POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master of Science Thesis

Analysis of Deep Learning Architectures for Human Action Recognition

Supervisors

Prof. Fabrizio Lamberti

Dr.ssa Lia Morra

Candidate

Angelo Filannino

Academic Year 2018-2019

Ai miei genitori,
Franco e Concetta, e a
mio fratello Luca.

Acknowledgements

I would like to acknowledge my supervisor, Dott.ssa Lia Morra, for her help and for her patience and perseverance. She helped me a lot during the developing of this work and, since I was hardly ever available during the day, I have appreciated her availability in answering and communicating even during the evening. I learned a lot with this thesis work and it was also thanks to her advice.

I would like to say a big *thank you* to my family. You have always been there, ready to support me even beyond your possibilities. I will be forever grateful for this.

Then, I would like to *thank* my friends Michelangelo, Massimo, Kevin and Alessandro to have been great teammates in the university projects and during exams preparation. It has been easier as a group.

Thank you to Nunzia, *thank you* to have been always by my side. You decided to stay even if you knew that, between work and university, the time for us would have decreased. Thank you to have been there during my moments of happiness and my mental breakdowns.

And, in the end, a huge *thank you* to Turin. A wonderful city that has changed my life.

Abstract

Given the advances done in Image Classification and the great success this field is obtaining, **Video Classification** is the natural follow-up, although this task has more difficulties. This thesis describes the results of the implementations of some of the most successful techniques and approaches and then, the results of their fusion in order to achieve better performance and accuracy. Firstly, the **Single Stream Architecture** has been proposed with an LSTM implementation. Then, to consider also the temporal behavior, the **Two Streams Architecture** has been reproposed with two LSTM subnetworks. Moreover, in order to consider long-range temporal information as well, the **Temporal Segment Network** main concepts have been implemented. The implementations exploit the potentiality of **Python**, programming language, and **Keras**, a Deep Learning Framework that provides high-level neural networks APIs. The experiments exploit the **Optical Flow Estimation** techniques, namely Farneback estimation and LiteFlowNet warped estimation. They have been performed on **UCF101 Human Action Recognition dataset**. The results obtained are almost consistent with the state of the art techniques.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, Artificial Intelligence is one of the main focus of the scientific communities and, day after day, its applications are becoming deeply involved in our daily life. The term "Artificial Intelligence" is exceptionally wide in scope. According to Andrew Moore [3], Dean of the School of Computer Science at Carnegie Mellon University and Chief Scientist of Google Cloud AI, "Artificial intelligence is the science and engineering of making computers behave in ways that, until recently, we thought required human intelligence."

One branch of artificial intelligence is **Machine Learning**. Professor and Former Chair of the Machine Learning Department at Carnegie Mellon University, Tom M. Mitchell: "Machine learning is the study of computer algorithms that improve automatically through experience" [1]. The basic concept is to examine and compare huge quantity of data, collected in datasets, to make decision, classify events or objects and find common patterns. ML is already involved in common system such as recommendations engine, web search, face recognition or speech-to-text software.

It is possible to divide Machine Learning in two macro areas: Supervised Learning and Unsupervised Learning. Classification is part of the first group of algorithms and methodologies and it is defined as the process of predicting the class (output) for a specific series of features (input). The general approach is described in the following steps:

Data Collection In this phase, a large dataset of 'input' is collected and labelled. It is possible to use an existent dataset and this is what has be done during this experiments. Usually the dataset is divided in training, validation and testing set.

Training In this phase, the training and validation subset inputs are 'seen' by the model which, according to algorithms, starts learning how to classify them. The model's goal is to minimize an objective function that measures the error value between the correct classification and the predicted value. During this minimization, hundreds of million of parameters, generally called 'weights', are updated to obtain a trained model that will perform the prediction.

Testing and Application Finally, the trained model can receive the test subset of the dataset and some metrics, like accuracy, can be evaluated. If the metrics satisfy some thresholds, the model has achieved the capacity to generalize previously unseen input and it is able to produce mostly correct answer to inputs that it has never seen before. It is ready to be used for a real application.

This approach is the standard *de-facto* for the most research or industrial usage of Machine Learning for classification. **Human Action Recognition** is a video classification process; starting from a video that contains one person, or more, doing a specific action, the algorithm should recognize the human action (or multiple) among a set of possible actions (class labels). This thesis is about a ML model that allows to recognize an human action among a subset of actions.

1.1 Purpose

The purpose of the project described in this thesis is to propose approaches that include the usage of a Recurrent Neural Network (RNN) for the video action recognition task. RNNs are a powerful type of neural networks which include a *feedback loop*, where, basically, the output of step N is affected by the output of step $N-1$. During my experiments, I have performed three different experiments emulating the state-of-art architectures but with a Recurrent Neural Network. In the end, with the approaches used, a real-time application has been created with the model with highest accuracy. This allows to possible future scenarios.

1.2 Human Action Recognition

Human Action Classification is a Machine Learning task that consists in prediction of different actions starting from a video clip, a sequence of bi-dimensional

frames. The approach could seem similar to Image Classification and its Deep Learning approach but the algorithms are quite different and the success, as well, obtained by this approach is lower. The main obstacles to be tackled for successful action recognition are the followings:

- **Long-range temporal structure modeling**

  The model should be able to recognize action inside a spatiotemporal context and the video could have some movements related to the scene changes but not related to the action, e.g. camera movement, same actions but change of viewpoint. Additionally, the action can last few seconds and an important correlation can involve the first frames with the last ones; long-range temporal features are very important for a correct prediction.

- **Computational cost**

  The most known CNNs have about 25 million parameters. There are not standards Deep Learning model for video classification but usually research papers describe model which have from 35 to 100 million parameters [24]. It implies long training time, research difficult and overfitting problems.

- **Dataset and benchmarks**

  The absence of standards and benchmarks makes comparing different Deep Learning models harder. In the last years, UCF101 and Sports1M have been the most popular datasets and the scientific researches are usually done on one of these two datasets; but not without problems. Looking at the number of generated frames, UCF101 seems to be coherent with the magnitude of ImageNet but it lacks diversity and so the generalization can be difficult. About Sports1M, the dataset suffers of the same problems and it has a number of frames higher then average. In May 2017, a new dataset has been released [25] and it seems to solve the problems previously detailed but only few scientific paper use it to evaluate a model.

- **Classification architecture design**

  Currently, there is not an architecture design, among the ones proposed, that completely outperform the other ones. There is only one certainty: considering the temporal features, with *optical flow*, *RGB differences* or any other methods, will increase the accuracy of the model. Starting from this assumption, there are many valid model architectures that can reach a good accuracy value.

1.3 Structure of the Document

In this paragraph, it is described what is the structure of this thesis work.

- The Chapter 1 is this one and it is an introduction to the work.
- In the Chapter 2 I describe the general techniques, approaches and algorithms. I start from the basic Convolutional Network concepts to arrive to Recurrent Neural Network ones, since they are used in general context of the thesis.
- In the Chapter 3 I present a Dataset Analysis that I have performed in order to better understand the available dataset. Then, I describe the state-of-art architectures that in the last years have reached great result and have introduced some pioneering concept to achieve the goal.
- In the Chapter 4 I described the implementation of my three experiments in details. Here, it is possible to find the details of the models, the diagrams and the hyper-parameters related to the experiments performed.
- In the Chapter 5 I present the results of the three experiments, analyzing the possible reasons of the accuracy values compared to the state-of-art ones.
- In the Chapter 6 I describe a possible implementation of a real-time application using the model which obtain the highest accuracy in my experiments.
- Finally, in the Chapter 7 I present the conclusions and some possible extensions and future works of the presented thesis.

Chapter 2

Algorithms, technologies and approaches

In this chapter, I will describe the common and consolidate techniques and approaches that are the core of several Machine Learning algorithms. The arguments go from the basic Convolutional Network concepts to arrive to Recurrent Neural Network ones, since they are used in general context of the thesis.

2.1 Backpropagation and Loss function

In every Machine Learning supervised learning model, there is a core function called **Loss Function**. In order to understand better the loss function, I first need to describe the score function. The **Score Function** is a function associated to each ML model that given a set of input data (pixels, words, numbers, etc...) it is able to evaluate a score. This score is an indication of how good the model is able to transform an input to a correct output. The score function is generated contemporary to the model creation: each layer of the model introduced a change in this function and the final model will have a specific form that considers all the layers that composed the model. The score function is parameterized by a set of **weights** W which are the trainable parameters of a ML model. The loss function is in charge to evaluate the agreement between the predicted scores and the actual scores. This means that, mathematically, **we must minimize this function to achieve better performance with our model**.

The loss function is parameterized by $\boldsymbol{W}$ and in order to minimize the result

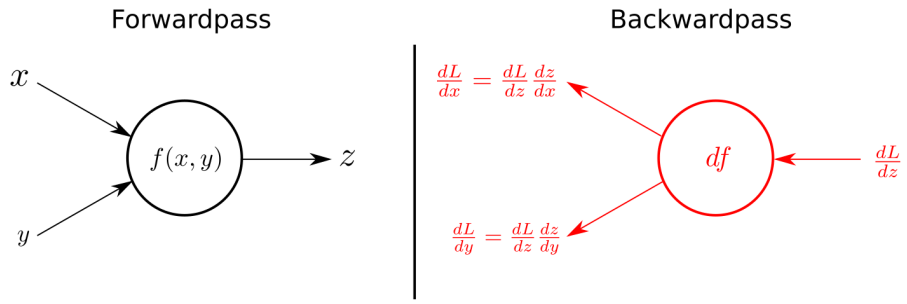

Figure 2.1. Forwardpass and Backwardpass in back-propagation. The former indicates how a result is evaluated. The latter indicates how the weights are updated according to the loss function value and its gradient. Figures from CS231n Stanford course [4].

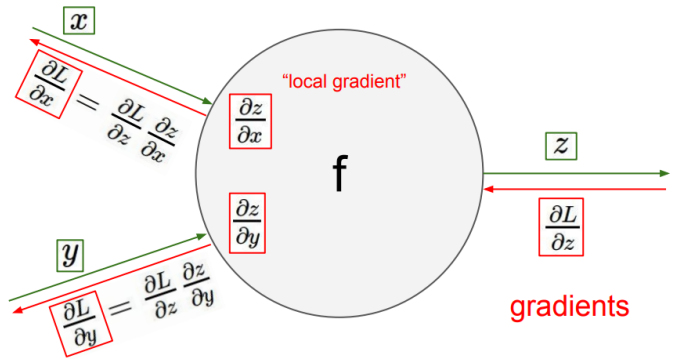

Figure 2.2. In this figure it is represented the mechanism of weights update according to the gradient evaluation. Figure from CS231n Stanford course [4].

of the function we have to update the parameters. The best way to update them is following the direction indicates by the **gradient**. The gradient can be seen as an evaluator of the slope of the function; the weights should be updated following the direction with the maximum slope in order to reach the global minimum of the function. More specifically, the gradient is a generalization of slope for functions that do not take a single number as input but a vector of numbers [5]. Namely, the gradient is a vector of derivatives (e.g. the slopes) for each dimension in the input space.

2.2 Learning Rate

There is an important hyper-parameter, the so-called **Learning Rate**. The gradient is used to estimate in which direction the function has the steepest rate of increase but it does not tell the quantity to move through. The learning rate (or

step size) is a multiplication factor that allow us to go "*slow*" or "*fast*" in the gradient direction with our steps. If the learning rate value is small, we will make a lot of small and well directioned steps so the time to reach a minimum value will be higher. If the learning rate value is big, we will make few steps, reducing the time of training, but there is the risk to reach and pass the minimum value, introducing this oscillatory inefficiency. The changes of learning rate value are represented in figure 2.3. A common approach of nowadays optimizers is to use a learning rate with **Momentum** and **Decay**.

Momentum

The *Momentum Update* is an approach used to increase the learning rate value and the convergence speed in deep networks. The main concept consist in increasing the learning rate when the previous update has obtained good result. Namely, if there is an update to the weights with a specific learning rate and this update allow to obtain a good result with the loss function, the learning rate is increased. And this happens until the condition is true. In this way, if we are correctly orientated through the minimum value of the loss function, it will be easy to reach it.

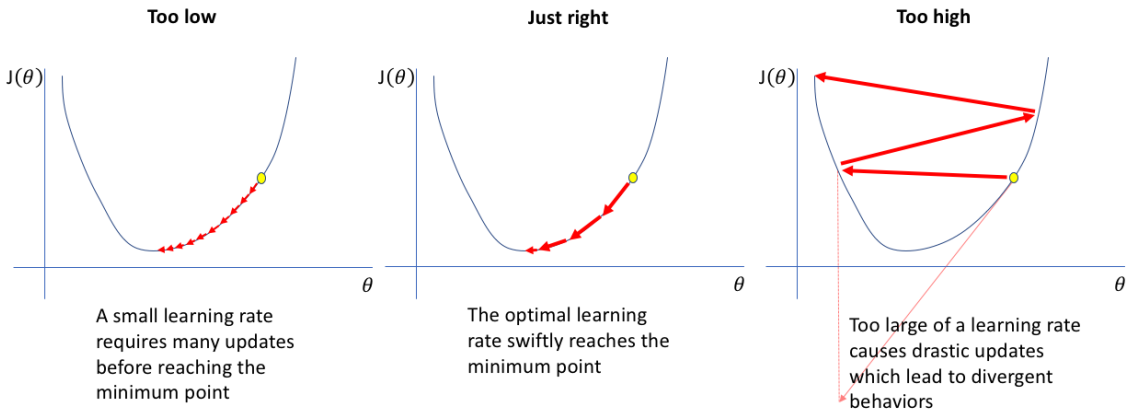

Figure 2.3. In this figure, it is represented a 2D plot of a loss function $\boldsymbol{J(\Theta)}$. In the center there is the correct case, where the learning rate (represented by the factor that modifies the size of the steps) is correctly updated and reach the minimum of the function. On the left the learning rate value is too small and this implies an increase of number of epochs before reaching the minimum value. On the right the learning rate value is too high and this implies that the minimum value will not be reach (divergence). Diagram from [11].

Decay

The *Learning Rate Decay* is an approach used to decrease the learning rate. This is necessary because, when the error function value is near to the minimum value, it could have a big learning rate value and this means that the function value will unpredictably go around the minimum value but without actually settle on it. On the other hand, a too fast decay can increase the convergence time and waste computation. According to CS231n course [4], the learning rate decay could be implemented with three techniques:

- **Step Decay**
  the learning rate is reduced by a specific factor every $\boldsymbol{N}$ epochs.
- **Exponential Decay**
  the learning rate has the mathematical form of $\boldsymbol{\alpha = \alpha_0 e^{-kt}}$ where $\boldsymbol{\alpha_0}$ and $\boldsymbol{k}$ are hyper-parameters while $\boldsymbol{t}$ is the number of the current epoch.
- **$1/t$ Decay**
  the learning rate has the mathematical form of $\boldsymbol{\alpha = \frac{\alpha_0}{(1+kt}}$ where $\boldsymbol{\alpha_0}$ and $\boldsymbol{k}$ are hyper-parameters while $\boldsymbol{t}$ is the number of the current epoch.

2.3 Deep Learning

Deep Learning is a class of techniques of Machine Learning that is involved in modelling algorithms inspired by the structure of the human brain. Basically, it is an extremism of Neural Network where the model generated is a sequence of a **huge number of layers**. The result is a model with a number of weights in the order of hundreds of million that is able to recognize several human-imperceptible pattern inside the dataset. This lead to really good result in the output predictions and they are as better as bigger and well diversified the input dataset is. LeCun *et ali* [2] define '**Representation learning**' as a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep learning methods are representation learning methods with multiple levels of representation obtained by composing a sequence of layers. Again [2] highlights that the key aspect of Deep Learning is that the layers of features are not designed by human engineers but they are learned from data using

a general-purpose learning procedure. These methods have dramatically improved the state-of-the-art in *image classification*, *speech recognition* and *object detection*.

2.4 Convolutional Neural Networks

In the last years, Convolutional Neural Networks (CNNs) have obtained a great success in the *image classification* task. According to Stanford CS231n course [4], CNNs are similar to the ordinary neural network but there is the explicit assumption that the inputs are images; this allows to encode certain properties into the architecture. The main concept of Neural Network assumes that there is a *single* input vector that is transformed crossing a series of hidden weights/layers in order to produce a single output vector. If the input is an image, the input has a shape like ***width*** $\times$ ***height*** $\times$ ***depth***. The usual approach with a Fully Connected (FC) layer would create a really big number of weights and this number is as bigger as bigger the image size is; this approach does not scale. CNN exploits the particular input shape of the image: the weights are arranged in three dimensions in the layers. Moreover, since each pixel (with shape $1\times1\times3$ where 3 is the RGB component, also called *3-channel*) has usually something in common with the adjacent ones, a CNN takes advantage of this property and it is able to summarize local area or common pattern reducing the number of weights. Among the possible type of layers, the following ones are the most common [6]: *Convolutional Layer*, *Pooling Layer*, *Fully Connected Layer* and *Activation Layer*.

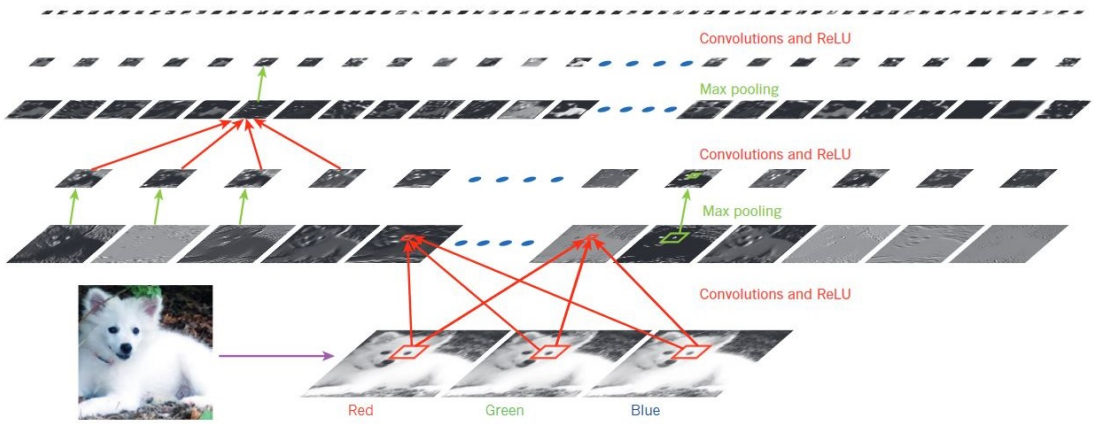

Figure 2.4. Common Architecture for Convolutional Neural Network

2.4.1 Convolutional Layer

Introduction

According to [6], the **Convolutional layer is the core building block of the Convolutional Networks** and it is also the one with the heaviest computational operations. The parameters of a Convolutional layer are a set of learnable filters and each filter has a shape $(\boldsymbol{height} \times \boldsymbol{width} \times \boldsymbol{depth})$. Usually, the height and width dimensions are custom and small while the depth dimension extends through the full depth of the input volume. Each **Convolutional Filter** slides across the width and the height of the input volume and, while sliding, dot products operations are performed. The dot product is done between the entries of the filter and a subset of the input according to the position of the filter while sliding. The **sliding convolutional filter will produce a 2-dimensional activation map that will evaluate the response of that filter in every position of the input volume**. The network will learn some patterns and will represent them in filters that activate when they see the same pattern. For example it is possible to learn edge, shape or block of same color. This will generate a set of filters, each one described by a distinct 2-dimensional activation maps but this leads to have a huge number of parameters for a Convolutional Network; this is the case of first Deep Convolutional Network [28].

Parameter Sharing

In order to reduce the number of parameters, the **Parameter Sharing** technique is usually used. This technique makes a reasonable assumption: **if one filter is able to recognize a pattern, a scheme or a feature at some spatial position, then it should also be useful to recognize it at a different position**; this means that the filter parameters does not change across the different slices in depth. Let's describe this in more details. When a filter is sliding along an input volume, it tries to learn a pattern for each level of depth. Once it has slided in height and width over a slice at the same depth, it has acquire an activation map able to recognize patterns. If we change the level of depth, using as input a matrix of same height and width, the same activation map could be used to recognize the same patterns. This assumption allows us to share the parameters of the filters among all the levels of depth of the input volume. Each filter tries to learn something different from the

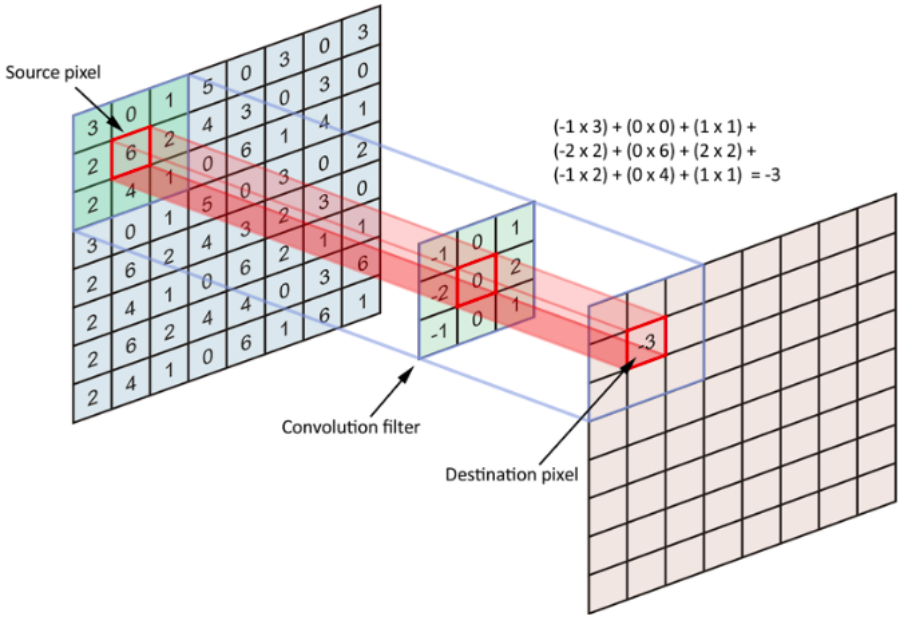

Figure 2.5. Convolutional Layer

input.

Dimensionality

As said, the Convolutional layer receives an input volume with shape ***height*** × ***width*** × ***depth***. There are three hyperparameters which control the size of the output: the depth, the stride and the zero-padding.

- The **depth** of the output volume depends on the number of filters we would like to use. The set of filters that are looking at the same subspace of input is called *depth column*.
- It is necessary to set the value of the **stride** with which we slide the filter. For example, with Stride = 1, the filter is moved one input value at a time.
- The **zero-padding** hyperparameter corresponds to the number of padding value inserted in the input volume. This is really useful to control the spatial size of the output volume.

Let's suppose there is an input volume with shape $\boldsymbol{W}$ and let's suppose there is a Convolutional layer with a filter with size $\boldsymbol{F}$, a stride value of $\boldsymbol{S}$ and a zero-padding

of value $\boldsymbol{P}$. The single dimension of the output volume is equal to:

$$\frac{(\boldsymbol{W} - \boldsymbol{F} + \mathbf{2}\boldsymbol{P}}{\boldsymbol{S} + \mathbf{1}} \tag{2.1}$$

There are some constraints on the hyperparameters and related to the values of stride and zero-padding. The equation 2.1 must produce an integer result.

Supposing that an input volume for this layer has size $\boldsymbol{W_1} \times \boldsymbol{H_1} \times \boldsymbol{D_1}$, supposing that there are $\boldsymbol{K}$ filters, the spatial extent is $\boldsymbol{F}$, the stride $\boldsymbol{S}$ and the amount of zero padding is $\boldsymbol{P}$, then the layer produces as output a volume of size $\boldsymbol{W_2} \times \boldsymbol{H_2} \times \boldsymbol{D_2}$:

$$\boldsymbol{W_2} = \frac{\boldsymbol{W_1} - \boldsymbol{F} + \mathbf{2}\boldsymbol{P}}{\boldsymbol{S} + \mathbf{1}} \tag{2.2}$$

$$\boldsymbol{H_2} = \frac{\boldsymbol{H_1} - \boldsymbol{F} + \mathbf{2}\boldsymbol{P}}{\boldsymbol{S} + \mathbf{1}} \tag{2.3}$$

$$\boldsymbol{D_2} = \boldsymbol{K} \tag{2.4}$$

Moreover, using the Parameters Sharing, the parameters introduced are $\boldsymbol{F} \cdot \boldsymbol{F} \cdot \boldsymbol{D_1}$ for each filter.

In the end, it is important to describe a particular configuration of the Convolution Later: the $\mathbf{1} \times \mathbf{1}$, introduced for the first time by [38]. This kind of layer is able to reduce the dimensional of the network, in fact even if the filter has a shape of $\mathbf{1} \times \mathbf{1}$, the dot products are done through the full depth of the input volume.

2.4.2 Pooling Layer

The Pooling layer purpose is to **reduce the spatial size of the input representation in that point of the network in order to reduce the amount of parameters, the computation of the network and the risk of overfitting**. This layer is commonly inserted between two consecutive Convolutional layers. The general input of this layer has a volume of size $\boldsymbol{W_1} \times \boldsymbol{H_1} \times \boldsymbol{D_1}$ and for a Pooling layer two hyperparameters must be set: the spatial extent $\boldsymbol{F}$ and the stride $\boldsymbol{S}$. The layer produces as output a volume of f size $\boldsymbol{W_2} \times \boldsymbol{H_2} \times \boldsymbol{D_2}$:

$$\boldsymbol{W_2} = \frac{\boldsymbol{W_1} - \boldsymbol{F}}{\boldsymbol{S} + \mathbf{1}} \tag{2.5}$$

$$\boldsymbol{H_2} = \frac{\boldsymbol{H_1} - \boldsymbol{F}}{\boldsymbol{S} + \mathbf{1}} \tag{2.6}$$

$$\boldsymbol{D_2 = D_1} \tag{2.7}$$

This layer operates on every slice along the depth ($\boldsymbol{D}$) dimension, not modifying this dimension but modifying the other two dimensions ($\boldsymbol{W}$ and $\boldsymbol{H}$) according to the hyperparameters selected. The dimensions reduction is due to the operation performed to each specific group of input value, usually a *MAX* operation.. The size of the group depends on hyperparameters. For example if the filter size is 2×2 with a stride of 2, it means that the input values are halved along the two dimensions and so the 75% of parameters are discarded. The *MAX* operation consists in taking the maximum value over the values in the region selected (the pool).

2.4.3 Fully Connected Layer

The Fully Connected layer is a common layer where the neurons in layer have full connections to all the activations in the previous layer. It is a general and traditional pattern introduced for the first time with the Multi-Layer Perceptron network. The activations can be computed with a complete matrix multiplications. It is represented in figure 2.7.

2.4.4 Activation Layer

The Activation layer is a neural layer able to produce an ouput, starting from an input, according to a specific **Activation Function**. **The activation function is a fixed mathematical operation and, therefore, it does not introduce**

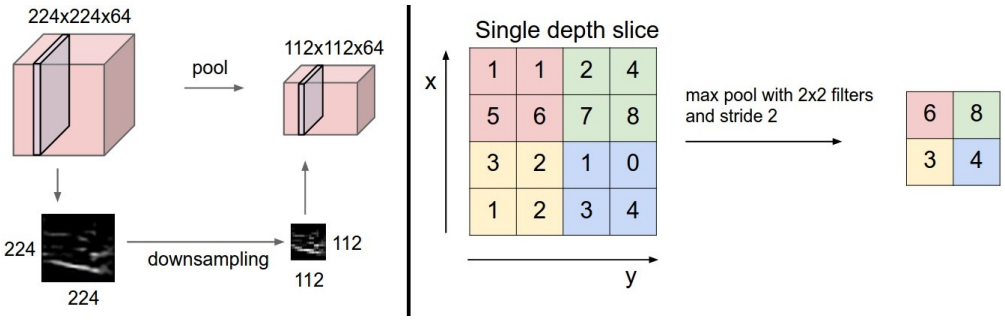

Figure 2.6. Max Pooling layer operations. On the left, there is an example of an input volume and the change of its dimension after a Pooling layer. The depth dimension is preserved. On the right, an example of Max Pooling operation with a 2×2 filter and stride = 2. The maximum value is taken from a pool of 4 values. Image from Stanford CS231n course [6].

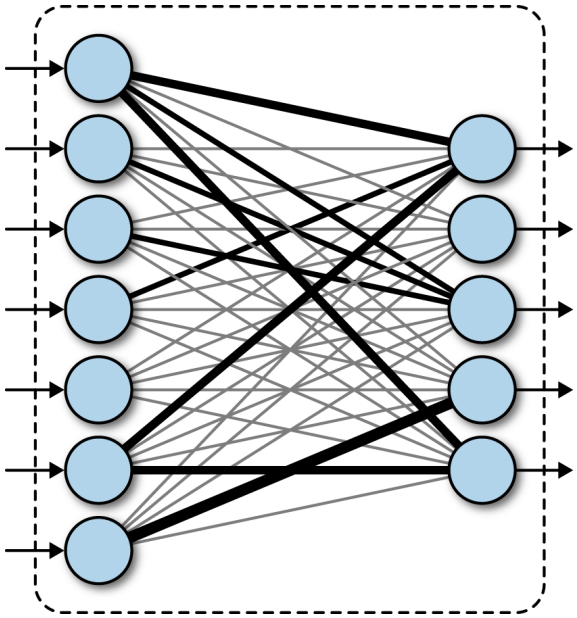

Figure 2.7. Fully Connected Layer

other parameters but it is able to highlight a label among the set of possible ones. The most used activation functions are the following:

Sigmoid

The sigmoid mathematical form is:

$$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-}x}} \tag{2.8}$$

The main ability of this function is to take a real value number as input and return a result between 0 and 1. Namely, large negative numbers are likely to tend to 0 while large positive numbers tend to 1. Historically, the sigmoid has been widely use but, nowadays, it is in disuse due to two important disadvantages [6]:

- The sigmoid **saturates** and "**kills**" the gradient. This means that when the output is in the tail of 0 or 1, the gradient of the function is almost zero and, therefore, during the backpropagation phase, this low value will affect the result of the whole objective function that will not correctly update the parameters.
- The sigmoid outputs are not zero-centered. This has an implication during the gradient descent phase because, according to the sign of the input $\boldsymbol{x}$ the gradient on the parameters update will be all positive or all negative, introducing an undesirable zig-zag update of the weights.

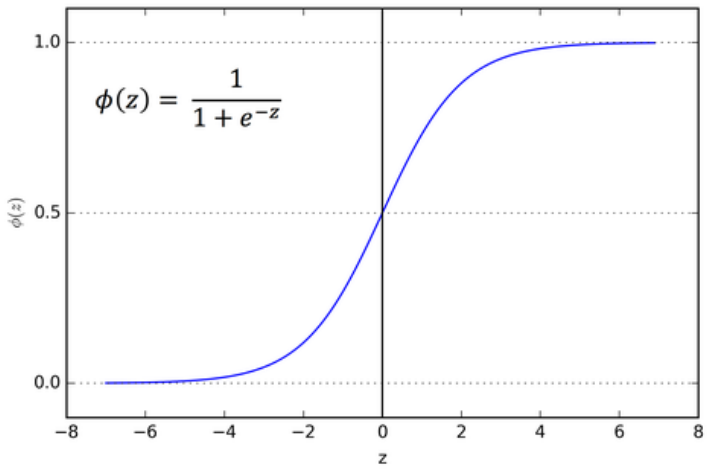

Figure 2.8. Sigmoid Activation Function. Plot from [42].

Tanh

The Tanh activation function has the same capabilities and disadvantages of Sigmoid function except for one thing. The output is from -1 and 1 and so it is zero-centered, avoiding the problem described for the Sigmoid activation function. The mathematical representation is:

$$\boldsymbol{tanh(x) = 2\sigma(2x) - 1} \tag{2.9}$$

In practice Tanh is always preferred to Sigmoid.

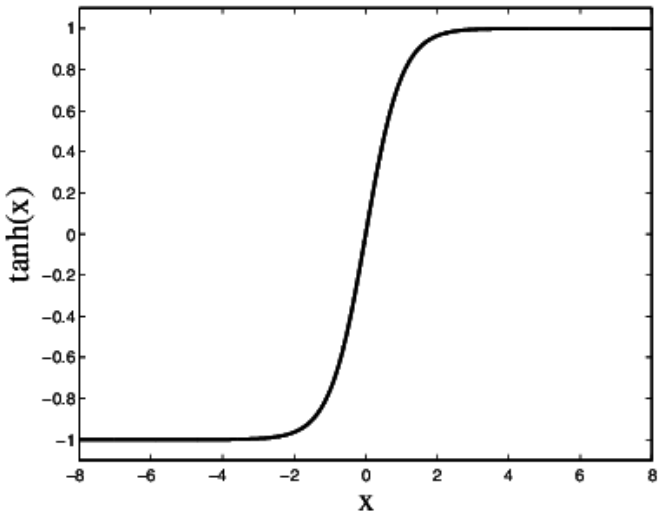

Figure 2.9. TanH Activation Function. Plot from [42].

Rectified Linear Unit - ReLU

The ReLU (REctified Linear Unit) is one of the most popular activation function; the mathematical representation is quite straightforward:

$$\boldsymbol{f(x) = max(0, x)} \tag{2.10}$$

It is simply a threshold at zero. The reason of the large usage are related to:

- It allows an acceleration in the convergence during the gradient descent compared to Sigmoid or Tanh
- The mathematical operation are easy. There are not expensive operations like exponential or fraction but there is only a simply threshold on the value 0

There are, of course, drawbacks as well. Namely, ReLU can suffer of an event called *dying ReLU*. This event happens when the parameter is updated in a way that the activation function will never activate it and if it happens, the gradient flowing in the layer of the Convolutional network will be zero from that point on. A tentative to reduce the drawback of the *dying ReLU* is the usage of **Leaky ReLU**. The mathematical representation is:

$$\boldsymbol{f(x)} = \begin{cases} \boldsymbol{\alpha x} & \text{for } \boldsymbol{x < 0} \\ \boldsymbol{x} & \text{for } \boldsymbol{x \geq 0} \end{cases}$$

where $\boldsymbol{\alpha}$ is small value used to have a small and changing value instead of 0.

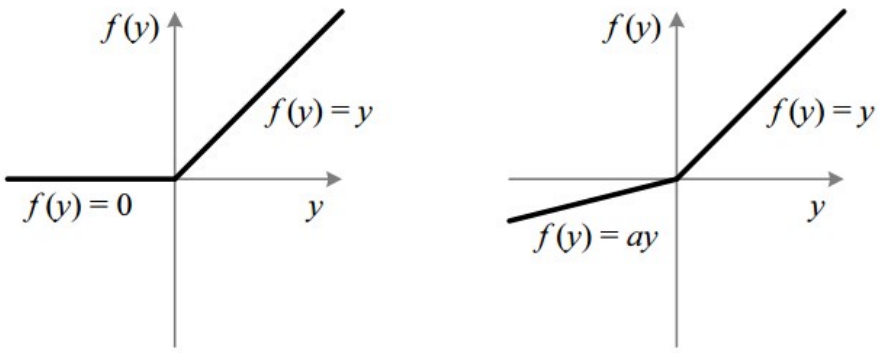

Figure 2.10. ReLU and Leaky ReLU Activation Functions. Plot from [42].

Softmax

The Softmax activation function is a function able to transform a sequence of numbers in input to another sequence of numbers which sum up to one. Basically, this function transform the input in a sequence of probabilities related to the possible labels; the inputs are transformed in output in a such way to be with value between 0 and 1 and the total sum of the outputs is equal to 1. Given $\boldsymbol{N}$ possible labels, the mathematical representation of this function is:

$$\boldsymbol{S(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}} \tag{2.11}$$

2.5 Pre-trained Convolutional Neural Network

2.5.1 LeNet

LeNet-5 is a pioneering CNN architecture introduced in 1998 by Yann LeCun et al [39]. The purpose of this architecture was to recognise hand-written number on bank documents recreated as 32×32 greyscale images. The architecture is pretty straightforward and, nowadays, it is mainly used for teaching purposes. It consists of two sets of convolutional and average pooling layers followed by a convolutional and two fully-connected layers and, in the end, a softmax classifier.

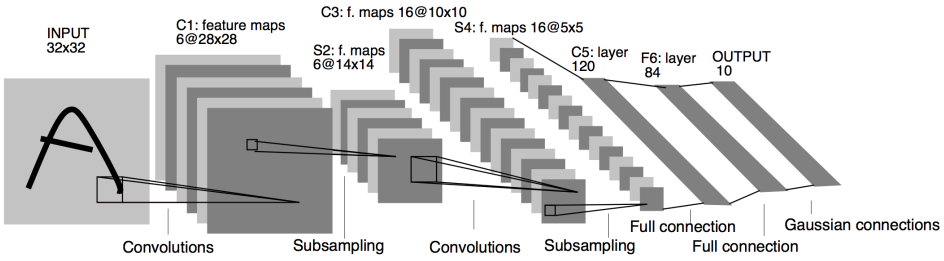

Figure 2.11. LeNet-5 Architecture. Diagram from original paper [39].

2.5.2 AlexNet

AlexNet is a Convolutional Neural Network that, back in 2012, won the Imagenet Large Scale Visual Recognition Challenge (ILSVRC), the annual challenge for object detection and image classification. The architecture was really efficient for that year, in fact it obtained 16% top-5 error and the second best result obtained 26.2%.

The AlexNet architecture is made up of 8 layers, of which 5 are Convolutional layers and 3 are fully connected. The output layer is an Activation layer with *SoftMax* activation function and moreover there are Activation layers with ReLU activation after all the other trainable layers. Moreover there are some non trainable layers as well: 3 Pooing layers, 2 Normalization layer and 1 Dropout layer. The last ones are used to reduce overfitting.

2.5.3 ResNet

ResNet (Residual Network) is a family of architectures which exploits the usage of some **residual blocks** in the model. Going in depth with neural network ensures an increase of the accuracy of the network but doing this we must face some problems and we should take care of them introducing some techniques:

- **Overfitting risk**
- **Vanishing gradient**
  The weights are updated, with the back-propagation mechanism, starting from the final layers. If the network is too depth, this problem arises and, basically, a very low value of gradient arrives to the earlier layers of the network and, although it should update them, its value is low and the update is not tangible.
- **Degradation of Training Error Value**
  This problem is related to *Vanishing gradient* and it happens when the training error value decreases too much when passing through the layers.

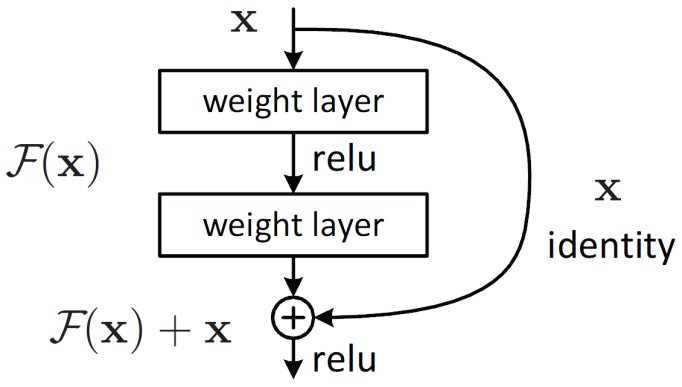

Figure 2.12. Residual block used in ResNet Architecture

The ResNet Architectures aims to reduce the previous problems by means of the introduction of **residual block**. The residual block (figure 2.12 creates a direct path between the input and the output of a specific set of layers (let's call it A). This means that those layers weights (A weights) are updated with a training error value but the following layers weights are update with the sum of this training error value and the training error coming from layers A. So, summarizing, the core idea of ResNet is the introduction of the so-called **Identity Shortcut Connection** which skips one or more layer and bring the value of training error to earlier layers without changing it. Following this approach, in [40] the authors refined the concept of residual block and proposed a **pre-activation variant of residual block**. With this new residual block approach, the gradients can flow through different shortcut paths to any other earlier layers.

2.5.4 Inception v3

In this paragraph, I will describe more in details the architecture of Inception-v3 neural network since it is the network that I used in my experiment described in chapter 4. The Inception-v3 architecture is an evolution of the previous version of the architecture obtained by rethinking the architecture itself, increasing the efficiency and decreasing the number of parameters.

The first version of Inception architecture was introduced as *GoogLeNet* in 2015. Later the architecture was refined with the introduction of **batch normalization** and Inception-v2 was released. Moreover, the architecture was re-factored in order to add **Factorization Convolution**, to modify the **Auxiliary Classifier** and to introduce an **Efficient Grid Size Reduction** and Inception-v3 version was released.

The **Factorization Convolution** consists in reducing the number of parameters without decreasing the network efficiency. The factorization techniques used in Inception-v3 are the following.

- **Factorization into smaller convolutions**
  This technique aims to increase the number of Convolutional layers, stacking them, but reducing the size of the kernel of each layer. For example, 1 layer with 7×7 kernel filter dimension has 49 parameters while 3 layers with 3×3 have 27 parameters. The number of parameters is reduced by 45%. An example

is shown in figure 2.13. With the usage of this technique is possible to modify a single Inception module (basic structure of Inception-vX architectures) and to reduce the number of the network parameters.

- **Factorization into asymmetric convolutions**
  This technique aims to reduce the number of parameters by means of adding some asymmetrical Convolutional layers. The main concept is that it is possible to replace a N×N filter with 2 consecutive layer of size 1×N and N×1 knowing $\boldsymbol{N^2}$ is usually greater than $\boldsymbol{2N}$. For example, 1 layer with 7×7 kernel filter dimension has 49 parameters while 2 layers with 1×7 and 7×1 have 14 parameters. The number of parameters is reduced by 72%. An example in shown in figure 2.14. With the usage of this technique is possible to modify a single Inception module and to reduce the number of the network parameters.

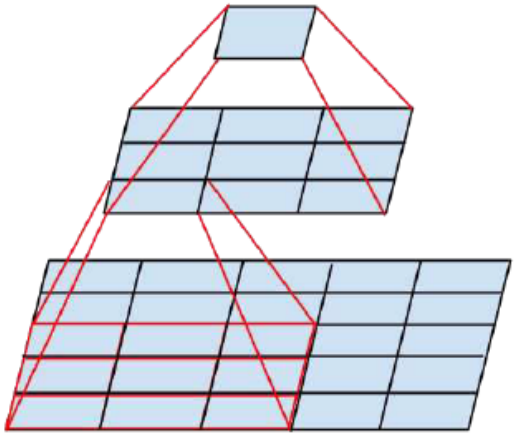

Figure 2.13. Graphic representation of Factorization into Smaller Convolutions where two consecutive 3×3 Convolutional layers replace one 5×5. Diagram from [10].

The **Auxiliar Classifier**, already present since Inception-v1, had some modification with Inception-v3. The v1 version has 2 Auxiliary Classifiers while the v3 version has only 1 Auxiliary Classifier on top of the last 17×17 layers. The purpose of the Auxiliary Classifier is also different: firstly it was used to allow having a deeper network, with the v3 it is used to regularize the network.

Usually, in order to reduce the number of weights, a Max Pooling layer is added. Sometimes, this layer is not really efficient is inserted before a Convolutional layer or it is too expensive if it is inserted after a Convolutional layer. The **Efficient Grid**

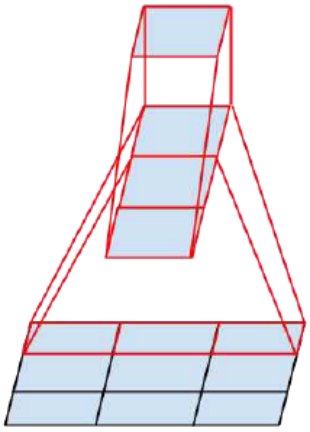

Figure 2.14. Graphic representation of Factorization into Asymmetric Convolutions where a 3×1 layer followed by a 1×3 replaces a 5×5. Diagram from [10].

Size Reduction is a technique that aims to reduce these problems and it consists in create an hybrid situation where in a layer there are both part of Convolutional layer and part of Max pooling layer, concatenated.

In figure 2.15, it is represented the entire **Inception-v3 Architecture**.

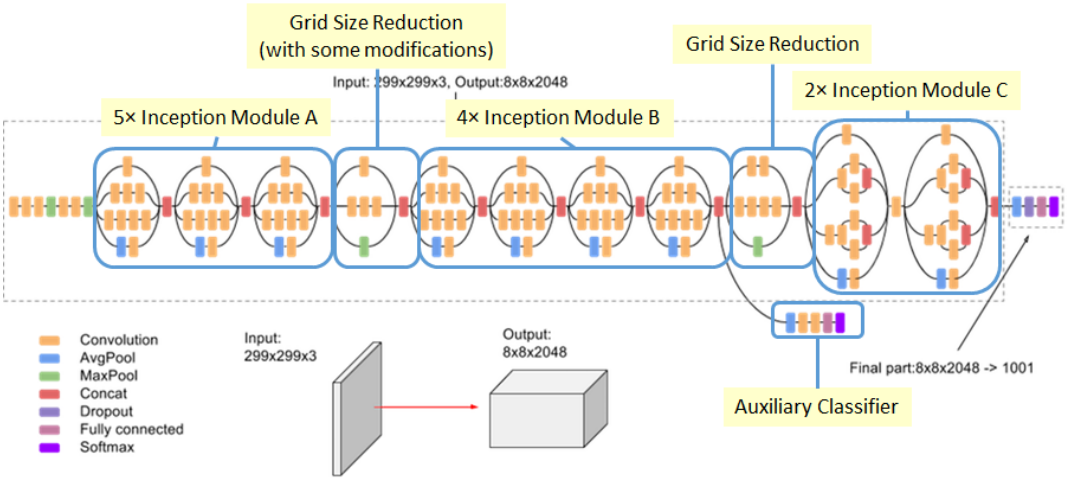

Figure 2.15. In this diagram it is represented the Inception-v3 Architecture. The Inception Module A refers to an Inception Module with Factorization into Smaller Convolutions. The Module B refers to an Inception Module with Factorization into Asymmetric Convolutions. The Module C refers to an Inception Module with a combination of Smaller and Asymmetric Convolutions. Diagram from [10].

2.6 Recurrent Neural Network

The most common pattern of Neural Network is feed-forward. This means that each layer receives the ouput of the very previous layer as input. In some situations, usually related to a sequence of actions/features/events, it is important to consider the previous inputs to generate a correct prediction. **Recurrent Neural Networks** are feed-back Neural Network; they act like they retain memory of what happens in the previous moments (previous input) and use this information to generate the prediction. This means that, at the same time, RNN has 2 inputs: the current input and the input in the previous step.

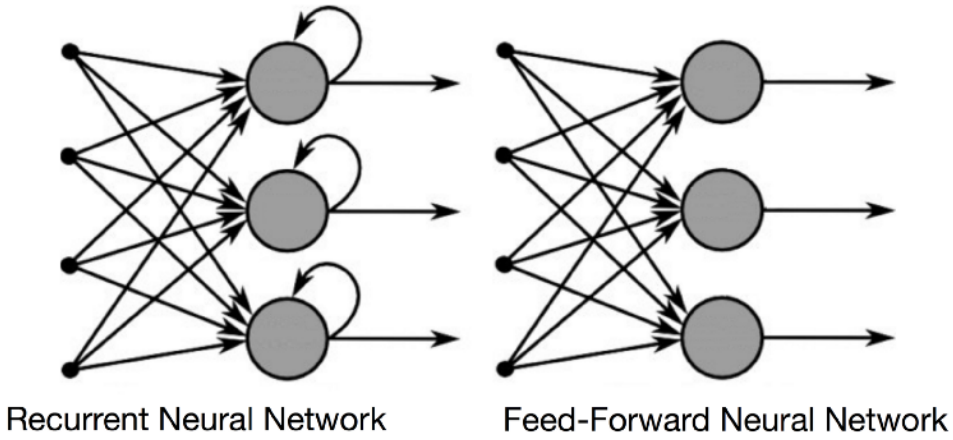

Figure 2.16. Recurrent Neural Network vs Feed-Forward Neural Network

As N.Donges says in [41], this is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

A Feed-Forward Neural Network assigns a weight matrix to its inputs and then produces the output. Namely, the RNN's apply weights to the current and also to the previous input and the weights are updated by means of both gradient descent and Backpropagation Through Time 2.6.1.

2.6.1 Backpropagation Through Time

Backpropagation Through Time, or BPTT, is a typology of Backpropagation training algorithm usually applied to Recurrent Neural Networks (RNNs) and

so to sequences of input data. The main concept in BPTT is that a sequence of data is unrolled and for each single input is applied the classic Backpropagation algorithm. Then, the objective function is evaluated and so the errors across each single input are calculated and accumulated. In the end, the sequence of data is recreated and the process start over again. By means of BPTT, the error is back-propagated from the last to the first input of the sequence, while unrolling all the input. This allows calculating the error for each input, which allows updating the weights.

2.6.2 Long-Short Term Memory

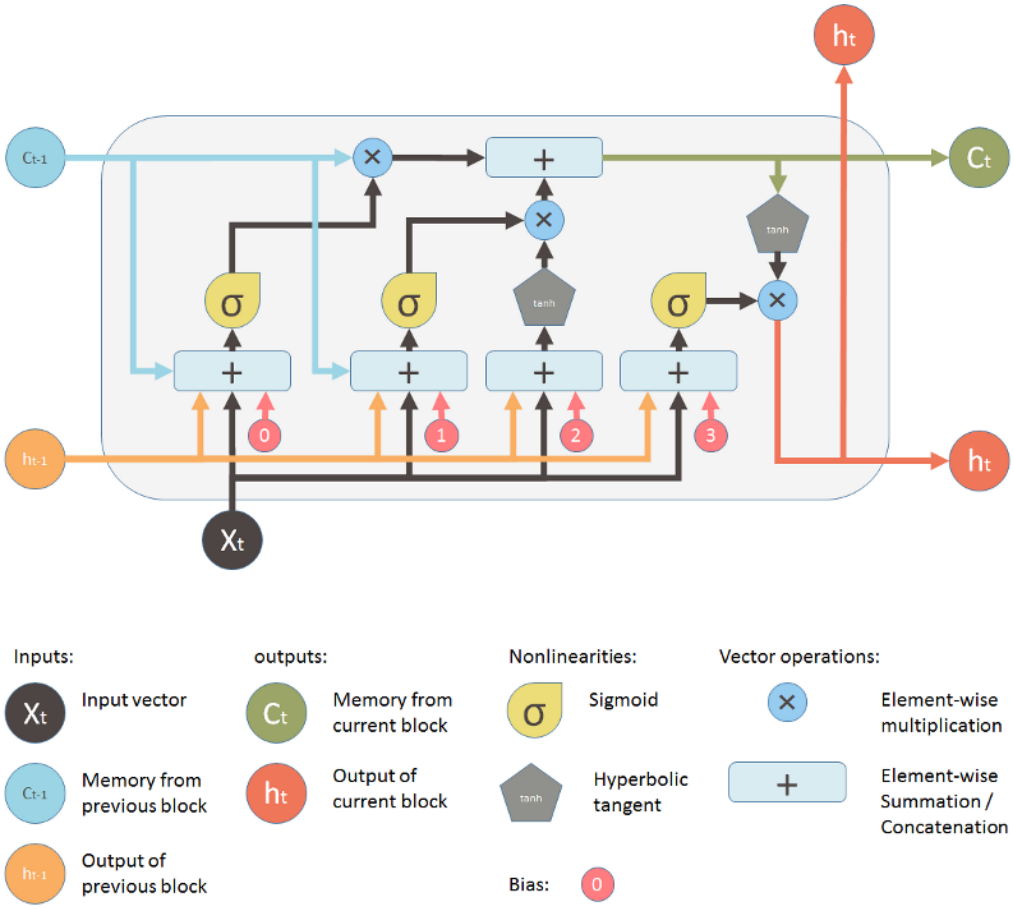

Figure 2.17. A single LSTM unit. It is possible to see that the output produced (new memory state and new output value) depends on different inputs (old memory state, old output value and new input value). Diagram from [8].

Long-Short Term Memory (LSTM) networks are a type of RNN. Namely, a RNN composed by one or more LSTM units is called LSTM network. LSTM networks have been designed to solve the problems that afflict RNN and obtain great success in remembering information for long periods of time.

The fundamental unit of LSTM networks is the *LSTM cell*. It is possible to observe in figure 2.17 that the unit has 3 inputs and and 2 outputs.

INPUT:

- X_t: input value of the current step
- C_{t-1}: memory value generated during the previous step
- h_{t-1}: output value of the previous step

OUTPUT:

- C_t: memory value generated in the current step
- h_t: output value of the current step

Obviously, the unit should not be used alone. The LSTM achieves better results if a *chain of LSTM* is created. In the chain 2.18 is more evident how the output and the memory cell of a step is used in the following steps.

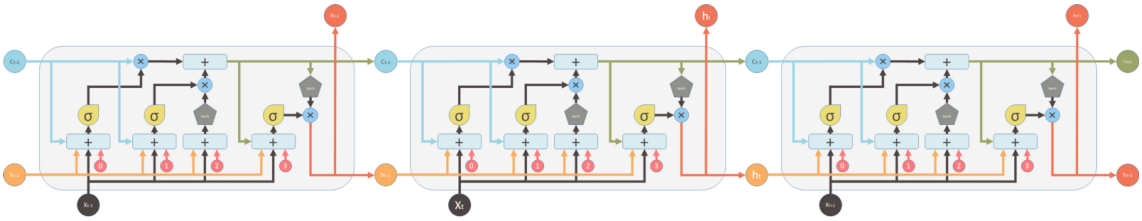

Figure 2.18. Sequence of LSTM unit, generating a LSTM network. Diagram from [8].

There are 2 important operations which are deeply involved in changing the value of the output of the LSTM unit.

- The first operation is an element-wise multiplication: this operation is in charge to set *how much* the memory value of the previous step affects the output memory value of the LSTM unit. It is highlighted in the figure 2.19.
- The second operation is an element-wise summation or concatenation: this operation is in charge to set how much the memory value generated during the previous step affects the output value of the unit. It is represented as the 4 light-blue rectangles with a + in the lower part of figure 2.17.

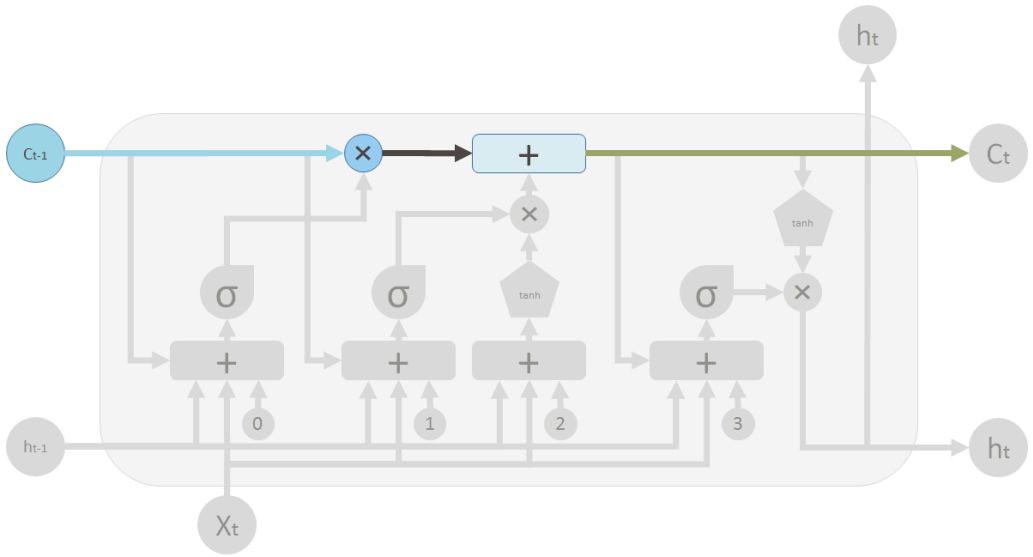

Figure 2.19. This diagram represents the way how the old memory value affects the new memory value. Diagram from [8].

2.7 Keras

Keras is an high-level Deep Learning framework. It runs on top of some of the main Machine Learning frameworks, namely Tensorflow and Theano, and it is written in Python. As [12] says, Keras has some important properties and principles:

- **Modularity**: during the creation of a Neural Network, each *component* (e.g. layer, activation function, optimizer, etc•) can be defined on its own and a model is build defining each component. This allows to highly customize the model, the approach and the experiments done.
- **Extensibility**: each component is a single functional unit, this allows the engineers and the researchers to use the framework for almost any scope and target.
- **User friendly**: the framework is build to be used by human beings, providing APIs which follow best practices and reduce the number of interactions with the below Machine Learning framework.

In Keras, it is possible to generate model in two different way: *Sequential* pattern and *Functional API*. Usage of Sequential pattern is pretty straightforward. The model is created by adding layers sequentially. It is enough to declare the input shape in the first layer and the parameters for each layer. On the other hand, Functional

API are used to created more elaborated model that can have *inception* module or multiple inputs/outputs.

Chapter 3

State of the Arts

This chapter introduces the state of the art achieved by some recent approaches to video classification problem and described in academic research papers. The chapter starts with a dataset analysis performed to better choose which dataset could be used in the experiment described in the next chapter. The analysis is followed by a sequence of video classification techniques which described the evolution that video classification is having through time.

3.1 Dataset analysis

Before starting my experiments, I performed an analysis on the public available dataset that I could use for video classification, namely Human Action Recognition task. The analysis involved several datasets and two criterion have been used to choose the dataset to use for the experiment:

- **Research usage**: if a dataset is used in several academic research papers, it is easier to make a comparison with my experiments to evaluate them.
- **The size of the dataset**: since I performed some offline preprocessing and I used a limited storage AWS instance, the dataset should not be too large so that performing an experiment lasts a reasonable time.

3.1.1 Available datasets for Human Action Recognition

This section presents a sequence of details about the main Human Action Recognition datasets. For each dataset the details are:

- Number of video clips
- Number of actions (number of unique labels)
- Video clips quality
- Year

YouTube-8M

YouTube-8M Dataset is a 2016 large-scale labeled video dataset [46]. It does not consist of video but YouTube video IDs correlated with a specific label. Specifically, there are 6,1 million video IDs for a total amount of 350k hours of video. The video clips are divided in 3862 different classes. Class label are not manually-generated but they are machine-generated from an Inception-V3 CNN trained on ImageNet.

Kinetics

Kinetics (namely Kinetics-600 [53] is a relatively new dataset of human action (2017). It is composed by 500k high-quality video clips from YouTube, single labeled. It is divided in 600 human action classes and for each class there are at least 600 video clips.

YouTube Sports-1M

YouTube Sports-1M is a large video dataset released in 2014 by Google with the Single Stream Network original paper [37]. The dataset is composed by 1 million YouTube video clips, labelled in 487 different action classes.

UCF101

UCF101 is an action recognition dataset [15] released in 2012 and composed by 13.320 video clips from YouTube manually labeled in 101 different human action. This dataset tries to give diversity in terms of variations of camera motion, object appearance, pose, viewpoint and background. This dataset is an extension of UCF50 dataset which has 50 action classes.

HMDB51

HMDB (Human Motion Database) is a 2011 dataset [47] composed by 6849 video clips from YouTube divided in 51 action categories, each containing at least 101 video clips.

ActivityNet 200

ActivityNet-200 [48] is a 2016 video action dataset used for benchmark. It aims to cover a wide range of huuman activities involving daily living. It is composed by 648 hours of untrimmed video divided in 200 labelled classes.

HOLLYWOOD2

HOLLYWOOD2 is one of the first publicly available dataset for video action classification. It is available since 2009 [49] and it is composed by 3,5k video clips divided in 12 classes.

KTH University - Recognition of human actions dataset

KTH Recognition of human actions dataset [54] is a relatively old action recognition dataset composed by around 2,4k video clips manually divided in 6 human action class. The actions are performed in 4 different scenarios.

Charades

Charades and Charades-Ego [50] are two dataset recent (2017) video action recognition datasets. The first one is composed by 9,8k video clips of daily indoors activities, containing 66k temporal annotations for 157 action classes while the second one is composed by 7,8k video clips of daily indoor activities, containing 68k temporal annotation for 157 action classes.

3.1.2 Available datasets for generic actions

20BN-Jester

The 20BN-Jester dataset [51] is a human hand gestures dataset composed by 148k video clips recorded in front of a laptop camera. It is manually labeled with

Dataset	video clips	labels	year
KTH University Dataset	2,4k	6	2005
HOLLYWOOD2	3,5k	12	2009
HMDB51	6,8k	51	2011
UCF101	13,3k	101	2012
YouTube Sports-1M	1 million	487	2014
ActivityNet 200	20k	200	2016
Kinetics	500k	30	2017
Charades	9,8k	157	2017
Charades-Ego	6,8	157	2017
YouTube-8M	6,1 million	3862	2018
20BN-Jester	220k	27k	2017
20BN-Something-Something	220k	27k	2017
Moments	1 million	339	2018

Table 3.1. Statistics about datasets analyzed

27 labels, this means that about 5k video clips are available for each hand gesture while there is the class (*Doing other things*) that collects about 12k video clips itself.

20BN-Something-Something

The BN-Something-Something v2 dataset [52] is a large dataset of video clips which involves humans performing some actions with everyday objects. It is composed by about 220k video clips divided in 174 labels, manually annotated. There are about 3,7k video clips for each label.

Moments in Time

Moments in Time is a large video dataset [55] related to a research project by the MIT-IBM Watson AI Lab. The project (2018) is dedicated to building a very large-scale dataset to help AI systems recognize and understand actions and events in videos. It is composed by 1 million 3 seconds video clips divided in 339 action classes.

3.2 Video classification architectures and approaches

3.2.1 Single Stream Network architecture

The Single Stream Network architecture is a work made by Karpathy et al. in June 2014 [37]. According to this work, everything starts from the great success achieved by **Convolutional Neural Network** in image classification, segmentation and detection. They have tried to apply the same concepts to video clip. As said in 3.1.1, they introduced their own dataset due to the lack of a benchmark dataset used by scientific community. About the techniques introduced, the term *single* is related to the fact that only the spatial information are evaluated and considered for the final prediction; in this early phase of deep learning usage for human action recognition, the spatiotemporal information are not considered. They explored and experimented how to represent the temporal behavior by **stacking a sequence of consecutive frames using 2D pre-trained convolutional neural networks**. Namely, they evaluated multiple ways to merge this sequence of frames.

Despite these good ideas in experimenting, the authors realized that the results were worse compared with the state-of-the-art which in that moment was related to the hand-crafted feature algorithms. They assume that the reasons of the failure are: the lack of spatiotemporal features extracted from the sequence of frame and the fact that is difficult to learn too much detailed features on a low-diversity dataset. On the other hand, they introduced a first good approach to transfer learning; they trained their model on YouTube Sports-1M and then transferred learning evaluating the UCF101 dataset.

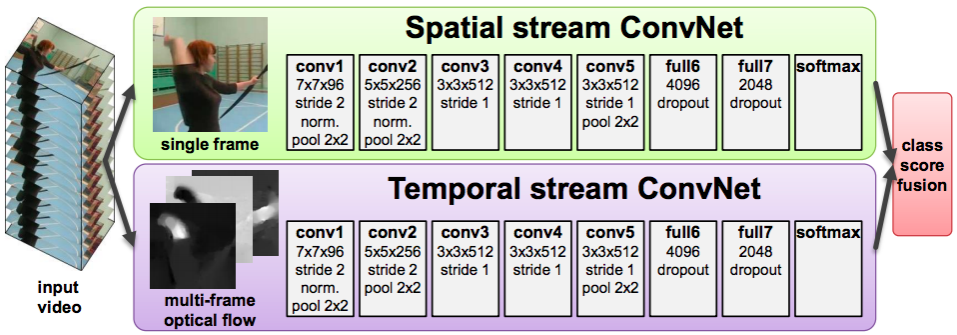

Figure 3.1. Two-stream architecture for video classification [29]

3.2.2 Two Streams Network architecture

The main concept of the Two Streams Network architecture is related to the pioneering work done by Simonyan and Zisserman in 2014 [29]. They understood which were the problems related with Single Stream Network and created an architecture that aims to make them less effective. In the name, *two* is related to the fact that two subnetworks are created to manage separately the **spatial features** and **temporal features** by means of **two separated convolutional neural networks**. The spatial subnetwork is fed with single frame in input while a sequence of 10 successive stacked optical flow frames is the input of the temporal subnetwork. The two streams are singly trained and the prediction results are combined by using Support Vector Machine, exploiting an average across sampled frames. This approach increase the performance of the single stream architecture due to the fact that temporal features are evaluated explicitly. The architecture laid the foundations for several future approaches. On the other hand, there are still some disadvantages:

- There is the need to pre-compute optical flow vectors and store them locally
- The final prediction is obtained from averaging the predictions coming from sampled frames so the long range temporal information is not evaluated in the feature.

3.2.3 Long-term Recurrent Convolutional Networks

In a previous work by Ng et al. [32] authors had explored the idea of using LSTM networks to see if they were able to capture temporal information from clips. The main concept of this technique is to use **Recurrent Neural Network** instead of Convolution Neural Network to better extract temporal features. Later, Donahue et al. [33] proposed a paper that, starting from Ng et al. works, introduced some new ideas and techniques by means of the usage of LSTM blocks (decoder) after convolution blocks(encoder) and using end-to-end training of entire architecture. They also noticed that the final prediction is higher if they consider more the optical flow prediction compared to RGB one. So that, they introduced a **weighted scoring prediction** technique that should ensure an higher accuracy. The architecture is trained with 16 frames (both optical flow and RGB) sampled from a video clips. The final prediction for each clip is evaluated as the average of predictions across each time step.

3.2.4 Temporal Segment Network

Temporal Segment Network [31] architecture is a good enhancement of the Two Stream Network and it currently generates the state-of-the-art results. According to the paper, there are two main problems during predictions generation and they are mostly a direct effect of the disadvantages explained in 3.2.2.

- Long-range temporal information has an important role in understanding the dynamics of a video clip. Common CNN approach is focused on short-range temporal information, usually a frame, and this leads to a reduced ability to understand the global contexts and so the actions.

- The size of the dataset plays an important role in the experiments. Usually, if the dataset is big and it contains heterogeneous elements, it allows CNNs, or any Deep Learning Networks, to achieve an higher accuracy and better performance. The datasets available for Human Action Recognition are limited in both size and diversity and, consequently, the network has an high risk of over-fitting.

The paper introduces two major improvements:

- Video clips are **divided in segments** and then are sampled to better model long range temporal features.

- In order to create the final prediction, temporal and spatial streams are firstly evaluated separately by averaging each video clip and the final spatial and temporal features are used for a **weighted average** that generate the final prediction over all classes.

Moreover, some important techniques and best practices are described in the same paper. They consolidate some general preprocessing operations that aim to increase the general performance of a video classification process with a dataset that can suffer overfitting problems. Namely, batch normalization, dropout and pre-training. Additionally, the authors have evaluated two different approaches to consider the optical frames: the warped optical flow and the RGB difference.

3.2.5 3D Convolutional Networks

This architecture has been introduced by Tran et al. [34] in 2015. Inspired by the deep learning breakthroughs in the image domains, they supposed that the same techniques could be applied to a video clip as well. Usually convolution is done on a 2D frame but the core concept of the 3D convolutional networks is that the **convolution technique is applied on the whole video**. This implies that, for video classification, convolutional operations are done across all the frames of a video, with a kernel size that includes also a new dimension, as it is shown in figure 3.2. The original idea involved the training of these networks on Sports1M and then the usage of the model as a features extractor for other datasets.

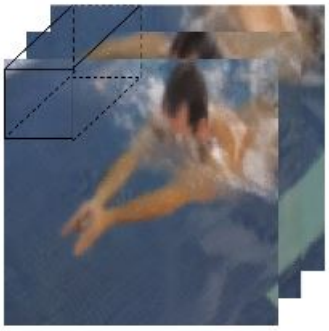

Figure 3.2. Convolutional operation done on a sequence of frames

3.2.6 Two-Stream Inflated 3D Convolutional Networks

J.Carreira and A.Zisserman proposed [35] a sequence for the 3D convolutional network approach. The main difference introduced with the Two-Stream Inflated 3D Convolutional Networks (I3D) is related to the fact that instead of a single 3D network, there are two different 3D networks, one for each stream of the Two Streams architecture.

Thanks to this approach, an important fact has been realized. The usage of a pre-trained 2D model can increase the performance of the system if applied to the sequences of video clip frames. The spatial stream input now consists of frames stacked in time dimension instead of single frames as in basic Two Streams architectures.

3.2.7 Temporal 3D Convolutional Networks

Diba et al. proposed [36] this new approach that continues to extend the work done on 3D Convolutional Network. The idea is the usage of a multi-depth temporal pooling layer, called *Temporal Transition Layer*, starting from a single stream 3D DenseNet based architecture. This new layer is positioned after dense blocks and it is used to capture different temporal depths. The multi-depth pooling is achieved performing pooling operations with variable temporal kernel sizes.

Moreover, in the same paper the authors also introduce a new technique of supervising transfer learning, represented in figure 3.3. This technique involves passing the learning from a pre-trained 2D convolutional neural networks to a Temporal 3D convolutional networks. RGB frames are generated from each video clip and then they are presented two by two as input of the two networks. Then the whole model (both networks joined) is trained to predict True/False if the pair of frames come from the same video clip or not. When the T3D makes a wrong prediction, the error is back-propagated and updated the weights of the T3D, effectively transferring knowledge from one network to the other.

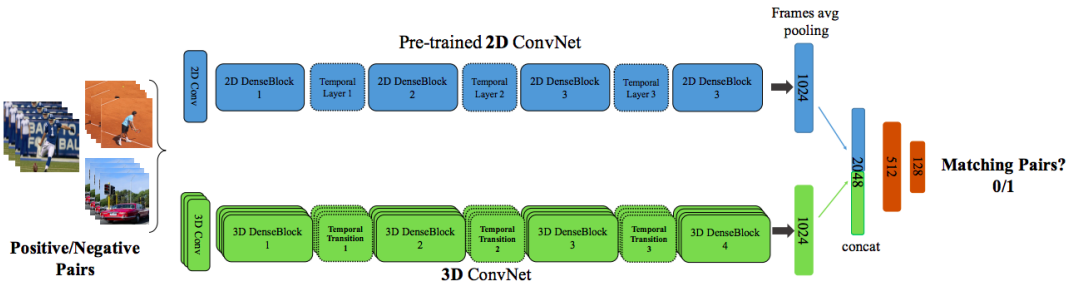

Figure 3.3. Knowledge transfer architecture from a pre-trained 2D ConvNet to 3D ConvNet

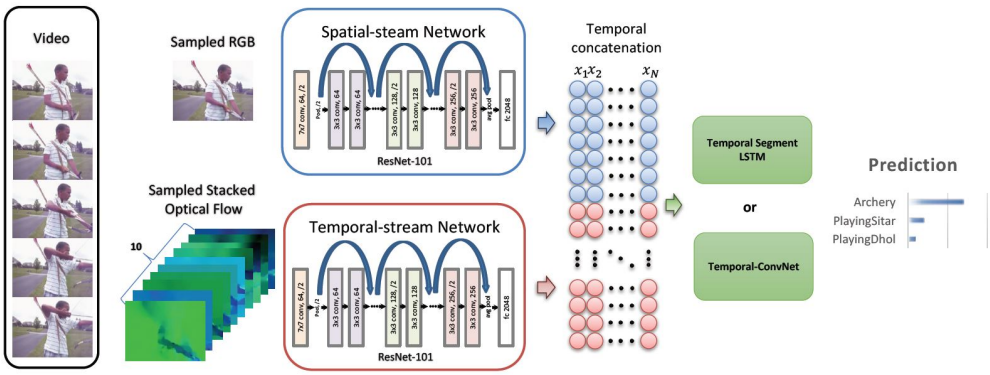

Figure 3.4. Model Architecture of TS-LSTM [43]

Chapter 4

Experiments Design and Implementation

4.1 Introduction

I performed a series of experiments where each one adds some enhancements to reach better results, taking ideas and approaches from the remarkable scientific papers. Among the available datasets, I chose to use **UCF101**. I have implemented the experiments in **Python 3.6** as programming language and **Keras v2 Tensorflow** backended as Deep Learning Framework. All the codes, the scripts and the results are available in a public repository in my GitHub profile [14].

In paragraph 2.6 I described the approaches and the capabilities of the **Recurrent Neural Networks**. They are obtaining a great success in supervisioned learning model with the presence of time series and, so, they could be suitable and equally efficient in a video classification where a video can be seen as a **sequence of image** and an image can be seen as a **sequence of values** generated by a trained Convolutional Neural Networks. Starting from three scientific papers, I referred to and try to improve 3 of them and namely the architecture described by them: **Single Stream Convolutional Network** [37] described in paragraph 3.2.1, **Two Stream Convolutional Network** [29] described in paragraph 3.2.2 and, in the end, **Temporal Segment Network** [31] described in paragraph 3.2.4. For each of them a propose a solution using Recurrent Neural Network by means of LSTM layer. Hyperparameters and other information are summarized in the table 4.1. Chih-Yao Ma *et al.* exploits a similar approach in [43] so the results of my experiments are compared

to Ma's ones as well. The experiments performed and their genesis are:

Single Stream LSTM Recurrent Neural Network

In this experiment, described in paragraph 4.5, I create a Recurrent Network starting from the architecture of Single Stream Convolutional Network [37]. The differences introduced are related to input of the network, the layers and the number of parameters. The **input** of the Single Stream Network is represented by a sequence of image while the input of the Single Stream RNN is represented by a sequence of predictions generated from the **RGB frames** of the video clips, by means of a pre-trained CNN (Inception-v3).

Two Streams LSTM Recurrent Neural Network

This experiment, described in paragraph 4.6, is a *natural* follow-up to the previous one. In the Two Stream Convolutional Network [29], starting from a video clip, a sequence of RGB and a sequence of Optical Flow frames are generated and are used as input of the model. The differences introduced are related to the input of the network, the layers and the the number of parameters. The **input** of the Two Streams RNN is represented by a sequence of predictions generated from the **RGB frames** of the video clips and a sequence of predictions generated from the **Optical Flow frames**, by means of a pre-trained CNN (Inception-v3).

Temporal Segment Network LSTM Recurrent Neural Network

This experiment, described in paragraph 4.7, extends the previous one with the usage of one of the remarkable technique introducing by Temporal Segment Network [31]. Namely, each video clip of the dataset is **divided in segments** and each segment is used as input of the model. The output is generated with a consensus function, evaluating the prediction of the segments belonging to the same video clip.

4.2 UCF101 - Action Recognition Dataset

I conducted experiments on a large dataset called UCF101 [15], this dataset is very popular in scientific research papers and so it can be easy to make a comparison

between models and architectures. It contains 101 action classes distributed among 13320 video clips; it is divided into 25 different groups, each composed by 4 to 7 video clips. Generally, video clips are quite different from each other in terms of background, sight corner, point of view but the ones inside the the same group share some visual properties. Moreover, it is possible to divide the classes in 5 macro-classes: *Human-Object Interaction*, *Body-Motion Only*, *Human-Human Interaction*, *Playing Musical Instruments* and *Sports*.

Figure 4.1. UCF101 dataset

I followed a cross-validation approach as evaluation scheme with the three official training and testing splits. In this evaluation approach, each official split is used as a training subset and validation subset; at the end, there will be 3 validation values for each measured metric. The final metric value is the average of the 3 values for each metric.

In my implementation, I found really helpful to model each video as an object of class *VideoObject* in order to have all necessary information (group and clip number, name, file path) in one single object. In the `preprocessing` module, there is a script called `dataset_split.py` that is in charge to divide the entire dataset into two lists of *VideoObject* objects, one for training and the other for validation, according to the three official splits.

4.3 Frames generation

Since all the experiments require frames as input, I chose to generate all the RGB frames offline. In order to achieve this task, I used **FFmpeg** [16] open source, free software project that provide some tools for handling video and photo files. I have generated a number of frames at 25 fps by means of FFmpeg and store them, using lexicographic order for the filename so that the temporal sequence could be re-created. Basically, the UCF101 dataset becomes a series of RGB frames file for each video clips.

As it is described in details in the following pages, the second and the third experiments require also *Optical Flow* frames. **Optical flow** [19] [21] is a pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. Given two frames, it is possible to generate an optical flow image that represents the displacement of each point from the position in the first frame to the position in the second one. Two assumptions are done: pixel intensities of an object do not change in two consecutive frames and neighbouring pixels have similar motion. These frames are used to *extract the temporal behavior* of a video clip.

According to my implementation, in order to generate RGB frames, it is possible to exploit the `frames_generator.py` Python script in the `preprocessing` module for all the experiments and the Python script `flow_frames_generator.py` in the same module for the experiments that require *Optical Flow* frames.

I generated the *Optical Flow* frames with two different techniques. The first technique I used is available in the **OpenCV2** [20] library for Python. The library provide different typologies of optical flow dense estimation but according to [17] **Farneback Optical Flow Estimation** [18] is the one which better estimates the temporal properties of this dataset video clips, among the ones available in the

library. The second technique consists in the usage of a CNN for the optical flow estimation. The network is called **LiteFlowNet** [22] and it is a recent optimization of the FlowNet2 network for the optical flow estimation where optimization consists in reducing the number of parameters and increasing the running speed. It provides a specific optical flow technique, called *Warped Optical Flow* that, according to [31], reaches better performance in action recognition experiments.

Moreover, recently NVIDIA released an SDK toolkit, called **NVIDIA Optical Flow SDK** [56], able to generate Optical Flow estimation frames, optimizing the computation for its GPU. Even if my experiment are done with an NVIDIA GPU, the SDK toolkit required a specific version of CUDA library (10.0) for working properly while the instance I used has an inferior version (9.0).

4.4 Hardware setup and callback functions

The experiments have been executed on a Amazon Web Services (AWS) Elastic Cloud Computing (EC2) instance provided by **GRAINS research group** at Politecnico di Torino. The EC2 instance is equipped with Nvidia Tesla K80 GPU with 12 GB of graphic memory.

In the experiments, I have used some callback utility functions:

- **ModelCheckpoint** This function is used to save the model weights after each epoch.

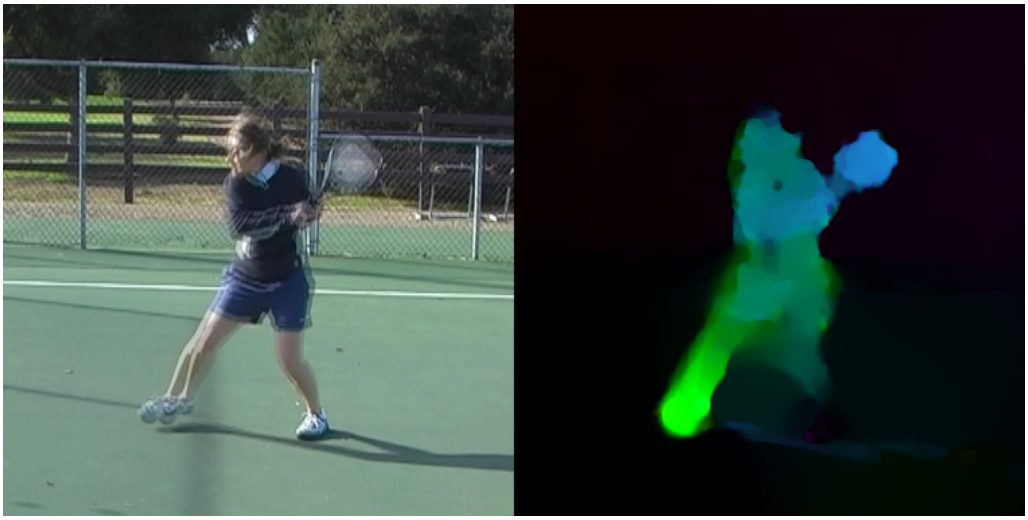

Figure 4.2. Optical Flow frame for a *Playing Tennis* action

- **CSVLogger** This function is able to write in a log file all the metrics, evaluated for both training and validation sets, for each epoch. The log file has been used to extract information and generate some plots of the training phase.
- **EarlyStopping** This function is able to stop the training process if the loss value on the validation set has not decreased after a specified number of consecutive epochs. It is useful to avoid overfitting.

4.5 Single Stream LSTM Recurrent Neural Network

4.5.1 Design

The architecture of the model consists of a Recurrent Neural Network with an LSTM layer. The input of the LSTM layer is a sequence of predictions extracted from Inception v3 CNN pre-trained on ImageNet dataset generated by RGB frames of the dataset video clips. The single prediction per RGB frame is represented as a 2048 length NumPy array and it is the output of the `avg_pool` layer of the original Inception-v3 model.

The design choice of this experiment is mainly related to the great success of Convolutional Neural Network in Image Classification. The final layers of a CNN contain important properties related to a frame, represented as an array; during my experiments I call *video feature* an array of values composed by a prediction for each RGB frame for that video clips. Moreover, I chose to use an LSTM layer since, as already described in 2.6.2, they are composed by LSTM units which consider what previously happens and a memory state to generate a prediction.

The output of the LSTM layer is a Dense layer, followed by a Dropout layer and another final Dense with softmax layer. I used the Sequential pattern in Keras to create the model and the Python script that was used to generate the model is available in the appendix A.1.

4.5.2 Preprocessing

Once the frames have been generated, they are fed one by one in the Inception-v3 CNN as input and a 2048-dimension array is generated with the `predict()` Keras

Model method. According to the `feature_sequence_length` hyper-parameter value, N predictions, coming from N sequential frames, are orderly stacked in a NumPy array. This means that for each video is generated a NumPy array with `(feature_sequence_length,2048)` shape and it is stored locally in the file system. Since Human Action Recognition is a classification of categorical variables, I use One Hot Encoding to represent each class as a binary vector.

In my implementation, it is possible to perform these operations by means of `features_extractor.py` Python script, inside the `preprocessing` module.

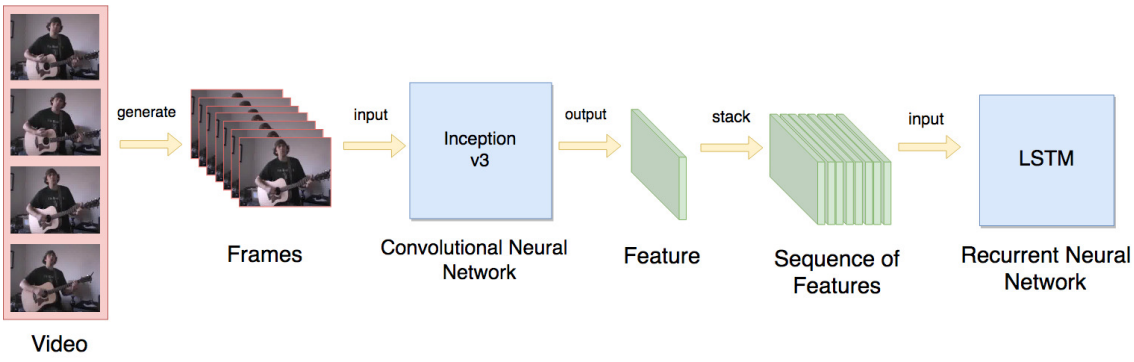

Figure 4.3. Preprocessing pipeline of Single Stream LSTM RNN

4.5.3 Training

I trained this model a couple of times and among the hyperparameters, I chose to change some of them and use constant values for others; specific information are available in table 4.1. Since I had to classify video clips among a specific subset, I used `categorical_crossentropy` [27] as loss function while `accuracy` and `top_k_categorical_accuracy` have been chosen as metrics. The optimizer used is `Adam` [26]. About the not-constant parameter, using the same name variable of the code snippet in appendix A.1 the `lstm_unit` value is set to 2048. I started from 1024 but increasing the parameter leads to better accuracy value. About the dropout values (parameters `a` and `c` in the appendix code A.1), I changed them in an interval between 0.4 and 0.7 but they do not reasonably change the accuracy value. This is in contrast with [29] but it is possible since the approach is slightly different. For the number of unit in the Dense layer (parameter `b`) the value chosen is 512. About the optimizer, the model has been trained with different values for `learning_rate` and `decay` but in the end, they have been set to 10^{-5} and 10^{-6} respectively. Each

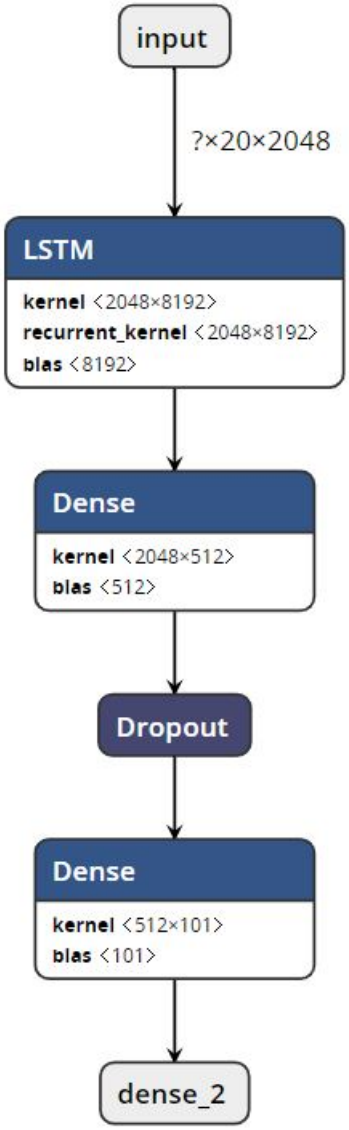

Figure 4.4. Single Stream LSTM Recurrent Neural Network model diagram.

training phase lasts about 45 minutes and then the Early Stopper callback function interrupts the training to avoid overfitting.

4.6 Two Streams LSTM Recurrent Neural Network

4.6.1 Design

As introduced in paragraph 4.1, this experiment aims to re-propose the **Two Stream Convolutional Network** [29] but with a **Recurrent Neural Network**. The architecture of the model is composed by 2 subnetwork, namely 2 Recurrent Neural Networks, each one with an LSTM layer. The reason of the presence of 2 subnetwork is related to the fact that both spatial and temporal behaviors are now analyzed. **The 2 subnetworks in the model are parallel** and have almost the same base structure. They converge in an `Average` layer to allow the generation of the final prediction. Each subnetwork receives a sequence of predictions as input and predictions are extracted from Inception v3 CNN pre-trained on ImageNet dataset. The prediction is represented as a 2048 length NumPy array and it is the output of the `avg_pool` layer of the original Inception v3 model. The input sequence of predictions is different for the two subnetworks: for the first it is created from a **temporal ordered sequence of RGB frames**, for the second it is created from a **temporal ordered of Optical Flow frames**.

Again, the design choice of this experiment is related to the great success of Convolutional Neural Network in Image Classification but I considered the pioneering success of [29] as well. This leads me to analyze the Optical Flow frame in a parallel Recurrent Neural subnetwork, emulating the Two Streams CNN approach. Also for this experiment, I chose to use LSTM since, as already describe in paragraph 4.5, an LSTM unit takes care of what previously happens considering previous output and a memory state. Similarly to the previous experiment, paragraph 4.5, the output of the LSTM layer is a Dense layer, followed by a Dropout layer and another Dense layer. Then, there is an Average layer which is in charge to summarize the features from both subnetwork. In the end, a final Dense with softmax layer produce the result. The Python script of the code creation is available in the appendix A.2.

4.6.2 Preprocessing

For this experiment, I used the frames generated for the previous experiment, paragraph4.5, but, additionally, I generated the Optical Flow frames using `OpenCV2`

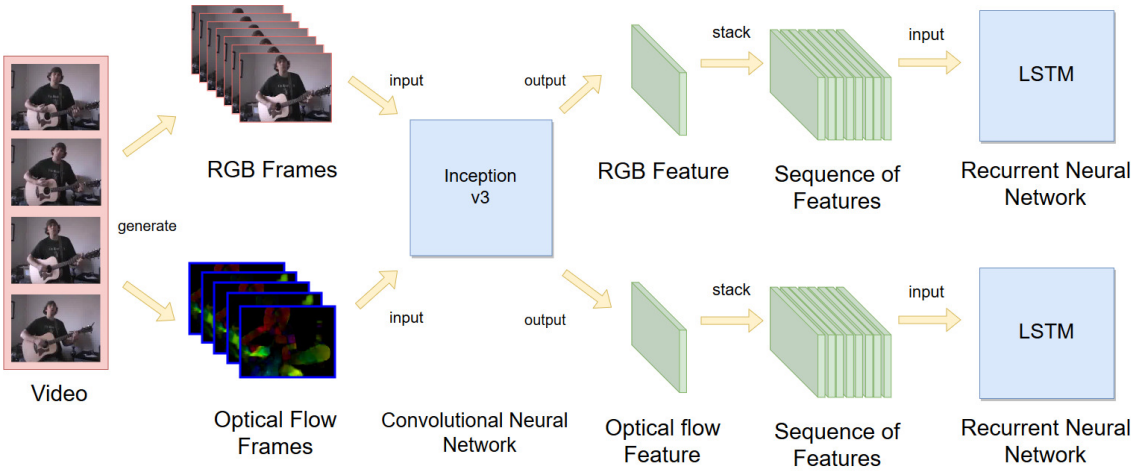

Figure 4.5. In the figure is shown the preprocess pipeline, from the video clip to LSTMs input. The Inception-v3 CNN is the official one, pre-trained on ImageNet, and it shares parameters between the two branches. The two LSTMs are different and trained for each branch.

as well. The already existing RGB frames and the Optical Flow ones are fed one by one in the Inception v3 CNN as input and a 2048-dimension array is generated with the `predict()` Keras Model method. According to the `feature_sequence_length` hyper-parameter value, N predictions, coming from N sequential frames, are stacked in order in a NumPy array and this is done twice: once for temporal using Optical Flow frames, once for spatial using RGB frames. This means that for each video are generated two NumPy arrays with `(feature_sequence_length,2048)` shape and it stored locally in the file system. Again, I use One Hot Encoding to represent each class as a binary vector.

In my implementation, it is possible to perform these operations by means of `features_extractor.py` Python script, inside the `preprocessing` module.

4.6.3 Training

I trained this model a couple of times and among the hyperparameters, I chose to change some of them and use constant values for others; specific information are available in table 4.1. Since I had to classify video clips among a specific subset, I used `categorical_crossentropy` [27] as loss function while `accuracy` and `top_k_categorical_accuracy` have been chosen as metrics. The optimizer used is `Adam` [26]. About the not-constant parameters, using the same name variable of the code snippet in 4.6.1 the `lstm_unit` value is set to 2048, while about dropout values (`a1`, `c1`, `a2`, and `c2` in the appendix code A.2), they have been set to `0.5`. For the number

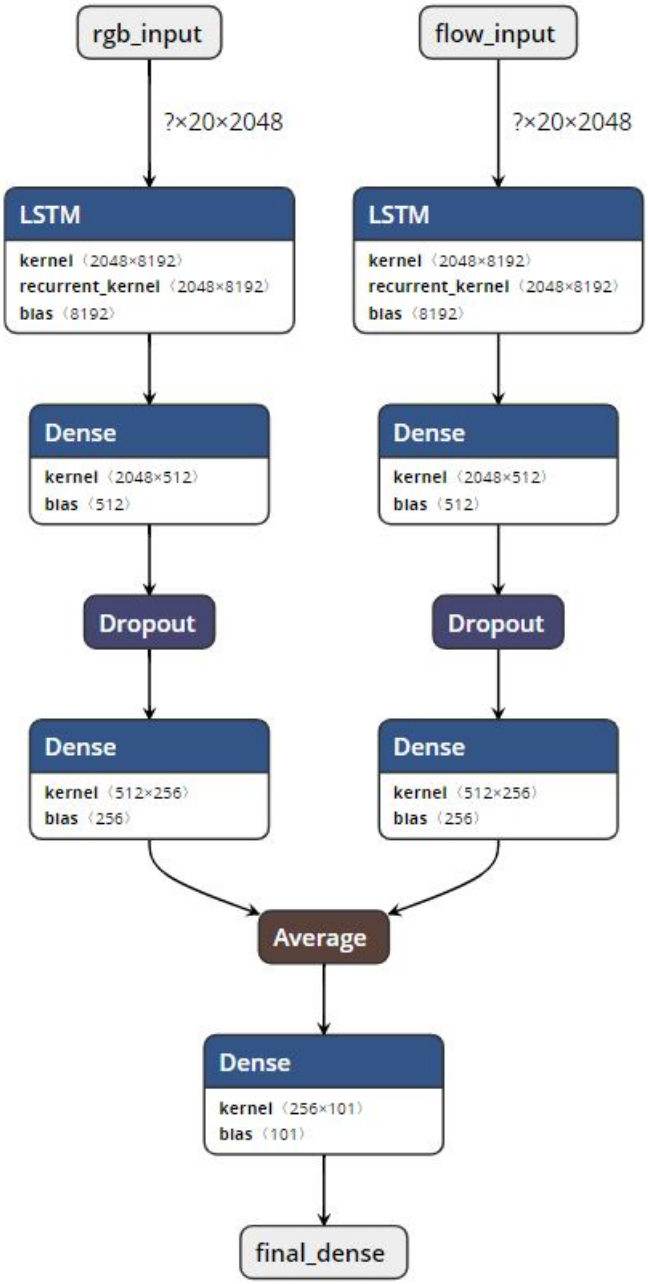

Figure 4.6. Two Streams LSTM Recurrent Neural Network

of unit in the Dense layer (parameters `b1` and `b2`) the value chosen is 512. About the optimizer, the model has been trained with different values for `learning_rate` and `decay` but in the end, they have been set to 10^{-5} and 10^{-6} respectively. The reason of these parameters value is related to the hyperparameters tuning done in the previous experiment, paragraph 4.5. Each training phase lasts about 140 minutes and then the Early Stopper callback function interrupts the training to avoid overfitting.

4.7 Temporal Segment LSTM Recurrent Neural Network

4.7.1 Design

This experiment aims to re-propose the **Temporal Segment Network Convolutional Network** with a Recurrent Neural Network. TSN networks are the current state of the art architectures for video action recognition Before going into details with the architecture, for this experiment the Inception-v3 CNN model has been re-trained on a dataset generated from the UCF101 video dataset. This model is in charge of generate single predictions for each frame, both RGB and Optical Flow. Exploiting a transfer learning approach and Keras framework capabilities, the Inception v3 CNN model, pre-trained on ImageNet dataset, has been re-trained on both RGB and Optical Flow frames from UCF101, using the UCF101 classes as label. Since the model has about 23 millions parameters, the training has be done only on the latest 250 layers, in order to maintain the discriminant abilities [30] of the first layers of this CNN. The training details are in paragraph 4.7.3.

The actual model exploits one of the major techniques described in the Temporal Segment Network paper [31]. As already described in 3.2.4, this should be able to

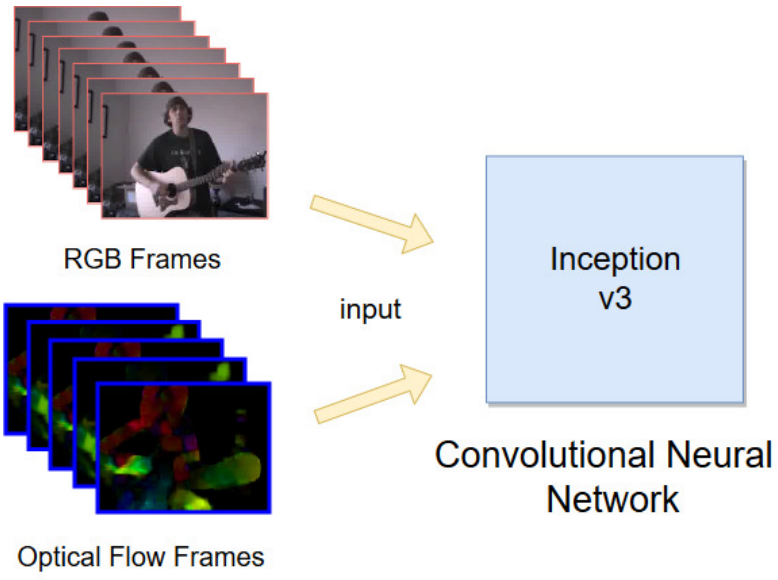

Figure 4.7. Frames and Optical flow Frames are used as input for the Inception v3 CNN in order to re-train it on UCF101 dataset

tackle two problems: the not-considering long-range temporal structure for understanding the dynamics in action videos and the size of the dataset. According to the paper, there is an important change in how input is fed in the model and how the prediction is produced. Each video clips is divided in $\boldsymbol{k}$ segments and each segment is used as input of the networks. The model keeps the same core architecture of the previous experiment, paragraph 4.6. Namely, there are two Recurrent Neural Network, that receive as input two sequence of predictions. The first sequence is generated by stacking prediction from temporally ordered RGB frames while the second sequence is generated by stacking prediction from temporally ordered Optical Flow frames. After the two parallel subnetwork there is a module in charge of generate the final prediction. The predictions of each segment of the same video clip are used as input to a third module which is able to generate a *consensus*, namely it obtains a unique prediction for a video clips, through a *consensus function* and this is done for both RGB and Optical Flow frames. In the end, the predictions for RGB subnetwork and Optical Flow one are merged together to produce the final prediction. The consensus modules, both local and global, are not a neural networks but they are functions applied on the predictions of the two subnetworks.

4.7.2 Preprocessing

For this experiment, I used the RGB frames generated for the previous experiment, both paragraph 4.5 and 4.6. Than, for the Optical Flow frames I evaluated two cases: the frames generation with Farneback Optical Flow Estimation by means of *OpenCV2* and the one with LiteFlowNet [22] by means of a PyTorch implementation [23] of LiteFlowNet. As we will in the result paragraphs 4.7.3 and table 5.1, the usage of a set of frames or of another set of frames will produce very different result. In both cases, once the frames have been correctly created, the Inception-v3 CNN has been trained. According to Keras official guidelines [13], the first layers of the Inception-v3 model should not be trained in order to preserve the basic knowledge of the network about basic structure like edge, shape, color and others, pre-trained on ImageNet dataset. I follow these specifications for CNN re-training and the first 250 layers of the Inception-v3 model have been frozen. The training lasted 15 epochs for about 25 hours per epoch.

The video clips are divided in $\boldsymbol{k}$ segments and from each segment I extracted

`features_sequence_length` series of frames. The frames, as in the previous experiments, paragraphs 4.5 and 4.6, are fed one by one in the Inception v3 CNN as input and the model generates a 2048-dimension array, for each frame, as output. Accordingly with the `feature_sequence_length` $\boldsymbol{N}$ value, $\boldsymbol{N}$ predictions, coming from the $\boldsymbol{N}$ sequential frames of each segment, are stacked in order in a NumPy array and this is done twice: once for temporal using Optical Flow frames, once for spatial using RGB frames. This means that for each segment of each video two NumPy arrays with `(feature_sequence_length,2048)` shape are generated and stored locally in the file system.

4.7.3 Training

First of all, I re-trained the Inception-v3 and I have done this operation twice because, according to what described in paragraph 4.3, the training has been done twice: once with the Optical Flow frames estimated with Farneback technique, once with the Optical Flow frames estimated with LiteFlowNet Convolutional Neural Network. Starting from the Inception-v3 model pre-trained on ImageNet, the training consists in two phases: in the first phase all the layers of the network are trained; in the second phase the first 250 layers are frozen and only the others are trained.

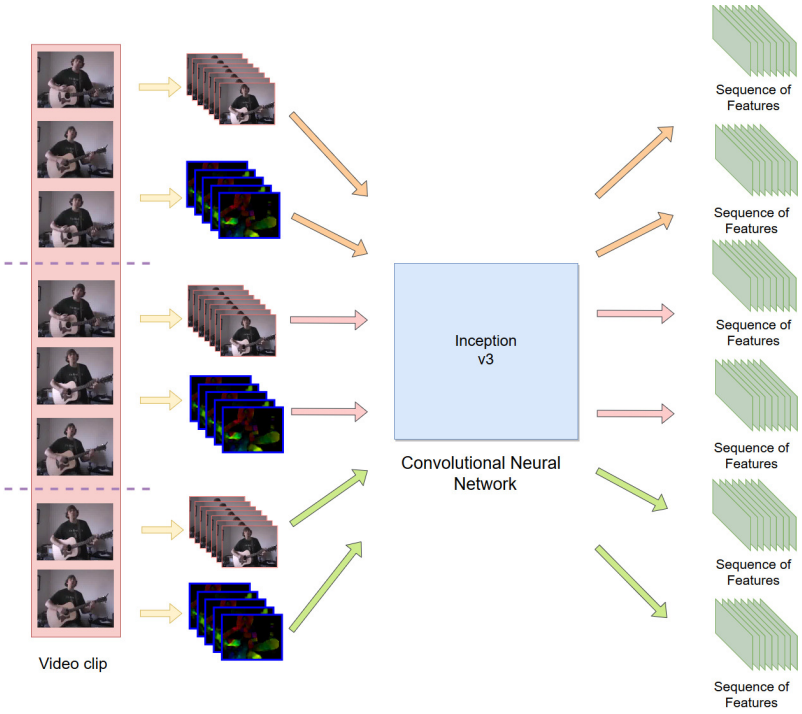

Figure 4.8. Preprocessing pipeline of Temporal Segment Network LSTM RNN

For both phases the optimizer chosen is `Adam` with *learning rate* value equal to 10^{-5} while the *decay* value is 10^{-6}. The loss evaluated is `categorical_crossentropy` where the labels selected changes from the 1000 possible value of ImageNet to the 101 possible action of UCF101.

According to this new architecture, each LSTM subnetwork is able to predict the video clip action given a segment of a video clip. In order to achieve this goal, the LSTM subnetworks are trained with the sequence of features extracted from each segment using the re-trained Inception v3 CNN. The output of these subnetworks is used as input for the local consensus module and then for the global consensus module. The training and the validation perform different operations: the model receives as input a segment of the video clip for the training phase while it receives all the segments of a video clip for the validation phase. This leads to a different implementation of the two phases and **a custom validation callback function** is used to generate the correct accuracy value.

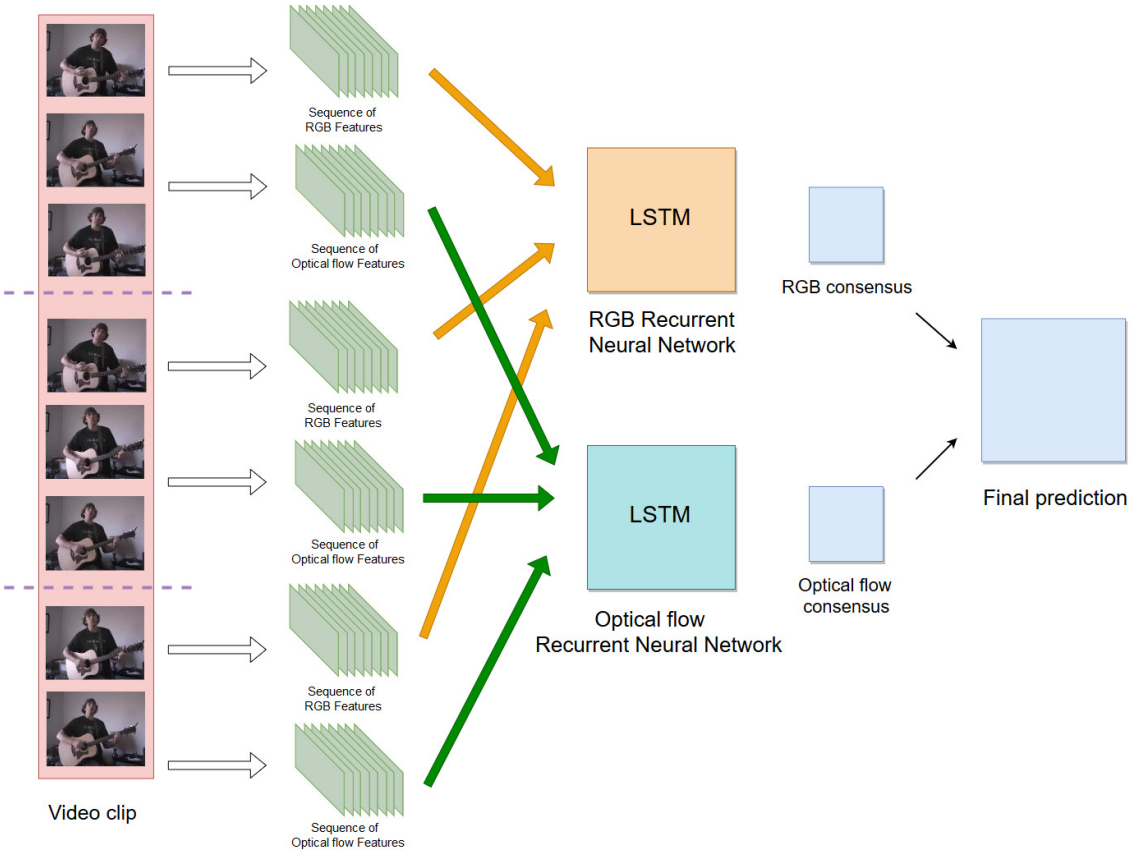

Figure 4.9. Training diagram for Temporal Segment Network LSTM RNN

Experiment	**Optimizer**	**Metrics**	**Loss**	**Dropout**
Single Stream LSTM	Adam lr = 10^{-4} decay = 5*10^{-6}	accuracy, top 5 categorical accuracy	categorical crossentropy	0.5
Two Streams LSTM	Adam lr = 10^{-5} decay = 10^{-6}	accuracy, top 5 categorical accuracy	categorical crossentropy	0.5
TSN LSTM	Adam lr = 10^{-5} decay = 10^{-6}	accuracy, top 5 categorical accuracy	categorical crossentropy	0.5

Table 4.1. Optimizer, hyper-parameters, metrics and loss function used in the experiments performed

Chapter 5

Experimental Results

5.1 Experimental Results

In this chapter the results of the experiments done are presented. For the Single Stream LSTM and the Two Streams LSTM experiments, the assessment consists in the evaluation of the validation set accuracy of the first official split of UCF-101 action recognition dataset. For the Temporal Segment LSTM experiment, the assessment consists in training the model over the three official splits of UCF-101 and evaluating the validation set accuracy mean value across the three splits. For each experiment, it is present a plot of the metrics analyzed, namely **Accuracy**, **top 5 Categorical Accuracy** and **Loss**.

5.1.1 Single Stream LSTM Recurrent Neural Network

This experiment has been evaluated on the first official split of UCF-101 dataset. It aimed to reach and overtake the Single Stream architecture. In the original experiments performed on Single Stream Architecture [37], the model has been trained on Sports-1M Dataset (described in 3.1.1) and then the top layers have been re-trained on the UCF-101 Action Recognition dataset obtaining 63,3% accuracy. My model obtained a better accuracy compared to the original model one; the accuracy over the validation set of the first split reached 72,5%. The result has been plotted in figures 5.1 and 5.2. The Top 5 categorical accuracy reached a value of 91,8%.

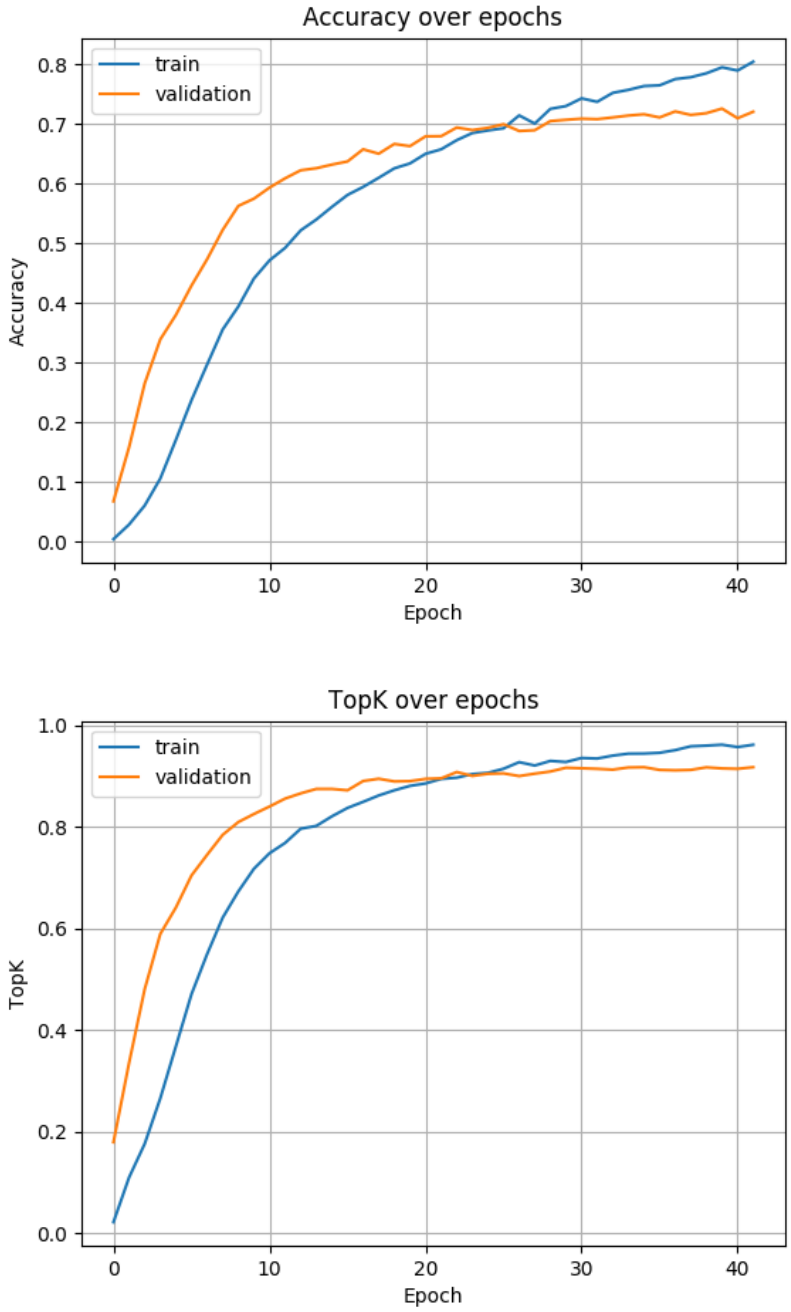

Figure 5.1. Accuracy and Top 5 Accuracy over epochs of Single Stream LSTM

5.1.2 Two Streams LSTM Recurrent Neural Network

As the previous case, in paragraph 5.1.1, the experiment has been evaluated on the first official split of UCF-101 dataset. In this case, I tried to reach the original Two Stream architecture result [29] but the results are lower in terms of accuracy. Namely, the model proposed by Simonyan and Zisserman reached an accuracy of

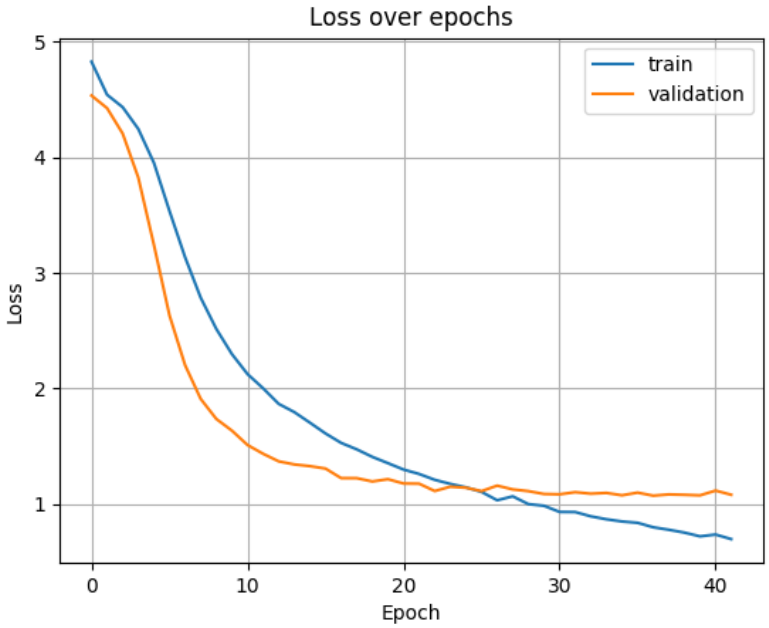

Figure 5.2. Loss over epochs of Single Stream LSTM

85,9% on the first UCF-101 split while my solution reached a value of 76,8%. Among the possible reasons of this result there are: the different values of the parameters and the different technique used to estimate the optical flow frames. The result has been plotted in figures 5.3 and 5.4. The Top 5 categorical accuracy reached a value of 94,0%.

5.1.3 Temporal Segment LSTM Recurrent Neural Network

In this paragraph two sets of results are described: first, the results related to the training done with Optical Flow frames extracted with *Farneback Optical Flow estimation* and, second, the results related to the training done with *LiteFlowNet Convolutional Neural Network*. With this experiment, I tried to reach the original Temporal Segment Networks architecture results [31] but re-proposing the architecture by introducing two LSTM subnetworks. The model proposed by Wang and Xiong reached an accuracy of 94%, averaging the validation accuracy values over the three official splits.

Using the **Farneback Optical Flow estimation**, the results are lower in terms of accuracy. Namely, the model reached a value of 76% on the split 1. Using the **LiteFlowNet Convolutional Neural Network**, the results reached better values

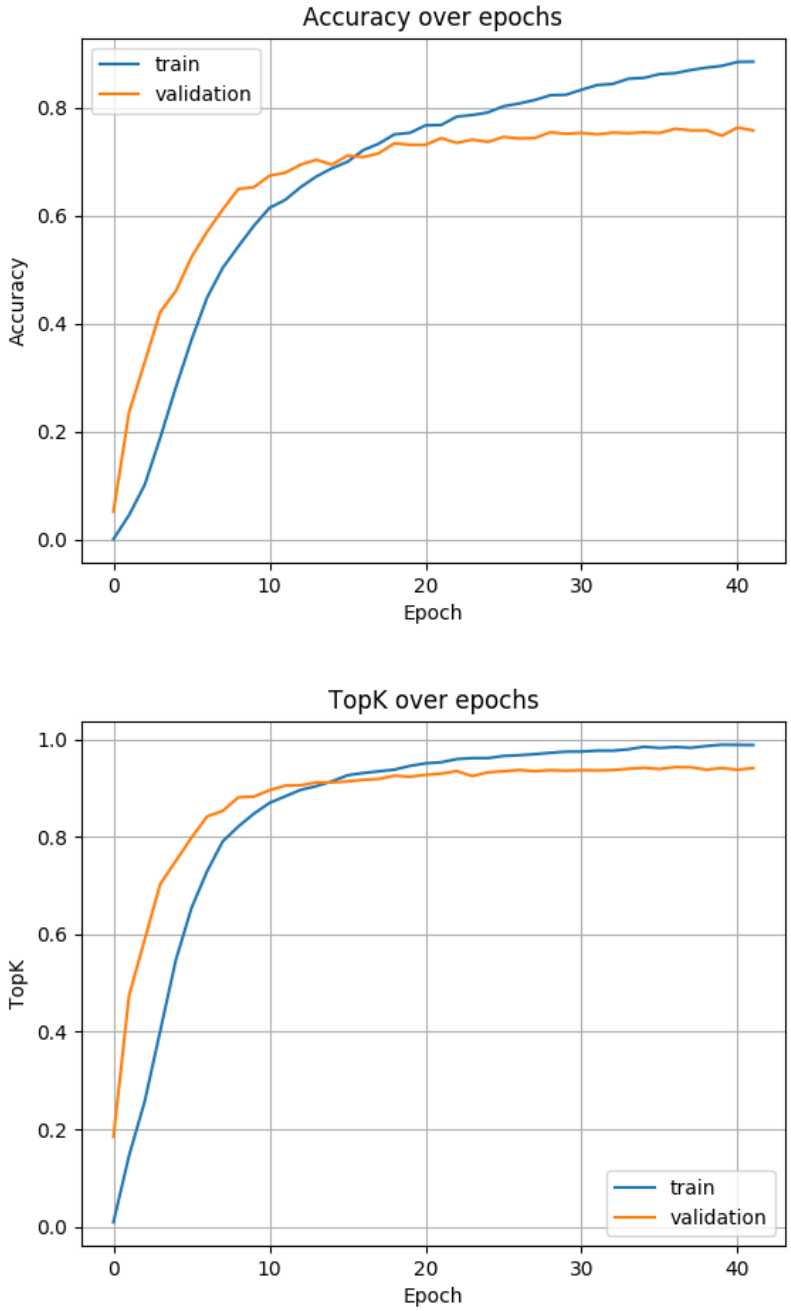

Figure 5.3. Accuracy and Top 5 Accuracy over epochs of Two Stream LSTM

compared to the previous training but it is slightly lower than Temporal Segment Network [31]. Namely, the model reached an accuracy of 81% averaged over the 3 official split.

The enhancement from 76% to 81% is given by the change of the technique of Optical Flow frames generation. The Pytorch implementation [23] of LiteFlowNet generate **Warped Optical Flow** and this allows a better accuracy in the Optical

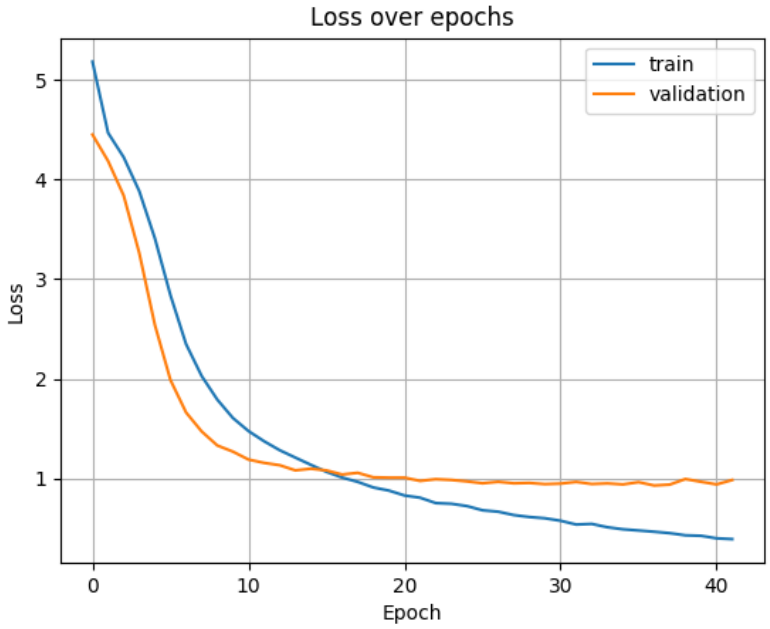

Figure 5.4. Loss over epochs of Two Stream LSTM

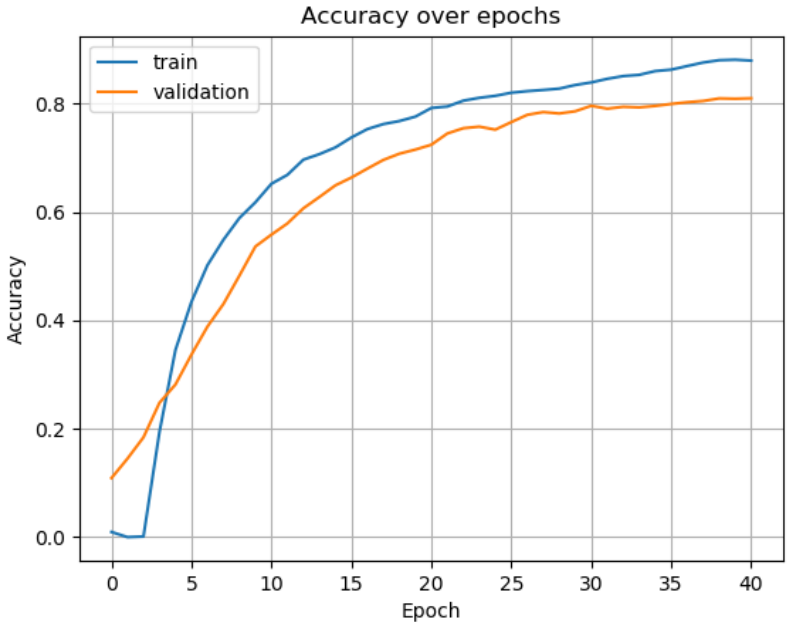

Figure 5.5. Accuracy over epochs of Tsn LSTM

Flow subnetwork of the model. With Farneback Optical Flow estimation, the accuracy of the Optical Flow subnetwork is 50%, while with LiteFlowNet the same accuracy is 73%, allowing to reach better result. The result with Farneback Optical Flow estimation is poor and it reduces the accuracy of the entire network but this

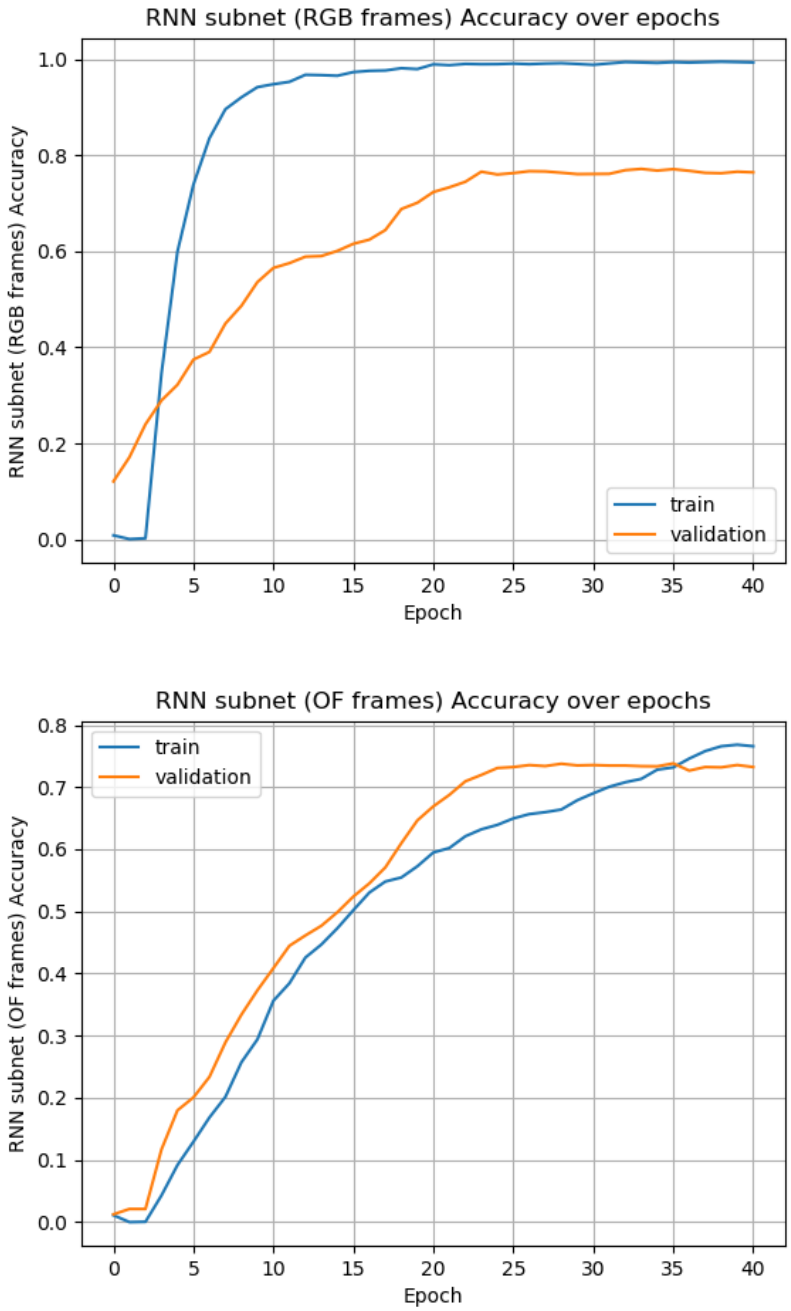

Figure 5.6. Tsn LSTM Accuracy of RNN subnet for RGB and Optical Flow frames over epochs

result were awaited since the model described in the TSN research paper [31] uses a *Warped Optical Flow* to reach those results.

Methods	Optical Flow	UCF101 acc
Single Stream Net [37]	No	0.63 (*)
Two Stream Net [29]	Brox	0.86
TSN Conv Net [31]	Warped	0.94
Ts LSTM Net [43]	Warped	0.94
Single Stream LSTM	No	0.72 (*)
Two Stream LSTM	Farnerback	0.76 (*)
Tsn LSTM Rnn	Farnerback	0.77 (*)
Tsn LSTM Rnn	Warped	0.81 (0.805)

Table 5.1. Accuracy and results comparison for UCF101 dataset. If the star is present, the training has been performed only on the first official split of UCF101 dataset. If the result is provided in brackets, it is related to a training done on the first official split.

5.2 Results Evaluations

5.2.1 Experiments Discussion

In this section I will comment the results obtained in the three experiments performed.

The **Single Stream LSTM** obtained an accuracy value greater of the Single Stream CNN, on the first official split of UCF101 dataset, namely 0.72 for the LSTM and 0.63 for the CNN. This result depends on the introduction of the Recurrent Neural Network.

The introduction of the analysis of the temporal behaviour with a parallel Optical Flow, as expected, increase the accuracy value in the **Two Stream LSTM** architecture from 0.72 to 0.76 confirming that the temporal behaviour of a video clip has a core role in the action classification. However, this increase is lower than expected since the Two Stream CNN reached an accuracy value equals to 0.86. I have analyzed the possible reasons of this result and they could be related to the architecture of the model. Namely, the Two Stream CNN has two different Convolutional Networks to predict separately the RGB and Optical Flow frames; in the end, the final prediction is chosen with a final score function. The Two Stream LSTM has one Convolutional Network and that provides input features for two different Recurrent Neural Network. It seems that the usage of only one Convolutional Network limits the accuracy of the architecture, probably because the network it is not efficient if it has to recognize two different types of frames (RGB and OF).

In the third experiment, I extended the Two Stream LSTM with the techniques introduced by the Temporal Segment CNN, obtaining a **Temporal Segment LSTM**. The usage of this new approach increase the accuracy of the model from 0.76 to 0.81 confirming the great improvement that the usage of Temporal Segment guarantees. However, the Temporal Segment CNN reached an accuracy value of 0.94. Comparing the 0.81 accuracy result with the state of art accuracy, Ma *et al.* in [43] proposed an architecture that introduced a Recurrent Neural Network by means of an LSTM layer. Their model reached an accuracy of 0.94 on UCF101 dataset, matching the Temporal Segment Network but with less computation. Among the possible reasons of the difference of my results and the results in [43] there are: the different model and the different values of the hyper-parameters. In details, their model architecture, figure 3.4, has two different Convolutional Networks (ResNet-101) for the features extraction and a single Recurrent Neural Network for the final prediction where the input is a concatenation of the two features extracted from the CNNs. My model architecture, figure 4.9, has one Convolutional Network (Inception-v3) for the features extraction and two parallel Recurrent Network for the final prediction where the input of each subnetwork is the prediction of an RGB frame or an Optical Flow frame from the CNN. It seems that this difference does not allow to completely exploit the temporal features and this could explain the results difference.

One of the great advantages of the usage of Temporal Segment Network is that it allows the usage of this architecture in a **real-time application**, as described in paragraph 6. This is related to the division in segment of each video clip; indeed, since the input of the architecture is a segment and not the entire video, it is possible to adapt it to receive video segment in real-time.

5.2.2 Classes Accuracy Evaluation

I have created the **confusion matrix** of the third experiment in order to understand which classes have a bad accuracy value and the reasons behind the value. In table 5.2, there are the classes with the **best accuracy values** obtained with a trained Temporal Segment LSTM model. The analysis has been performed on the result of the first official split of UCF-101 dataset. Some example of the classes are available in figure 5.7. Then, in table 5.3, there are the classes with the **worst accuracy values** obtained with a trained Temporal Segment LSTM model. In this case as well, the analysis has been performed on the result of the first official split

of UCF-101 dataset. Comparing the results in the tables and the samples in figures 5.7 and 5.8 it is possible to notice the following things:

- The classes that are quite *unique*, e.g. Mixing, UnevenBars, PlayingDaf or SkyDiving, obtain a great accuracy result
- The classes that are composed by video clips with **similar movements, background or shape, obtain a low level of accuracy**. For example, the model has some difficulties in classifying ApplyMakeUp, ApplyLipstick, BrushingTeeth and ShavingBeard and often it confuses video clips belonging to this classes because they are really similar and the main pattern is the generic human face with an hand on it. The same happens with the classes JavelinThrow, LongJump, FloorGymnastics and HandstandingWalk where the main pattern is the human body running in some directions.

Classes	**Accuracy**
Mixing	1.00
SkyDiving	1.00
PlayingDaf	1.00
WalkingWithDog	1.00
VolleyballSpiking	1.00
UnevenBars	1.00
Typing	1.00
TrampolineJumping	1.00
PlayingSitar	1.00
Billiards	1.00
Punch	0.97
Surfing	0.97

Table 5.2. In this table are represented the UCF101 classes that have the highest accuracy on the validation set of the first official split using Temporal Segment LSTM architecture, described in paragraph 5.1.3

Classes	Accuracy	Often confused with:
JavelinThrow	0.18	HighJump, FloorGymnstics
MoppingFloor	0.29	HandstandWalking, Archery, FloorGymnstics
HandsandWalking	0.29	BodyWeightSquats, Basketball, BabyCrawling
Nunchucks	0.29	BodyWeightSquats, GolfSwing, FrisbeeCatch
PizzaTossing	0.30	Nunchucks, JugglingBalls, HammerThrow
BrushingTeeth	0.35	ShavingBeard, ApplyEyeMakeUp, ApplyLipstick
CricketBowling	0.38	FloorGymnastic, CrickeShot, CliffDiving

Table 5.3. In this table are represented the UCF101 classes that have the lowest accuracy on the validation set of the first official split using Temporal Segment LSTM architecture, described in paragraph 5.1.3

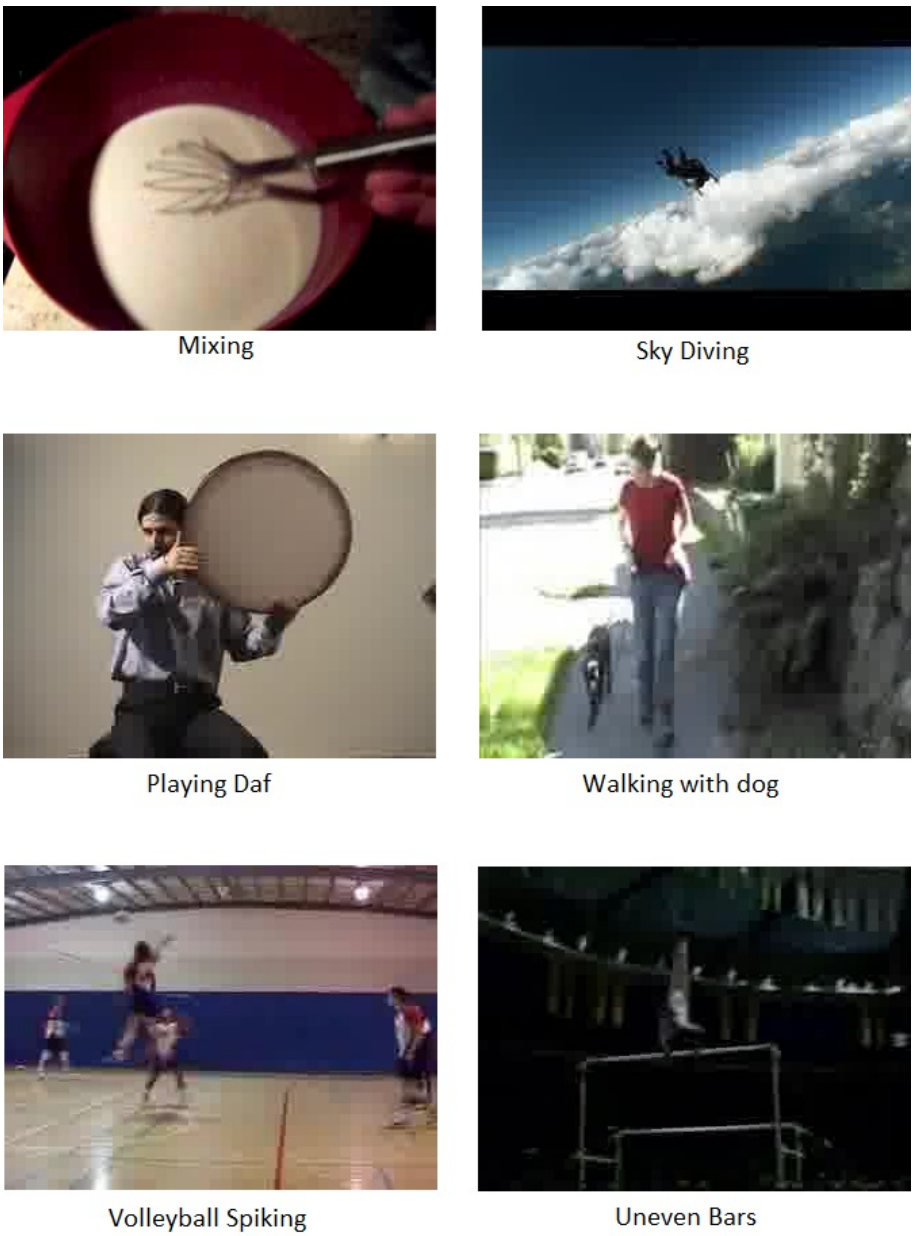

Figure 5.7. In these figures are shown samples from video clips of classes with the highest accuracy value with Temporal Segment LSTM after training.

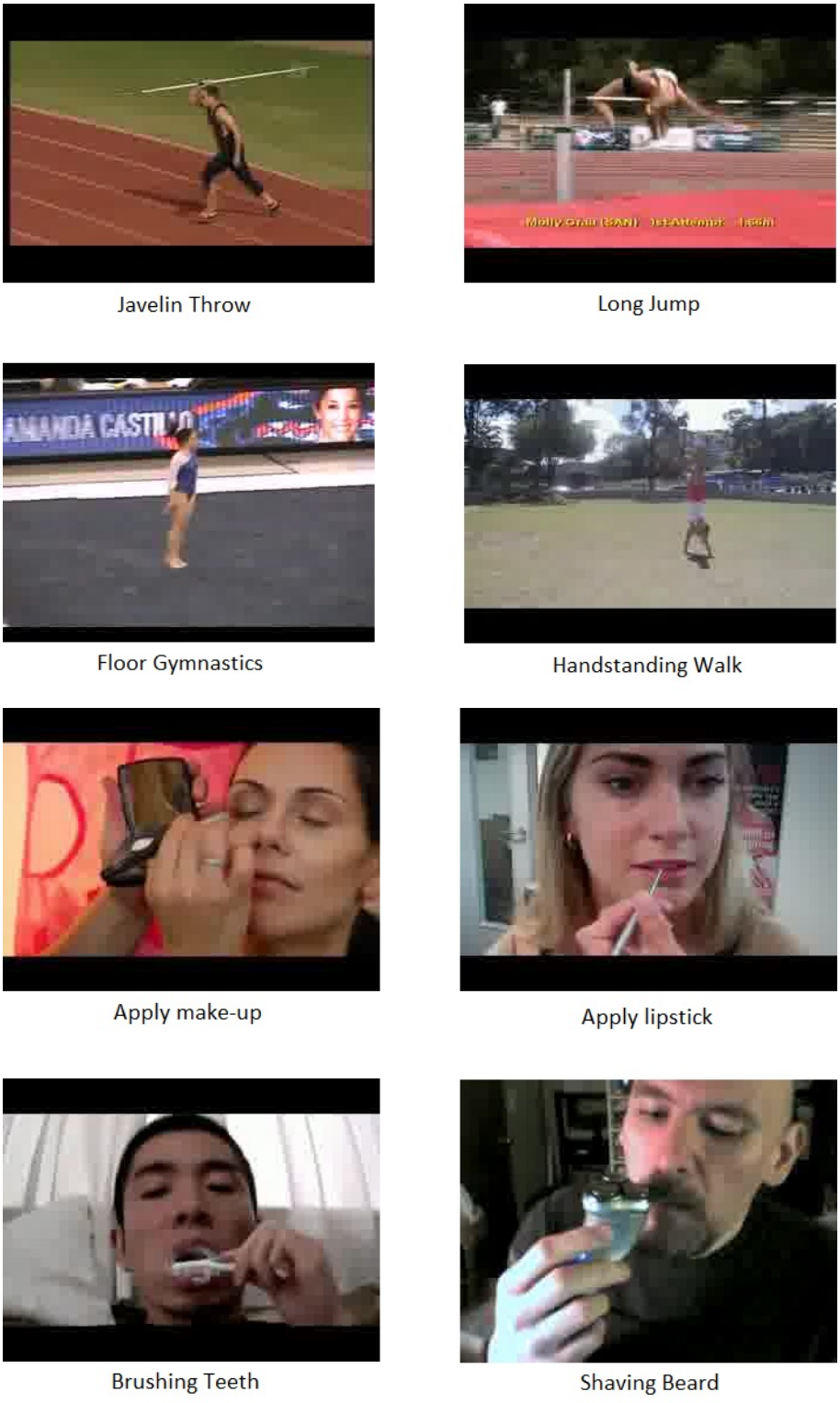

Figure 5.8. In this figures, it is possible to analyze samples from video clips of classes with the lowest accuracy value with Temporal Segment LSTM after training. In details, the first four classes are often confused within each other because they are similar and the main pattern is the human body running in some directions. Similarly, the last four classes are often confused within each other because the main pattern is the generic human face with an hand on it.

Chapter 6

Real-time Application

6.1 Introduction

In this chapter, I will describe a real-time application that uses the model that obtains the highest accuracy during my experiments, described in chapter 5. The main difficulties found during this implementation are related to:

- A common input is usually a continuous stream of frames and not a trimmed video clip with a specific action.
- The flow of frames is untrimmed and it is unknown when an action will start within it.
- The absence of standard methodologies and metrics to evaluate the application.
- The absence of a set of valid input for this use case.

6.2 Analysis and Design

As described in paragraph 6.1, one of the difficulties found is related to the fact that the input is generally a continuous stream of frames. In paragraph 4.7, I described my implementation of Temporal Segment Network with two LSTM subnetworks. The input of this model is a sequence of feature, namely an array of

predictions extracted with a re-trained Inception v3 CNN. This could be helpful because we can divided the continuous stream in a sequence of segments, where each one is a sequence of frames, and then, use it as input for the model.

Therefore, it is necessary to select a **time interval *t*** that will represent a segment. Within this interval, we have to sample a sequence of frames and extract the prediction from each one with the Inception v3 CNN. Moreover, we have to generate the optical flow frames and use them to extract the sequence of predictions with Inception v3 CNN. Now we have the minimal input to obtain an action prediction. This process is iterated over all the stream of frames.

Another difficulty found is that we do not know when the frames related to an action will start in the stream. In order to better estimate the presence of a specific action, I supposed to evaluate multiple and parallel segments with a ***d* displacement** while processing the stream, with $d < t$. The general approach is represented in figure 6.1.

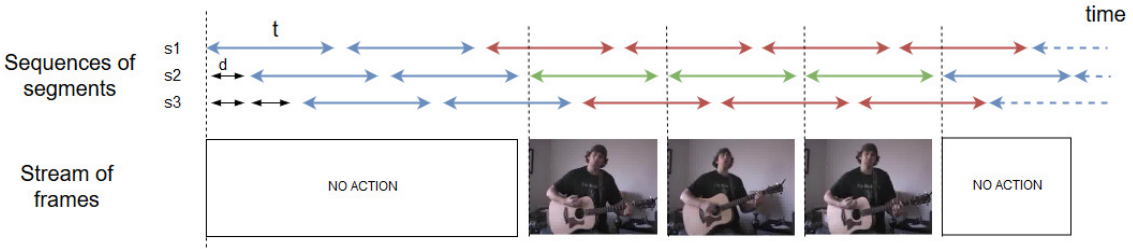

Figure 6.1. Below, the stream of frames is represented. In the stream there could be some frames that are not associated with any actions. On top, there is the splitting of the stream in consecutive segments. There are some parallel segments with a ***d*** displacement to better hit the segments that contain an action.

6.3 Software Implementation

In this paragraph, I describe the software architecture of the real-time application. The input of the application is an **untrimmed video clip**; the output is a **CSV file** where it is detailed the starting and ending seconds and the prediction accuracy of the classification, for each action found in the input video clip. The application is composed by modules, each one with specific task, and it is represented in figure 6.2.

- **Frame Extractor**

  The Frame Extractor is a module in charge of generating both RGB and

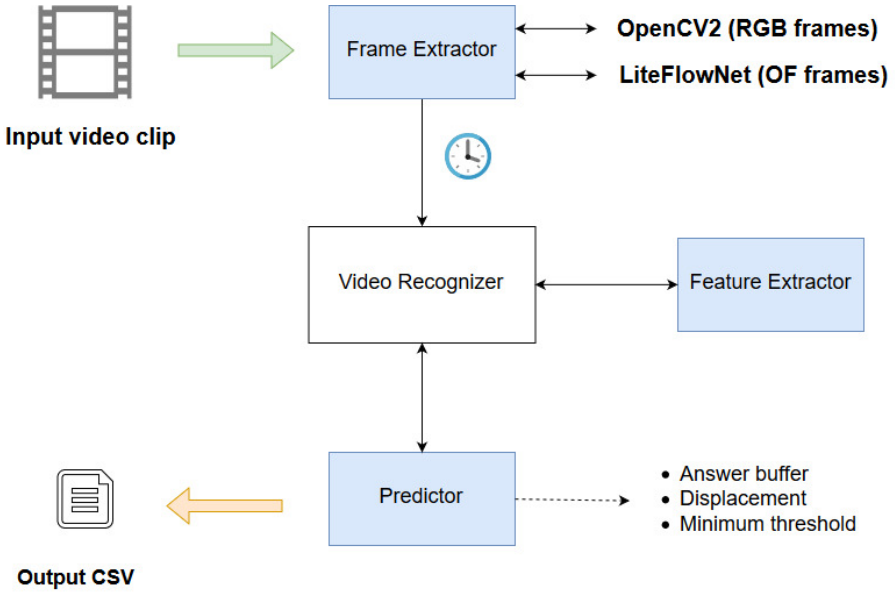

Figure 6.2. Real-time Application Software Architecture

Optical Flow frames for the Video Recognizer. It is the first module used by the application and it maintains an internal state with the information related to the timing of the video, the segment size, the fps and the displacement. Practically, it has a method that create a Python Generator which is able to yield a new set of frames belonging to a new segment. This means that the application is able to work with the segments of input video clip and not with the entire video, allowing the generation of a prediction without reading all the frames. The frames are generated by means of OpenCV2 (RGB frames) and a PyTorch representation [23] of LiteFlowNet (Optical frames).

- **Video Recognizer**
  The Video Recognizer is a core module in charge of connecting the modules. It requires new frames from Frame Extractor, sends them to the Feature Extractor that returns some new features, receives the features and send them the Predictor which analyze the input and generate the result.

- **Feature Extractor**
  The Feature Extractor is a module in charge of generating a sequence of predictions starting from a sequence of frames. It receives the frames generated in by the Frame Extractor and, by means of a trained Inception-v3 Convolutional Network, it generates a prediction for each frames and stack them togheter in a temporal ordered sequence of predictions.

- **Predictor**

  The Predictor is a module in charge of generate the final prediction. It receives in input the sequences of predictions generated by the Feature Extractor and produce a final prediction. Reading the current timing and video information, it is able to write the result in the output CSV. Inside the predictor there is an *Answer Buffer*; it is a mechanism that allow to consider as input also the previous two segments and not only the current one. The Predictor considers a prediction as a recognized action if it exceeds a *Minimum Threshold*.

Chapter 7

Conclusions

This work presents and describes some possible architectures able to recognize a video action by means of the usage of Recurrent Neural Networks. For this purpose, we have designed and implemented three different architectures starting from the state-of-art ones. The Temporal Segment LSTM is a supervised machine learning model which receives, as input, the predictions extracted from some video clip segments through a Convolutional Network and uses them to predict the human action class by means of Recurrent Network, ensured by the presence of a Long-Short Term Memory layer. There are some architecture choices that could have afflict the accuracy value of the results, not allowing to fully exploit the presence of Optical Flow frames but despite this, the architectures obtained accuracy values that are comparable with the state-of-art ones, highlighting the potential of the usage the Recurrent Networks in the video classification.

7.1 Future works

At the end of my work, there are some possible future works that is possible to perform. Due to the lack of time, we had to compromise on the number of different experiments. In paragraph 5.2, I have written some possible reasons for the results obtained and, starting from there, it is possible to follow up this work trying to achieve better accuracy with **some changes on the network architecture**. Moreover, I chose to use UCF101 Action Recognition Dataset for my experiments for the reasons explained in paragraph 4.2 but it could be possible to perform some new

experiments with the same architecture but on a **different dataset** like Kinetics-600 [53] or YouTube-8M [46], where in general it will require more time or more computational power. It is even possible to perform a **transfer learning** experiment to evaluate if the discriminatory abilities of the network can be used on other datasets or for other tasks. Finally, it is possible to **evaluate the real-time application** against the THUMOS Challenge 2014 [57], a common challenge used to evaluate action classification model with temporally untrimmed video clips.

Appendix A

Neural Networks Model

A.1 Single Stream LSTM Recurrent Neural Network

The following lines of code generate a LSTM networks:

```
1    model = Sequential()
2    model.add(LSTM(lstm_unit, return_sequences=False,
        input_shape=input_shape, dropout=a))
3    model.add(Dense(b, activation='relu'))
4    model.add(Dropout(c))
5    model.add(Dense(classes_size, activation='softmax'))
```

A.2 Two Streams LSTM Recurrent Neural Network

The following lines of code generate a RNN networks composed by two LSTM subnetworks:

```
1    rgb_input = Input(shape=input_shape, name='rgb_input')
2    rgb_lstm = LSTM(lstm_unit, return_sequences=False, dropout=a1,
        name='rgb_lstm')(rgb_input)
3    rgb_dense1 = Dense(b1, name='rgb_dense1')(rgb_lstm)
4    rgb_dropout = Dropout(c1, name='rgb_dropout')(rgb_dense1)
```

```
5     rgb_dense2 = Dense(d1, name='rgb_dense2')(rgb_dropout)
6
7     flow_input = Input(shape=input_shape, name='flow_input')
8     flow_lstm = LSTM(lstm_unit, return_sequences=False,
          dropout=a2, name='flow_lstm')(flow_input)
9     flow_dense1 = Dense(b2, name='flow_dense1')(flow_lstm)
10    flow_dropout = Dropout(c2, name='flow_dropout')(flow_dense1)
11    flow_dense2 = Dense(d2, name='flow_dense2')(flow_dropout)
12
13    avg = average([rgb_dense2, flow_dense2])
14    final_dense = Dense(classes_size, activation='softmax',
          name='final_dense')(avg)
15
16    model = Model(inputs=[rgb_input, flow_input],
          outputs=final_dense)
```

A.3 Temporal Segment LSTM Recurrent Neural Network

The following lines of code generate the RNN used for the third experiment:

```
1     rgb_input = Input(shape=input_shape, name='rgb_input')
2     rgb_lstm = LSTM(2560, return_sequences=False, dropout=0.5,
          name='rgb_lstm')(rgb_input)
3     rgb_dense1 = Dense(512, name='rgb_dense1')(rgb_lstm)
4     rgb_dropout = Dropout(0.5, name='rgb_dropout')(rgb_dense1)
5     rgb_dense_final = Dense(classes_size, activation='softmax',
          name='rgb_dense_final')(rgb_dropout)
6
7     flow_input = Input(shape=input_shape, name='flow_input')
8     flow_lstm = LSTM(2560, return_sequences=False, dropout=0.5,
          name='flow_lstm')(flow_input)
9     flow_dense1 = Dense(512, name='flow_dense1')(flow_lstm)
10    flow_dropout = Dropout(0.5, name='flow_dropout')(flow_dense1)
```

```
11      flow_dense_final = Dense(classes_size, activation='softmax',
            name='flow_dense_final')(flow_dropout)
12
13      model = Model(inputs=[rgb_input, flow_input],
            outputs=[rgb_dense_final, flow_dense_final])
14
15      # Hyper parameters
16      loss = 'categorical_crossentropy'
17      optimizer = Adam(lr=1e-5, decay=1e-6)
18      metrics = ['accuracy', 'top_k_categorical_accuracy']
```

Bibliography

[1] Tom M. Mitchell, "The Discipline of Machine Learning", July 2006, CMU-ML-06-108, http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf

[2] Y.LeCun, Y.Bengio, G.Hinton, "Deep Learning", Nature, Vol. 521, 28 May 2015, pp. 436, https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf, DOI 10.1038/nature14539

[3] A. Moore, Journal Article, https://www.forbes.com/sites/peterhigh/2017/10/30/carnegie-mellon-dean-of-computer-science-on-the-\future-of-ai

[4] "CS231 in Convolutional Neural Networks for Visual Recognition", http://cs231n.github.io/

[5] "Loss function and Optimization - CS231 in Convolutional Neural Networks for Visual Recognition", http://cs231n.github.io/optimization-1/

[6] "Convolutional Neural Network - CS231 in Convolutional Neural Networks for Visual Recognition", http://cs231n.github.io/convolutional-networks/

[7] "Long short - term memory", S.Hochreiter, J.Schmidhuber, "Neural Computation", 9 (8), 1997, https://www.bioinf.jku.at/publications/older/2604.pdf

[8] S. Yan, "Understanding LSTM and its diagrams", https://github.com/shi-yan/FreeWill/blob/master/Docs/Diagrams/lstm_diagram.pptx

[9] Arden Dertat, "Applied Deep Learning",
Towardsdatascience Internet Webpage,
https://towardsdatascience.com/applied-deep-learning-part-4-\convolutional-neural-networks-584bc134c1e2

[10] S.H.Tsang, "Inception-v3 Review", Medium Article Page, (2018),
https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-\image-classification-in-ilsvrc-2015-17915421f77c

[11] Jeremy Jordan, "Setting the learning rate of your neural network", personal

website, (2018) https://www.jeremyjordan.me/nn-learning-rate/

[12] Keras: The Python Deep Learning library, https://keras.io/

[13] Keras: Fine-tuning of Inception v3 https://keras.io/applications/#fine-tune-inceptionv3-on-a-new-set-of-classes

[14] A. Filannino, "Human Action Classification", https://github.com/afilannino/video-recognition-ML

[15] UCF, Center for Research in Computer Vision, "UCF101 - Action Recognition Data Set", http://crcv.ucf.edu/data/UCF101.php

[16] FFmpeg, web page project, https://www.ffmpeg.org/

[17] G.Varol, I.Laptev, C.Schmid, "Long-term Temporal Convolutinos for Action Recognition", arXiv:1064.04494, 2017, https://arxiv.org/abs/1604.04494

[18] G.Farneback, "Two-Frame Motion Estimation Based on Polynomial Expansion", Computer Vision Laboratory, Linkoping University, In: Image analysis. 2749. 363-370, DOI 10.1007/3-540-45103-X_50

[19] OpenCV, Open Source Computer Vision, "Optical Flow, Lucas Kanade approach", https://docs.opencv.org/3.4/d7/d8b/tutorial_py_lucas_kanade.html

[20] OpenCV, Open Source Computer Vision, https://opencv.org/

[21] D.J.Fleet, Y.Weiss, "Optical Flow Estimation", In Paragios, et al., Handbook of Mathematical Models in Computer Vision, 2007

[22] T.W.Hui, X.Tang, C.C.Loy, "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation", CUHK-SenseTime Joint Lab, The Chinese University of Hong Kong, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2018, http://mmlab.ie.cuhk.edu.hk/projects/LiteFlowNet/

[23] Simon Niklaus, "A PyTorch implementation of LiteFlowNet", https://github.com/sniklaus/pytorch-liteflownet

[24] Keras: The Python Deep Learning library, Available models, https://keras.io/applications/#available-models

[25] W.Kay, J.Carreira, K.Simonyan, B.Zhang, C.Hillier, S.Vijanarasimhan, F.Viola, T.Green, T.Back, P.Natsev, M.Suleyman, A.Zisserman, arXiv:1705.06950, (2017), https://arxiv.org/abs/1705.06950

[26] D.P.Kingma, J.Ba, "Adam: A Method for Stochastic Optimization", arXiv:1412.6980, (2014), https://arxiv.org/abs/1412.6980

[27] S.Mannor, D.Peleg, R.Rubinstein, "The cross entropy method for classification", DOI 10.1.1.324.4845

[28] A.Krizhevsky, I.Sutskever, G.E.Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", University of Toronto `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-\convolutional-neural-networks.pdf`

[29] K.Simonyan, A.Zisserman, "Two-Stream Convolutional Networks for Action Recognition in Videos", arXiv:1406.2199, (2014), `https://arxiv.org/abs/1406.2199`

[30] C.Szegedy, V.Vanhoucke, S.Ioffe, J.Shlens, Z.Wojna, "Rethinking the Inception Architecture for Computer Vision", arXiv:1512.00567, (2016), `https://arxiv.org/abs/1512.00567`

[31] L.Wang, Y.Xiong, Z.Wang, Y.Qiao, D.Lin, X.Tang, L.Van Gool, "Temporal Segment Networks: Towards Good Practices for Deep Action Recognition", arXiv:1608.00859, (2017), `https://arxiv.org/abs/1608.00859`

[32] Ng Yue, M.Hausknecht, S.Vijayanarasimhan, O.Vinyals, R.Monga, G.Toderici, "Beyond Short Snippets: Deep Networks for Video Classification", arXiv:1503.08909, (2015), `https://arxiv.org/abs/1503.08909`

[33] J.Donahue, L.A.Hendricks, M.Rohrbach, S.Venugopalan, S.Guadarrama, K.Saenko, T.Darrell, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", arXiv:1411.4389, (2016), `https://arxiv.org/pdf/1411.4389`

[34] D.Tran, L.Bourdev, R.Fergus, L.Torresani, M.Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks", Facebook AI Research, Dartmouth College, arXiv:1412.0767, (2015), `https://arxiv.org/abs/1412.0767`

[35] J.Carreira, A.Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", DeepMind, Department of Engineering Science, University of Oxford, arXiv:1705.07750, (2017), `https://arxiv.org/abs/1705.07750`

[36] A.Diba, M.Fayyaz, V.Sharma, A.H.Karami, M.M.Arzani, R.Yousefzadeh, L.Van Gool, "Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification", (2017), `https://arxiv.org/abs/1711.08200`

[37] A.Karpathy, Li Fei-Fei, G.Toderici, S.Shetty, T.Leung, R.Sukthankar, "Large-scale Video Classification with Convolutional Neural Networks", Google Research, Computer Science Department, Stanford University, (2014)

[38] M.Lin, Q.Chen, S.Yan, "Network In Network", National University of Singapore, Singapore, arXiv:1312.4400, (2014) https://arxiv.org/abs/1312.4400

[39] Yann LeCun, L.Bottou, Y.Bengio, P.Haffner "Gradient-based learning applied to document recognition", Proceedings of the IEEE, November 1998, http://yann.lecun.com/exdb/lenet/

[40] R.S.Srivastava, K.Greff, J.Schmidhuber, "Training Very Deep Networks", The Swiss AI Lab, arXiv:1507.06228, https://arxiv.org/pdf/1507.06228.pdf

[41] Niklas Donges, "Recurrent Neural Networks and LSTM", Towardsdatascience Internet Webpage, (2018), https://towardsdatascience.com/recurrent-neural-networks-and-\lstm-4b601dd822a5

[42] Sagar Sharma, Towardsdatascience Internet Webpage, "Activation Functions in Neural Networks", (2017) https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[43] C.Ma, M.Chen, Z.Kira, G.AlRegib, "TS-LSTM and Temporal-Inception: Exploiting Spatiotemporal Dynamics for Activity Recognition", Georgia Institute of Technology, Georgia Tech Research Institution, (2017), arXiv:1703.10667 https://arxiv.org/pdf/1703.10667.pdf

[44] Keras: Python Image Generator, https://keras.io/preprocessing/image/

[45] GRaphics and INtelligent Systems group http://grains.polito.it/index.php

[46] Dataset: Google YouTube-8M, 2016 https://research.google.com/youtube8m/

[47] Dataset: A large human motion database, 2011 http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/

[48] Dataset: Activity Net 200, 2016 http://activity-net.org/

[49] Dataset: HOLLYWOOD 2, 2009 https://www.di.ens.fr/~laptev/actions/hollywood2/

[50] Dataset: Charades and Charades-Ego, 2017 https://allenai.org/plato/charades/

[51] Dataset: BN-Jester, 2017 https://20bn.com/datasets/jester

[52] Dataset: BN-SomethingSomething v2, 2017 https://20bn.com/datasets/something-something

[53] Dataset: Kinetics-600, 2017 https://deepmind.com/research/open-source/open-source-datasets/kinetics/

[54] Dataset: KTH - Recognition of human actions, 2005 http://www.nada.kth.

se/cvap/actions/

[55] Dataset: Moments in Time, 2018 http://moments.csail.mit.edu/

[56] NVIDIA Optical Flow SDK, NVIDIA Developer, (2019) https://developer.nvidia.com/opticalflow-sdk

[57] Y.G.Jiang, J.Liu, A.Roshan Zamir, G.Toderici, I.Laptev, M.Shah, R.Sukthankar, THUMOS Challenge 2014: Action Recognition with a Large Number of Classes, THUMOS Challenge 2014, https://www.crcv.ucf.edu/THUMOS14/home.html