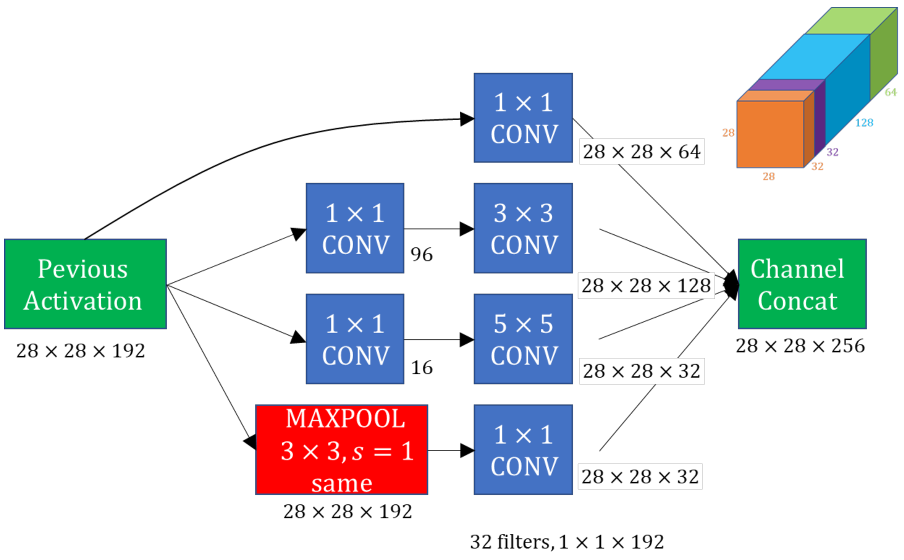


Figure 2.5 – "Zoomed In" repeated structure of Inception Network
Source: [41]

Transfer Learning

The concept of transfer learning refers to the re-purpose of a pre-trained network to fit a different dataset. One clear example of this is re-purposing a NN trained on ImageNet, a dataset containing thousands of classes and millions of images from all types, to train on specific classes that the user wants. As an example, Google's Inception, which is heavily based on feature extraction using CNN layers (see Section 2.1.3) while having the last fully-connected layers used for classification. Thus, it is possible to freeze (i.e. impede the change on the parameters) the training on the convolutional layers (which contain filters able to classify a variety of classes) while training only the last layers of the network for the novel data.

This process allows for a quick implementation of networks since it is possible to reuse a big part of a validated network to accelerate the training process. The concept of transfer learning will appear in this work in the context of using image classificators in video classification structures or even using networks trained for offline classification to real-time use.

2.1.3 Neural Network Structures

So far we have been discussing fully connected neural networks, which are still a relevant structure for the majority of proposed NN architectures. In this section, we will discuss other configurations of neuron structures, in which each node might not be connected to all other nodes from adjacent layers. These specific configurations were created having in mind a problem to solve or an adaptation to approach a specific data type more efficiently. Specific structures are commonly combined in layers to create a network.

CNN - Convolutional Neural Networks

One of the obstacles when trying to use regular fully connected networks to classify images is the needed number of neurons required to represent each pixel of an image. Considering an RGB image of dimensions $[100, 100, 3]$ (100 pixels of height, 100 pixels of width and 3 color channels) it would require a total of 30k weights per neuron in the first layer of the network. For larger images or more complex networks, these values can scale to unpractical numbers, limiting the application of such networks.

Convolutional Neural Networks (CNN) are specialized structures for neural networks that do not connect all the neurons between adjacent layers. Instead, it uses the operation of convolution between kernels (parameters of the network, which will be trained to fit a certain dataset) and the input data. The 1D discrete convolution operation, for an input f with length n and a kernel g with length m, is presented here (adapted from [10]):

$$(f * g)[i] = \sum_{j=1}^{m} g[j].f(i - j + m/2) \tag{2.4}$$

The convolutional process can be used for any dimension and is commonly used for images in the form of 2D convolutions. For the rest of this section, we will be using 2D kernels and images as the input of the networks to better illustrate how CNNs operate, however all the concepts here demonstrated can be adapted to other dimensionalities. The example of input image, in numeric form, and a 2D kernel is shown in Figure 2.6.

For the case of two-dimensional inputs and kernel, the convolution operation can be thought as if the kernel is positioned on top of the image, performing a pixel by pixel multiplication, summed and then moved to the right to repeat the process. As an example, let's consider the values for the image and kernel shown in Figure 2.6, and do a step by step calculation of the result of the convolution between the two, demonstrated at Figure 2.7 (steps "a" to "i"). In the first step, labeled as "a", the

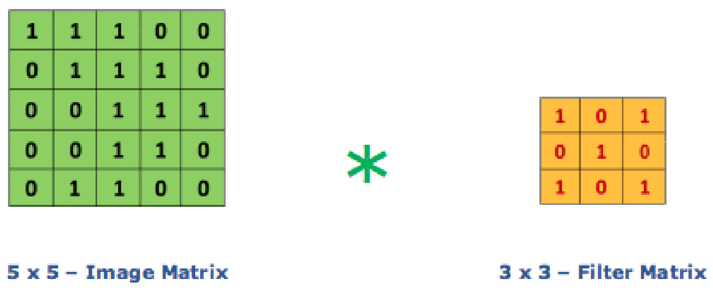


Figure 2.6 – Image matrix and 2D Kernel example
Source: [32]

original image is represented in green and the kernel is positioned on top of it, aligned in the first row and first column (its values are shown in red, as a multiplication) the overlap of both matrices is shown in yellow. This arrangement will yield the first value of the resulting convolution, by multiplying pixel by pixel of this combination and adding the resulting values, which for this case is 4. To calculate the next value of the convolution, the kernel moved one column to the right (see step "b") and the same process is repeated, yielding a 3 as the next value. The same process is repeated for step "c", and for step "d" the kernel is positioned in the first column again (since it cannot advance another column), however, shifted downwards by one line. This process is repeated for the remaining steps until the kernel is aligned with the right-most column and the last line. The result of each step is represented in the "Convolved Feature" matrix.

The shape of the resulting matrix depends on the shape of the original image, the shape of the kernel, stride, and padding. In this example, it was considered a stride of one, which refers to how many columns and lines the kernel moved between steps. A stride of two, for example, would imply skipping steps "b", "d", "e", "f" and "h", resulting in a result of size 2 by 2, composed by the values seen at steps "a", "c", "g" and "i". Also, there was no padding, if there was a padding in the example the original image would have a "frame" of zeros around it (in the case of zero-padding), and the kernel move across this frame containing the original image, increasing the number of steps needed to roam on top of it, and therefore resulting in an increased shape of the output of the operation.

This example described a convolution between an image with dimensions $[5,5]$ and a kernel $[3,3]$ with stride on one pixel and no padding. The stride represents how many pixels the kernel moves between each step of its calculation. A bigger stride will result in fewer steps in the convolutional process, and thus a smaller output image from the process. This can be used as a way of reducing the dimensionality of the network. The padding in a convolution is applied in the input image (usually

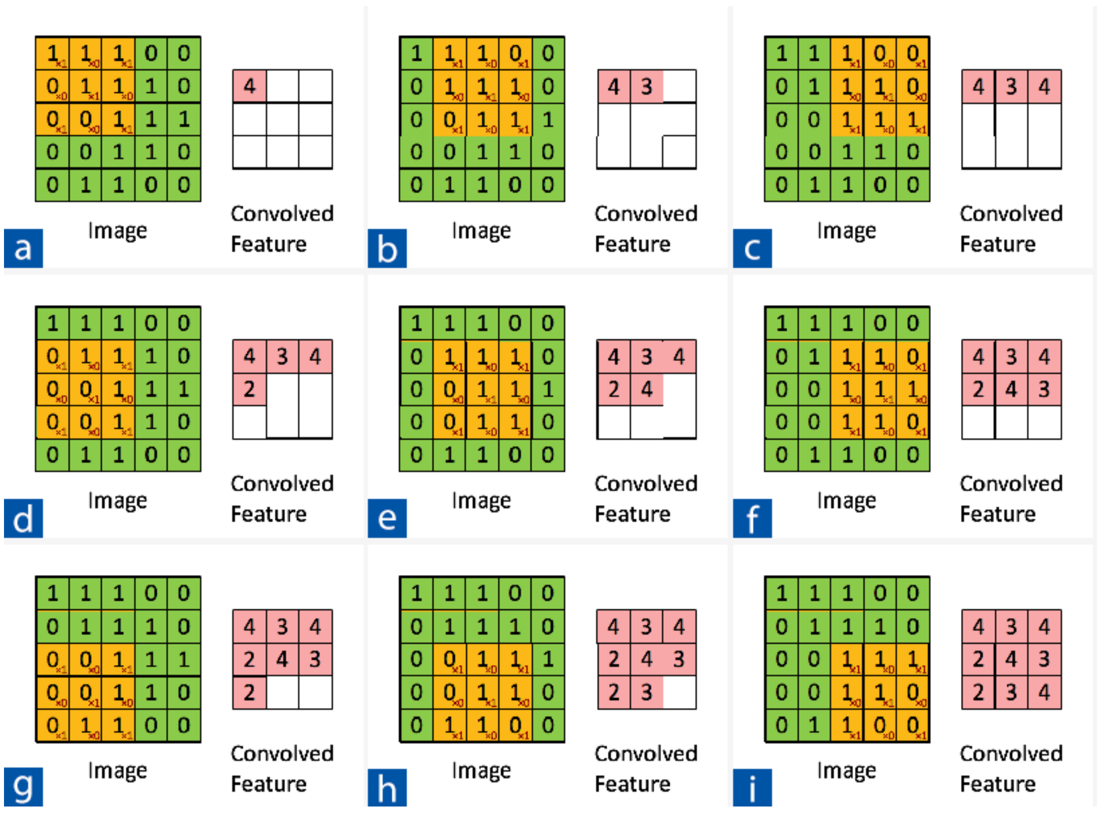


Figure 2.7 – Demonstration of a convolution between a 5x5 image and a 3x3 kernel
Adapted from: [32]

with zeroes, but other approaches are common as well) primarily in order to control the size of the output of the operation. In the example case, since there was no padding in the input image, the output image has dimension [3, 3], smaller the input image. If a padding of 1 pixel was applied to the input image (thus resulting in a [7, 7] image, with the original image surrounded by zeroes) the output image would have a dimension of [5, 5].

The use of kernels on images to extract features predates the CNNs. Specific kernels can be used to achieve edge detection, to sharpen an image, to blur an image and other features. Some examples of uses of kernels are shown in Figure 2.8.

These examples show the use of predetermined kernels, with known uses. CNNs have the values of their kernels modified by the training process. The examples shown contemplate small filter of dimensions [3, 3], however, bigger filters can be used to identify specific textures and even parts of objects. A smaller filter can still be used in succession (many CNN layers connected) to obtain the same effect of bigger filters, as suggested by [50]. In their work, it is possible to visualize what the filters of CNNs are abstracting from image datasets, and it is possible to recognize shapes that fig/kernels_hd.png

Figure 2.8 – Demonstration of application for some 3x3 kernels
Source: [32]

reassemble human body parts (eyes, faces) and parts of an object (e.g. a tire from a car). CNNs are capable of creating these kernels without direct human input, by training on a certain dataset.

Similar to the fully-connected networks, the CNNs usually have an activation function following the convolution operation, the most common approach is to use $ReLU$. In order to reduce the dimensionality of the tensors through the network, pooling operation is sometimes used as well. In the case of images, the process of pooling creates a smaller output image with each pixel based on several pixels from the original image. For example, a pooling of dimension $[2, 2]$ will transform every 4 pixels (in a $[2, 2]$ shape) of the original image into 1 pixel of the output. Some examples of pooling are Max pooling (which will take the biggest value out of the 4 pixels) and average pooling (which averages the value of the 4 pixels). Another common inclusion in CNNs is a step called batch normalization. As explained in [14], normalizing the values of signals allows the use of bigger learning rates, thus improving the speed of training of the network, makes the network less susceptible to initialization problems and helps to generalize data acquired from different sources.

A full CNN network will usually have several layers of convolution, activation function, pooling and, possibly, normalization. After all these steps, the tensor (which is at this point usually smaller than the original data due to mainly pooling) is flattened, losing its two-dimensionality, in order to be fed to a fully-connected layer that will yield the final predictions for the network. The structure of a classic CNN, ResNet-18, is shown in Figure 2.9. This network was initially developed to perform handwritten digit recognition, using as input small monochromatic images of a single written digit. The dropout layers that appear in the image are a special type of layer, used in training only, that deactivate some percentage of signals randomly. This prevents the network from relying on a single path developed, which would make it more prone to overfitting.

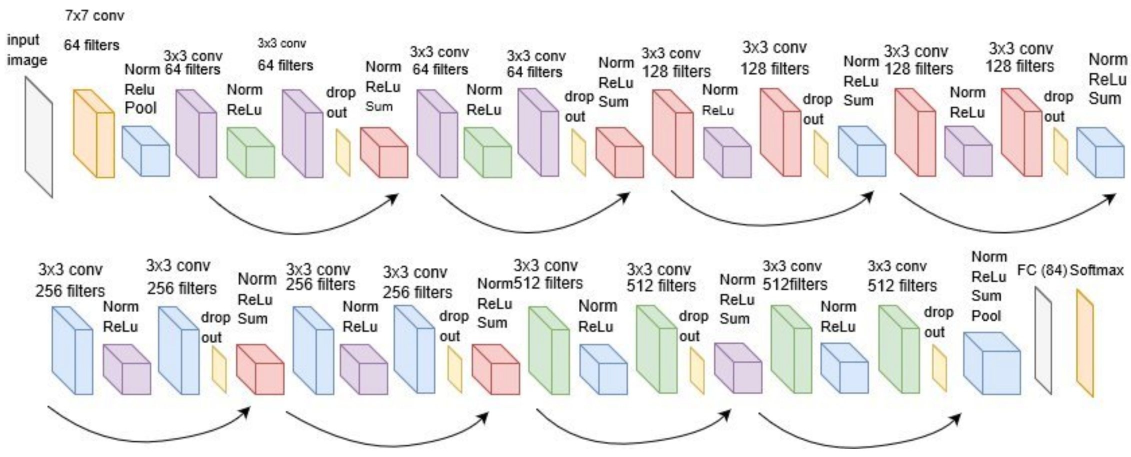


Figure 2.9 – Structure of ResNet-18
Source: [1]

RNN and LSTM Networks

So far, we have discussed a specialized network for spatially related data, such as images. In this section, we will comment on the most common networks utilized for temporally related data (a sequence of some sort), which are RNN (Recurrent Neural Networks) and its more complex successor: Long Short-Term Memory (LSTM) Networks. In a fully connected network, all the data are presented at the input of the network, without any context of the order. Considering an application where the network is responsible for predicting the temperature for the next day, feeding the daily temperature of the past year would be very helpful. However, in a fully connected network, the notion of order is not present, making it difficult for the network to learn patterns of the temperature across the year. In this case, a hypothesis of what might occur is the network relying only on the neurons containing the temperature of the most recent days, and making a prediction close to those values.

Recurrent Neural Networks try to solve this problem by feeding data "sequentially". Each data point (for our previous example, the daily temperature) will pass the network in sequence, taking into consideration parameters from the previous data point. More specifically, the hidden layers (or part of them) are fed as inputs to the next data point in addition to the actual data point for that cycle. An RNN can be unrolled to see that, actually, data does not need to be fed sequentially, but it just means that older data points have more computations before reaching the layers containing the final prediction. Figure 2.10 shows a RNN, presented showing its feedback loop and in its unrolled form.

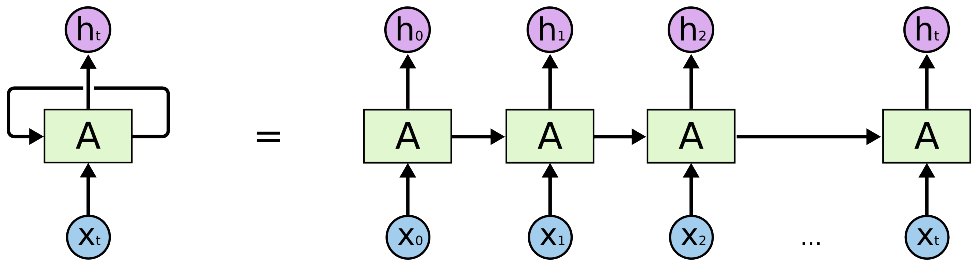


Figure 2.10 – An unrolled recurrent neural network
Source: [7]

One of the problems that the RNN structure creates is the difficulty to train longer sequences due to the vanishing gradient problem. For our previous example, there would be at least 365 layers between the input of a certain neuron and the outputs. Due to how backpropagation works, the gradients have less and less impact the closer a neuron is to the beginning of the network, not being able to modify properly its values. Because of this, RNNs do not perform well when dealing with longer data sequences when compared to LSTM networks, which were created as an alternative to RNNs when dealing with more complex data.

The Long Short-Term Memory Network (LSTM) was first proposed by [12] in LSTM as a variation of an RNN capable of learning long-term dependencies. LSTM replace the conventional RNN structure with LSTM cells (shown in Figure 2.11). These cells add a more complex structure with the added state (the top line that runs through the cells) and gates (depicted by pink elements) driven by neural network layers (depicted in yellow).

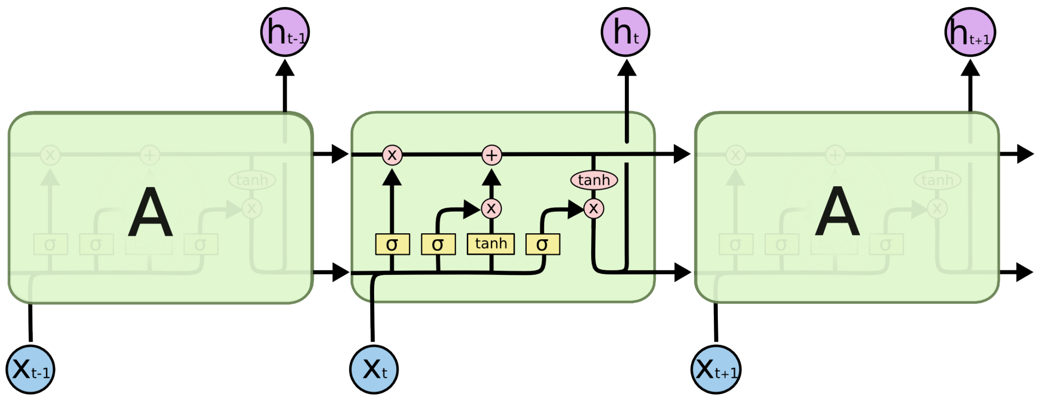


Figure 2.11 – Representation of an LSTM cell, inside a LSTM network
Source: [7].

The main gates represented in Figure 2.11 are the forget gate, input gate, and output gate. The forget gate (Figure 2.12 a) is responsible for "forgetting" information from the previous cell. The input gate (Figure 2.12 b) is responsible for adding new information to the state of the cell. The process is divided into two parts: a sigmoid (σ) network decides which values will be updated and next the *tanh* layer decides which are the candidates to be added to the new cell state (Figure 2.12 c). Lastly, the output gate controls what the output the cell will be, based on the cell state. A sigmoid (σ) network determines which parts of the cell will move to the output and are multiplied by the output of a *tanh* function from the state of the cell (Figure 2.12 d). All these neural networks are fed by the previous cell output and the inputs for this step of the sequence and are trained in to be able to forget/remember information that will render the lowest loss for the network.

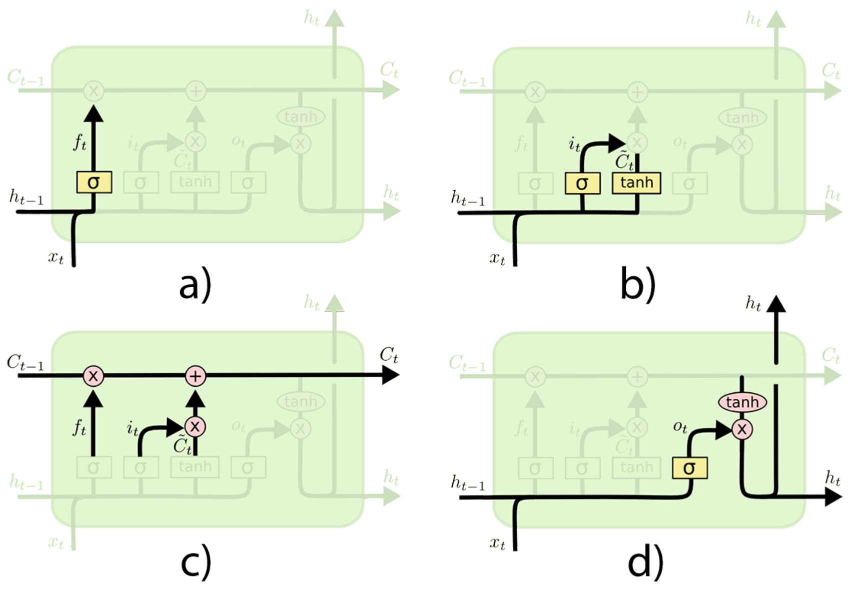


Figure 2.12 – Different parts of a LSTM cell: a) forget gate; b) input gate; c) state path; d) output gate
Source: [7].

LSTM networks achieved state-of-the-art results in many areas, such as speech recognition [34], video classification [21] and NLP (Natural Language Processing) in general [8].

2.2 State-of-the-art in Neural Networks for video classification

The task of video classification can be broadly defined as the task to extract information from video to attribute a class (or label) to it, or an event contained on it. For example, identifying what sport is being played in a certain sample of video (classification of the entire video) or detecting in a security camera when suspicious activity might occur, therefore labeling a specific event inside the video. The task of gesture recognition can be considered a type of video classification, in which the labels are the different gestures recognizable by the network and the task consists of identifying which and/or which gesture is being performed in a given sample.

Another aspect to consider in the task of video classification is if the classification is done online or offline. The former refers to yielding a label as frames of the video are being processed, in a predictive manner, unable to have access to future frames of the video. There is a time aspect in this mode of operation, i.e. the label is yielded at a given time (at a given frame of the video, for instance), and cannot be corrected later on if the label was later discovered as incorrectly classified. In online classification, the delay to make the prediction is relevant because if the network waits for higher confidence before yielding a label, it might introduce a bigger delay in the prediction, while if it is more sensitive (more likely to yield a label) it might result in an increase of false positives. Online classification is relevant for applications in which the video is being acquired concomitantly to the classification process and the issuing of the label has a deadline.

On the other hand, offline classification refers to the same process of labeling, but without any of the constraints mentioned before, a sample can be analyzed in its plenitude before issuing the labels related to it. It can be used in applications in which the sample to be analyzed is finite and its end is known, e.g. when a video is uploaded to YouTube, it is automatically tested for containing copyrighted content, nudity, violence and other types of content that not allowed in the platform.

Video classification tasks have two major aspects to be retrieved from the video data: spacial features (visual characteristics of single frames) and temporal features (information taken from the sequence of frames of a video). This dissertation is focused on the specific application of hand gesture recognition, and since gesture recognition is a specific sub-group of video classification, this section will contemplate networks that have state-of-the-art performance in more common video classification datasets.

The majority of studies in video classification is focused on offline classification, i.e.checking how different network topologies can result in the highest accuracy for a given dataset or task. Online classification has gained more attention in recent years [18] due to the possibility of using these networks in real-time applications, which is a hard task given the computational power needed to run video classification models at compatible speeds to meet deadlines in these applications. This dissertation will consider the gesture classification problem as an embedded software problem, not only accounting for how accuracy performance is affected in online operation, but also take into consideration power consumption, the complexity of implementation, delay time, and others while choosing the network topology. Although this work will not specify a hardware platform to run this code, the considerations needed to implement the proposed approach in lower-cost equipment are taken into account while designing the proposed topology.

This section will present some of the traditional and state-of-the-art techniques used to achieve high accuracy in video datasets such as Youtube 8M, Something-Something, Jester, EgoGesture, HMDB51, and UCF-101. Although most of the work presented in this section represent offline implementations of video classification tasks, these represent the techniques which, so far, yield the best accuracy on most common video classification datasets. Based on these topologies, the proposed technique of this dissertation was based on, picking elements that are, in a first moment, good candidates to work in an embedded system.

2.2.1 Convolutional Networks + LSTM Networks

Convolutional networks (CNN) are responsible for high performance of image classification tasks in deep learning[41] , in datasets such as ImageNet and CIFAR-10, some examples include: ResNet [11], DenseNet [13] and Inception [41]. On the other hand, LSTM networks have achieved state-of-the-art results when dealing with sequential data, such as sentiment analysis and speech recognition [29, 34]. Combining the capabilities of extracting spatial features with convolutional networks and using LSTM networks to connect those features temporally achieved state-of-the-art such as the results presented in works as [21, 22, 47]. One of the facilitators of using 2D convolution for visual tasks is the possibility of using transfer learning from networks previously trained on other databases (e.g ImageNet). This allows the network to re-use kernels which will help the network to train faster, and possibly achieve higher accuracy.

An example of an application of this kind of network is seen in [22]. In this work, the authors present the structure of an "attention-based" network, which decides which part of the video should be focused on. The authors use RGB and flow frames as the input of a CNN, in a two-stream configuration, for action recognition and automated video captioning. A simple demonstration of the network's structure is shown in Figure 2.13.

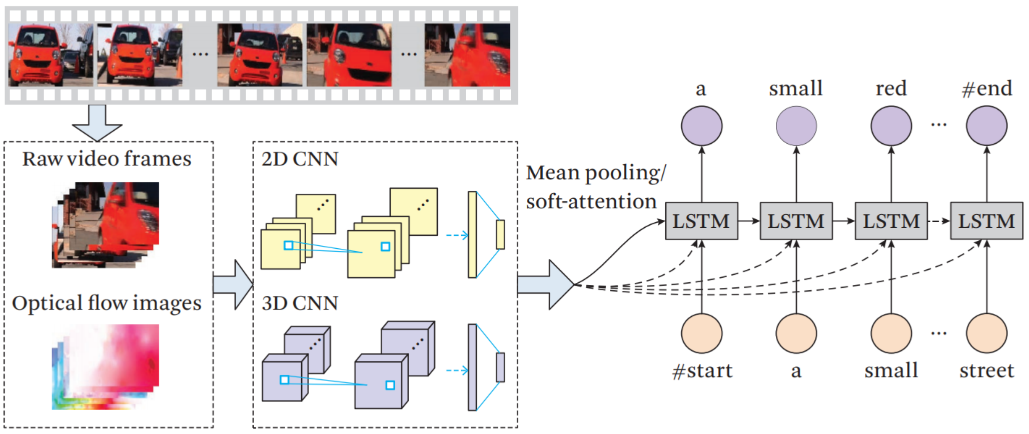


Figure 2.13 – VideoLSTM network structure
Source: [22].

This network also contains 3D CNN structures and the use of flow frames, which will be further commented on in the following sections. The use of CNNs and LSTM in conjunction is to have the CNN extract the spatial information and feed this data to the LSTM (which normally only operates with flattened data, although multi-dimensional networks can be used). The LSTM in this example can be used to make a video label prediction or to generate a caption describing the content of the video. The structure of an LSTM allows each cell to make a word prediction based on the predictions of previous cells, this can also be used for text generation (based on some samples) such as demonstrated in [31].

2.2.2 3D Convolutional Networks

The natural adaptation of the successful convolutional networks to work with a sequence of images is to expand the network's kernels an extra dimension and input all the frames at once, adding a new dimension to the input data. A 3d convolutional layer can extract spatial-temporal features, which is usually combined with a linear layer to adapt the output to the number of classes of a given dataset. This technique has been used in previous applications such as action labeling [42], real-time object recognition [24] and action recognition [21, 37], hand-gesture recognition (using an additional depth sensor) [26] and large-scale video recognition (Youtube 1M) [16].

One difference from the 2d convolutions is the added difficulty to re-purpose other trained models, resulting in most of the work with those networks having to train from scratch. Although is hard to re-use state-of-the-art trained 3d convolutional networks for a different application, these structures are commonly present in many state-of-the-art approaches for video classification, such as in [19, 20].

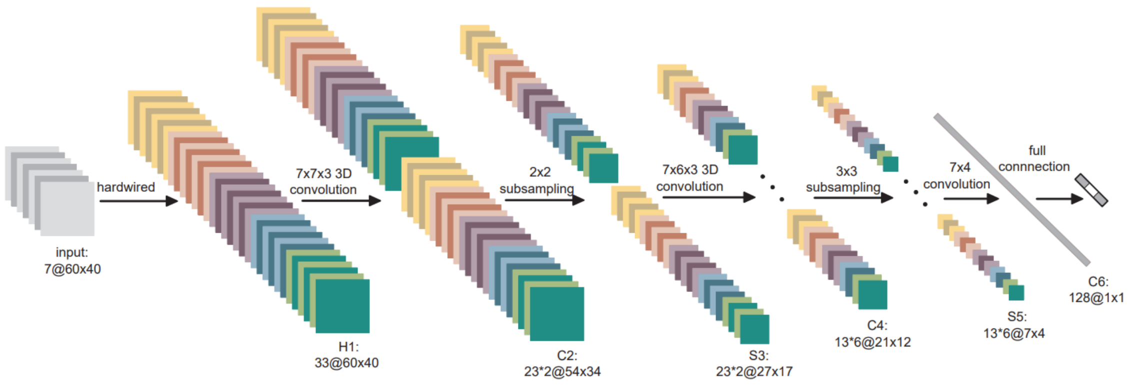


Figure 2.14 – A 3D CNN architecture for human action recognition.
Source: [15].

In the Figure 2.14, the structure of a 3DCNN presented by [15] is shown. In this work, the network should be able to identify gestures in real-time scenarios, such as humans interacting with objects, putting the cellphone to their ears, pointing, etc. This network shows a network using only RGB frames as its input and 3D CNNs as the feature extraction layer. Although 3D CNNs are not necessarily easy to compute in real-time, this network did not rely on any handcrafted feature and was suitable for real-time operation.

2.2.3 Handcrafted features

In addition to features learned by NN through training, it is possible to introduce hand-craft features to the network as an attempt to facilitate the training of network by feeding into its data which are believed to be easier to extract information. As mentioned before, one of the advantages of CNNs is that there is no need to "manually" create features for the detection step of the gesture recognition process. Nonetheless, the introduction of hand-crafted features i.e. features created by a human and not the network itself, is shown to help with the network performance (accuracy) in some cases. These features are often introduced to the network as a pre-processing of the data, modifying the data before feeding it into the network.

One example of pre-processing that could be used for hand gesture is OpenPose [5], which is able to trace arms, torso and fingers position form a 2D video source. A demonstration from this tool is shown in Figure 2.15. The output of this tool is not the overlay image shown in the example itself, but rather a description of each member recognized, its junctions, angles, etc. Using the output of this library, [30] was able to perform full-body gesture estimations in real-time.

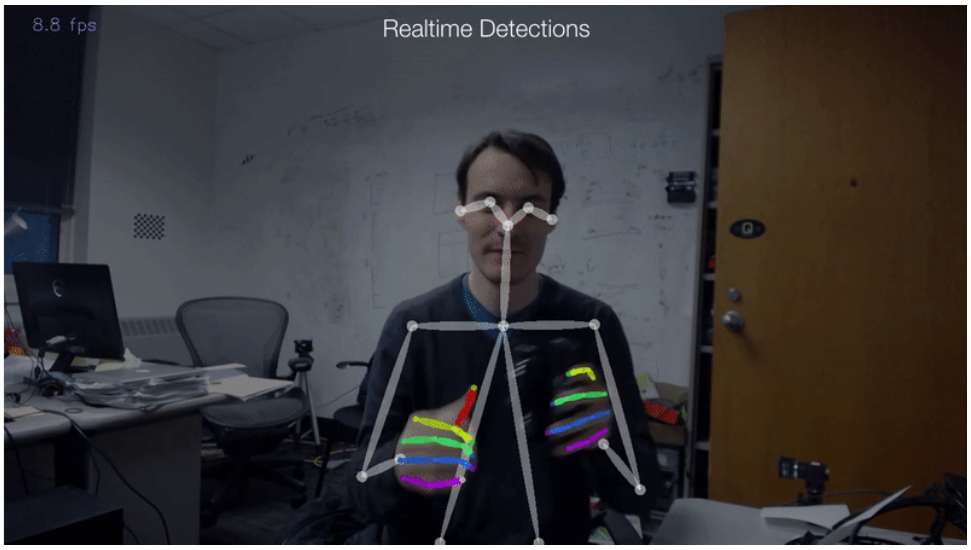


Figure 2.15 – A demonstration of OpenPose capabilities of tracking the human body.
Source: [5].

The most present handcrafted feature in video classification is possibly optical flow [22], which is a process that extracts the perception of motion from two or more frames into a visual representation. Figure 2.16 shows a demonstration of this process. On the left, there are two overlapped frames from a tennis player moving (the position in the camera for both pictures is the same). The overlapping is only present for demonstration purposes since both frames would look the same side-by-side. Based on those two frames, an algorithm (optical flow) creates 2 arrays that represent the perceived movement between the two frames. One array represents the horizontal movement between the images while the second array represents the vertical movement. The image on the right side, in Figure 2.16, is a combined version of those two arrays, created to visualize the effect of vertical and horizontal movement in a single RGB image. The name given to these arrays or the combined version of them is a "flow frame".

A flow frame represents the movement between two video frames, reinforcing the movement happening among videos frames (e.g. where the tennis player is moving), instead of the spatial characteristics (e.g. the clothes used, the racket, colors, textures, etc). There are multiple implementations of optical flow available, and studies aiming to develop more accurate and less computational expensive approaches to it, such as [38].

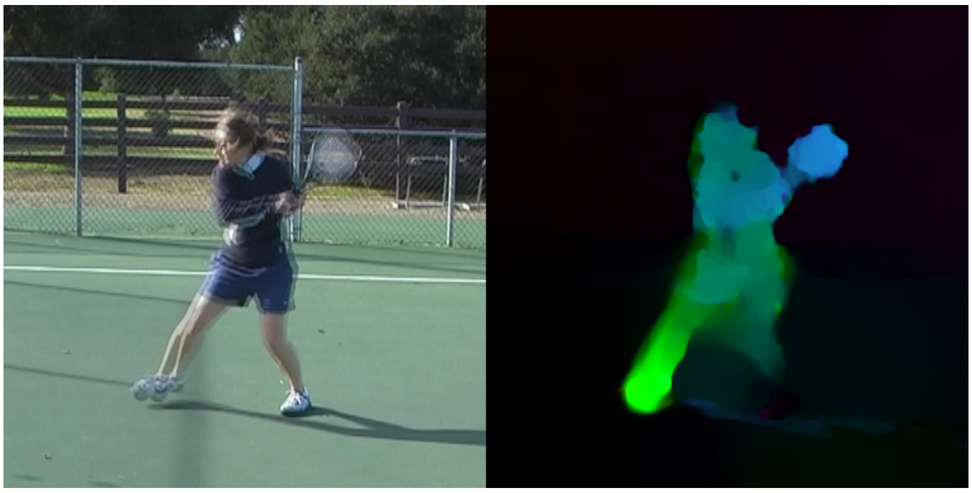


Figure 2.16 – A demonstration of the optical frame (right) resulted from the two overlapped original frames (left).
Source: [4].

An example of an application using optical flow is shown by [36]. Is this paper, the authors are inspired by the human visual cortex, which is divided into two parts: the ventral stream (which performs object recognition) and the dorsal stream (which recognizes motion). Their suggested network divides the task of recognizing the spatial-temporal aspects of videos in two networks. The network responsible for detecting spatial characteristics performs action recognition from still video frames, using a CNN pre-trained on ImageNet. The temporal network instead of receiving the raw video receives flow frames (optical flow was implemented using OpenCV toolkit), which analyzes the relative movement of pixels across frames of the videos, and produces images that reflect the movement present in the footage. The result of both networks is then combined (late fusion) to produce only one prediction from both networks. This network was able to achieve top performance at the UCF-101 dataset and state-of-the-art results in other datasets such as HMDB51. A representation of the network is shown in Figure 2.17, it is possible to see both branches of the network: on the top the spacial part of the network, which will be able to detect characteristics such as the presence of a bow or not and the bottom branch of the network receives information about the movement occurring in the video. Both outputs from each branch combine at the end to predict what is the correct label for the video.

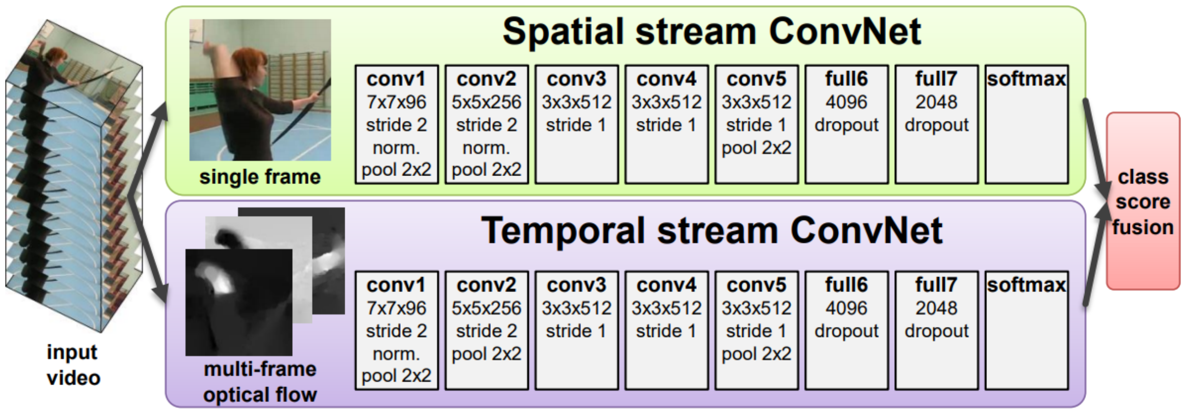


Figure 2.17 – Two-stream architecture for video classification.
Source: [36].

2.2.4 Techniques Comparison

Some recent approaches for action and hand gesture detection are listed in Table 2.1. These were selected based on the datasets used: Jester (the main focus of this work), nvGesture, EcoGesture and other staples of action recognition as UCF101 and HMDB51. The latter two datasets share similarities with Jester by having users interacting mostly with their hands with other objects, in contrast to other general-purpose video datasets. The nvGesture and EcoGesture are datasets of gestures, however from a different angle of capture and different labels.

The techniques presented that are not marked for online operation are usually concerned with achieving maximum accuracy within each dataset, making use of every possible resource that might help achieve it. These are the state-of-the-art techniques and their results in each dataset represent the standard to be chased. It is worth noting that the computing power required for online classification is

Table 2.1 – Related techniques

Tecnique	Year	Datasets	Mode of Operation
ECO[53]	2018	UCF101 HMDB51	Online
EMV-CNN [51]	2016	UCF101 HMDB51	Online
ConvNet[6]	2017	UCF101 HMDB51	Offline
MFF[19]	2018	Jester	Offline
MFNet[20]	2018	Jester	Offline
VideoLSTM[21]	2018	Jester	Offline
3DCNN [18]	2019	Jester nvGesture EcoGesture	Online

not a project constrain focused by the authors, for this reason, it is possible that these approaches are unable to perform in real-time on most hardware platforms.

Neural networks for gesture recognition in real-time applications (in which video acquisition and classification must happen concomitantly), should take into consideration the resources needed to run online classification in real-time. In [53] and [51], the authors developed an approach mainly based on 2D and 3D CNNs, without the need for flow frames (which are computationally expensive for real-time use) by using motion vector CNNs. The former technique is able to produce captions descriptions of videos in online operation. The author was able to reach accuracy results comparable to those of techniques using flow frames, with the advantage of being able to run in real-time. In the EMV-CNN technique [51], the author compares the performance of executing Optical Flow and executing the Motion Vector network, resulting in a reduction of 27 times for the computation of the latter. In the work of [18] a network built for gesture detection was build for online operation, utilizing mainly a 3D CNN topology. The network was first trained in the Jester dataset due to its large number of samples and then retrained for benchmarking in the EgoGesture and nvGesture datasets.

The technique proposed in this dissertation aims to find an efficient neural network design for hand gesture classification in online operation, considering not only the accuracy obtained in each dataset (such as most of the offline techniques presented in Table 2.1) but also the implications on online accuracy performance and aiming at lower-capable hardware, such as smartphones. Differently on what is done in the online techniques [18, 53, 51] presented in Table 2.1, this dissertation aims for an even lower computational cost of network, possibly affecting the overall accuracy in the tested datasets, but enabling it to run on cheaper and less powerful hardware. This is done by reducing the number of frames input to the network, avoiding the use of computationally expensive pre-processing techniques and controlling the number of parameters used in the network (or the number of bytes needed to represent the operations in the network). The following chapter will list the project specifications for this dissertation and Chapter 4 will discuss the experimented topologies.

3. PROJECT SPECIFICATION

In this chapter, we will discuss the overall constricts and design characteristics for this work. This work comprehends the data journey from RGB images of a front view of a person performing gestures (not including the image acquisition itself) to an online gesture detection indicator, which should display when and which gesture was performed continuously. For this work, real-time operation detection will be considered as a detection up to 500ms [25] (based on human perception of real-time) after the gesture is performed that can work continuously.

The proposed implementation shall be hardware agnostic, developed considering only performance and power (energy consumption) in mind, but not taking into account the specific architectures from any hardware.

3.1 Functional-related Requirements

This section lists the functional requirements for the system. Since the performance of the neural network (power consumption, time to process, etc) depend highly on which network structure is implemented, which tools are available on the hardware and other external factors, performance requirements will be treated as non-functional requirements due to case-by-case analysis required to measure those metrics.

The main metric used for comparison with other deep learning gesture detection techniques will be the accuracy of the technique in datasets such as Jester, Nvidia Gesture (nvGesture) and EgoGesture. The Jester dataset will be used for most of the work, due to its quantity of videos and a more convenient and natural angle of capture from the videos of the gestures. At the time of this work, the best reported accuracy for the Jester dataset is around 96%. Since this work aims for a lighter implementation of a gesture detection network, using less complex models and using fewer frames of the video, the accuracy of the network will be targeted at 85% or higher, and further effort will be put into the post-processing of the network to yield a better gesture prediction capability in real-time.

Preliminary testing with the planned structures and the dataset allowed to establish a reference for the functional requirements. Early developed networks, without further optimization, were able to achieve above 80% accuracy (in the Jester dataset) with less than 50MB of parameters. For the delay requirement, similar work in gesture recognition, such as [18], shows that these models are able to predict gestures before its completion, yielding a negative delay in terms of response, when considering the end of the performed gesture as the reference.

A list of the main project functional-related requirements is presented below:

1. The prediction should occur before the gesture performance ends (negative delay);
2. The accuracy of the network on the Jester Dataset should be greater than 85%;
3. The neural network model used should allocate at maximum 100MB of video memory.

3.2 Quality-related Requirements

A list of the main project quality-related requirements is presented below:

1. The implementation must work in a moving window of frames environment, without indications of when a gesture is starting or ending;
2. The performed gestures should be recognized only once during the time that it is performed;
3. There should be no prediction while the user, in front of the camera, is not performing any of the gestures known by the network;
4. Contrary to previous work presented in Chapter 2, whereas performance was not the main concern of design, the proposed technique have it as it main focus;
5. The implementation should take into account computational performance of the overall system as well;
6. The system shall receive only RGB frames as its input. Any other image processing such as Optical Flow should be considered part of the system, when benchmarking for power and performance.

4. IMPLEMENTATION

In this section, we describe the development of the work, the considered constrains for an online classification application, and how our network is organized to be efficient in terms of computing power while maintaining as high accuracy as possible to similar state-of-the-art proposals. Particularly, we first define how the online classification is done and how data is structured. Then, we introduce the network structures (the layers contained in it) considered to be part of the final model and how and why some structures were or were not a good fit for the network.

4.1 Datasets

In our context, datasets are a well defined group of labeled data, which are elaborated with the intent of serving as a source of data to test algorithms and train them (in the case of machine learning algorithms). Specifically for this work, the datasets described in this section contain labeled video samples of gestures being performed. These are used not only to train the models mentioned in this dissertation, but also serve as a fair common ground to compare to other techniques related to this work.

For this work, we are using three datasets in order to train, validate and compare the proposed models in this dissertation. The main dataset used is called Jester [43], and it was the dataset primarily used to experiment with different network topologies, train the models proposed from scratch and compare the results to other offline techniques. The datasets nvGesture [39] and EgoGesture [52] are used to validate the offline and online operation of the models previously trained on Jester. Utilizing a process of transfer learning, it is possible to modify a previously trained model to perform a different task. Specifically in this case, utilizing transfer learning it was possible to adapt models trained to perform well on the Jester dataset to make use of the features learned beforehand to perform well in a different dataset, such as nvGesture and EgoGesture.

Utilizing more than one dataset to validate the proposed model shows that the the trained neural networks are actually learning useful features for gesture recognition, instead of being artificially good at a specific dataset. In this sections we will describe the datasets utilized in this work the differences between them and show an example of what each sample look like for each dataset.

4.1.1 The Jester Dataset

The main dataset used for this work is called Jester, and it is provided by TwentyBN. It contains over 148k videos of 27 different labels [43]. A sample of 5 frames from each class of this dataset is shown in APPENDIX A (the frames are not necessarily consecutive, and the main objective is to demonstrate how the gesture is performed). The labels for this dataset, and its amount of samples are shown in Table 4.1.

Table 4.1 – Classes in Jester dataset and its number of samples

Label	Samples
Doing other things	12,416
Drumming Fingers	5,444
No gesture	5,344
Pulling Hand In	5,379
Pulling Two Fingers In	5,315
Pushing Hand Away	5,434
Pushing Two Fingers Away	5,358
Rolling Hand Backward	5,031
Rolling Hand Forward	5,165
Shaking Hand	5,314
Sliding Two Fingers Down	5,410
Sliding Two Fingers Left	5,345
Sliding Two Fingers Right	5,244
Sliding Two Fingers Up	5,262
Stop Sign	5,413
Swiping Down	5,303
Swiping Left	5,160
Swiping Right	5,066
Swiping Up	5,240
Thumb Down	5,460
Thumb Up	5,457
Turning Hand Clockwise	3,980
Turning Hand Counterclockwise	4,181
Zooming In With Full Hand	5,307
Zooming In With Two Fingers	5,355
Zooming Out With Full Hand	5,330
Zooming Out With Two Fingers	5,379

The labels "No gesture" and "Doing other things" represent a sample without the performance of a gesture. The first of the two consists of a static person, with no particular movement being done. The "Doing other things" label includes all kinds of actions non related to any of the gestures, such as drinking water, interacting with a pet, checking their phone and so on. Due to this variety present in this class, its number of samples is considerably larger if compared to the, on average 5200 samples per class.

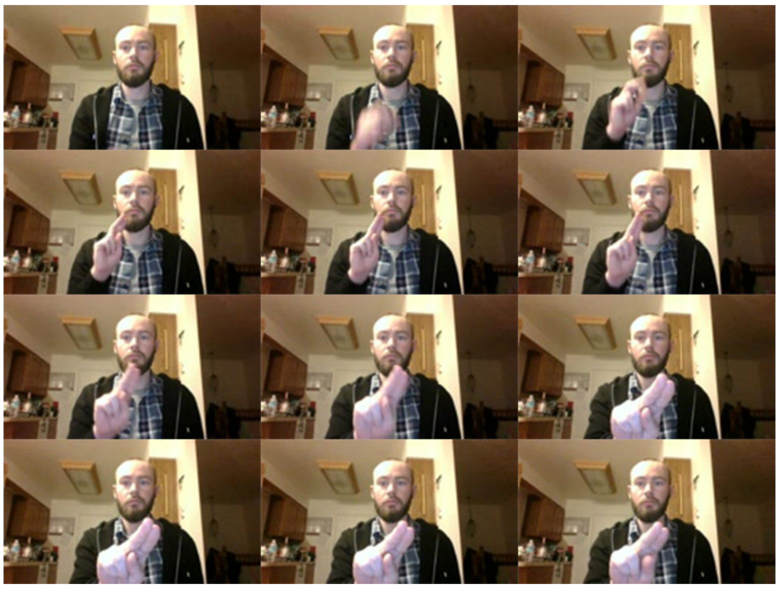


Figure 4.1 – Sample of a gesture from Jester dataset (12 out of 26 images shown, for this specific sample)

The data is provided in the form of JPEG images (RGB, 3 channels) for each frame of each video, having on average 32 frames per video, although some are longer or shorter. The video was captured at 12 FPS, resulting in each frame spaced timely by 83ms. The size of each image is also also varies a bit depending on the specific sample. Each image has exactly 100 pixels of height, however they can have 100 to 160 pixels of width. The dataset is divided in a train set, validation set and test set in the respective proportions: 80%, 10% and 10%. The labels for train and validation sets are provided, however the labels for the test set are purposefully not present, since their own website ranks approaches based on sent labels from this test set.

4.1.2 The nvGesture Dataset

This dataset contains gestures performed indoors in a car simulator by a driver, with a camera located in the center panel of the car (where the radio controls usually are). Its intended use is to allow the driver to perform touch-less gestures as human computer interface. The dataset contains, similarly to Jester, many classes of gestures however there are some main differences on how they are presented:

- The angle of capture of the camera, which is slightly off centered and from a lower point of view, if compared with Jester;
- The video contain large portions of a driving simulation, only a small part of the sample (exactly 80 frames) represent the gesture being performed;
- There is no specific class for "No Gesture".
- There is far less samples to learn from (1532 in total compared to the 148k from Jester);
- There are not only RBG frames, but also depth and stereo-IR sensors.
- The frames were captured at 30 FPS (33.33 ms between frames).

This dataset is only going to be used in the validation section, in order to evaluate the capacity of the network to be applied with different gestures and in a different environment. For this process, we are only going to use RGB frames, since that is in line with the proposal of this work of not requiring additional equipment. With a transfer learning process, a network pre-trained on Jester will be adapted to fit the data from nvGesture. Due to the low amount of samples, the overfit problem was more evident when training with this dataset, i.e. the network had the tendency to over tune the parameters of the network to maximize an accuracy in the train dataset which did not translate to improvements in the validation dataset.

In addition to that, since the dataset have longer samples, which do not only encompass the gesture itself, it can be used to simulate online operation and evaluate the accuracy and delay of the network when predicting the gesture.

This dataset does not name any of its 25 gestures, so list of gestures will not be presented for this dataset. An example of what a gesture look like from of a full sample is shown in Figure 4.2. This is represents the portion of the video containing a gesture being made, there were frames of video before and after the gesture when the subject was simulating driving movements. This dataset is distributed by NVIDIA [27], which allows the use of it for academic purposes.

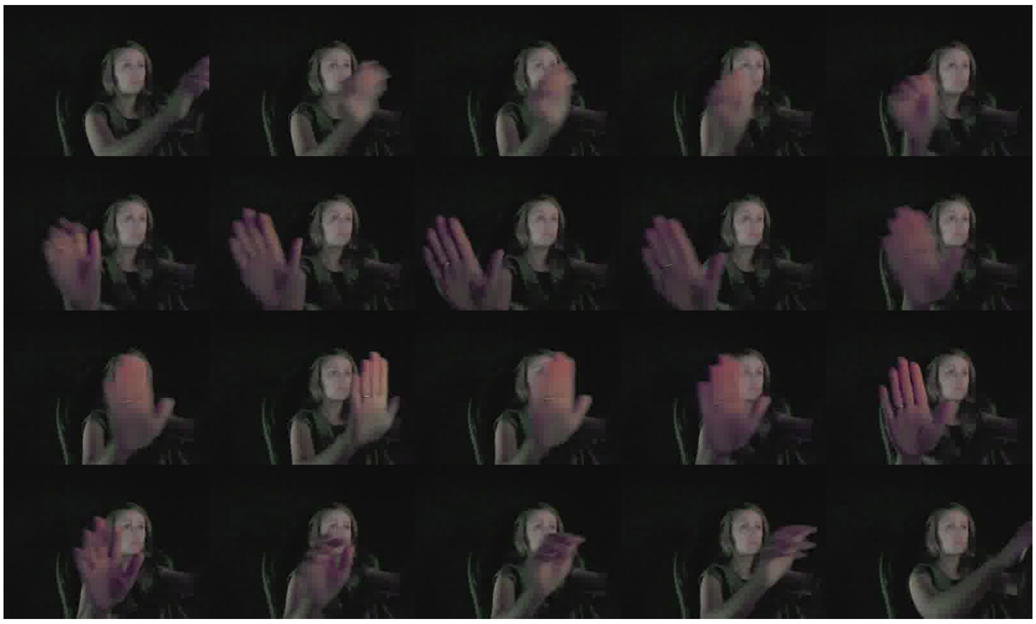


Figure 4.2 – Sample of a gesture from nvGesture dataset (handpicked frames for demonstration purposes)

4.1.3 The EgoGesture Dataset

This dataset consists of 83 classes of dynamic or static gestures, containing in total 2081 videos and 24.161 samples of gestures (almost evenly split among all video classes), the samples contain also depth frames, which are not utilized in this work. Each video contain 9 to 14 gestures being performed in a single shot. The camera utilized for the capture of the sample was located in the users head, resulting in a first-person point of view, the videos mostly contain a background (indoors or outdoors, static or dynamic) and the user's hand in the first plane making the gestures.

Since the model utilized in validation tests for this dataset is pre-trained on the Jester Dataset, some differences between the two are listed bellow:

- The angle of capture of the camera, which is from a first-person perspective, and except for its hands, does not capture the rest of the user in the frame;
- The video contain contains more than one gesture per sample, with moments in between of no gesture being performed;
- There is no specific class for "No Gesture".
- In addition to RGB frames,there are also depth frames (not utilized in this work);

- The background can be dynamic, ,i.e. with part of it moving or with the camera moving;
- There are more classes of gesture, and proportionally less samples per class.
- The frames were captured at 30 FPS (33.33 ms between frames).

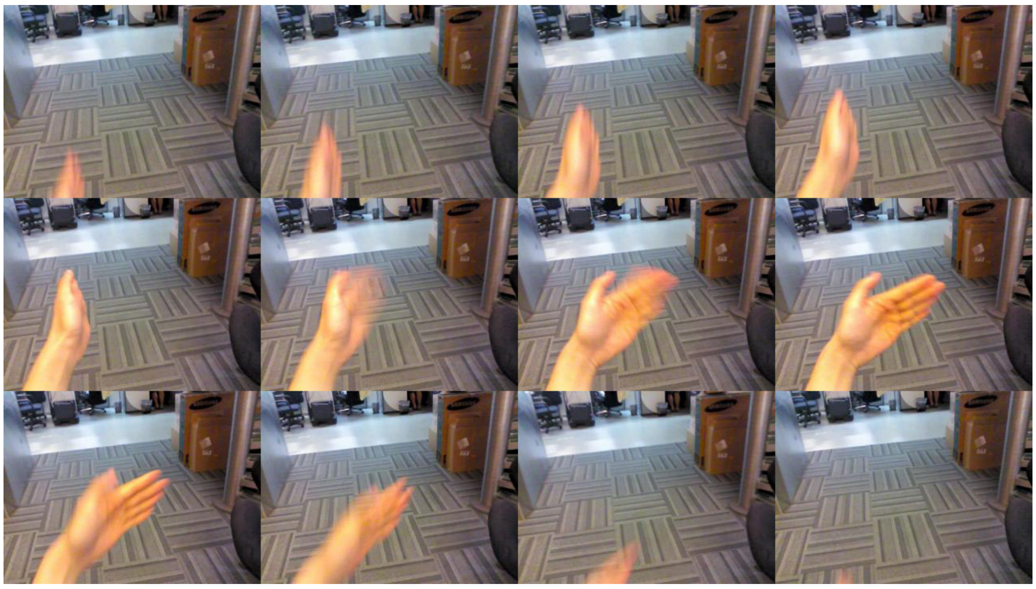


Figure 4.3 – Sample of a gesture from EgoGesture dataset (handpicked frames for demonstration purposes)

Figure 4.3 shows a sample of the video, demonstrating the point of view previously mentioned. A diagram showing each one of the 83 gestures is shown in ATTACHMENT A.

This dataset was created focused in gesture classification from continuous data. From the datasets mentioned, it represents the best the types of samples which online classification techniques would have to classify in a real-time application, due to its longer samples and multiple gestures per sample.

This dataset is distributed by Yifan Zhang [52], and the database is released for research and educational purposes.

4.2 Network Structure

In this work we present three versions of online capable gesture recognition networks. All of them are using 3D convolutional networks as it core components in the network, due to 3D CNNs being present in most of the state-of-the-art techniques for both offline and online video classification. Two of the networks presented in this section are essentially 3D CNN networks (RT3D_8F and RT3D_16F), containing layers of 2D convolution, 3D convolution and a liner layer for the final classification.

Based on the success of optical flow in some of state-of-the-art techniques that achieved the highest accuracy in offline gesture classification datasets and the fact that new lighter implementations of optical flow were developed recently, this section presents a third network (RT_FLOW), a variation of the RT_RGB-16F network, which in addition to RGB frames, has an additional component in its model to generate flow frames that are fed to the network. Since the use of OF is often prohibitive for real-time applications, the RT_FLOW model is an attempt to check if these lighter implementations of OF can translate into benefits for the network's accuracy and what impact it may cause on computing performance overall.

4.2.1 Models RT_RGB-8F and RT_RGB-16F

The general structure for all three networks can be seen in Figure 4.4 .The network consists of basically 3 major steps: Reducing the dimensionality of the frames using 2D CNN layers, extracting spatial temporal features with 3D CNN layers and use this features to make a prediction with linear layers. One of the biggest challenges of working with a neural network that can perform online video classification is the amount of data needed to be processed by the network at a given time, due to the complexity of the techniques used.

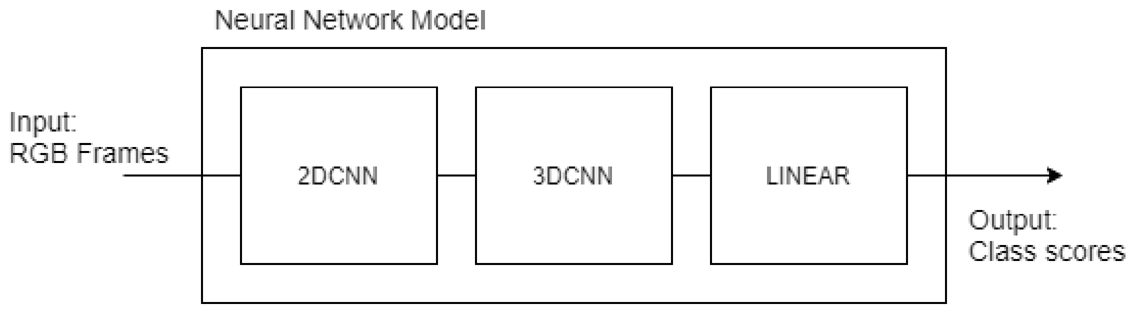


Figure 4.4 – High-level block diagram for models RT_RGB-8F and RT_RGB-16F

Since one of the objectives of this work is to create a light (low on resource usage) neural network capable of gesture classification, reducing as much complexity of the network as possible is necessary in order to make these networks usable in simpler hardware. And for this network, the 3D CNN layers are the part of the network that requires the most parameters and computing time in order to perform properly. With that in mind, it was opted to pre-process the images with a 2D CNN before feeding into the 3D CNN. This design choice allowed to include some pooling layers while extracting spacial features, reducing the size of the data without as big of a loss of information that pooling only operations would result, while allowing the use a of simpler 3DCNN network inside the model.

The use of pretrained models and transfer learning was studied, and it was opted to use a custom structure for the convolutional layers due to satisfactory results being achieved with very simple network structures in comparison to more complex and general models commonly used as a starting point (e.g ResNet, Inception). One of the reasons of why it was opted to train convolutional layers from scratch is due to how general purpose networks have the tools available to discern various types of features in a image, while for a gesture dataset most of the actual images from frames of different gesture look alike, and a general purpose network would probably be underused for a niche task like gesture recognition. Due to this classification problem being limited to what the network has of visual feature to distinguish, training a network from scratch seemed a more efficient approach rather than sub-utilizing a more powerful network.

Model RT3D_8F

For the simpler network (RT3D_8F) only 8 frames from the original samples from Jester dataset are used, which results in skipping 3 out of every 4 frames. Considering that the original rate of frames per second (FPS) for the dataset is 12 FPS, this model operates with images sample at 3 FPS (since it only uses a quarter of the frames). This value is taken into account when utilizing this model with datasets which are captured in a different frame rate. The frames are selected starting on the first frame and skip 3 frames to select the next one, and in the case of not having enough frames to represent the data using this method, the final frame is repeated. In addition to that, the images are center cropped in a $[96, 96]$ size (96 pixels of width and height). This choice allows every sample to be cropped in a smaller size than its original size, and the number 96 is used instead of 100 due to the first being divisible by two multiple times, which, in the pooling operations of the network, avoids the need of padding to complete the operation.

In this first 2D CNN there are 5 layers, with progressively increase the number of channels and progressively pool the original image in order to reduce its size. Figure 4.5 shows the transformation

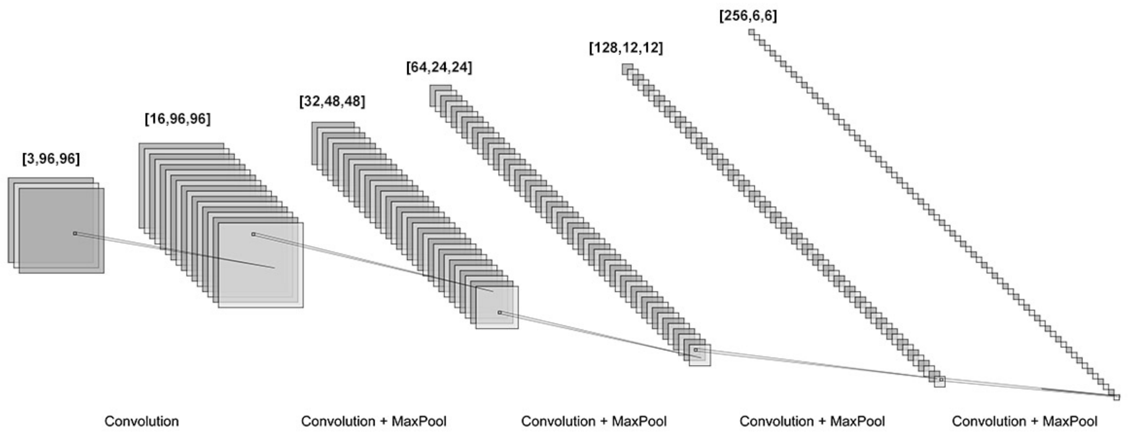


Figure 4.5 – Representation of data shape throughout the 2D CNN in RT_RGB-8F

that each single frames passes at each layer, starting with an RGB image (3 channels) and progressively increasing the number of channels and reducing the width and height with pooling operations. Other works with CNNs [17] frequently use similar structures of progressive pooling in multiple layers. All kernels in this network are of size $[3,3]$, as shown by [37] that multiple layers with smaller kernels ($[3,3]$) have a similar effect as having a bigger kernels, resulting in the capability of detecting more complex features. Since the input of this first layer are 8 frames (equally spaced) from the original video source, therefore the inputs in this layer consists of 8 frames of shape $[3,96,96]$ (3 color channels, 96 pixels of height, 96 pixels of width). After all the convolutional layers and pooling layers of the 2D CNN section of the network, the dimensions of the tensor referent of each of the original 8 frames will be the following: $[256,6,6]$.

The next step in the network is the 3D CNN, the core of the network. It is the part of the network which requires the most parameters (due to its multiple tridimensional kernels) and it is responsible for extracting the spacial-temporal features of the data. Until this point, each frame of the video was individually processed, i.e. the convolution process of the layers is individual for each frame. Before entering the 3D CNN part of the model, the data is concatenated into a 4D tensor of shape $[8,256,6,6]$ (the 8 represents the number of frames), and after this point the data from different frames is fused. There are 3 layers of 3D CNN, in which all kernels are size $[3,3,3]$ and pooling is done progressively. Pooling is done not only on the height and width of the image, but also across frames of the image. Pooling in different frames mean that pixels from different frame are compared and combined in order to reduce the number of frames, similarly to what is done in height and width.

For the 3D CNN, the number of channels is kept the same for all layers (256, the same number from the 2D part of the network). The output result of each kernel from the 3D CNN contains more meaningful data, for example, each value can be representing shapes (hands, fingers, arms), movement (left to right, upward, etc) or even specific motions such as a closed fist opening its fingers. It is possible to try to decode which features the network learned during training, as some work has demonstrated in [50], and thus seeing what each feature is likely trying to identify in the image, however this would be a merit of a different work itself due to the complexity involved.

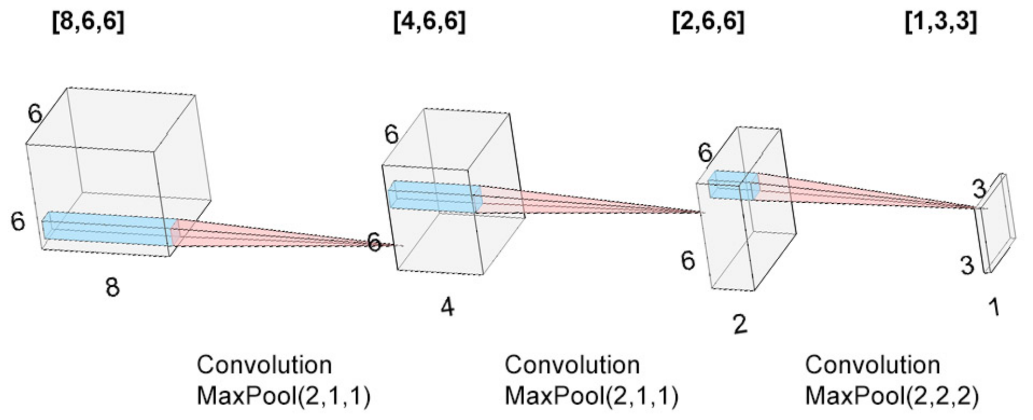


Figure 4.6 – Representation of data shape throughout the 3D CNN in RT_RGB-8F

In order to show the tensor transformation (similarly to what was shown for the 2D CNN in Figure 4.5), it was opted to show the transformation of only one of the 256 channels in the network, since the number of channels does not change throughout the layers and the representation of a 4D tensor would not yield a good representation in a figure. Figure 4.6 shows the transformation of the tensor relative to one of the 256 channels. The value on top of each tensor is the shape of the data for each point. The name at the bottom, between each transformation refers to the operations made to the tensor.For the case of a pooling operation, the value in the brackets is the shape of the pooling, e.g. a value of $(2, 1, 1)$ represents pooling in the depth dimension (the number of frames) reducing its size by a factor of 2, while leaving untouched the height and width of the tensor. The first tensor of shape $[8, 6, 6]$ represents a stack of the 8 frames of the $[256, 6, 6]$ channels generated by the 2D CNN. Note that, as mentioned before, the actual shape of data at this point is $[8, 256, 6, 6]$, however only one of the 256 channels is shown in the figure. As the tensor passes through the layers of the network, the pooling operations reduces the size of the shape of the tensor to $[1, 3, 3]$; or considering the other channels: $[1, 256, 3, 3]$.

At this point, the network was able to transform the values of pixels into meaningful features, the next step is to feed this features to a fully-connected network in order to make a class prediction. The data that is output from the 3D CNN is the shape of $[1, 256, 3, 3]$, at this stage the tensor is flattened (i.e. lost its shape to become a 1D vector) to pass to the linear layers (which do not accept multi dimensional data). The general concept of how neural networks work (specially focused on fully-connected architectures) is reviewed in Section 2.1. The number of parameters in this point of the network can be calculated by multiplying each value of the shape of the network: $256x1x3x3 = 2304$.

For the classification part of the network, there are two main decisions to be made: the number of layers and how many neurons will be put in each layer. As many other implementation decisions in neural networks, there is not a single right way to choose those parameters based on the literature. Most decisions are based on what other successful works used and through experimentation. About the number of layers, according to [10]: "A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly". For this work it was experimented with 2 and 3 layers, opting to use 2 with RT3D_8F due to unnoticeable gain of performance when using 3 layers. The decision on the number of neurons was chose to create a progression from the number of parameters that the 3D CNN output (2034) and the number of classes (27), having a linear layers in between with 1024 neurons. These layers also contain batch normalization steps and $ReLU$ as its activation function.

Model RT3D_16F

As mentioned before, this network is just a variation on the previously shown model, adapted to work with 16 frames instead of 8. In addition to that, it was noted that the cropping of the original frames to a $[96, 96]$ shape could be affecting the performance of the network, since some of the samples from the Jester dataset were leaving relevant information (the arms and/or hands of the subject) outside of the cropped version in a small sample of the analysed results. It was opted for this model (RT3D_16F) to have in addition to more frames, slightly wider images as its frames (of shape $[140, 100]$). The reason for developing this variation on the first network was to observe the increased gains in accuracy and compare that to the penalty in performance resulted from running a more complex model. Although this model is more complex than the previous one, it is still light on resource usage if compared to the techniques presented in Section 2.2, and possibly viable for usage in embedded systems (results sustaining this argument will be presented later in Section 5.3).

To increase the complexity of the network, it was opted to add more layers to both the 2D and 3D CNN networks (which adds more kernels to the network, and therefore the capacity of detecting more complex features) and reducing and delaying the pooling operations done to the tensors (increasing the amount of data needed to be processed in the network). The overall designing of the network is the same for the model RT3D_8F shown in Figure 4.4, while the main differences are inside both the 2D and 3D CNNs.

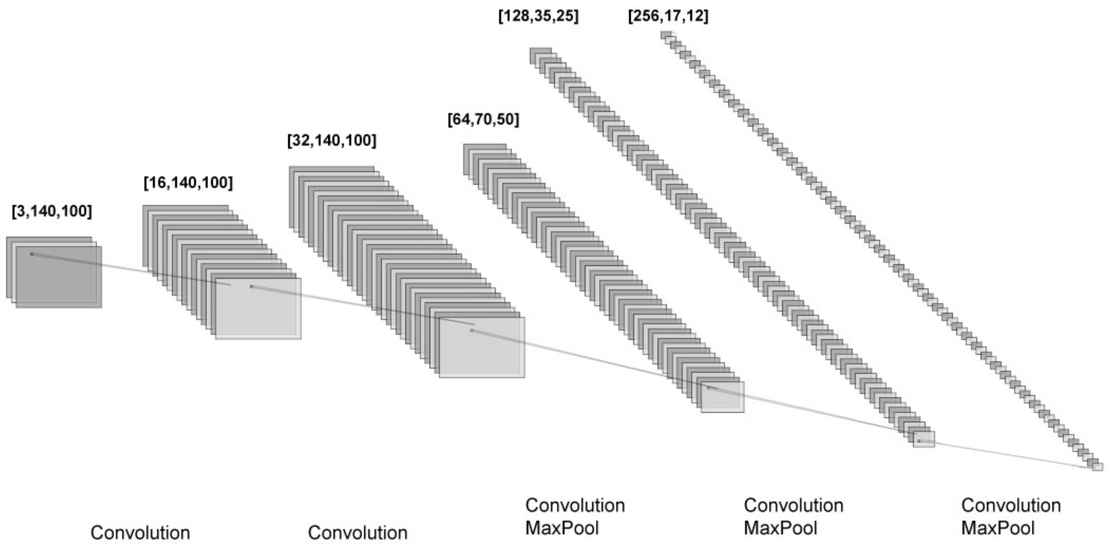


Figure 4.7 – Representation of data shape throughout the 2D CNN in RT_RGB-16F

Figure 4.7 shows the tensors shape during the 2D CNN layers of the network. The main difference between the RT_RGB-8F model layers, is the size of the original frames, which is bigger, and the lack of pooling in the second layer of the network. As a result, more data will be passed to the 3D CNN portion of the network (16 frames of shape $[256, 17, 12]$, instead of 8 frames of shape $[256, 6, 6]$).

Figure 4.8 shows the tensors shape during the 3D CNN layers of the network, considering only one channel of the tensor. In this network, it was added an additional layer of 3D convolution, and the pooling operations have changed been changed to reduce the size of the data in later layers of

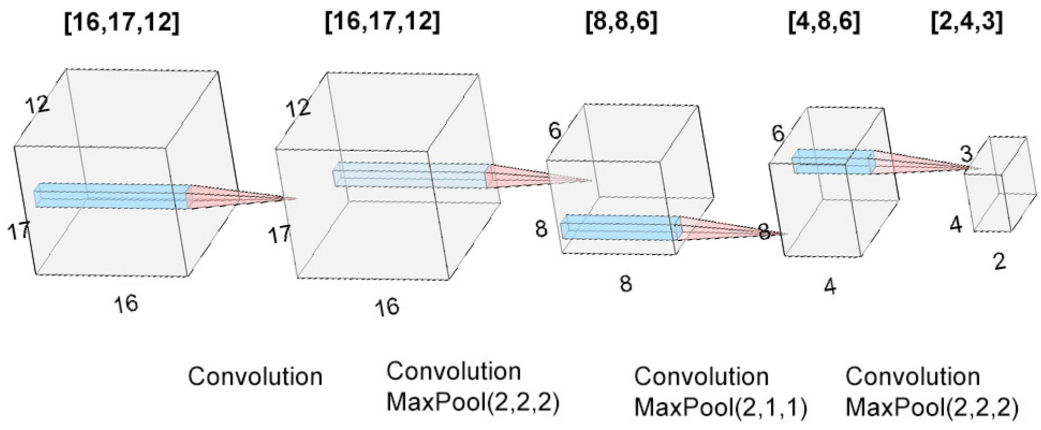


Figure 4.8 – Representation of data shape throughout the 3D CNN in RT_RGB-16F

the network. The result of this portion of the network is a tensor of shape $[2, 256, 4, 3]$, which when flattened results in a vector of 6144 values.

The linear portion of the network contains one extra as well, from 6144 to 3072 neurons, then to 1024 and finally to the 27 for the class scores.

4.2.2 Model RT_FLOW

This variation of the network structure is inspired by two-stream models, having a RGB image processing path, and a flow path, however, instead of combining both with late fusion at the end in order to make a prediction, the RGB and flow frames are concatenated at the start of the network, following, then, a similar structure from RT_RGB-16F. A block diagram illustrating the network structure is shown in Figure 4.9. In order to use flow frames in real time, it was needed to find a light implementation of it, since most optical flow implementations are computationally expensive, rendering a real-time implementation impractical or requiring special hardware to run it. For this work, the implementation used is called PWC_Net [38], provided by NVIDIA. This implementation is lightweight and a order of magnitude faster to run than FlowNet [9] and DCFlow [48], and it presented state-of-the-art results in datasets as "Flying chairs" and MPI Sintel (datasets commonly used to evaluate optical flow algorithms).

The contents of both the 2D CNN and 3D CNN are the same as shown in Figures 4.7 and 4.8, with the only difference being in the 2D CNN, which starts with 5 channels instead of 3, being 3

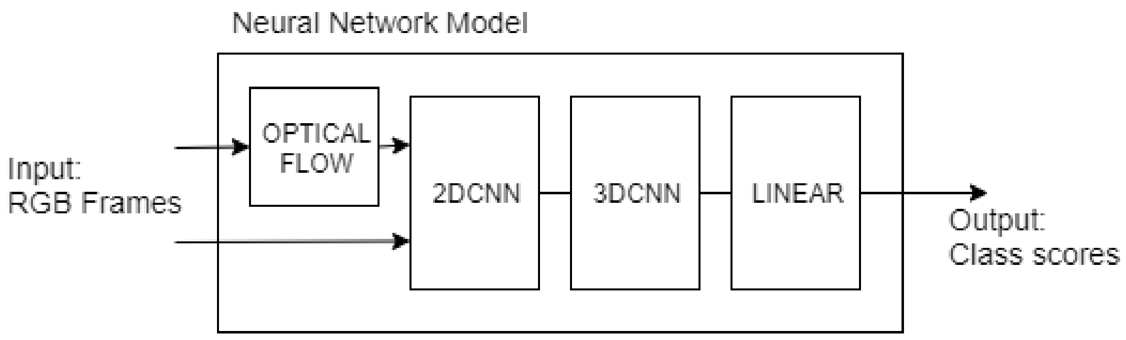


Figure 4.9 – Representation of data shape throughout the 3D CNN in RT_RGB-16F

from the original image (red, green and blue) and the other two from the flow frames (horizontal and vertical movement).The benefit of this implementation in online operation is that the output of the 2D convolutional layers can be reused for the next cycles. Since the data only fused data from RGB and Flow frames, and not from adjacent frames, the same computation of the 2D CNN layers would be repeated each cycle, making possible to skip this step for old frames, while only computing the flow frame itself and the 2D CNN layers for the new RGB frame and flow frame pair.

The conception of this network was inspired by three main factors: the success of flow frames in action recognition, the possibility of reusing flow frames between cycles and the effectiveness of light convolutional networks to process video inputs in this formats.

There are two main challenges with this design of network: the first is checking whether the flow frames generated from a lighter implementation of optical flow can actually improve the network's capacity to detect gestures and if so what is the overhead in computing performance to do so.

4.3 Network Training

In this section we will discuss how the training on the network was done, specially mentioning which measures were taken in order to avoid overfitting and also which general parameters were used during development.

4.3.1 Data Augmentation

As stated in Section 2.1.2, having a small amount of samples to train a network might result in overfitting, while that is not a specific case for the Jester dataset, which has a considerably high amount of samples per class, the more samples to learn from, the better are the chances that the network will be able to generalize its learning examples to samples from outside the dataset. Data augmentation refers to techniques to artificially increase the number of samples that a network is going to see, by applying modifications to the data (in our case, images from videos) to make them different each time they are fed to the network. Some examples that can be applied to images are: random cropping, random zooming, flipping horizontally/vertically adding blur, etc. The intent of these modifications is to keep the information contained in the data the same while changing the way it is presented to the network. This helps during the training process making the network focus on developing useful features which are not sample specific. If a certain feature fails to classify the same sample video with some modifications, it is probably not useful, and it will be eventually replaced for something that helps to minimize the loss function by a greater amount.

For this work two data augmentation techniques are implemented to help to prevent the network from overfitting during training: random cropping and random rotating.The Figure 4.10 shows how these two transformations modify the frames fed to the network during training. Image "a" shows the original image, of size (176, 100), figure b) shows a random crop of the image and figure c) show the the rotation of the cropped part (inserting black pixels where the image is no longer present). Both the transformations are applied in the same manner for all frames of a single sample, i.e. all frames are cropped and rotated together. For the random cropping, it was imposed a random rotation from -20° to +20°. Higher rotation ended up cropping too much from the original source, hindering predictions on that information. For the random cropping, since the source image is larger (see Section 4.2)than the used input for this network ($[96, 96]$), during training a random crop is applied in the source image, with the correct dimensions needed. When not training the network, the validation set is evaluated using no rotation and center cropped.

Some common and simple transformations used in similar work were not possible to be used in this dataset due to ambiguity that it would cause depending on the gesture. As seem in

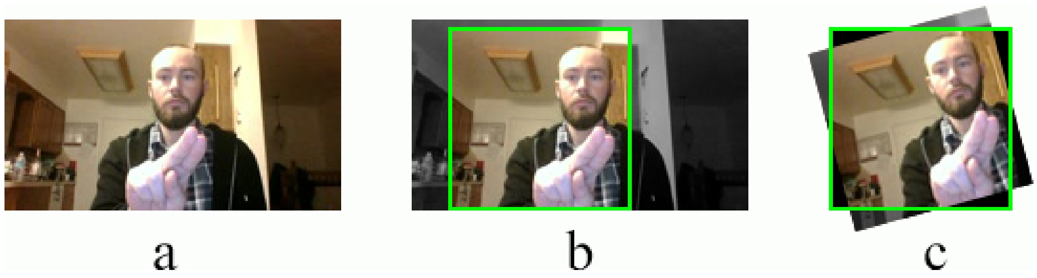


Figure 4.10 – Data Augmentation for the Jester Dataset. a) Orignal Image, b) "Randomly" cropped image and c) "Randomly" rotated Image

Section 4.1.1, some gestures can be performed as a mirror counterpart of another (e.g. Swiping Left and Swiping Right). In these cases, a horizontal flipping would make the samples presented to the network indistinguishable even for a human, inducing the network to create sample specific features to distinguish the two. Vertical flipping, even though not as drastic as horizontal flipping in the cases, can cause problems similarly. In the case of "Swiping Up" and "Swiping Down" it is possible to distinguish one from a vertically flipped, other (since the head of the user would be upside down), however this would hinder the ability of the network to associate "upward" or "downward" motion to some labels. It was not opted to use vertical flip due to possibly being prejudicial to the training process. Applying a zoom to the image was also opted to not be used, specially because since the network already crops part of the network, any more cropping could cause it to produce unrecognizable gesture by letting out important parts of the image.

4.3.2 Training parameters

Dropout amount

The first parameter mentioned here is relative to the dropout layers. These layers are introduced in the network during training, deactivating some neurons (i.e. transforming their output signal to zero) in order to stimulate the network to create many paths to a correct response, instead of relying on one strong feature. The parameter in the dropout layer is a probability that a certain neuron will be deactivated (from 0 to 1). For this work, dropout layers are used only in between the linear layers, with a dropout probability of 70%. Although is possible to use dropout layers anywhere in the network, they are more commonly seen in linear layers, and thus it was opted to make this use. The bigger the chance of dropout, the less likely a network is to overfit, however a big enough value might hamper the training process. For this work the values of 50% and 90% were experimented, and the mentioned value was observed to help with overfitting by observing train and validation accuracy without any observed train performance loss.

Loss Calculation

As mentioned in Section 2.1, the loss function is a design choice. For this work, it was opted to use Cross Entropy as the loss function, which is shown in Equation 2.2. It was attempted to make a weighted loss function, giving some classes a higher importance than others in an attempt to improve accuracy in the worst performing classes. This attempt would train a network focused only in some classes in order to be run in conjunction with the current one. The results did not show any improvement in the classification of those classes, while increasing significantly the complexity of the network, therefore it was opted to leave this part out of the final design.

Optimiser, Learning rate and Momentum

In order to train the network, the Stochastic Gradient Descent (SGD) was used. Other optimiser were not heavily experimented with. The learning rates used started at 0.001 for a randomly initialized model, decreasing by tenfold every time the loss was not decreasing anymore (about 50 to 100 epochs depending on the dataset). It was used momentum of 0.9, which means that the gradient that will adjust the weights of the network is summed to the gradient of last step with a weight of 1 and 9 respectively. With this, at each step the network will have adjust its weights roughly at the same direction, since there is a component from the previous step's gradient. This helps the network to train in the "general direction" that the samples are pointing, instead of going exactly to a direction that the current sample would be optimized. These values of learning rate and momentum were chosen based on common practices in the field, such as in the works of [22, 19, 37].

Stop train criteria

Each model during experimentation was trained for a maximum of 4 reductions of learning rate, each one with 50 to 100 epochs, adding up to 200 epochs per training of a certain network. In addition to that, training loss, validation loss and accuracy was also observed for the majority of the networks. If it was observed that a certain learning rate was not being effective anymore in reducing the calculated loss, instead of waiting to finish the 50 epochs, the learning rate is reduced immediately. The training loss is observed in real-time during the training process, and with the aid of a filter (due to high variance of loss within the steps), it is possible to visually inspect the progress, or lack of, of the model performance. This was done using the library tensorboardX [44], and its graphs for accuracy and train loss can be seen in Figure 4.11.

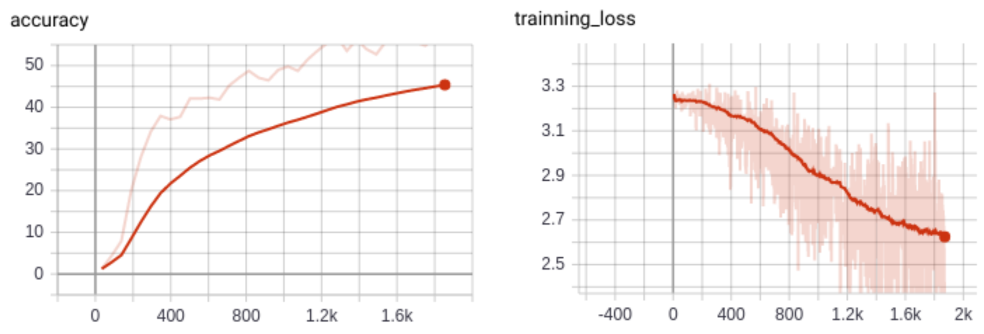


Figure 4.11 – Observation of accuracy and train loss during training of a model

4.4 Modifications towards online operation

For this work, we will consider the video data as a sequence of images instead of an encoded format, although the latter can be explored in a different work. In a real-time application, we consider a given video input with a certain frame rate, yielding that many images per second. In order to recognize gestures, a certain window frame have to be analysed by the network to classify as a gesture or not. The hand gestures in the Jester dataset [43] have a duration from 1 to 2 seconds on average, meaning that frame window around this duration should be able to identify all the gestures for this dataset.

One approach to gesture recognition would be to wait until N video frames are acquired and make a prediction on the given gesture of the interval, and after that start the window over, sharing no frames in common. One problem to this approach is the delay introduced between one window and the next one, if the window represents a high enough interval (a second, for instance). In the case of a smaller window frame, a gesture might not be able to be able to be entirely represented inside the window frame, impairing the gesture recognition capabilities of the network. In addition to that, it might be the case that a gesture starts at the end of a window frame, and ends at the start of the next one.

Considering a moving window of N video frames, i.e. whenever a new frame is acquired and the oldest one is dropped (first in, first out), each consecutive window will share $N - 1$ frames in common with the previous one. For this scenario, a prediction can be made for every new frame acquired by the video input, and the moving frame will guarantee that a given gesture will always be able to have all its frames inside a window for prediction. The main disadvantage of this approach is the number of prediction that have to be made per second, making the necessary computing power significantly higher.

The following sections present the modifications implemented towards rendering the approach to operate in real-time mode.

4.4.1 Data Reuse

As mentioned before, in a moving window input scenario, most of the video frames form the previous window will be the same, with the exception of the new ones. Being able to reuse some information about those frames in the next processing cycle will save computing power and allow a new prediction to be made in less time, therefore reducing the delay of the prediction and the needed computing power to execute the operation.

In order to reuse information from one frame to the next, we opt for a late-fusing strategy in the neural network, processing the frames individually for the most part of the network, only fusing the data of different frames in the latter layers of the network. This strategy allows a significant portion of the data to be reused, while for each cycle only the processing of new frames and the final layers have to be made. The two main structure that are discussed in this work are 2D CNNs and the creation of flow frames, which both can be processed individually for each new frame. That means that each frame processed by the 2DCNN contains data only from a single frame, and therefore if it was to be reprocessed next cycle, the information would be the same, allowing to save this output of the 2DCNN and only process the output of the 2DCNN for the new acquired frame. This process cannot be done wiht the 3DCNN due to its output be a result from all input frames, and therefore the output with the new acquired frame will be significantly different.

4.4.2 Online operation

In this section we describe the main differences in the network implementation related to online operation. There are two main aspects to be noted in this section, the first being how the data is interpreted in an online application, i.e. how the model output, which was trained in a offline environment, will be utilized to yield a usable gesture recognition application in continuous online operation. The second aspect noted is how data is processed and what information needs to be kept in memory for the next cycle of operation.

Continuous Online Classification

During offline testing, a sample small enough to be fed to the network was presented, and the network, then, calculated which label best represented the sample. On the contrary, during online operation a moving window of frames yields a new output from the network at each new acquired frame, which may or may not yield a detected gesture "label" at that point (e.g. the user in front of the camera it is not performing any gesture). As mentioned in Chapter 3, the network should only yield a label once per gesture performed. To illustrate this process, Figure 4.12 shows the output of the network for a gesture sample (from the nvGesture dataset) containing 47 frames. The first 15 frames of this sample were passed through the model, in order to first fill moving window of frames, which supports 16 frames as its input, and thus further in this paragraph we will only refer to the analysed frames. Each of the lines in the graph represents the probability of the network for a given label (based on a softmax operation on the class scores). The brown line, which is predominant at the start and towards the end, is the "No Gesture" label, while the blue line, which peaks around frame 25, is a gesture labeled as "Gesture 1" on the dataset.

This sample chosen represents a long time with no gesture being performed, at some point "Gesture 1" is performed and towards the end of the sample, the user already finished the gesture and goes back to a neutral position.

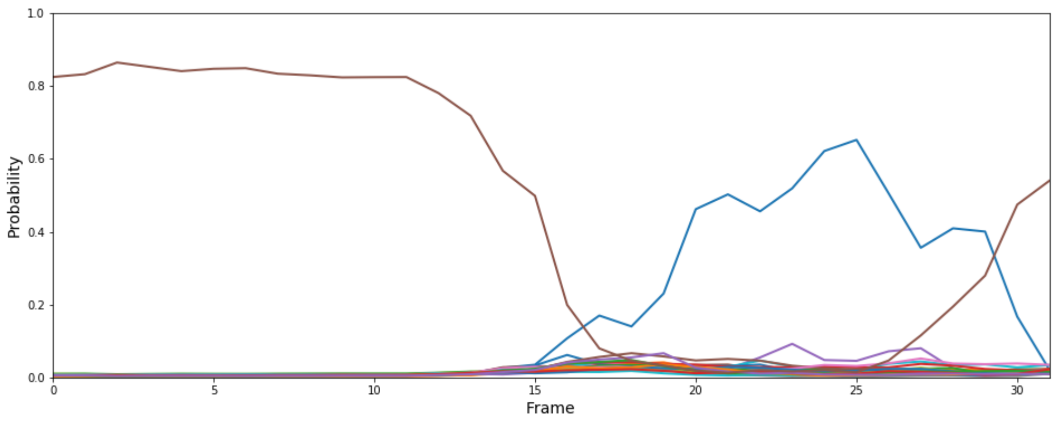


Figure 4.12 – Predictions for a moving window of frames of a sample from the nvGesture dataset

In order to simplify the this example, all the other lines other than the ones representing "No Gesture" and "Gesture 1" were removed from the image in Figure 4.13. Also, a green background was added between frames 9 (start) and 25 (end) to show were the gesture was actually being performed. This sample's correct label is "Gesture 1", however the gesture only starts being performed at frame 9, and the objective is to yield a "Gesture 1" only once during this period, and preferably before the gesture actually ends at frame 25.

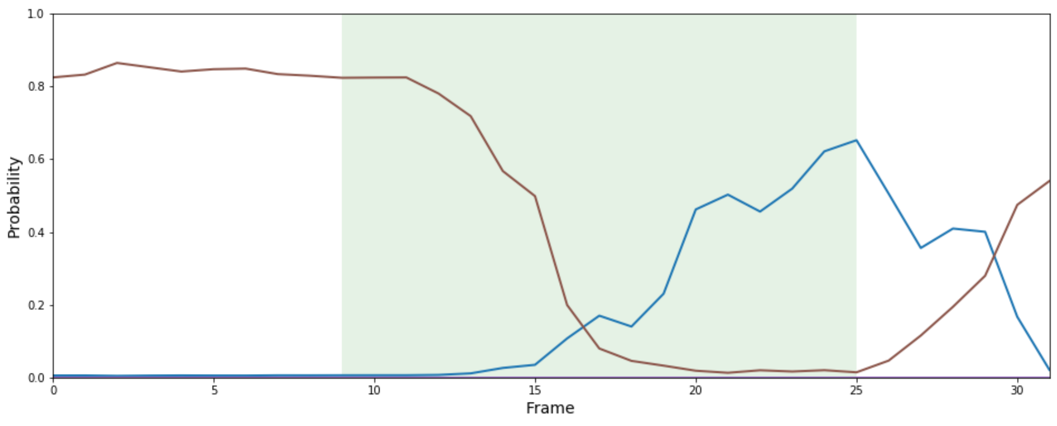


Figure 4.13 – Predictions for a moving window of frames of a sample from the nvGesture dataset

The online detection and classification is done with the post-processing of the trained models from offline training, i.e. multiple outputs from the network will be analysed in order to yield the continuous online classification. The application saves N predictions from consecutive frames from the network, i.e. for $N = 3$ the application should remember the predictions from the current and two most-recent outputs from the network, and average the probability of the predicted class for this period. If the averaged probability of the highest scoring class is above a determined confidence threshold C_{th}, the network will yield a prediction of that class. In order to avoid yielding the same prediction in the next cycle, there is a "cooldown" system, which prevents the network from yielding a prediction for T frames, where T is a chosen parameter, representing the number of frames after a prediction in which the same gesture cannot be predicted again.

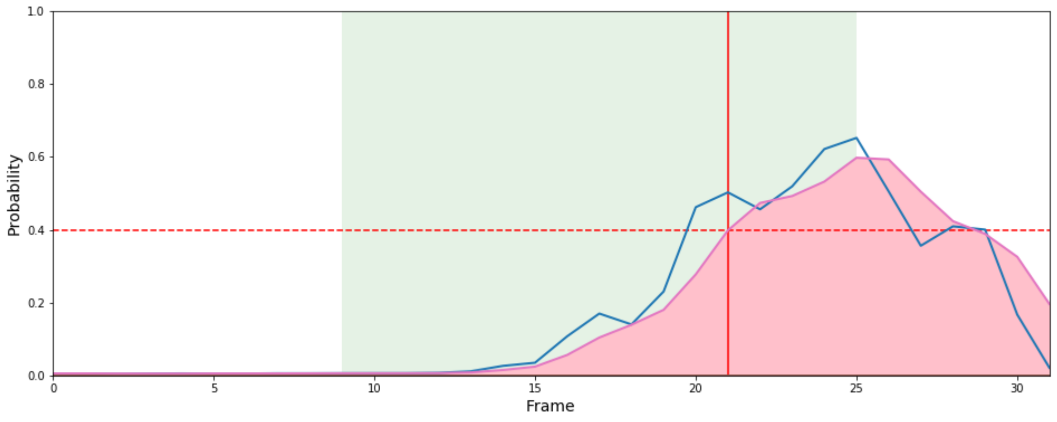


Figure 4.14 – Averaged signal (pink) of the probability of a gesture (blue) surpassing the threshold (dotted line) and yield a label (vertical line)

In Figure 4.14 the same sample is illustrated, this time omitting the "No Gesture" label, which is ignored in the application, and adding a line representing the averaged value of the value for "Gesture 1" for 3 frames ($N = 3$), in pink with a filling. The dotted horizontal line in red represents the confidence threshold C_{th} set to 0.4 (or 40% probability). The red vertical line (at frame 21) represents the moment in which the network will yield a label "Gesture 1", due to the averaged value of its prediction surpassing the established threshold. Note that in the absence of a cooldown system, in the following frames (22, 23, 24...) the value of the observed averaged probability is still above the threshold, which would yield a new label for the same gesture. In this image the cooldown was not represented in order to keep the visualization simple, however it is possible to notice that a T value of 10 frames would be enough for this sample to avoid yield the label twice for the same gesture. Of course, the other probabilities for the remaining gestures were hidden for visualization purposes, however in reality the highest scoring class (the one with the highest probability) is being compared to the threshold at any given sample.

Flow diagram for the presented technique

In order to reuse the data, some layers of the network must not fuse information from multiple frames. That allows to "skip" some of the steps of the network for frames present in the previous window. Some other implementations that have more complex pre-processing on individual frames (Or every two frames, like optical flow) can benefit significantly more from this approach. Data from individual frames can be saved for use in the next cycles.

In the application level, the system should set the acquisition frame rate of the camera to be the same as the dataset (or at least a multiple of it) in order to match the time between frames to what the network was trained. Considering a window of W frames (W could be 8 or 16 for the networks shown in this work), for every incoming frame, the network should pre-process all the steps that do not fuse data from other frames, i.e. 2D CNNs and optical flow computation. This results should be stored in memory and the original frame can be removed from memory once they are not necessary for other computations. Once W frames have been computed, the network can do a proper prediction by feeding all the data to the next steps of the network (3DCNN, Linear). After the prediction is made, the application should discard the data that is not going to be used for the next cycle, and at this point the network is able to perform one new prediction for every new frame acquired, following the rules described above. The flow diagram showing the algorithm used for class prediction in this work is shown in the Figure 4.15.

4.4.3 Summary of the innovations presented

To finish this section, we present Table 4.2, which contains a summary of the innovations proposed in this technique and their respective benefits when compared to others.

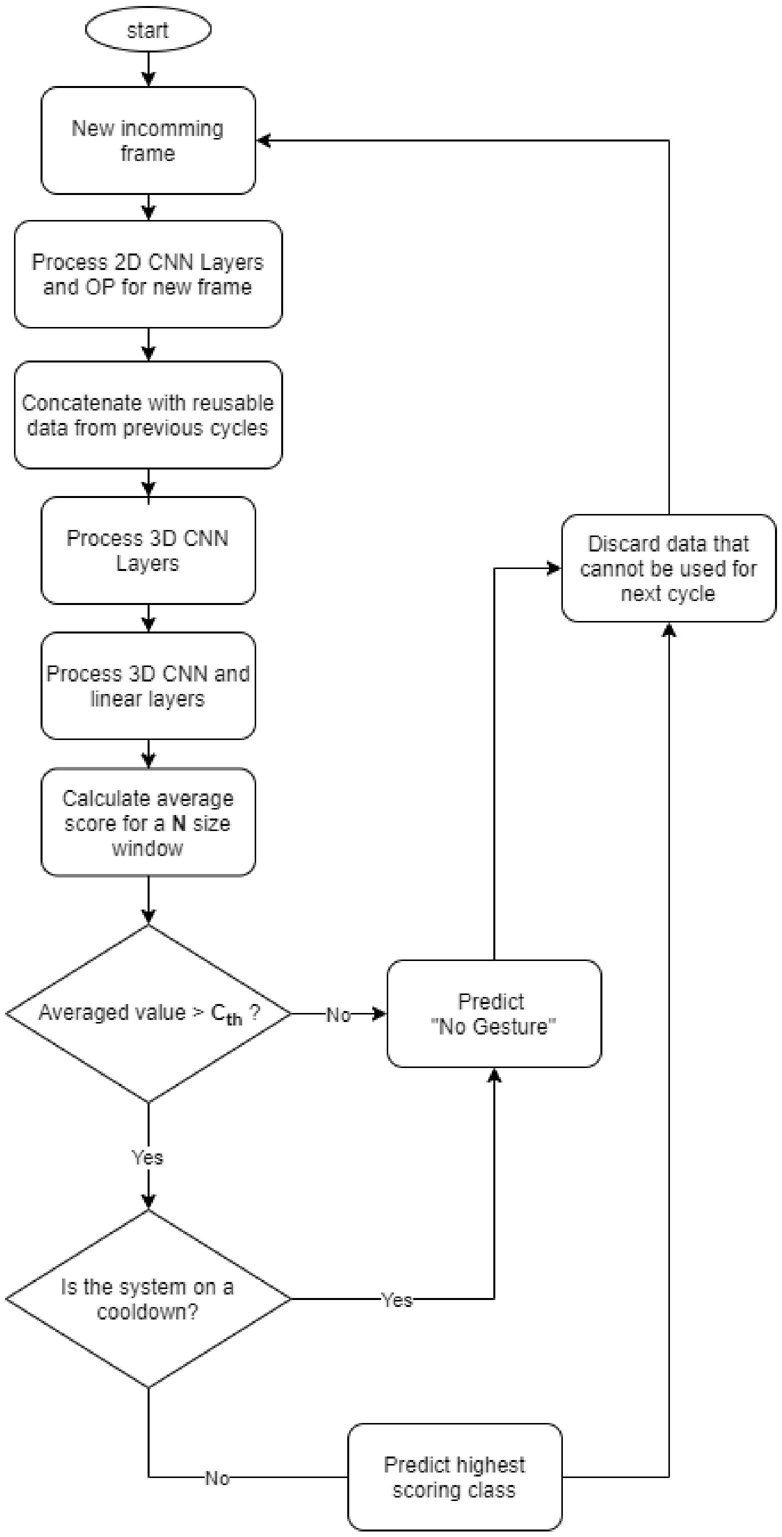

start
New incomming frame
Process 2D CNN Layers and OP for new frame
Concatenate with reusable data from previous cycles
Process 3D CNN Layers
Process 3D CNN and linear layers
Calculate average score for a **N** size window
Averaged value > C_{th} ?
No
Predict "No Gesture"
Yes
Is the system on a cooldown?
Yes
No
Predict highest scoring class
Discard data that cannot be used for next cycle

Figure 4.15 – Flow diagram for the proposed approach

Table 4.2 – Summary of the innovations proposed by our approach when compared to similar techniques

Feature	Advantage
Insertion of a 2DCNN before the 3DCNN to pre-process the original frames	Reduction of the dimentionaloty of the data for the 3DCNN (reducing its complexity) and the possibility of reusing the output of the 2DCNN in between cycles which contain the same frames, saving computation during online classification
Reduction in the frames per second needed to identify a gesture	Reduced amount of data to be processed by network and a longer time frame to process the output of the network for a given new frame
Adaptation from an offline model to an online mode of operation.	The proposed model is simple to implement, with low impact on the overall performance of the system. The cooldown system proposed prevents the multiple computation of the same performance of a gesture.

5. VALIDATION

In this chapter, we will validate the proposed model, by, in a first moment, evaluating the overall accuracy of the network on the following datasets: Jester, nvGesture and EgoGesture. The results of accuracy from offline and online tests will be compared to alternative techniques. The last section of this chapter contains the evaluation of the model in real-time conditions, by checking the gain of performance operating in real-time and comparing results to another real-time network, specifically the one proposed by [18], which share the most similarities to the objective of this proposed work.

5.1 Offline Validation

For offline validation, the main aspect observed in each dataset will be its accuracy on the dataset itself. The Jester dataset was utilized to train all of the initial models, due to its increased number of samples to learn from. The models for the nvGesture and EgoGesture dataset started their training process utilizing the most accurate version of the model trained on the Jester dataset.

5.1.1 Jester Dataset Validation

Due to its high amount of samples, the Jester dataset is the main dataset used for this work. For this offline test, the three models presented in Section 4 of this dissertation were tested. The results from these models are presented in Table 5.1 were tested with the validation set provided by the creators of the dataset. The scores represent how often the correct label was presented in the top-scoring classes, i.e. how often it was correct, how often it was in the top three classes and the top five classes (top1, top3 and top5 accuracy respectively). Top1 accuracy is the actual accuracy of the network, while top3 and top5 represent how close the network was on choosing the correct label.

Table 5.1 – Scores obtained in Jester's validation set

Technique	top1	top3	top5	Model's size
RT3D_8F	90.11 %	97.15 %	98.47 %	32.4MB
RT3D_16F	93.00 %	98.43 %	99.16 %	118.1MB
RTFLOW_16F	91.68 %	98.17 %	98.92 %	103.5MB

The proper test set from Jester's dataset does not provide its labels, instead, a file is submitted with the predicted classes for each sample of the test set. The creators have a proper leaderboard with all user's submissions and their accuracy (top1). The results from other authors in this leaderboard can be seen in Table 5.2. Since the site allows only one submission for evaluation, it was opted to send the best performing model based on the validation set.

Table 5.2 – Score on Jester's test set, as in the official leaderboard

Tecnique	accuracy
MFNet[28]	96.22 %
Motion Fused Frames (MFF) [19]	96.28 %
RT3D-16F-WIDE	92.65 %
20BN Jester System [43]	82.34 %

From these results, it can be seen that adding more information to the network (more frames and a wider field-of-view) did increase the accuracy of model RT3D_16F when compared to RT3D_8F, albeit with diminishing returns. That is, the correlation between the amount of data provided to the network (and thus its complexity to process the data) and its observed accuracy is not linear.

The attempt to use flow frames generated from low computing power techniques did not perform as expected. During all the experimentation with this model, there was no instance where it performed better than the base model it was based on (i.e. the same network, but only using RGB frames). One hypothesis of why the flow frames did not help to make a better prediction is that the models utilized to create the flow frames have good results in synthetic datasets (computer-generated videos). The application utilizing these models to create flow frames from the Jester dataset has to consider not ideal conditions for the capture, such as imperfect lighting, motion blur, and low resolution, which might hinder the final results of the flow frames.

While all the other models represented a trade-off between network's complexity and performance, the last one, utilizing flow frames, was discontinued from further testing, since it did not perform better than the simpler models, while presenting increased complexity and overall computational time.

5.1.2 nvGesture Dataset Validation

This dataset was trained using transfer learning from the Jester dataset, i.e. the network that was trained for the Jester dataset was utilized as a starting point for this dataset. This allows not only for a quicker learning process (since fewer parameters were changed) but also helps to reduce problems such as overfitting. One of the biggest differences when training for this dataset is the reduced number of samples. If the same training procedure used for Jester was attempted for the nvGesture, the problem of overfitting would be much more noticeable, since the train set can be easily "memorized" by the model to get high accuracy on the train set, while not learning useful features for the validation set.

For this transfer learning process, both the 2D and 3D CNN parameters were locked (impeding its parameters to change) only adjusting the final linear layers of the network. If the model previously trained learnt useful features for detecting hand gestures, these CNN networks should provide useful parameters for the newly adjusted linear layers to predict classes in the new dataset. This dataset was experimented with the simpler model (RT3D_8F) and the best performing model (RT3D_16F). Table 5.3 shows the result from the network when presenting only the gesture (not the full sample) similarly to the conditions in the Jester dataset. Table 5.4 presents the results from the gesture trying to predict the gesture from the full sample (a 10 seconds sample that contains portions of "No Gesture"), compared to other state-of-the-art techniques. Note that this dataset contains depth images, which can considerably increase the accuracy when used, however since this work only uses RGB frames, the other results were omitted for comparison purposes.

Table 5.3 – Scores obtained in nvGesture's validation set - Isolated gesture only

Technique	top1	top3	top5	Model's size
RT3D-8F	54.98 %	79.88 %	85.06 %	33MB
RT3D-16F-WIDE	69.92 %	86.51 %	91.7 %	117MB

Table 5.4 – Scores obtained in nvGesture's validation set - Full sample

Technique	Input	Accuracy	Model's size
RT3D-8F	8-frames	**43.36 %**	33MB
RT3D-16F-WIDE	16-frames	**67.42 %**	117MB
C3D [18]	16-frames	62.67 %	N/A
ResNeXt-101 [18]	16-frames	66.40 %	N/A
ResNeXt-101 [18]	32-frames	78.63 %	363MB
R3DCNN [27]	32-frames	74.10 %	N/A

5.1.3 EgoGesture Dataset Validation

Similarly to the process established for the nvGesture dataset, this dataset had as a starting point the model pre-trained on the Jester dataset. In a first moment the model was trained without changes in the parameters of the CNN layers, and even though the Jester dataset has significant differences when it comes to what the samples look like ,due to the different points of view from the camera, the parameters learnt from the Jester dataset were effective to achieve an accuracy above 70% for the RT3D_16F model.

After that, the trained process of the model continued by unlocking the parameters of the CNN layers, and then further increasing the accuracy of the network to the values shown in Table 5.5. It is worth noting that despite having more classes, the increased number of samples of this dataset (when compared to nvGesture) allowed for better results in both offline and online results, possibly due to the model not being affected with overfitting as much as the nvGesture dataset did and thus benefiting from unlocking the CNN parameters.

Table 5.5 – Scores obtained in nvGesture's validation set - Gesture only

Technique	top1	top3	top5	Model's size
RT3D_8F	82.72	92.85 %	95.19	33MB
RT3D_16F	86.09	94.43 %	96.36	117MB

The values for the validation test, comparing the techniques with other are shown in Table 5.6. In this test only the segments containing isolated gestures were tested. The column "Input" in the table represents how many frames are used in the network to represent the gesture. Similarly to nvGesture, techniques using depth frames were omitted from this table.

Table 5.6 – Scores obtained in EgoGesture's validation set - Segmented gestures

Technique	Input	Accuracy	Model's size
RT3D-8F	8-frames	**82.72 %**	33MB
RT3D-16F-WIDE	16-frames	**86.09 %**	117MB
VGG-16 [52]	16-frames	62.50 %	N/A
VGG-16 + LSTM [52]	16-frames	74.70 %	N/A
ResNeXt-101 [18]	16-frames	90.94 %	N/A
C3D+LSTM+RSTTM [52]	16-frames	89.30 %	N/A
ResNeXt-101 [18]	32-frames	93.75 %	363MB

5.2 Online Validation

To evaluate the network's online performance, the nvGesture and EgoGesture datasets will be used, since they contain samples which allow this type of test. In the nvGesture dataset, for example, the samples are longer, contain periods in which the person in front of the camera is either performing no gesture at all and simulates driving, or performing an unrelated action with one or more than one person in the frame. A single gesture is performed for each sample, but in these tests, the network has no information about how long the sample is or in which part of the video a gesture is being performed. This allows to test how the C_{th} and the number of frames averaged (described in Section 4.4.2, impact on the online accuracy of the network.

In the EgoGesture dataset, there are multiple gestures per sample, in a different camera angle for the previous dataset. This dataset closely best simulated the real-world scenarios which would be seen in a real-time application, with longer samples (approaching the continuous nature of the task) and multiple gestures being performed. In this dataset, the cooldown system proposed can be evaluated (which was no the case for the nvGesture dataset), observing its capability to prevent multiple computations of the same gesture without interfering in the following gestures performed. The algorithm used to transform the results trained in an offline environment to an online operation was described in Section 4.4.2.

5.2.1 Calculating Accuracy for Online Operation

In contrast to offline operation in which there was a single label for a given sample and the network's objective was to simply predict the most likely label, during online operation there can be multiple labels or no label at all for a given sample. For example, in the sample from the example in Section 4.4.2, there was a single gesture being performed during the entirety of the sample's duration. The network is not hard-coded to yield a single prediction for those samples, instead, it can yield multiple labels or no label at all. To describe the true label for that sample, let's use the notion [1], which represents an array of labels that are present in sequence for that given sample. Since there was only a single "Gesture 1" in the sample ("No Gesture" or "Doing other things" is not considered in this notion) the array [1] represents the true label for that sample. If, for examplé, the sample contained a "Gesture 2", "Gesture 1", "Gesture 4" and "Gesture 3" in sequence, the correct label for that sample would be $[2, 1, 4, 3]$. Likewise, coming back to the original example of the sample which label is [1], the network could yield no label for the sample, thus the predicted a label [] (an empty array). In addition to that, the application could yield more than one label to that sample, $[1, 2]$ for example. Supposing a true label of $[1, 2, 3, 4]$ and predicted label of $[1, 2, 9, 3, 4]$ the question of how the accuracy should be calculated is not trivial, resulting in multiple ways of implementing it. This section will explain the accuracy was calculated for online operation in this dissertation.

In the paper [18], the authors suggest utilizing the Levenshtein Distance (LD) to calculate the accuracy. The LD is a metric for measuring the distance between two sequences. It can be interpreted that it measures the number of "mistakes" between the two. For the following examples, let's consider the sequence $[1, 2, 3, 4]$ as the true label for a given sample. The LD between the true label sequence and the predicted sequence $[1, 2, 9, 3, 4]$ is 1 (the "mistake" here would be the insertion of a 9 if compared to the original). Similarly, the following sequences would also have a LD of 1: $[1, 2, 3]$ (missing one element), $[1, 2, 3, 5]$ (replacing one element with a different one), $[1, 2, 3, 4, 5]$ (assign an additional element). An example of a sequence with a LD of 2 would be: $[1, 2, 4, 3]$ (here, swapping the 3 and 4 counts as missing both), $[1, 2, 9, 3, 4, 5]$ (adding two additional elements), $[1, 2]$ (missing two elements). In order to transform the LD in a metric to measure accuracy (which should be between 0 and 1) the distance is divided by the number of elements in the true label, and then the resulted value is removed from 1 to result in the accuracy. Equation 5.1 shows the equation for accuracy (A), where $LD(T_S, P_S)$ is the Levenshtein Distance between the true sequence (T_S) and predicted sequence (P_S) and Len_{TL} is the number of elements in the true sequence. Since there can be a greater LD than the number of elements in the original sequence, and thus yielding a negative number, the final accuracy value is set to 0 in these cases.

$$A = min(1 - \frac{LD(T_S, P_S)}{Len_{TL}}, 0) \tag{5.1}$$

This section will show the results obtained in the nvGesture and EgoGesture dataset, as well as the process utilized to choose the confidence threshold, the number of frames averaged and the cooldown time, based on the final accuracy and delay obtained for different values experimented.

5.2.2 nvGesture Dataset

As mentioned in the introduction of the dataset, it contains longer samples if compared to the Jester dataset, while still containing only a single label per sample. To find the accuracy for the online operation for this dataset, a combination of values for C_{th}, N and T were experimented, observing the results in the overall accuracy and the delay needed to yield a correct label. The number of frames for cooldown T should not have an impact on the results since all the samples contain only one gesture. As mentioned in the specification chapter, the label's prediction can be yielded before the gesture's performance is over, resulting in a negative delay when compared to the end of the gesture.

During testing for both models RT3D_8F and RT3D_16F, the former did not present satisfactory accuracy for the combination of the parameters mentioned. The model RT3D_8F obtained, at best, and accuracy of 26% for a combination of 13 averaged frames and a threshold of 0.03. The results presented in this section represents the results obtained for the model RT3D_16F.

To determine the best combination of the parameters mentioned before, it was experimented with a range of parameters, trying to find an optimal point for the dataset. Figure 5.1 shows the accuracy obtained for a combination of C_{th} (x-axis) and N (different lines). The T parameter was fixed for the size of a full window that the model can analyze, which is 16 frames for the model. It is possible to investigate that increasing the number of frames averaged, the correspondent optimal C_{th} is lower. Averaging more frames results in better accuracy, however with diminishing results for a large

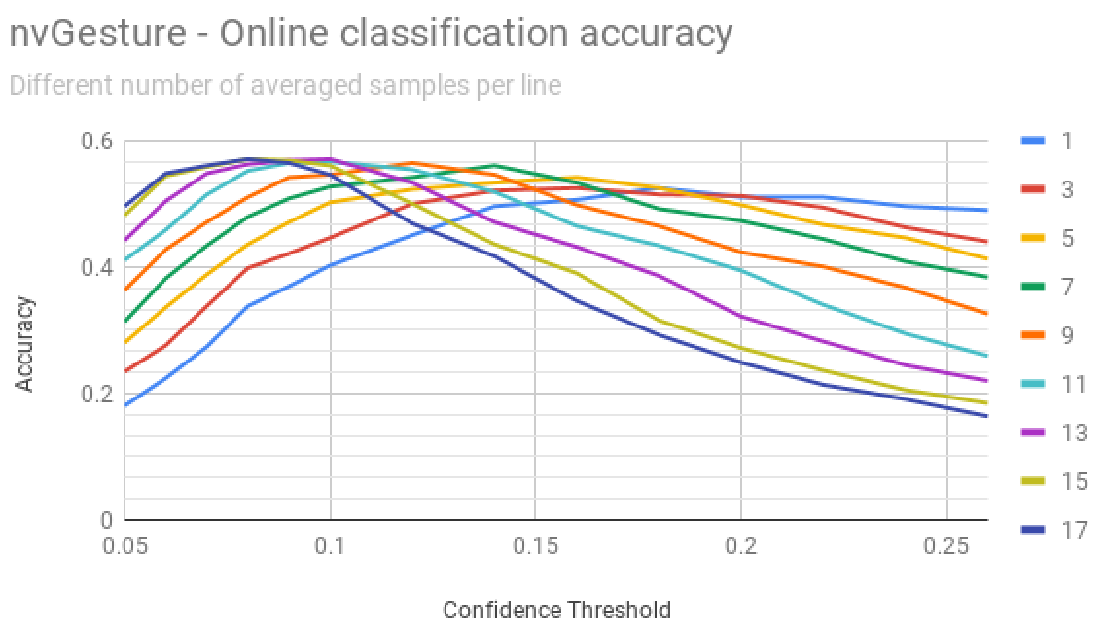


Figure 5.1 – Accuracy of online operation for the nvGesture dataset. Each line represents a different number of averaged frames

enough N (the accuracy peaks for $N = 13$). It is expected as well that further increasing the number of averaged frames (past the 17 shown in the graph) might hinder the network capability of detecting gestures, because a gesture can be fully represented within those frames, and increasing the detection window might blend data from other gestures being performed. For this test, further increasing N did not change the values for accuracy and delay (making the results for $N = 19$ the same as $N = 17$), likely due to the sample not being long enough or not containing multiple gestures to effectively make a noticeable difference for this test.

On the other hand, one of the implications of increasing the number of averaged frames is the increase in the delay to make the prediction. Figure 5.2 show the number of frames on average in which the prediction occurred before the deadline (higher is better). This metric only represents the delay when the gesture is correctly classified, and thus for lower accuracy points of the graph, the sample size is considerably smaller. The points in each line of the graph represent the best accuracy for each N, base on the values from Figure 5.1. There is a trade-off between accuracy and detection time when changing N, which depending on the application might be needed to opt for a lower accuracy to yield faster predictions. Note that even for large N in the graph, the system is still able to detect the gesture before its completion .Since this data intended for a use at 16 frames per second, each

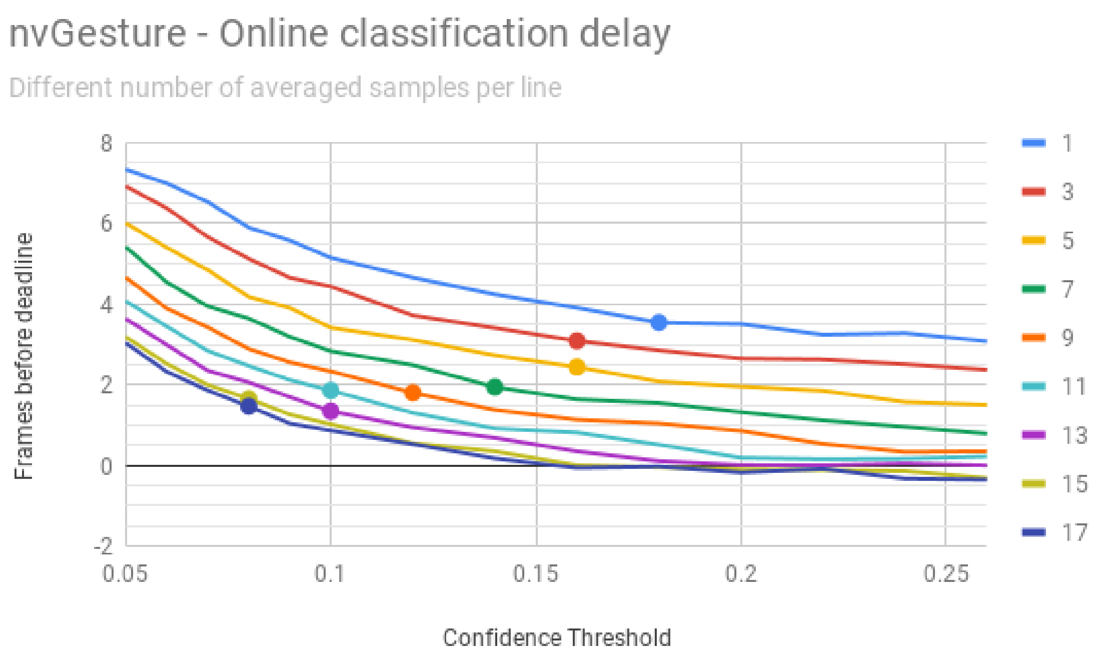


Figure 5.2 – Number of frames in which the classification occurred before the deadline of online operation for the nvGesture dataset. Each line represents a different number of averaged frames

frame represents 0.167s of time. Table 5.7 shows for each N tested, the C_{th} which resulted in the best accuracy and the delay for each of these combinations (same points shown in Figure 5.2).

Table 5.7 – Best combinations of Cth for each N

N	C_{th}	Accuracy	Frames before deadline	Time before deadline
1	0.18	52.4 %	3.549	591 ms
3	0.16	52.4 %	3.094	515 ms
5	0.16	54.1 %	2.444	407 ms
7	0.14	56.0 %	1.948	324 ms
9	0.12	56.4 %	1.808	301 ms
11	0.1	56.6 %	1.864	310 ms
13	0.1	57.1 %	1.352	225 ms
15	0.08	57.1 %	1.661	276 ms
17	0.08	57.1 %	1.472	245 ms

5.2.3 EgoGesture Dataset

This dataset's samples is the closely that resembles a scenario of online classification in a real-time application due to it having the longest samples from the three datasets presented, and containing multiple gestures being performed. The process of training and evaluation for this dataset will be the same used for the nvGesture dataset, just presented. The network was tested for a range of N, C_{th} and T, to check the best performing parameters for this dataset.

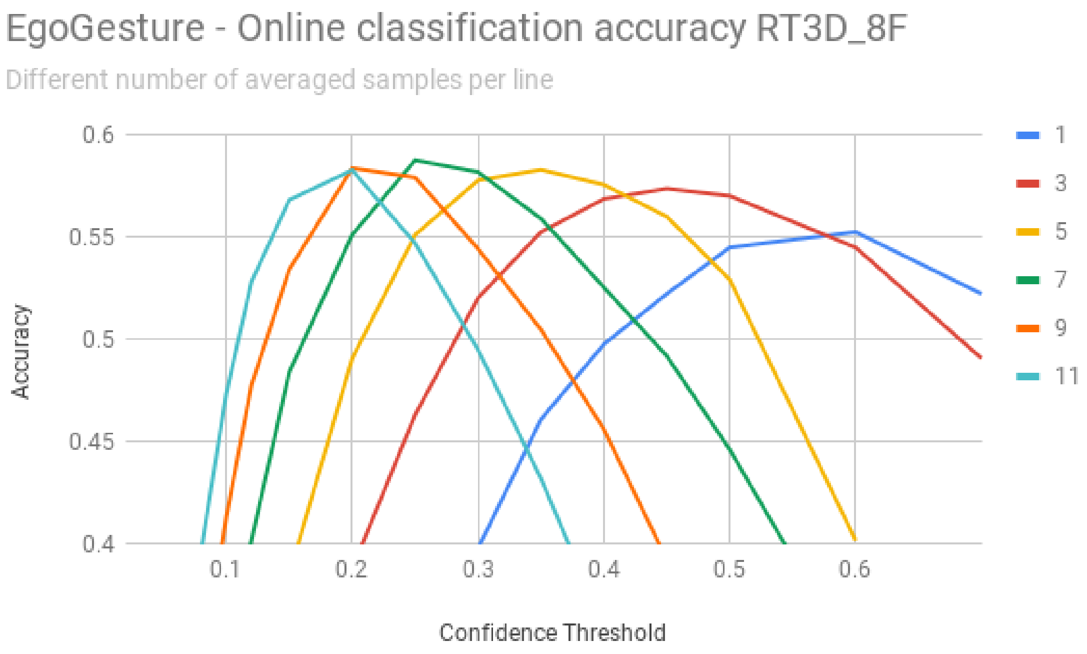


Figure 5.3 – Online accuracy for the RT3D_8F model on the EgoGesture dataset. Each line represents a different number of averaged frames

The accuracy for both network are shown in Figures 5.3 and 5.4. A pattern similar to what was observed in the nvGesture dataset can be observed. There is an optimal point for a given number of averaged frames. The results for the more complex model RT3D_16F did not improve much upon the model RT3D_8F, even though it displayed a slightly higher accuracy during the offline tests.

For the delay calculation of this dataset, it was opted to only consider the first gesture of each sample. This measure was adopted to isolate the delay itself, not depending on a correct prediction of previous gestures. This time, for simplification purposes, the average delay per C_{th} was plotted (instead of plotting multiple lines, one per different N). Figures 5.5 and 5.6 show the frames before deadline, averaged for $N = 1$ to $N = 11$ for both the RT3D_8F and RT3D_16F networks. Note that for the conversion of the delay in seconds, the acquisition rate (FPS) must be taken into consideration, which is different between the networks.

Table 5.8 shows the best accuracy points for each N evaluated. It is possible to see that for most of the points presented, the average prediction occurs after the deadline for a given gesture,

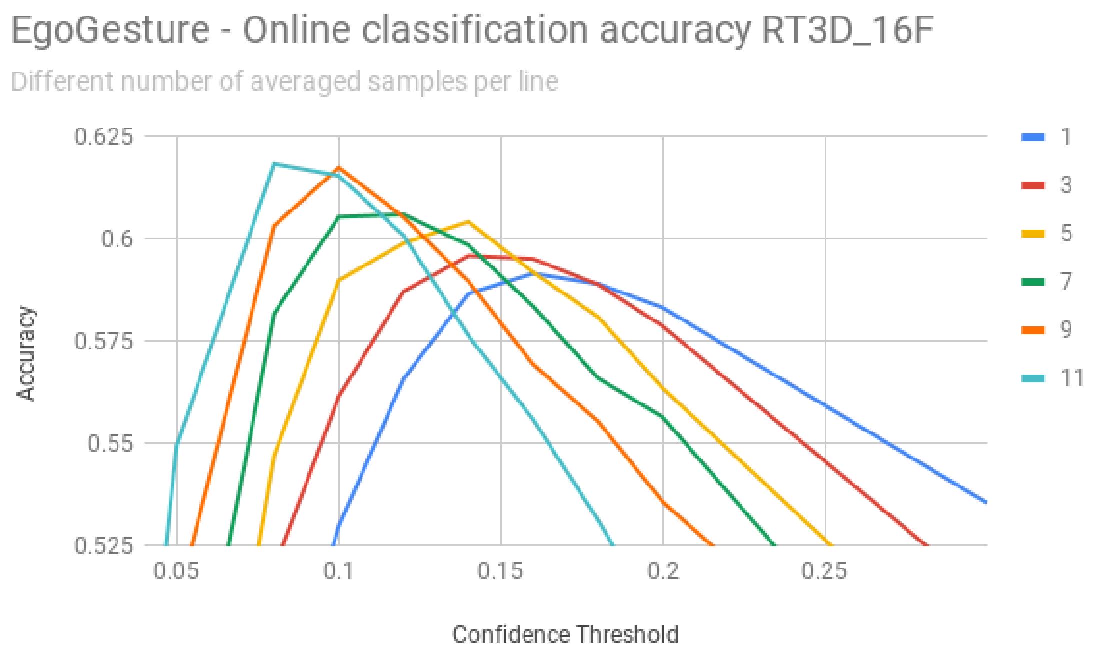


Figure 5.4 – Online accuracy for the RT3D_16F model on the EgoGesture dataset. Each line represents a different number of averaged frames

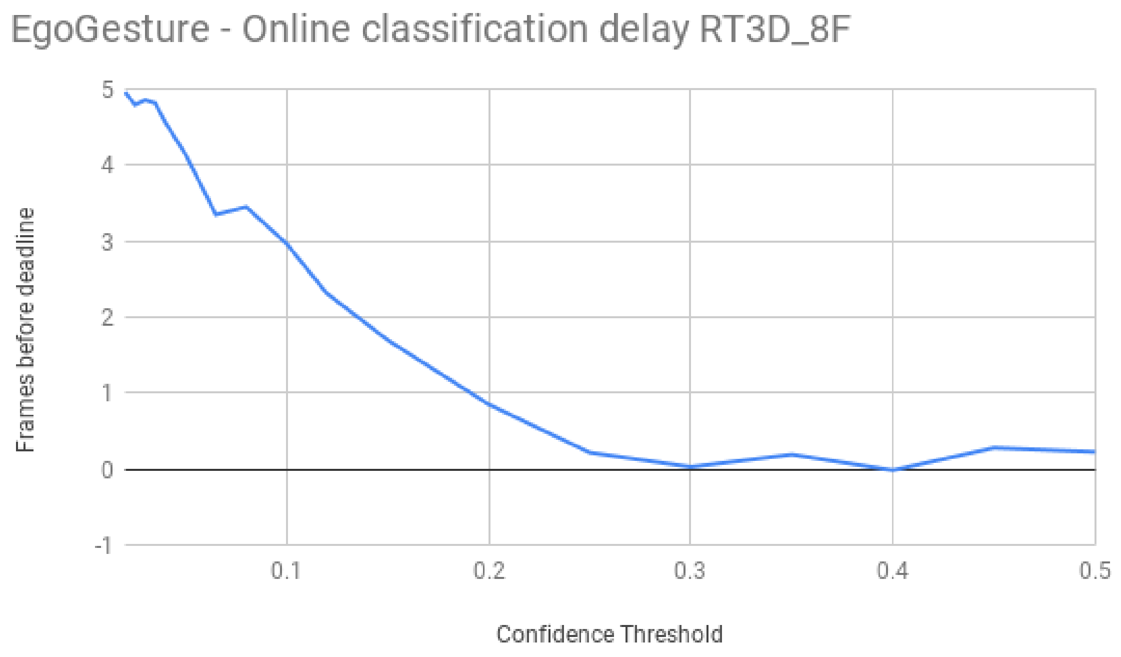


Figure 5.5 – Online delay for the RT3D_8F model on the EgoGesture dataset. Results averaged for a given confidence threshold

which was not the case for the nvGesture dataset. In these cases it is possible to opt for values of N and C_{th} which would yield a negative delay, however with reduced accuracy.

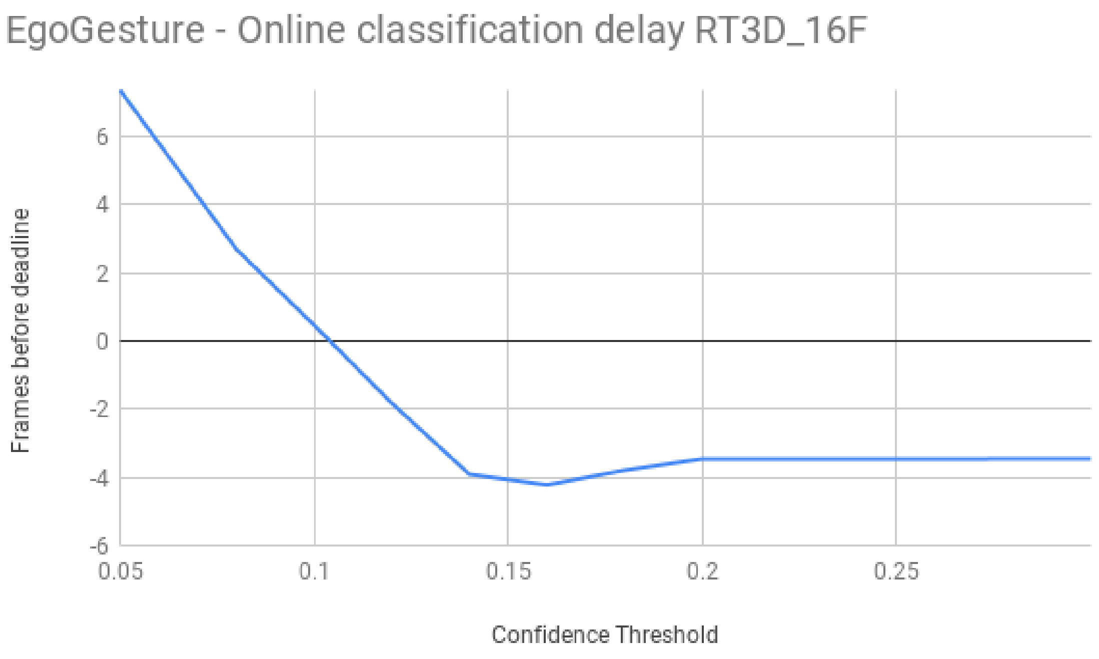


Figure 5.6 – Online delay for the RT3D_16F model on the EgoGesture dataset. Results averaged for a given confidence threshold

5.3 Performance validation

To evaluate the performance of the proposed models, a test was elaborated to measure the time needed to run a classification through the model 1000 times, that is, how much time was needed to run a thousand windows of frames. In this test, a random tensor of shape identical to what the input images for a given window of frames would have is input to the classification model of the network, then the scores for each class are calculated, and the process is repeated in a loop. The time elapsed was saved for each tested model. Also, the VRAM usage (video memory) was measured before and during running the tests, not considering the tensor size used as an input to the model. The model used for the test were all trained for the EgoGesture dataset. The hardware used was a PC, with an i7-6700k processor and a GTX 1080TI GPU (11GB VRAM), running Ubuntu 18.10.

The proposed models RT3D_8F and RT3D_16F were tested in two runs, utilizing the optimizations of data re-usage and not re-using data. This allows us to evaluate the performance gains using the approach of inserting a 2DCNN in the network to do some of the spacial-feature detection.

The technique for online gesture classification presented in [18] was evaluated through the same test to compare the proposed model with existing techniques in the literature. This technique

Table 5.8 – Best combinations of Cth for each *N*

Model	N	C_{th}	Accuracy	Frames before deadline	Time before deadline
RT3D_8F	1	0.5	0.545	-0.196	-32.7 ms
RT3D_8F	3	0.45	0.574	-1.164	-194 ms
RT3D_8F	5	0.35	0.583	0.293	48.8 ms
RT3D_8F	7	0.25	0.587	-0.575	-95.9 ms
RT3D_8F	9	0.2	0.584	-0.392	-65.4 ms
RT3D_8F	11	0.2	0.583	-0.066	-11.1 ms
RT3D_16F	1	0.16	0.592	-1.887	-125.7 ms
RT3D_16F	3	0.14	0.596	-1.316	-87.7 ms
RT3D_16F	5	0.14	0.604	-5.688	-379.2 ms
RT3D_16F	7	0.12	0.606	-4.033	-268.8 ms
RT3D_16F	9	0.1	0.618	-3.226	-215.1 ms
RT3D_16F	11	0.08	0.618	-2.617	-174.4 ms
RT3D_16F	13	0.08	0.626	-3.442	-229.4 ms

was chosen due to it having the same premise as the proposed model (i.e. online gesture classification) and due to the author publicly providing the code and its trained model for the EgoGesture dataset, making it possible to test its model in our proposed test.

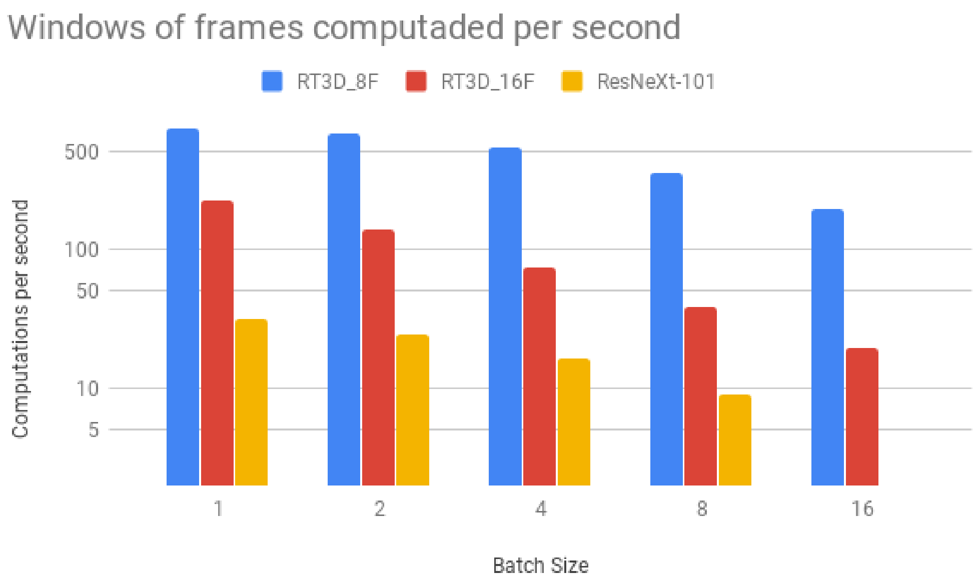


Figure 5.7 – Number of computations per second performed on each model for different batch sizes.

Figure 5.7 and Table 5.9 show the number of computations of windows per second obtained for each model in a different number of batch sizes. As observed, due to layers of hardware, drivers, libraries used and others, the number of operations per second is not linear with the amount of data processed (represented by the batch size). Running the tests with an increased batch size was opted to

Table 5.9 – Number of computations per second performed on each model for different batch sizes

Batch size	RT3D_8F	RT3D_16F	ResNeXt-101
1	739.24	225.08	31.76
2	670.45	138.90	24.47
4	538.11	74.28	16.26
8	349.18	38.52	9.07
16	195.21	19.54	1.98

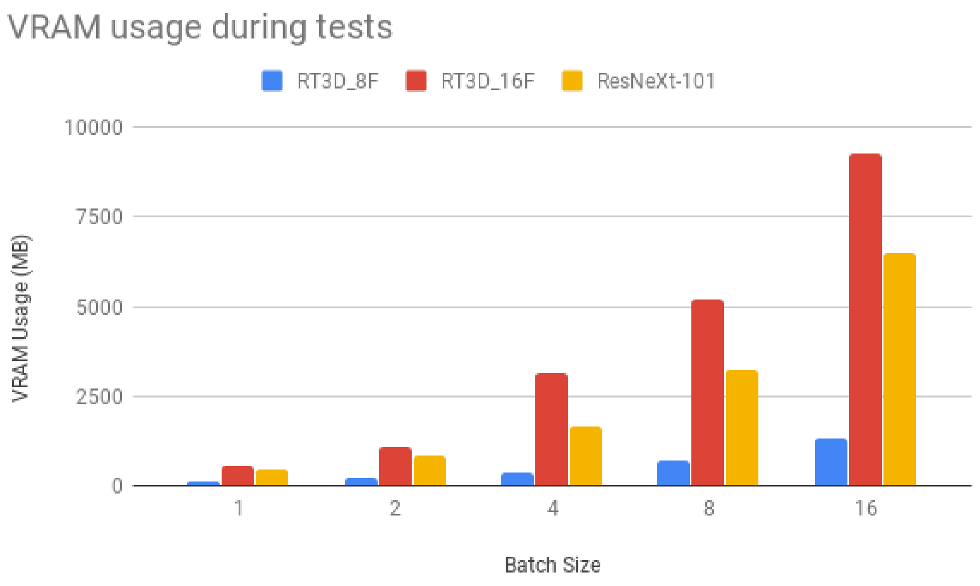


Figure 5.8 – VRAM allocated during the tests for each model and batch size.

see how the techniques would perform in an environment in which the hardware is underused versus one in which the hardware is close to its limitations. A batch size of 32 was not possible to run on the models RT3D_16F and ResNeXt-101 due to VRAM limitations of the system used. The VRAM usage for the tests is shown in Figure 5.8 and Table 5.10.

The results obtained in the tests show distinct levels of performance for each technique. The RT3D_8F model was able to perform, on average, 6.88 times faster than the RT3D_16F model and 44.16 times faster than the ResNeXt-101 model, while using 16.31% and 23.61% of the VRAM usage, respectively. When comparing the RT3D_16F model with the ResNeXt-101 model, the proposed model presented a 6.28 faster time to process a given window of frames, on average, however utilizing 50% more VRAM.

It is important to mention that this test would have to be performed on specific hardware to evaluate which model has a better performance. There are several layers of hardware and software

Table 5.10 – VRAM allocated during the tests for each model and batch size

Batch size	RT3D_8F	RT3D_16F	ResNeXt-101
1	133 MB	564 MB	447 MB
2	210 MB	1098 MB	851 MB
4	362 MB	3152 MB	1627 MB
8	681 MB	5184 MB	3230 MB
16	1319 MB	9274 MB	6494 MB

which might benefit one technique or the other, and thus these results serve as an estimate of the complexity of the techniques, and should not be interpreted that in any given hardware the same differences in performance would be measured the same.

The same test, comparing the optimized and unoptimized versions of models RT3D_8F and RT3D_16F, i.e. not re-utilizing data from the 2DCNN output between cycles, was performed to evaluate how much impact it had on the overall performance of the models. Across the same conditions of the test before, the optimized version of the model RT3D_8F performed 2.14 times faster than the unoptimized version, and the optimized version of the model RT3D_16F performed 2.74 times faster than the unoptimized version. Table 5.11 shows the values obtained in for each test.

Table 5.11 – Number of computations per second performed on each model for different batch sizes, compared to its unoptimized version

Batch size	Optimized?	RT3D_16F	RT3D_8F
1	No	103.823	526.499
	Yes	225.085	739.243
2	No	57.330	346.042
	Yes	138.904	670.457
4	No	22.655	218.519
	Yes	74.279	538.114
8	No	12.267	126.965
	Yes	38.524	349.183
16	No	6.348	64.439
	Yes	19.536	195.211

5.4 Discussion

In this section, we analyzed the results obtained with the proposed models in three different datasets. The model utilizing flow frames (RT_FLOW) did not perform as expected. The use of a lighter implementation of optical flow would still cause an overhead in the computing performance, considering that the computation time of optical flow itself, even in its lighter implementations, is still in the same realm of the models discussed. In preliminary testing, the calculation of the flow frames by themselves, not considering the processing for the 2DCNN and following layers, took around 26ms for a window of 8 RGB frames of size $[96, 96]$. As a comparison, the model with the highest accuracy, ResNeXt-101, took $31ms$ to process its entire score. Besides that, during our testing, it was not observed any gain in accuracy performance when utilizing the flow frames in conjunction with the RGB frames. One possible cause for that is that the utilized network for optical flow (PWC_Net) was optimized for synthetic datasets, which have more consistent lighting, contours, shadows, etc. The added difficulty of producing flow frames with real-word footage might have resulted in a noisier result, not aggregating more information to the network in comparison to the already present RGB frames. Because of the lack of improvement in comparison to the RT3D_16F model, it was opted to discontinue testing with RT_FLOW.

In the Jester dataset, the RT_RGB-16F model obtained an accuracy 3.77% smaller than the top-performing technique (MFF [19]), 92.65% compared to 96.28% accuracy, or almost twice as likely (97.58% increase) to make an incorrect prediction. For the case of RT3D_8F, the accuracy was obtained for the validation set, which if it was compared to the test set accuracy of the best accuracy technique would result in a result 6.41% smaller, and 165.86% more likely to make a wrong prediction.

For the nvGesture dataset, the results were analyzed comparing techniques that utilize RGB frames only. The offline results of the dataset for the RT3D_16F model are comparable to other techniques that have a similar window size (16 frames). When compared to the overall best technique (ResNeXt-101 [18]) it had an accuracy 14.25% smaller than the mentioned technique (52.45% more likely to make an incorrect prediction). The less powerful RT3D_8F model, had an accuracy 55.15% smaller than the previously mentioned technique, or 165.04% more likely to make a wrong prediction.

In the EgoGesture dataset, the offline results considered techniques utilizing only RGB frames. The RT3D_16F model obtained an accuracy of 86.09%, when compared to the best performing technique analyzed (ResNeXt-101 [18]) it had a decrease of 8.17% in accuracy performance, making it 122.56% more likely to make an incorrect prediction. The RT3D_8F model obtained a 82.72% accuracy, a 11.76% reduction if compared to the same technique (176.48% more likely to make an incorrect prediction).

When analyzing the online performance of the models, there are fewer techniques in the literature to compare the results. The main paper analyzed was from [18], which proposed the calculation of accuracy for online classification utilizing the Levenshtein distance. In their work, they present the achieved their Levenshtein accuracy for both the nvGesture dataset and EgoGesture dataset, however, the only value presented is for their best performing techniques, which in both cases utilizes depth frames. The use of depth frames facilitates the task of gesture classification, increasing the author's accuracy from 93.75% to 94.03% on their best performing model for the EgoGesture dataset and from 78.63% to 83.82% in their best performing model for the nvGesture dataset. Taking that into account, the Levenshtein accuracy calculated for the ResNeXt-101 model [18] in the nvGesture dataset was 77.39% and for the EgoGesture dataset was 91.04%. The best Levenshtein accuracy obtained in this dissertation for the nvGesture and EgoGesture dataset are respectively 57.10% (a decrease of 26.21%) and 62.60% (a decrease of 28.44%).

Overall, although the accuracy of the presented models does not directly compete with state-of-the-art techniques, it is possible to consider that such models would be appropriate for use in embedded systems. The reduce in computational cost observed in this section can represent the possibility of being able to run this technique on a given hardware or not. In the cases where there are fewer classes to distinguish from, therefore allowing for better accuracy, the lower computational cost when compared to other techniques might make a good fit for the application in a given hardware. The presented technique in this dissertation offer an option for applications running on less powerful hardware, while still to achieve the best accuracy possible.

The existence of a dataset rich in sample size also showed to have a big impact on the models. The nvGesture dataset, which had the fewest number of samples per class, ended up having an accuracy lower than the other dataset trained by transfer learning, EgoGesture, which had a considerably higher number of classes to distinguish from, and its model was able to achieve higher accuracy.

5.5 Threats to validity

In this section, we will discuss the threats to validity of the presented work, discussing the hypothesis in which this work is based and how they can affect the final conclusions of the work.

One of the focus of this dissertation is to proposed a methodology for a Convolutional Neural Network adequate for the use in embedded systems, i.e. with a smaller computing cost to performance ratio than other available techniques. In our comparisons, there is a lack of techniques that proposed models as simple as the one proposed in this dissertation. The main hypothesis here is that our model offers something new to the literature, providing an network topology which benefits implementations in simpler hardware. The main threat here, is the possibility that the existing models can be easily adapted (e.g. by reducing the number of layers, diminishing the image size, etc.) to perform in a similar level as our proposed model. Since these simpler models were not discussed in the original papers, we decided that the results presented offer something new to the literature, approaching the problem of gesture recognition from a embedded system point of view.

6. CONCLUSION

In this dissertation, neural network techniques suitable for online gesture recognition in embedded systems were studied. This study was limited the used of RGB frames only, in order to avoid the need for additional equipment other than a camera for the capture of the footage. For such, it was first reviewed the basics of neural networks and the different topologies developed with time (such as CNNs) and then the state-of-the-art techniques for video classification, observing which techniques were more present among the works, and which ones would be preliminarily viable in an embedded system.

Out of the studied possibilities, it was chosen to experiment mainly with 3D convolutional networks, which are part of other online classifications models. In addition to that, it was observed that Optical Flow was commonly used in the top-performing action and gesture recognition techniques, and a different model, utilizing flow frames, was also proposed. Due to the possibility of reusing data with the inclusion of a 2DCNN, it was opted to introduce 2DCNN layers before the 3DCNN. These layers are capable of extracting spatial features, while in between pooling layers reduce the size of the tensor, allowing the following 3DCNN to be reduced in size. It was originally proposed three models using this 2DCNN and 3DCNN: RT3D_8F utilizing a window size of 8 frames, RT3D_16F utilizing 16 frames with increased image size and RT_FLOW which used the same frame size as RT3D_8F, 16 frames and includes the use of flow frames in the model.

During our testings, the originally proposed model RT_FLOW did not perform as expected, sharing a similar accuracy to its variant without the use of flow frames. The chosen implementation for optical flow (PWC-Net) was trained to best perform on synthetic datasets (computer-generated videos), and its computing cost is low, compared to other optical flow algorithms. The lack of accuracy gain when utilizing the flow frames produced by the technique and the computing time overhead from the generation of flow frames suggests that the use of optical flow in real-time online operation will depend either on more powerful hardware or in the creation of low computational cost implementations of optical flow for real-world footage.

On the other hand, the two other proposed models RT3D_8F and RT3D_16F presented an offline classification accuracy in line with other tested techniques. While not directly competing for the highest accuracy in any dataset, the achieved accuracy was still above 85% for both the EgoGesture and Jester datasets. For the nvGesture, due to the reduced samples to learn from, the accuracy obtained was significantly smaller than the other two datasets (which contain a higher number of classes to distinguish from).

The performance analysis, that is the time to compute a given model, shows the benefits of the proposed technique. The presented models were able to perform over 6 times faster for the case of the RT3D_16F model or over 44 times faster in the case of the RT3D_8F model to the compared technique ResNeXt-101 [18] (which was also developed for real-time usage). While the accuracy shown is significantly reduced when compared to other techniques in the literature, the reduced need for computational performance to runs the model can make them suitable for lighter applications in embedded systems.

In summary, in this dissertation, models for online gesture classification were shown, presenting faster computing times while maintaining an accuracy comparable to other techniques present in the literature. Some of the design choices for this project proved to be a good fit for the proposed application in embedded systems, such as the data re-usage of the 2DCNN and the cooldown system, which allowed a good performance in online operation while maintaining accuracy within the expected values. The area of online video classification using neural networks, especially for embedded systems, is relatively unexplored if compared to other topics of machine learning. This dissertation approached a new perspective on the usage of online gesture classification techniques by adapting it to better perform in embedded systems and thus adding to the state-of-the-art for the field.

BIBLIOGRAPHY

[1] Alif, M. A. R.; Ahmed, S.; Hasan, M. A. “Isolated bangla handwritten character recognition with convolutional neural network”. In: 2017 20th International Conference of Computer and Information Technology (ICCIT), 2017, pp. 1–6.

[2] Andrew Ng. “Machine learning - coursera”. https://www.coursera.org/learn/machine-learning, Last accessed on 2019-07-30.

[3] Arden Dertat. “Applied deep learning - part 1: Artificial neural networks”. https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6, Last accessed on 2019-07-30.

[4] Brox, T.; Malik, J. “Large displacement optical flow: descriptor matching in variational motion estimation”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33–3, 2011, pp. 500–513.

[5] Cao, Z.; Hidalgo, G.; Simon, T.; Wei, S.; Sheikh, Y. “Openpose: Realtime multi-person 2d pose estimation using part affinity fields”, *CoRR*, vol. abs/1812.08008, 2018, 1812.08008.

[6] Carreira, J.; Zisserman, A. “Quo vadis, action recognition? A new model and the kinetics dataset”, *CoRR*, vol. abs/1705.07750, 2017, 1705.07750.

[7] Christopher Olah. “Understanding LSTM networks”. http://colah.github.io/posts/2015-08-Understanding-LSTMs, Last accessed on 2019-07-30.

[8] Donahue, J.; Hendricks, L. A.; Rohrbach, M.; Venugopalan, S.; Guadarrama, S.; Saenko, K.; Darrell, T. “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39–4, apr 2017, pp. 677–691.

[9] Fischer, P.; Dosovitskiy, A.; Ilg, E.; Häusser, P.; Hazirbas, C.; Golkov, V.; van der Smagt, P.; Cremers, D.; Brox, T. “Flownet: Learning optical flow with convolutional networks”, *CoRR*, vol. abs/1504.06852, 2015, 1504.06852.

[10] Goodfellow, I.; Bengio, Y.; Courville, A. “Deep Learning”. MIT Press, 2016, http://www.deeplearningbook.org.

[11] He, K.; Zhang, X.; Ren, S.; Sun, J. “Deep residual learning for image recognition”, *CoRR*, vol. abs/1512.03385, 2015, 1512.03385.

[12] Hochreiter, S.; Schmidhuber, J. “Long Short-Term Memory”, *Neural Comput.*, vol. 9–8, nov 1997, pp. 1735–1780.

[13] Huang, G.; Liu, Z.; Weinberger, K. Q. “Densely connected convolutional networks”, *CoRR*, vol. abs/1608.06993, 2016, 1608.06993.

[14] Ioffe, S.; Szegedy, C. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, *CoRR*, vol. abs/1502.03167, 2015, 1502.03167.

[15] Ji, S.; Xu, W.; Yang, M.; Yu, K. “3D Convolutional Neural Networks for Human Action Recognition”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35–1, Jan 2013, pp. 221–231.

[16] Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. “Large-Scale Video Classification with Convolutional Neural Networks”. In: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[17] Khan, A.; Sohail, A.; Zahoora, U.; Qureshi, A. S. “A survey of the recent architectures of deep convolutional neural networks”, *CoRR*, vol. abs/1901.06032, 2019, 1901.06032.

[18] Köpüklü, O.; Gunduz, A.; Kose, N.; Rigoll, G. “Real-time hand gesture detection and classification using convolutional neural networks”, *CoRR*, vol. abs/1901.10323, 2019, 1901.10323.

[19] Köpüklü, O.; Köse, N.; Rigoll, G. “Motion fused frames: Data level fusion strategy for hand gesture recognition”, *CoRR*, vol. abs/1804.07187, 2018, 1804.07187.

[20] Lee, M.; Lee, S.; Son, S.; Park, G.; Kwak, N. "Motion Feature Network: Fixed Motion Filter for Action Recognition", 2018, pp. 1–17, 1807.10037.

[21] Li, Z.; Gavrilyuk, K.; Gavves, E.; Jain, M.; Snoek, C. G. "VideoLSTM convolves, attends and flows for action recognition", *Computer Vision and Image Understanding*, vol. 166, 2018, pp. 41–50, 1607.01794.

[22] Li, Z.; Gavves, E.; Jain, M.; Snoek, C. G. M. "VideoLSTM Convolves, Attends and Flows for Action Recognition", *CoRR*, vol. abs/1607.0, 2016, 1607.01794.

[23] Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; Alsaadi, F. E. "A survey of deep neural network architectures and their applications", *Neurocomputing*, vol. 234, 2017, pp. 11–26.

[24] Maturana, D.; Scherer, S. "VoxNet: A 3D convolutional neural network for real-time object recognition", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 922–928.

[25] Miller, R. B. "Response time in man-computer conversational transactions". In: Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I, 1968, pp. 267–277.

[26] Molchanov, P.; Gupta, S.; Kim, K.; Kautz, J. "Hand gesture recognition with 3d convolutional neural networks". In: 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2015, pp. 1–7.

[27] Molchanov, P.; Yang, X.; Gupta, S.; Kim, K.; Tyree, S.; Kautz, J. "Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks". In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4207–4215.

[28] Nguyen, T. P.; Pham, C. C.; Ha, S. V. U.; Jeon, J. W. "Change Detection by Training a Triplet Network for Motion Feature Extraction", *IEEE Transactions on Circuits and Systems for Video Technology*, 2018, pp. 1–14.

[29] Niu, Y.; Zou, D.; Niu, Y.; He, Z.; Tan, H. "A breakthrough in Speech emotion recognition using Deep Retinal Convolution Neural Networks", 2017, pp. 1–7, 1707.09917.

[30] Osokin, D. “Real-time 2d multi-person pose estimation on CPU: lightweight openpose”, *CoRR*, vol. abs/1811.12004, 2018, 1811.12004.

[31] Potash, P.; Romanov, A.; Rumshisky, A. “Ghostwriter: Using an lstm for automatic rap lyric generation”. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1919–1924.

[32] Raghav Prabhu. “Understanding of convolutional neural network (cnn) — deep learning”. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148, Last accessed on 2019-07-30.

[33] Rautaray, S. S.; Agrawal, A. “Vision based hand gesture recognition for human computer interaction: a survey”, *Artificial Intelligence Review*, vol. 43–1, jan 2015, pp. 1–54.

[34] Sainath, T. N.; Vinyals, O.; Senior, A.; Sak, H. “Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks”, *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2015-Augus, 2015, pp. 4580–4584, arXiv:1011.1669v3.

[35] Shruti Jadon. “Introduction to different activation functions for deep learning”. https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092, Last accessed on 2019-09-30.

[36] Simonyan, K.; Zisserman, A. “Two-Stream Convolutional Networks for Action Recognition in Videos”, *CoRR*, vol. abs/1406.2, 2014, 1406.2199.

[37] Simonyan, K.; Zisserman, A. “Very Deep Convolutional Networks for Large-Scale Image Recognition”, 2014, pp. 1–14, 1409.1556.

[38] Sun, D.; Yang, X.; Liu, M.-y.; Kautz, J. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”, vol. D, arXiv:1709.02371v3.

[39] Sun, J.; Wang, J.; Yeh, T.-C. “Video Understanding: From Video Classification to Captioning”, 2017, pp. 1–9.

[40] Sze, V.; Chen, Y.; Yang, T.; Emer, J. S. "Efficient processing of deep neural networks: A tutorial and survey", *Proceedings of the IEEE*, vol. 105–12, Dec 2017, pp. 2295–2329.

[41] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A.; Hill, C.; Arbor, A. "Going Deeper with Convolutions", 2014, pp. 1–9, 1409.4842.

[42] Tran, D.; Bourdev, L.; Fergus, R.; Torresani, L.; Paluri, M. "Learning Spatiotemporal Features with 3D Convolutional Networks", 2014, 1412.0767.

[43] Twenty Billion Neurons GmbH. "The 20bn-jester dataset v1". https://www.20bn.com/datasets/jester/v1, Last accessed on 2019-10-10.

[44] Tzu-Wei Huang (lanpa). "tensorboardx". https://github.com/lanpa/tensorboardX, Last accessed on 2019-10-30.

[45] Voulodimos, A.; Doulamis, N.; Doulamis, A.; Protopapadakis, E. "Deep learning for computer vision: A brief review", *Computational Intelligence and Neuroscience*, vol. 2018, 02 2018, pp. 1–13.

[46] Wang, R. Y.; Popović, J. "Real-time hand-tracking with a color glove", *ACM Trans. Graph.*, vol. 28–3, Jul 2009, pp. 63:1–63:8.

[47] Wu, Z.; Wang, X.; Jiang, Y.-G.; Ye, H.; Xue, X. "Modeling Spatial-Temporal Clues in a Hybrid Deep Learning Framework for Video Classification", 2015, 1504.01561.

[48] Xu, J.; Ranftl, R.; Koltun, V. "Accurate optical flow via direct cost volume processing", *CoRR*, vol. abs/1704.07325, 2017, 1704.07325.

[49] Young, T.; Hazarika, D.; Poria, S.; Cambria, E. "Recent trends in deep learning based natural language processing [review article]", *IEEE Computational Intelligence Magazine*, vol. 13–3, Aug 2018, pp. 55–75.

[50] Zeiler, M. D.; Fergus, R. "Visualizing and Understanding Convolutional Networks", *CoRR*, vol. abs/1311.2, 2013, 1311.2901.

[51] Zhang, B.; Wang, L.; Wang, Z.; . . . , Y. Q. P. o. t. I.; undefined 2016. “Real-time action recognition with enhanced motion vector CNNs”, *Cv-Foundation.Org*, pp. 2718–2726.

[52] Zhang, Y.; Cao, C.; Cheng, J.; Lu, H. “Egogesture: A new dataset and benchmark for egocentric hand gesture recognition”, *IEEE Transactions on Multimedia*, vol. 20–5, May 2018, pp. 1038–1050.

[53] Zolfaghari, M.; Singh, K.; Brox, T. “ECO: efficient convolutional network for online video understanding”, *CoRR*, vol. abs/1804.09066, 2018, 1804.09066.

APPENDIX A – JESTER DATASET SAMPLES

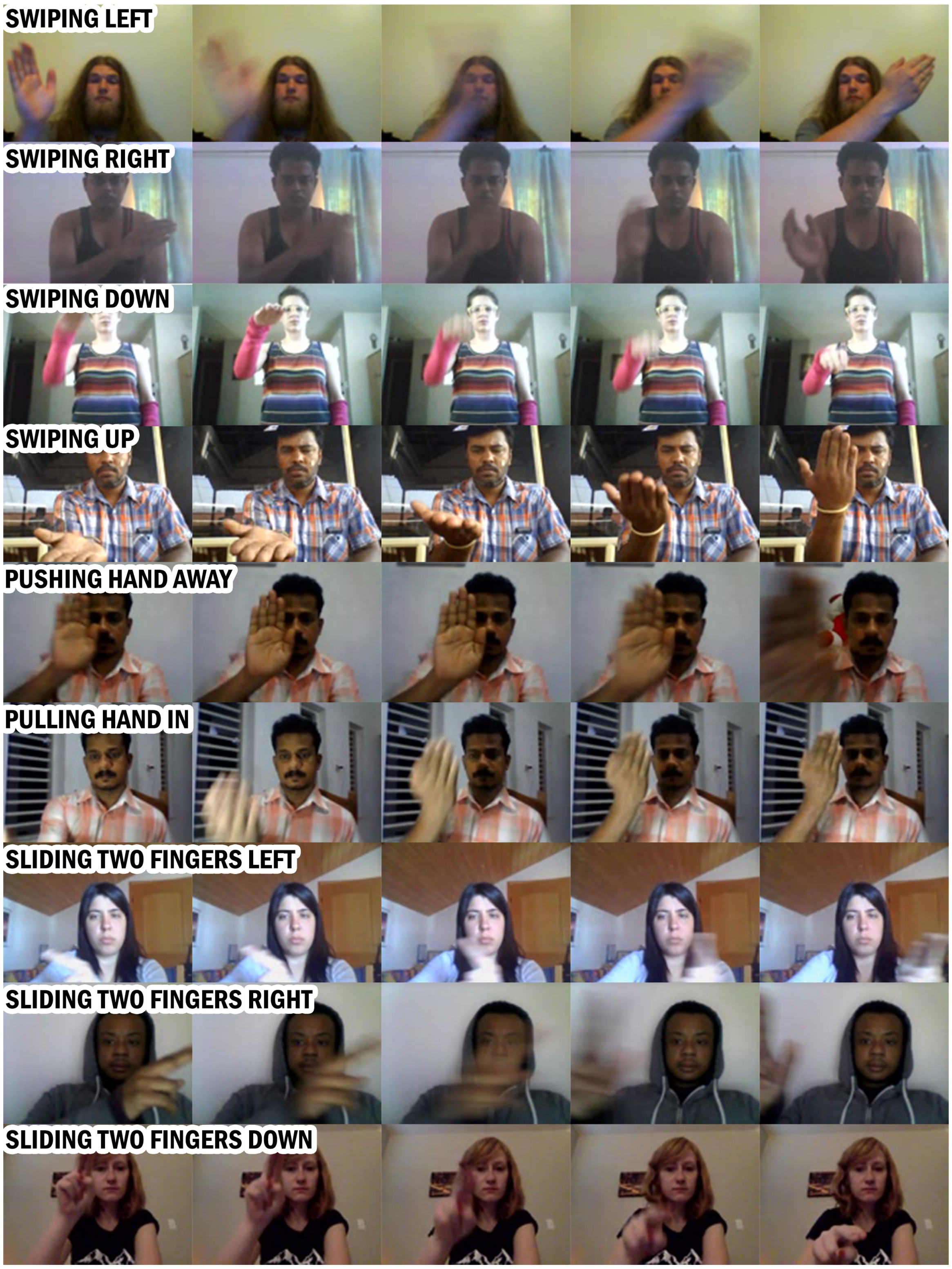


Figure APPENDIX A.1 – Samples from classes 0 to 8 from Jester's dataset

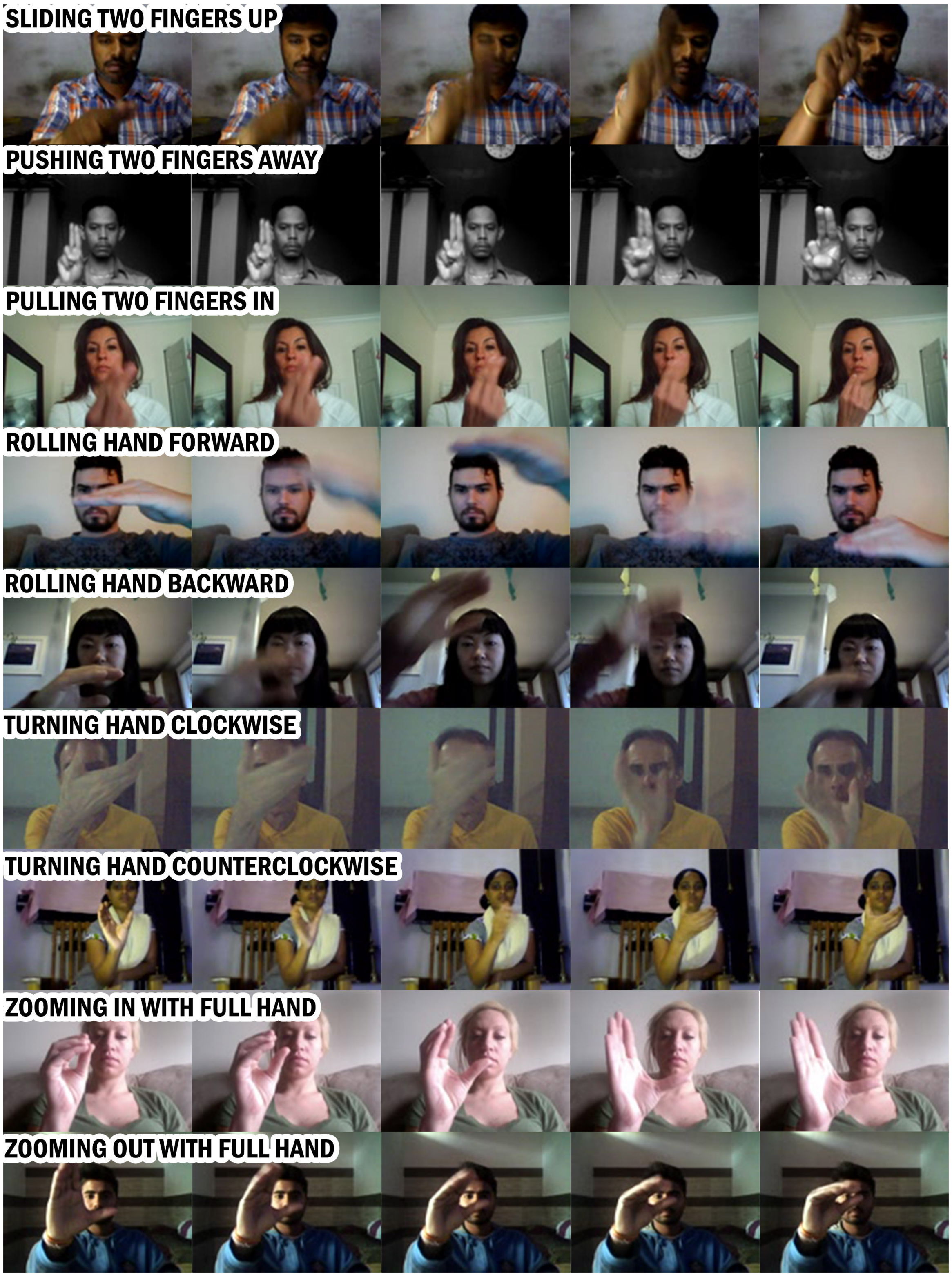


Figure APPENDIX A.2 – Samples from classes 9 to 17 from Jester's dataset

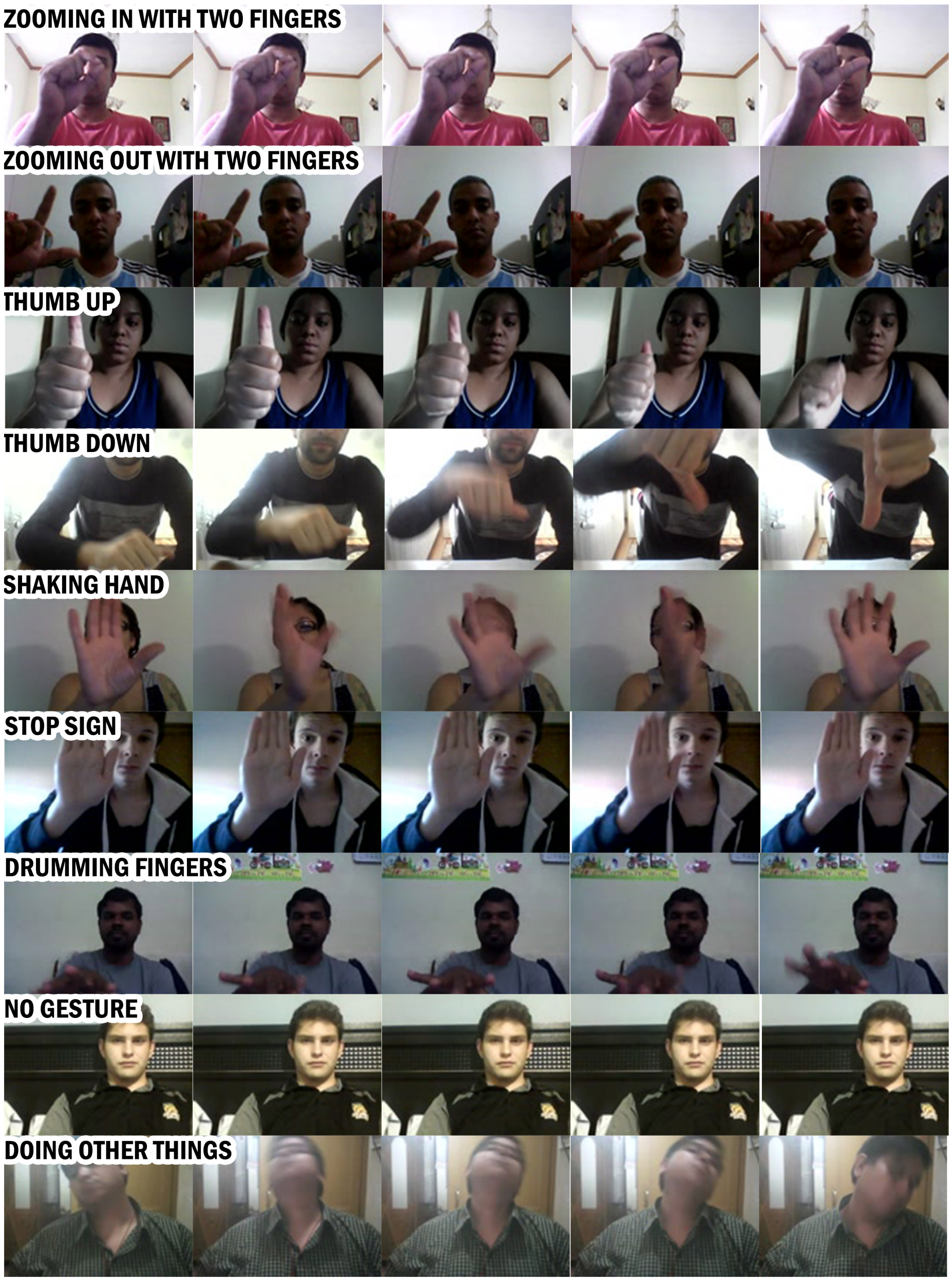


Figure APPENDIX A.3 – Samples from classes 18 to 26 from Jester's dataset

ATTACHMENT A – EgoGesture classes of hand gestures

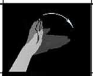
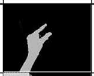
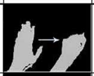
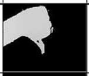
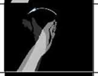
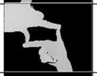
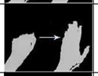
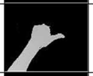
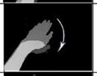
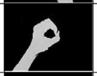
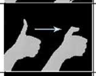
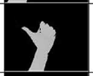
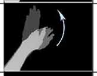
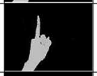
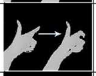
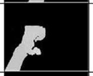
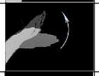
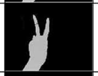
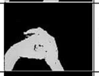
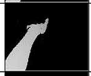
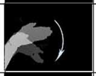
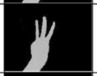
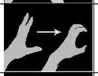
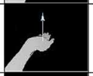
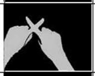
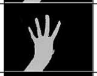
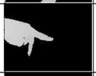
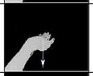
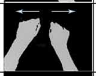
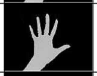
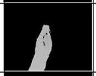
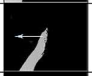
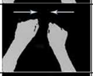
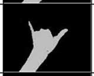
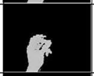
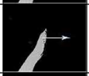
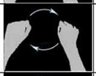
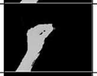
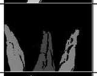
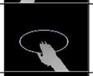
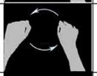
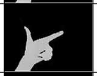
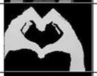
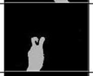
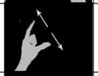
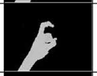
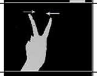
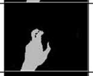
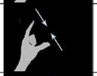
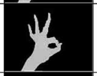
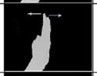
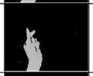
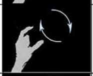
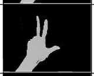
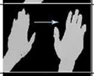
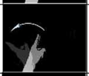
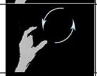
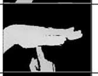
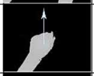
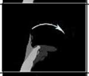
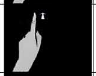
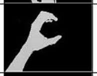
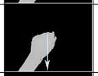
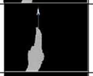
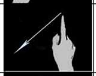
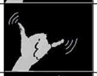
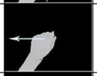
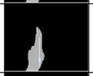
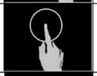
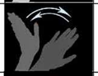
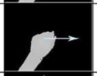
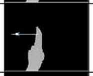
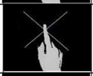
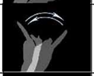
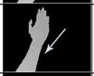
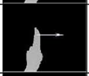
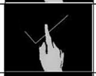
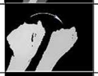
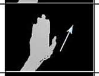
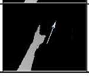
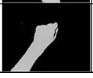
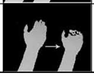
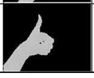
Label	Illustration	Instruction	Label	Illustration	Instruction	Label	Illustration	Instruction	Label	Illustration	Instruction
1		Scroll hand towards right	22		Measure (distance)	43		Palm to fist	64		Thumb downward
2		Scroll hand towards left	23		Photo frame	44		Fist to Palm	65		Thumb towards right
3		Scroll hand downward	24		Number 0	45		Trigger with thumb	66		Thumb towards left
4		Scroll hand upward	25		Number 1	46		Trigger with index finger	67		Thumbs backward
5		Scroll hand forward	26		Number 2	47		Hold fist in the other hand	68		Thumbs forward
6		Scroll hand backward	27		Number 3	48		Grasp	69		Move hand upward
7		Cross index fingers	28		Number 4	49		Walk	70		Move hand downward
8		Zoom in with fists	29		Number 5	50		Gather fingers	71		Move hand towards left
9		Zoom out with fists	30		Number 6	51		Snap fingers	72		Move hand towards right
10		Rotate fists clockwise	31		Number 7	52		Applaud	73		Draw circle with hand in horizontal surface
11		Rotate fists counterclockwise	32		Number 8	53		Dual hands heart	74		Bent number 2
12		Zoom in with fingers	33		Number 9	54		Put two fingers togethe	75		Bent another number 3
13		Zoom out with fingers	34		OK	55		Take two fingers apart	76		Dual fingers heart
14		Rotate fingers clockwise	35		Another number 3	56		Turn over	77		Scroll fingers toward left
15		Rotate fingers counterclockwise	36		Pause	57		Move fist upward	78		Scroll fingers toward right
16		Click with index finger	37		Shape C	58		Move fist downward	79		Move fingers upward
17		Sweep diagonal	38		Make a phone call	59		Move fist towards left	80		Move fingers downward
18		Sweep circle	39		Wave hand	60		Move fist towards righ	81		Move fingers toward left
19		Sweep cross	40		Wave finger	61		Bring hand close	82		Move fingers toward right
20		Sweep checkmark	41		Knock	62		Push away	83		Move fingers forward
21		Static fist	42		Beckon	63		Thumb upward			

Figure ATTACHMENT A.1 – Illustrations from the 83 classes of gestures contained in the EgoGesture dataset

Source: [52]