POLITECNICO DI TORINO

Master of Science in

Mathematical Engineering

Deep Learning Algorithms for Video Classification: Application on Real-Time Hand Gesture Recognition

Supervisor:

Prof. Elio Piccolo

Co-supervisor:

Kent Danielsson

Student:

Francesco Dalla Serra

Year 2017-2018

Abstract

In the last few years, deep learning has obtained successful results in the challenging tasks of video understanding. In the first part of this thesis we review some of the most heavily researched tasks in video classification with the corresponding datasets.

Afterwards, we present a survey of the state-of-the-art deep learning methods, explaining the different intuitions to capture the temporal evolution of videos. More precisely, deep neural networks are extended to understand not only what a video contains, as it was for image content analysis, but also what is happening, capturing dynamic information from different frames.

Next, we describe more in detail the Inflated 3D network (I3D), characterized by 3D convolution, and the new configurations adopted in this work, which include different solutions: non-local blocks, spatiotemporal feature gates and temporal dilation, designed to better capture spatio-temporal features. We demonstrate how the model benefits from these techniques, which we believe can become fundamental elements of future video-based neural networks.

We train these models on the Jester hand gesture dataset and we compare the different results. Spatiotemporal feature gates and temporal dilation are then combined, in order to build the most accurate model of this research, called Modified-I3D. This model reaches comparable results to other state-of-the-art solutions, while keeping a good balance between accuracy and computational efficiency.

Finally, we introduce such model in a real-time hand gesture recognition system, as a first step for a novel human-computer interaction through camera. We visualize, with the Grad-CAM algorithm extended to videos, on which input regions the Modified-I3D activates. We observe that these correspond to the hand movements, showing that the model predictions are consistent.

Contents

Chapter 1

Introduction

The work of this thesis consists in implementing and optimizing an algorithm capable of correctly classifying the content of videos with high confidence. In particular, we build a hand gesture classifier for a novel video-based human-computer interaction.

To develop a robust video classifier is essential to capture both spatial and temporal events. The classifer needs to extract the spatial contents from each frame and to analyze the motion information from the sequence of frames. To tackle this problems, we choose to exploit deep learning solutions, since convolutional neural networks have shown great results for both image and video content analysis. More precisely, we modify a 3D convolutional neural network, pre-trained on a human action dataset, proposing various solutions designed to improve spatio-temporal features retrieval. Afterward, we fine-tune the distinct configurations on the selected hand gesture dataset and we apply the best solution for real-time video stream analysis.

1.1 Introduction to Deep Learning

1.1.1 From Visual Perception to Deep Neural Networks

In animal evolution, the origin of eyes is believed to have happened around 540 million years ago and the major developments of the eye supposingly evolved in only a few million years. This event has been a turning point in animal history, since it provided new ways of interacting with the outer world. The fight for survival made it necessary for all animals to develop eyes, each species started evolving in parallel multiple eye types and subtypes, adapting to the surrounding environment. A convincing theory, proposed by

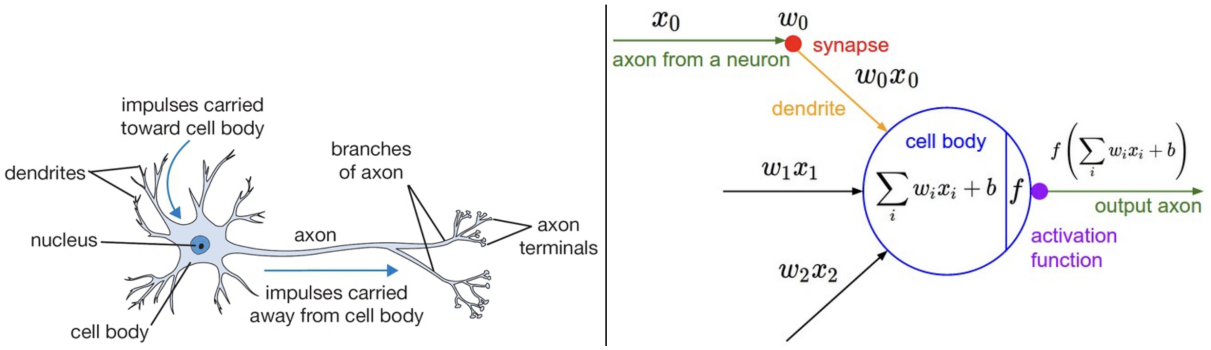


Figure 1.1: Biological neuron (left) and its mathematical model counterpart (right).

zoologist Andrew Parker, supports that the evolution of eyes played a significant role in the explosion of the number of animal species [26].

At first, predecessors of eyes were capable of spotting ambient brightness, in particular to discriminate light from dark. However, this was not enough to be considered vision, since no shapes or light direction were detected. Furthermore, with the evolution of optical systems, animals were capable of detecting shapes and colors of objects, allowing them to better interact with the surrounding habitat.

More precisely, the animal eyes collect images, which are then sent as electrical signals to the visual cortex and other areas of the brain through optic nerves. From the studies of Hubel and Wiesel in the early 50s, which describe the visual mechanism of mammals, it was noted that the primary visual cortex responds to simple information, such as oriented edges, and, as the information moves along the visual processing pathway, the brain capture more and more the complexity of visual informations, until the complex visual world is recognized. These studies inspired the recent development of computer vision with artificial neural networks, which loosely model the human brain capability of processing visual and acustic signals.

The human nervous system is composed by approximately 86 billion neurons, connected by around 10^{14} - 10^{15} synapses. Similarly, the basic computational units of artificial neural networks, called as well neuron, is a simplified mathematical model of its biological counterpart, as shown in figure 1.1. It consists of a weighted sum

$$y = Wx + b, \tag{1.1}$$

where W and b are the weight matrix and the biases vector respectively, while x is either

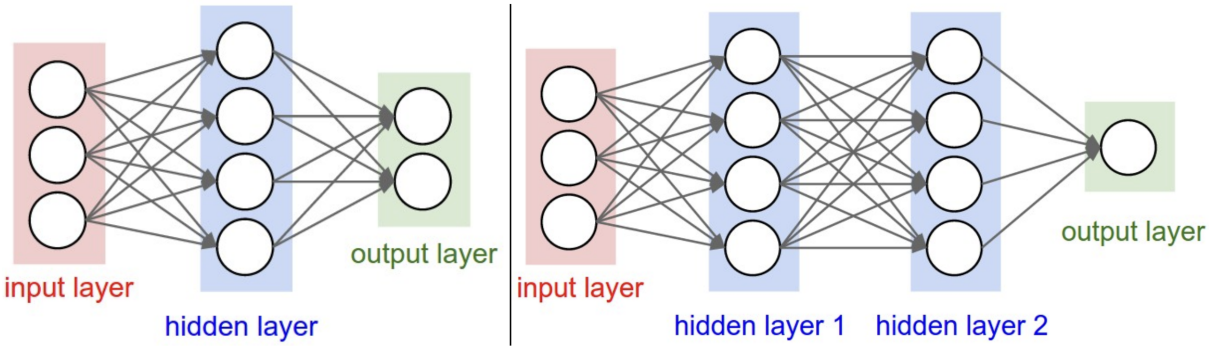


Figure 1.2: Different configurations of Neural Networks. 2-layer Neural Network: three inputs, one hidden layer of 4 neurons and one output layer with 2 neurons (left). 3-layer Neural Network with three inputs, two hidden layers of 4 neurons each and one output layer (right). Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

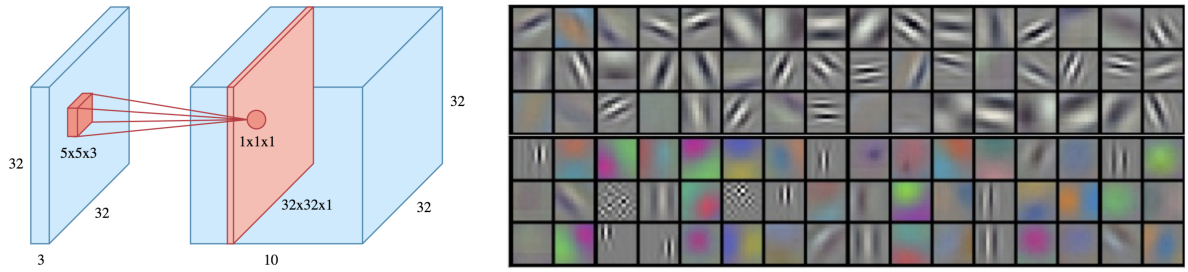
the input of the network or the outputs of neurons from earlier layers. On top of eq. 1.1 a non-linearity f is applied, called *activation function*, necessary to the entire network to capture non-trivial information.

The artificial neurons are then collected in multiple layers, normally organized in a cascade fashion, where each layer takes as input the output of the previous one (figure 1.2). This enables the network to capture multiple level of representation. **Deep learning** takes its name by the growing number of layers, that recent neural networks have adopted, expanding the potential of such algorithms to learn additional and more complex features. These solutions have been applied to various fields such as computer vision, speech recognition, natural language processing and machine translation.

Several types of layers have been introduced, depending on the type of application we are interested in. In computer vision tasks, the core building block of the network is called **convolutional layer**, which gives the name to the family of convolutional neural networks (CNNs). The standard 2D convolutional layers are composed by a set of learnable 3D filters, spatially small but with the same depth[1] extention of the input volume, as shown in figure 1.3a. This operation performes a convolution of such filters across the height and width of the input activation map, produced by the previous hidden layer, or the input image, and it computes dot products between the filters' weights and the input regions.

Each convolutional layer produces an output volume, which corresponds to the con-

[1]With depth we mean the 3rd dimension of the input volume, i.e. for an RGB input image the depth corresponds to the 3 color channels



(a) Convolutional layer example

(b) Filters visualization

Figure 1.3: Example of a convolutional layer on an input image $32 \times 32 \times 3$, with filter size $5 \times 5 \times 3$ and a total of 10 filters. The resulting activation map of the layer is shown as the $32 \times 32 \times 10$ volume (left). Visualization of the learned filters of size $11 \times 11 \times 3$ from the first convolutional layer in the AlexNet architecture [20] (right).

catenation of the 2D activation maps computed by each filter. As shown in figure 1.3a, the depth of the output volume is equal to the number of filters contained inside the layer. Filters can learn different types of patterns and they activate when similar features are spotted in the corresponding input feature map. For example the first layer usually detects various edge orientations or color blobs, as shown in figure 1.3b.

1.1.2 Backpropagation and Parameters Update

So far we have only discussed the forward step of a neural network, where an input signal propagates through the units of the hidden layers to produce the output. However, in order to find the best set of parameters that maps the inputs to the correct outputs, we need to train the network with a backward step. This corresponds to an optimization problem computed in two steps, where the weights are updated in order to minimize a given error function:

- **backpropagation**: calculating the partial derivatives of the error with respect to the weights;
- **parameters update**: adopting an optimization algorithm, such as stochastic gradient descent (SGD), to update the weights using the derivatives.

The two steps are applied iteratively, first by computing the gradient of the error (loss) function $E(l, F(x; W))$, where l and F correspond respectively to the correct set of labels (*ground truth*) and the model function; then by updating the weights W. The

SGD algorithm, at a timestep t, computes the update as

$$W_{t+1} = W_t - \alpha \nabla_{W_t} E(l_i, F(x_i; W_t)) \tag{1.2}$$

where $\alpha \in [0, 1]$ is a constant coefficient, called *learning rate*, and (l_i, x_i) represents a sample of N tuples of labels and inputs from the whole dataset.

Backpropagation is computed using the chain rule twice. Given the output o_j of a neuron j

$$o_j = \phi(\text{net}_j) = \phi\left(\sum_{k=1}^{n} w_{kj} o_k\right), \tag{1.3}$$

where ϕ is a non-linear activation function and net_j is the input of j, we compute the partial derivate of the error function E with respect to a weight w_{ij} as

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}}. \tag{1.4}$$

The last derivatives of the right-hand term is computed as

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i; \tag{1.5}$$

if the neuron belongs to the first hidden layer, o_i is equal to the input x_i.

The partial derivative of o_j with respect to its input net_j corresponds to

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial}{\partial \text{net}_j} \phi(\text{net}_j); \tag{1.6}$$

if ϕ is not differentiable, such as the ReLU function, this is dealt with some approximations.

Finally the partial derivative of the error with respect to o_j is

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y}, \tag{1.7}$$

when j is in the output layer and y is the output of the network. Otherwise, considering E as a function of the neurons set $L = u, v, ..., z$, composed by the neurons that recives input from the neuron j, we obtain the following recursive formula

$$\frac{\partial E}{\partial o_j} = \frac{\partial E(\text{net}_u, \text{net}_v, ..., \text{net}_z)}{\partial o_j} = \sum_{l \in L} \left(\frac{\partial E}{\partial \text{net}_l} \frac{\partial net_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial \text{net}_l} w_{jl} \right). \quad (1.8)$$

1.2 Computer Vision and Deep Learning Libraries

In this section we introduce and describe the main libraries employed to implement this thesis project. First of all, we base our implementations on Python, as programming language. This choice is determined by its readability and because most of the open source machine learning libraries use Python interfaces. This accelerates the research, due to the high amount of open source material available, on top of which we base our experiments.

The core library, chosen to implement computer vision tasks, is *OpenCV* (Open Source Computer Vision), originally developed by Intel Corporation, then by Willow Garage and later by Itseez. Its design aims at computational efficiency and real-time applications. Its optimized code is used by the large community of computer vision researchers to develop a wide range of tasks. In our case, we benefit from this tool in various way, such as:

- video preprocessing: extracting subsets of frames from video clips and resizing them;
- real-time application: extraction of frames from cameras and visualization in real-time;
- video postprocessing: capturing relevant video frames and applying heatmaps.

Many open source machine learning frameworks are available for research in the field of deep learning. Among them we choose the TensorFlow library, developed by the Google Brain team. This choice was made considering its high number of contributors and the easy deployment across multiple CPUs and GPUs. TensorFlow is a library for large-scale computations, basing its structure on *tensors* and *graphs*. Tensors are the core units, represented as multi-dimensional arrays, whose values depend on the input data of the graph. Instead, the computational graphs contain all the operations and tensors necessary for a specific task. Graphs are composed by nodes, which represent the operations that consume and produce tensors, and by edges, that depict the tensors values flowing through the graph.

Furthermore, we build our neural networks with the use of Sonnet, a library build on top of TensorFlow, developed by the DeepMind team.

1.3 Purpose of the Thesis

Computer vision received a great boost with the introduction of deep learning algorithms, reaching state-of-the-art on many tasks and expanding the number of applications where computer vision can be applied. The variety of branches covered so far consist of image understanding (i.e. image classification and object detection,), video understanding (i.e. video classification and action detection), style transfer (i.e. artistic style transfer), image generation and image resolution improvement. Among these, the major contributions and research are produced in the field of image understanding, while many others are still in the early development.

The aim of the thesis consists in the study of the recent tasks of video classification and the corresponding state-of-the-art algorithms. More specifically, we focus on human actions classification, implementing an upgraded deep learning network built for video classification. Finally, we fine-tune such network on a hand gesture dataset, for further real-time applications.

We believe that video classification research is still in its early stages. In fact, most of the solutions proposed are just a straightforward evolution from image understanding CNNs. This is reasonable, due to the great success of extracting spatial features with convolutional layers. However, we assume that more suitable building blocks can become fundamental elements of newer architectures, to capture temporal evolution more accurately. For this purpose, we design new configurations of an already existing neural network, in order to compare different building blocks, which contribute to improving the original model.

1.4 Application

The applications covered in video classification mostly depend on the type of videos we want to classify: sports, surveillance cameras, movies and more. For this reason many task specific datasets recently came out to help researchers in new developments.

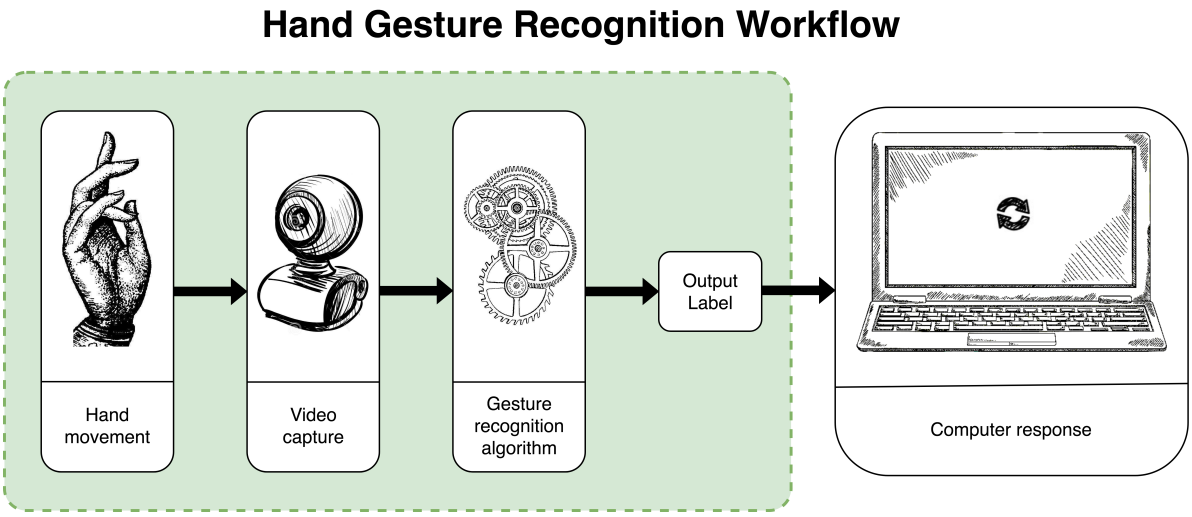


Figure 1.4: Hand gesture recognition workflow. The higlighted part in green shows the steps that are covered in this thesis.

Our research focuses on the task of **gesture recognition**, which is defined as the interpretation of human gestures by a computer. The types of gestures can be categorized in three main groups: hand gestures, body gestures and facial gestures. Depending on the application we wish to develop, we choose a subset of gestures to be classified by the computer. As a new way of interacting, gesture recognition is shown to have a strong potential in different fields, for example

- *sign language recognition*: to facilitate deaf and non-deaf people to comunicate;
- *video games interface*: to provide a more realistic and controller-free interface with the game;
- *human-computer interaction*: to support users to interact with a computer with simple gestures performed in front of a video camera.

In the work of this thesis we focus on human-computer interaction. We apply our solutions to the video stream captured by a computer camera, where the goal is to correctly classify a set of human hand gestures, performed in front of the computer. Similarly to a touchpad or a touchscreen, which link the movement of the fingers on top of it to a computer response, our solution permits a computer to understand different gestures through a video camera and to associate them to a corresponding action, without the help of any external device. In figure 1.4 we show the workflow steps for a hand gesture recognition system.

1.5 Contribution and Acknowledgements

The work of this thesis has been produced in collaboration with the company Addfor s.r.l., under the supervision of Kent Danielsson and the help of all the other company members.

I wish to thank the university supervisor, professor Elio Piccolo, for the valuable advices and the revision work.

The final thanks go to my flatmates, my girlfriend, my family and all the friends, who supported me during this years.

Chapter 2

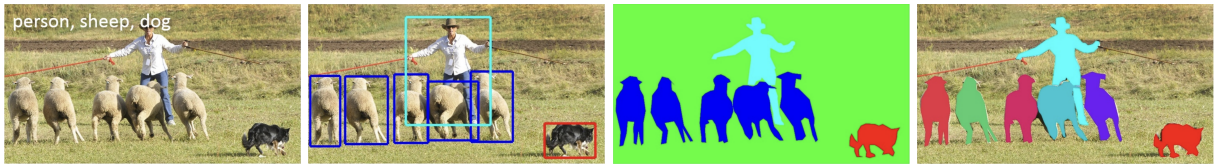
Video Classification

Deep learning architectures have been very successful in computer vision, producing results comparable and in a few cases superior to human accuracy. In particular, after the presentation of the AlexNet architecture [20], in the 2012 ImageNet competition, convolutional neural networks start spreading for image content analysis, such as:

- *image classification*: assigning a label to an input image, given a fixed set of categories, figure 2.1a;
- *object detection*: localizing and classifying instances of objects in an input image, figure 2.1b;
- *semantic segmentation*: pixel-wise partitioning of an image into regions (segments), each having the same semantic content, figure 2.1c;
- *instance segmentation*: pixel-wise partitioning of the individual entities within a scene, figure 2.1d;
- *image captioning*: generating text, which describes the content of the image, figure 2.1e.

After AlexNet, many other architectures had been proposed aiming at higher performances, faster computations and greater generality.

While many of the previously listed tasks have been widely developed and many solutions have been proposed, much less work has been done regarding video related tasks. The upgrade of computer performances and the growth of video data on internet



(a) Image classification (b) Object detection (c) Semantic segm. (d) Instance segm.

(e) Image captioning

Figure 2.1: Examples of different image related tasks. Taken from COCO dataset [22].

help researchers to develop new deep learning architectures for video content analysis. The major contributions have been done in **video classification**, which is a form of pattern recognition applied to videos. Similarly to image classification, the objective here consists in training a classifier to identify objects, animals, actions or other types of content from a video, in order to output the correct label from a set of categories, whose semantic content depends on the considered application.

We can distinguish between **supervised** and **unsupervised learning**, depending on the type of input data. Supervised machine learning algorithms are trained on already labeled data. In this case the algorithms adjust their parameters to predict the correct output for the largest number of input data from the training set. On the contrary, unsupervised algorithms base their training on unlabeled data. In this case there is no correct answer and the algorithms can autonomously discover interesting structure in the data.

As for the majority of the machine learning algorithms, the approaches adopted for our task are based on supervised learning.

2.1 Video Overview

Videos are discrete temporal sequences of images/frames, displayed in rapid succesion to simulate movement. The higher the *frame rate* (frame per second) and *frame resolution* (number of pixel of each frame), the greater the affinity with the real scene that is recorded. Our studies consist in enabling computers to capture spatial and temporal information from videos, analogous to humans comprehension of the real world through the visual system.

While the recognition of objects or animals can be done from still pictures, their interactions with the surrounding environment can be equivocal from static images. To collect such information we analyze the temporal evolution of videos, to determine more clearly *what is happening*. For example, analyzing an image of a football player shooting the ball loses a lot of information from the original scene. First, a shot might be mistaken for a cross and, even though a more expert eye could distinguish between the two types of actions from the body position of the player the image does not provide enough clues to answer to the following questions: *Did the player score? Was the shot blocked by someone? Where did the ball go?*. This briefly introduces why working with video increases the number of useful information to extract, in order to answer in a more exhaustive way not only to the question *Who/What is appearing?*, but also to *What is going on?*.

The additional temporal dimension in the video domain makes the problem more complex to handle compared to images. For such reason, it is fundamental to implement a solution capable of capturing representative patterns from multiple frames and not on a single frame level. In terms of deep neural networks, in order to extract spatio-temporal informations, the straightforward solutions consist in a form of transfer learning from image based CNNs. Transfer learning refers to the reuse of a pre-trained neural network for a different but releted problem. Most of the times CNNs are not trained from scratch with random initialization, but they frequently exploit pre-trained models on larger and more generic datesets as good initialization.

Different approaches are implemented, depending on the type of videos in consideration: **trimmed** or **untrimmed**. Trimmed videos are short segments of videos, containing a particular action, object or event for their entire length. On the contrary, untrimmed videos are full videos, where only short parts have significant content. In other words, from an untrimmed video we can extract as many trimmed videos as the number of signif-

Name	Type	Size	Classes	Year
Hollywood2[1] [23]	Action & scene recognition	3669	22	2009
HMDB51 [21]	Action recognition	6766	51	2011
UCF101 [32]	Action recognition	13320	101	2012
Sports-1M [18]	Sport action recognition	1100000	487	2014
ActivityNet [3]	Action recognition	28000	203	2015
YouTube-8M[2] [1]	Video classification	8000000	4716	2016
20BN-Something-Something [11]	Action recognition	108000	174	2017
20BN-Jester [37]	Hand gesture recognition	148000	27	2017
Moments in Time [24]	Action recognition	1000000	339	2017
Kinetics [19]	Action recognition	306000	400	2017

Table 2.1: Some of the most relevant video datasets used for computer vision applications.

icant segments. For this reason working with untrimmed videos requires more attention in the selection of the input frames.

Furthermore, the presence of camera motion in videos can often reprensent an obstacle in the prediction of our model. To overcome this challenge, it is important to develop robust model trained on datasets with a wide variety of videos, containing both camera motion and fixed camera.

Another difficulty, also faced when working with images, is how to extract 3D information from videos. For this reason a few multi-view datasets have been published, where the same video is recorded from different angles, with multiple cameras.

2.2 Tasks and Datasets

While the research on images spread widely on numerous tasks, working with videos reached satisfying results in a limited number of tasks. Due to the high complexity of videos, the prospect of a machine capable of understanding at a human level the surrounding world through visual inputs is still out of reach. For this reason the wide field of video understanding has been partitioned in subtasks and different approaches have been proposed, depending on the type of problem.

For each task, a satisfying number of public datasets are now available and are used as benchmarks, in order to save time and resources for researchers, who can only focus on implementing the best architectures. Datasets play a crucial role in machine learning, shaping a model in what and how well it can learn. The size, in terms of number of videos,

[1]The classes of Hollywood2 are divided in 12 action classes and 10 scene classes.
[2]YouTube-8M contains a mix of video entities, such as activities, objects, scenes and events.

and the variation of videos in each class (different lights, viewpoints, contexts, etc.) are fundamental for the ability of models to generalize. Very large datasets are especially necessary for deep neural networks. For this reason, recent datasets show a conspicuos growth in size and number of categories. Moreover, the number of classes and what they represent are significant for the type of application of interest. For instance, we have sport datasets, movie scenes datasets and dynamic scenes datasets. Also the most prevalent human activity datasets can differentiate between heterogeneus action sets, typically a wide variaty of actions in various situations or scenarios, and specific action sets, such as detection of abandoned objects, recognition of activities of daily living, crowd behaviour, detection of human falls, gait analysis, or pose and gesture recognition.

Furthermore, the way actions are grouped inside of datasets can strongly characterize the concepts learned by models. Usually datasets contain high-level actions, where the goal is to choose a class in a discriminative way, teaching the model to recognize what it univocally characterizes such action. As a drawback, this classification of actions do not allow the model to learn motion primitives, such as “moving”, “pushing”, “dropping”, “holding”, “poking” and many others, that are intrinsically contained in high-level actions. For example, the phrase “opening SOMETHING” will have drastically different visual counterparts, depending on whether “SOMETHING” in this phrase is replaced by “door”, “zipper”, “blinds”, “bag”, or “mouth”. However, the concept of “opening” has some common features, such as the movement of something to allow access, passage, or a view through an empty space. To truly understand these concepts, the ability to generalize among all the possible use cases and to learn the common features is required. For this specific purpose, the ”something something” video database was introduced [11].

In the following sections, we present some of the most heavily reserched tasks.

2.2.1 Dynamic Scene Recognition

From the early research on image-based scene classification, new studies on video-based dynamic scene recognition came out as a natural evolution of this field. Here the target consists in classifying scenes from dynamic patterns, in a wide range of circumstances. For example, if the scene in consideration is a waterfall, the water flow represents an important feature, not taken in consideration when working with images.

This area of studies can be useful to provide priors for subsequent operations, such as

(a) Beach (b) Forest Fire (c) Highway (d) Waterfall

Figure 2.2: Sample frames of dynamic scenes, taken from YUP++ Dynamic Scenes dataset [9].

object and action recognition, or it could be useful in browsing image or video databases.

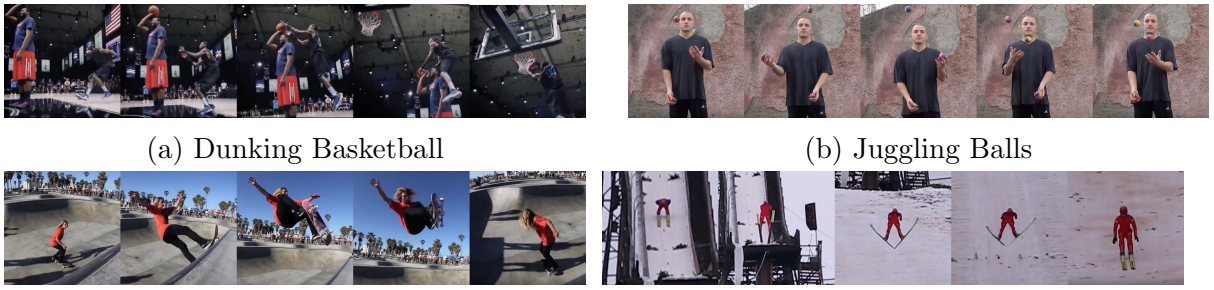
Recent results show that video-based deep learning approaches greatly outperforms both static or hand-crafted methods.

A new challenging dataset of dynamic scenes, called YUP++ [9], has recently been released, which more than doubles the size of those previously available. It contains 20 classes and they are grouped in two subsets: with static camera and with camera motion, in order to highlight the different performances in both situation and to build more robust models. The set of classes contained in the dataset are: beach, city street, elevator, forest fire, fountain, highway, lightning storm, ocean, railway, rushing river, sky clouds, snowing, waterfall, windmill farm, building collapse, escalator, falling trees, fireworks, marathon and waving flags. We see some example in figure 2.2.

2.2.2 Action Recognition

Video surveillance, human-computer interaction, learning for robotics, web-video search and retrieval, medical diagnosis, retail analytics, elderly care, sports analytics and many other high-impact societal applications made the understanding of human actions an essential task to accomplish in computer vision; with the term action we mean a "meaningful interaction" between humans and the surrounding environment. Given all these applications, action recognition became the most heavily researched task in the field of video classification, due to the impossibility of manually analyzing all data. Here the task consists in the recognition of human activities, which can be grouped in the following macro-classes:

- *body-motion*: e.g. running, jumping, smiling;
- *human-human interaction*: e.g. shaking hands, kissing, hugging;
- *human-object interaction*: e.g. writing, driving, drinking.



(a) Dunking Basketball

(b) Juggling Balls

(c) Skateboarding

(d) Ski Jumping

Figure 2.3: Sample frames containing human actions, taken from Kinetics human action video dataset [19].

In this challenge, temporal features play a fundamental role. An action is strongly characterized by its temporal evolution and it is necessary to distinguish among ambiguos classes. For example, a single frame is usually not sufficient to determine if a person is sitting or getting up, eating or yawning etc.

The huge amount of existing actions and the numerous possible ways these can be performed require very large datasets, to improve the performances of models. However, this is not always possible and many of them include only the classes needed for a specific application.

As shown for image classification and image detection, a conspicuous boost of performances on small datasets is noted when a model is pre-trained on larger ones, such as ImageNet [6] or COCO [22]. For this reason the Kinetics human actions dataset [19] was proposed, containing 400 classes with at least 400 video clips for action, where each clip is obtained by cropping 10s from a YouTube video. Some examples are shown in figure 2.3.

2.2.3 Event Recognition

The recognition of events, defined as a set of characterizing concepts, is a more complex task. Here the surrounding context, the objects and the actions involved have fundamental roles in correctly classifying such events. For example, a party can be characterized by a group of people dancing (body-motion), people talking (human-human interaction), beverages and food (objects) and many other concepts. Likewise, every sport is defined by a set of human actions (running, jumping, etc.), objects (balls, rackets, etc.) and location (gym, swimming pool, etc.).

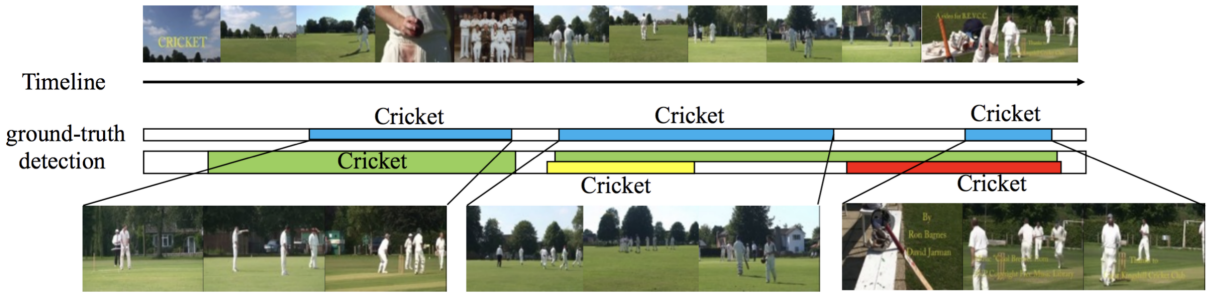


Figure 2.4: Example of action detection, where blue lines denote the ground-truth instances, the red and yellow lines denote the cases of bad localization and multiple detection, respectively. Image taken from [43].

2.2.4 Temporal Action Detection

This field is not stricktly related with video classification, since the goal is not simply to attribute a label to a video, but it consists in detecting the temporal window in which an instance takes place. This task is more a step beyond action recognition. More precisely, the objective is not only the classification of human actions appearing in continuous, untrimmed video streams, but also the localization of the activity in time. One common way to handle this task is to adopt the paradigm *proposal + classification* [42], inspired by *bounding box proposals* from object detection methods [28].

Chapter 3

Methods

While image content analysis requires only the extraction of spatial features, for video understanding we wish to capture also the temporal features in order to have a better insight of how spatial information evolve over time. 2D convolutional layers have been the core building block for spatial feature extraction, which simply look for spatially small patterns at different level of abstraction, such as edge orientation, depending on the depth of the architecture where the layers are located. For example, the first hidden layer, which directly "looks" at the input image, activates when some relatively simple patterns are spotted, e.g. the filters shown in figure 1.3; the next few layers learn to recognize collections of shapes like eyes or noses, and deeper layers learn even higher-order features, like faces.

Based on this intuition, numerous new CNNs architectures have been developed, which tend to chain more and more convolutional layers in a deeper fashion. This allows the CNN to learn meaningful patterns from complex data, such as images, and to improve generalization. On the contrary, very wide and shallow architectures can be very good in memorization, but they easily tend to overfit the training dataset. Numerous deep image classification networks have been proposed, such as Inception [33], VGG-16 [31], ResNet [12] and DenseNet [35], and used as a valuable initialization for video understanding.

A trivial solution to extend these models to video classification can be thought of as follow: using an image classification network to output a label for each distinct frame and assigning to the whole video the most prevalent label. This can yield good accuracy, but it does not extract any information on the temporal evolution of the video, which represents the second main component to correctly classify videos. For example, this solution is not

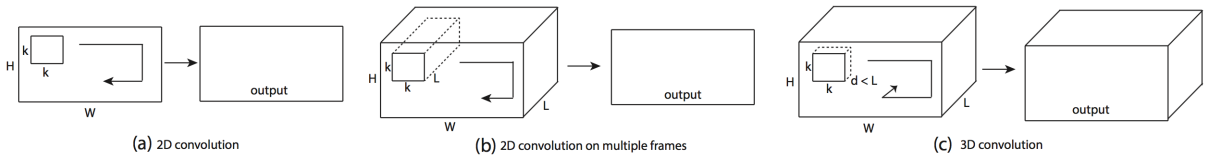


Figure 3.1: **2D and 3D convolution operations.** a) Applying 2D convolution on a 2D input. b) Applying 2D convolution on a 3D input. c) Applying 3D convolution on 3D input, preserving temporal information of the input signal. Image taken from [34].

capable of distinguishing between an opening door or a closing door, since the network captures well the spatial information about the door, but it is not able to collect any clues about the movement of the door.

Different type of spatio-temporal features extractors have been proposed, adjusting for our tasks the previously mentioned image-based architectures, with specific solutions. They can be grouped into three main categories:

- 3D-CNNs.
- Two-Stream networks.
- RNN-based networks.

3.1 3D-CNNs

An intuitive solution proposed to tackle this problem is extending the convolutional layers in a 3D fashion [16], which is implemented by using 3D kernels, figure 3.1. In this way the new architectures are able to extract not only spatial features, but also to retrive temporal information between adjacent frames. It was empirically demonstrated by [34] that $3 \times 3 \times 3$ is the best kernel size to implement 3D convolution, currently used in most of the proposed networks based on 3D-CNNs, such as Inflated 3D ConvNet (I3D) [4].

This approach achieves better results than previously proposed hand-crafted methods. However, some problems emerged in using this type of architecture. First, the high computational cost and high memory demand of these network require great computational resources (GPUs). Second, this solution makes it extremely difficult to train very deep networks from scratch.

3.1.1 3D Convolution

During the forward step, 2D convolution is performed by convolving across the width and height of the input volume, and computing a dot product between the convolutional filters and the input region, in order to extract spatial features. More properly, the output value v_{ij}^{xy} of a 2D convolutional layer i, at position (x, y) of the jth feature map, where $j = 1, ..., F_i$ and F_i is the number of filters (or kernels) inside of the layer, is computed as

$$v_{ij}^{xy} = \phi\left(b_{ij} + \sum_{m=0}^{F_{i-1}-1} \sum_{p=0}^{H_i-1} \sum_{q=0}^{W_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)}\right). \tag{3.1}$$

ϕ is an activation function such as *tanh*, *ReLU*, $\max(0, x)$, sigmoid or other *non-linear* functions, essential for neural networks to compute nontrivial problems; b_{ij} is the bias of the analyzed feature map; w_{ijm}^{pq} is the value of the jth kernel at position (p, q) linked to the mth feature map from the previous layer; H_i and W_i are the height and the width of kernels in the ith layer. For a 2D convolutional layer the set of parameters to train is $w_i \in \mathbb{R}^{F_i \times F_{i-1} \times H_i \times W_i}$.

This solution is extended for videos in a 3D fashion, in order to capture representative patterns in multiple adjacent frames. This is achieved by convolving 3D kernels not only spatially, but also along the temporal dimension of the 4D input hyper-volume, thereby capturing motion information. The value at position (x, y, z) of a feature map j, obtained by the 3D convolution operation, is now

$$v_{ij}^{xyz} = \phi\left(b_{ij} + \sum_{m=0}^{F_{i-1}-1} \sum_{p=0}^{H_i-1} \sum_{q=0}^{W_i-1} \sum_{r=0}^{D_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right), \tag{3.2}$$

where D_i is the size of the 3D kernels in the ith layer along the temporal dimension. For a 3D convolutional layer the set of parameters to train is $w_i \in \mathbb{R}^{F_i \times F_{i-1} \times H_i \times W_i \times D_i}$. As previously mentioned, the size of H_i, W_i and D_i are generally set to 3.

3.1.2 Layer Redistribution

Afterwards, a new redistribution of 3D convolutional layers has been proposed, in order to overcome the previously listed problems. In particular, $3 \times 3 \times 3$ convolutions are rearranged as a combination of $1 \times 3 \times 3$ filters (equivalent to 2D spatial convolution) plus

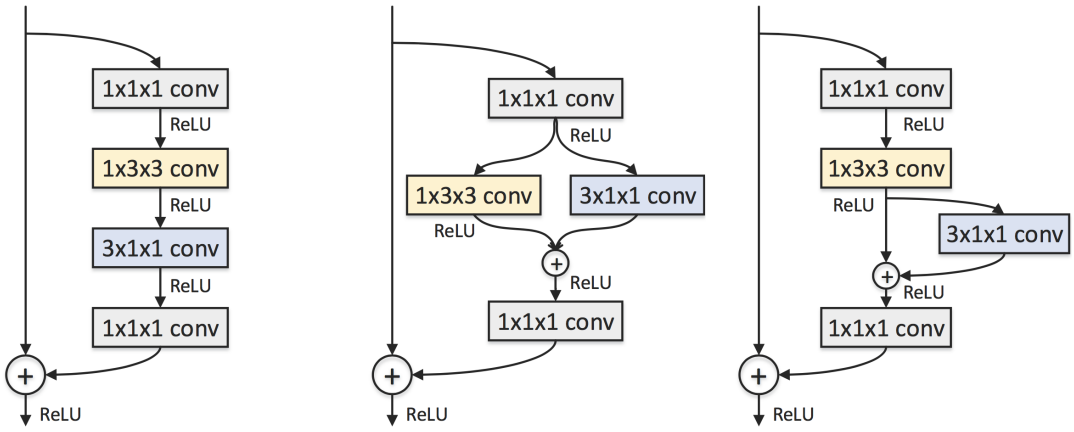


Figure 3.2: Different Pseudo-3D Convolutional blocks built on top of Residual Unit. Image taken from [27].

$3 \times 1 \times 1$ filters (equivalent to 1D temporal convolution) in a parallel or series fashion[1]. With this approach spatial and temporal computations are decoupled, in the way that spatial convolutions are computed as in eq. 3.1 and the value obtained from temporal convolution, at position z of the jth feature map, is

$$v_{ij}^{z} = \phi\left(b_{ij} + \sum_{m=0}^{F_{i-1}-1} \sum_{r=0}^{D_i-1} w_{ijm}^{r} v_{(i-1)m}^{z+r}\right), \tag{3.3}$$

with $w_i \in \mathbb{R}^{F_i \times F_{i-1} \times D_i}$ as set of learnable parameters and D_i usually set to 3.

Multiple ways of arranging these two layers have been exploited (figure 3.2) and the efficiency of different configurations depends specifically on the type of architecture used as backbone (Inception, ResNet, etc.). We call this redistribution as **pseudo-3D convolution**, from the paper [27]. By adopting this solution, the number of parameters of the networks decreases and it facilitates the fine-tuning of pre-trained models. Nontheless, the accuracy of the models tends to increase.

3.2 Two-Stream Networks

First proposed by [30], the two-stream approach takes inspiration from the widely accepted two-stream hypothesis in the human visual system [10]. The hypothesis argues that humans have two pathways for capturing visual information:

[1]This notation presumes as dimensions ordering: $frames \times hight \times width$. Other notations could use $3 \times 3 \times 1$ filters for 2D spatial convolution and $1 \times 1 \times 3$ for 1D temporal convolution.

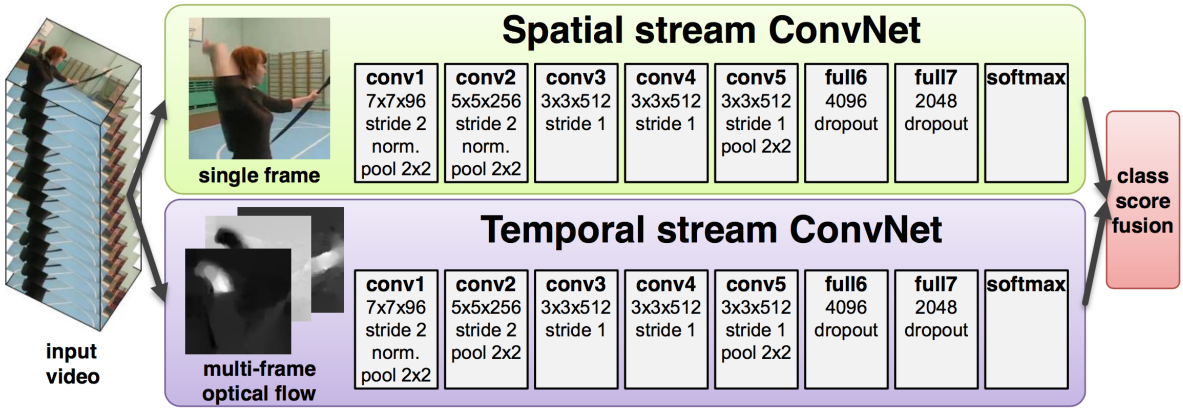


Figure 3.3: **Two-Stream architecture**. Image taken from [30].

- the ventral stream (or "what pathway"), which is in charge of the recognition of objects;
- the dorsal stream (or "where pathway"), which is involved in determining the objects' spatial location relative to the viewer.

Similarly, the Two-Stream architectures, proposed for learning spatial and temporal features, incorporates two separate streams:

- the **spatial/appearence stream**;
- the **temporal/motion stream**.

Each stream is implemented with a deep CNN, normally the same architecture for both, and the two scores are then combined with a fusion method (averaging, SVM, etc.).

What characterizes the two streams is the input they receive. The spatial stream takes as input the RGB frames and it is able to capture static features[2]. It is shown that the spatial stream can represent a competitive solution on its own, but it fails in the distinction of those classes with the same spatial representation and different temporal evolution. For this reason the temporal stream is introduced, which normally receives as input optical flow [14].

3.2.1 Optical Flow

Optical flow describes the apparent motion between two consecutive frames caused by the movement of objects or camera motion. It is represented as a 2D vector field, where

[2]This stream is nothing else than the naive approach described before.

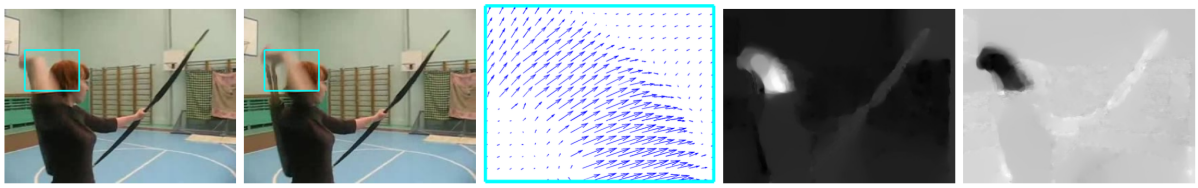


Figure 3.4: **Optical Flow**. Image taken from [30].

each vector shows the direction and the intensity of the movement in a specific location (pixel-wise). It is computed on top of the brightness constancy constraint, which asserts that the brightness of two adjacent frames remains constant. In other words, given the image intensity $I(x, y, t)$ at point (x, y) and time t, the previews assumption affirms that

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t). \tag{3.4}$$

Expanding the right hand side with Taylor series approximation, we obtain

$$I(x, y, t) = I(x, y, t) + \delta x \frac{\partial I}{\partial x} + \delta y \frac{\partial I}{\partial y} + \delta t \frac{\partial I}{\partial t} + \epsilon. \tag{3.5}$$

Subtracting $I(x, y, t)$, dividing by δt and in the limit of $\delta t \to 0$, we get

$$\frac{dx}{dt}\frac{\partial I}{\partial x} + \frac{dy}{dt}\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0, \tag{3.6}$$

which can be rewritten as

$$\nabla I \cdot V^T = -I_t, \tag{3.7}$$

where $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$ and $V = \left[\frac{dx}{dt}, \frac{dy}{dt}\right] = [u, v]$ is the velocity vector or optical flow of $I(x, y, t)$. Since we have two unknown variables (u, v), we need another constraint to solve this equation. Many methods have been proposed, the one implemented by OpenCV[3] is based on the Lucas-Kanade method. This solution is based on the assumption that neighbouring pixels have similar motion, in particular, the Lucas-Kanade method takes a 3×3 patch as neighbours. We have now 9 equations for two unknown variables, hence the

[3]This implementation is used by many of the Two-Stream networks.

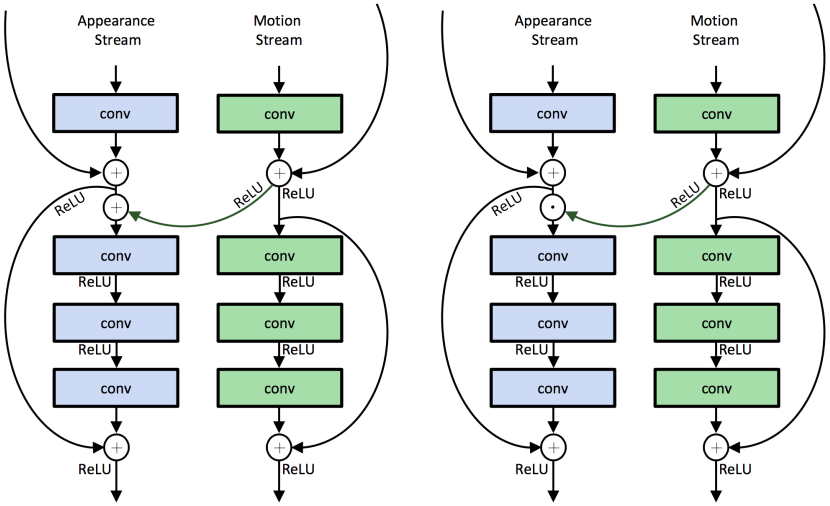


Figure 3.5: Residual connection (on the left) and multiplicative connection (on the right). Image taken from [8].

system is over-determined, and by applying the least square fit method we finally obtain

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_{x_i}^2 & \sum_i I_{x_i} I_{y_i} \\ \sum_i I_{x_i} I_{y_i} & \sum_i I_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_{x_i} I_{t_i} \\ -\sum_i I_{y_i} I_{t_i} \end{bmatrix}. \tag{3.8}$$

As a drawback, optical flow requires intensive computations and it has to be computed before training. For this reason RGB difference has been proposed [39] as a faster and competitive alternative to optical flow, suitable for real-time applications.

3.2.2 Improving Two-Stream Networks

The research on Two-Stream networks has brought to many improvements, for example introducing 3D or pseudo-3D convolutions inside the networks, to enable each stream to capture spatio-temporal features autonomously.

Furthermore, [7] and [8] have proposed new types of connections between motion stream and appearence stream, to allow interaction between the two streams (figure 3.5). They are placed at different locations of the network, in order to exchange information at many possible scales.

Another solution is to extend the Two-Stream architectures with an additional Audio stream, able to extract relevant information from the audio of a video.

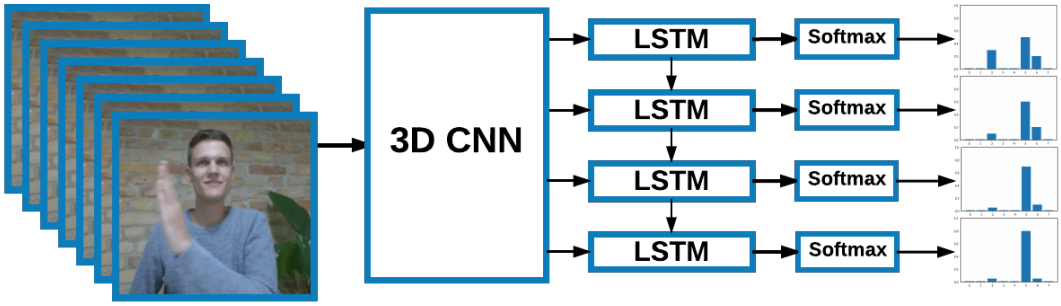


Figure 3.6: Example of architecture 3D-CNN + LSTM. Image taken from [36].

3.3 RNN-based Networks

Another solution is based on modelling long-term sequences with Recurrent Neural Networks (RNNs). This is done by connecting a multi-layer RNN to the output of a CNN, to enable the network to extract spatio-temporal features from the global content of the video. RNNs are adequate solutions for video related tasks, since they can directly map variable-length input and can model complex temporal dynamics. Most of the implementations use Long Short-Term Memory (LSTM) networks [13] or Gated Recurrent Units (GRUs) [5]. Similarly to previous methods, also for this type of architecture, replacing 2D-CNN with 3D-CNN or Pseudo-3D-CNN can improve the network performance effectively.

This approach has not yet reached the accuracy of Two-Stream networks, but it is mostly used for real-time applications. This is due to its ability of extracting long-term temporal features from continuous video streams.

Chapter 4

Networks Architecture

In order to choose the best deep neural network to use as a backbone for our experiment, several aspects were taken into account. First, we choose to implement our research in TensorFlow with Python interface, due to its highly flexible system architecture and the high amount of documentation and contributors, which is arguably considered one of the best deep learning framework. Second, we select a model with pre-trained weights on large datasets, in order to speed up the fine-tuning process on new datasets and improving the achieved results. Third, we select a network with a simple architecture, to better evaluate when new solutions boost the model accuracy.

For these reasons, we adopt in this research the Inflated 3D architecture (I3D) [4], already implemented in TensorFlow and pre-trained on ImageNet and then Kinetics-400 training sets. For computational reasons, we decide to limit our reasearch only on the RGB stream of the architecture and to discard the flow stream, which requires longer computations for the optical flow extraction and the training process of Two-Stream networks.

This architecture obtains great results on multiple datasets and it was the pioneer in Kinetics dataset training, generated by the same team. They demonstrate how, using pre-trained weights on Kinetics, contributes to boosting the fine-tuning process on smaller datasets, similarly to ImageNet for image classification. We give a more detailed description of the I3D and the novel architecture configurations in the next subsections.

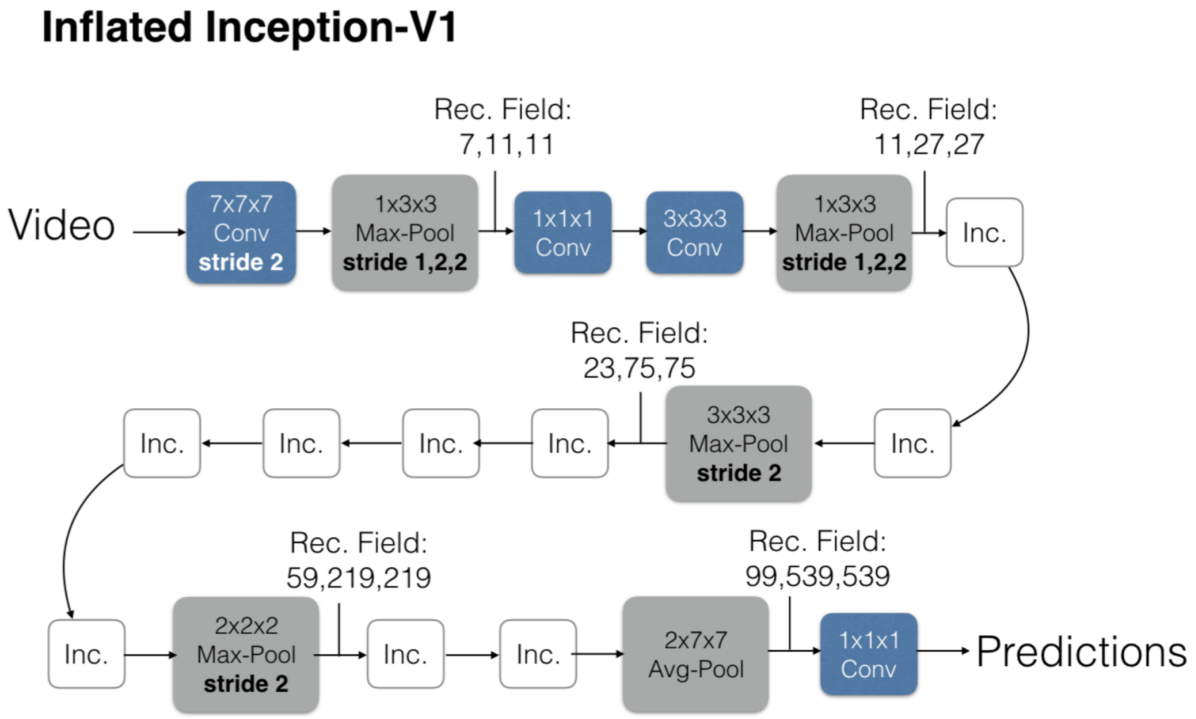
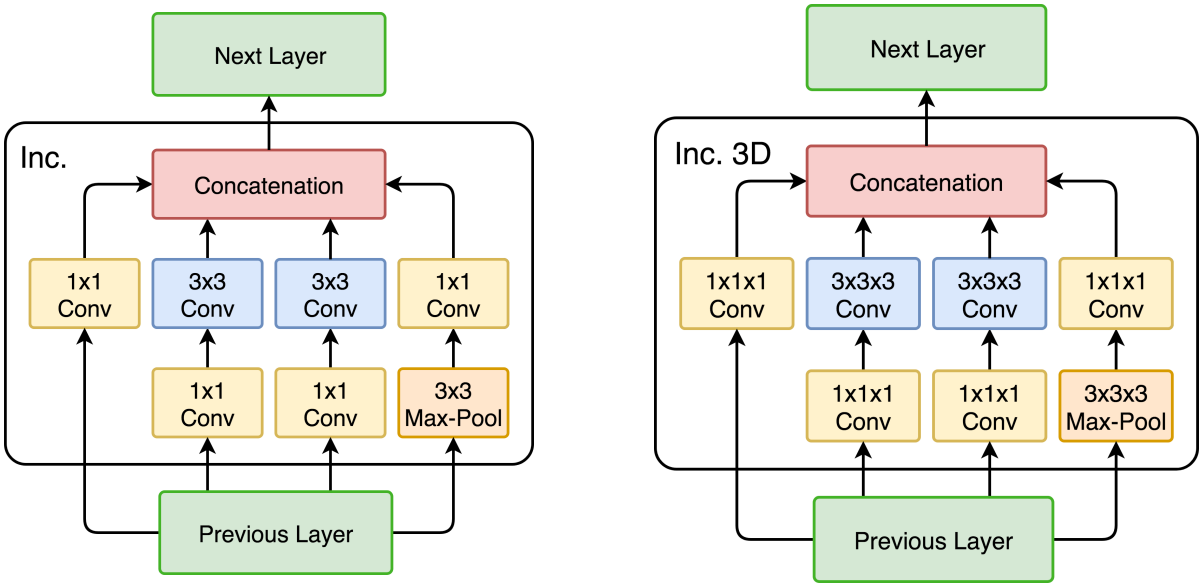


Figure 4.1: The Inflated Inception-V1 architecture. Where Inc. indicates the inflated inception module, shown in figure 4.2b. The stride of convolution and pooling operators is 1 where it is not specified. The batch normalization layers, ReLUs and the final softmax are not shown. The theoretical sizes of receptive field sizes for a few layers in the network are provided in the format "time,height,width" – the units are frames and pixels. The predictions are obtained convolutionally in time and averaged. Taken from [15].



(a) Inception module with dimensionality reduction [33].

(b) Inflated inception module with dimensionality reduction [4].

Figure 4.2: Comparison of 2D inception module (left) and 3D inception module (right).

4.1 I3D Architecture

Image classification has led the researchers to develop very deep convolutional neural network, in order to capture a higher number of features and hence improving the accuracy. Later, many of these networks have been reproposed and adapted to carry out video related tasks. In our case, the Inflated 3D network exploites the Inception-v1 architecture [33], reshaping the 2D weights, pre-trained on ImageNet, as valuable initialization for the 3D convolutional layers. This is obtained by repeating the weights of the 2D kernels N times along the temporal dimension of the corresponding 3D kernels and dividing them by N.

The basic unit of the Inception-v1 architecture is called **inception module**, shown in figure 4.2a. This is composed by a set of convolutional layers with different filters' sizes (1×1 and 3×3) and a 3×3 max pooling layer, arranged in both parallel and sequential ways; lastly the outputs are concatenated. The 1×1 convolutional layers is placed before the 3×3, as a form of embedding, in order to reduce the number of parameters, removing computational bottlenecks.

In the I3D architecture, with inflation we mean the transformation of the inception blocks in a 3D fashion, in particular we transform 2D convolution (N×N) into 3D convolution (N×N×N), as we previously described. In figure 4.2 we show the comparison

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 4.3: Batch Normalizing Transform. Taken from [15].

between the inception module structure, in the Inception-v1, and its inflated version, in the I3D architecture.

Moreover, in the I3D network, a batch normalization layer is used [15], as explained in figure 4.3, and a Rectified linear unit (ReLU) activation function

$$ReLU(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{4.1}$$

following each convolutional layer, except for the last one.

Furthermore, other adjustments on the original Inception-v1 architecture are made, to better face the temporal evolution of videos. Particular attention was paid to the **receptive field** in the temporal dimension. The receptive field is the region in the input space that a feature is looking at. This is characterized by its center location and its size. The size of the receptive field of a layer i along one dimension can be computed recursively as

$$l_i = l_{i-1} + (k_i - 1)\prod_{j=1}^{i-1} s_j, \tag{4.2}$$

where l_{i-1} is the size of the receptive field of the previous layer, k_i is the filter size of layer i and s_j is the stride of layer j. Moreover, the closer the pixels to the center of the receptive field, the more they contribute to the output of the feature, which suggests that a feature focuses its attention on the center of such region.

In spatial convolution, the receptive field is treated symmetrically along the horizontal

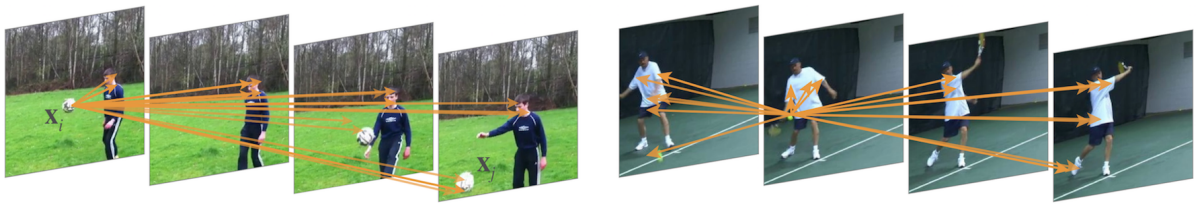


Figure 4.4: Examples of long range dependencies from distant pixels in space and time. Images taken from [40].

and vertical extentions of the input image, since equal strides and kernels are generally adopted and the input image has usually equal height and width.

Different considerations have to be made regarding temporal convolution. Here, the receptive field corresponds to the number of frames from the input space that are seen by a temporal feature. For this reason, it was chosen for the first two max-pooling layers to adopt asymmetric filters ($1 \times 3 \times 3$) and temporal stride equal to 1, while adapting the remaining max pooling layers with symmetric structures.

The final average pooling layer has kernel equal to $2 \times 7 \times 7$ or $2 \times 4 \times 4$ depending on the size of the input image, 224×224 and 100×100 respectively.

It is shown in figure 4.1 the complete architecture and the receptive field sizes of few layers.

4.2 Non-Local Operators

Most of the architectures proposed for video understanding usually just leverage on the strength of image related CNNs, while only a few ad hoc solutions have been proposed. To further improve the modelling of temporal evolutions in videos, we believe that is necessary to investigate new solutions, specifically designed to capture spatio-temporal features. For this reason, we start analyzing two non-local operations as new components of the network, in order to capture dependencies from distant pixels in space and time, as shown in 4.4. These operations are designed to learn features from any two positions, on the contrary to convolution, which operates on local neighbours of the input, separately.

Our aim is to test and compare such non-local operations on the I3D architecture, seeking for a higher accuracy of the network.

First, we introduce the **non-local block**, proposed by [40]. Here the non-local oper-

ation is performed as

$$y_i = \frac{1}{C(x)} \sum_{\forall j} f(x_i, x_j) g(x_j), \tag{4.3}$$

where x and y are respectively the input and output signals of the block, both with equal size. The function f captures the relationship between different locations of the input i and j; g computes a representation of the input signal and $C(x)$ corresponds to a normalization factor. Notice that all positions in space and time ($\forall j$) are put in relation with each other.

More precisely, the normalization factor is computed as $C(x) = \sum_{\forall j} f(x_i, x_j)$, g represents a linear embedding computed with a $1 \times 1 \times 1$ convolution weight matrix W_g, which reduces in half the number of the input channels:

$$g(x_j) = W_g x_j. \tag{4.4}$$

Furthermore, we choose from [40] the *embedded gaussian* implementation of f:

$$f(x_i, x_j) = e^{\theta(x_i)^T \phi(x_j)}, \tag{4.5}$$

with $\theta(x_i)$ and $\phi(x_j)$ representing two linear embeddings similar to g. The eq. 4.3 represents a generalization of the self-attention module from [38], used in machine translation. For a given i, $\frac{1}{C(x)} f(x_i, x_j)$ represents the softmax along the dimension j. For this reason, we can rewrite eq. 4.3 as

$$y = \text{softmax}(x^T W_\theta^T W_\phi x) g(x). \tag{4.6}$$

Finally, the non-local block is defined as

$$z_i = W_z y_i + x_i \tag{4.7}$$

where W_z computes a position-wise embedding, also computed with $1 \times 1 \times 1$ convolution. In this case we double the number of channels, in order to output a matrix matching the size of the input x. A representation of the non-local block is shown in figure 4.5. This block follows the structure of residual connections, typical of ResNets architectures [12], which are the networks where these blocks were originally introduced [40]. We

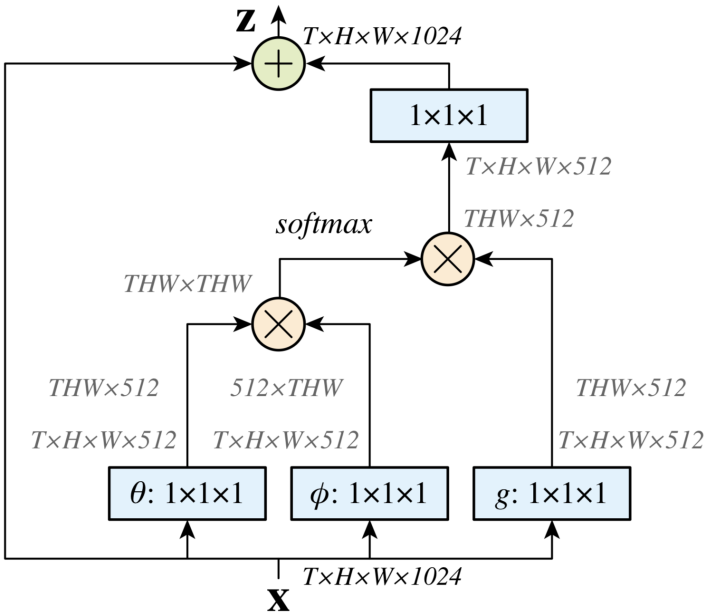


Figure 4.5: A spacetime non-local block. The shape of the feature maps are shown, e.g. $T \times H \times W \times 512$ for 512 channels. $1 \times 1 \times 1$ convolutions are represented as blue boxes, $\otimes$ denotes matrix multiplication and $\oplus$ denotes element-wise sum. Taken from [40].

study if non-local blocks guarantee a substantial improvement, also on a network with an extremely different structure from ResNet, such as I3D.

In the I3D architecture, we choose to insert only 2 non-local blocks, in order to prevent the overloading of its computational complexity. We place the 2 blocks at different depths, to extract non-local features at distinct levels of abstraction. Following [41], which shows that higher-level layers better model temporal evolution compared to lower-level's one, we choose to include the non-local blocks in between deeper and more abstract inception modules. In particular, one is set in between the 6th and 7th inception module and the other right before the last inception module. We refer to this modified version of the architecture as **Non Local I3D** (**NL-I3D**).

Another solution for combining information across space and time was recently proposed by [41], called **spatiotemporal feature gating**. The purpose of this paper is to directly improve the I3D network, first by replacing 3D convolutions with spatiotemporal-separable 3D convolutions (pseudo-3D), which are mentioned in chapter 3.1.2. However, we do not further investigate on this direction.

The spatiotemporal feature gating is defined as

$$y_i = \mathrm{A} \otimes x_i, \tag{4.8}$$

where $x_i \in \mathbb{R}^D$ is a feature vector at a specific time frame and position of the full feature matrix $x \in \mathbb{R}^{T \times W \times H \times D}$. The feature vector x_i is multiplied elementwise by an adaptive weight vector $\mathrm{A} \in \mathbb{R}^D$, computed as

$$\mathrm{A} = \sigma(\mathrm{W} avg\_pool_{ST}(x)), \tag{4.9}$$

with $W \in \mathbb{R}^{D \times D}$ representing a weight matrix, $\sigma()$ the sigmoid function and $avg\_pool_{ST} : \mathbb{R}^{T \times W \times H \times D} \rightarrow \mathbb{R}^D$ is a spatiotemporal average pooling function.

Following the design of the S3D-G architecture from [41], we place such feature gates after the 3D convolutional layers inside of inception modules with a filter size greater than $1 \times 1 \times 1$, and again we insert the gates in the 6th and the second-to-last inception modules, similarly as before, to better compare this solution with the non-local block. We call this version of the architecture as **Gated I3D** (**G-I3D**).

4.3 Temporal Convolution Dilation

In this section we further investigate how to capture long range dependencies between frames, in order to help the network to learn global aspects of videos from distant frames. The small motion features captured by the standard temporal convolution (included in 3D convolution) are often not sufficient to adequatly identify an action. For this reason, **random dilation** is included in the temporal dimension of the 3D convolutional filters. Such solution exponentially increases the receptive field of the network and captures more contextual information. This choice is inspired by other research on fields characterized by temporal evolution, such as text-to-speech [25] and machine-translation [17].

During the forward step of convolution dilation, also referred to as convolution with "holes", we slide its filters on non-contiguous regions of the input activation map, with a gap in the kernels with size equal to the dilation factor d, as shown in figure 4.6. More precisely, using the same notations than eq. 3.2, 3D temporal dilated convolution is computed as

$$v_{ij}^{xyz} = \phi\left(b_{ij} + \sum_{m=0}^{F_{i-1}-1} \sum_{p=0}^{H_i-1} \sum_{q=0}^{W_i-1} \sum_{r=0}^{D_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+d\times r)}\right) \tag{4.10}$$

where the temporal dilation computes the output value in z, by applying the filters on

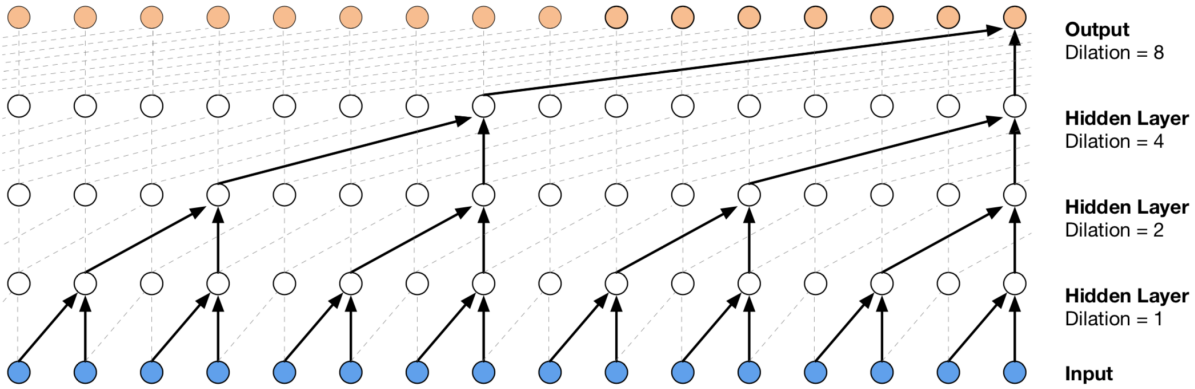


Figure 4.6: Visualization of dilated convolutional layer each with various dilation rate ($d = 1, 2, 4, 8$). Taken from [25].

positions $(z + d \times r)$ of the input feature map, $\forall r$.

During training, we choose to apply dilation on every $3 \times 3 \times 3$ convolutional layer of I3D, with a dilation factor $d \in \{1, 2\}$ selected randomly[1]. During evaluation, the same layers compute an activation map for both $d = 1$ and $d = 2$, which are then averaged in one output feature map.

[1]When $d = 1$ no dilation is applied.

Chapter 5

Application in Human-Computer Interaction

With the diffusion of computers, the number and the type of users interacting with them start growing considerably. In the early stage of computers, technicians were almost the only users, while nowaday personal computers are part of the daily life of many people. This was possible thanks to the advancement of human-computer interaction, which focuses on the interfaces between people and computers. Following a chronological order, the usability of current computers was mostly affected by monitors and keyboards, the mouse together with a Graphical User Interface (GUI), the touch screen and the more recent voice-control, which allows a vocal interaction with the machine.

In this section we apply the previously mentioned neural networks in the direction of a novel human-computer interaction. We leverage on the great results obtained from video-based CNNs to build a hand gesture recognition system, the first step for a new interaction through cameras. In particular, our intention is to build the most accurate model that correctly classifies a small and simple set of hand gestures, which can be assigned to a specific action (i.e. sliding a text file with simple movement of the hand in front of the camera). The work of this thesis may serve as a preliminary work for such human-computer video interaction.

20BN-JESTER-DATASET classes	
Name	**Number of videos**
Doing other things	12416
Drumming Fingers	5444
No gesture	5344
Pulling Hand In	5379
Pulling Two Fingers In	5315
Pushing Hand Away	5434
Pushing Two Fingers Away	5358
Rolling Hand Backward	5031
Rolling Hand Forward	5165
Shaking Hand	5314
Sliding Two Fingers Down	5410
Sliding Two Fingers Left	5345
Sliding Two Fingers Right	5244
Sliding Two Fingers Up	5262
Stop Sign	5413
Swiping Down	5303
Swiping Left	5160
Swiping Right	5066
Swiping Up	5240
Thumb Down	5460
Thumb Up	5457
Turning Hand Clockwise	3980
Turning Hand Counterclockwise	4181
Zooming In With Full Hand	5307
Zooming In With Two Fingers	5355
Zooming Out With Full Hand	5330
Zooming Out With Two Fingers	5379

Table 5.1: Labels and corresponding number of videos of the Jester dataset.

5.1 Hand Gesture Recognition

To develop this application, the key components are the I3D network, with Kinetics pre-trained weights, and its modified versions demonstrated in earlier chapters. Moreover, we choose to fine-tune I3D on the Jester hand gesture dataset [37], which is composed by a training set (118562 videos), a validation set (14787 videos) and a test set (14743 videos w/o labels). Jester is composed by 27 classes, listed in table 5.1 with the respective number of videos.

The datasets are saved in the *TFRecord* file format, the binary storage format from TensorFlow. This can improve significantly the import pipeline, since binary data occupy less space on the disk, take less time to copy and can be read more efficiently from the disk.

From each video we save 40 frames[1], which we resize with an aspect ratio $height \times width$ equal to 100×140.

5.1.1 Data Augmentation

The size of a dataset plays an important role for the neural networks in learning generic features. During training, when the size of the dataset is too small and no data augmentation is applied, a model tends to overfit, which means that it fits the training set well, while it poorly adapts to new datasets. An easy and effective way to overcome this problem is to apply small alterations on the training set; in this way the model is trained on a more diversified amount of data. During training, we use different types of data augmentation. However, the decision whether to apply these transformations or not is taken randomly.

At first, we employ **per-frame rotation**, which is implemented by randomly rotating the frames t of a video, each with a different angle $\theta(t) \in (-15, 15)^\circ$. In this way we simulate camera movement. The rotation is done by spinning the image around its center, mapping each pixel position $(x(t), y(t))$ into $(x'(t), y'(t))$ with the following transformation[2]

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{5.1}$$

Afterwards, we adopt **random cropping**, which consists in cropping a random region $S \times S$ from the original or rotated frame $H \times W$, with $S \leq \min\{H, W\}$. We choose randomly the location of the cropped region, differently for each frame, again to mimic camera motion.

We then rescale the pixels' values of the cropped regions in the range $p(x, y, t)_i \in [-1, 1]$, for each channel $i = R, G, B$ and every frame t.

Whereupon, we randomly apply **color jittering** uniformly on the whole video, which consists in a random combination of the following 4 types of color variation:

- *random brightness*: adjusts the brightness of the frames by a random factor b, for

[1]Eventually, when a video has less than 40 frames, we replicate the first frames sequentially. This appears as a slowed down version of the original video.

[2]We suppose that the image is 2D, for each pixel of an RGB image $(x, y)_i$ the transformation is analogous, with $i = R, G, B$.

each channel $i = R, G, B$:

$$p'(x, y, t)_i = p(x, y, t)_i + b, \qquad \forall x, y, t; \tag{5.2}$$

- *random saturation*: converts an RGB image into HSV (Hue Saturation Value), multiplies the saturation channel by a random saturation factor s and transforms the HSV back to the RGB format. For each voxel location (x,y,t), the convertion to HSV is computed as

$$p_H = \begin{cases} 0°, & \text{if } \Delta = 0 \\ 60° \cdot \left(\frac{p_G - p_B}{\Delta} \mod 6\right), & \text{if } C_{max} = p_R \\ 60° \cdot \left(\frac{p_B - p_R}{\Delta} + 4\right), & \text{if } C_{max} = p_G \\ 60° \cdot \left(\frac{p_R - p_G}{\Delta} + 2\right), & \text{if } C_{max} = p_B \end{cases}$$

$$p_S = \begin{cases} 0, & \text{if } C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & \text{if } C_{max} \neq 0 \end{cases}$$

$$p_V = C_{max},$$

given $C_{max}(x, y, t) = \max_i\{p(x, y, t)_i\}$, $C_{min}(x, y, t) = \min_i\{p(x, y, t)_i\}$ and $\Delta = C_{max} - C_{min}$. The saturation transformation is computed for each pixel as

$$p'(x, y, t)_S = p(x, y, t)_S \cdot s, \qquad \forall x, y, t; \tag{5.3}$$

- *random hue*: analogously to the provious variation, it multiplies the hue channel by a random factor h:

$$p'(x, y, t)_H = p(x, y, t)_H \cdot h, \qquad \forall x, y, t; \tag{5.4}$$

- *random contrast*: adjusts the contrast, independently for each channel $i = R, G, B$, by a random factor c:

$$p'(x,y,t)_i = c \cdot (p(x,y,t)_i - M_i) + M_i, \qquad \forall x, y, t, \tag{5.5}$$

where M_i is the mean value for each channel.

Image jittering is very helpful to build robust features, because it teaches the model to equally classify the same instances under different lights and color variations.

Finally, we apply **horizontal image flipping**, identically to the whole set of frames of a video. We need to pay particular attention when the flipping is applied on videos representing an unidirectional action, i.e. *Swiping Left*. When this occurs, the label of the video needs to be inverted, in our case from *left* to *right* and vice versa.

5.1.2 Training I3D

Sequential Training We fine-tune the I3D model on Jester dataset using a *GeForce GTX 1080* GPU, which computes both the training and the evaluation processes in a sequential fashion. We exploit the pre-trained weights on the Kinetics dataset as a valuable inizialization, except for the *logits block*, whose weights are randomly initialized. The logits block is composed by the average pooling layer, with size $2 \times 4 \times 4$, and the final $1 \times 1 \times 1$ convolutional layer, whose depth corresponds to the number of classes of the dataset.

We train the networks using stochastic gradient descent (SGD) with momentum. This method computes the current updates v_{t+1} with a sum between the updates of the previous iteration v_t, weighted by the momentum parameter $\mu = 0.9$, and the partial derivatives ∇E, weighted by the learning rate α. The whole backward step is performed as follow

$$\begin{aligned} v_{t+1} &= \mu \cdot v_t + \alpha \cdot \nabla E, \\ W_{t+1} &= W_t - v_{t+1}. \end{aligned} \tag{5.6}$$

The loss function E used to train our model is defined as **softmax cross entropy**, which more precisely computes the cross entropy between the true probability distribution p and the estimated probability distribution q:

$$E(p,q) = -\sum_{k=1}^{K} p_k \log(q_k), \tag{5.7}$$

with K as the total number of classes. q_k is computed with the softmax function on the output vector of the network $z \in \mathbb{R}^K$:

$$q_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \tag{5.8}$$

and p is a one-hot vector, with probability mass concentrated on the correct class l:

$$p_k = \begin{cases} 0, & \text{if } k \neq l \\ 1, & \text{if } k = l \end{cases}. \tag{5.9}$$

Finally, we can rewrite the loss of a given input x_i, with ground-truth l, as

$$E(x_i) = -\log\left(\frac{e^{z(x_i)_l}}{\sum_{j=1}^{K} e^{z(x_i)_j}}\right) = -z(x_i)_l + \log\left(\sum_{j=1}^{K} e^{z(x_i)_j}\right). \tag{5.10}$$

Due to a memory constraint we set the batch size equal to 16. For the first 140k steps we use a learning rates $\alpha = 1 \times 10^{-3}$, until step 160k $\alpha = 1 \times 10^{-4}$ and until step 170k $\alpha = 1 \times 10^{-5}$, for a total of 23 epochs.

With this settings we are able to reach a top 1 accuracy equal to 92.48% on Jester validation dataset and the top 5 accuracy[3] is higher than 99%. Since it leaves no room for improvement for the following models, we choose to only adopt the top 1 accuracy as benchmark.

Distributed Training Given the large size of the input videos, it would be appropriate to train the network in a distributed fashion, on 2 or more GPUs. In particular, we tested for few steps a parallel configuration on 2 GPUs, where 3 workers compute the gradients of the parameters on the same batch, each using half of a GPU. The gradients are then sent to the parameter saver, working on CPUs, whose task is to update the parameters with an optimization algorithm. Half of a GPU is left to compute the evaluation on the validation set, in order to monitor the evolution of the accuracy during training. In this way, we better exploit the whole GPUs capacity and we speed up the training/evaluating procedure. We visualize such configuration in figure 5.1.

[3]Top 5 accuracy measures the percentage of samples with ground-truth contained in the 5 highest probability predictions of the model.

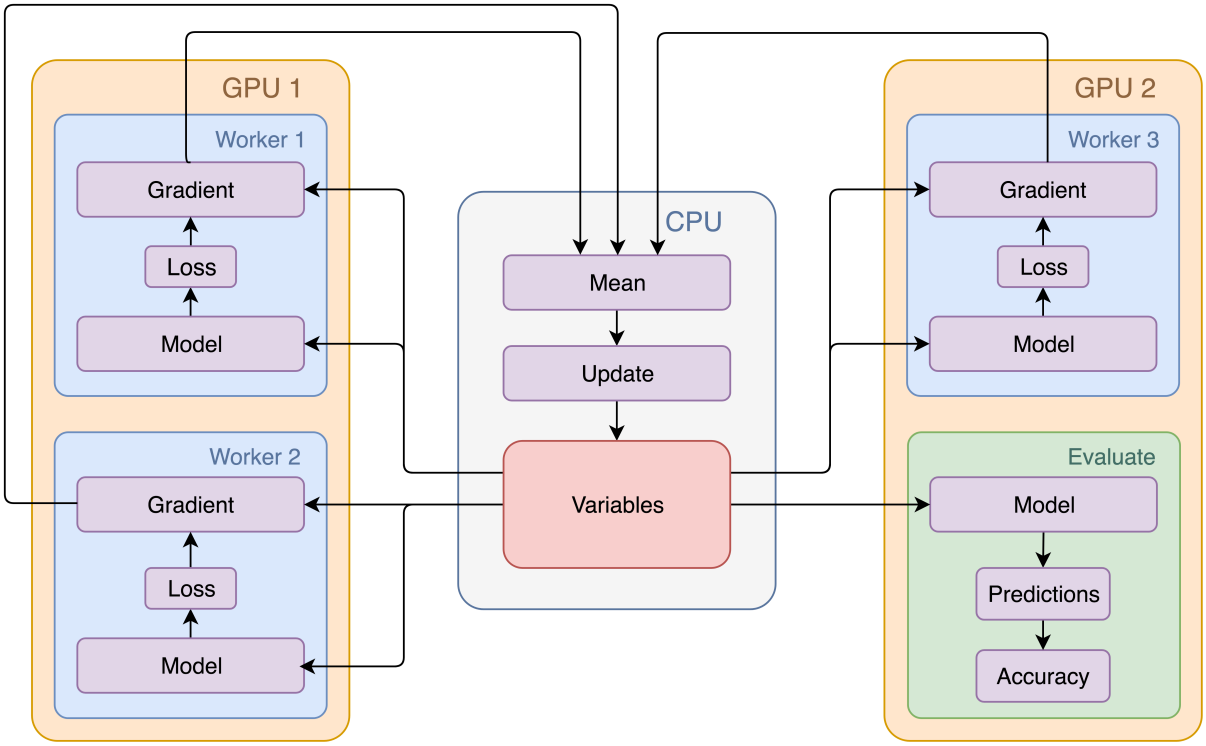


Figure 5.1: Distributed synchronous training and evaluation on 2 GPUs. Repartition on 2 GPUs of 3 training workers and 1 evaluation to compute the accuracy of the validation set every time a new checkpoint is saved.

The parameters update is computed with a *synchronous* approach[4]: at each iteration, the gradients, computed in parallel by all the workers, are averaged by the parameter saver before the updates. Again, we train the networks using stochastic gradient descent with momentum.

The whole **Synchronous Distributed SGD+Momentum** algorithm is summarized as pseudo code in algorithm 1, with N_w as number of workers and a maximum of T iterations.

We do not further investigate on this approach for time constraints. However, it would be interesting in future research to analyze if distributed training obtains competitive results to sequantial training.

[4]This is implemented using TensorFlow pre-built classes *SyncReplicasOptimizer* and *MonitoredSession*.

Algorithm 1 Synchronous Distributed SGD+Momentum

while $t < T$ **do**
  select a batch of input labels and videos (l_i, x_i)
  for each worker j **do**
    compute the gradient of the loss: $\nabla E_j(l_i, F(x_i, W_t))$
  end for
  sum the gradients of all workers: $\Delta W_t = \sum_{j=1}^{N_w} \nabla E_j$
  compute SGD+Momentum update: $v_{t+1} = \mu \cdot v_t + \alpha \cdot \frac{\Delta W_t}{N_w}$
  apply update: $W_{t+1} = W_t - v_{t+1}$
  $t = t + 1$
end while

5.1.3 Training NL-I3D

We train the NL-I3D network on the Jester dataset, initializing the parameters with the I3D weights pre-trained on the same dataset[5], except for the weights of the non-local blocks, which are zero initialized. In this way we do not modify the initial behaviour of the network.

The training step is performed in a sequential fashion, again with the batch size equal to 16; we schedule the learning rate with an initial value $\alpha = 1 \times 10^{-3}$ and we reduce it by a factor of 10^{-1} after 140k steps and after 160k steps, for a total of 170k iterations (23 epochs). With this setting plus the addition of the two non-local blocks we slightly improve the I3D model, achieving a top 1 accuracy equal to 92.79%. This improvement is given by the increment of number of learnable parameters and by the structure of the non-local blocks, that captures long-range dependencies of non-adjacent pixels and frames, as we previously described. The number of parameters of NL-I3D is approximately 14.2M, an increment of almost 2M parameters compared to I3D, mostly situated in the second non-local block, the nearest to the output of the network.

5.1.4 Training G-I3D

We train G-I3D with the same configurations of NL-I3D, to have a more valid comparison of the two solutions.

The top 1 accuracy reached by this model is 92.85%, which slightly improves the NL-I3D. More notable is the fact that this network introduces only an extra 205824 of learnable parameters, for a total of 12.5M. For this reason we consider the spatiotemporal

[5]We use the I3D weights after 40k iterations of training on the Jester dataset, which reach an accuracy around 86%.

feature gating more suitable for the improvement of the I3D architecture. This result could be induced by the architecture in use, given that the gated solution was also optimized on a network with Inception-v1 backbone, while the non-local block was built on top of a ResNet 3D architecture. As counterproof, we should test these two solutions also on a ResNet 3D network. We leave this research as an open topic and we do not further investigate in this thesis.

5.1.5 Training Dilate-I3D

This version of I3D introduces random temporal dilation on the $3 \times 3 \times 3$ convolutional layers, as previously described. This enlarges the temporal receptive field of the layers, in this way the model is able to capture long range temporal features from the early layers. More specifically, we call **Full-Dilate-I3D** the I3D model with temporal dilation applied on every $3 \times 3 \times 3$ convolutional layer during evaluation.

We adopt again the same training configurations as NL-I3D and G-I3D, with equivalent batch size and learning rate schedule. With this settings the Full-Dilate-I3D model reaches an accuracy equal to 91.8% on the validation set, inferior to the standard I3D model.

We analyze which part of the architecture benefits the most from temporal dilation. To investigate this we evaluate such model applying dilation only on a partial set of the convolutional layers, at different level of abstraction. At first, we only dilate the N closest convolutional layers and inception modules to the input, and we reference to this model as **Top-N-Dilate-I3D**. Afterwards, we evaluate the model applying dilation only on the bottom layers before the output, which we call **Bottom-N-Dilate-I3D**. Finally, we consider the limit case with $N = 0$ where no dilation is applied during evaluation; this version is referenced as **No-Dilate-I3D**.

From table 5.2, we notice that the best result is not obtained when dilation is applied on the whole architecture, but only on the top layers. In particular, the highest accuracy is achieved by the **Top-5-Dilate-I3D**. This peculiar behaviour could be explained by the fact that early layers in the original I3D architecture have small temporal receptive fields, hence they can benefits more from dilation compared to deeper layers, whose receptive fields already cover the entire length of the full input videos.

As a final consideration, Dilate-I3D does not enlarge the number of learnable parameters; however, during evaluation, it computes twice each convolutional operation with

Top-N-Dilate-I3D		**Bottom-N-Dilate-I3D**	
N	**accuracy**	**N**	**accuracy**
1	91.72	1	91.50
2	91.74	2	91.53
3	91.74	3	91.61
4	91.88	4	91.63
5	**92.01**	5	91.53
6	91.97	6	91.76
7	91.99	7	91.77
8	91.78	8	91.65
9	91.82	9	91.62
Full-Dilate-I3D		**No-Dilate-I3D**	
N	**accuracy**	**N**	**accuracy**
10	91.80	0	91.56

Table 5.2: Top 1 accuracies of the D-I3D architecture applying dilation on different sets of the convolutional layers.

dilation. Because of these reasons, the speed of the training process does not decrease, while during evaluation the number of operations increases with the growth of N.

5.1.6 Modified-I3D

We conclude our research implementing our best network architecture, the **Modified-I3D**, which simply combines the G-I3D with the Top-5-D-I3D. More precisely the Modified-I3D places the spatiotemporal feature gating at the same positions of G-I3D and employs random dilation in the first $3 \times 3 \times 3$ convolutional layer and the initial four inception modules.

We start training the Modified-I3D from the final checkpoint of D-I3D computed on the Jester dataset and we set the same batch size and learning rate schedule as before, for a total of 170k steps. From figure 5.2, we observe that the training benefits from the reduction of the learning rate, given that in both cases the curve shows an improvement of accuracy. As we expected, the Modified-I3D reaches the highest top 1 accuracy with 93.36% of correct predictions on the validation set. This result, compared to the G-I3D top 1 accuracy, shows that using temporal dilation on a proper set of convolutional layers increases the model accuracy.

The Modified-I3D is the only architecture evaluated on the Jester test set, achiving a top 1 accuracy of 93.41%.

Finally, we want to test if random dilation improves the model predictions, as a sort

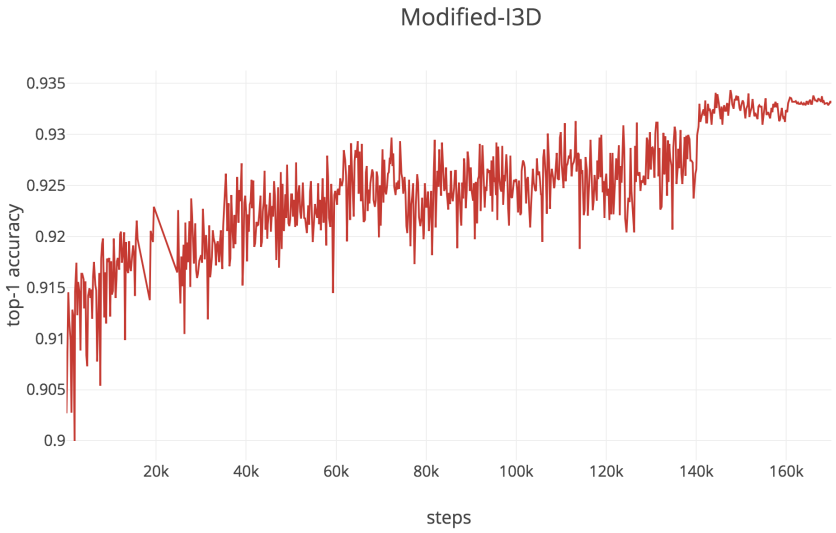


Figure 5.2: Top 1 accuracy of the Modified-I3D model computed on the Jester validation set at different training steps.

of data augmentation. We investigate this by evaluating the Modified-I3D without the use of temporal dilation on the Jester validation set and we obtain an accuracy equal to 93.32%, higher than each other network. This result shows that temporal dilation can help the networks to capture more generic temporal independent features, e.g. it helps the network to predict with higher confidence the same action performed at different speed. To further analyze this study, it would be appropriate to employ higher dilation factors during training. However, we leave this as an open topic for future research.

In figure 5.3 is shown the accuracy for each class obtained from the different networks of this research. From such histogram we notice that, for each architecture, the classes *Turning Hand Clockwise* and *Turning Hand Counterclockwise* always obtain the lowest accuracy, while for the remaining classes the networks often reach an accuracy higher than 90%. This could be caused by the imbalance in the Jester dataset, considering that both classes have around 1000 videos less compared to the rest. This should be verified by training the models on a balanced subset of the Jester dataset, i.e. reducing the number of video clips to 4000 for each class. We leave this as an open topic for future research.

The Modified-I3D architecture does not always have the highest per class accuracy, however it is the one reaching the highest accuracy for *Turning Hand Clockwise* and *Turning Hand Counterclockwise*, 69.61% and 76.44% respectively, and the only one having top 1 accuracy higher than 90% for each other class.

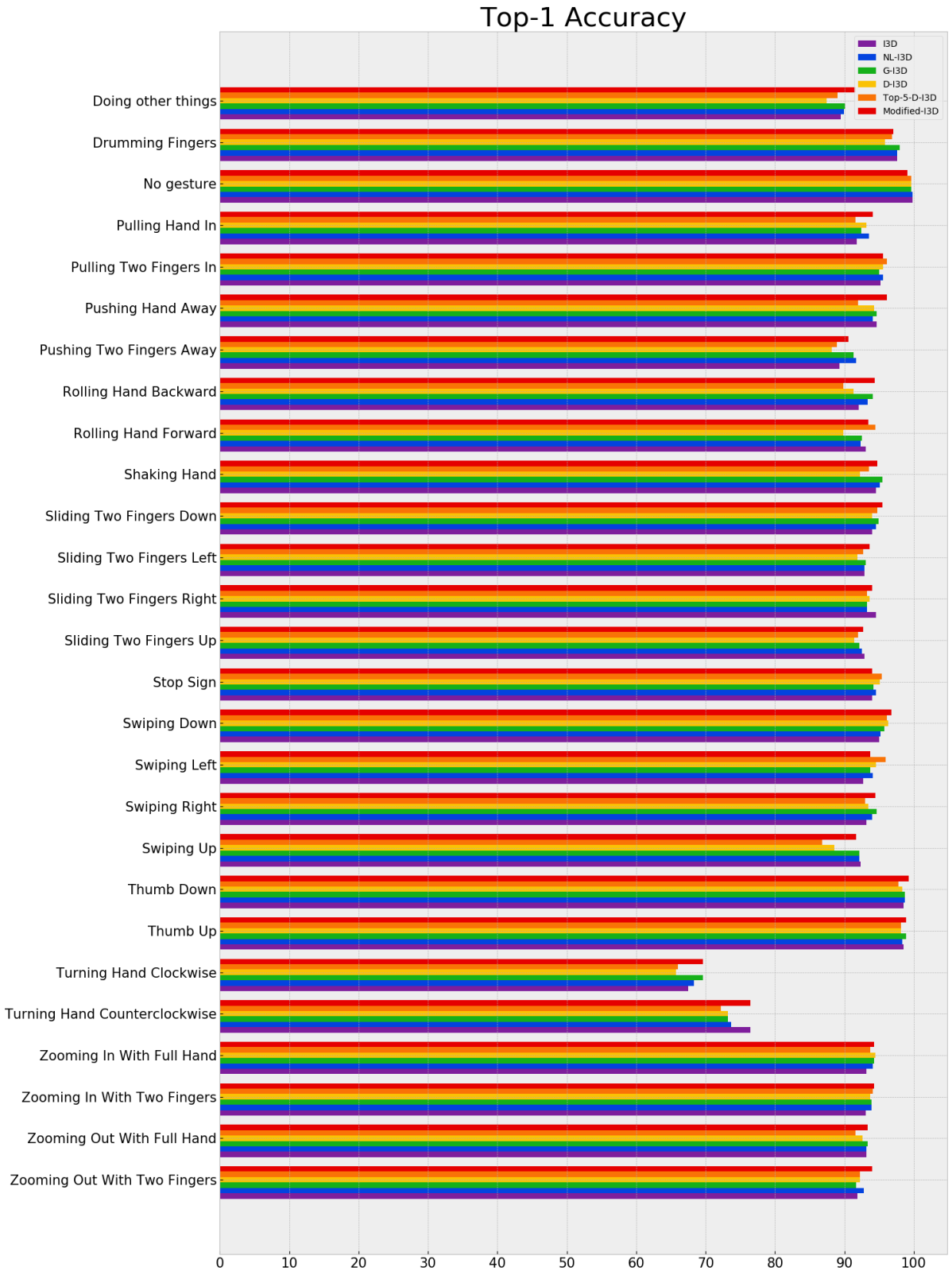


Figure 5.3: Per class top 1 accuracy reached by the different models.

Model	Top1 accuracy
I3D	92.48%
NL-I3D	92.79%
G-I3D	92.85%
D-I3D	91.80%
Top5-D-I3D	92.01%
Modified-I3D	**93.36%**
Modified-I3D (no dilation)	93.32%

Table 5.3: Top 1 accuracy of the different architectures investigated.

5.2 Real Time

We now describe the chosen structure of the real time application. This is divided in the following steps: first it captures the video stream with a mobile device, such as phones, laptops or fixed cameras; then the video is sent to a *GeForce GTX 1080* GPU, which computes the video preprocessing, the gesture recognition algorithm and the video postprocessing; finally, the live video stream and the predicted class are displayed in real time.

If we want the most accurate gesture recognition system, it is evident from the previous section that the Modified-I3D is the best solution. Alternatively, we could choose the Modified-I3D with no dilation for a speed improvement.

The width of each frame is resized to 120 pixels and it is kept the same aspect ratio of the original frame. Afterwards, we crop the central 100×100 portion of the image. We then save the 40 most recent preprocessed frames into a video clip, which corresponds to $1.3sec$ of the video stream, in order to keep the same number of frames adopted during the training and evaluation procedures. The video clip is repeatedly updated every time a new frame is captured. However, we design the application to compute a prediction every 20 frames ($0.65sec$), hence every time the video clips replaces the 50% of its frames. This configuration allows us to have nearly continuous predictions, since we estimate that each gesture is performed in around $1sec$, and without drawbacks in computations. This workflow process around $30fps$.

Moreover, we consider a prediction as valid only when the probabilty of the outputed label is higher than 50%, in order to build a more robust sytem. This solution avoids to have predictions with high uncertainty, which typically characterizes the video clips containing frames of two separate gestures performed one immediately after the other.

Figure 5.4: Examples of the real time interface while performing three different gestures correctly classified.

We show in figure 5.4 the visual interface of this real time application, where the white box represents the central cropped area, the green text shows the label predicted with high confindence and the blue text shows the probability and the class name of every predictions, even when the probability is lower than 50%.

5.3 Visualization

In this section we want to show which input regions mostly affect the prediction of a network. Various techniques have been developed to interpret and visualize what a neural network "sees" when it recognizes an object in an image, e.g. what are the features that a network recognizes to discriminate a cat from a dog. Among them we choose to adapt the Gradient-weighted Class Activation Mapping (Grad-CAM) [29] approach to videos, in order to obtain a class-discriminative heat-map describing which area of the input video activates a specific layer. Of particular interest are the final layers, which severly affect the prediction of the output label.

Using similar notation of [29], we compute the class-discriminative localization map of a class c, $L^c_{Grad-CAM} \in \mathbb{R}^{U \times V \times T}$, where U and V are the spatial dimensions and T the temporal dimension of the feature maps in consideration, $A^k \in \mathbb{R}^{U \times V \times T}$. More precisely, $L^c_{Grad-CAM}$ is computed as a weighted combination of the K activation maps of the selected layer, followed by a ReLU:

$$L^c_{Grad-CAM} = ReLU\left(\sum_{k=1}^{K} \alpha^c_k A^k\right). \tag{5.11}$$

The weights α^c_k represent how much the different activation maps A^k influence y^c, which is the score of the output layer for the target class c before the softmax function. The

weights are obtained by computing the gradient of y^c with respect to the feature map in the following way

$$\alpha_k^c = \frac{1}{Z} \sum_{u=1}^{U} \sum_{v=1}^{V} \sum_{t=1}^{T} \frac{\partial y^c}{\partial A_{uvt}^k}. \tag{5.12}$$

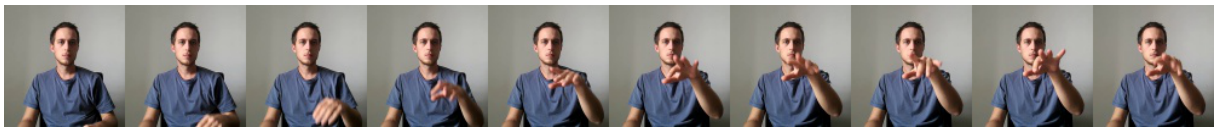
In our case, we compute the normalization factor as

$$Z = \sqrt{\frac{1}{UVT} \sum_{u=1}^{U} \sum_{v=1}^{V} \sum_{t=1}^{T} \left(\frac{\partial y^c}{\partial A_{uvt}^k} \right)^2 + p} \tag{5.13}$$

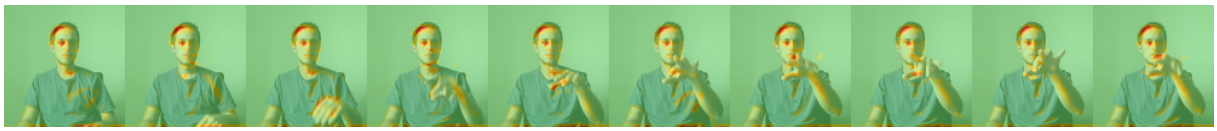
where $p = 1 \times 10^{-4}$ is necessary to avoid zero gradients.

Once $L_{Grad-CAM}^c$ is computed, in order to have a more clear visualization, we resize it to match the dimensions of the input video, in our case with $height \times width \times frames$ equal to $100 \times 100 \times 40$. We apply this technique to the output of every 3D convolutional layer or inception module of the Modified-I3D architecture and we visualize in figure 5.5 the resulting heatmaps. It shows that the early 3D convolutional layers activate mostly on small details of the video frames, i.e. eyes, nose, mouth, hair and more. This is motivated by the fact that the output activation maps of early layers have large spatial and temporal dimensions, so the localization map is more detailed. Moreover, the receptive field of these layers is small, hence they do not have a global understanding of the video, but they focus on small patterns. By going deeper in the architecture we notice how the layers activate when the hand movement is spotted, with the exception of the final inception module, which activates on a larger slice of the input video. This behaviour is noticed also on other examples and it is justified by the very small dimensions of the output activation map of the final layer, in our case $4 \times 4 \times 5$.

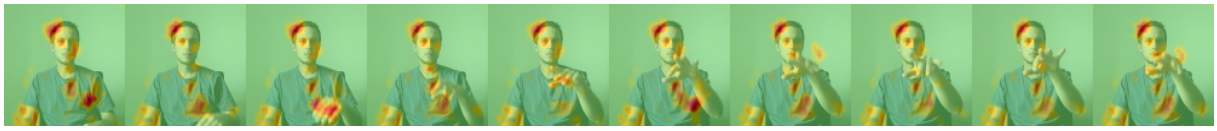
Not every localization map has an easy interpretation of why each layer focuses its attention on a specific area. We observe that, for most of the examined samples, the inception modules 6, 7 and 8 are those which more clearly focus on the hands' movements. We observe that these layers activate mostly on those frames where the specific action takes place, showing that the model is robust. We show in figure 5.6 other good examples of localization maps computed on these inception modules.



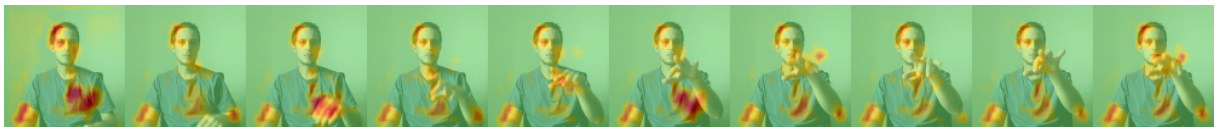
(a) Original frames.



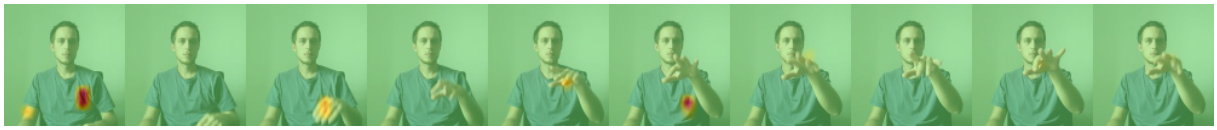
(b) $7 \times 7 \times 7$ conv. 3D layer.



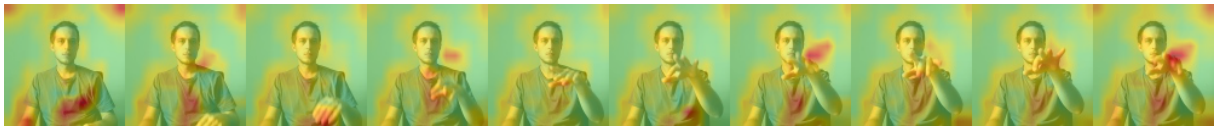
(c) $1 \times 1 \times 1$ conv. 3D layer.



(d) $3 \times 3 \times 3$ conv. 3D layer.

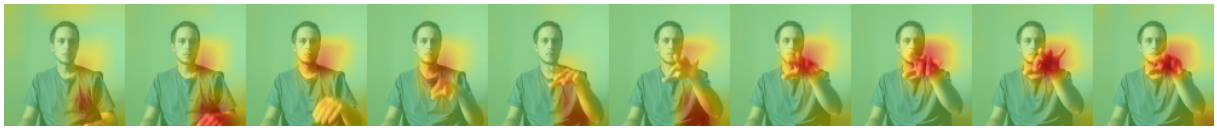


(e) Inception module 1.

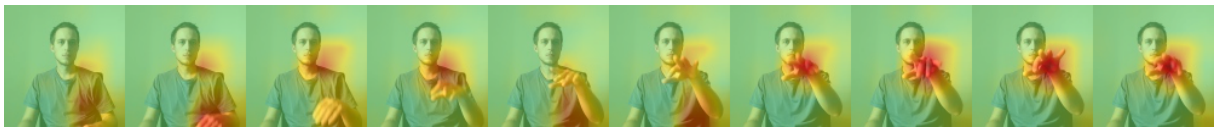


(f) Inception module 2.

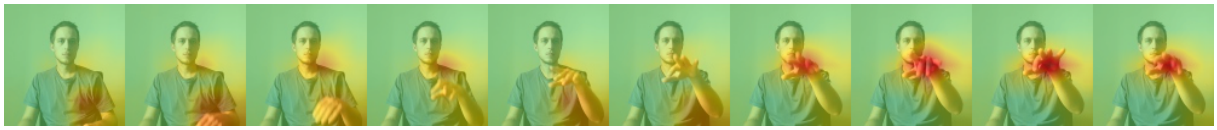
(g) Inception module 3.



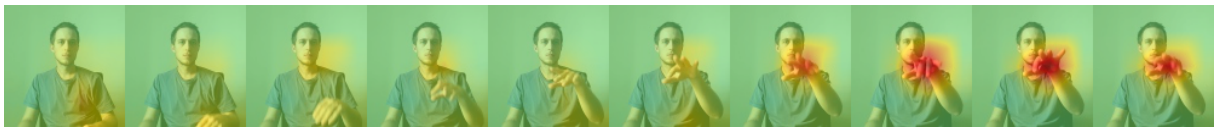
(h) Inception module 4.



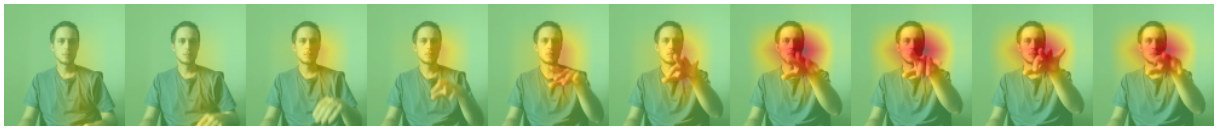
(i) Inception module 5.



(j) Inception module 6.



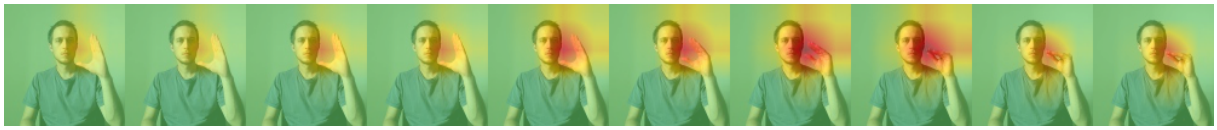
(k) Inception module 7.



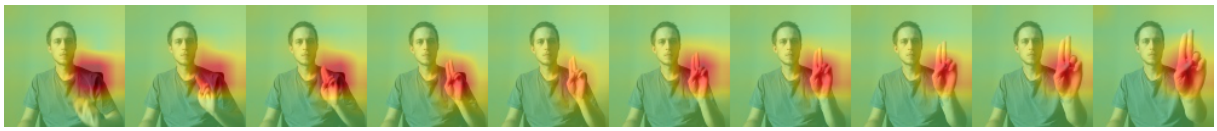
(l) Inception module 8.

(m) Inception module 9.

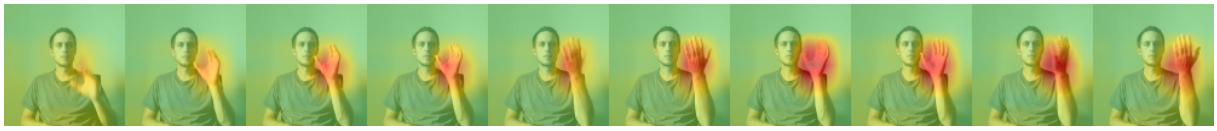
Figure 5.5: Grad-CAM visualizations for different layers of the Modified-I3D architecture. Note that red regions corresponds to high scores for the class *Drumming Fingers*.



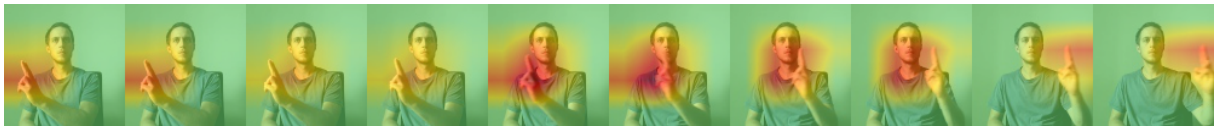
(a) Zooming Out With Full Hand.



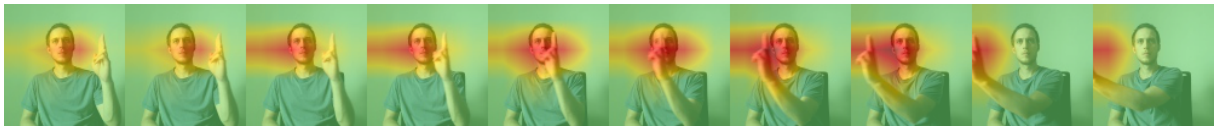
(b) Pushing Two Fingers Away.



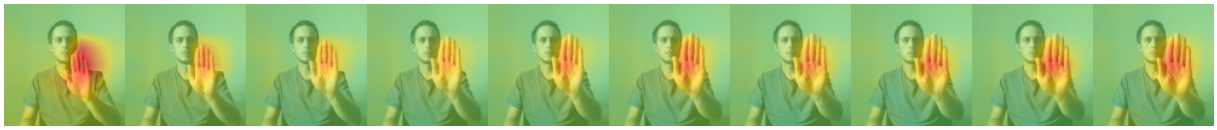
(c) Shaking Hand.



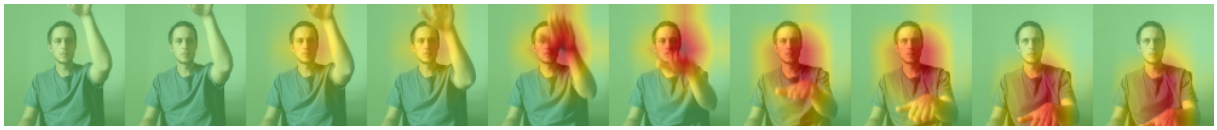
(d) Sliding Two Fingers Right.



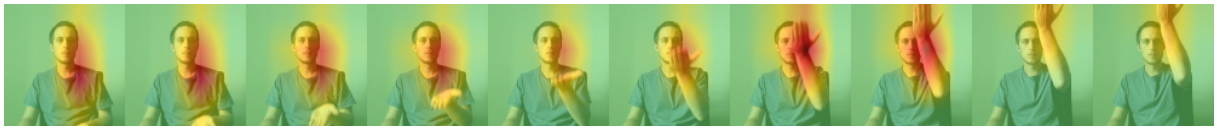
(e) ISliding Two Fingers Left.



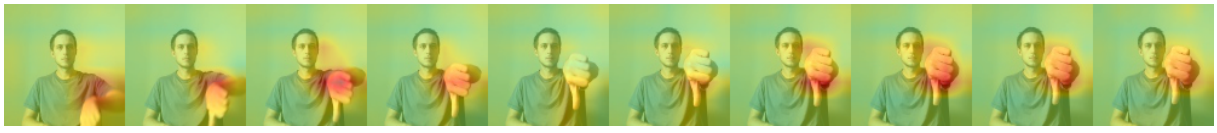
(f) Stop Sign.



(g) Swiping Down.



(h) Swiping Up.



(i) Thumb Down.

Figure 5.6: Samples of activation heat-maps computed with Grad-CAM on some video clips. We notice how the Modified-I3D activates where the gestures take place.

Chapter 6

Conclusion

The work of this thesis lays the foundations for future research in video action recognition and more generally in video understanding. The results of this thesis can help researchers to build novel 3D-CNNs exploiting non-local blocks, spatiotemporal feature gates or temporal dilation, in order to enlarge the capacity of the models to capture spatio-temporal features. We observe that both the non-local blocks and the spatiotemporal feature gates contribute to increase the model accuracy, while temporal dilation seems to perform better when it is located in the early convolutional layers.

In this work we build our best solution from the I3D architecture introducing a combination of spatiotemporal feature gates and temporal dilation, that we name Modified-I3D. However, we demonstrate that a given deep neural network can benefit from these techniques with a small increment of parameters, but we do not further investigate if a better configuration exists.

Finally, the hand gesture recognition system was built to test the efficiency of the Modified-I3D architecture in real-time, showing great results. This is the first and most challenging step for the implementation of a gesture-based human-computer interface, which we believe it will be soon integrated in future generations of computers.

6.1 Future Research

As previously mentioned, the next step of this research consists in finding the best configuration for the Modified-I3D. In particular, by testing the spatiotemporal feature gates in other locations and adopting dilation with higher dilation factors.

Furthermore, we would like to expand this research on new topics. For instance, implementing a sign language recognition system, in order to facilitate the comunication between deaf and non-deaf people. The goal is again to build a hand gesture classifier, where the classes depend on the sign languages. For example, the American Sign Language (ASL) is composed by 6000 hand gestures for common words, plus the fingerspelling of loan words and proper names. In this case it is necessary to have a proper ASL dataset with a good amount of labeled data. Afterwards, we can fine-tune our models for the new task, following the steps of this research.

Bibliography

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. In *arXiv preprint arXiv:1609.08675*, 2016.

[2] J. L. Barron, D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. Performance of optical flow techniques. In *CVPR*, 1992.

[3] F. Caba Heilbron, V. Escorcia, B. Ghanem, and J. Carlos Niebles. ActivityNet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] J. Carreira and A. Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *arXiv preprint arXiv:1705.07750v2*, 2017.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. In *ICML*, 2015.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[7] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *CVPR*, 2017.

[8] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016.

[9] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Temporal residual networks for dynamic scene recognition. In *CVPR*, 2017.

[10] M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. In *Trends in Neurosciences*, 1992.

[11] R. Goyal, S. Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, et al. The "something something" video database for learning and evaluating visual common sense. In *arXiv preprint arXiv:1706.04261*, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural Computation*, 1997.

[14] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 1981.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *arXiv preprint arXiv:1502.03167*, 2015.

[16] S. Ji, W. Xu, M. Yang and K. Yu. 3D Convolutional Neural Networks for Human Action Recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[17] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. In *arXiv preprint arXiv:1610.10099*, 2016.

[18] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[19] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman. The kinetics human action video dataset. *arXiv preprint*, 2017.

[20] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[21] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A large video database for human motion recognition. In *ICCV*, 2011.

[22] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[23] M. Marszałek, I. Laptev, and C. Schmid. Actions in context. In *CVPR*, 2009.

[24] M. Monfort, B. Zhou, S. A. Bargal, A. Andonian, T. Yan, K. Ramakrishnan, L. Brown, Q. Fan, D. Gutfruend, C. Vondrick, et al. Moments in time dataset: one million videos for event understanding. In *arXiv preprint arXiv:1801.03150*, 2018.

[25] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. In *arXiv preprint arXiv:1609.03499*, 2016.

[26] A. Parker. In the Blink of an Eye: How Vision Sparked the Big Bang of Evolution. Cambridge, MA: Perseus Pub. 2003.

[27] Z. Qiu, T. Yao, and T. Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *ICCV*, 2017.

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[29]] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *arXiv 1610.02391v3*, 2017.

[30] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[32] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *arXiv preprint arXiv:1212.0402*, 2012

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[34] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[35] G. Huang, Z. Liu, K.Q. Weinberger. Densely Connected Convolutional Networks. In *CoRR*, 2016.

[36] Twentybn. Gesture recognition using end-to-end learning from a large video database. Available: https://medium.com/twentybn/gesture-recognition-using-end-to-end-learning-from-a-large-video-database-2ecbfb4659ff, 2017.

[37] Twentybn. Jester dataset: a hand gesture dataset. Available: https://20bn.com/datasets/jester, 2017.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[39] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool. Temporal Segment Networks for Action Recognition in Videos. In *arXiv preprint arXiv:1705.02953v1*, 2017.

[40] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *CVPR*, 2018.

[41] S. Xie, C. Sun, J. Huang, Z. Tu, K. Murphy. Rethinking Spatiotemporal Feature Learning For Video Understanding. In *arXiv preprint arXiv:1712.04851*, 2017.

[42] Y. Zhao, Y. Xiong, L. Wang, Z. Wu, D. Lin, and X. Tang. Temporal action detection with structured segment networks. In *ICCV*, 2017.

[43] Y. Zhao, Y. Xiong, L. Wang, Z. Wu, D. Lin, and X. Tang. Temporal action detection with structured segment networks supplementary material. In *ICCV*, 2017.