POLITECNICO DI TORINO

Master degree in Data Science and Engineering

Master Thesis

EVEgo

Egocentric event-data for cross-domain analysis in first-person action recognition

Supervisor
Prof.ssa Barbara Caputo

Co-supervisors:
Prof. Matteo Matteucci
Dott. Mirco Planamente
Dott.ssa Chiara Plizzari

Candidate
Gabriele Goletto

October 2021

EVEgo: Egocentric Event-data for cross-domain analysis in first-person action recognition
Master thesis. Politecnico di Torino, Turin.

October 2021.

The work in this thesis will be part of a submission to the *Conference on Computer Vision and Pattern Recognition (CVPR) 2021*

Abstract

Event cameras are novel bio-inspired sensors, which asynchronously capture pixel-level intensity changes in the form of “events". The innovative way they acquire data presents several advantages over standard devices, especially in poor lighting and high-speed motion conditions. In particular, their high pixel bandwidth results in reduced motion blur, and their high dynamic range make them a suitable alternative to traditional cameras when dealing with challenging robotics scenarios. Moreover, the latency and low power consumption of these novel sensors enable their use in real-world applications. Indeed, those peculiarities make them perfect to tackle well-known issues which arise from the use of wearable devices, such as fast camera motion and background clutter. However, their potential in these applications, such as egocentric action recognition, is still underexplored. In this work, we bring to light the potentiality of event sensors in first-person action recognition, showing the advantages that they offer over traditional cameras. Specifically, the latter suffer from egomotion, a phenomenon arising from the rapid and involuntary motion of the wearable device, which inevitably moves around with the user. Indeed, this characteristic enables event cameras to extract continuous information from the video. The recent release of the EPIC-Kitchen large-scale dataset, comprehensive of multiple input modalities, i.e., audio, RGB and optical flow, offers the possibility to show the advantages of the event modality over the traditional ones from the first person viewpoint.

In this thesis, we propose an event version of the large scale EPIC-Kitchens dataset, unlocking the possibility to explore the behavior of event data in First Person Action Recognition scenarios. Extensive experiments have been carried out by repurposing a variety of popular action recognition architectures in conjunction with recent temporal modelling methods. Those show the potentiality of event data in both intra- and cross-domain scenarios, establishing a large egocentric action recognition benchmark.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the last several years, expressions like "Artificial Intelligence", "Machine Learning" and "Deep Learning" have started permeating our daily lives and activities. This phenomenon has increased exponentially with the rising popularity of smart devices. Indeed, those are able to steadily collect data of all kinds and to provide users with an increasingly accurate and tailored experience in the most diverse contexts. Nowadays, recommendation systems based on machine learning algorithms can monitor our online activity and purchases to propose products we might be interested to buy. Virtual assistants can make recommendations after considering a number of variables that a human would not be able to handle and they can communicate with us thanks to Natural Language Processing algorithms. Autonomous driving systems are able to correctly interpret road signs and recognize moving objects and advances in robotics allow machines to collaborate with humans in delicate ares such as the medical one. Every day, all over the world, a countless number of new AI applications are studied and developed. The algorithms behind these systems are able to improve their ability to solve their tasks through experience, i.e, the analysis of large amounts of data that allows them to discover patterns, extract useful information and process them to take decisions. This is similar to how humans behave when they have to learn something new.

Deep Learning is a particular branch of Artificial Intelligence where, in order to achieve this ability, the architectures employed by intelligent systems must be able to extract abstract representations from data in a hierarchical fashion, from simple to complex. A field of Artificial Intelligence where Deep Learning architectures have become the state of the art is Computer

Vision. It has the goal of enabling computers and machines to gain high-level understanding from images or videos to solve specific tasks. One of the most researched areas in the field is Action Recognition, a task which aims to automatically predict what action is taking place in a video. Its applications range from assistance systems (a tool could alert someone if an elderly person falls and is alone in that moment) to video surveillance (a system could automatically stop a train if it recognize a person falling on the track). First Person Action Recognition (FPAR) is a sub-field of Action Recognition which entails analyzing images and videos captured by a wearable camera [28]. This sector is gaining great interest thanks to the increasing spread of wearable devices, the release of large and well-annotated datasets [9] and the huge investments in novel technologies e.g., autonomous drones and robots, self-driving systems. Computer Vision tasks are very challenging by nature and even though huge improvements in performances and computational power requirements have been made, the most accurate and technologically advanced architectures in existence are far from biological vision systems. This is because, more recently, a new field of research, related to Computer Vision, is emerging.

The neuromorphic approaches to Computer Vision are insipired by brain structures and can lead to systems which are successful where conventional ones fail. In particular, Dynamic Vision Sensors are novel bio-inspired devices able to asynchronously detect pixel-wise brightness changes. This innovative data acquisition method provides significant advantages over conventional cameras especially in low-light and high-speed motion scenarios. Besides, their low power consumption and low latency would allow the development of many new real-world applications. Nevertheless, the potential of these sensors in certain areas, such as First Person Action Recognition, is still unexplored.

In our work we investigate the behavior of this new data modality in combination with traditional ones in the context of First Person Action Recognition developing a complete novel benchmark of results given by popular models in literature. Our experiments also analyze multi-modal settings which allow the creation of models capable of uniformly processing, integrating and interpreting heterogeneous data flows coming from different sensors. Indeed, multi-sensory information can provide complementary features leading to great improvements in performances. We strongly believe it is critical to include cross-domain analysis into the benchmark we have built in order to provide a fair evaluation of a model considering its ability to generalize on unseen domains because this is what happens in real-world scenarios.

Figure 1.1. Example of annotated RGB frames from an EPIC-Kitchens video.

1.1 Research goals and motivations

This work seeks to explore the behavior of the event modality in the context of First Person Action Recognition when used alone or in a multi-modal fashion with RGB in both intra- and cross-domain scenarios. The egocentric perspective has emerged especially in the last years due to the rapid spread of wearable devices. Its objective is to illustrate how humans interact with the surrounding world and this field could lead to the development of supervisor models able to check whether robots are correctly interacting with the environment while performing some human actions. For example this is critical in fields like the patients care one. Many of the challenges posed by this task can be addressed using the event modality. In fact, the high pixel bandwidth and the wide dynamic range of DVS sensors result in a reduction of motion blur and a greater capacity in dimly lit environments. We claim that event modality, focusing on the motion and not on colors, appearance or textures of an image, is robust to domain shifts like the flow modality [29] but, differently from the latter, it has many practical advantages that make it employable in real-time applications: events are generated real-time by particular sensors with little energy consumption and low computational cost.

We chose to build our benchmark on the EPIC-Kitchens [9] dataset. Indeed, this dataset marked a turning point in this research area and it is the largest egocentric vision dataset already including three modalities (RGB, optical flow and audio). The huge success of this dataset is attributed, in addition to its size, to the challenges standing behind it [1]. Its creation has

[1] https://epic-kitchens.github.io/2021

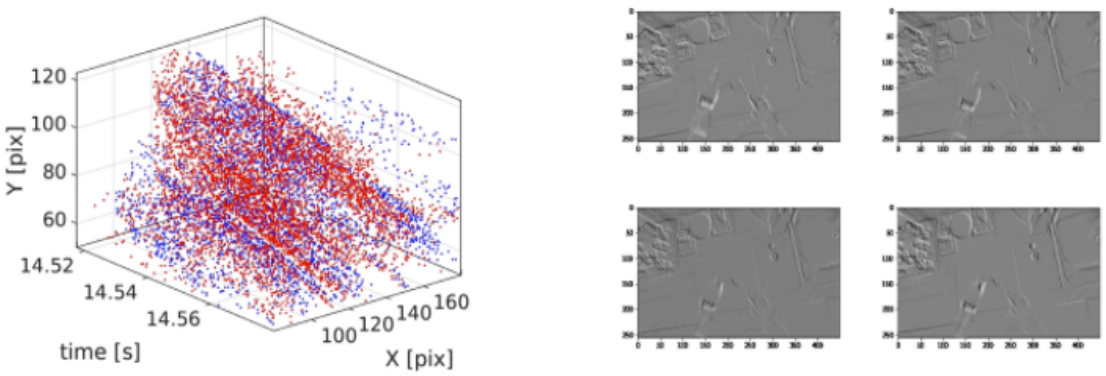


Figure 1.2. On the left a stream of event data. On the right a 4 channels frame-like representation of the asynchronous stream.

gathered participants from all over the world allowing to grasp highly diverse traditions, cooking styles and kitchen habits. Besides, the videos were recorded using a non-scripted (Section 3.4) approach which was adopted to show not just ordinary human actions involving the interaction with a single object but also natural multitasking situations that can occur in real life. Therefore, EPIC-Kitchens is more challenging with respect to many other datasets but it gives a unique and realistic perspective. All these features perfectly match our goal of building a fair and reliable benchmark to evaluate different architectures, methods and data modalities.

It is important to highlight that we also addressed the egocentric action recognition task using multi-modal approach [29, 30]. Indeed, in many cases, a collection of heterogeneous input data from different sensors might offer additional knowledge which reflects the contextual nature of the faced task [10].

The increasingly interest in FPAR context has also brought to light some cross-domain issues [14, 13, 31, 32, 33, 29]: big drops in performances occur when a model, previously trained in a certain environment, is employed in a different context. This is due to an "environmental bias" which prevents the model from being able to generalize on unseen domains. We claim that, since this scenario is frequent in real-world applications, it is crucial to evaluate cross-domain performances of a model not limiting on in intra-domain ones. For this reason, we include in our benchmark both the above-mentioned analyses.

1.2 Main contributions

Recently, innovative learning techniques based on event data have produced remarkable outcomes in circumstances where traditional camera networks fail [34, 35, 36, 37, 38]. These findings demonstrate that event data contains all of the visual information necessary to perform the same tasks as conventional cameras. Unfortunately, effective architectures require a significant quantity of event data for training, which is scarce due to the novelty of event sensors: event cameras only became commercially accessible in 2008 [24]. To address this issue, there is an enormous demand for low-cost, high-quality synthetic, labelled events for algorithm development.

In this work we propose an event extension of the EPIC-Kitchens dataset, created employing a simulator [22] able to generate synthetic event data from existing videos recorded with standard cameras. In addition, we provide a large benchmark of the most well-known architectures in the egocentric action recognition field. This result has been reached through extensive experiments with the aim of showing how event modality perform compared to RGB and optical flow ones, when used in single- or in multi-modal fashion in both intra- and cross-domain scenarios. We hope our contributions can help the scientific community unlock the potential of event data.

Part I

First part: Background

Chapter 2

Deep Learning

This chapter and its subsections provide an overview of deep learning in general, beginning with the most fundamental notions and progressing inward to uncover those most relevant to our work (and particularly the kind of neural networks we adopted). Firstly, Section 2.1 explains the motivations behind the birth and the development of the new deep learning paradigm. Then, in Section 2.2, starting from the beginning, the notion of perceptron will be explained, which will evolve into the first feedforward neural networks as a result of the new backpropagation algorithm. Finally, in Sections 2.3 and 2.4, a detailed examination of the architectures employed in this study will be conducted, including the usage of convolutional neural networks to capture spatial information and residual connections to enable deeper models.

2.1 Deep Learning

For decades, the primary goal of artificial intelligence researchers has been to replicate the behavior of the human brain and to make machines able to perform real world tasks. However, until recently, the majority of approaches for machine learning and data processing relied on shallow-structured architectures. Typically, these designs include no more than one or two layers of non-linear feature transformations. They have been demonstrated to be successful in handling a variety of basic or well-constrained issues, but their limited modeling and representational capabilities can offer challenges when dealing with more sophisticated real-world applications. In particular, as clearly stated by [39],even though perceptual capacities were increased, due to the expanding capabilities of sensors and the vast quantity of information included in input data, the major difficulty encountered by such models was

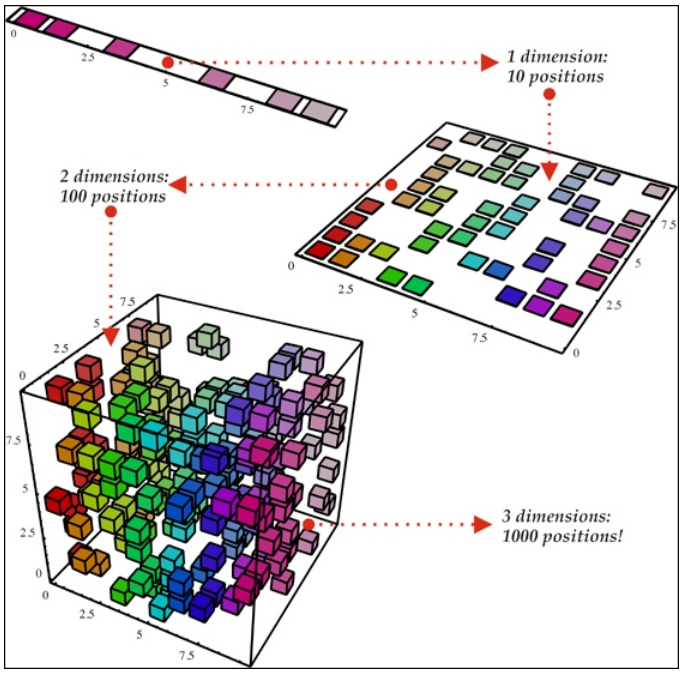


Figure 2.1. With a rise in the number of relevant dimensions in the data, the number of interesting combinations may expand exponentially. Thus, in order to differentiate between d dimensions and v values along each axis, we appear to require $O(v^d)$ regions and instances. [2]

the so-called curse of dimensionality. This term, coined by Richard Bellman in the context of dynamic programming [40], has gained even more popularity in the contexts of artificial intelligence and big data because it succinctly expresses the inherent difficulties in learning in a high-dimensional space where data becomes sparse and statistically insignificant. To address this issue, many approaches have been investigated in order to minimize the dimensionality of data and make it easier for the learning algorithm to extract meaningful knowledge. All of these dimensionality reduction strategies are referred to as feature extraction techniques, which are used to preprocess data in order to increase the relevance of significant information and decrease the amount of duplicated information, thus avoiding the curse of dimensionality. This process can be time consuming and costly, and may result in the extraction of erroneous characteristics supplied in the input.

2.2 Neural Networks

Deep learning arises as a subfield of machine learning with the purpose of avoiding extensive manual preprocessing while drawing loose inspiration from the hierarchical organization of information representation in human brains. Deep learning is a branch of machine learning concerned with the development of algorithms for learning several layers of representation in order to describe complex relationships between data. Thus, higher-level features and concepts are specified in terms of their lower-level counterparts, and this type of feature hierarchy is referred to as deep architecture. In practice, it employs additional non-linear layers to extract the necessary information from the raw input data (as stated by [41]). Three main reasons for deep learning's current popularity are the dramatically enhanced chip processing capabilities enabled by the advent of GPUs, the greatly expanded size of training data, and recent improvements in machine learning. All of these advancements have enabled deep learning approaches to exploit complicated, compositional nonlinear functions successfully, to learn distributed and hierarchical feature representations efficiently, and to make good use of both labeled and unlabeled data. Neural Networks are the core components of Deep Learning. Indeed, they are particular architectures designed to work similarly to how human brain does. The term "deep" refers to the number of layers. In fact, a deep learning algorithm is one that uses a neural network with more than three layers. However, in order to grasp the concept of neural networks, it is necessary to begin with the most fundamental design, the perceptron, and understand what brought us to the huge models we observe nowadays.

2.2.1 Perceptron

The perceptron is the ancestor of the most fundamental unit of a deep neural network, the so called artifical neuron, The origins of the perceptron can be traced back to the cognitive science area, in particular:

- McCulloch & Pitts in [42], provided the first complete report, particularly from a formal standpoint, of how basic units with excitatory and inhibitory synapses and a specified threshold may represent complicated ideas;
- Hebb stated in [43] that *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some*

growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased". In other words, it argued not only that when two neurons fire simultaneously, their connection is strengthened, but also that this activity is a necessary component of learning and memory (known as Hebb's rule);

Resembling to how the biological brain processes information, perceptrons were invented in 1958 by the scientist Frank Rosenblatt [44], who defined them as entities consisting of an input layer and an output layer. The algorithm they adopted by perceptrons is a learning rule based on error minimization that adjusts the weights of the connections (called synapses) proportionally to the difference between the actual and desired output (learning from examples). Specifically, the perceptron takes a sequence of input signals $x_1, x_2...$ and converts them to a single binary output signal. To do so each input x_i is multiplied by its relative weight w_i (i.e., higher weight means higher importance of a certain input), then all the results are summed together with a certain bias w_0. The bias is an element that adjusts the boundary away from origin without any dependence on the input value. Finally, due to the output's binary nature, the real number acquired by the weighted sum is converted to 0 or 1 based on a specified threshold (parameter of the neuron). To put it mathematically:

$$output = \begin{cases} 1, & \text{if } \sum_j x_j w_j + w_0 > threshold \\ 0, & \text{if } \sum_j x_j w_j + w_0 <= threshold \end{cases} \tag{2.1}$$

Note that in this case the binary step has been used to discretize the output of the perceptron. However, different *activation functions* can be used.

The perceptron's learning algorithm is illustrated in 1. While iterating through all the inputs, the weights are only modified in classification error cases, otherwise they are left unchanged. This approach adheres to the previously mentioned Hebb's rule, allowing the model to learn how to properly classify the input samples on its own.

2.2.2 Feedforward Neural Networks

Despite initial enthusiasm, thanks to the prove of convergence of the perceptron algorithm in the case of linearly separable data, Minsky and Papert's 1969 work [45] demonstrated all the limitations of such a reductive model. By posing the simple XOR problem they demonstrated the perceptron's inability to solve it (see Figure 2.3), as well as all other non-linearly separable

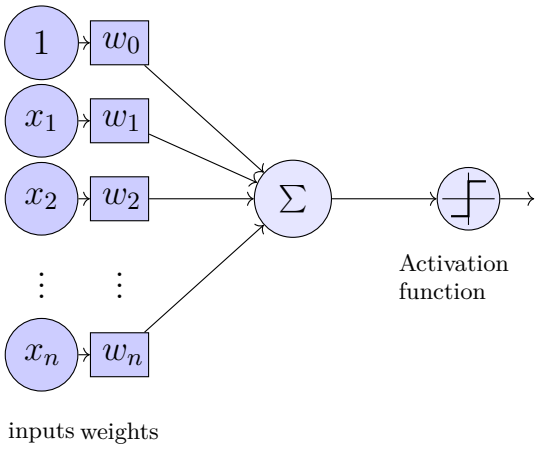


Figure 2.2. Perceptron scheme

Algorithm 1 Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1
$N \leftarrow$ inputs with label 0
Initialize **w** randomly
while !convergence **do**
  Pick random $x \in P \cup N$
  if $x \in P$ and $\langle x, w \rangle < 0$ **then** ▷ Classification error
    $w = w + x$
  end if
  if $x \in N$ and $\langle x, w \rangle \geq 0$ **then** ▷ Classification error
    $w = w - x$
  end if
end while

problems. There was a need to add layers between the perceptron's input and output in order to solve these kinds of problems. To solve this issue multilayer perceptrons have been introduced.

Deep feedforward networks or multilayer perceptrons (MLPs) are fundamental deep learning models. A feedforward network's purpose is to approximate a function f^* by learning the values of the parameters $\boldsymbol{\theta}$ that result in the best function approximation by defining a mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$. Due to the direction of the flow of information, passing onward from the input x, through intermediate calculations, and finally to the output y, these models are referred to as feedforward models. Furthermore they are called networks

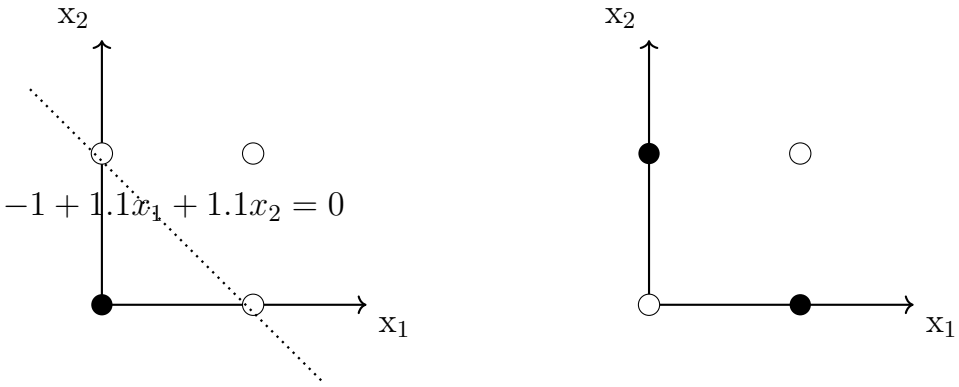


Figure 2.3. Linearly separable domain of the logical operator OR (left) vs non-linearly separable domain of the logical operator XOR (right)

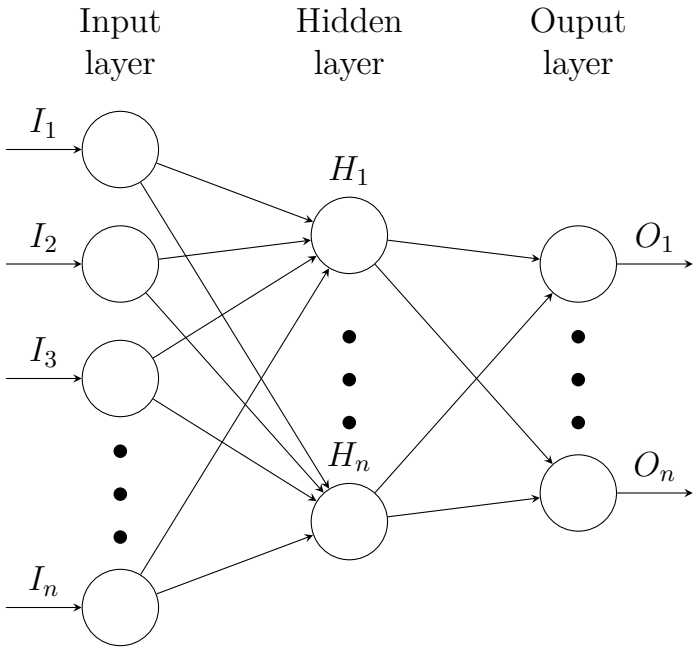


Figure 2.4. A Multi Layer Perceptron

since they are often represented as a collection of several functions. The model is associated with a directed acyclic graph that describes the composition of the functions. As seen in Figure 2.4, data flow from the input layer (the first) through a series of hidden layers (so-called because they are not directly visible) and finally emerge from the output layer. The length of

the chain as a whole represents the model's depth, which is why the term "deep learning" is used. The primary distinction between linear models and neural networks is that the non-linearity of neural networks results in the non-convexity of the most interesting loss functions. This means that neural networks are typically trained using iterative, gradient-based optimizers that simply drive the cost function to a very low value, rather than using linear equation solvers for linear regression models or convex optimization algorithms with guaranteed global convergence. Of course, we can train models such as linear regression and support vector machines using gradient descent as well, and this is frequently done when the training set is quite large. In this sense, training a neural network is similar to training any other model. Calculating the gradient for a neural network is little more complicated, initially there was no algorithm that could provide an appropriate way to efficiently compute gradients until David Rumelhart, Geoffrey Hinton, and Ronald Williams published a work ([46]) in 1986 which made the interest in deep learning grow again.

2.2.3 Gradient Descent

The first thing that must be understood in order to comprehend how the perceptron model was extended is the optimization strategy used to train models with additional layers. To do so, we will describe what optimization means in the context of machine learning and one of the methods to do it. Optimization is the process of reducing or increasing the value of a function $f(\mathbf{x})$ by the manipulation of $\mathbf{x}$. Most optimization problems are typically phrased in terms of minimization of $f(\mathbf{x})$. The objective function or criterion denotes the function we wish to minimize or maximize and it is sometimes referred to as the cost function, loss function, or error function. The derivative is advantageous for minimizing a function because it indicates how to adjust $\mathbf{x}$ in order to enhance $f(\mathbf{x})$ slightly. So, we can minimize $f(\mathbf{x})$ by gradually increasing $\mathbf{x}$ in the opposite direction of the derivative. Gradient descent, presented in [47], is the name of this approach. Due to the fact that we are dealing with multidimensional data, the derivative's generalization is the gradient ∇, and so we can decrease f by travelling in the direction of the negative gradient (see Figure 2.5). This is referred to as the steepest descent technique or gradient descent technique. Steepest descent introduces a new point:

$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}) \tag{2.2}$$

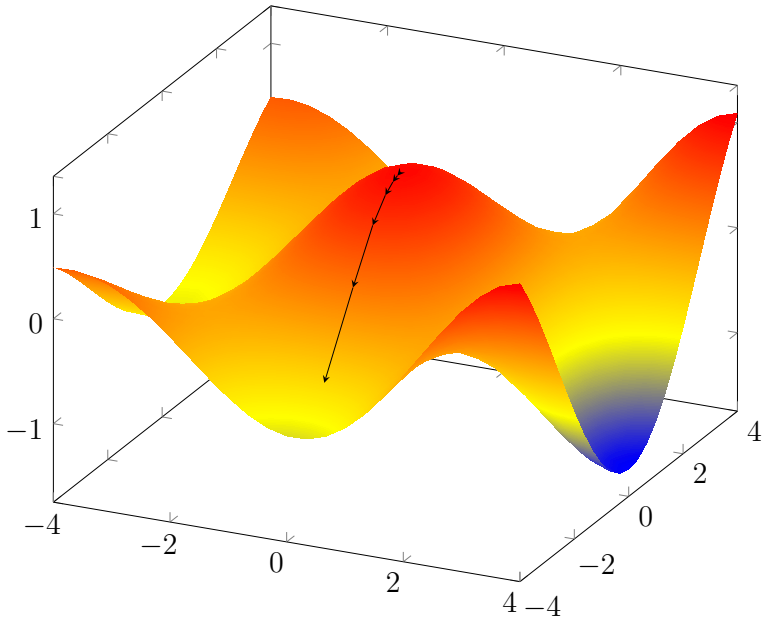


Figure 2.5. An example of gradient descent in a multidimensional space

where ϵ is the learning rate, a positive scalar that determines the step size and setting it to a small constant is a popular technique. Note that this optimization can lead to local minima that are not globally optimal. In other words, we optimize functions that may have numerous suboptimal local minima and numerous saddle points surrounded by extremely flat regions. This is the primary reason why it is critical to appropriately configure the learning rate and schedule for this method, all those techniques will be described in Section 2.2.7. Despite being gradient descent the main pillar for the optimization in neural networks, a recurring issue in deep learning is that while bigger training sets are required for good generalization, they are also computationally more expensive. Because of the high computing cost of large training sets, stochastic gradient descent is an extension of gradient descent which makes the observation that the gradient is an expectation, so, with a limited sample size, the expectation can be approximated. Specifically, we can sample a minibatch of samples $B = x^1, ..., x^m$ taken evenly from the training set at each stage of the algorithm. Typically, the minibatch size m is set to a small number of samples, ranging from one to a few hundred. We can fit a training set with billions of instances using only a hundred examples

as updates. To estimate the gradient the following formula is used:

$$\mathbf{g} = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(x^i, y^i, \theta) \tag{2.3}$$

where θ indicates the parameters of the network, x^i are the samples, y^i the respective labels and $L(x^i, y^i, \theta)$ is that he loss function to be minimized with respect to the weights of the model. The stochastic gradient descent algorithm then descends the predicted gradient using instances from the mini-batch as follows:

$$\theta = \theta - \epsilon \mathbf{g} \tag{2.4}$$

where ϵ is the learning rate as in the gradient descent case.

2.2.4 Back-propagation

Despite being a widely used approach for training deep learning models, Stochastic Gradient Descent would be meaningless without the back-propagation algorithm, which computes the gradient of each network parameter at an acceptable computing cost. The idea of backpropagation came around 1970, but its significance wasn't fully appreciated until David Rumelhart, Geoffrey Hinton, and Ronald Williams published a seminal paper [46] in 1986 when backpropagation was formally introduced as the learning procedure to train neural networks. Calculating the gradient analytically is trivial, but numerically evaluating it can be computationally expensive. The back-propagation method accomplishes this through a straightforward and affordable procedure. Notice that back-propagation is employed merely to compute the gradient; other techniques, such as the ones presented in Section 2.2.3, are utilized to accomplish learning with this gradient. The gradient that we most frequently require in learning algorithms is the gradient of the cost function with respect to the parameters, $\nabla_\theta L(\nabla)$ (see Section 2.2.3). To fully comprehend how the algorithm works, it is necessary to establish the following concepts.

Computational Graph

The computational graph, referring to a directed graph in which the nodes represent variables and the edges represent actions. Variables can be used to feed the value of operations, which are functions of one or more variables. Each node in the graph is defined in this fashion as a function of the variables (see Figure 2.6).

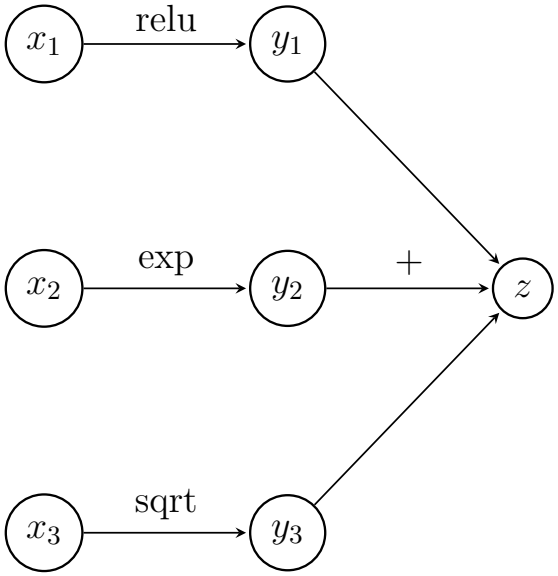


Figure 2.6. An example computational graph

Chain Rule

The calculus's chain rule, which is used to compute the derivatives of functions constructed by assembling known derivatives of other functions. In mathematical terms, if x is a real number and f and g two real functions such that $y = g(x)$ and $z = f(g(x)) = f(y)$, then:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.5}$$

Notice that the rule can be extended to the multidimensional case where $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$ in the following way:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{2.6}$$

which in vectorial terms is:

$$\nabla_{\mathbf{x}^\mathbf{z}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^\top \nabla_{\mathbf{y}^\mathbf{z}} \tag{2.7}$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of the function g mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$.

The back-propagation algorithm employs this Jacobian-gradient product for each operation in the graph. Thus, having established the essential notions enabling the development of feedforward networks, we can now see those models, bearing in mind that the core concepts underlying modern feedforward networks have largely stayed unaltered since the 1980s. The same back-propagation technique and gradient descent approaches are still used. The majority of the progress in neural network performance can be attributable to two reasons:

- larger datasets have alleviated some of the difficulties associated with statistical generalization for neural networks;
- neural networks have grown in size significantly as a result of more powerful processors and improved software infrastructure;

2.2.5 Activation functions

An important concept in the field of neural networks is the activation function. In fact, they are the source of non-linearity of the models which makes it possible to learn complex tasks. This kind of function is often inserted in the hidden layers of the network and there are actually more possible choices which can be made. In abstract terms, activation functions are utilized to determine whether or not a neuron fires.

Binary Step Function

The most basic function to be used to decide when a neuron fires is a threshold-based classifier (as in the perceptron) which determines whether or not the node should be activated based on the value from the linear transformation. In other words, if the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated (Figure 2.7). It may be noticed that this function can be used just when the number of classes is lower than two. Additionally, the gradient of the step function is zero, posing a barrier to back propagation. That is, when the derivative of f(x) with respect to x is calculated, it equals 0.

Sigmoid

The *sigmoid* function is a very popular non-linear activation function. The *sigmoid* function changes values between 0 and 1 (Figure 2.8). Notably,

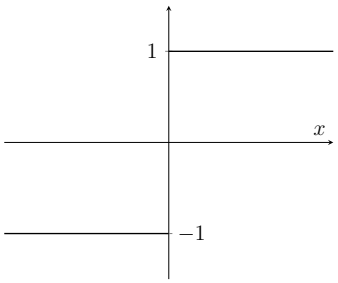


Figure 2.7. Binary Step

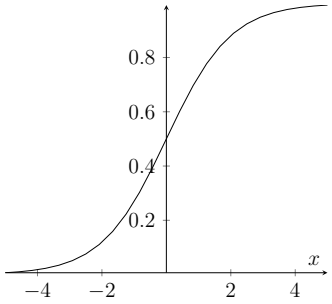


Figure 2.8. Sigmoid Activation Function

in contrast to the binary step and linear functions, the *sigmoid* is a non-linear function. Thus, when having multiple neurons with *sigmoid* activation functions, the output is also nonlinear. Although the gradient values are considerable between -3 and 3, the graph becomes significantly flatter in other places. This implies that for values more than 3 or less than -3, the gradients will be extremely small. Gradient values approaching zero indicate that the network is not truly learning. Additionally, the *sigmoid* function is asymmetrical around zero. Thus, the output of all neurons will have the same sign.

Hyperbolic Tangent

Unlike the *sigmoid* function, the *tanh* function is symmetric around the origin (Figure 2.9), with the value range being between -1 to 1. As a result, the inputs to subsequent levels will not always be the same in sign. The *tanh* function, like the *sigmoid*, is continuous and differentiable at all locations but it has a steeper gradient than the *sigmoid* function. Generally, *tanh* is chosen over *sigmoid* because it is zero-centered and the gradients are not

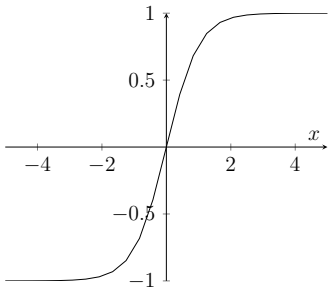


Figure 2.9. Hyperbolic Tangent Activation Function

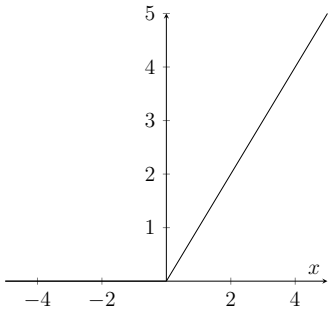


Figure 2.10. ReLU Activation Function

confined to move in a particular direction.

ReLU

Another non-linear activation function that has gained prominence in the deep learning sector is the ReLU function. Rectified Linear Unit is abbreviated as ReLU (Figure 2.10). The primary advantage of the ReLU function over other activation functions is that it does not simultaneously stimulate all neurons. This signifies that neurons will be deactivated only if the linear transformation's output value is less than 0. Due to the fact that just a limited number of neurons are engaged, the ReLU function is significantly more computationally efficient than the sigmoid and *tanh* functions. If you examine the graph's negative side, you'll discover that the gradient value is zero. As a result, certain neurons' weights and biases are not updated during the backpropagation process. This can result in the death of neurons that are never stimulated.

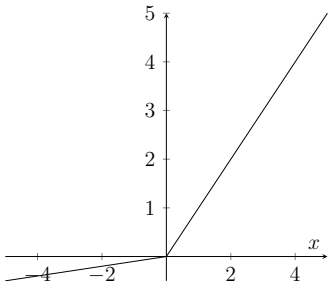


Figure 2.11. Leaky ReLU Activation Function

Leaky ReLU

The Leaky ReLU function is a modified version of the ReLU function (Figure 2.11). Leaky ReLU is developed to overcome the issue of deactivation of neurons for $x < 0$; thus, rather than declaring the ReLU function as 0 for the negative inputs, we define it as an extremely tiny linear component of x. By making this minor adjustment, the gradient on the left side of the graph becomes non-zero. As a result, we would experience no more dead neurons in that region.

2.2.6 Loss functions

As previously stated, the ultimate goal of all machine learning algorithms is to minimize a loss function. The type of loss function that is considered varies according to the type of problem to be addressed and is also related to the activation function chosen for the final layer.

Mean Squared Error

The Mean Squared Error (MSE) loss is used to get the average of the errors' sum of squares. It takes the squared difference between the estimated and real numbers and averages it. It is a type of risk function in which the difference between the actual and projected values is squared and averaged over the number of instances in the model. MSE values that are close to 0 are preferable because they indicate that the model has less error. MSE can be formulated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{true} - y_i^{predicted})^2 \tag{2.8}$$

where n is the number of samples. Notably, this function cannot be used when the result is discrete (as is the case with classification issues), but only when the output is continuous (as is the case with regression). Its primary advantage over a Mean Absolute Error ($MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i^{true} - y_i^{predicted}|$) is that, due to its quadratic structure, the MSE does not contain local minima and penalizes larger errors more severely.

Cross Entropy Loss

The Cross Entropy loss, or log loss, is a metric used to evaluate the effectiveness of a classification model whose output is a probability between 0 and 1. To generate a probability value as the network's output for a classification task, a special function called *softmax* is applied to the output of the final layer to generate a probability vector (whose elements sum to 1), and typically one output neuron is reserved for each possible class in the output vector (one-hot encoding of the classes). The *softmax* fuction can be calculated as it follows:

$$f(o)_i = \frac{e^{o_i}}{\sum_j^C e^{o_j}} \tag{2.9}$$

where s_i is the output for class i on one specific neuron while C is the number of different classes. Finally, the formula to calculate the Cross Entropy Loss on the *softmax* output can be formulated as:

$$CE = -\sum_i^C t_i log(f(o)_i) \tag{2.10}$$

where t_i is 1 for the observations predicted correctly, 0 otherwise.

2.2.7 Learning rate

The learning rate is a tuning parameter that controls the step size for each iteration in order to minimize the loss function. Since it determines the extent to which new information modifies the weights it is a metaphor for the rate at which a deep learning model "learns". When choosing a learning rate, a trade-off must be made between pace of convergence and overshooting. While the direction of descent is typically decided by the gradient of the loss function, the learning rate dictates the magnitude of the step taken in that direction. A learning rate that is too high will cause the learning to skip minima, whereas a learning rate that is too slow will either take too long to converge or will become stuck in an undesirable local minimum. To improve

faster convergence, avoid oscillations, and avoid local minima during training, the learning rate is frequently changed, either according to a learning rate schedule or by utilizing an adjustable learning rate. A step-based learning schedule alters the rate of learning in response to predetermined stages. The decay application formula is defined as it follows:

$$\epsilon_n = \epsilon_0 d^{floor\left(\frac{1+n}{r}\right)} \tag{2.11}$$

where ϵ_0 is the initial learning rate, ϵ_n is the learning rate at iteration n, r is the iteration at which the drop happens while d is the drop rate (how much the learning rate decreases). Finally there are optimizers which automatically alter the learning rate parameter such as Adam, devised by [48]. They adapt learning rates through the usage of momentum, it is a variant of the gradient descent technique.

2.2.8 Weight Initialization

In order to facilitate network's convergence, it is critical how we initialize the network weights. Indeed, two extremely typical challenges in Neural Networks are:

- the vanishing gradients problem, which arises when gradients tend to shrink as errors propagate back via deep neural networks' buried layers. Because the gradient is nearly zero around the asymptotes, gradients do not spread properly.
- the exploding gradient problem, which arises as gradients in prior layers become considerably bigger, causing instability.

Numerous recent studies have focus on developing effective weight initialization procedures, both to ensure gradient stability during training and to accelerate the learning process. The authors of [49] devised the Xavier or Normalized Initialization, whose formulation is:

$$W \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right) \tag{2.12}$$

where n_j is the size of the j-th layer while $\mathcal{U}$ is the uniform distribution.

Another frequent technique is to employ a pretrained neural network that has previously learned to extract powerful and informative features as a starting point for learning a new task. The bulk of pretrained networks in the

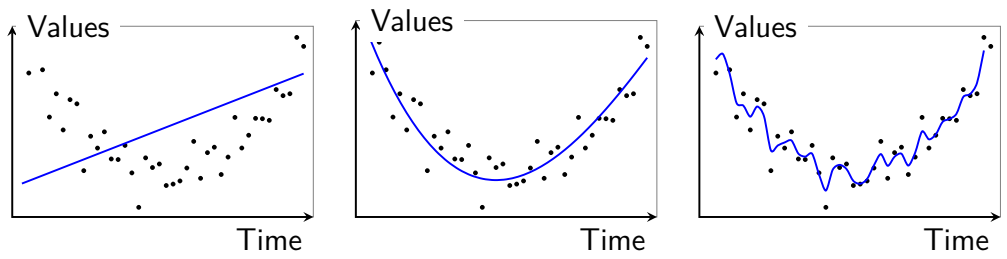


Figure 2.12. **Left**: Underfitting **Center**: Proper Fitting **Right**: Overfitting

context of CNNs (which will be discussed in Section 2.3) are trained on the ImageNet database (proposed in [50]). These networks have been trained on massive amounts of data and can be utilized as a starting point for weights, resulting in a more rapid and straightforward training process than beginning from scratch.

2.2.9 Overfitting and Regularization

Overfitting is a data science notion that refers to a situation in which a statistical model fits perfectly to its training data. When this occurs, the algorithm is unable to execute accurately on unseen data, negating the method's goal. In fact, the ability of a model to generalize to new data is ultimately what enables us to use machine learning algorithms to make predictions and classify new data. When machine learning algorithms are developed, they are often trained using a sample dataset. However, if the model is trained for an extended period of time on sample data or if the model is very sophisticated, it can begin to learn about the dataset's noise or irrelevant information. When a model memorizes the noise and becomes excessively similar to the training set, it gets overfitted and it becomes incapable of generalizing well to new data. If a model is unable to generalize adequately to new data, it will be incapable of performing the classification or prediction tasks for which it was designed. Low error rates and a large variance indicate overfitting. To avoid this type of behavior, a portion of the training dataset is often set aside as the "test set" to ensure that the model does not overfit. If the error rate of the training data is low while the error rate of the test data is high, this indicates overfitting, as it can be seen in Figure 2.12. If overtraining or model complexity results in overfitting, a sensible preventative approach would be to either interrupt the training process sooner, often known as *early stopping* or to lower the model's complexity by removing irrelevant inputs.

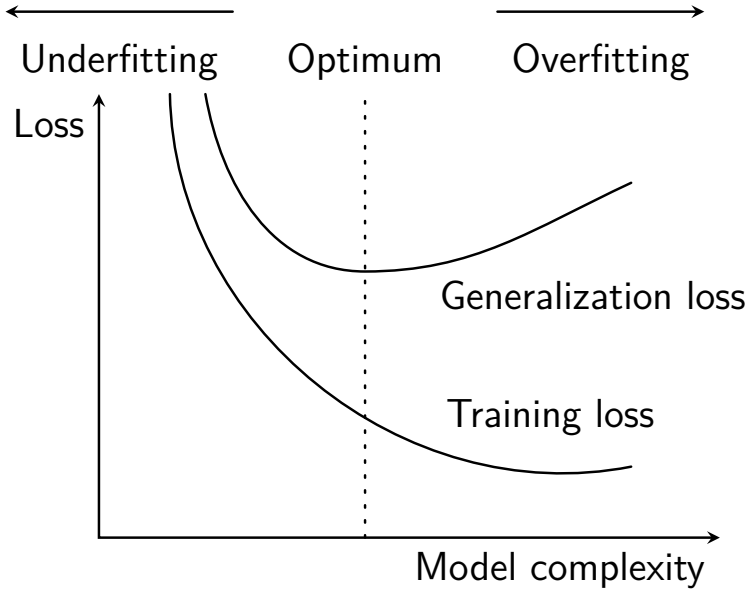


Figure 2.13. Optimum stopping condition to avoid overfitting

However, if you halt too soon or eliminate too many critical data, you risk encountering the reverse problem and underfitting your model. Underfitting occurs when the model has not been trained for a sufficiently long period of time or when the input variables are insufficiently significant to establish a meaningful relationship between the input and output variables. In comparison to overfitted models, underfitted models exhibit a high degree of bias and less variance in their predictions. This demonstrates the bias-variance tradeoff that happens when an underfitted model becomes overfitted. While the model's bias decreases as it learns, its variance may grow as it starts overfitting. When fitting a model, the objective is to locate the *sweet spot* (Figure 2.13) between underfitting and overfitting, enabling the model to establish a dominating trend and be applied broadly to new datasets. There are a number of possible strategies for avoiding overfitting and increasing the model's generalizability. To mention some, one could apply the previously mentioned *early stopping*, increasing the dataset size (where possible) or using data augmentation to increase the amount of data by adding slightly modified copies of existing data or newly created synthetic samples. Another strategy is regularization, which constrains the model to avoid excessive complexity. *Weight decay* is a typical regularization strategy that consists in including a penalty term in the cost function ($cost = loss + penalty$). The additional term, in

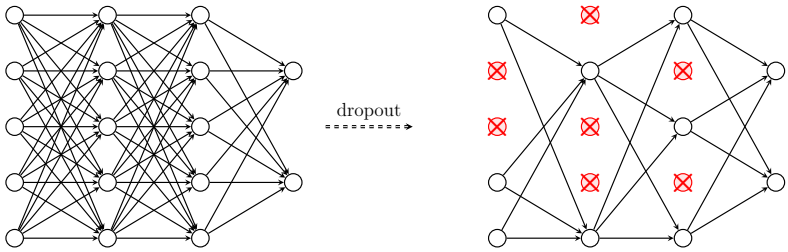


Figure 2.14. Example of dropout effect

particular in the $L2$ case is defined in the following:

$$L2_{penalty} = \frac{\lambda}{2} \sum_i w_i^2 \tag{2.13}$$

it encourages the sum of the squares of the parameters to be small, whereas the $L1$ penalty penalizes the sum of the weights' absolute values as illustrated in the following equation:

$$L1_{penalty} = \frac{\lambda}{2} \sum_i |w_i| \tag{2.14}$$

where λ is the weight for the penalty. Finally, another well-known normalizing strategy is dropout, which refers to the practice of setting to zero units, i.e., neurons, during the training phase of a randomly chosen set of neurons. This implies that these units are not taken into account during a given forward or reverse pass (see Figure 2.14). More precisely, during each training stage, individual nodes are either eliminated from the network with probability $1 - p$ or retained with probability p, resulting in a smaller network; also, the incoming and outgoing edges to a dropped-out node are erased.

2.2.10 Batch Normalization

During the training stage of neural networks, as the parameters of preceding layers change, the distribution of inputs to the current layer also changes, requiring the current layer to constantly readjust to new distributions. This is a particularly serious problem for deep networks, because tiny changes in shallower hidden layers are amplified as they propagate through the network, resulting in considerable shifts in deeper hidden layers. Internal covariate shift (as stated by the authors of [51]) is the term used to characterize this

effect on the distribution of inputs to internal layers during training which is caused by the distribution of each layer's input changes after each training step as a result of the previous layer's weights being updated. A frequently used technique for mitigating the impacts of internal covariate shift is to employ a slower learning rate. This technique, however, has the disadvantage of greatly slowing down the training process. Given that a neural network's input is often normalized, the authors of [52] advises normalizing the input of hidden or output layers as illustrated in the following equation:

$$f_{BN}(\mathbf{x}) \triangleq (\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma} \tag{2.15}$$

with $\oslash$ being the element-wise division and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ being the mean and the standard deviation of the features:

$$\begin{aligned} \mu_i &= E[x_i] \\ \sigma_i &= \sqrt{\delta + Var[x_i]} \end{aligned} \tag{2.16}$$

with δ as a small number in order to avoid division by zero in the normalization phase. During training, statistics are produced for the current batch, and at test time, they can be used to infer for single samples using previously gathered data. Due to the fact that batch normalization might lower a model's expressive ability, it is often formulated as:

$$f_{BN}(\mathbf{x}) \triangleq \boldsymbol{\gamma}[(\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}] - \boldsymbol{\beta} \tag{2.17}$$

This formulation allows for any mean and variance in the output features, but both learnable values ($\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$) are inferred as single parameters for optimization, preserving the benefit achieved through batch normalization. Apart from lowering internal covariate shift, batch normalization is claimed to have a lot of additional benefits. This additional function enables the network to operate at a faster learning rate without experiencing vanishing or exploding gradients. Additionally, batch normalization appears to have a regularizing impact, enhancing the network's generalization qualities, obviating the need for dropout to minimize overfitting. Additionally, it has been discovered that using a batch normalization strengthens the network's resistance to diverse initialization schemes of weights and learning rates.

2.3 Convolutional Neural Networks

Convolutional Neural Networks, or CNNs, are a subset of neural networks used to analyze data with a known, grid-like topology, as stated by [53].

Time series data, for example, can be thought of as a 1D grid with samples taken at regular time intervals, while picture data can be thought of as a 2D grid of pixels. Convolutional networks have proved enormously successful. In fact, the first difficulty that multilayer perceptrons encounter when working with images is that their completely connected structure introduces an enormous number of parameters. Furthermore, MLPs lacks in capturing spatial information, which prevents the network from taking into account the location of pixels in the image. The term "convolutional neural network" derives from the basic operation they use, which is called "convolution". It is, basically, a systematic process in which two sources of information are combined, an operation that transforms one function into another. Convolutions have been used in image processing for a long period of time, mainly to blur and sharpen images, but also to conduct other operations (e.g. enhance edges and emboss).

2.3.1 Convolutional layer

To impose a pattern of local connection between neurons in neighbouring layers, CNNs adopt convolutions. In particular, convolutional layers perform a convolution on the input and transmit the result to the following layer. Convolutions work through a certain number of filters or kernels sliding over the incoming data and multiplying it element by element. The result is achieved by adding all the elements of the multiplication result, see Figure 2.15. Mathematically, the result of a convolutional layer is obtained through the following formula:

$$conv(p,q) = \sum_i \sum_j f_{(i,j)} x_{(i+p,j+q)} \tag{2.18}$$

where f is the filter, x is the input image and $conv$ is the resulting feature map. The indexes p and q are used to move on the input image (and refers to where the filter is on the input image) while i and j are used to iterate over the filter's dimensions, multiplying each weight by the corresponding value of the input image's pixel. The kernel will repeat the operation for each point it slides over, and because there are more kernels, there will be more output feature maps, all of which will be piled together. Take note that the filter's weights are spread throughout the entire image. This technique, intuitively, allows us to extract certain local characteristics from the input, and indeed, the output matrix is referred to as a *feature map*. Finally, three hyperparameters regulate the spatial organization of the output volume: *depth*, *stride*,

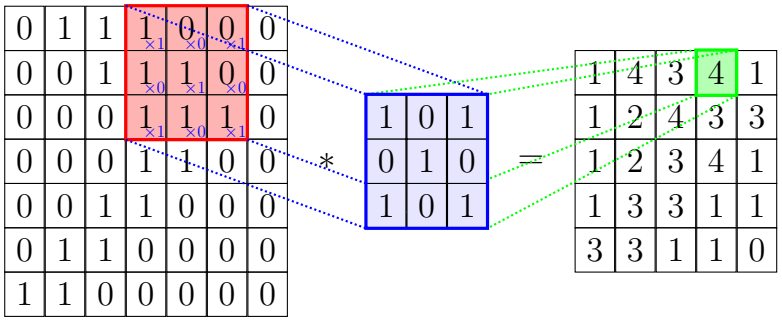


Figure 2.15. Example of convolution operation

and *padding*. The *depth* parameter specifies the number of filters we want to utilize, which corresponds to the number of feature maps we will have. The *stride* parameter determines of how many pixels is the filter moved while sliding across the image. The *padding* is used to increase the dimensions of the input with a variety of techniques (e.g. zero padding in images, adding zero pixels to the input data). Through the use of these three hyperparameters (D, S, and P), the dimension of the filter ($H_f \times W_f$) and the dimension of the input (e.g. RGB picture $C \times H \times W$), the following simple formula can be used to regulate the output's proportions:

$$\begin{aligned} H_{out} &= \frac{H - H_f + 2P}{S} + 1 \\ W_{out} &= \frac{W - W_f + 2P}{S} + 1 \\ C_{out} &= D \end{aligned} \tag{2.19}$$

2.3.2 Pooling layer

A pooling function replaces the output of a network at a particular point with a summary statistic of the network's adjacent outputs. The max pooling method, described in [54] (see Figure 2.16, for example, returns the maximum output within a rectangle region. Additionally, the average of a rectangle neighborhood, the $L2$ norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel are all popular pooling functions.

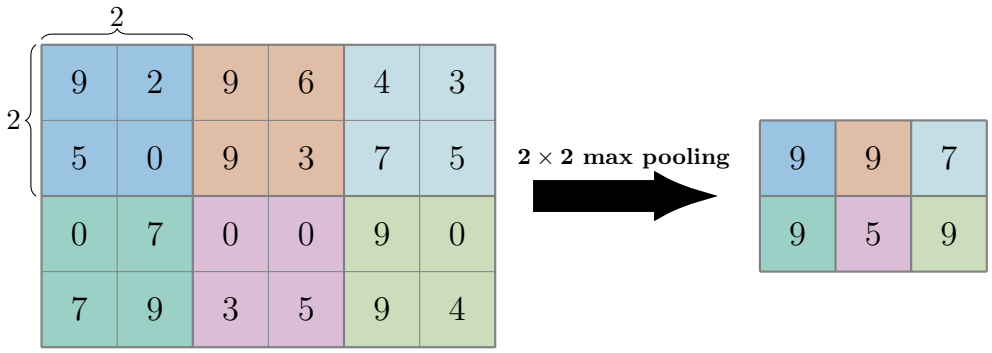


Figure 2.16. Example of max pooling

In each scenario, pooling contributes to the representation becoming approximately invariant to modest input translations. Invariance to translation implies that when the input is translated by a tiny amount, the values of the majority of pooled outputs remain unchanged. Invariance to local translation might be a highly beneficial trait if we are more concerned with the presence of a feature than with its precise location. When the number of parameters in the subsequent layer is proportional to the size of its input (as when the subsequent layer is fully connected, which will be seen in Section 2.3.3, and based on matrix multiplication), this reduction in the input size can also result in increased statistical efficiency and decreased memory requirements for storing the parameters. Pooling is critical for many jobs since it allows for the management of inputs of varied sizes. For instance, if we want to identify photographs of varying sizes, the classification layer's input must be fixed in size. This is typically performed by adjusting the amount of the offset between pooling areas such that the classification layer always receives the same number of summary statistics regardless of the size of the input. For example, regardless of the image size, the final pooling layer of the network may be programmed to produce four sets of summary statistics, one for each quadrant of the image.

2.3.3 Fully Connected layer

Following feature extraction (which is mostly accomplished through convolutional and pooling layers), we must combine the highest-level abstractions and relationships to take the final output decisions. This can be accomplished using a fully connected (FC) layer. The fully connected layers learn

a (potentially non-linear) function between the convolutional layers' output high-level features. Fully connected layers in a neural network are those components responsible of connecting the input from one layer to each activation unit in the following one. As such, they can be defined as a particular type of multilayer perceptrons.

2.4 Residual Neural Networks

As models become deeper and more complex, it becomes increasingly difficult for the layers to transport information from shallow layers to deeper layers, and the information is lost. This is referred to as the *degradation problem*. In fact, the authors of [3] demonstrated empirically that the classic CNN model has a maximum depth threshold (see Figure 2.17). As network depth increases, accuracy becomes saturated and then rapidly declines. Surprisingly, such degradation is not due to overfitting, because adding additional layers to a sufficiently deep model results in increased training error. However, the relevance of network depth has been clear since 2012, when the authors of [55], for the first time, showed the potentialities of Deep Convolutional Neural Networks thanks to the performance obtained by their model (AlexNet) that outperformed manual feature learning on the ImageNet. Additional layers are important to gradually acquire increasingly complicated properties. The first layers acquires knowledge of edges and then, going more in depth, shapes, objects, eyes and so forth. In addition to that, it is possible to see how increasing the model's depth has a significant impact on its performance also through the works done by [56, 57]. To address the *degradation problem* and allow more complex models, the authors of [3] proposed residual networks operating via shortcut connections. The building block for these types of networks is depicted in Figure 2.18, where the concept is explained. Formally, expressing the the desired underlying mapping as $\mathcal{H}(x)$, the stacked nonlinear layers suit another mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$. The initial mapping is converted to $\mathcal{F}(x)+x$. [3] suggest this change through the idea that optimizing the residual mapping is more straightforward than optimizing the initial, unreferenced mapping. $\mathcal{F}(x)+x$ can be implemented using feedforward neural networks with *shortcut connections*. The term *shortcut connections* refers to those that bypass one or more layers. In this scenario, the shortcut connections simply conduct identity mapping, with their outputs being added to the stacked layers' outputs. No additional parameters or computational complexity are added by identity shortcut links. This reformulation is inspired

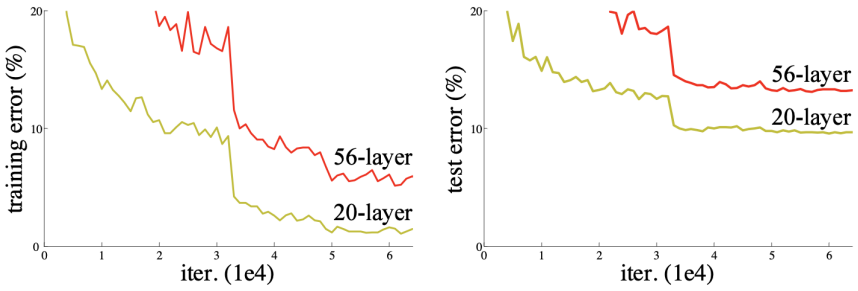


Figure 2.17. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. [3]

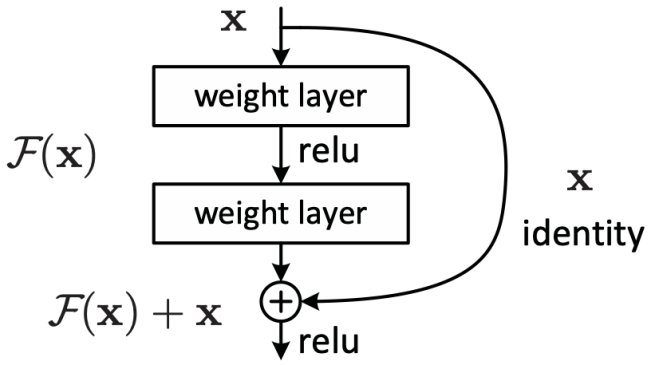


Figure 2.18. Residual block [3]

by the seeming paradoxes around the degradation problem. The degradation problem implies that solvers may encounter difficulties approximating identity mappings using many nonlinear layers. If identity mappings are optimal, solvers can simply drive the weights of the many nonlinear layers toward zero to approximate identity mappings using the residual learning reformulation. While identity mappings are unlikely to be optimal in practice, our reformulation may help precondition the problem. If the optimal function is closer to an identity mapping than to a zero mapping, finding perturbations with reference to an identity mapping should be quicker for the solver than learning the function as a new one. Finally, in [3] more architectures are suggested with a total of 18 to 152 levels. Each of the recommended networks is composed of many stacked residual blocks. The one that is most relevant to this work is the model with 50 layers, known as ResNet50, however, for spatial

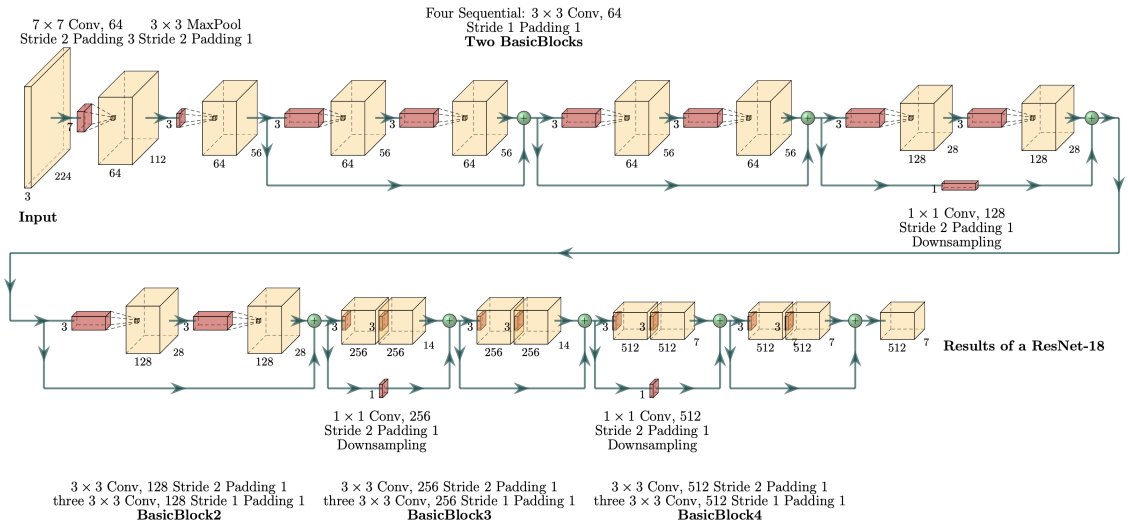


Figure 2.19. ResNet-18 feature extractor

reason in Figure 2.19 it is shown just ResNet18.

Chapter 3

Action Recognition from videos

In this chapter we give an overview of the main task faced in our work, focusing on the achievements reached in this field in the recent years.

3.1 Task

Action Recognition is a classification task comparable to the well-known object recognition task performed on images. The objective is to predict the class to which a segment of a video belongs. The temporal interval of each segment is known a priori. In principle the action recognition task could also be performed for images but many actions would not be distinguishable from one another: for example the actions "pull the door" and "push the door" can be easily recognized if the temporal reference is given and this is only possible if we are dealing with videos and not images.

While object recognition has become more and more refined, the current architectures tackling action recognition have not yet attained optimal results. This is certainly due to a greater difficulty of this task that, in addition to a modelling of spatial information, also requires a modelling of temporal information. Many other obstacles must also be tackled:

- handling videos instead of images is computationally expensive, requires powerful hardware and this can also cause more overfitting problems and long training time;
- the videos to classify are not of a fixed length so the developed model

Figure 3.1. Example frames from videos in popular action recognition datasets [4].

have to deal with different temporal extensions;

- the videos may have been registered not in a controlled environment and this can result in blundering background, different lighting conditions, low quality frames or even occlusions.

If on one hand it is true that for the action recognition task the label is an action, things may vary when employing different datasets. In fact, recently, several new datasets are designed with finer-grained action labels. An in-depth analysis in this sense will be given in the chapter describing the used dataset in our work. Automatically recognizing the content of a video, or, more precisely, the activities represented in it, is not only valuable for organizing massive video datasets. There can be positive sides in a countless number of other applications such as assistance systems (a tool could alert someone if an elderly person falls and is alone in that moment), video surveillance (a system could automatically stop a train if it recognize a person falling on the track) or human-computer interactions (a film could automatically paused if the person who is watching it temporarily leaves the room).

3.2 Third Person Action Recognition

Following the success of image recognition, detecting actions in videos seemed a natural development, particularly considering the massive amount of videos published on the Internet every day. While the inclusion of the time dimension might aid in action discrimination, it also represents a significant challenge to be faced.

Prior to the advent of deep learning, hand-crafted features were the state of the art in this field. The standard pipeline started with the extraction of the features. Then, they were processed or encoded differently to improve their quality, and finally a classifier (SVM most of the time) performed the prediction. Following the creation of larger and more complex datasets and due to the deep learning's ongoing progress with images, the deep learning approach began to be used also to perform action recognition of videos. At the beginning, frame level features, extracted with CNNs pretrained on image datasets, were used instead of the traditional hand-crafted ones. The drawback of this naïve method was the lack of explicit modelling of the temporal dimension. Later on, several attempts were made to employ CNNs trained in an end-to-end fashion and this resulted in a total replacement of the old conventional pipeline.

In the following, we present an overview of the most popular techniques and architectures in the field.

Two-stream networks. This method has the advantage of using different data modalities which often bring orthogonal information to the RGB data stream (for more details read Section 3.5). In [5] they propose this kind of architecture. Raw video frames are fed as input to the spatial stream with the aim of extracting visual features. The temporal stream, instead, is able to extract motion information by receiving optical flow images obtained by scaling the components of the estimated flow to a [0, 255] range.

Deeper architectures. The advancement in Deep Learning has brought to the rise of deeper architectures which, however, can bring to overfitting issues. Wang et al. [58] established some good practices (e.g., synchronized batch normalization, large dropout, cross-modality initialization) to tackle these problems. Thanks to this research, they managed to develop a two-stream network using VGG16 model [57].

Recurrent neural networks. Considering a video like a temporal sequence is a perspective which opens to the usage of Recurrent neural networks, particularly Long Short-Term Memory (LSTM) [59].

Segment-wise approaches. Two-stream networks are not able to retain

information spanning over large temporal windows. In [6] Wang et al. introduced an architecture to implement video-level action recognition taking into account the entire input video. First, the whole video is divided into segments uniformly scattered over time. Then, from each segment, a single frame is taken and fed into the network whose weights are shared among all the input frames. The final prediction is reached via consensus techniques over the intermediate predictions (e.g., max or average pooling, bilinear encoding).

3D CNNs. The usage of this networks comes into play when a video is perceived like a 3D tensor (2 spatial dimension and 1 time dimension). [60] gave birth to this approach in 2012 but a turning point took place only in 2017 with I3D network [7].

3.3 Egocentric Action Recognition

In the last years, with the rapid spread of wearable devices, a new perspective has emerged: the egocentric one. In this case, rather than using a fixed camera to record a scene, as it is done when producing films, egocentric videos are registered from the perspective of the individual who is performing a certain action. Usually, the wearable cameras are mounted on the head of the participant. Differently from third-person videos, in this case objects and actions can occupy a greater portion of each frame. Nevertheless, egocentric domain poses many other challenges. Indeed, the body or head motion of the participant constitute a noise in the global scene recording. Besides, during the video, the hands could sometimes obscure relevant objects or some actions could occur while the participant is not directly looking at them with the camera. The objective of the egocentric domain is to illustrate how humans interact with the surrounding world and, as a result, it would be possible to build supervisor models able to check whether robots are correctly interacting with the environment while performing some human actions. This research area could bring several benefits in delicate applications such as the one in the patients care field.

Fine-grained action recognition is an extension of the action recognition task introduced so far which deals with very specific labels. It is the problem of distinguishing very specific actions such as "cutting a tomato" or "tightening a bolt" instead of coarse-grained interactions such as "preparing a meal" [29]. This task further requires techniques able to reason not only about the general action performed and the principal object of interest but also on the

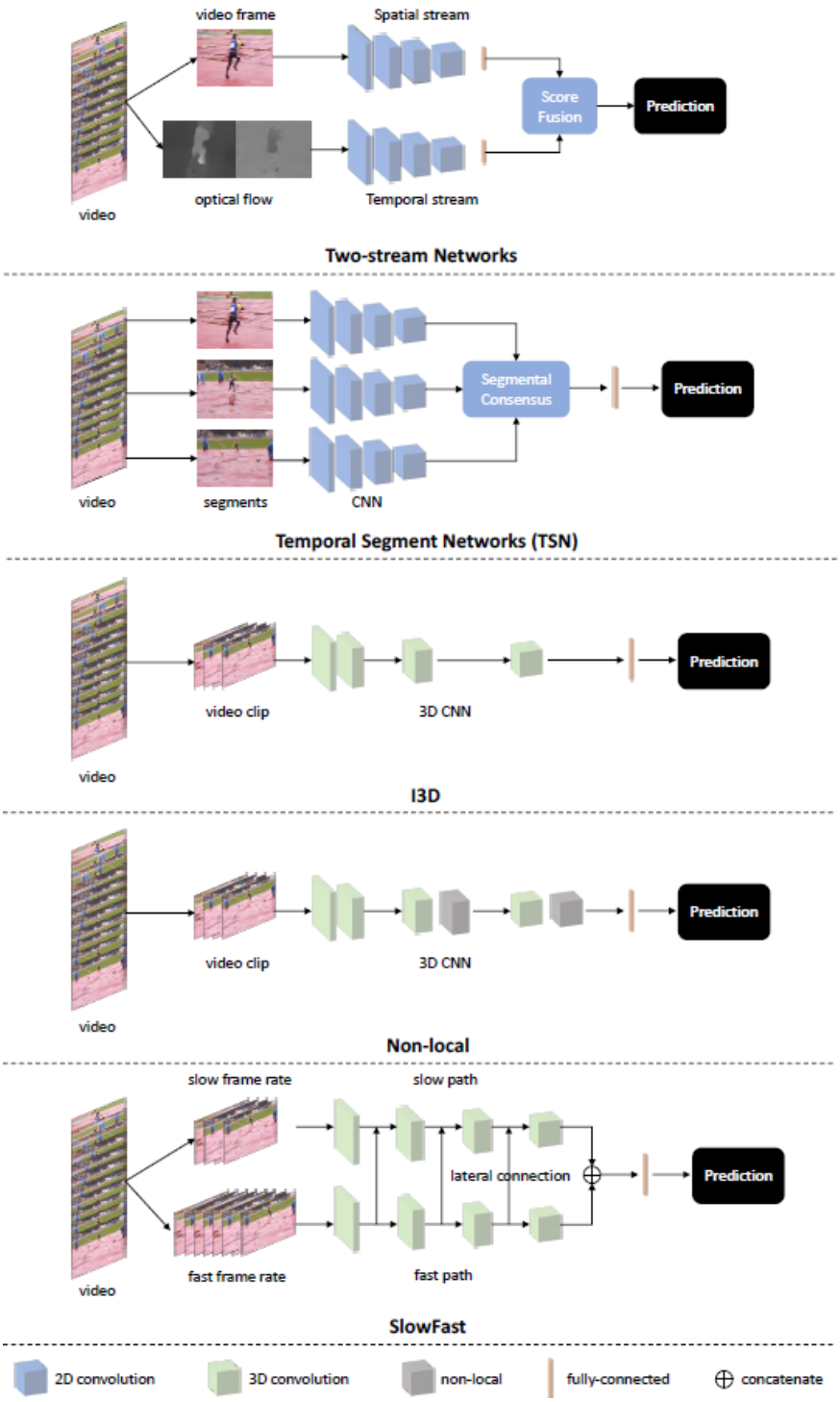


Figure 3.2. Overview of the mentioned architectures employed in the context of Action Recognition [4]. From top to bottom: two-stream networks [5], TSN [6], I3D [7], Non Local [8]

way the participant is interacting with the environment and with surrounding items.

The most employed architectures and techniques used in the egocentric context are derived from the third person one (more details in Section 3.2). 2D networks [5, 6, 61, 11, 62], 3D ones [63, 64, 65, 5] and LSTM [66, 67, 68] have been successfully adopted to solve First Person Action Recognition tasks. The architectures involved in our novel benchmark are detailed in further sections.

3.4 EPIC-Kitchens Dataset

In this section we provide the reader with a detailed description of the EPIC-Kitchens dataset [9] which is the dataset we have chosen for our work and its extension is one of our contributes.

In the last decade, significant improvement has been made in a variety of Computer Vision related fields, including object detection and image classification. This success has been possible thanks to the advancements in the deep learning architectures and the availability of new large image benchmarks such as ImageNet [50], ADE [69] or VOC [70]. This success, however, has not been seen in the video understanding field since only a small number of annotated video datasets were available to the research community. Only recently this has begun to change with the release of many new datasets: most of these consisted of a collection of short videos, each recording a single action performed by the user. A small turning point came with Hollywood in Homes [71] because this dataset, unlike the previous ones, was an attempt to collect longer videos showing humans performing the most varied actions. The issue with this approach is that the videos are recorded in a scripted way (i.e., participants sequentially follow a series of instructions) and this results in less natural videos which, in addition, do not represent multi-tasking actions which are typical of human behavior.

EPIC-Kitchens [9] is the largest egocentric dataset and it includes videos recorder in the kitchens of 32 participants from 10 different countries. All the recordings are performed using a head-mounted Go-Pro able to capture both video and audio. The participants were asked to take a video every time they entered the kitchen for at least three consecutive days and the recording was only stopped before leaving the room. All these gimmicks have allowed the creation of an extremely rich and varied dataset, from many points of view: the multi-ethnicity of the participants made it possible to grasp highly

Figure 3.3. Frames from the 32 environments[9]

diverse traditions, cooking styles and kitchen habits. The non-scripted approach has led to a dataset which does not show just ordinary human actions involving the interaction with a single object but rather on natural multitasking which occurs, for example, when a person washes some dishes while cooking. Besides, being able to recreate a scenario similar to what actually occurs in the real world, EPIC-Kitchens gives a unique and novel perspective on how individuals interact with objects taking into account their attention, their intention (which may change over the course of the recordings) and even unexpected events. This makes EPIC-Kitchens a more realistic but also more challenging dataset. According to [9], EPIC-Kitchens has videos for a total of 55 hours and 11.5 millions of frames along with bounding boxes surrounding the objects the user interact with. Another peculiarity of this dataset concerns the employed annotation technique: participants were asked to narrate their own recordings afterward and this approach can reveal the actual intentions of each user. In fact they are the more qualified than an external observer to label the recorded actions; besides, the participants are not disturbed by this activity while they are taking videos since the authors opted for a post-recording narration [9]. After this coarse annotation, the authors collected manual transcriptions and start/end timestamps of each action using Amazon Mechanical Turk. The free text annotation performed by

Figure 3.4. Head-mounted GoPro used to get the recordings [9]

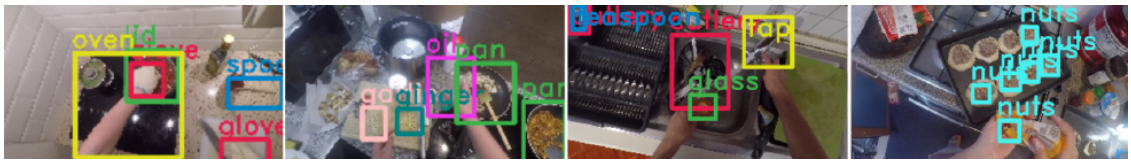


Figure 3.5. Example of object annotation [9]

participants consists of a huge variety of nouns and verbs. This means that the same action could be referred to using two different verbs by two different participants. In order to be coherent with the typical multi-class approach where each sample of the dataset is classified as belonging to a single class, the authors tried to cluster all the collected verbs into semantic classes: firstly, they tried to obtain automatic clustering via WordNet/Word2Vec combined with Part-of-Speech techniques to distinguish nouns and verbs but, in the end, they performed a manual clustering as a result of the many issues encountered with the first approach. EPIC-Kitchens is an innovative dataset also because the authors decided to monitor the community's progress in the First Person Action Recognition field proposing a series of challenges annually. Besides, this dataset is allowing deeper development in the application of multi-modal techniques and this is a novel aspect not to be overlooked.

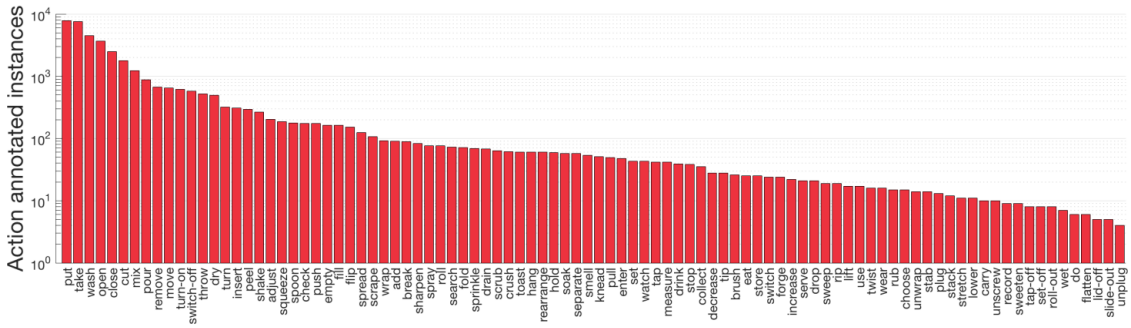


Figure 3.6. Frequency plot of verb classes in EPIC-Kitchens dataset [9].

3.5 The multi-modal approach

In this section we provide an overview of the multi-modal approach which is a set of techniques that allow a model to take into account a variety of input modalities to solve a task. This approach promises more robustness [72] and it is fundamental in real-world scenarios.

3.5.1 Multi-modal learning

In recent years, a variety of deep learning algorithms focusing on first person action recognition have been created. Several researches have been conducted in an attempt to improve the performances employing two or more data sources (e.g., audio and video) [73, 74, 72]. The interest towards multi-modal deep learning is growing more and more and this is aided by the availability of huge multi-modal datasets collected through the employment of many sensors [10] such as depth cameras.

The process of extracting features from more data streams, of different nature, living in different spaces, and learning how to combine and fuse them is referred to as multi-modal learning. The research in this sense has grown significantly during the previous decade in a variety of fields, most notably Computer Vision [10]. The rising capacity of deep learning algorithms and multi-modal data streams has led to the creation of models capable of uniformly processing, integrating and interpreting heterogeneous data. Unstructured real-world data may take on a variety of formats, referred to as modalities, which frequently include visual, audio and even textual information [10]. The advantage of this approach is that in many situations, a

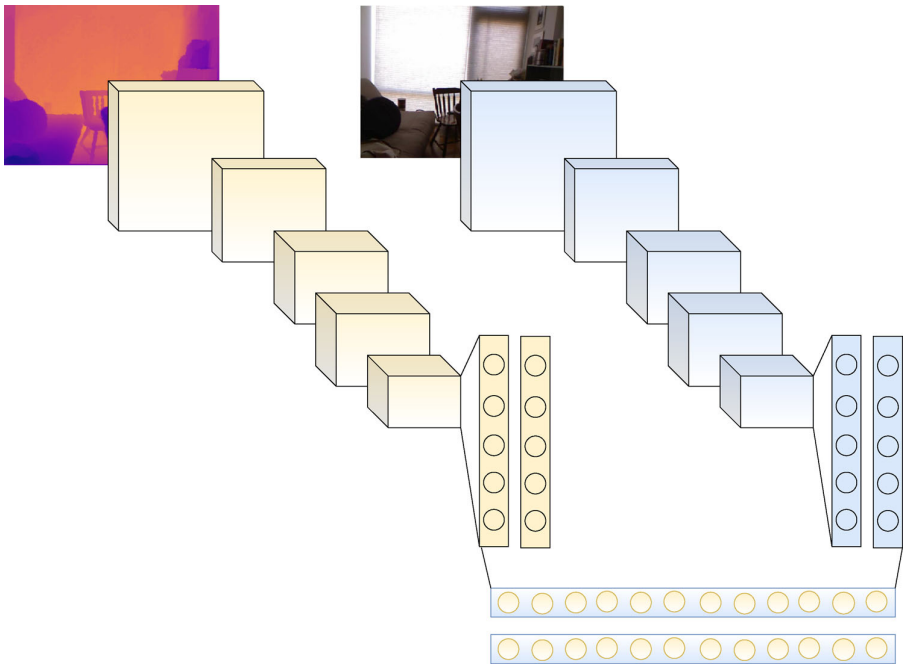


Figure 3.7. General structure of a bimodal CNN receiving RGB and depth images as input modalities [10].

collection of disparate inputs from different modalities and sensors might provide extra knowledge about the context of a task: for example audio-visual modalities, if correctly combined, can be complementary to each other in many contexts. However, while integrating multiple modalities to increase the accuracy of a model is an appealing approach, the challenge is to isolate possible noise and avoid conflicts between different input streams minimizing biases associated with their heterogeneity. Besides, the present literature's dearth of labelled multi-modal datasets might result in less accuracy and flexibility. What is encouraing, though, is that day by day there is more and more visual data available and this is the result of the widespread of devices of any sort. This means that it will be more and more frequent the collection of data from several sensors resulting in significantly better results than the ones obtained using a single input data flow. The potentiality hidden in the multi-modal approaches can be unlocked also thanks to the tecnhnological progress that has led to very fast and powerful GPUs.

The multi-modal approach has been widely employed in literature [7, 29, 6, 75, 76]. RGB data streams are usually combined with optical flow ones since the latter encode motion information which can be complementary to

the former. While optical flow has been shown to be a powerful modality in the action recognition context, its computational cost prevents it from being employable in online applications. This issue is addressed in more ways.

- There can be adopted strategies which avoid computing optical flow at test time still preserving the performances of two-stream architectures. In [77] this is done training a 3D CNN using RGB data minimizing the distance between the features from the layer before the network's final fully connected layer and the features from the motion stream.

- Original approaches using RGB (or, in general, single-stream architectures) can be studied. The network developed in [68] is able to jointly exploit motion and appearance knowledge from a RGB input stream. This is done by exploiting a self-supervised motion task at training time.

- RGB can combined with other modalities other than optical flow. For example, audio modality has been demonstrated to be effective in the egocentric context [30, 78, 79].

3.5.2 Fusion algorithms

In order to exploit the information carried by multiple data streams, attention must be payed in the techniques used to create a joint embedding. This is a delicate and crucial task. For example, an image can be difficultly defined using non-visual concepts while a textual representation is, by nature, more sparse and so, combining these two modalities in an efficient way is not an ordinary task. This is the reason why much importance must be given to the development of multi-modal fusion approaches.

There are three common techniques [10] used to tackle the fusion of different data streams.

- *Early fusion*: low-level features, extracted separately for each modality, are fused before the employment of a classifier which gives a prediction basing only on the fused features. The extracted features are usually very different in their appearance.

- *Late fusion*: each modality is treated separately performing both a feature extraction and a classification. Afterwards, the different predictions can be combined to give a final prediction using several techniques such as majority voting [80] or low-ranking [81].

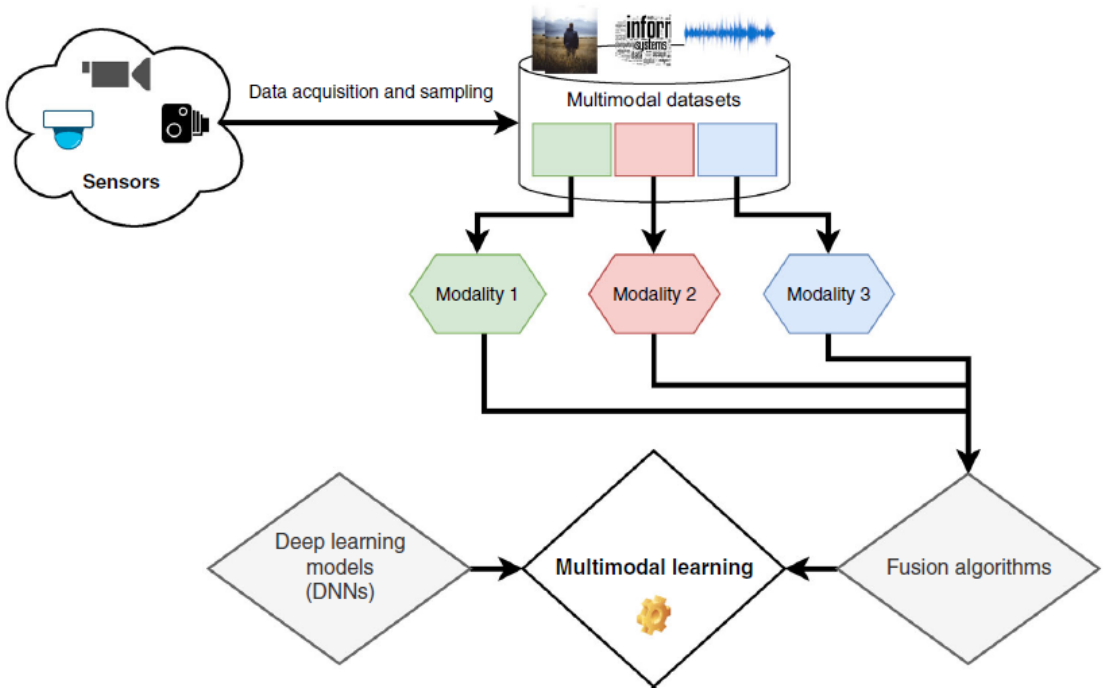


Figure 3.8. Example of a typical multi-modal pipeline with three input modalities [10].

- *Hybrid fusion*: in this case the different modalities are first treated with an early fusion approach and then with a late fusion one. As a result different predictions scores are created and they are finally combined together.

3.5.3 Pitfalls and promises

Deep learning has demonstrated its capacity to exceed human knowledge in a variety of domains over the last decades [10]. These algorithms employ a huge quantity of nonlinear processing units to extract and manipulate feature vectors from raw input [10]. The tendency towards the employment of deep learning in a broader variety of applications has grown recently [10] and this means that it will be more and more important to develop algorithms directly usable in real-life contexts. Indeed, multi-modal deep learning, along with the most innovative scene-content analysis techniques, is still limited in this respect and the community is pursuing a more favourable trade-off between

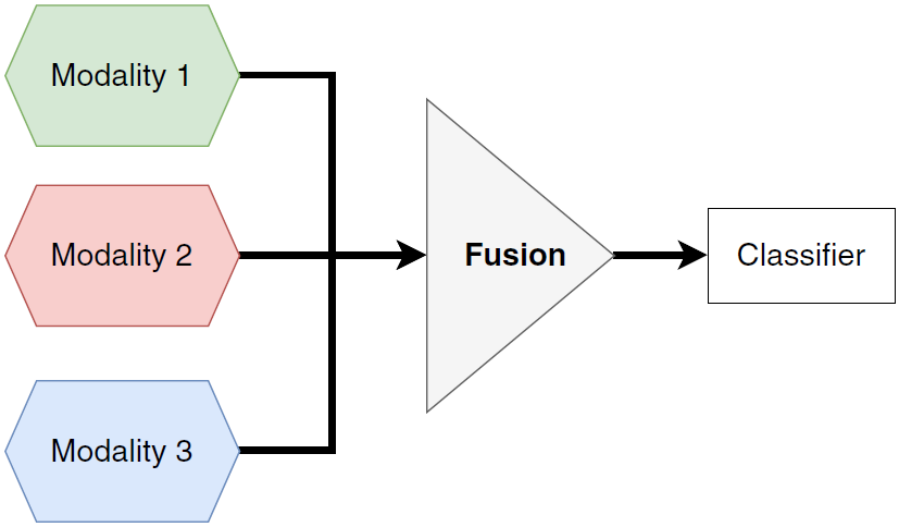


Figure 3.9. Representation of early fusion approach [10].

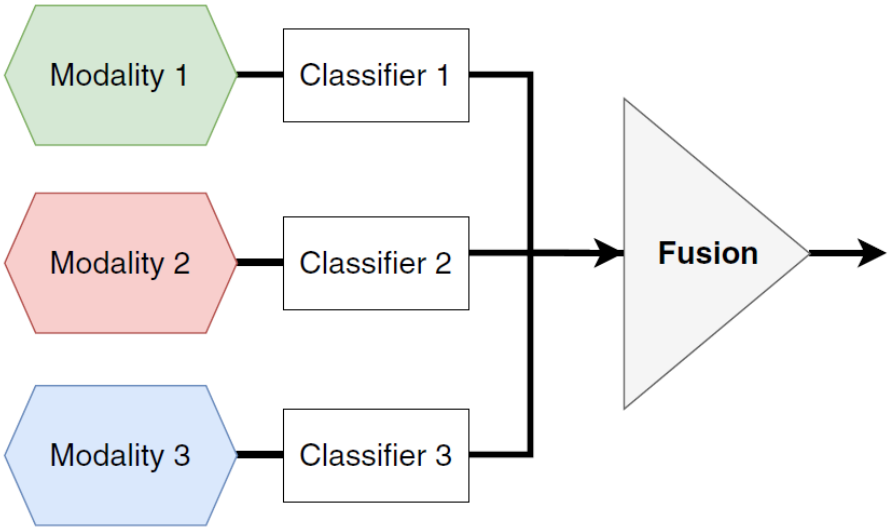


Figure 3.10. Representation of late fusion approach [10].

the complexity of the models, the computing power need, the memory consumption and the real-time processing capability [10]. The main challenges can be summarized as reported in [10]:

- Conflicts between different data sources can arise since they can be provided in a wide variety of forms. This is an obstacle for the extraction

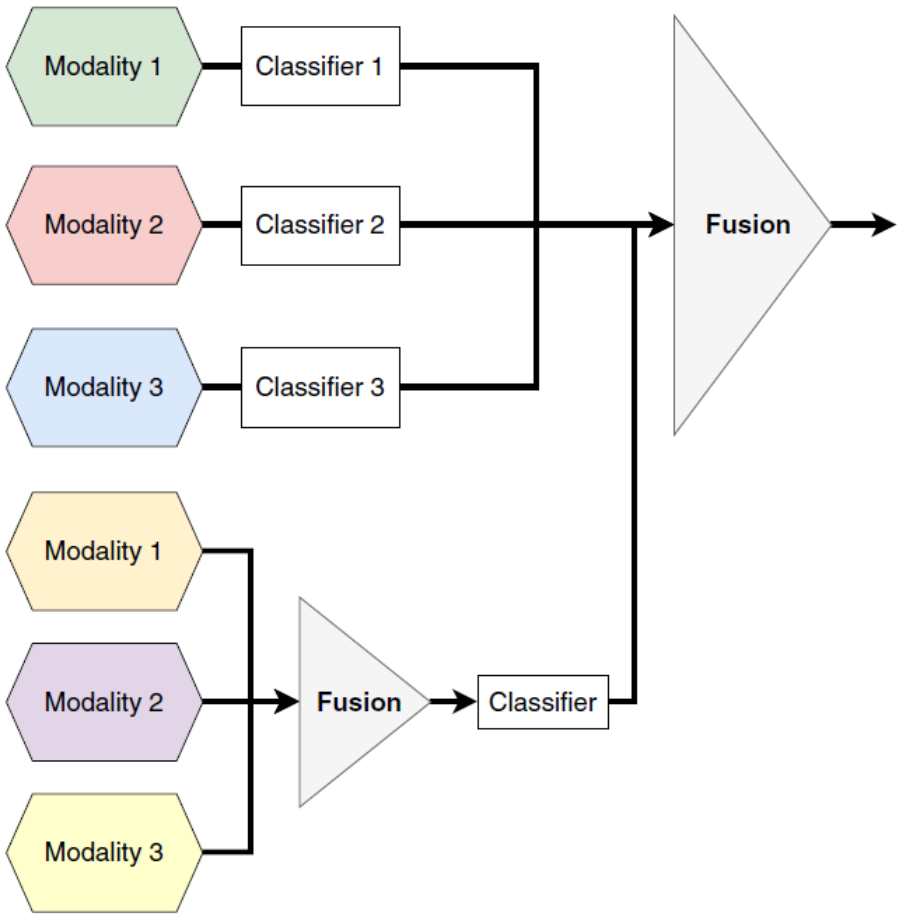


Figure 3.11. Representation of hybrid fusion approach [10].

of useful features.

- Large datasets are needed or the performances of the model would be severely affected by this lack.
- The primary challenge remains the attempt to minimise the computational power required by the complex multi-modal algorithms without affecting their accuracy in order to make them more scalable and more ready for real-time and real-world applications.

In the context of multi-modal FPAR, the most common datasets include the visual modalities of RGB, depth and flow. The attention on these three modalities is justified by the fact that it has been demonstrated, in many researches [10, 82], that their combination produces the best results reached

in literature. The optical flow modality was introduced by Horn et al. [83] in 1981; it captures the motion in a video and the fact that this kind of data has been successfully employed for more than three decades is a demonstration of how motion is a precious source of information useful in solving many real-world issues. The optical flow estimation relies on two assumptions [82]:

- *Brightness constancy constraint.* Along the motion trajectory, the pixel intensity remains constant. In formulas, if a pixel is moving of a quantity Δx and Δy respectively along the x-axis and y-axis in a time interval Δt, it holds true that

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{3.1}$$

  where I is the brightness.

- *When motion occurs, it is small.* In the video we are handling the motion must be smooth and it locally appears as a translation; this assumption allows us to use the Taylor series expansion on I.

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{3.2}$$

Substituting 3.1 into 3.2 and dividing all the terms by Δt we obtain the following equation:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{3.3}$$

where V_x and V_y are the two components of the optical flow of I [84]. To determine the optical flow, an extra set of equations is required: several methods in literature introduce additional constraints in order to estimate the real optical flow.

The optical flow is a robust modality when changing domain [30] because, unlike RGB modality, it does not focus on the colors or textures of an image (features that can change a lot from one context to another). However, the big drawback is that it cannot be used in real-time applications. This is one of the reasons why, in our work, we have focused on the event modality (which will be thorough in the next chapter). In fact, this modality appears similar to the flow modality as we can see in Figure 3.13 and it has many practical advantages that make it employable for real-time applications: events are simply generated by particular cameras.

Figure 3.12. Example of flow modality sample from EPIC-Kitchens dataset. It is not too much influenced by the appearance of the images.

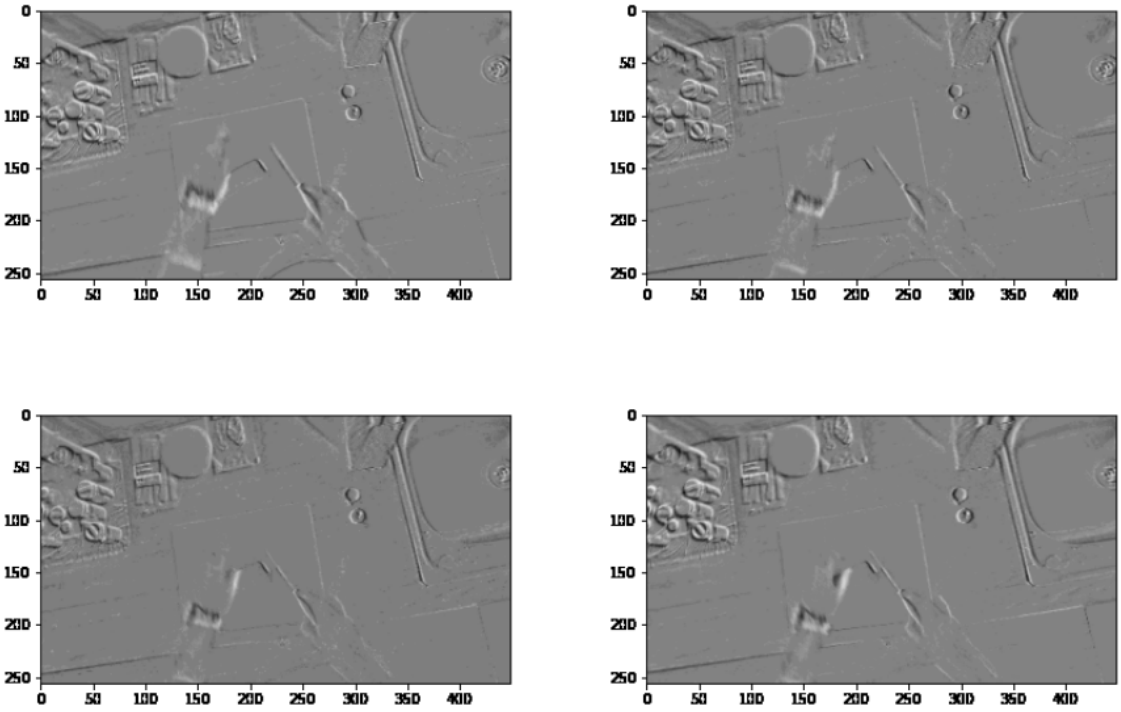


Figure 3.13. How event modality looks like once processed (notice the similarity with Flow modality).

3.6 Architectures

The recent advancements in convolutional neural networks and the availability of large-scale video benchmark datasets has brought deep learning approaches to dominate the area of video action recognition by utilizing 2D-CNNs, as in [12, 6, 85], 3D-CNNs such as [7, 86, 8], a combination of the two (for example [87, 62]) or bulding on already existing architectures structures which grasp the temporal information such as [11]. While 2D CNNs execute temporal modeling independently of 2D spatial convolutions, their 3D counterparts use 3D convolution to learn both space and time information.

For the sake of this work, we will briefly describe the models used in the benchmarking, noting that their arrangement for multi-modality and event data will be discussed in the thesis's second part.

3.6.1 Temporal Segment Network

According to the authors of [6], there is a substantial obstacle to using CNNs for video-based action recognition: comprehending video dynamics needs observation of long-range temporal patterns. Numerous approaches rely on intensive temporal sampling with a fixed interval of sampling (*dense sampling*). When used to prolonged video sequences, these algorithms would incur an excessive computing cost, limiting their application and introducing the risk of missing vital information for videos that exceed the sequence's maximum time. To address this issue, in [6] it is introduced the temporal segment network (TSN), which aims to anticipate at the video-level by utilizing the visual information included in the entire video. To do this, TSN uses a sparsely sampled succession of video samples, each of which generates its own early prediction of the event. Finally, a consensus is reached among the snippets in order to make the final video-level forecast, see Figure 3.14 for a graphical representation. Mathematically, if we have a video V which has been divided in K segments $S_1, S_2, ..., S_K$ of equal duration then the results of the prediction is computed as follows:

$$TSN(T_1, T_2, ..., T_K) = \mathcal{H}(\mathcal{G}(\mathcal{F}(T_1; \mathbf{W}), \mathcal{F}(T_2; \mathbf{W}), ..., \mathcal{F}(T_K; \mathbf{W}))) \quad (3.4)$$

where:

- $(T_1, T_2, ..., T_K)$ is the sequence of snippets, each of which has been sampled from its corresponding segment S_k;

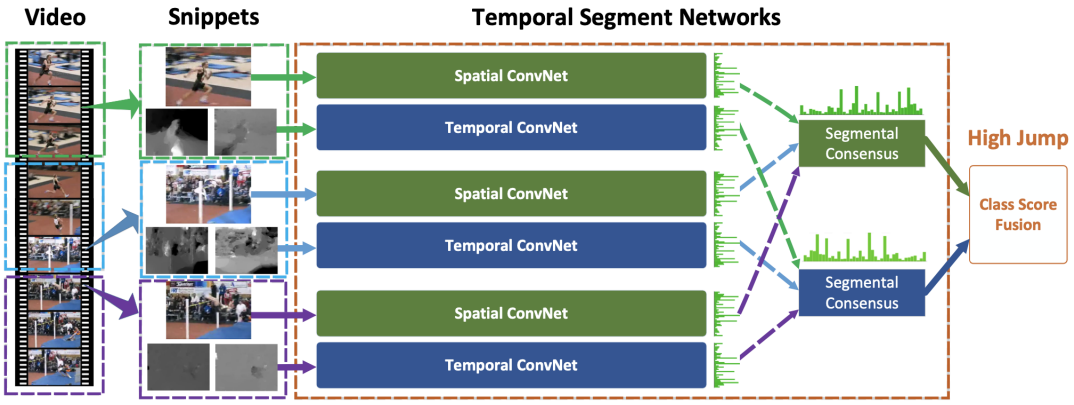


Figure 3.14. Temporal segment network. In this case the ultimate forecast is created by fusing predictions from more modalities [6]

- $\mathcal{F}(T_k; \mathbf{W})$ is the function obtained by passing the snippet T_k through the CNN with weights $\mathbf{W}$;
- $\mathcal{G}$ combines the outputs of the snipped passed through CNNs ($\mathcal{F}(T_k; \mathbf{W})$) to obtain the final consensus which will be called $\mathbf{G}$;
- $\mathcal{H}$ predicts the probability of each action class for the whole video. In the case of the [6] the authors have decided to use a classical *Softmax* function);

In terms of the loss on the final consensus, when a conventional cross-entropy loss is considered, it becomes:

$$\mathcal{L}(y, \mathbf{G}) = -\sum_{i=1}^{C} y_i \left(G_i - log \sum_{j=1}^{C} \exp\, G_j \right) \tag{3.5}$$

with C as the number of possible outputs/actions and y_i the true label for sample i. The consensus functions G employed in this work are straightforward aggregation functions (maximum, evenly and weighted averaging) that infer the final consensus from the scores of the same class on all the snippets. If the consensus function is chosen correctly, the segment network is differentiable, allowing for the simultaneous optimization of model parameters utilizing all snippets via the following loss function:

$$\frac{\partial \mathcal{L}(y, \mathbf{G})}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{G}} \sum_{k=1}^{K} \frac{\partial \mathcal{G}}{\partial \mathcal{F}(T_k)} \frac{\partial \mathcal{F}(T_k)}{\partial \mathbf{W}} \tag{3.6}$$

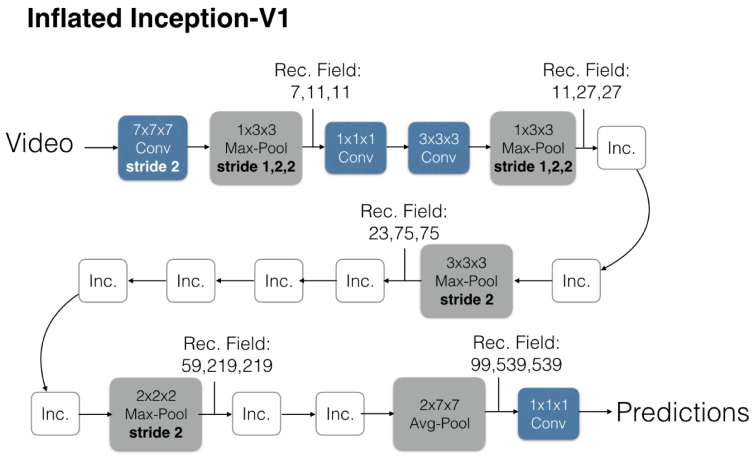


Figure 3.15. The Inflated Inception-V1 architecture [7]

In this way TSN can learn parameters from the entire video and not just from a single snippet.

3.6.2 Inflated 3D ConvNets

Beginning with one of the most well-known models used in the field of action recognition, Inflated 3D ConvNets (I3D) from [7], which has served as a "gatekeeper" baseline against which any newly published approaches can be compared. Given that pre-training always improves performance, the goal of I3D is to leverage pre-trained models on a variety of image datasets in order to achieve excellent performance in the video domain as well. I3D builds on existing image classification architectures, but inflates their filters and pooling kernels (and optionally their parameters) into three dimensions, resulting in extremely deep, naturally spatio-temporal classifiers. 3D CNNs appear to be a natural approach to video modeling; they are similar to ordinary convolutional networks but include spatio-temporal filters. They share a crucial property: they generate hierarchical representations of spatio-temporal data directly. However, the main disadvantage of these models is that they contain many more parameters than 2D CNNs due to the addition of the kernel dimension, making them more difficult to train. Additionally, they appear to preclude the benefits of ImageNet pre-training, and hence earlier works like [60, 88, 89, 90] has created and trained rather shallow customized architectures from scratch with not good performances. Thus, rather than

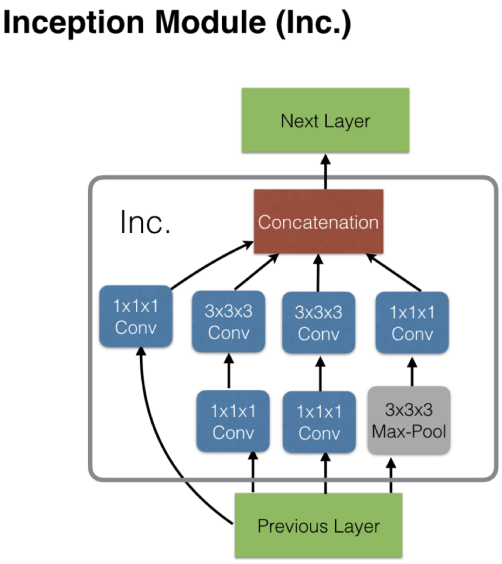


Figure 3.16. Details on the inception submodule [7]

restarting the procedure for spatio-temporal models, the goal is to simply move successful image classification models in two dimensions (2D) to three dimensions (3D) (3D CNNs). This can be accomplished by starting with a two-dimensional architecture and expanding all filters and pooling kernels, endowing them with a temporal dimension. Typically, filters are square, and they are simply converted to cubic. Basically $N \times N$ filters become $N \times N \times N$. Even if the technique for expanding the filters and pooling kernels is straightforward, doing so would miss the purpose of reusing the parameters. Thus, in addition to the architecture, the inflation logic is to bootstrap parameters from ImageNet pretrained models. To accomplish this, consider how a picture can be transformed to a (boring) video by continuously copying it into a video sequence. The 3D models can then be implicitly trained on ImageNet by meeting what is referred to in the paper as the boring-video fixed point: the pooled activations on a boring video should be identical to those on the initial single-image input. Due to linearity, this may be accomplished by repeating the weights of the two-dimensional filters N times along the time dimension and rescaling them by dividing by N. This ensures that the response of the convolutional filter is identical. Because the outputs of convolutional layers are constant in time for boring movies, the outputs of pointwise non-linearity layers, average and max-pooling layers are identical

to those in the 2D case, and so the total network response respects the fixed point for boring films.

3.6.3 Temporal Relation Network

The authors of [11], observing the relevance of temporal relational reasoning for activity recognition on both short- and long-term timescales, propose a new module, called Temporal Relation Network (TRN), with the aim to fill this gap in the literature. It is used to characterize the temporal relationships between video observations and is a generic and adaptable module that plugs into any current CNN architecture. Starting from the definition of pairwise temporal relation as it follows:

$$T_2 = h_\phi \left(\sum_{i<j} g_\theta(f_i, f_j) \right) \tag{3.7}$$

with V being the input video, $(f_1, f_2, ..., f_n)$ being the features extracted by a standard CNN from the corresponding n frames of the video, g_θ and h_ϕ as the functions which fuse frames features at different levels. The authors extended this concept to higher frame relations, e.g. 3-frame as it follows:

$$T_3 = h'_\phi \left(\sum_{i<j<k} g'_\theta(f_i, f_j, f_k) \right) \tag{3.8}$$

Finally, all the previously mentioned notions are used for capturing temporal relationships at multiple time scales, by accumulating all frame relations at different scales in the following manner:

$$MT_N(V) = T_2(V) + T_3(V)... + T_N(V) \tag{3.9}$$

where each relation term T_d captures temporal relationships between d ordered frames and has its own $h_\phi^{(d)}$ and $g_\theta^{(d)}$. A schematic example of the module can be grasped from Figure 3.17.

3.6.4 Temporal Shift Module

The authors of [12] observes that the main problem of 2D CNNs is that, on individual frames, they are incapable of accurately modeling temporal information. While 3D CNNs can learn spatial and temporal information concurrently but their high computational cost makes deployment on edge

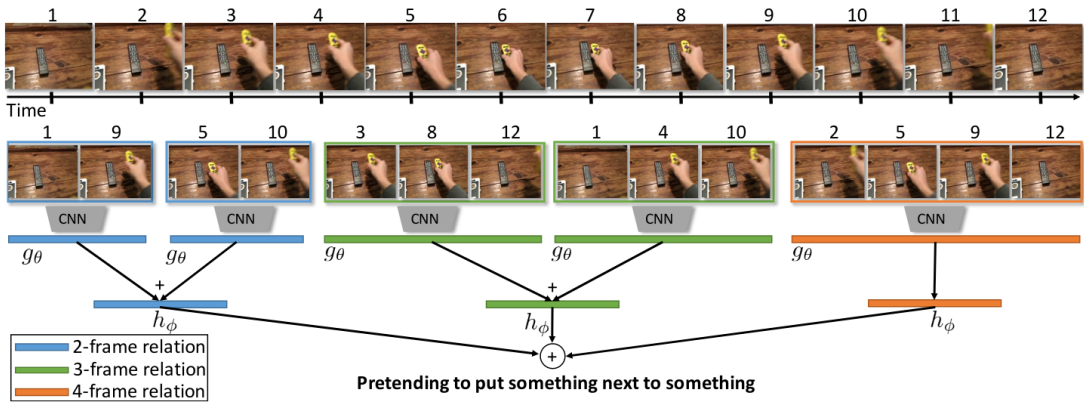
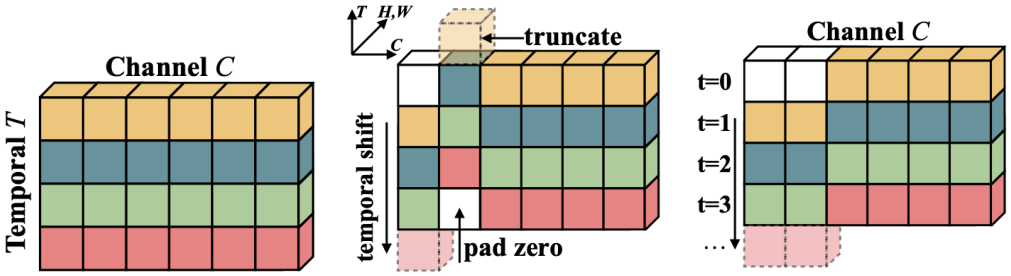


Figure 3.17. TRN module [11]

devices challenging; so they cannot be used for real-time online video recognition. There are techniques for balancing temporal modeling and computation, for example, post-hoc fusion and mid-level temporal fusion. These methods sacrifice low-level temporal modeling in favor of efficiency, but a significant amount of important information is lost during feature extraction prior to temporal fusion. The Temporal Shift Module (TSM) is designed with the intention of shifting the activation channels along the temporal dimension, both forward and backward, see Figure 3.18. Indeed, consider a 1D convolution process with a kernel of size 3 and weights $W = (w_1, w_2, w_3)$. If the input X is a one-dimensional vector of indefinite length, the operation's outcome can be expressed as $Y_i = w_1 X_{i-1} + w_2 X_i + w_3 X_{i+1}$. Two steps can be taken to decouple the processes required to achieve Y: *shift* and *multiply-accumulate*. The input is shifted by -1, 0, $+1$ and multiplied it by w_1, w_2, w_3, the sum of which is Y. The initial step shift can be performed without the use of multiplicators. The second stage, on the other hand, is more computationally expensive.

TSM's idea is to move the time dimension and fold the multiply-accumulate from the time dimension to the channel dimension, allowing for very low-cost temporal reasoning. The primary concerns that arise from the use of such a module are as follows:

- Decreased efficiency as a result of significant data migration. While the shift operation does not require any computing, it does entail data movement. Data mobility increases the memory footprint and inference time on the hardware;



(a) Original tensor (b) Bi-direction temporal shift (c) Uni-direction temporal shift

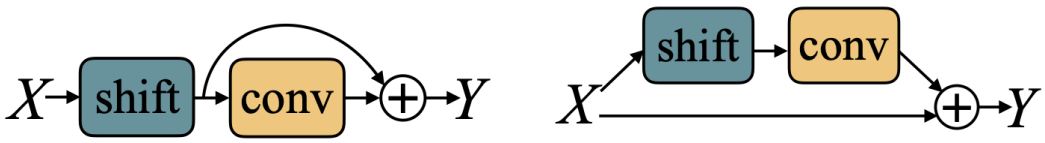
Figure 3.18. Temporal Shift Module (TSM) functioning [12]

- Performance deterioration due to a reduced ability to model spatially. By relocating a portion of the channels to neighboring frames, the information contained in the channels becomes inaccessible for the current frame, which may impair the 2D CNN backbone's spatial modeling capabilities;

To address the first issue, a partial shift approach (e.g. 1/8 of the channels) is used to reduce memory transfer costs dramatically. Then, as indicated in Figure 3.19(a), a straightforward way to use TSM is to place it before each convolutional layer or residual block (in-place shift). However, it impairs the backbone model's ability to learn spatial features, particularly when a high number of channels are moved, as the information stored in the shifted channels is lost for the current frame. Thus, rather than inserting the module in-place, it is placed inside the residual branch in a residual block to balance the model's capacity for spatial and temporal feature learning. This is referred to as residual shift and is seen in Figure 3.19(b). Residual shift can be used to address the problem of poor spatial feature learning, as all information contained in the initial activation is still accessible following temporal shift via identity mapping.

There are two ways to achieve the final prediction:

- **Offline model with bi-directional TSM**: given a video V, we sample T frames from the video $F_1, ..., F_T$. Following frame sampling, 2D CNN baselines process each frame individually and then average the resulting logits to produce the final forecast. The TSM model has the exact same parameters and computing cost as the two-dimensional model. The distinction is that TSM is added for each residual block, which permits



(a) In-place shift (b) Residual shift

Figure 3.19. Different ways of using TSM [12]

no-computational fusing of temporal information;

- **Online model with unidirectional TSM**: offline TSM bidirectionally changes a portion of the channels, requiring features from future frames to replace those in the current frame. Rather of that, the online model simply shifts the feature from prior to current frames (see Figure 3.18(c)), enabling online recognition with uni-directional TSM;

3.7 Cross Domain Analysis

First Person Action Recognition is increasingly gaining the scientific community's attention [67, 76, 30, 64]. This is due to the growing popularity of wearable cameras along with the multiple applications that this topic can have in real-world scenarios ranging from industry sector to assistive technologies. Besides, the release of the EPIC-Kitchens dataset [9] has greatly encouraged and enhanced the research in First Person Action Recognition.

The convolutional neural networks are state of the art in this context [29] and they can be divided into two categories: 2D [6, 12, 61, 11] and 3D [7, 63, 64, 65, 91] methods. The first crucial works [60, 88] used only the RGB modality but in 2014, for the first time, a multi-modal approach was used [5]: Simonyan and Zisserman faced the lack of motion features that occurs when using only RGB data by adding a second input stream consisting of the Optical Flow modality. It turned out that the latter approach outperformed the former one: since then, the multi-modal approach, generally combining RGB with optical flow or audio, has become the most popular technique in First Person Action Recognition [29, 6, 9, 92]. Recently, more complex architectures, detailed in the previous sections have been developed [12, 6, 8, 93].

The incredible interest in FPAR context is has also brought to light some

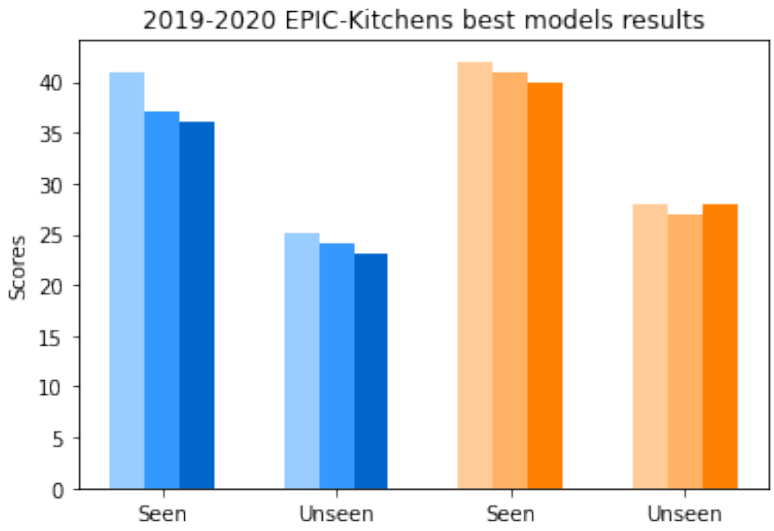


Figure 3.20. Results of the 3 best models in 2019 [13] and 2020 [14]. A drop in performances occurs when testing on unseen kitchens.

cross-domain issues: in fact when a model is trained in a certain environment but used in a different one, we witness a big drop in performances because the model has an intrinsic "environmental bias". This makes a model unable to generalize on unseen domains that can be, for example, real-world scenarios related to important applications. This phenomenon is more evident in egocentric settings than in third person ones and to have an idea of its extent it is sufficient to compare the performances of the best model of the EPIC-Kitchens challenges on seen and unseen domains [14, 13].

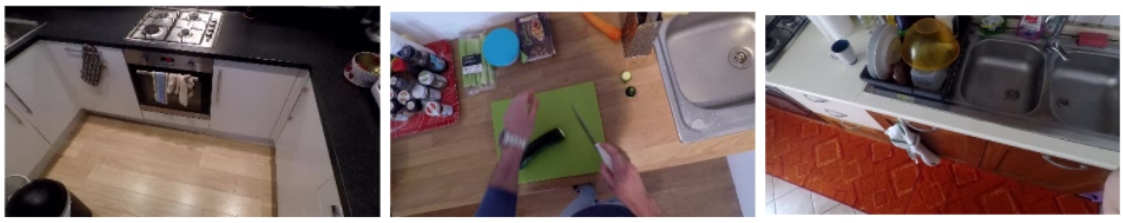


Figure 3.21. Example of 3 different kitchens in the EPIC-Kitchens dataset. Differences in colors, textures and appearance cause the domain shift.

This issue, also called domain shift, has been addressed by several studies

in literature especially with Unsupervised Domain Adaption (UDA) techniques. UDA has the goal of making a model, previously trained on a labelled source domain, able to perform well on an unlabelled target domain. These techniques are widely used in other vision tasks [94, 95, 96] and only recently some improvements are being made in the FPAR context [31, 32]. Some works [33, 29] have also used a multi-modal approach for UDA.

Nevertheless, none of these approaches were designed to handle cross-domain settings. Indeed, in the UDA settings, the target domain, even if unlabelled, is avaiable at training time but this situation is not realistic at all: in real-world contexts the target domain is often unknown or having access to it in the model development phase could be very expensive. For this reason, in this thesis, one contribution is the creation of a large benchmark showing how different modalities perform when used in single- or in multi-modal fashion in both intra- and cross-domain scenarios. We would like to emphasize the fact that a benchmark should also analyze the ability of a modality to generalize in order to provide a fair and trustworthy evaluation of a model which takes into account its ability to generalize on unseen domains. In particular, as will be clear in the second part of our work, we have focused on the event modality which can be complementary to the RGB one (hence, reducing the domain shift) since it holds motion information, Besides, the event modality does not suffer from problems such as the high computational power required which would make it impossible to use in online scenario as it is, instead, for the Optical Flow modality.

Chapter 4

Dynamic Vision Sensor

The purpose of this chapter is to provide an overview of neuromorphic approaches to computer vision: this new paradigm, inspired by brain structures, can lead to systems which are successful where conventional cameras fail. The first section discusses the history of these novel devices, their functioning, and their benefits, with a particular emphasis on the DVS camera. Afterwards, in order to demonstrate the potential of this innovative way of sensing, we report some interesting real-world applications and we describe how events are treated in this work. The final section is devoted to describe two architectures that we combined together in order to obtain the EPIC-Kitchen dataset's event extension. Even though not all the notions discussed here find direct application in our work, it is crucial to provide the reader with a comprehensive overview of this field so that he can better perceive the rational behind the choices we made throughout the work.

4.1 Introduction to Neuromorphic Approaches in Computer Vision

Traditional cameras collect visual information by recording frames at constant rates. As a result of this functioning, redundant useless data are collected: a frame is generated even if none of its pixels produce a brightness change and this behavior is typical, for example, of the background pixels in a scene. The consequence is that these devices are power hungry, require large storage capacities, hence the frame rate will always be severely limited by several technological constraints. The conventional cameras do not focus on the motion which is instead critical in a large number of scientific fields.

Figure 4.1. In a scene like this, conventional cameras would produce frames with redundant data about the background, circled in red, while neglecting the movement, circled in yellow, which is the core of the scene [15].

Neuromorphic Computer Vision has the objective to develop cameras able to mimic the computational efficient behavior of retinas which are a good example of how neuro-biological systems are able to asynchronously capture relevant information at an higher temporal resolution with a lower energy and memory allocation requirement [18].

4.2 History of Neuromorphic Devices

In 1991, in order to imitate the millions of point-to-point neural connections in the biological systems, the Caltech Institute introduced the Address-Event Representation protocol [97, 98] whose functioning is schematized in Figure 4.2.

According to AER standard, in an electronic camera designed to mimic the retina functioning, each memory address is uniquely associated to a single image pixel[1] and they all share the same digital bus on which they can asynchronously output events. The AER standard is, still today, at the base of modern event cameras as we will explain in the further section.

The first imaging device embodying this protocol was the silicon retina

[1] in general applications, the memory addresses are associated to units asynchronously spiking data

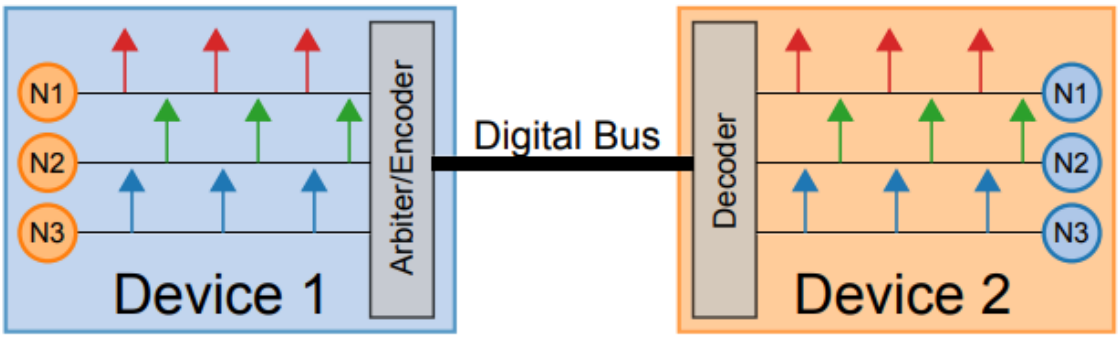


Figure 4.2. A general schematization of AER, the protocol that allows multiple devices to use the same bus. The arbitrer assigns distinct addresses to the neurons of the first device. When these neurons spike, the generated event is sent to the shared digital bus and the second device is able to reconstruct the information via decoder [16]

developed in 1991 by Mahowald and Mead [99]. A second milestone in this journey was the introduction, in 2005, of a more realistic silicon retina nevertheless having some weaknesses that did not allow it to be fully exploitable in real-world applications[100]. The turning point for what concerns practical applications was given by the realization of the Dynamic Vision Sensors [101]. Over the last decade, several new sensors have been developed and they all can be classified into two main categories based on how they operate.

The Temporal Difference devices are made of a matrix of independent pixels that generate data only when there are brightness changes. In other words, a pixel is triggered by an event occurring in its receptive field (e.g., a pixel viewing a static scene will not provide any event to the camera). This category is also known as event-based cameras and DVS falls into this class of devices.

The second-class devices are made of pixels consisting of two subpixels: the change detector (implemented by a DVS pixel) and the exposure measurement module. In this architecture the first subpixel, activated when it detects an illumination change, triggers the second circuit which measures the absolute intensity of that pixel. Coupling these two modules, the camera measures the absolute intensity of the changing pixels encoding this information through the generation of two events: the brighter the illumination, the shorter the time interval between them [21]. An example of this kind of devices is the Asynchronous Time Based Image Sensor [102] whose architecture

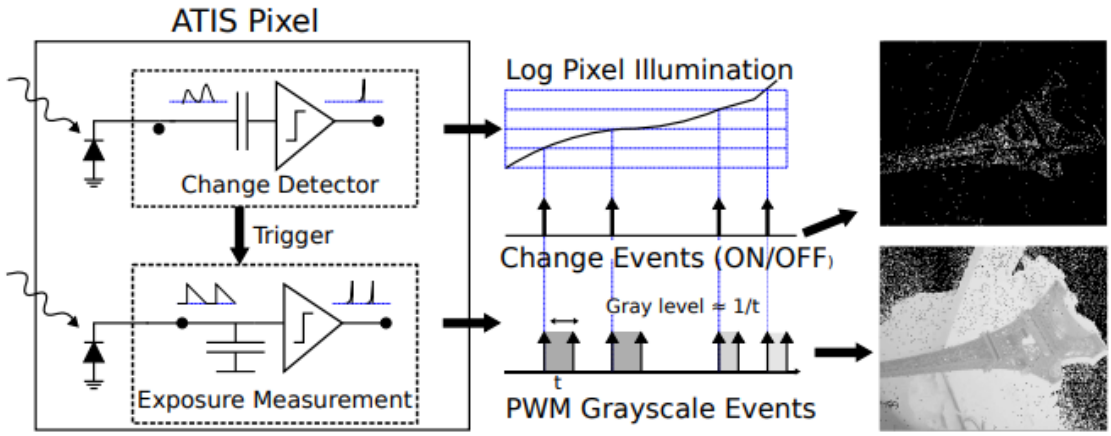


Figure 4.3. The architecture of the ATIS device showing the interaction between the Change Detector module and the Exposure Measurement one. The two diagrams in the middle refer to a camera pixel taken as example [17]

is depicted in Figure 4.3.

Nowadays, these devices are a commodity that all major companies around the world can build or buy. This will hopefully enhance the research in this sector since, despite the impressive progress made, biological retina is still far ahead of event-based camera in terms of precise timing, dynamic range and efficiency.

4.3 DVS Functioning

In this section we provide a more detailed description of the DVS camera since the stream of events it generates is what we will aim to simulate at a certain point of our work.

DVS are novel bio-inspired camera able to asynchronously detect pixel-wise brightness changes called "events". The result is a stream of data encoding time, pixel location and sign of the captured intensity changes. This stream is recorded at a variable data-rate: in general, the quicker the motion, the more events are produced. More specifically, every time a pixel in the camera detects a change in brightness that exceeds a defined threshold, an event is sent from the pixel matrix using a shared digital bus regulated with a multiplexing strategy [21]. A clear formalization of this process can

be found in [21]. A single event is so defined:

$$e_k \doteq (x_k, y_k, t_k, p_k) \tag{4.1}$$

where (x_k, y_k) is the coordinate of the pixel which generated the event at time t_k. In 4.1 the last value is called *polarity* $p \in \{+1, -1\}$ and encodes the sign of luminosity change. For each event e_k we can define the brightness change detected by the pixel at time t_k with respect to the last event occurred at the same pixel at time $t_k - \Delta t$:

$$\Delta L(x_k, y_k, t_k) \doteq L(x_k, y_k, t_k) - L(x_k, y_k, t_k - \Delta t_k) \tag{4.2}$$

$$L \doteq log(I) \tag{4.3}$$

In the above equations L represents the logarithm of the photocurrent[2]. An event is generated only if the brightness change exceeds a threshold C determined by the DVS. Formally:

$$\Delta L(x_k, y_k, t_k) = p_k C \tag{4.4}$$

Hence, the determination of the polarity:

- $p_k = 1$ refers to an ON event after an increase in brightness;
- $p_k = -1$ refers to an OFF event after a brightness decrease.

Depending on the C value, a camera can generate more or less events. As a rule of thumb, the less ideal the lighting conditions of the recorded scene, the higher C should be in order to prevent random noise from triggering useless events. Digging into math, we can linearize equation 4.4 assuming, for simplicity, a constant illumination [RFF 7]:

$$\Delta L \approx -\nabla L \cdot v \Delta t \tag{4.5}$$

This means that, considering small Δt, the change is produced by a brightness gradient travelling with velocity v across the image and, applying the dot product's rule, in 4.5 we are demonstrating that the maximum rate of event generation is reached when motion is perpendicular to an edge while no event is sent after a motion parallel to it. It is important to emphasise that what has been discussed so far is an ideal model of the DVS that does not account for possible transistor noise and mismatch. An example of events stream as output of a DVS sensor is depicted in Figure 4.4.

[2]electric current that flows through a photosensitive device

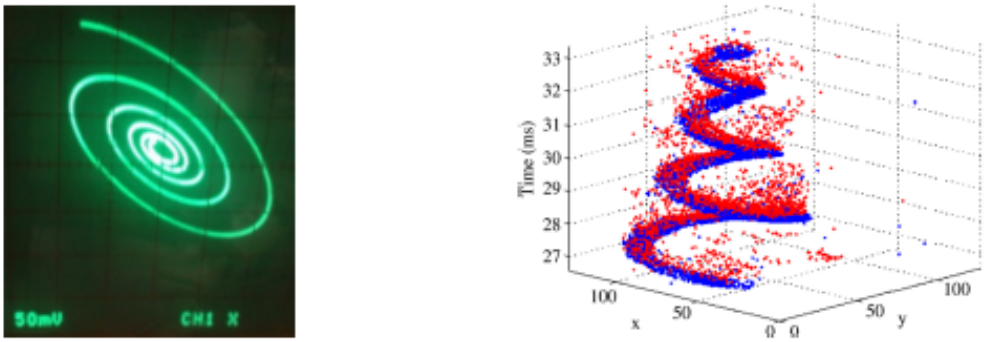


Figure 4.4. On the right we can observe the events generated by DVS camera put in front of an oscilloscope producing a spiral [18].

4.4 Advantages and applications

In this section we draw attention to the intrinsic technical advantages of event cameras with respect to traditional ones. These features can lead the way to novel and innovative real-world applications which were unfeasible before.

4.4.1 Technical specifications

This neuromorphic approach in the data acquisition process provides significant advantages over conventional sensors, particularly in low-light and high-speed motion conditions. Indeed, their high pixel bandwidth[3] reduces motion blur, a phenomenon which arises from their rapid and involuntary movements caused by the user, and their wide dynamic range[4] makes them an attractive alternative to traditional cameras in challenging robotics and computer vision scenarios. Moreover, the low latency and low power consumption of these novel devices enable their use in several new real-world applications, especially related to the field of wearable devices. The aforementioned peculiarities make them ideal for addressing well-known issues associated with the usage of wearable devices, such as continuous visual stimuli and background clutter. Of course, like any physical transducer, some technological constraints are present but they do not undermine the efficiency of

[3]events are timestamped with microsecond resolution [21]

[4]it is the ratio between the maximum and minimum measurable light intensities and for a DVS it is 140 vs. 60 dB of standard cameras [21]

these devices in the applications I'm going to mention in this section. There is a maximum rate beyond which the DVS pixel is not able to record light variations and this rate increases with the brightness intensity. Another limit derives from a possible saturation of the digital bus, influencing the time with which events are transmitted by the pixels.

4.4.2 Space-tracking

A fascinating field in which event cameras excel is space-junk tracking [19]. With each passing year, we become more aware of the number of pieces of old or broken satellites orbiting our planet, increasing the risk of collisions. An issue that must be addressed by mankind is therefore cleaning the space of these objects but the first step to achieve this goal consists in precisely identifying and tracking them: neuromorphic telescopes can help in this endeavour. In this context, the benefits of neuromrphic imaging can be fully leveraged for several reasons:

- because there are few changes in lighting when observing a starry sky, a frame-based camera would generate a large amount of useless data to be stored and processed;

- while traditioanl telescopes need observatories whose strucutre is perfectly stable, these innovative neuromorphic telescopes can be located in small, cheap and mobile containers. Indeed, they can tolerate motion and even benefit from it, as it help them in detecting moving objects in the space.

A result achieved by this novel telescope is shown in Figure 4.5. A similar application to the aforementioned one is star-tracking [103].

4.4.3 Quadrotors

Another interesting DVS application is introduced in [20].Before the realization of these quadrotors, the agility of a robot was highly limited by the latency of its perception pipeline. DVS can be used to overcome these limitations paving the way for novel future applications. A possible examples is the employment of super-agile quadrotors in dangerous rescue operations where the environment is prone to abrupt and unpredictable changes. Observing Figure 4.6 the high performances of these devices are crystal clear.

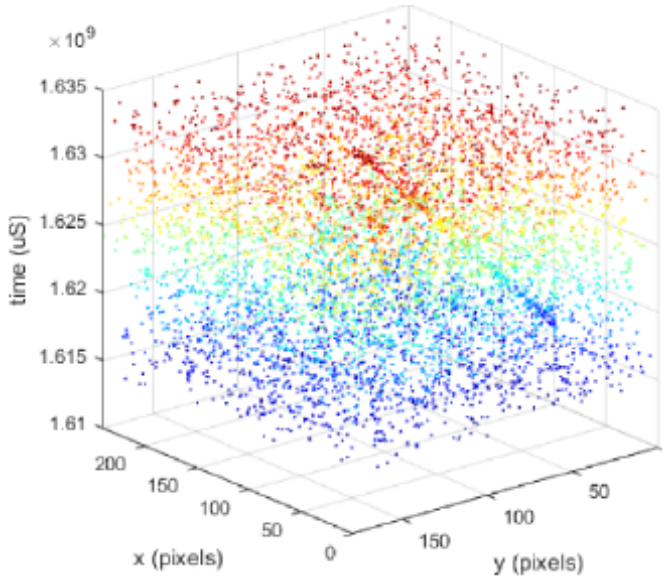


Figure 4.5. Illustration of the output of a neuromorphic telescope in an imaging mission of a geosynchronous satellite. The trajectory of the satellite appears in the form of a continuous line in the 3D events space [19].

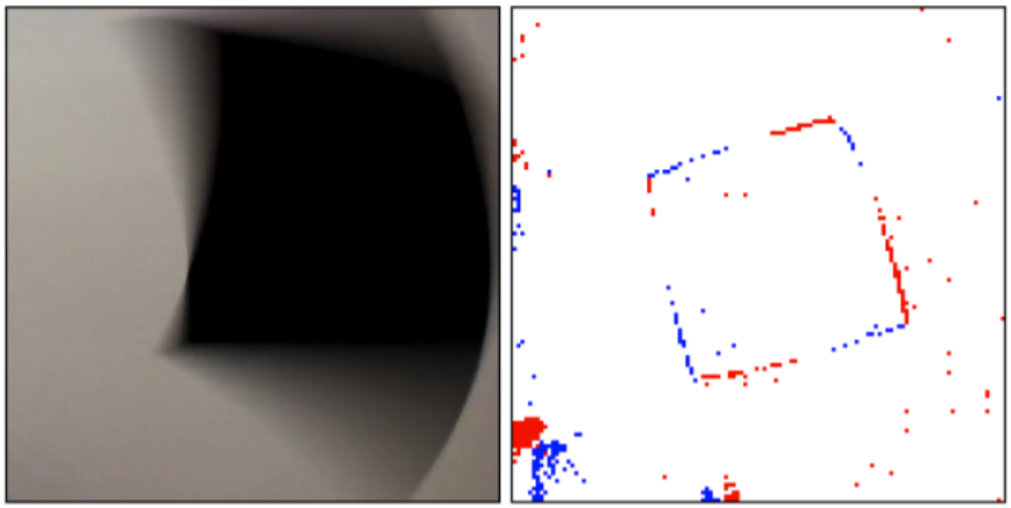


Figure 4.6. The output of two different quadrotors performing a flip. The left picture refers to a standard quadrotor (motion blur is present) while the one on the right derives from a quadrotor equipped with a DVS camera which is robust to fast motion [20].

Figure 4.7. Super-agile quadrotors could be in future employed in dangerous rescue operations where a human could difficultly operate [20].

4.4.4 Other applications

Other fields in which event-cameras are becoming widely adopted range from surveillance [104] to object recognition [105], passing by wearable electronics [106] and Simultaneous Localization and Mapping [107]. A sign of how pervasive this new technology can become is iniVation[5]: a company which designs and builds neuromorphic vision systems. Among their most successful products we can find an high speed 3D laser profiling employed in production quality controls, a micropower intelligent scene analysis for mobile and IoT and the world's fastest eye tracker.

4.5 Event Representation

Event sensors have posed a paradigm shift in the way visual information is acquired. As a result, they introduce the problem of developing innovative algorithms to process the acquired data and fully exploit the camera's capabilities. As highlighted in [21], the new challenges are consequences of the following facts:

[5]https://inivation.com/

- while traditional images are synchronous and spatially dense, the events stream is asynchronous and sparse. Hence, if we want to use traditional frame-based algorithms a preprocessing on the events is needed;
- DVS are intrinsically noisy due to their non-ideal transistor circuits;
- while conventional cameras record in grayscale, each event in this context carries binary information encoded in the polarity focusing on the motion.

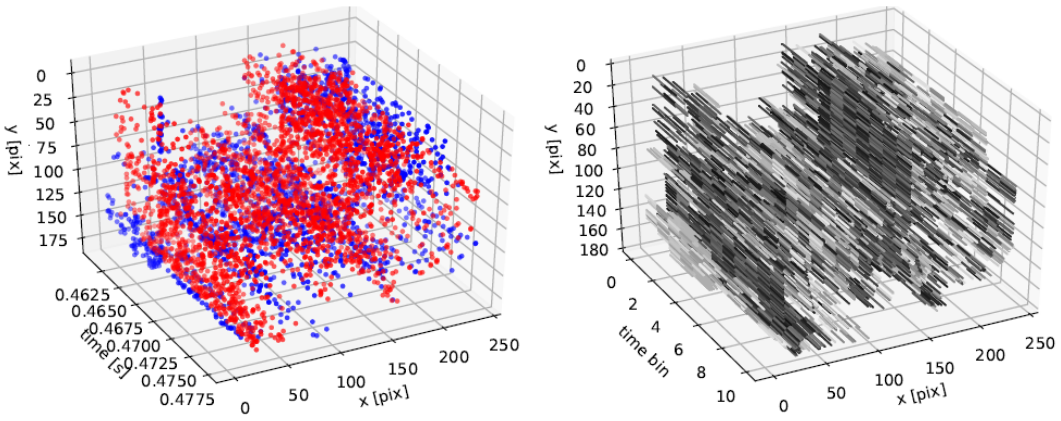
Over the last few years several studies have been conducted on how to deal with events data: the answer to this question is far from obvious as each task and each working context may require a different process. Looking at the literature today, we can divide [21] the algorithms employed to extract features from the events into two big categories depending on the number of events simultaneously processed:

- *Event-by-event methods.* Every time a new event is recorded, it is compared with the current state of the system and the representation is updated. This process results in minimal latency. A famous example of these methods are the Spiking Neural Networks and the probabilistic filters.
- *Packet of events methods.* An update of the representation occurs when a new temporal window, within which many events have been recorded, passes. The principle behind this methods is that a single event is very prone to noise, therefore only a group of events is able to carry useful information.

4.5.1 Voxel Grid Representation

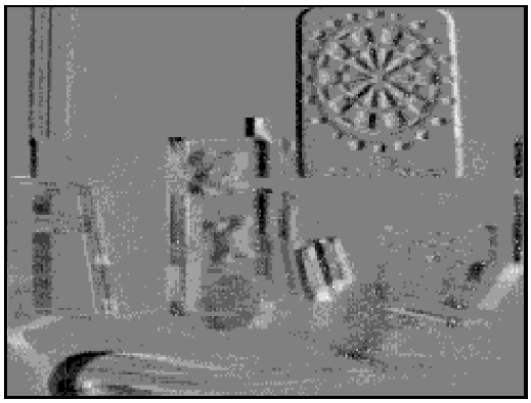
In this section we provide a detailed description of the *Voxel Grid* because it is the representation used in our work. This decision is motivated by the fact that Voxel Grid is one of the most performing representations in literature. Besides, it allows us to exploit, making the appropriate readjustments, all the pre-existing knowledge (both in terms of feature extractor architectures and, for example, domain adaptation techniques) in Computer Vision.

Voxel Grid [37] is an unsupervised representation of the events stream in the form of several discretized volumes which preserves the temporal distributions of the recorded events. The only parameter to fix is B which indicates



(a) Stream of events colored according to the polarity.

(b) Interpolated Voxel Grid with 10 bins.



(c) One of the ten resulting channels composing the final representation. The coloration follows the polarity (i.e., the darker it is, the more negative it is).

Figure 4.8. The operational flow leading to the creation of the Voxel Grid representation [21].

the number of volumes, called bins, we aim to obtain. The objective is to process a stream of N events $(x_i, y_i, t_i, p_i)_{i\in[1,N]}$ discretizing the time axis.

First of all, we rescale the timestamps t_i to the range $[0, B-1]$:

$$t_i^* = (B-1)(t_i - t_0)/(t_N - t_1) \tag{4.6}$$

A single event bin is created grouping the events using a linearly weighted

accumulation. Formally:

$$V(x, y, t) = \sum_{i}^{N} p_i k_b(x - x_i) k_b(y - y_i) k_b(t - t_i^*) \tag{4.7}$$

$$k_b(a) = max(0, 1 - |a|) \tag{4.8}$$

Voxel Grid generates an input volume made by B frames with the same length and with of the original events stream. Thus, the value of a pixel in each frame is determined using a technique similar to bilinear interpolation. More precisely, $k_b(a)$ serves as a bilinear sampling kernel [37].

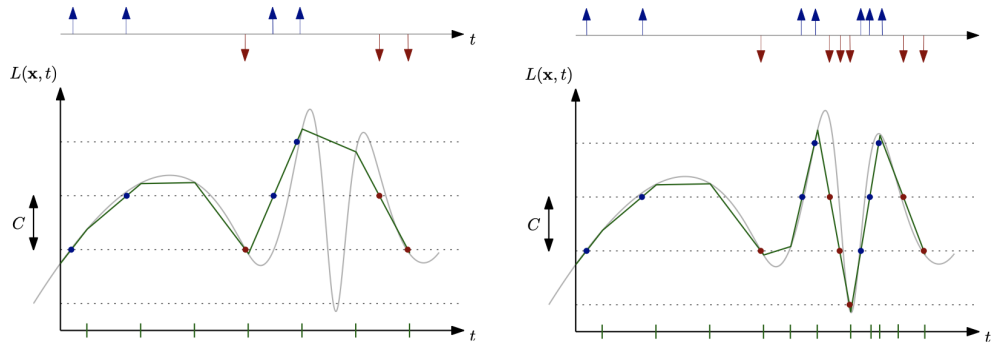
After detailing how this representation treats events, it becomes more evident that we are considering the time domain as the traditional channels in a 2D image: downstream architectures will have to be adapted in order to handle B-channels images instead of the typical RGB ones.

4.6 Event Simulation

Despite the enormous interest and promise of this data, event cameras remain scarce and expensive, impeding the research community's development. Furthermore, commercially available event camera hardware is still in the prototype stage and has a number of practical constraints, like low resolution, low signal-to-noise ratio, and complex sensor design that requires specialist expertise. To solve the research difficulties, there is a great demand for low-cost, high-quality **synthetic**, labeled events for algorithm prototyping, deep learning, and benchmarking. This chapter will detail the typical instruments that comprise the pipeline used to convert the classic RGB dataset into event ones. They enabled us to add a new synthetic event modality to the EPIC-Kitchen despite not having an actual event camera.

4.6.1 Event-camera Simulator

The authors of [22] has developed the first event camera simulator architecture (called ESIM) that intimately connects the event simulator with the rendering engine to enable accurate event simulation via an adaptive sampling technique. In fact, creating a simulator of this type is not straightforward, as event cameras operate fundamentally differently than normal cameras. As a result, the amount of data required to replicate the scene is proportional to the amount of motion present, so the simulator must be capable of reproducing this variable data rate.



(a) Synchronous sampling event simulator (b) Adaptive sampling event simulator

Figure 4.9. Different sampling techniques for event simulators [22]

To emulate an event camera, one would need continuous representations of the visual signal at each pixel, which are not available in practice. To address this issue, previous work like [108] proposed sampling the visual signal synchronously, at a very high frame rate, and performing linear interpolation between the samples to reconstruct a piecewise linear approximation of the underlying continuous visual signal, which is used to emulate the operation of an event camera, Figure 4.9(a). ESIM takes the same general approach to simulating events by sampling the visual signal (by rendering images along the camera trajectory), but with one critical difference: rather than choosing an arbitrary rendering framerate and sampling frames uniformly across time at that framerate, the authors of [22] propose to sample frames adaptively, adapting the sampling rate based on the predicted visual dynamics, Figure 4.9(b). ESIM architecture is depicted in Figure 4.10. It is strongly coupled to the rendering engine (which outputs the motion field and the irrandiance of the scene) and the event simulator, allowing the event simulator to query visual samples (i.e. frames) adaptively based on the visual signal's dynamics. As illustrated in Figure 4.9, adaptive sampling more faithfully reproduces the ideal output of an event camera, but let us now examine how this sampling strategy works. Adaptive sampling can be achieved in two different ways:

- **Brightness change**: A first-order Taylor expansion of the brightness constancy assumption under the assumption of Lambertian surfaces [6]

[6]The apparent brightness of a Lambertian surface to an observer is the same regardless

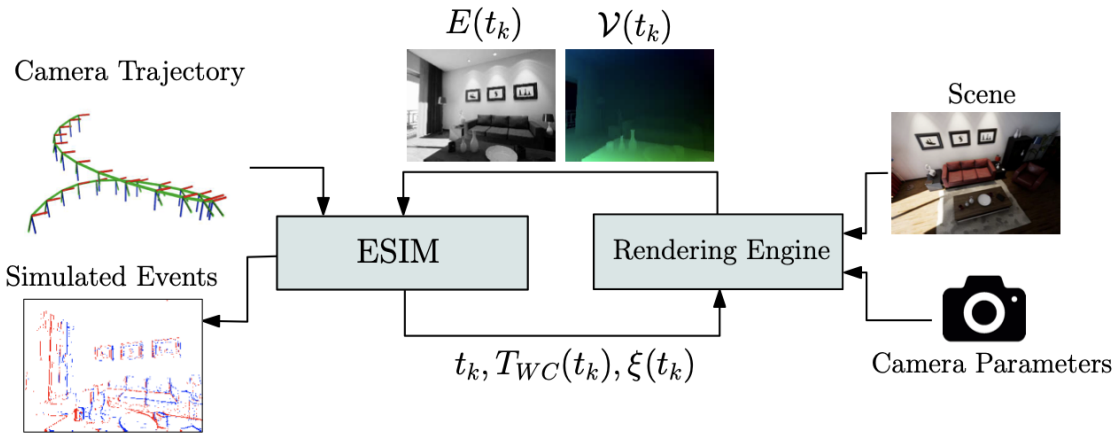


Figure 4.10. ESIM architecture [22]

yields:

$$\frac{\partial \mathcal{L}(x;t_k)}{\partial t} \simeq -\langle \nabla \mathcal{L}(x;t_k), \mathcal{V}(x;t_k) \rangle \tag{4.9}$$

where $\nabla \mathcal{L}$ is the gradient of the brightness picture and $\mathcal{V}$ is the motion field. In other words, the expected (signed) brightness change at pixel x and time t_k throughout the course of a particular time interval t is:

$$\Delta \mathcal{L} \simeq \frac{\partial \mathcal{L}(x;t_k)}{\partial t} \Delta t \tag{4.10}$$

We want to ensure that for each pixel x, $|\Delta \mathcal{L}| \leq C$ (i.e. the brightness change is constrained by the simulated event camera's intended contrast threshold C). Selecting the next rendering time t_{k+1} in the following manner enables this:

$$t_{k+1} = t_k + \lambda_b C \left| \frac{\partial \mathcal{L}}{\partial t} \right|_m^{-1} \tag{4.11}$$

where $\left| \frac{\partial \mathcal{L}}{\partial t} \right|_m$ is the maximum predicted rate of brightness change over the image plane, and λ_b is a parameter that determines the speed-accuracy trade-off of the rendering. Note that the validity of the formula depends

of the observer's angle of view

on the assumption of Lambertian surface, brightness constancy, and linearity of local image brightness changes therefore, in order to take into account all possible non-linearities a stricter criterion can be imposed by chosing $\lambda_b < 1$;

- **Pixel displacement**: A more straightforward technique for performing adaptive sampling is to ensure that the maximum displacement of a pixel between two successive samples (rendered frames) is constrained. This can be accomplished by selecting the following sampling time t_{k+1} in this way:

$$t_{k+1} = t_k + \lambda_v |\mathcal{V}(x; t_k)|_m^{-1} \tag{4.12}$$

where $|\mathcal{V}(x; t_k)|_m$ is the highest value for the motion field at time t_k and λ_v may be set < 0 in order to take into account the non-linearities;

4.6.2 Video interpolation

To simulate the high temporal resolution of event cameras from a dataset of videos shot at frame rates lower than the objective one, another step must be taken. In particular, video interpolation is the objective we want to achieve. Its objective is to generate intermediate frames that can be used to construct spatially and temporally coherent video sequences so to artificially increase the frame rate of our videos. Interpolation accuracy is frequently employed to evaluate optical flow algorithms in the classical approach to video interpolation like in [109, 110]. These techniques are capable of generating intermediate frames at any time interval between two input frames. However, motion borders and severe occlusions continue to pose difficulties for present flow algorithms, and so interpolated frames frequently contain artifacts around moving object boundaries. Additionally, the intermediate flow calculation (i.e., flow interpolation) and occlusion reasoning are heuristic-based and are not trainable end-to-end. Deep learning's success in high-level vision tasks has encouraged the development of various deep models for low-level vision tasks, including frame interpolation. The authors of [111] learn CNN models for optical flow using frame interpolation as a supervision signal. However, because their primary objective is optical flow, the interpolated frames are frequently hazy. The work in [112] treats frame interpolation as a local convolution between the two input frames and use a CNN to develop a spatially adaptive convolution kernel for each pixel. Their approach produces high-quality results. However, predicting a kernel for each pixel is computationally and memory heavy. Furthermore, these CNN-based approaches for

single-frame interpolation are unsuitable for multi-frame interpolation.

Super Slow Motion

The authors of Super Slow Motion (or Super SloMo), in [23], proposes a high-quality variable-length multi-frame interpolation algorithm that can interpolate a frame between two frames at any arbitrary time step. The paper's premise is that, to a first approximation, the difference between two frames of video can be conceived of as an optical flow. That is, the majority of pixels in one frame match to pixels in the adjacent frame, moved in a particular direction. Calculating optical flow between pictures is a well-known algorithm (including CNN-based approaches). Once an optical flow is established, intermediate frames can be interpolated by simply multiplying pixels by the interpolated quantity t. This can be further enhanced by the addition of time-reversed optical flow. Unfortunately, this solution does work perfectly on its own, particularly around the edges of fast-moving objects, because some pixels are occluded in one or more frames. (On rare occasions, intermediate pixels are obliterated in both the start and end frames). Additionally, there is the issue of opacity. As a result, Super SloMo computes visibility maps [7] (to track occlusions) while simultaneously fine-tuning the initial estimation of the bi-directional flow maps in a second CNN. By adding visibility maps to the warped pictures prior to fusion, occluded pixels may be excluded from the interpolated intermediate frame, hence decreasing artifacts. Combining flow map refinement and visibility training dramatically increased performance, which makes sense given that the same features are expected to predict flow and occlusion. To be more precise, the flow computation CNN is used to estimate the bidirectional optical flow between the two input images, which is then linearly fused to approximate the intermediate optical flow required to warp the input images. Both the flow computation and interpolation networks have characteristics that are independent of the time step to be interpolated, which is an input to the flow interpolation network. As a result, the technique is capable of simultaneously generating as many intermediate frames as required. In mathematical words, the objective is to predict the intermediate picture $\hat{I}_t$ at time $T = t \in (0,1)$ given two input images I_0 and I_1. To do so the two optical flows $F_{t\to 0}$ from I_t to I_0 and $F_{t\to 1}$ from I_t to I_1 are approximated (since the target I_t is not accessible) using

[7] $V_{t\leftarrow 0}(p) \in [0,1]$ denotes whether the pixel p remains visible from time 0 to time t (0 means fully occluded)

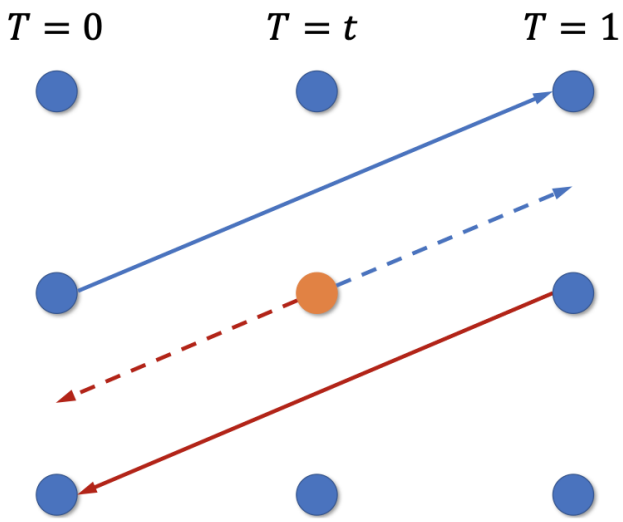


Figure 4.11. Intermediate flow estimation visually [23]

$F_{1\to0}$ and $F_{0\to1}$.

Consider the toy example depicted in Figure 4.11, where each column represents a certain time step and each dot represents a pixel. We are interested in simulating the optical flow of the orange dot at $T = t$ to the same pixel p at time $t = 1$ ($F_{t\to1}(p)$). One straightforward method is to borrow the optical flow from the same grid pixel at $T = 0$ and $T = 1$, provided that the optical flow field is smooth locally, and approximate the target as follows:

$$\hat{F}_{t\to1}(p) = (1-t)F_{0\to1}(p)$$
$$\text{or}$$
$$\hat{F}_{t\to1}(p) = -(1-t)F_{1\to0}(p)$$

From this toy example, passing in vector form and considering the bi-directional flow we can estimate intermediate optical flow as shown:

$$\hat{F}_{t\to0} = -(1-t)tF_{0\to1} + t^2F_{1\to0}$$
$$\hat{F}_{t\to1} = (1-t)^2F_{0\to1} - t(1-t)F_{1\to0}$$

Notice that these approximations, which as mentioned before works well in smooth regions, may have problems in motion boundaries. To reduce problems in those regions, and consequently artifacts, [23] proposes to refine the estimates through a flow interpolation network. Finally, given the values

of $\hat{F}_{t\to 0}$ and $\hat{F}_{t\to 1}$ we can interpolate $\hat{I}_t$ such that:

$$\hat{I}_t = \alpha_0 \odot g(I_0, \hat{F}_{t\to 0}) + (1-\alpha_0) \odot g(I_1, \hat{F}_{t\to 1}) \tag{4.13}$$

where $g(\cdot,\cdot)$ is a *backward warping* function to pass (in the case of the work the bilinear interpolation has been used) while α_0 is a weighting factor to give more relevance to the estimate obtained starting from I_0 or from I_1. The value of α_0 may be higher if t is closer to 0 and viceversa (temporal reasoning). At the same time the weight is also influenced by occlusion, that is why visibility maps come into play. Taking into consideration both aspects the formula becomes:

$$\hat{I}_t = \frac{1}{Z} \odot ((1-t)V_{t\leftarrow 0} \odot g(I_0, \hat{F}_{t\to 0}) + tV_{t\leftarrow 1} \odot g(I_1, \hat{F}_{t\to 1})) \tag{4.14}$$

where $Z = (1-t)V_{t\leftarrow 0} + tV_{t\leftarrow 1}$ is a normalization factor. Since the values of the visibility maps need to be estimated as well, they are actually predicted by the same flow interpolation network with the contrain that $V_{t\leftarrow 0} = 1 - V_{t\leftarrow 1}$ to avoid network divergence. Due to the usage of soft visibility maps, even if the pixel is visible both in I_0 and I_1 the networks learns by itself how to combine the information. Notice that:

- $F_{0\to 1}$ and $F_{1\to 0}$ are obtained via a flow computation CNN which uses two input images I_0 and I_1 to predict both the forward and the backward flow;
- Instead of directly predicting the intermediate flow estimates, the flow interpolation network performs a slightly better by computing the residual flow residuals:

$$\Delta F_{t\to 0} = F_{t\to 0} - \hat{F}_{t\to 0}$$
$$\Delta F_{t\to 1} = F_{t\to 1} - \hat{F}_{t\to 1}$$

The final architecture can be seen in Figure 4.12 where it is possible to observe what are the inputs and the outputs of both the CNNs adopted for flow computation and then interpolation. In Figure 4.13 it is shown in detail the architecture of the CNNs which are both U-Net models.

The last important aspect about this method is the loss function used to train end-to-end the whole model. It is a linear combination of four different losses:

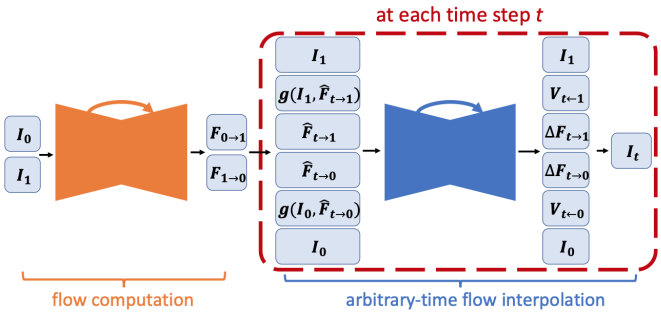


Figure 4.12. Super SloMo architecture schematics [23]

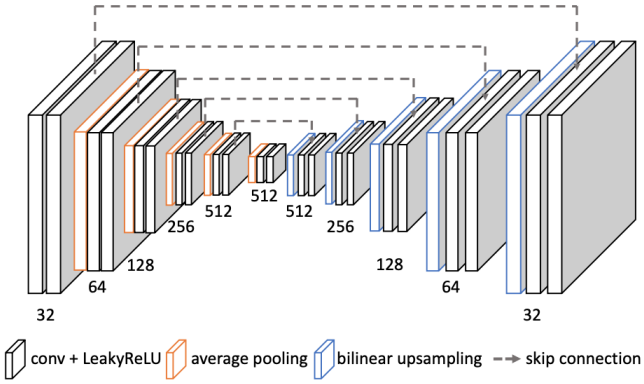


Figure 4.13. Super SloMo architecture details [23]

- **reconstruction loss**: which is a $L1$ loss in the RGB space which compares the predicted intermediate frame against the true one;
- **perceptual loss**: which compares the features, from an ImageNet pretrained VGG16, generated from the predicted frame against the ones generated from the true frame through an $L2$ loss to reduce image blur and make the frame sharper;
- **warping loss**: comparing frames to their flow-warped counterparts using the reverse warping function g;
- **smoothness loss**: encouraging neighboring pixels to have similar flow values;

Part II

Second part: EVEgo

Chapter 5

Dataset

There is a tremendous interest and potential in the events modality but event camears are quite rare nowadays and this slows down the development in this particular area of Computer Vision. However, in situations like this, researchers usually rely on synthetic dataset. This means that much attention must be paid in this process since the quality of the generated dataset is very crucial and constitutes the premise for being able to carry out truthful studies. As studies are at the beginning and event dataset are lacking, a fundamental contribution for the Computer Vision community is the extension of the EPIC-Kitchens dataset with its synthetic event-modality version.

5.1 EPIC-Kitchens event extension

Since the event cameras work at very high fps, traditional videos, usually recorded at 30-60 fps, are not enough to generate a good amount of events. This is the reason why in the pipeline we followed, the first step consisted in generating an upsampled version of the original RGB input video using Super SloMo [4.6.2]. Along with the upsampled frames [1], SloMo generates a file containing all the corresponding timestamps of the interpolated frames and this information will be useful in our work as it will be explained in this chapter. Going into more details, after an RGB video from EPIC-Kitchens was given as input to SloMo to obtain the upsampled frames, we could have followed three different pipelines to achieve our goal.

[1] upsampled frames are in gray-scale but this is not a problem since ESIM [4.6.1] would have converted RGB frames into gray-scale ones.

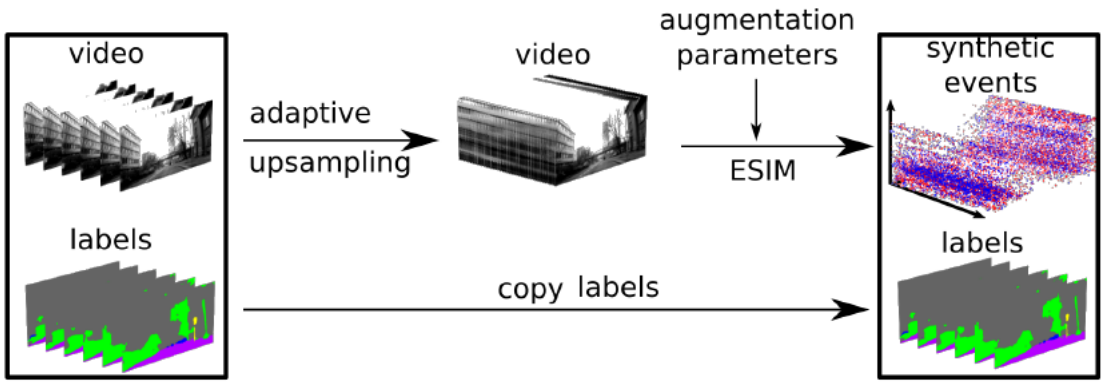


Figure 5.1. Method overview. The original video is first upsampled using SloMo and the generated video is fed to the ESIM simulator which produces the stream of asynchronous events [24]

- *Pipeline 1.* All the upsampled frames are fed into ESIM and the synthetic events are used to generate a single CxHxW Voxel representation [4.5.1].
- *Pipeline 2.* All the upsampled frames are fed into ESIM and the syntethic events in output are divided into groups and for each group I generate a CxHxW Voxel representation.
- *Pipeline 3.* All the upsampled frames are divided into groups. Then, from each group we obtain the corresponding syntethic events which are used to generate a CxHxW Voxel representation for each group.

5.2 Pipeline

In our work we chose *Pipeline 3.* Given two consecutive original RGB frames i and $i+1$ we call *slomo-frames*$_i$ the sequence of upsampled frames obtained by SloMo. We know the frame rate used to record each video of the EPIC-Kitchens dataset [9] and we know the timestamps of all the upsampled frames generated so this means that we are able to correctly identify the frames belonging to each set *slomo-frames*$_i$ with $i \in \{1, n-1\}$ where n is the number of original frames of the video in question. At this point of the pipeline a factor must be chosen and it will be like an hyperparameter of the conversion process. A factor equal to 3, for example, indicates that a single Voxel representation is created taking as input 3 consecutive sets of upsampled

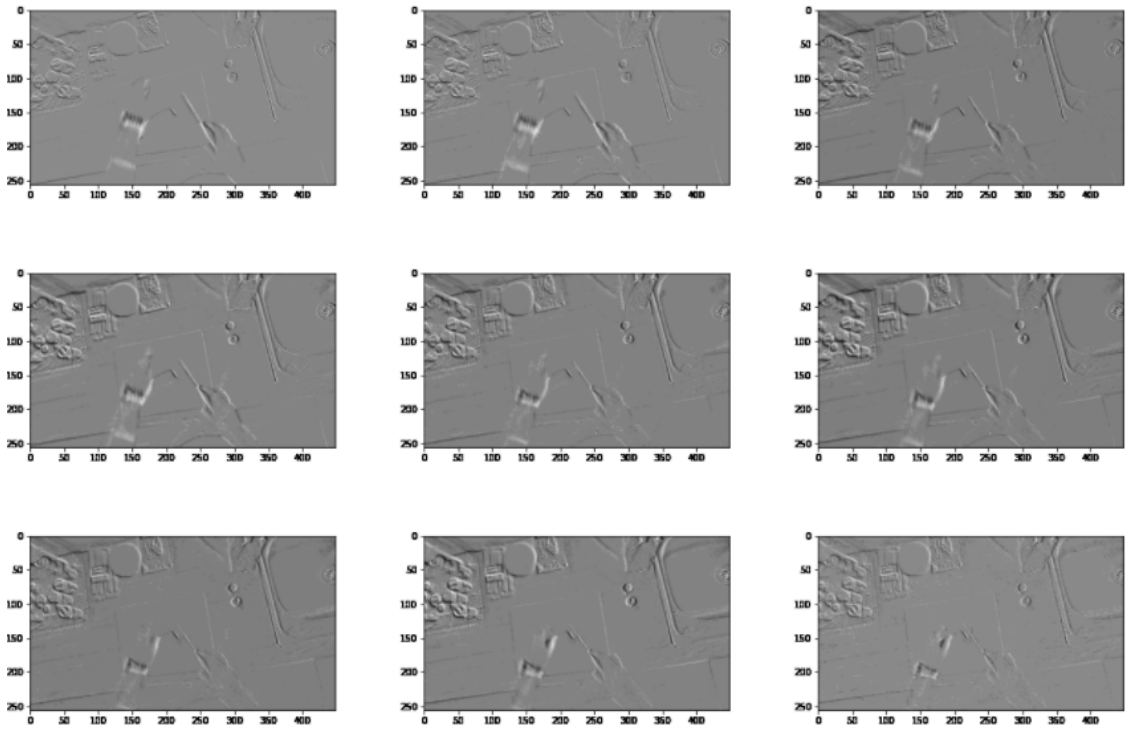


Figure 5.2. Example of a 9 channels Voxel Grid representation obtained at the end of the conversion process

frames or, equivalently, taking 3 original RGB frames as input. In our work we decided to use a factor equal to 6 because, since the original videos are recorded at 60 fps [9], the chosen factor means taking into account a temporal window of 100ms in the input RGB video and in literature this quantity is recommended when generating a single Voxel Grid [113]. We would like to clarify that before using ESIM, we manually added the last frame of the set *slomo-frames*$_i$ to the set *slomo-frames*$_{i+1}$ as if it was the first upsampled frame of the latter set. This is done because otherwise we would have lost the synthetic events generated comparing the last and the first frames of two consecutive sets. We would like to highlight that we have set ESIM to work in the log domain (i.e., the pixel brightness is measured in logarithmic scale) to have a more realistic setting: real event cameras operate in this domain measuring changes of the log-brightness to reach high dynamic ranges [22]. As for the threshold C beyond which events must be generated (look description in Section 4.3), we chose the value 0.05 which is the minimum

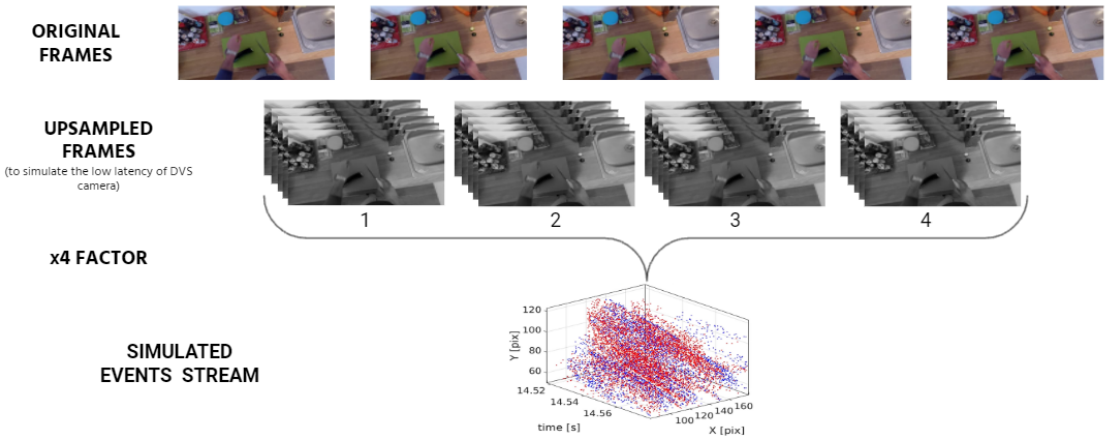


Figure 5.3. A representation of the pipeline employed in our work when using a factor equal to 4.

recommended value in [24]

During the conversion process we also noticed that, sometimes, in the original videos from EPIC-Kitchens there were two identical consecutive frames and this is due to the fact that the GoPRO cameras used to record this dataset had a low light mode active that replicated frames in case of not well lit scenes. However, in our setting, this phenomenon is not a problem because we use a factor greater than 1 in the Voxel generation so these replicated frames causes a negligible effect.

5.3 Solving the Blocking Artifacts

Another issue we faced in creating this synthetic dataset is the Blocking Artifacts encountered in the original RGB frames that were released in JPEG format. A compression artifact is a visible distortion of audio, images or video caused by the compression algorithm which has the task of discarding some of the medias' data in order to reduce its size on the disk [114]. Artifacts can occur because sometimes the used algorithms are unable of distinguishing between distortions of little importance and those which are, instead, unpleasant to the user [114]. In particular we had to face the Blocking Artifact which manifests itself with the appearance of abnormally large pixel blocks and it is a typical distortion of the JPEG format. During our work we noticed the presence of this artifacts in the original RGB frames and

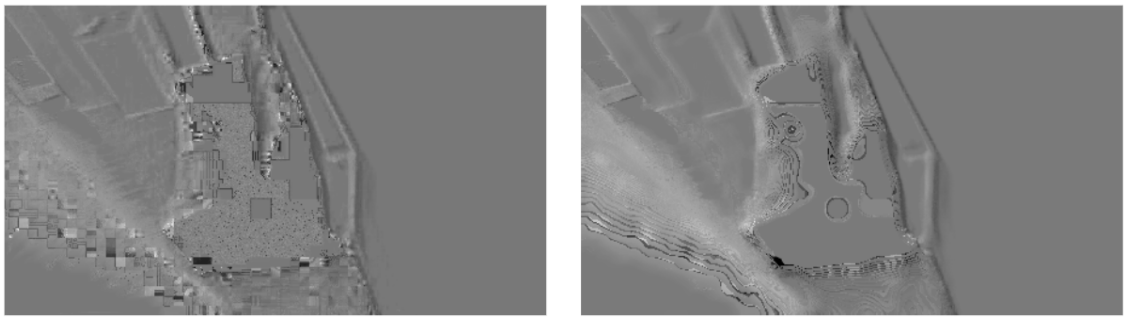


Figure 5.4. On the left, a channel of a Voxel representation when original RGB frames suffer from blocking artifact. On the right, the result after applying the Bilateral Filtering.

this had a bad impact on the final generated Voxel as shown in Figure 5.4. We made several attempts to counteract these distortions by creating some blur or adding filters on the original RGB frames. Finally, the technique which proved most effective and that we incorporated into the pipeline was adding a Bilateral Filter [115] on the upsampled frames extracted by SloMo.

In the widest definition of the word "filtering", the pixel at location (x_i, y_i) of a filtered image is a function of the input image's pixels in a neighborhood around it so that the farther a pixel is from (x_i, y_i), the less is its weight on the filtered pixel value at location (x_i, y_i). The intuition behind this approach is that images usually change slowly in space and so adjacent pixels are likely to have similar pixel values. Since the noise values which corrupt these neighbouring pixels are less correlated one another than the real signal values, a filter would average away the noise while retaining the signal [25]. However, this assumption is not true when it comes to edges.

The Bilateral Filtering has the objective to average the smooth regions of an image while preserving sharp edges. This is done combining a domain filtering (which is based on the distance between two pixels) with a range one (based on the difference in value of two pixels) enforcing both geometric and photometric locality [25]. So, the Bilateral Filtering operates by replacing a pixel with a weighted average of similar and near pixels. Going into formulas, this filter is so defined [115]:

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(||I(x_i) - I(x)||) g_s(||x_i - x||) \tag{5.1}$$

W_p is a normalization term

$$W_p = \sum_{x_i \in \Omega} f_r(||I(x_i) - I(x)||) g_s(||x_i - x||) \tag{5.2}$$

where:

- I and $I^{filtered}$ are respectively the original image and the resulting image after applying the Bilateral Filter and $I(x)$ indicates the intensity of the original pixel located at x;
- x indicates a location in coordinates of the pixel that is going to be filtered;
- Ω indicates the neighbourhood of pixels having the pixel located at x as its center;
- f_r is the range kernel which smooths value differences between two pixels;
- g_s is the domain or spatial kernel which smooths the differences in coordinates between two pixels.

In our work the kernels f_r and g_s were Gaussian functions. This means that if we want to filter a pixel with coordinates (i, j) and its neighbouring pixel is located at (k, l), the latter has a filtering weight on the former equal to:

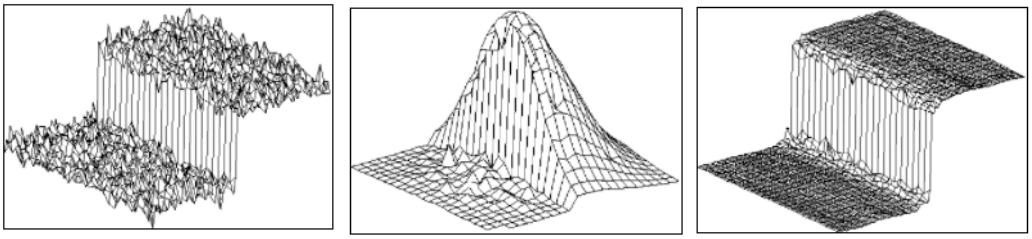
$$exp(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{||I(i,j) - I(k,l)||^2}{2\sigma_r^2}) \tag{5.3}$$

where:

- σ_r is the filtering range parameter and the higher it is, the more different colors of neighbouring pixels will be mixed together;
- σ_d is the filtering domain parameter and the higher it is, the more farther pixels will influence each other.

In our work we used a Bilateral Filter with a neighborhood diameter of 15 and the two σ parameters equals to 75. Indeed small σ values would have resulted in no filtering at all while high values would have generated "cartoonish" images [116].

We would like to remark the peculiarity of this Filter with an example: in smooth regions, pixels values within a small neighbourhood are comparable and the bilateral filter operates in a manner similar to a conventional domain



(a) Edge between dark and bright regions

(b) Values of the range kernel centered in a bright pixel

(c) Values of the resulting filtered pixels

Figure 5.5. How the Bilateral Filtering acts on sharp edges [25].

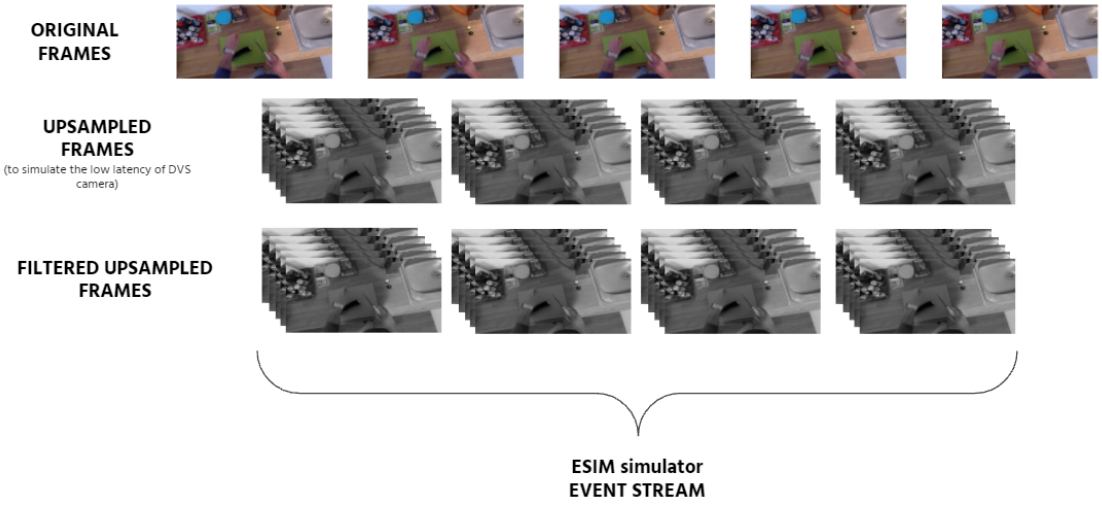
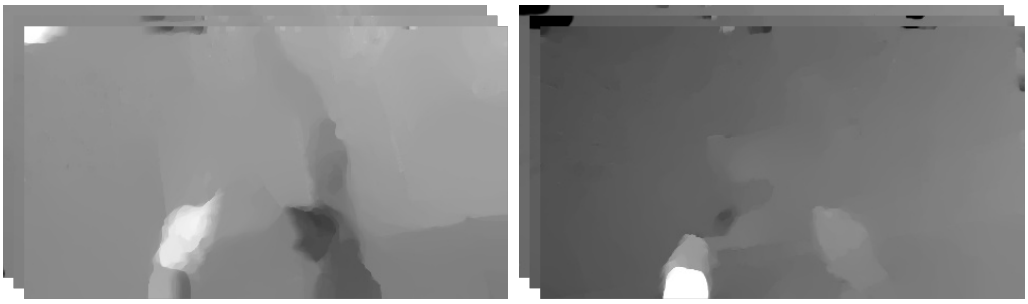


Figure 5.6. Representation of the final conversion pipeline obtained adding the filtering layer.

filter [25] but let us consider the case of a sharp edge between a dark region and a bright one like in Figure 5.5. The range kernel centered in a pixel on the bright side has values close to 1 in pixels on that same side while values close to 0 in pixels located on the dark side of the edge. This means that the pixel in question is replaced basically ignoring the dark pixels. The final result is shown in Figure 5.5 on the right. It is crystal clear that a good behavior is achieved in this difficult situation: sharp edges are preserved while smooth regions are averaged removing noise.



(a) Optical flow in the x direction (b) Optical flow in the y direction

Figure 5.7. Optical flow sample

5.4 Results

Finally, as a last part of this Chapter we report some qualitative results of the conversion of the dataset. In Figure 5.8 it is shown a sample converted first into events and then the event cloud brought in a Voxel representation. Finally, in Figure 5.7 the optical flow of the same sample is reported in order to show the differences and the similarities with respect to the Voxel and RGB data. It is possible to see that the optical flow data is the more abstract one, the one which gives more relevance to the pure movement, however, the Voxel representations are much more similar to it with respect to the original RGB frames. Event data, despite capturing, for example in the top-right corner of the frame, some affordances of the environment, contains less information non-related to the movement.

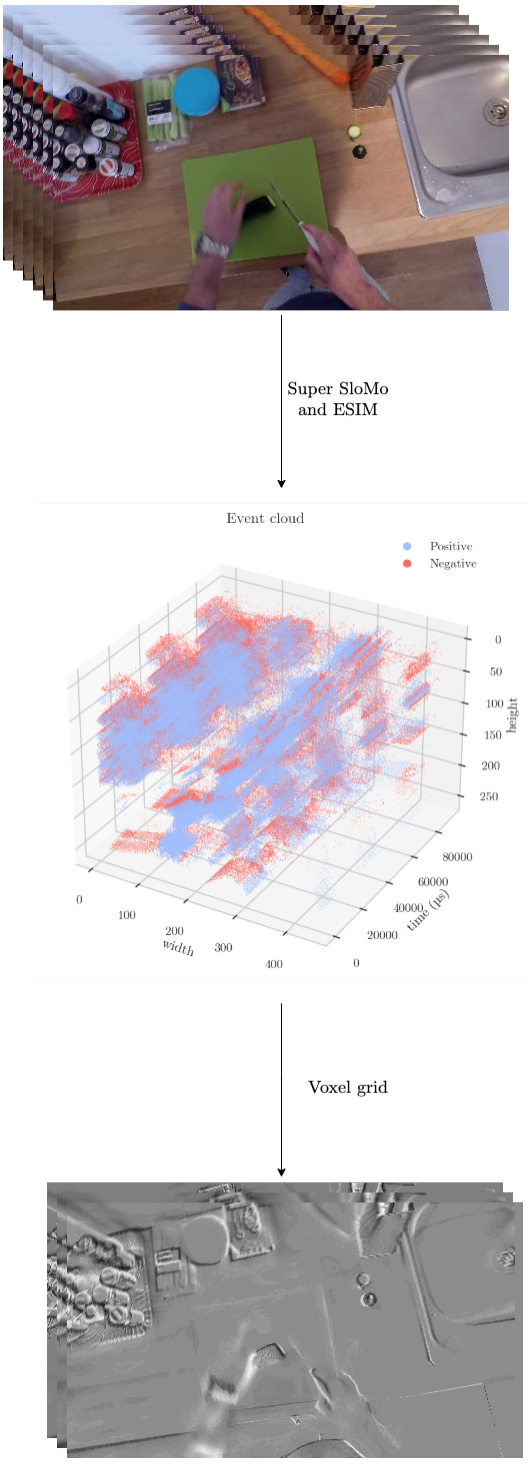


Figure 5.8. Example of conversion

Chapter 6

Implementation details

This chapter explains the implementation details necessary to attain the benchmark. To begin, Section 6.1 describes the various sampling procedures used, i.e., dense and uniform sampling. Second, Section 6.2 will outline the preparation pipeline, regarding all data preprocessing techniques used on the data before feeding them to the models. Finally, Sections 6.3 and 6.4 cover respectively the details about the backbone architecture and how the models have been changed in order to fit for the event data and the details of the training schedule with all the parameters adopted.

6.1 Sampling techniques

Two widely used sampling procedures are used in action recognition to generate model inputs. The first method, known as uniform sampling, is frequently used in 2D graphics. It divides a video into numerous equal-length segments and then randomly selects one frame from each segment, it is possible to observe how it works from Figure 6.1. Instead, the other method utilized by 3D models, dense sampling, uses an input set of continuous (or with a statically strided) frames directly starting from a randomly sampled frame inside the sample, as you can see from Figure 6.2. As a result, we used a dense sampling strategy for the I3D model and a uniform sampling method for the TSN and TSM models. Finally, notice that for video action recognition, there are two primary evaluation metrics: clip-level accuracy and video-level

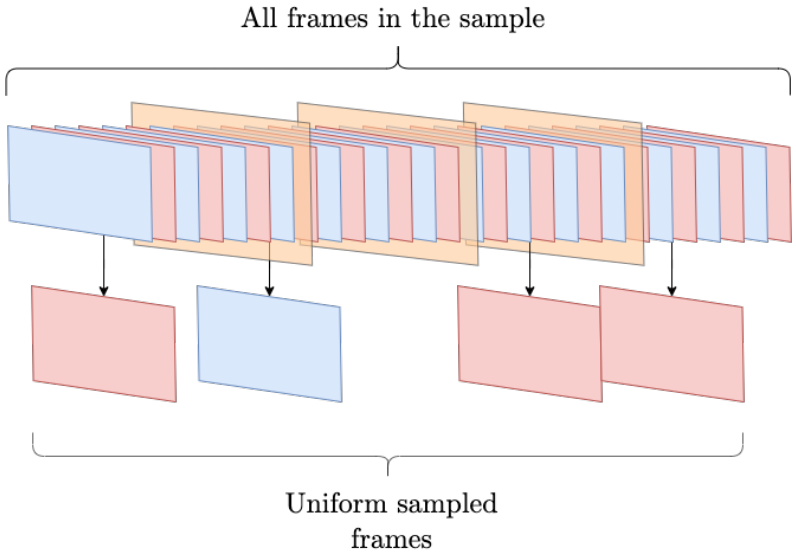


Figure 6.1. Example of uniform sampling

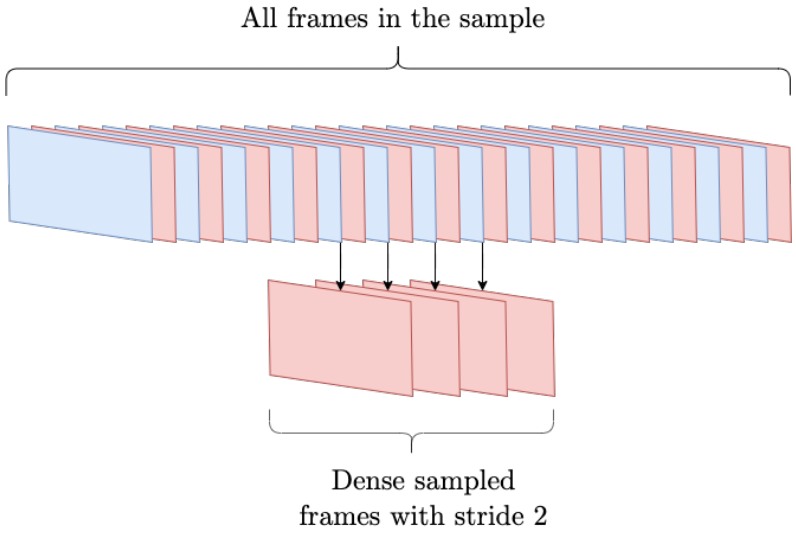


Figure 6.2. Example of dense sampling with stride 2

accuracy. Clip-level accuracy is calculated by putting a single clip[1] into the

[1]A clip is a collection of frames obtained by one sampling round, e.g., in Figure 6.1 and 6.2 the result is one clip

network, whereas video-level accuracy is calculated by combining the predictions of numerous clips; hence, video-level accuracy is typically greater than clip-level accuracy.

6.2 Preprocessing

To begin, it should be noted that data augmentation techniques are used to add training data in order to minimize overfitting and allow the model to learn from the greatest possible number of examples. As a result, the training and testing pipelines are slightly different. When discussing the training pipeline. In fact, while in training we want the network to learn how to generalize properly, in testing we want to see how it would perform on real data without any alteration. To be more precise, while the multiscale crop and the random horizontal flip (described respectively in Sections 6.2.1 and 6.2.3) are used in training, the central crop (Section 6.2.2) is applied during test; finally, normalization and clipping (Sections 6.2.4 and 6.2.5) are techniques adopted independently of the purposes. Furthermore, observe that all of these transformations are performed on a single record, which is essentially the composition of all the frames sampled from an action (basically, in our setting a sample corresponds to an action); in this sense, it may be viewed as if they were modifying a group of frames.

6.2.1 Multiscale Crop

The aim of this transformation is to crop images in different manner in order to add a layer of randomization which works as data augmentation. To do so, an initial crop on the original image is done chosing the offsets and the dimensions of the crop in a quite sophisticated way:

- At first the possible crop dimensions are determined. To do so the smaller size of the image is taken and then scaled by multiplying it for a list of certain values passed by the operator (which in our case are 1, 0.875, 0.75);

- All possible cropping dimensions (width and height) are made by coupling the possible values obtained in the previous step. A certain maximum value of distortion can be set in order to allow reduce the discrepancy among the dimensions and avoid changing the image too much;

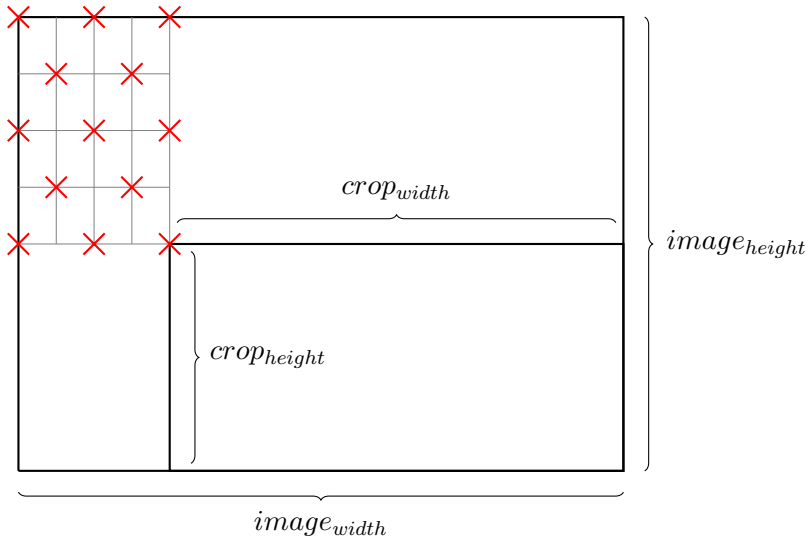


Figure 6.3. Example of possible offsets (red crosses) for multiscale crop

- Randomly among the possible pairs the dimensions of the crop are selected, i.e., $(crop_{width}, crop_{height})$;
- The cropping offset can be selected in two different ways: either randomly among the available pixels (e.g., $offset_{width} \leq image_{width} - crop_{width}$) or among some possible fixed positions. The fixed positions are obtained by structuring the available offset space (which depends on the crop dimensions and on the image dimensions) in a grid-like manner, the grid and the possible offsets can be seen, within an example, in Figure 6.3;

Notice that, after having cropped the image, the result may be also smaller than the desired size because of the effect of scaling. Despite this, the final step of the transformation is a re-scaling of the crop to the dimensions wanted. It may cause some distortion on the result as it can be observed in Figure 6.4. Since this is a transformation for all the group of frames, once sampled, crop dimensions and position are the same for all the frames.

6.2.2 Center Crop

As mentioned before, at test time we want to achieve the highest possible accuracy, to do so, since we want to be sure to keep the relevant information into the image, a central patch is taken from the data. In particular, since

(a) Original image (b) Cropped image

Figure 6.4. Example of multiscale crop application

the dimension of the crop is known, the offsets are computed as it follows:

$$\begin{aligned} offset_h &= (image_h - crop_h)/2 \\ offset_w &= (image_w - crop_w)/2 \end{aligned} \tag{6.1}$$

6.2.3 Random Horizontal Flip

The main aim of this transformation is to avoid network overfitting by randomly flipping horizontally the images with a certain probability p. Since the sample considered in our case is a collection of frames, the flip is done either for all the images or for none of them.

6.2.4 Normalization

Regarding the normalization technique adopted, since the models have always been used starting from a pretrained setting on famous datasets (Kinetics dataset proposed in [117] or ImageNet one which have been published in [50]). Due to this reason, to avoid a negative impact on the model while finetuning it, data have been fed to the network in the same form it received pretrained samples. So, new records have been rescaled in the expected range and then the mean and the standard deviation of the pretrained samples have been

(a) Original image (b) Horizontally flipped image

Figure 6.5. Example of multiscale crop application

used to normalize them. Mathematically, the formula used is the following one:

$$x_{rescaled} = \frac{x - min_x}{max_x - min_x}(max_{range} - min_{range}) + min_{range}$$
$$x_{normalized} = \frac{x_{rescaled} - \mu}{\sigma} \tag{6.2}$$

where μ and σ are respectively the mean and the standard deviation of the pretrained data and the normalization is done by channel. It means that also μ and σ have the same number of values as the number of channels of the input. Finally notice that x_{min} and x_{max} used to rescale the input, in our case, are computed over the whole group of frames related to one action.

6.2.5 Clipping

Clipping is a type of distortion in which a signal is limited when it surpasses a certain threshold. Due to the peaks detected in the Voxels generated from the dataset, one preprocessing technique tested has been clipping. The values of the Voxel Grids have been limited to a threshold or to a certain percentile and then all the interval has been rescaled into a certain range. Doing so

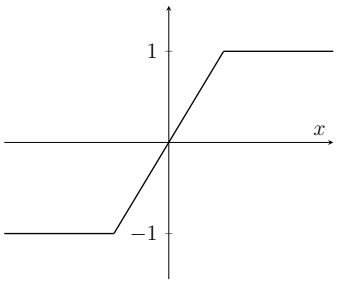


Figure 6.6. Clipping function with *threshold* equal to 1

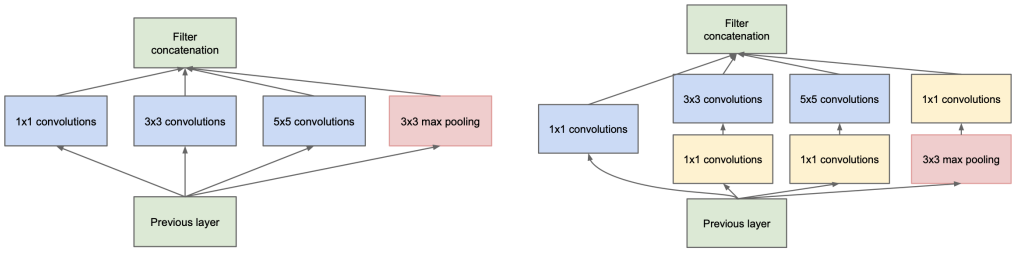
the values of the pixels are more spread and the network avoids to give more relevance and to overfit just based on the peaks. The mathematical formulation of this preprocessing technique is the following one:

$$value = \begin{cases} value, & \text{if } |value| \leq threshold \\ threshold, & \text{if } |value| > threshold \end{cases} \tag{6.3}$$

and a representation of the function can be seen in Figure 6.6.

6.3 Architectures

In terms of the models used in the benchmark, we chose those previously discussed in Section 3.6. Because they are higher-level structures that can be adapted to multiple low-level architectures, the models have distinct backbones. The I3D and TSN models have been implemented using a slight variation of the Batch Normalized Inception (or BN-Inception v1, proposed in [27]) for the first while the exact BN-Inception model for the second. It will be properly described in Section 6.3.1 and 6.3.2. While TSM is based on the ResNet-50 architecture, already described in Section 2.4 and proposed in [3]. The aim of this Section is to present the BN-Inception architecture which has been used in the benchmark and, at the same time, describe the implementation details which have been applied on the different models tackled in order to adapt them for the new data modality. At the same time we will present how the TRN module has been tried on top of the previous models and a new averaging temporal aggregation method tried, respectively in Section 6.3.6 and 6.3.7.



(a) Naïve inception module

(b) Inception module with dimension reduction

Figure 6.7. Inception modules [26]

6.3.1 GoogLeNet

GoogLeNet (or Inception v1) architecture arises in [26] as a result of the issue of prominent parts of an image that can vary significantly in size. Due of the large fluctuation in the location of the information, selecting the optimal kernel size for convolution becomes difficult. A larger kernel is preferable for globally dispersed information, whereas a smaller kernel is chosen for locally distributed information. Comparing this situation with biology: using just dense connections to gain knowledge about different sized objects is expensive while adopting sparsity connections, as in biology, and exploiting clustering correlated outputs seems a more reasonable choice (also suggested by the work of Arora et al. [118]). However, non-uniform sparse matrix calculations are expensive while the dense ones are much more efficient. So, in the way of finding out how an optimal local sparse structure in a convolutional neural network can be approximated and covered by dense components, the authors of [26] propose a module with filters of various sizes on the same level. With the network becoming slightly "wider" rather than "deeper." It is possible to witness what has been dubbed the "naïve" inception module in Figure 6.7(a). It conducts convolution on an input signal using three different filter sizes (1×1, 3×3, 5×5). Additionally, maximum pooling is carried out. Concatenated outputs are forwarded to the following conception module. One significant disadvantage of the preceding module is that even a small number of 5×5 convolutions can be prohibitively expensive when applied on top of a convolutional layer with a large number of filters. This issue gets much more acute when pooling units are included: their total number of output filters equals the total number of filters in the preceding stage. Merging the pooling layer's output with the output of convolutional layers

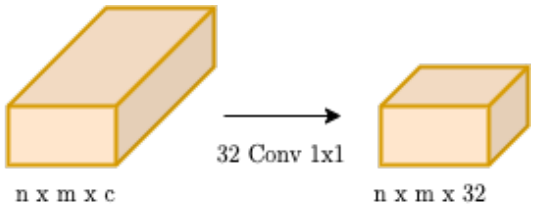


Figure 6.8. Effect of 1×1 convolutions

would inevitably result in an increase in the number of outputs from stage to stage. While this architecture would cover the best sparse structure, it would do so in a very wasteful manner, resulting in a computational explosion after a few steps. To keep the cost down, the authors of [26] include an additional 1×1 convolution before the 3×3 and 5×5 convolutions, as you can see in Figure 6.7(b). While it may seem illogical to add another operation, 1×1 convolutions are far less expensive than 5×5 convolutions, and the lower number of input channels also helps. In fact, it is possible to see those kind of convolutions as a sort of pooling in depth as you can see from Figure 6.8. Take notice, however, that the 1×1 convolution occurs after the max pooling layer, not before. In general, an Inception network is a collection of modules of the aforementioned sort built on top of one another, with the addition of periodic max-pooling layers with stride 2 to reduce the grid's resolution. In fact, Inception v1 is composed of 9 stacked inception modules for a total of 22 layers.

6.3.2 Batch Normalized Inception

Batch Normalized Inception (or BN-Inception) is proposed by the authors of [27] in order to improve the performance of their previous model applying some arrangements. In particular, the main issue which is addressed by this version of the Inception architecutre is the internal covariate shift already mentioned in detail in Section 2.2.10. To do so the previously mentioned mechanism of batch normalization has been used after each convolution. Another relevant change done in this architecture with respect to the previous one is the fact that, to decrease computational cost at no representational loss, the 5×5 convolution in the inception module has been substituted by two consecutive 3×3 convolutions. The final inception module adopted by this architecture is depicted in Figure 6.9.

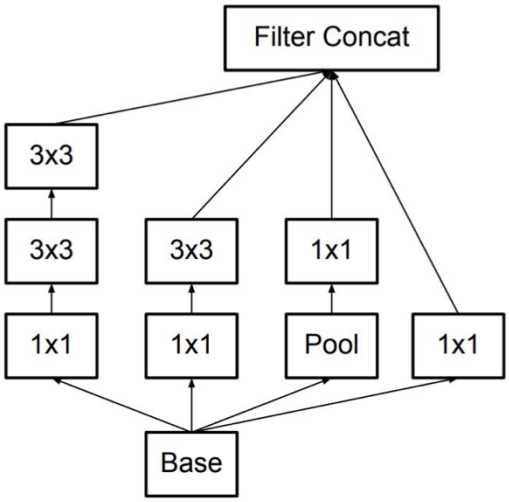


Figure 6.9. Inception module for BN-Inception network [27]

6.3.3 Temporal Segment Network Adaptation

Due to the fact that the TSN model is based on the concept of sparsely sampling throughout the video and combining the properties of all the snippets, it is easily adaptable to the Voxel Grid input. Indeed, because what was referred to in Section 3.6.1 as $\mathcal{F}$ (the CNN that generated the snippet output) in our case is the BN-Inception architecture, the only thing that needs to be changed to adapt it to the input is the first convolutional layer, which can be extended or shrunk in size depending on the number of input channels. Because the weights of the BN-Inception backbone are initialized using models pre-trained on ImageNet [50], as specified in the original GitHub repository (`https://github.com/yjxiong/temporal-segment-networks`), and we do not wish to lose this advantage, we changed the pre-trained weights of the first convolutional layer appropriately. To do this, the function *resize* from the *numpy* library (proposed in [119]) was used; by just feeding the model the number of input channels, it becomes capable of treating any type of Voxel Grid; in fact, as illustrated in Figure 6.10, it circularly replicates the weights of the first convolutional layer. Furthermore, when the input has a different channel count than RGB pictures, the mean and standard deviation vectors already provided by [6] and computed on the ImageNet dataset are appropriately expanded with the *resize* size function. In terms of the function $\mathcal{G}$ used to calculate the consensus, the average function has been used. Finally, the frames employed in this design are extracted one by one from

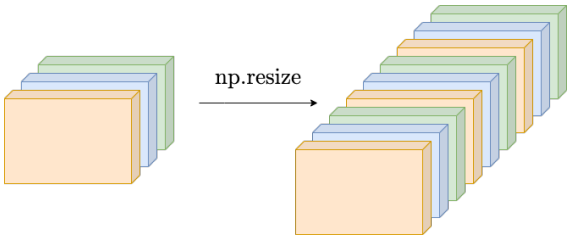


Figure 6.10. How resize function works along channel dimension

the sparsely sampled snippets.

6.3.4 Inflated 3D ConvNets Adaptation

As mentioned in the beginning of the Chapter, the backbone inflated for the I3D model is slightly different from BN-Inception. In fact, it uses batch normalization but it does not have a branch with a double 3×3 convolution. Basically the scheme for the Inflated Inception module used is presented in Figure 6.11. In addition to that, weights of the backbone architecture are initialized using models pre-trained on Kinetics [117], because of that the same procedure used for TSN is followed both for the first convolutional layer and for the mean and the standard deviation given for the normalization (in this case the ones computed on Kinetics dataset).

6.3.5 Temporal Shift Module Adaptation

For what concerns the TSM model, another backbone architecture has been adopted since BN-Inception was not compatible with the residual shift presented in Chapter 3.6.4. Because of this, the backbone adopted in this case has been ResNet-50 model. Also in this case it has been used with the weights initialized using models pre-trained on ImageNet and even in this case the adaptation of the first convolution and of the mean and standard deviation has been performed through the *resize* function. Finally all the parameters of the network has been left as default from the original repository (`https://github.com/mit-han-lab/temporal-shift-module`) and the bi-directional TSM has been adopted due to our offline setting.

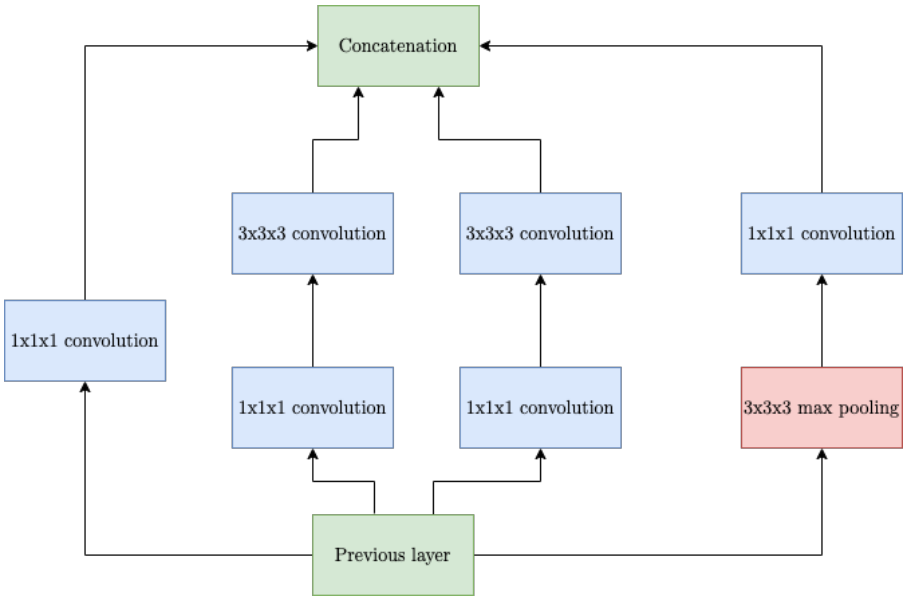


Figure 6.11. Inception module for I3D

6.3.6 Temporal Relation Network Adaptation

Since this module is used on top of existing CNNs we exploited the models obtained through previous training as CNN feature extractor for the TRN module. In particular 5 clips are extracted from the video and passed through the CNN feature extractor. The dimensionality of the features changes accordingly to the model adopted: I3D and TSN return features with dimension 1024 while TSM returns features of dimension 2048. Before being passed to the TRN module those features pass through a fully connected layer which reduces their dimension to 512 in every case. Then all features enter into the TRN module. In the module there are different phases:

- At first the features are concatenated in groups of 2, 3, 4 or 5. Up to 3 groups of a certain length are allowed and the temporal order of the frames is respected while generating them;
- The groups of concatenated features pass through a fully connected layer (there are is one fully connected layer per each group length). After this phase the dimensionality is already reduced to the number of classes, so, basically the fully connected layer works as a classifier for each group of concatenated frames;
- All fully connected layers predictions are summed and the final output

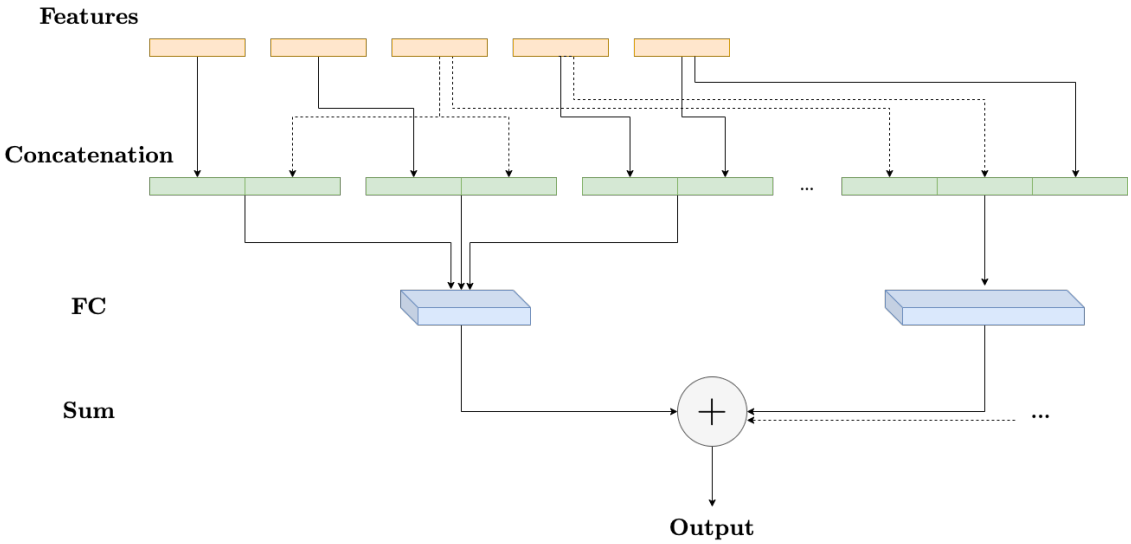


Figure 6.12. Details on the TRN module

predition is obtained;

6.3.7 Averaging Temporal Aggregation

The idea behind this module is simply to aggregate features of different clips instead of the predictions themselves. As in the previous case, this module is used on top of existing CNNs so we used it with the same number of frames per clip and with 5 clips. Once features from all clips are obtained they pass through a fully connected layer which cuts their dimension to 512 and then all the clips feature are averaged. The result is a single feature vector of dimension 512 which passes through a final fully connected that basically works as a classifier. The schema of the module can be seen in Figure 6.13.

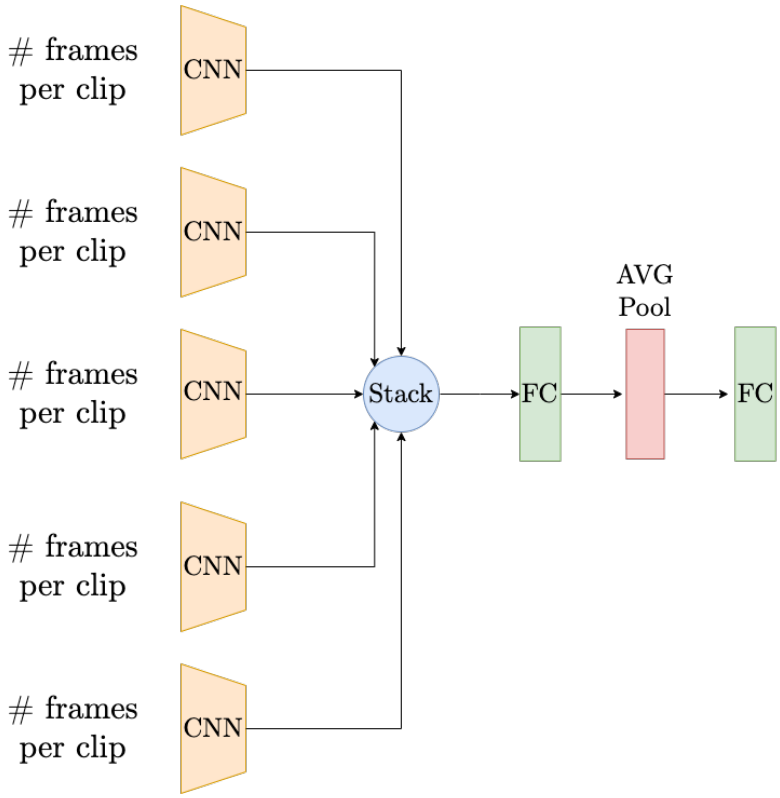


Figure 6.13. Details on the Averaging module

6.4 Training details

All models have been implemented in PyTorch [120]. In addition to that, the models have been trained with the Stochastic Gradient Descent optimizer (already mentioned in Section 2.2.3) with momentum [121] with a starting learning rate η of 0.01, a weight decay of 10^{-7} and a momentum μ of 0.9. We trained for a total of 5000 iteration since we observed it to be a good trade-off in terms of performance and speed of the training. Notice that, in our setting, due to the size of the dataset, an iteration is not as an epoch, so not all the dataset is passed through the model, instead just the batch size is considered. Due to the large size of the samples, despite training on 4 NVIDIA Tesla V100 16Gb GPUs and exploiting parallel computation

through the usage of *DataParallel*[2], we adopted a gradient accumulation[3] technique. With a mini-batch size adopted of 64 where possible and of 32 in some multi-modal cases and a batch size of 128 simulated through gradient accumulation. We trained for a total of 5000 iteration since we observed it to be a good trade-off in terms of performance and speed of the training. Notice that, in our setting, due to the size of the dataset, an iteration is not as an epoch, so not all the dataset is passed through the model, instead just the batch size of 128 is considered in an iteration. The learning rate is decayed by a factor of 10 at iteration 3000. I3D, TSN and TSM are trained end-to-end and the number of frames used changes accordingly to the modality and the model as follows:

- I3D model: 10 frames per clip for the Voxel, 16 for RGB and 16 for optical flow;
- TSN and TSM models: 5 frames per clip for all the modalities

For those models, while in clip-level testing just 1 clip has been considered, at video level always 5 clips have been adopted. Finally, TRN and Averaging methods are built on top of the previous models so the same number of frames per clip are used while the features are extracted from 5 different clips. Apart from the ablation study in Section 7.3.1 all the experiments in the next Section involving event data are done with the usage of 9 channel Voxel Grids. Finally, in all multi-modal cases the different outputs from all the networks from the different modalities are summed before the computing the final prediction.

[2]DataParallel parallelizes the application of the specified module by chunking the input across the defined devices (other objects will be copied once per device). The module is reproduced on each device in the forward pass, with each replica handling a piece of the input. Gradients from each copy are added to the original module during the backward pass

[3]Gradient accumulation refers to the process of doing a certain number of steps without updating the model variables while accumulating the gradients of those steps and then computing the variable updates using the accumulated gradients

Chapter 7

Experiments

After having explained both the conversion pipeline and the implementation details we will now show all the experiments done on the various modalities provided by the Epic Kitchens dataset. This Chapter will begin with an analysis on the most important conversion and preprocessing hyperparameters in order to feed to models with the most performing Voxel data (Section 7.1). Then, in Section 7.2 all the experiments performed will be shown with the respective performance properly shown and commented. Finally in Section 7.3, a couple of interesting ablation studies: altering the number of Voxel channels and treating those channels differently (as temporal information) are presented.

7.1 Event Anaylsis

In this chapter we show an analysis in order to identify an optimal preprocessing approach for a completely different input modality, such as the Voxel Grid, several clipping policies were tested. This has helped to spread the pixels values, which can be seen to have a high number of cells with low value and a few outliers in the extremes in Figure 7.1. However, since the clipping function has a threshold it must be determined in order to optimize the results of model. In particular, several different experiments have been carried on a single kitchen of the dataset in order to see the best threshold making the input best suited for the network. In Table 7.1 it is possible to observe the effects of clipping with different static thresholds. However, we also considered dynamic thresholds computing the 90^{th}, 95^{th} and 99^{th} percentiles directly on the sample (collection of sampled frames), but this method resulted in the divergence of the training schedule so we opted for

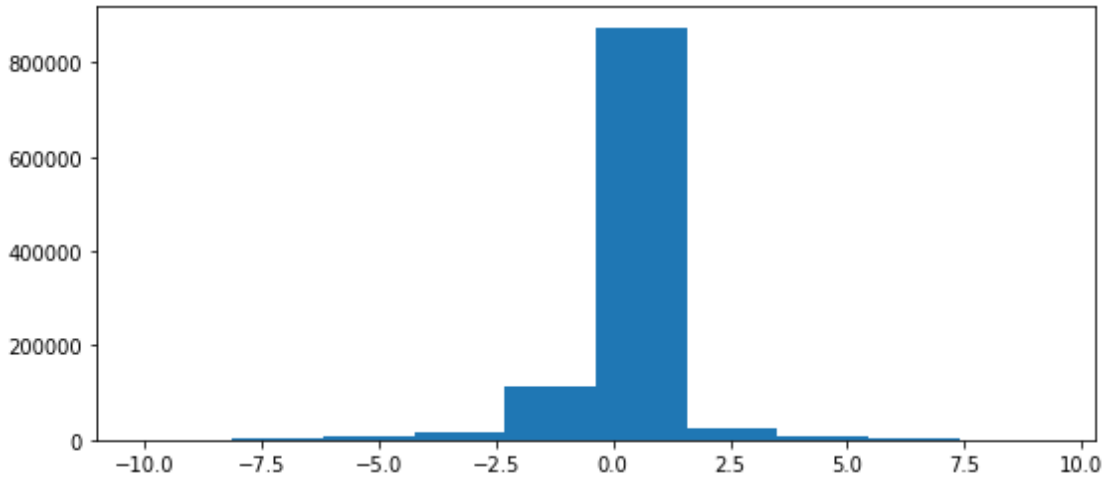
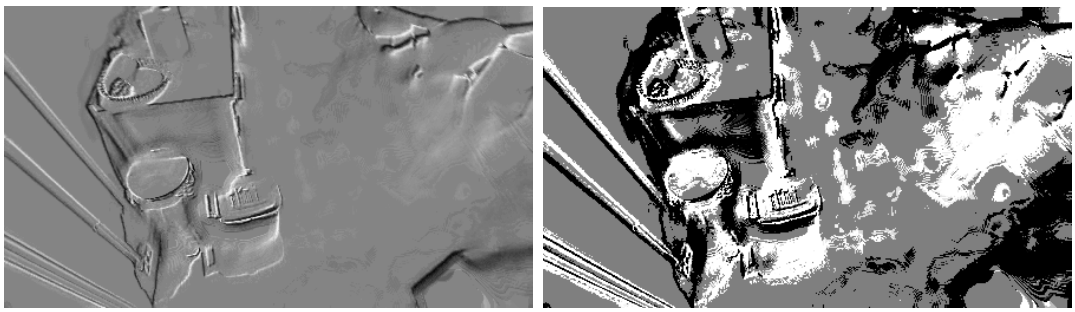


Figure 7.1. Number pixels ($y-axis$) with a certain value ($x-axis$) in a sample Voxel



(a) Original Voxel Grid (b) Voxel Grid after clipping at 0.5

Figure 7.2. Effect of clipping on Voxel Grid representation

a simple static one. In the light of the results obtained, which shows higher accuracy both at iteration 5000 and 9000 we finally decided to clip the values with $\theta = 0.5$. The qualitative effect of this transformation can be seen in Figure 7.2.

Finally a crucial parameter in the conversion pipeline has been the conversion factor indicating the number of consecutive upsampled RGB frames to be used to create that a single Voxel representation. The first aspect we considered under this point of view is the number of events extracted depending

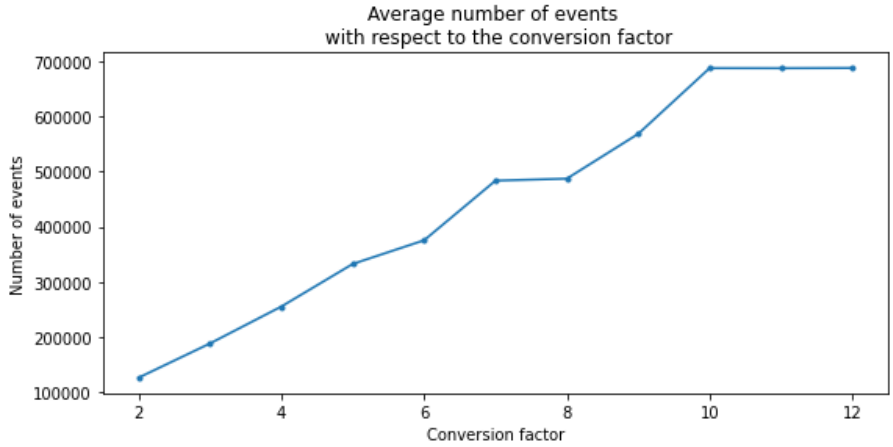


Figure 7.3. Conversion factor vs average events generated on a sample

Table 7.1. Results of different clipping thresholds θ

	No preprocessing		Clipping with $\theta = 5$		Clipping with $\theta = 1$		Clipping with $\theta = 0.5$	
Iteration	D3→D3	D3→D1	D3→D3	D3→D1	D3→D3	D3→D1	D3→D3	D3→D1
3000	50,92	32,38	48,53	32,64	54,03	36,60	54,20	36,34
4000	50,34	31,06	47,32	30,93	52,58	33,28	54,13	34,43
4500	49,87	31,62	46,81	31,01	52,67	33,51	52,91	33,84
5000	50,08	31,21	46,76	30,91	53,41	34,79	54,82	34,97
9000	51,72	33,15	48,36	33,82	54,65	35,35	56,08	36,58

on the number of frames used. To investigate it we took a sample and observed the average number of events generated against the conversion factor. The results are shown in Figure 7.3, they demonstrate that if the number of frames used is too low there are not enough events to generate a proper Voxel representation. However, increasing the conversion factor would also decrease substantially the dimensionality of the dataset. At the same time, in an hypothetical online setting, an high conversion factor is equivalent to increase the time before the generation of a Voxel. In order to take into account all the issues properly we decided to adopt a trade-off conversion factor of **6 frames per Voxel**. In fact, considering that the videos of the Epic-Kitchen dataset that we used have all been sampled at 60 frames per second, using 6 as a conversion factor does also mean that we are considering a temporal interval of $100ms$ for the creation of the Voxel which is consistent with N-Cars dataset samples, proposed in [122], that have the same temporal length.

7.2 Event EPIC-Kitchens Benchmark

In this section, we provide our novel benchmark, which includes the results obtained using RGB, event, and optical flow modality in single- and multi-modal fashion.

- Table 7.2 and 7.3 shows the performances reached using the *I3D* architecture respectively in a *intra-* and *cross-domain* scenario.
- Table 7.4 and 7.5 shows the performances reached using the *TSM* architecture respectively in a *intra-* and *cross-domain* scenario.
- Table 7.6 and 7.7 shows the performances reached using the *TSN* architecture respectively in a *intra-* and *cross-domain* scenario.

Notice that the Aggregation technique at feature level may be performed or not (late fusion). In case it is not it is possible to test both at clip or video level (with 5 clips). We reported all the possible combinations for completeness.

Inflated 3D ConvNets

Starting from the architecture composed by 3D convolutions we observe a surprising general trend, event data work up to 1% better than RGB in the cross-domain setting, validating the potentialities of event data (see Table 7.3). As it can be seen from Table 7.2, in a supervised setting RGB still works better. This is expected, since RGB data present a lot of details and environmental bias they tend to *"overfit"* on the source domain and be less capable of adapting to unseen ones. Another important aspect to be observed is that, in this case, video level testing performs usually better than the clip one. Indeed, because the I3D design is built on dense sampling, it can observe more video segments despite having a substantially greater processing cost.

Table 7.2. I3D results in supervised scenario.

I3D					
		Supervised			
Modality	Aggregation	D1	D2	D3	Mean
RGB	Late fusion (clip)	53.66	61.86	59.49	58.34
	Late fusion (video)	53.67	61.12	60.7	58.49
	Average	54.33	60.62	62.66	59.20
	TRN	54.35	61.46	63.34	**59.72**
Event	Late fusion (clip)	44.16	54.28	51.07	49.84
	Late fusion (video)	47.66	56.32	53.52	53.50
	Average	51.03	57.36	59.08	**55.82**
	TRN	49.60	56.84	57.79	54.74
Flow	Late fusion (clip)	52.34	56.03	50.18	52.85
	Late fusion (video)	56.44	61.67	57.29	58.47
	Average	59.31	67.06	64.57	63.65
	TRN	59.39	70.29	69.48	**66.39**
RGB+Flow	Late fusion (clip)	54.89	59.64	57.19	57.24
	Late fusion (video)	59.23	65.10	61.87	62.07
	Average	57.93	64.47	66.07	62.82
	TRN	57.24	67.98	67.62	**64.28**
RGB+Event	Late fusion (clip)	50.91	60.86	58.28	56.68
	Late fusion (video)	53.43	63.82	60.36	59.20
	Average	56.55	62.8	64.71	61.35
	TRN	55.66	63.61	65.28	**61.52**

Table 7.3. I3D results in cross-domain scenario.

I3D								
		Cross Domain						
Modality	Aggregation	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
RGB	Late fusion (clip)	32.36	34.58	31.52	33.94	33.54	34.64	33.43
	Late fusion (video)	34.50	35.70	34.94	36.46	33.93	38.37	**35.65**
	Average	32.46	35.79	34.94	35.21	36.12	38.26	35.46
	TRN	33.20	35.83	34.02	36.22	35.76	38.35	35.56
Event	Late fusion (clip)	31.96	31.81	33.33	36.95	33.10	40.00	34.52
	Late fusion (video)	35.26	34.98	33.41	35.55	35.33	42.90	36.24
	Average	35.08	31.72	34.48	36.32	35.53	42.44	35.93
	TRN	34.65	32.37	35.14	38.68	36.17	42.43	**36.57**
Flow	Late fusion (clip)	34.50	35.70	34.94	36.46	33.93	38.37	35.65
	Late fusion (video)	42.80	36.46	39.97	45.56	43.60	52.03	43.40
	Average	45.06	37.20	41.04	44.76	48.27	54.13	45.08
	TRN	48.86	37.56	44.72	45.34	53.43	58.03	**47.99**
RGB+Flow	Late fusion (clip)	41.62	38.19	39.59	41.27	41.51	45.53	41.28
	Late fusion (video)	44.44	40.94	42.86	43.60	43.96	51.54	**44.56**
	Average	34.53	38.08	38.75	39.77	38.85	43.55	38.92
	TRN	36.26	37.96	40.92	42.42	42.01	45.63	40.87
RGB+Event	Late fusion (clip)	33.22	36.19	32.93	39.57	35.35	37.41	35.78
	Late fusion (video)	35.72	37.78	35.50	40.48	36.65	42.30	**38.07**
	Average	30.75	36.17	36.14	36.66	33.84	41.95	35.92
	TRN	33.6	35.84	36.78	37.1	34.79	40.81	36.49

Temporal Shift Module

In this model (whose results are reported in Table 7.4 and 7.5) the difference among RGB and event modality is even more remarkable in the cross-domain scenario. In fact, the event modality reaches up to 2% higher performance than RGB data. Again, the fact that the event modality does not provide an improvement w.r.t. RGB in the supervised scenario is comprehensible as mentioned above. It is also worth noting that in this scenario, clip and video level test performance are comparable. This is probably due to the fact that the uniform sampling technique adopted already enhance the ability of the network to see a great portion of the video. Finally, adding a further layer of temporal aggregation, i.e., TRN, helps also in the cross-domain scenario while with I3D it could also damage the performance of the network. This may be due to the different type of features extracted by a 3D CNN model which implicitly extract temporal features against the one of a 2D CNN model adapted for temporal modelling.

Table 7.4. TSM results in supervised scenario.

TSM					
		Supervised			
Modality	Aggregation	D1	D2	D3	Mean
RGB	Late fusion (clip)	61.15	75.29	74.47	70.30
	Late fusion (video)	61.62	75.07	74.94	**70.54**
	Average	61.60	74.4	74.53	70.18
	TRN	60.46	73.86	74.23	69.52
Event	Late fusion (clip)	57.47	68.30	66.45	**64.07**
	Late fusion (video)	56.22	68.74	65.32	63.43
	Average	54.20	68.8	65.42	62.81
	TRN	55.63	69.33	66.11	63.69
Flow	Late fusion (clip)	55.20	64.73	58.94	59.62
	Late fusion (video)	55.68	64.19	57.71	59.19
	Average	57.62	70.13	66.77	**64.84**
	TRN	56.73	69.24	65.27	63.75
RGB+Flow	Late fusion (clip)	59.67	74.96	74.49	69.71
	Late fusion (video)	60.66	75.02	73.84	69.84
	Average	64.13	76.8	76.89	**72.61**
	TRN	57.93	69.97	66.65	64.85
RGB+Event	Late fusion (clip)	59.67	76.90	73.18	69.92
	Late fusion (video)	59.69	75.99	72.80	69.49
	Average	61.60	76.37	77.58	**71.85**
	TRN	61.65	76.3	77.34	71.76

Table 7.5. TSM results in cross-domain scenario.

TSM								
		Cross Domain						
Modality	Aggregation	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
RGB	Late fusion (clip)	37.02	32.03	35.02	38.28	33.92	36.47	35.46
	Late fusion (video)	36.52	32.26	35.61	37.82	33.41	37.91	35.59
	Average	36.8	31.93	36.09	40.35	36.09	37.6	**36.48**
	TRN	37.86	30.90	34.48	40.03	34.48	38.53	36.05
Event	Late fusion (clip)	29.42	33.85	36.76	43.82	34.84	44.33	37.17
	Late fusion (video)	29.53	33.95	36.73	43.82	31.80	43.05	36.48
	Average	31.02	33.06	37.44	45.32	36.09	43.63	37.76
	TRN	30.22	34.14	35.63	46.6	35.73	44.32	**37.77**
Flow	Late fusion (clip)	39.38	38.10	41.53	46.49	37.65	53.14	42.72
	Late fusion (video)	42.00	38.47	41.87	45.27	39.87	52.40	43.31
	Average	40.92	39.31	40.28	44.05	41.96	54.96	43.58
	TRN	41.6	40.11	42.75	44.21	44.05	55.09	**44.64**
RGB+Flow	Late fusion (clip)	39.45	34.52	37.32	45.13	37.29	43.87	39.60
	Late fusion (video)	38.90	38.90	36.50	45.19	37.27	43.93	40.11
	Average	39.2	35.11	39.77	43.02	39.59	47.06	40.63
	TRN	40.32	40.53	42.52	43.66	43.08	54.47	**44.10**
RGB+Event	Late fusion (clip)	33.79	37.00	36.63	45.19	37.73	43.93	39.04
	Late fusion (video)	33.54	36.59	35.30	45.39	38.21	43.91	38.82
	Average	34.41	34.88	37.55	47.74	39.08	44.93	**39.77**
	TRN	34.04	34.22	38.57	46.65	38.36	46.56	39.73

Temporal Segment Network

Finally, passing to the last architecture we can see, through Table 7.6 and 7.7, that in this case the TSN model is the one achieving a lower improvement in the cross-domain setting with respect to the RGB setting but still achieving on-par accuracy results. Again aggregation models do not affect cross-domain accuracies supporting the hypothesis done in the TSM chapter. The same can be said regarding video against clip level performance which are not really affected by the usage of multiple clips probably because of the uniform sampling strategy adopted.

General observations

Comparison between architectures. Among the three employed architectures, TSN is the worst in the cross-domain setting reaching 32.42%, 33.07% and 39.77% when fed respectively with RGB, event and flow data. I3D reaches slightly better accuracies while TSM is the most capable architecture to generalize. It shows average improvements up to 4%, 5% and 5%

Table 7.6. TSN results in supervised scenario.

TSN					
		Supervised			
Modality	Aggregation	D1	D2	D3	Mean
RGB	Late fusion (clip)	55.25	66.22	67.33	62.93
	Late fusion (video)	55.40	66.21	67.31	62.97
	Average	55.32	65.6	65.91	62.28
	TRN	57.01	65.98	66.4	**63.13**
Event	Late fusion (clip)	50.19	61.69	60.64	57.51
	Late fusion (video)	51.39	62.27	62.39	58.68
	Average	49.65	63.73	62.52	58.63
	TRN	50.11	64.93	61.72	**58.92**
Flow	Late fusion (clip)	50.14	63.39	58.70	57.41
	Late fusion (video)	49.65	63.73	60.69	58.02
	Average	57.93	67.06	68.98	**64.66**
	TRN	57.93	68.59	62.78	63.10
RGB+Flow	Late fusion (clip)	63.12	70.46	68.37	**67.31**
	Late fusion (video)	61.10	69.84	68.78	66.57
	Average	60.08	69.15	70.16	66.46
	TRN	61.09	70.12	70.23	67.15
RGB+Event	Late fusion (clip)	56.40	67.72	67.76	63.96
	Late fusion (video)	55.94	68.35	67.70	64.00
	Average	57.93	67.06	68.98	64.66
	TRN	58.16	68.13	68.27	**64.85**

when dealing respectively with RGB, event and flow modalities. In the supervised setting, TSM outperforms the others in many scenarios. It presents improvements of 8% and 10% with respect to TSN and I3D with RGB input data and it shows accuracy increases of 7% and 10% with respect to the same architectures with RGB+Event modality.

The importance of temporal information. We expected the abovementioned differences in performances among the architectures because when dealing with this task the encoding of temporal information is crucial. Indeed TSN, as explained in Section 3.6.1, does not directly encode the temporal information so it looses some information of the sample video. I3D instead is a 3D CNN able to directly learn temporal features but it only reaches slightly better accuracy results when fed with event data and there are no big leaps in performances. This was to be expected as in [113] an attempt to use a 3D architecture instead of a 2D one on Voxel representation was made and the reached improvement was not relevant. TSM achieves the best results when dealing with event in cross-domain scenarios. This discovery can be a turning point for online applications. In fact, while 3D CNNs have high computational cost making their deployment on edge or wearable devices

Table 7.7. TSN results in cross-domain scenario.

TSN								
		Cross Domain						
Modality	Aggregation	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
RGB	Late fusion (clip)	31.81	27.81	29.73	34.02	32.87	36.71	32.16
	Late fusion (video)	31.79	27.86	29.94	34.12	33.00	36.89	32.27
	Average	32.13	28.39	31.95	33.57	32.64	35.46	32.36
	TRN	32.13	27.74	31.95	35.72	31.49	35.48	**32.42**
Event	Late fusion (clip)	27.01	28.95	29.89	37.36	31.23	37.72	32.03
	Late fusion (video)	27.08	28.55	28.23	39.77	31.49	39.29	32.40
	Average	28.80	29.26	29.65	37.22	31.72	37.86	32.42
	TRN	28.34	28.63	31.03	38.93	33.79	37.72	**33.07**
Flow	Late fusion (clip)	32.92	33.76	36.32	38.78	38.93	45.67	37.73
	Late fusion (video)	34.83	37.68	37.32	41.44	38.60	46.31	39.36
	Average	35.06	29.97	33.79	39.63	36.78	37.33	35.43
	TRN	35.88	35.71	39.77	41.24	40.1	45.94	**39.77**
RGB+Flow	Late fusion (clip)	35.22	30.58	35.81	40.73	36.27	41.04	36.61
	Late fusion (video)	35.50	30.23	35.25	40.91	36.76	41.35	36.66
	Average	35.27	31.29	35.09	36.91	37.96	42.07	36.43
	TRN	36.08	33.17	35.58	41.5	39.61	43.14	**38.18**
RGB+Event	Late fusion (clip)	34.99	30.49	32.26	36.62	34.07	37.08	34.25
	Late fusion (video)	34.18	30.79	33.00	36.93	33.92	37.42	34.37
	Average	35.06	29.97	33.79	39.63	36.78	37.33	35.43
	TRN	33.51	29.35	34.25	41.47	36.96	39.2	**35.79**

often impossible, TSM manage to model spatial and temporal information using uni-direction temporal shift. This technique (Section 3.6.4) can be used in online context since the shift is done from prior to current frames.

A focus on event modality. We can see that in cross-domain scenarios the event modality is slightly better than RGB as it shows, on average improvements of 1%. This is very crucial as this scenario is typical of real-world applications and event cameras have many technical and practical advantages with respect to conventional ones. This finding can lead the way to applications which were technically unfeasible before. In fact, even if optical flow is still the most effective modality in cross-domain contexts, it comes with a lot of technical issues and drawbacks such as high computational cost and high energy consumption. Besides, this is the first benchmark in which event modality is comparable to RGB. A recent paper [123] introduced the event version of ImageNet dataset [50] but the performances reached with the event modality were far below that of RGB state-of-the-art ones showing a gap of 50% in accuracy [123]. This outcome could also be caused by the intrinsic nature of the task we are dealing with. Indeed, in FPAR context, differently from Image Recognition one, the architecture must give much importance

to the motion information in order to solve it and these features come with event modality which does not focus on texture, colors and appearance of videos.

Multi-modal approaches. Throughout our entire work we have claimed the importance of investigating the usage of the event modality in combination with the RGB one. They bring orthogonal and complementary features which can lead to improvements in the performances. The former encodes motion information disregarding textures and colors which are instead carried by the latter. This hypothesis is confirmed by the reached results. As a matter of fact, regardless of the architectures employed, both in intra- and cross-domain scenarios, the combination RGB+Event performs better than RGB alone. In particular, TSN and TSM show average improvements of over 3% in the cross-domain scenario when fed with RGB+Event data streams.

7.3 Ablation studies

In order to understand the effects of the event representation, in this Section, we performed a couple of ablation studies to find the best performing Voxel choice. In particular, in Section 7.3.1 different number of Voxel channels are tried in order to understand the most effective one in the different models while in Section 7.3.2 we tried to move the channels of the Voxel representation into the temporal dimension before feeding them to the model.

7.3.1 Number of Voxel channels

Since we have a representation with a number of channels different from 3 but we are exploiting pre-trained backbones there are a couple of methods which can be used to transfer the adapt the model to the new data:

- replacing the first convolutional block with a new one and train it from scratch;
- repeating the weights of the first convolutional in a circualar way as shown in Figure 6.10;

The latter is the method adopted for the overmentioned experiments. It tries to limit the effects of having an input with a different number of channels in a pre-trained context.

However, despite the second option could help in using pre-trained information, both these methods may be damaging in a cross-domain scenario.

Table 7.8. Different Voxel channels accuracies on TSM Supervised

Modality	# Voxel ch.	Testing	D1	D2	D3	Mean
Event	9	Video	56,22	68,74	65,32	63,43
		Clip	57,47	68,30	66,45	64,07
	3	Video	55,61	68,09	66,40	63,37
		Clip	54,43	68,92	67,02	63,46
	1	Video	52,64	65,59	61,04	59,76
		Clip	54,64	64,95	61,65	60,41
RGB	-	Video	61,62	75,07	74,94	70,54
		Clip	61,15	75,29	74,47	70,30
RGB+Event	9	Video	59,69	75,99	72,80	69,49
		Clip	59,67	76,90	73,18	69,92
	3	Video	61,30	74,36	72,82	69,49
		Clip	60,89	74,82	73,01	69,57

Table 7.9. Different Voxel channels accuracies on TSM Cross Domain

Modality	# Voxel ch.	Testing	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
Event	9	Video	29,53	33,95	36,73	43,82	31,80	43,05	36,48
		Clip	29,42	33,85	36,76	43,82	34,84	44,33	37,17
	3	Video	27,91	30,71	39,75	40,94	35,48	40,90	35,95
		Clip	29,38	32,99	39,11	40,92	36,68	42,86	36,99
	1	Video	30,58	30,08	34,23	40,89	36,07	40,04	35,32
		Clip	31,07	29,51	31,83	40,59	33,61	38,64	34,21
RGB	-	Video	36,52	32,26	35,61	37,82	33,41	37,91	35,59
		Clip	37,02	32,03	35,02	38,28	33,92	36,47	35,46
RGB+Event	9	Video	33,54	36,59	35,30	45,39	38,21	43,91	38,82
		Clip	33,79	37,00	36,63	45,19	37,73	43,93	39,04
	3	Video	32,55	33,80	38,31	45,00	36,53	42,03	38,04
		Clip	32,99	34,06	40,43	45,04	36,35	42,36	38,54

Indeed, the research indicates that the early layers of the network are typically the most influenced by domain shifts as stated in [124, 125], and so changing the structure of the first convolutional layer of the network, may cause a specialization on the source domain while performing badly in the target domain. Indeed, when pre-trained layers are transferred properly, the network can make use of robust low-level features. To test this hyphotesis and to investigate the optimal number of Voxel channels we analyzed the differences of having Voxels with 1, 3 or 9 channels. In Figure 7.4 it is possible to see the qualitative results. They show that decreasing the number of channels the temporal resolution of the Voxel decreases, the borders are more blurred since the number of events included in a single channels is much higher so more "movement" is collected in less channels. However, to better understand the effect of the different number of channels we quantitatively

Table 7.10. Different Voxel channels accuracies on I3D Supervised

Modality	# Voxel ch.	Testing	D1	D2	D3	Mean
Event	9	Video	47,66	56,32	53,52	52,50
		Clip	44,16	54,28	51,07	49,84
	3	Video	50,32	58,33	57,99	55,55
		Clip	48,02	57,76	55,48	53,75
RGB	-	Video	53,67	61,12	60,70	58,49
		Clip	53,66	61,86	59,49	58,34
RGB+Event	9	Video	53,43	63,82	60,36	59,20
		Clip	50,91	60,86	58,28	56,68
	3	Video	53,90	62,39	61,07	59,12
		Clip	50,78	59,91	59,33	56,67

Table 7.11. Different Voxel channels accuracies on I3D Cross Domain

Modality	# Voxel ch.	Testing	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
Event	9	Video	35,26	34,98	33,41	35,55	35,33	42,90	36,24
		Clip	31,96	31,81	33,33	36,95	33,10	40,00	34,53
	3	Video	37,27	39,12	32,98	36,52	35,68	43,56	37,52
		Clip	36,92	37,21	29,94	34,91	34,61	41,81	35,90
RGB	-	Video	34,50	35,70	34,94	36,46	33,93	38,37	35,65
		Clip	32,36	34,58	31,52	33,94	33,54	34,64	33,43
RGB+Event	9	Video	35,72	37,78	35,50	40,48	36,65	42,30	38,07
		Clip	33,22	36,19	32,93	39,57	35,35	37,41	35,78
	3	Video	36,49	39,39	35,53	39,90	33,82	43,63	38,13
		Clip	35,35	37,74	33,13	39,72	34,10	38,76	36,47

tested it. Starting from the TSM model. The results obtained are shown in Table 7.9 and 7.8, the trend they show is not completely clear, however it is possible to understand that the 1 channel setting is not the best one. However, it is not clear from the results of the experiment on TSM alone which choice between 3 or 9 channels is significantly better, which is why we also went on to explore the behaviour of these two cases on the other two models. In Table 7.10, 7.11, 7.12 and 7.13 it is possible to see all the results obtained. From these it can be seen that switching from 9 to 3 channels often helps the performance a lot and even leads to improvements of more than 2%. As stated in the beginning, we assume this is due to the use of a network pretrained on RGB data with 3 channels that therefore behaves better when the input data is as similar as possible since it limits the effect of domain shift. Notice, in fact, the results in Table 7.13 and 7.12 where the effect of using just 3 channels is much more beneficial in the cross-domain case than in the supervised one. After having observed such improvement we tested it in the RGB+Event multi-modal scenario as well in order to observe

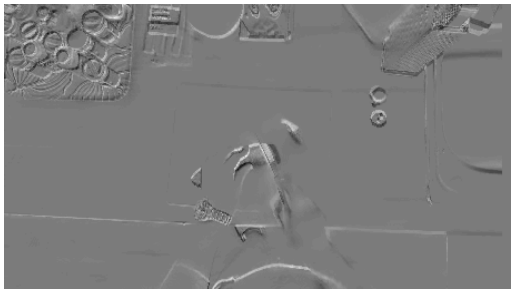
Table 7.12. Different Voxel channels accuracies on TSN Supervised

Modality	# Voxel ch.	Testing	D1	D2	D3	Mean
Event	9	Video	51,39	62,27	62,39	58,68
		Clip	50,19	61,69	60,64	57,51
	3	Video	52,06	64,36	62,54	59,65
		Clip	51,83	63,60	60,86	58,76
RGB	-	Video	55,40	66,21	67,31	62,97
		Clip	55,25	66,22	67,33	62,93
RGB+Event	9	Video	55,94	68,35	67,70	64,00
		Clip	56,40	67,72	67,76	63,96
	3	Video	57,24	68,56	66,74	64,18
		Clip	57,17	67,44	66,22	63,61

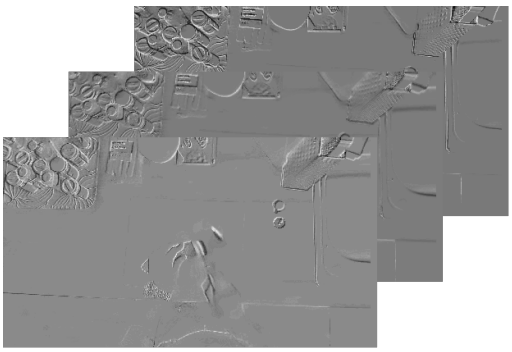
Table 7.13. Different Voxel channels accuracies on TSN Cross Domain

Modality	# Voxel ch.	Testing	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
Event	9	Video	27,08	28,55	28,23	39,77	31,49	39,29	32,40
		Clip	27,01	28,95	29,89	37,36	31,23	37,72	32,03
	3	Video	30,65	32,36	31,80	38,67	34,82	39,36	34,61
		Clip	31,17	31,41	32,49	38,66	34,92	37,84	34,42
RGB	3	Video	31,79	27,86	29,94	34,12	33,00	36,89	32,27
		Clip	31,81	27,81	29,73	34,02	32,87	36,71	32,16
RGB+Event	9	Video	34,18	30,79	33,00	36,93	33,92	37,42	34,37
		Clip	34,99	30,49	32,26	36,62	34,07	37,08	34,25
	3	Video	32,24	27,07	30,91	36,38	34,10	37,42	33,02
		Clip	32,46	26,91	32,08	36,85	35,07	37,41	33,46

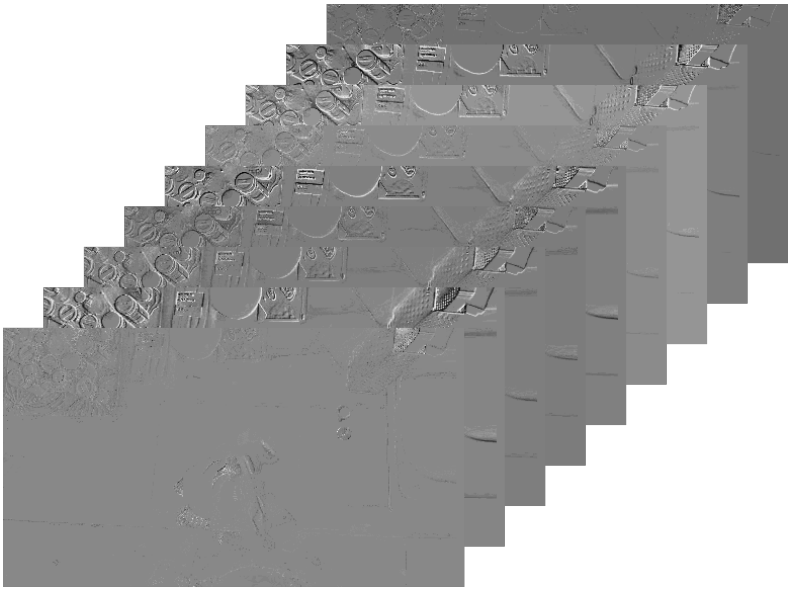
if the same effect would have happened in that case. The results reported in the Tables show that the change in the mean accuracy, in this case, is less evident, probably the RGB data used in combination with 9 channels Voxel data mitigate the domain shift effect. Finally, the choice of putting RGB results in comparison with the ones of 3 channels Voxel has been done in order to show the strength of event data in the cross-domain setting. In the different architectures the 3 channels Voxel is able to perform on average 2% better than the RGB modality. This confirms the hypothesized advantages of the event modality over the traditional RGB one in settings where environmental bias may decrease performance as the cross-domain one.



(a) Voxel Grid with 1 channel



(b) Voxel Grid with 3 channels



(c) Voxel Grid with 9 channels

Figure 7.4. Different number of Voxel channels

7.3.2 Exploiting Voxel channels

Different aspects have been analyzed in order to find the best way to feed Voxel data to the architectures. In particular, an interesting ablation study has been performed by moving the channels of the Voxel data into the temporal dimension before feeding it to the network. In fact, since the Voxel Grid is a representation that encodes the temporal dimension by discretizing it into different channels, the latter can be interpreted as different greyscale images acquired at different timestamps (or like a 1 channel Voxel computed with a temporal interval reduced by a factor $1/C$ where C is the number of channels of the original Voxel). Experiments on I3D model are reported in Table 7.14 and 7.15 where it is possible to observe that, effectively, in 3D models by moving channels in the temporal dimension we let the architecture model better temporal information. It increases the extraction of correlations among frames allowing a better temporal reasoning. From these ablation studies we found an even higher gap in performance among the RGB and the event modality in cross-domain setting as 3 channels Voxel data where channels are moved into the temporal dimension reach up to 38.76% accuracy with I3D against the best result of 35.56% of RGB data. This demonstrate the great potentialities of this new modality, and that further research can be done in order to find an even better way to help the model learn even better from event data by generating ad ad-hoc way of feeding the Voxel data into the classical CNNs. Notice, however, that this approach does not have the same benefit in a 2D model such as TSM where the temporal reasoning is more independent from the input form due to the temporal shift performed at several feature maps level. In fact, we tried to move some of the Voxel channels into the temporal dimension and run them like that on TSM but the results (which can be oserved in Table 7.16 and 7.17) are not as good as in the I3D case.

Table 7.14. Channels moving on I3D Supervised

Modality	# Voxel ch.	Testing	D1	D2	D3	Mean
Voxel	9	Video	47,66	56,32	53,52	52,50
		Clip	44,16	54,28	51,07	49,84
Voxel moving channels		Video	51,19	62,55	60,43	58,05
		Clip	47,77	60,00	57,88	55,22
Voxel	3	Video	50,32	58,33	57,99	55,54
		Clip	48,02	57,76	55,48	53,75
Voxel moving channels		Video	50,52	62,99	60,11	57,87
		Clip	51,37	62,12	56,90	56,80

Table 7.15. Channels moving on I3D Cross Domain

Modality	# Voxel ch.	Testing	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
Voxel	9	Video	35,26	34,98	33,41	35,55	35,33	42,90	36,24
		Clip	31,96	31,81	33,33	36,95	33,10	40,00	34,53
Voxel moving channels		Video	33,30	32,73	31,62	37,34	38,93	43,02	36,16
		Clip	33,81	33,62	31,11	38,40	37,75	42,67	36,23
Voxel	3	Video	37,27	39,12	32,98	36,52	35,68	43,56	37,52
		Clip	36,92	37,21	29,94	34,91	34,61	41,81	35,90
Voxel moving channels		Video	38,07	38,71	35,02	38,49	36,73	45,53	38,76
		Clip	38,95	38,27	34,48	38,66	32,90	41,84	37,52

Table 7.16. Partial channels moving on TSM Supervised

Modality	# Voxel ch.	Testing	D1	D2	D3	Mean
Voxel	9	Video	56,22	68,74	65,32	63,43
		Clip	57,47	68,30	66,45	64,07
Voxel moving channels		Video	56,55	67,97	64,42	62,98
		Clip	56,86	69,39	65,62	63,96

Table 7.17. Channels moving on TSM Cross Domain

Modality	# Voxel ch.	Testing	D1→D2	D1→D3	D2→D1	D2→D3	D3→D1	D3→D2	Mean
Voxel	9	Video	29,53	33,95	36,73	43,82	31,80	43,05	36,48
		Clip	29,42	33,85	36,76	43,82	34,84	44,33	37,17
Voxel moving channels		Video	29,27	34,84	35,81	42,36	35,05	40,52	36,31
		Clip	28,30	35,42	35,15	41,58	38,29	41,72	36,74

Chapter 8

Conclusions

We started our work by identifying the EPIC-Kitchens [9] dataset as the one of interest for the research due to its relevance in the field of egocentric vision tasks. Thanks to the nature of this dataset, composed of *multi-modal* data retrieved by settings with completely different affordances allowed us to experiment the performances both in *intra-*, *cross-domain* scenarios and *multi-modal* settings. To the best of our knowledge, this is the first analysis which allows the research community to compare and comprehend what are the benefits and the limitations of the event modality in the egocentric scenario. The first step conducted to reach our purposes has been the creation of a proper pipeline for the extrapolation of simulated event data from the RGB frames of the dataset. Thanks to the methodologies recently proposed by the authors of [22] and [23] we have been able to simulate the low latency of DVS-cameras and to extract synthetic events. The generation of a proper event dataset is a crucial point in this field, in fact event methods require a substantial amount of training data, which is scarce due to the novelty of event sensors in computer vision research. What we propose is an event extension of the Epic Kitchens dataset. As far as we know (in particular from the works in [1, 21]), this will be the biggest dynamic event dataset, and, in particular the first First Person event dataset (see Table 8.1).

Table 8.1. Dynamic event datasets dimension [1]

Dataset	# classes	Total samples	Avg. length (s)
DVS128	11	1,464	6.52
UCF50	50	6,681	6.8
EpicKitchens (**ours**)	8	10,094	2.84

Secondly, we provided a complete benchmark of the event data over the most important action recognition architectures such as TSN [6], I3D [7], TSM [12] and TRN [11]. To do so we used 9 channels Voxel Grid representations (proposed in [37]) for the event data in order to make them suited for classical RGB model topologies and adapted the first convolutional layer for the number of Voxel input channels. All modalities have been explored both in *single-* and *multi-modal* setting. For an in-depth comparative analysis we adopted both *clip* and *video* level testing methods, following the recent work provided in [126], in order to better understand the differences among the different models. The results unleashed the potential of event data in first person action recognition, in fact, this is the first benchmark in which event modality is comparable to RGB and overcomes it in in *cross-domain* setting.

Further research can be done in order to find a way to help the model learn even less environmental bias by generating ad ad-hoc way of feeding the Voxel data into the classical CNNs. An important step towards that direction has been the testing of different Voxel channels in Section 7.3 but we believe even more could be possibly done. Our contributions can also encourage the scientific community to develop new ways of tackling other issues such as egocentric action anticipation or even problems which traditionally do not belong to the AR context such as motion compensation, event tracking, image reconstruction.

Bibliography

[1] C. Liu, X. Qi, E. Lam, and N. Wong, "Aet-efn: A versatile design for static and dynamic event-based vision," 03 2021.

[2] D. Dev, *Deep learning with Hadoop build, implement and scale distributed deep learning models for large-scale datasets.* Packt Publishing, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[4] Y. Zhu, X. Li, C. Liu, M. Zolfaghari, Y. Xiong, C. Wu, Z. Zhang, J. Tighe, R. Manmatha, and M. Li, "A comprehensive study of deep video action recognition," *CoRR*, vol. abs/2012.06567, 2020.

[5] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *CoRR*, vol. abs/1406.2199, 2014.

[6] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool, "Temporal segment networks: Towards good practices for deep action recognition," 2016.

[7] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," 2018.

[8] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," 2019.

[9] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "The EPIC-KITCHENS dataset: Collection, challenges and baselines," *CoRR*, vol. abs/2005.00343, 2020.

[10] K. Bayoudh, R. Knani, F. Hamdaoui, and A. Mtibaa, "A survey on deep multimodal learning for computer vision: advances, trends, applications, and datasets," *The Visual Computer*, no. 9, 2021.

[11] B. Zhou, A. Andonian, and A. Torralba, "Temporal relational reasoning in videos," *CoRR*, vol. abs/1711.08496, 2017.

[12] J. Lin, C. Gan, and S. Han, "Tsm: Temporal shift module for efficient

video understanding," 2019.

[13] D. Damen, G. M. Farinella, A. Furnari, E. Kazakos, and P. Will, "Epic-kitchens - 2019 challenges report."

[14] D. Damen, H. Doughty, G. M. Farinella, A. Furnari, E. Kazakos, P. Will, and M. Jian, "Epic-kitchens-55 - 2020 challenges report."

[15] Benosman, "Event computer vision 10 years assessment: Where we came from, where we are and where we are heading to."

[16] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "Aer image filtering architecture for vision-processing systems," *IEEE Transactions on Circuits and Systems I-regular Papers*, vol. 46, pp. 1064–1071, 1999.

[17] G. K. Cohen, "Event-based feature detection, recognition and classification. robotics," 2016.

[18] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014.

[19] G. Cohen, S. Afshar, A. van Schaik, A. Wabnitz, T. Bessell, M. Rutten, and B. Morreale, "Event-based sensing for space situational awareness," 11 2017.

[20] E. Mueggler, B. Huber, and D. Scaramuzza, "Event-based, 6-dof pose tracking for high-speed maneuvers," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2761–2768, 2014.

[21] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *CoRR*, vol. abs/1904.08405, 2019.

[22] H. Rebecq, D. Gehrig, and D. Scaramuzza, "Esim: an open event camera simulator," in *Proceedings of The 2nd Conference on Robot Learning* (A. Billard, A. Dragan, J. Peters, and J. Morimoto, eds.), vol. 87 of *Proceedings of Machine Learning Research*, pp. 969–982, PMLR, 29–31 Oct 2018.

[23] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," 2018.

[24] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, and D. Scaramuzza, "Video to events: Bringing modern computer vision closer to event cameras," *CoRR*, vol. abs/1912.03095, 2019.

[25] T. U. of Edinburgh, "Bilateral filtering for gray and color images."

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[28] Wikipedia contributors, "Egocentric vision — Wikipedia, the free encyclopedia," 2021. [Online; accessed 12-October-2021].

[29] J. Munro and D. Damen, "Multi-modal domain adaptation for fine-grained action recognition," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Conference on Computer Vision and Pattern Recognition (CVPR), (United States), pp. 119–129, Institute of Electrical and Electronics Engineers (IEEE), Aug. 2020. Computer Vision and Pattern Recognition ; Conference date: 14-06-2020 Through 19-06-2020.

[30] E. Kazakos, A. Nagrani, A. Zisserman, and D. Damen, "Epic-fusion: Audio-visual temporal binding for egocentric action recognition," *CoRR*, vol. abs/1908.08498, 2019.

[31] M.-H. Chen, Z. Kira, G. AlRegib, J. Yoo, R. Chen, and J. Zheng, "Temporal attentive alignment for large-scale video domain adaptation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[32] A. Jamal, V. Namboodiri, D. Deodhare, and K. Venkatesh, "Deep domain adaptation in action space," Jan. 2019. 29th British Machine Vision Conference, BMVC 2018, BMVC 2018 ; Conference date: 03-09-2018 Through 06-09-2018.

[33] R. Arandjelovic and A. Zisserman, "Look, listen and learn," *CoRR*, vol. abs/1705.08168, 2017.

[34] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, "Event-based vision meets deep learning on steering prediction for self-driving cars," *CoRR*, vol. abs/1804.01310, 2018.

[35] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Ev-flownet: Self-supervised optical flow estimation for event-based cameras," *CoRR*, vol. abs/1802.06898, 2018.

[36] I. Alonso and A. C. Murillo, "Ev-segnet: Semantic segmentation for event-based cameras," *CoRR*, vol. abs/1811.12039, 2018.

[37] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based learning of optical flow, depth, and egomotion," *CoRR*, vol. abs/1812.08156, 2018.

[38] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-video: Bringing modern computer vision to event cameras," *CoRR*, vol. abs/1904.08298, 2019.

[39] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. `http://www.deeplearningbook.org`.

[40] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[41] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[42] F. B. Fitch, "Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133.," *Journal of Symbolic Logic*, vol. 9, no. 2, p. 49–50, 1944.

[43] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. New York: Wiley, June 1949.

[44] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.

[45] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[47] A. CAUCHY, "Methode generale pour la resolution des systemes d'equations simultanees," *C.R. Acad. Sci. Paris*, vol. 25, pp. 536–538, 1847.

[48] "[1412.6980] adam: A method for stochastic optimization." `https://arxiv.org/abs/1412.6980`. (Accessed on 09/27/2021).

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[51] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning*

and Inference, vol. 90, no. 2, pp. 227–244, 2000.

[52] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[53] Y. LeCun, "Generalization and network design strategies," in *Connectionism in Perspective* (R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, eds.), (Zurich, Switzerland), Elsevier, 1989. an extended version was published as a technical report of the University of Toronto.

[54] Y. Zhou and R. Chellappa, "Computation of optical flow using a neural network," *IEEE 1988 International Conference on Neural Networks*, pp. 71–78 vol.2, 1988.

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[57] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[58] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, "Towards good practices for very deep two-stream convnets," *CoRR*, vol. abs/1507.02159, 2015.

[59] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[60] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[61] M. Ma, H. Fan, and K. M. Kitani, "Going deeper into first-person activity recognition," *CoRR*, vol. abs/1605.03688, 2016.

[62] S. Sudhakaran, S. Escalera, and O. Lanz, "Gate-shift networks for video action recognition," 2020.

[63] S. Singh, C. Arora, and C. V. Jawahar, "First person action recognition using deep learned descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[64] C. Wu, C. Feichtenhofer, H. Fan, K. He, P. Krähenbühl, and R. B. Girshick, "Long-term feature banks for detailed video understanding," *CoRR*, vol. abs/1812.05038, 2018.

[65] G. Kapidis, R. Poppe, E. A. van Dam, L. P. J. J. Noldus, and R. C.

Veltkamp, "Multitask learning to improve egocentric action recognition," *CoRR*, vol. abs/1909.06761, 2019.

[66] S. Sudhakaran and O. Lanz, "Convolutional long short-term memory networks for recognizing first person interactions," *CoRR*, vol. abs/1709.06495, 2017.

[67] S. Sudhakaran and O. Lanz, "Attention is all we need: Nailing down object-centric attention for egocentric activity recognition," *CoRR*, vol. abs/1807.11794, 2018.

[68] M. Planamente, A. Bottino, and B. Caputo, "Joint encoding of appearance and motion features with self-supervision for first person action recognition," *CoRR*, vol. abs/2002.03982, 2020.

[69] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[70] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," in *International Journal of Computer Vision*, June 2010.

[71] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta, "Hollywood in homes: Crowdsourcing data collection for activity understanding," *CoRR*, vol. abs/1604.01753, 2016.

[72] J. H. Koo, S. W. Cho, N. R. Baek, M. C. Kim, and K. R. Park, "Cnn-based multimodal human recognition in surveillance environments," *Sensors*, vol. 18, no. 9, 2018.

[73] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (D. van Dyk and M. Welling, eds.), vol. 5 of *Proceedings of Machine Learning Research*, (Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA), pp. 448–455, PMLR, 16–18 Apr 2009.

[74] M. Alam, M. Bennamoun, R. Togneri, and F. Sohel, "A deep neural network for audio-visual person recognition," in *Biometrics Theory, Applications and Systems (BTAS), 2015 IEEE 7th International Conference*, vol. N/A, (United States), pp. 1–6, IEEE, Institute of Electrical and Electronics Engineers, 2015. Biometrics Theory, Applications and Systems (BTAS) 2015 ; Conference date: 08-09-2015 Through 11-09-2015.

[75] S. Sudhakaran, S. Escalera, and O. Lanz, "LSTA: long short-term attention for egocentric action recognition," *CoRR*, vol. abs/1811.10698,

2018.

[76] A. Furnari and G. M. Farinella, "Rolling-unrolling lstms for action anticipation from first-person video," *CoRR*, vol. abs/2005.02190, 2020.

[77] N. Crasto, P. Weinzaepfel, K. Alahari, and C. Schmid, "Mars: Motion-augmented rgb stream for action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[78] A. Cartas, J. Luque, P. Radeva, C. Segura, and M. Dimiccoli, "Seeing and hearing egocentric actions: How much can we learn?," *CoRR*, vol. abs/1910.06693, 2019.

[79] E. Kazakos, A. Nagrani, A. Zisserman, and D. Damen, "Slow-fast auditory streams for audio recognition," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 855–859, 2021.

[80] E. Morvant, A. Habrard, and S. Ayache, "Majority vote of diverse classifiers for late fusion," in *Structural, Syntactic, and Statistical Pattern Recognition* (P. Fränti, G. Brown, M. Loog, F. Escolano, and M. Pelillo, eds.), (Berlin, Heidelberg), pp. 153–162, Springer Berlin Heidelberg, 2014.

[81] Z. Liu, Y. Shen, V. B. Lakshminarasimhan, P. P. Liang, A. Zadeh, and L. Morency, "Efficient low-rank multimodal fusion with modality-specific factors," *CoRR*, vol. abs/1806.00064, 2018.

[82] S. Savian, M. Elahi, and T. Tillo, "Optical flow estimation with deep learning, a survey on recent advances," 01 2020.

[83] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981.

[84] Wikipedia contributors, "Optical flow — Wikipedia, the free encyclopedia," 2021. [Online; accessed 9-October-2021].

[85] Q. Fan, C.-F. Chen, H. Kuehne, M. Pistoia, and D. Cox, "More is less: Learning efficient video representations by big-little network and depthwise temporal aggregation," 2019.

[86] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?," 2018.

[87] C. Luo and A. Yuille, "Grouped spatial-temporal aggregation for efficient action recognition," 2019.

[88] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[89] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional learning of spatio-temporal features," in *Computer Vision – ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), (Berlin, Heidelberg), pp. 140–153, Springer Berlin Heidelberg, 2010.

[90] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," 2015.

[91] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "C3D: generic features for video analysis," *CoRR*, vol. abs/1412.0767, 2014.

[92] D. Damen, H. Doughty, G. M. Farinella, A. Furnari, E. Kazakos, J. Ma, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "Rescaling egocentric vision," *CoRR*, vol. abs/2006.13256, 2020.

[93] B. Jiang, M. Wang, W. Gan, W. Wu, and J. Yan, "Stm: Spatiotemporal and motion encoding for action recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[94] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain adaptive neural networks for object recognition," *CoRR*, vol. abs/1409.6041, 2014.

[95] M. Long and J. Wang, "Learning transferable features with deep adaptation networks," *CoRR*, vol. abs/1502.02791, 2015.

[96] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 2208–2217, PMLR, 06–11 Aug 2017.

[97] M. A. Sivilotti, "Wiring considerations in analog vlsi systems, with application to field-programmable networks," 1992.

[98] G. Indiveri, "Computation in neuromorphic analog vlsi systems," in *Neural Nets WIRN Vietri-01* (R. Tagliaferri and M. Marinaro, eds.), (London), pp. 3–20, Springer London, 2002.

[99] M. Mahowald and C. Mead, "The silicon retina," *Scientific American*, vol. 264, p. 76—82, May 1991.

[100] K. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip i: Outer and inner retina models," *IEEE transactions on biomedical engineering*, vol. 51, pp. 657–66, 05 2004.

[101] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 128 120 db 15 s latency asynchronous temporal contrast vision sensor," 2006.

[102] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *IEEE Journal of Solid-State Circuits*,

vol. 46, no. 1, pp. 259–275, 2011.

[103] T. Chin, S. Bagchi, A. P. Eriksson, and A. van Schaik, "Star tracking using an event camera," *CoRR*, vol. abs/1812.02895, 2018.

[104] M. Litzenberger, B. Kohn, A. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml, "Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor," in *2006 IEEE Intelligent Transportation Systems Conference*, pp. 653–658, 2006.

[105] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. V. Thakor, and R. Benosman, "Hfirst: A temporal approach to object recognition," *CoRR*, vol. abs/1508.01176, 2015.

[106] T. Delbruck, "Neuromorphic vision sensing and processing," in *2016 46th European Solid-State Device Research Conference (ESSDERC)*, pp. 7–14, 2016.

[107] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Hybrid, frame and event based visual inertial odometry for robust, autonomous navigation of quadrotors," *CoRR*, vol. abs/1709.06310, 2017.

[108] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, p. 142–149, Feb 2017.

[109] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, pp. 43–77, 02 1994.

[110] E. Herbst, S. Seitz, and S. Baker, "Occlusion reasoning for temporal interpolation using optical flow," August 2009.

[111] G. Long, L. Kneip, J. M. Alvarez, and H. Li, "Learning image matching by simply watching video," 2016.

[112] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive convolution," 2017.

[113] D. Gehrig, A. Loquercio, K. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data," 10 2019.

[114] Wikipedia contributors, "Compression artifact — Wikipedia, the free encyclopedia," 2021. [Online; accessed 9-October-2021].

[115] Wikipedia contributors, "Bilateral filter — Wikipedia, the free encyclopedia," 2021. [Online; accessed 9-October-2021].

[116] OpenCV.

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

[118] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," 2013.

[119] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[120] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[121] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[122] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "Hats: Histograms of averaged time surfaces for robust event-based object classification," 2018.

[123] J. Kim, J. Bae, G. Park, D. Zhang, and Y. M. Kim, "N-imagenet: Towards robust, fine-grained object recognition with event cameras," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2146–2156, October 2021.

[124] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," 2014.

[125] M. Planamente, C. Plizzari, M. Cannici, M. Ciccone, F. Strada, A. Bottino, M. Matteucci, and B. Caputo, "Da4event: towards bridging the sim-to-real gap for event cameras using domain adaptation," 2021.

[126] C.-F. Chen, R. Panda, K. Ramakrishnan, R. Feris, J. Cohn, A. Oliva, and Q. Fan, "Deep analysis of cnn-based spatio-temporal representations for action recognition," 2021.