POLITECNICO DI TORINO

Department of control and computer Engineering

Master of Science in Data Science and Engineering

Master Thesis

Long-Term temporal attention in Efficient Human Action Recognition Architectures

Supervisors:
Prof. Andrea Bottino

Candidate:
Lorenzo Atzeni

Academic year 2020-2021

Contents

List of Tables

List of Figures

Abstract

Human activity recognition focuses on automatically understanding the activity performed by humans. This field is of particular interest thanks to many real-world applications such as video indexing/retrieval, surveillance, Human-Machine interaction. In particular, this work focuses on RGB video data, which, in the last decade, has made huge improvements, mainly due to the progress made in the field of deep learning and the emergence of high-quality large-scale datasets. However, there are many challenges to overcome in the field of RGB Human Action Recognition, such as the high computational requirements of the current architectures, mainly due to the high dimensionality of the input. RGB videos are represented by two spatial dimension and a temporal one, which remains a major challenge. It is, in fact, difficult for the current architectures to reason about the events that happened far in the past or to grasp details dislocated in particular frames along the temporal dimension. The goal of this thesis is to tackle the problem related to the capability of the system to grasp information along the temporal dimension while maintaining low computational requirements for the system. This is achieved by combining efficient convolutional architectures, with a classification head based on the Transformer architecture. In this work, the MoViNets architectures family are used as backbones, while the Transformer-XL is used as Transformer head. The efficient MoViNet acts like a feature extractor, while the Transformer architecture processes the feature extracted. Thanks to its ability to process sequential information, the Transformer architecture reasons about feature extracted, attending to long-term relationships between frames and particular salient information contained in one or more particular frames. Furthermore, the architectures created are able to handle video streams and run real time on a mobile device. The use of three Transformer architectures having different computational requirements is investigated. The first experiments are performed on two small

datasets, namely HMDB51 and UCF101, where the results show that the Transformer architecture performs consistently with the original MoViNets architecture. To further investigate the new architectures, experiments on the Something-to-Something (SSv1) dataset are performed. This dataset is both more complex in the temporal dimension and larger than the dataset used in the previous experiments. The results on this dataset, show a 2.91% increase in performances for the lightest model over the original MoViNet architecture, reaching a new State-of-the-Art result. The results of the largest Transformer model, instead, report a slight reduction in performance with respect to the original MoViNet architecture, this may indicate that a larger Transformer head is necessary for larger models. The proposed new Transformer architectures have brought new SOTA results on SSv1, but further research is needed in order to explore the applicability to larger models. The proposed Transformer architectures are also considerable comparable with the standard MoViNets for HMDB51 and UCF101. Furthermore, the results show that the Transformer architecture with the lower number of parameters still performs consistently with both the architecture with higher parameters and the original convolutional architecture for HMDB51 and UCF101, providing a valid alternative in case of a memory constrain.

Introduction

Human activity recognition can be defined as the task of automatically understanding the activity performed by humans. The ability to recognize the tasks performed by humans is particularly interesting for different applications such as video indexing/retrieval, surveillance and human-machine interactions. Furthermore, recent progresses in the Deep Leaning field and the creation of larger datasets have brought improvements in reliability and performances, generating an increasing interest in this field.
The actions can be represented through different types of data modalities, including skeleton, RGB video, optical flow, accelerometer data and point clouds. This thesis work will focus on RGB Video based Action recognition, defined as the task of recognizing human actions from RGB videos. RGB Video Action Recognition faces important challenges, such as the high computational requirements of current Action Recognition Systems. These requirements are mainly due to the high dimensionality of the input, which has three dimensions: the temporal one and the two spatial dimensions. For the two spatial dimensions, a lot of work has been done in the field of image classification in order to provide efficient models, while the additional temporal dimension remains a major challenge.
Currently, the topic of efficient Human Action Recognition has been tackled using 3D convolutional neural network created with NAS (Neural architecture search). This practice is highly computationally expensive, but it does provide state-of-the-art results. The most recent and efficient networks are X3D [10] and MoViNets [22].
As far as the temporal dimension is concerned, there has been an effort to create architectures that are capable of capturing long-term relationships between frames. One example is [48] where the authors propose to use the transformer to model the outputs of multiple short clip segments computed by a 3D CNN, extending the receptive field of the network and improving the

details along the temporal dimension. The approach used in [20] proposed to remove the Temporal Global Average Pooling layer and to feed the output of the 3D CNN to the transformer. This would enable for better performances determined by a more accurate modeling of the temporal dimension.
The goal of this thesis is to address the limitations of the current models for Human Action Recognition, providing a new architecture that is able to tackle the problem related to high computational requirements of current Human Action Recognition systems and that is also able to attend to long-term relationships along the temporal dimensions. The approach is based on the idea proposed by [20] to remove the Temporal Global Average Pooling layer and to feed the output of the 3D CNN to the transformer architecture. The backbone is constituted by a 3D CNN of the MoViNets family, and the transformer of choice is the Transformer-XL [6]. This new architecture is both efficient and capable of keeping the original properties of the MoViNets such as low latency and possibility to receive as input a stream of frames.
The use of three Transformer architectures having different computational requirements is investigated. The first experiments are performed on two small datasets, namely HMDB51 [24] and UCF101 [41]. The results show that the architectures with the Transformer head perform consistently with the original convolutional architecture. To further investigate the new architectures, experiments on the Something-to-Something dataset are performed. This dataset is both more complex in the temporal dimension and is larger than the dataset used in the previous experiments. The results on this dataset, show a 2.91% increase in performances for the lightest model over the standard MoViNet, reaching a new State-of-the-Art result. The largest model reports, instead, a slight reduction in performance with respect to the standard MoViNet, we believe this may indicate that a larger Transformer head may be necessary for larger models. The experiments carried out in this thesis have been performed while doing an internship in Addfor S.p.a.
This thesis is structured as follows. Chapter 1 discusses the field of Human action recognition, its challenges and applications. Chapter 2 presents the current State-of-the-Art used in the field. The newly proposed architecture and their properties are discussed in Chapter 3. Chapter 4 describes the experiments performed and the results obtained, showing a comparison between the different architectures. Finally, conclusions are drawn.

Chapter 1

Human activity recognition

Human activity recognition focuses on automatically understanding the activity performed by humans. The actions can be represented through different types of data modalities, including skeleton, RGB video, optical flow, accelerometer data and point clouds. Different types of data contain different types of information, which can be used together to achieve better results, or can be used separately, providing different advantages or disadvantages in terms of accuracy, robustness, memory and computing requirements.

- Skeleton data includes the joints and the bones of the human body and how they move in space-time. Skeleton data is sufficient when the actions don't require information coming from the environment or from the objects that are being manipulated.

- "Optical flow is defined as the apparent motion of individual pixels on the image plane. It often serves as a good approximation of the true physical motion projected onto the image plane." [20] This kind of information does contain a lot of low-level information about the motion, but most of the information about the original image is lost, making it impossible to identify objects that are not in motion with respect to the camera. For this reason, in order to keep information about the original video, optical flow is often used together with RGB frames.

- RGB video data is the richest source of information compared to the previous two types, since it is the source from which both the skeleton and optical-flow information are generated.

1.1 Applications

Human action recognition has attracted a lot of interest in the field of computer vision, thanks to the wide variety of applications, such as:

- Video indexing/retrieval [16]: The ability to understand the actions that are being performed is crucial for indexing and retrieving video with specific content.
- Surveillance [29]: Surveillance applications require the system to identify malevolent actions.
- Human-Machine interaction [39]: Having machines that truly and easily understand our actions is a crucial step towards the integration of machines in our daily life, or in our work environment.
- Physical activity recognition [37]: Having sensors that are capable of recognizing human activities can help us keep track of our physical activity, with good consequences on our health.

1.2 Types

Body-worn sensor based

Body-worn sensor based Activity recognition uses sensors worn by the user to recognize the actions performed [4]. Sensors can be a smartphone or a smartwatch and are precise enough to provide estimation of the power consumption that happened through the physical exercise. it's also possible to recognize steps, running, stairs up/down and other useful information to monitor the physical activity [37].

Video based Action recognition

Video based Action recognition is the task of recognition of human action from videos. It's probably the most commonly used approach for Action Recognition, and it is currently investigated in the area of Deep Learning, with important results. In this category are included also the tasks that involve methodologies that manipulate features extracted from the video, i.e. optical-flow or skeleton data.

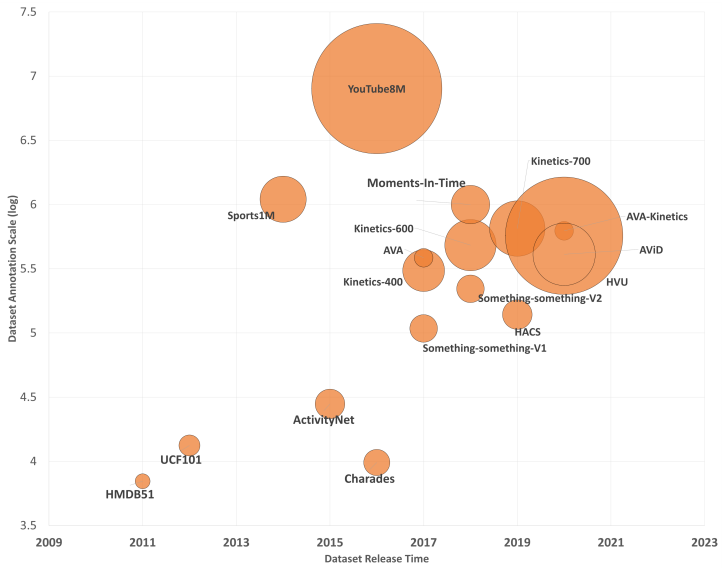


Figure 1.1: Increase in dimension of the dataset over the last decade, in log scale. The dataset release time is shown on the x-axis, the dataset dimension on the y-axis (log scale). The circle dimensions represent the number of labels. (from [51])

1.3 RGB Video based Action recognition

RGB Video based Action recognition is the task of recognition human action from RGB videos. It is a very important task due to numerous applications in the real world such as video indexing/retrieval, surveillance, human-machine interaction, physical activity recognition. In the last decade the field of RGB Video Based Action recognition has made huge improvements, manly due to the progress made in the field of deep learning, and the emergence of high-quality large-scale datasets. Fig. 1.1 shows the increase in dimension of the dataset over the last decade, in log scale.

1.3.1 Challenges

RGB Video based Human Action Recognition does face important challenges. The first challenge is defining a label space. Everyday actions are generally composed of sub-actions, following a hierarchical structure, where the most elementary actions are called Atomic Actions. This hierarchical structure

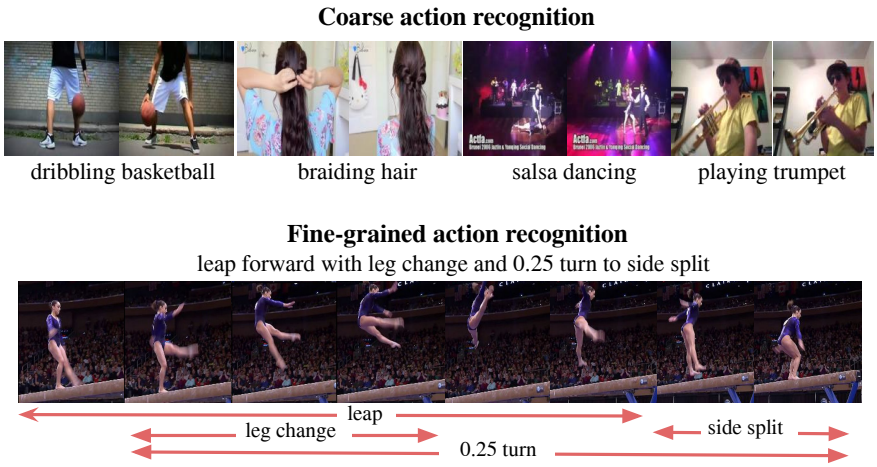


Figure 1.2: At the top: Coarse Action Recognition. It's possible to infer the action displayed just by a few frames. The environment does also give strong clues about the action performed. At the bottom: Fine-Grained Action Recognition. The athlete performs different actions that are characterized by small differences in the pose and a precise temporal sequence. The context information is not enough to classify the action. (from [49])

makes it often unclear the granularity to consider during the construction of a dataset, making the labeling choice not well-defined.
The second challenge is determined by the different ways an action can appear. The same action can in fact be seen from different viewpoints, or can be shown just partially, or can be composed of different sub-actions. All these different ways an action can appear make it hard for the action recognition system to generalize to unseen videos. For example the simple action of opening something can have very different sub-actions based on the object that the actor is manipulating. E.g. opening a door looks very different from opening an umbrella.
The third challenge is represented by the high computational requirements of current Action Recognition Systems. These requirements are mainly due to the high dimensionality of the input, which has three dimensions: the temporal one and the two spatial dimensions. For the two spatial dimensions, a lot of work has been done in the field of image classification in order to provide efficient models, while the additional temporal dimension remains a major challenge.

1.3.2 Types

Coarse Action Recognition

Chuhan Zhang et al [49] define coarse action recognition as the task of recognizing actions from videos where objects and background can strongly help infer the label of the video and the action can be easily determined by few frames. Typical datasets of Coarse Action Recognition are UCF101 [41], HMDB51 [24], and Kinetics [21]. In Fig. 1.2 at the top we can observe an example of Coarse Action Recognition: additional frames provide little to no additional information, furthermore there is no need to pay attention to details due to the amount of information coming from the environment.
While most of the datasets for video action recognition belong to this category, we have to stay aware that us humans have the ability to pay careful attention to details. While models and datasets created for coarse action recognition are actually suited for most applications, in reality application like human-machine interactions in work environments require a more Fine-Grained grasping of the actions.

Fine-Grained Action Recognition

Chuhan Zhang et al [49] define Fine-Grained Action Recognition as the task of recognizing actions from videos where the details are important both in the temporal and in the spatial domain. Subtle differences in the pose, or the specific sequence in which they appear on the video, may describe different actions. In the Fig. 1.2, we can observe the difference between Coarse and Fine-Grained Action Recognition.
This kind of datasets are suited to benchmark the capability of the model to grasp details. The capability of the model to pay attention to details dislocated along the temporal dimension is not only desirable, but a necessity in certain applications.

Discussion on the necessity of fine-grained details

Humans are extremely good at grasping fine-grained details along the temporal dimension. They are in fact capable to easily recognize subtle movements along the temporal dimension that would put in trouble modern Human Action Recognition Systems. In the Fig 1.5 someone pretends to put something in two different bowls without actually leaving it on any of the two. This

example is very complex for the temporal component because it is necessary to pay close attention to the details dislocated along all the video. Action Recognition in the real world is even more complex than staged datasets, setting the bar for defining a successful Action Recognition system even higher. Going back to the previously mentioned application regarding Human Machine Interaction, in our daily life most of the actions that we perform are not grossly defined by the environment, and they are often characterized by specific Fine-Grained manipulation of objects. In order to achieve a useful and successful collaboration between humans and machines through videos, it is necessary that the machines are able to spontaneously grasp the details of the actions performed. If the Action Recognition System is not capable of detecting such details, the user may be forced to repeat the same action, trying to make it unnaturally slower or clearer. Spontaneous collaboration with human-computer interfaces is not a problem restricted to Human action recognition, but it involves also speech recognition systems. Rebman et al. [38] provide a study about the acceptance rate of speech recognition systems in physicians work environment, showing how non-natural conversations and difficulty of the system to grasp the words of the physician can bring to a very low acceptance rate, up to consider the system counterproductive according to 31% of the physicians, while 36% where neutral and only 33% thought it was a good idea to introduce it in the work environment.

1.4 Datasets

Datasets are an important part of the improvements happened in recent years in RGb Video Action Recognition. In the field of deep learning, the increase in size of the datasets usually brings important improvements in performances. The increase of the datasets over the years is shown in Fig. 1.1. The datasets used in this work are the UCF101 [41], the HMDB51 [24] and the Something-to-Something dataset [12]. The first two being Coarse Action Recognition Datasets, while the last one is a Fine-Grained action recognition dataset.

1.4.1 UCF-101

UCF101 [41], with $13,320$ clips and 101 action classes, is currently considered a small dataset of coarse action recognition. It has been released in Novem-

Actions	101
Clips	13320
Groups per Action	25
Clips per Group	4-7
Mean Clip Length	7.21 sec
Total Duration	1600 mins
Min Clip Length	1.06 sec
Max Clip Length	71.04 sec
Frame Rate	25 fps
Resolution	320×240

Table 1.1: Summary of characteristics of UCF101. (from [41])

ber 2012 and at the time it was considered to be the largest video dataset available.
The clips are obtained from YouTube videos, and they have a resolution of 320×240 pixels and a frame rate of 25 FPS. The clips of one class are divided in 25 groups, which contain 4-7 clips each. Clips belonging to the same group have common features, such as background or actor.
The train/test split have been created to ensure that clips from the same group are not shared in train and test split, since the clips of one single group are obtained from a single video. The statistics of this dataset are presented in Table 1.1.

1.4.2 HMDB-51

HMDB-51 [24] is a commonly used benchmark dataset for Coarse Action Recognition. The dataset is nowadays considered a small dataset, with $6,766$ manually annotated clips extracted from different sources, including YouTube. This dataset contains a total of 51 action categories, each containing at least 101 clips. The camera motion has also been reduced through the use of offline algorithms. The statistics of the dataset can be found in Fig. 1.3.
Due to the acquisition modalities, broadly speaking internet videos, the actions are generally easily identifiable from the context and often few frames are enough for the classification. Some action of the dataset are displayed in

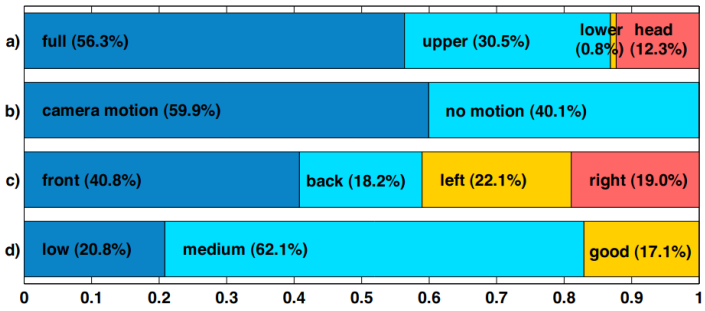


Figure 1.3: HMDB51 statistics relative to: a) visible body part, b) camera motion, c) camera view point, d) clip quality. (from [24])

Figure 1.4: Some classes of the HMDB51 dataset. The action are not challenging for the temporal dimension, nor they require to capture small differences in pose. (from [24])

Dataset Specifications	
Number of videos	108,499
Number of class labels	174
Average duration of videos (in seconds)	4.03
Average number of videos per class	620

Table 1.2: Something to Something V1 dataset summary. (from [12])

Figure 1.5: An example video, representing the action "Pretending to put *something* into *something*". Crowd-workers are asked to record the video representing the action and providing a placeholder for "*something*".

Fig. 1.4.

Training and testing set generation

The clips are provided with additional meta tags that include: camera viewpoint, presence of camera motion, video quality, number of actors involved. Those meta tags are used in order to create training and test set that not only have balanced classes, but they also present balanced meta tags. For each action category they select sets of 70 training clips and 30 test splits in order to obtain a 70/30 balance for each meta tag. Clips of the same video are also not present in both training and test set.

1.4.3 Something to Something V1

Something to Something V1 [12] is a dataset which involves manipulation of objects of daily life. The intent of the dataset is to force the model to learn characteristics and abstract concepts related to the objects that are being manipulated. The dataset consists of 108,499 short clips across 174 labels, with duration between 2 and 6 seconds, labeled with simple textual descriptions. The textual description describes details about the object and the action that is being performed. Learning the label will ideally provide the model with the ability to understand physical properties of the world.
The videos are specifically created for this dataset through crowdsourcing methodologies. By doing so it's possible to have a much better control over the creation of the dataset, as it is required for the particular intent for which this dataset has been created. The actors would create videos based on textual templates such as "putting [*something*] into [*something*]" where *something* is replaced by the actual objects. An example of video from the dataset is displayed in Fig. 1.5. The objects are chosen by the actor performing the action. The dataset presents an 80/10/10 split respectively for training, evaluation and test sets. The video created by a same actor are located on the same split. A dataset summary is displayed in Table 1.2.
For the reasons discussed above, this dataset can be considered a dataset of Fine-Grained Action Recognition.

1.4.4 Kinetics

Kinetics [21] is a large, video dataset for human action recognition. The Kinetics dataset has been built with the intent of providing significant benefits in accuracy when pre-training on it and fine-tuning on another dataset. The videos are obtained from YouTube, and each clip is captured from a different video. For this reasons this dataset provides a wide variety of actors and a lot of differences in how the actions are performed. They are also not professional videos, so they may include camera-motions and low-quality footage.
The actions of the dataset include: *Person Actions* like swimming or running, *Person-Person Actions* like kissing, hugging and *Person-Object Actions* like sharpening a pencil, welding. The dataset has 400 human action classes, with 400-1150 clips for each action, and each of the clips is around 10 seconds long. There are a total of 306,345 videos, divided in three splits. Training split with 250-1000 videos per class, validation split with 50 videos per class

and testing split with 100 videos per class.

1.5 Thesis goals

The differences between Coarse and Fine-Grained action recognition have been discussed in previous sections, highlighting the limitations of Coarse Action Recognition and the necessity of properly modeling the temporal dimension. Another important challenge is represented by the high computational requirements of current Action Recognition System, as discussed in Chapter 1.3.1.
The goal of this thesis is to address the limitation of the current models for Human Action Recognition, and provide a new architecture that is both computational efficient and capable of reasoning effectively over long-term RGB videos.

1.5.1 Modeling the Temporal Dimension

Most of the effort of this work will focus on properly modeling the Temporal dimension in order to capture:

- Short-term relationships between frames, defined as the ability to capture information about frames that are few steps away from each others.
- Long-term relationships between frames, defined as the ability to capture information about frames that are many steps away from each others, up to seconds or minutes away from each others.
- Single frame information, defined as the ability to capture information contained in a particular frame, ignoring non-relevant information contained in the remaining part of the video.

The first point is very important in order to capture low-level information, such as the moving of objects' edges and other motion cues, that are generally neglected by networks that perform late fusing of the temporal dimension. This kind of information can be considered as the information collected by feature extractor like optical-flow or 3D convolutions.
The ability to capture Long-Term information is a very important property as well. Imagine an application such as a kitchen assistant, it would be important for the assistant to remember things such as tasks that we have

already performed, or remember the object that you are manipulating even though they are currently out of sight. These are all properties that humans have, and they are necessary for a comfortable human-machine interaction. The ability to detect Single Frame information can be crucial in tasks where some action is performed very fast and can change completely the label of the action itself. An example of this kind of tasks can be found in the Something to Something dataset. Looking at the Fig. 1.5, it looks like the actor is going to put something into the bowls multiple times. It's just afterwards, when the actor leaves the bowls without any additional item, that you can classify correctly the action. Furthermore, what happened in the last few frames has changed completely the action itself.

1.5.2 Efficient online RGB Video Action Recognition

Most of the real-world application don't need state-of-the-art results, a few percentages increase in accuracy may not change dramatically the applicability of the model in a real-world environment. This is especially true when we consider that the increase in accuracy is often a consequence of an increase in computational requirements and/or an increase in the latency of the model itself.
Real-world applications have constrains in both latency and model computational requirements. Human-Computer interfaces generally have strong constrains in latency, due to the fact that we perceive as natural only close to real-time interaction, such as the ones that we experience with other people. An increase in latency can result in customers being frustrating and discouraged to use the technological device.
A focus of this thesis is to work on models that are feasible for deployment on mobile devices. Both online and offline versions of the model will be studied, due to the drop in accuracy often inevitable when using online models.

Chapter 2

Architercutres

This chapter will focus on the presentation of the current state-of-the-art architectures. The first sections will be dedicated to explaining the basic building blocks. The later sections will be instead dedicated to the explanation of current state-of-the-art Human Action Recognition systems, their characteristics, advantages and disadvantages.

2.1 Convolutional Neural Network

Convolutional neural network (CNN) is a type of Artificial Neural Network, based on the operation of multiplying a sub-sample of the input values by a kernel and summing them. The kernel slides along the input dimensions allowing for parameter reuse, furthermore it interacts only with a sub-sample of the input at every kernel evaluation, for this reason CNNs are characterized by a local receptive field. "With local receptive fields, neurons can learn to extract elementary features as oriented edges, end-points, corners. [...] These features are then combined by the subsequent layers in order to detect higher-order features" [27]. In addition, for some specific input data like images or videos, elementary feature detectors that are useful on one part of the input are likely to be useful across the entire input [27]. Convolutional neural networks have been successfully used in many computer vision tasks such as image recognition, image segmentation, video recognition and detection.
Formally they are refered to as convolutional neural network because they are in fact using a mathematical operation called convolution, described, in

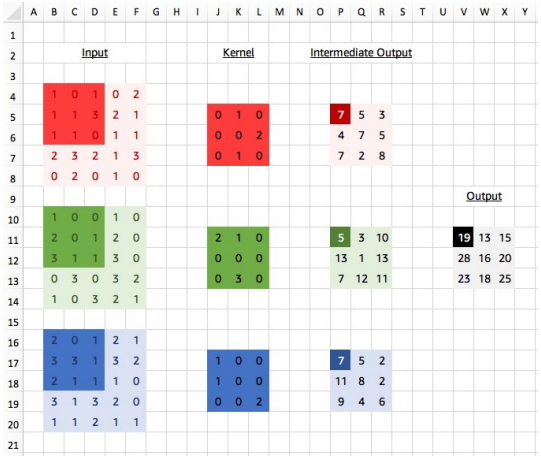


Figure 2.1: 2D convolutions [26]

the discrete form, as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.1}$$

Where x denotes the input function and w denotes the kernel function. The computer implementation of the convolution is performed using multidimensional arrays referred to as tensors [11]. The values of the functions x and w are zeros whenever we are outside the dimensions of the tensors, while they assume the values of the tensor itself whenever they are inside the tensor dimensions.

2.1.1 2D convolutions

2D convolutions take inputs of two dimensions (+ channel dimension) and compute the output by multiplying the input with the kernel, a two-dimensional matrix (+ channel dimension), and summing the results. Sliding the kernel over the two dimensions of the input we obtain the output matrix as displayed in figure 2.1.
Formally we can expand the previous formula of the convolution operation to convolve over two dimensions at the time:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{2.2}$$

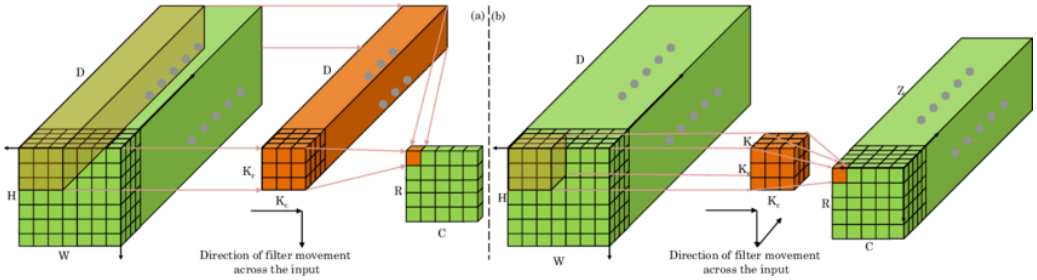


Figure 2.2: Comparison between 2D (a) and 3D convolutions (b) (from [31])

Or, since the convolution is commutative:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.3}$$

Where I is the two dimensional function representing the input, while K is the two dimensional function representing the kernel. [11]

2.1.2 3D convolutions

3D convolutions are a natural evolution of 2D convolutions for inputs with 3 dimensions (+ channel dimension). They are, for example, suited for videos which have one temporal dimension and two spatial dimensions. The kernels of a 3D convolution have 3 dimensions. For this reason a 3D kernel with dimensions (k, k, k), where k is the kernel dimension for each of the 3 dimensions, will have a higher number of parameters and will require higher computation for a single kernel evaluation when compared with the 2D kernel of the dimensions (k, k). The kernel slides along all of the 3 dimensions, increasing the number of kernel evaluations when going from an input dimension of (h, w) to (t, h, w), where h and w represent height and width, while t represent the temporal dimension. Compared with 2D convolutions, 3D convolutions are characterized by higher computational cost, due to a more expensive single kernel evaluation and to a higher number of evaluations caused by the additional dimension.
Formally, expanding the same concept of convolutions to the case of 3 dimensions, we obtain the following formula:

$$S(t,i,j) = (K * I)(t,i,j) = \sum_u \sum_m \sum_n I(t-u, i-m, j-n)K(u,m,n)$$

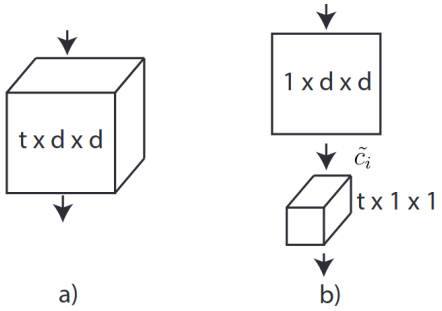


Figure 2.3: 3D convolution (a) and (2+1)D convolution (b). (a) The 3D convolution performs a single spatio-temporal operation using a kernel of size $t \times d \times d$, while (b) the (2+1)D convolution performs one spatial operation followed by a temporal one [43].

Now the I is the tree dimensional function representing the input, while K is the tree dimensional function representing the kernel.

2.1.3 (2+1)D convolutions

Given the problems related to the higher computation requirements introduced by 3D convolutions, there is a high interest in providing efficient way to compute the 3D convolution operation, or an equivalent one. A possibility has been presented by [43] where the 3D convolution has been approximated by a 2D convolution followed by a 1D convolution, decomposing the computation in two steps. Let c_i be the channels of the output of the ith layer and $\tilde{c}_i$ be the intermediate channel of the 2D convolution at layer ith. Let t, w, h be the kernel dimensions, w and h being the spatial dimensions and t the added $3rd$ dimension. The 3D convolution having kernel size $c_{i-1} \times t \times d \times d$ is replaced by one 2D convolution having kernel size $c_{i-1}, 1 \times d \times d$ followed by a 1D convolution having kernel size $\tilde{c}_i \times t \times 1 \times 1$. The hyperparameter $\tilde{c}_i$ can be chosen arbitrarily in order to increase or reduce the computational requirements and the number of parameters of the computation. An illustration of the two different types of convolutions can be found in Fig. 2.3.

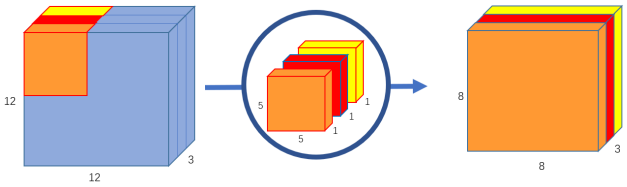


Figure 2.4: Depth-wise convolution. The filters and the corresponding input over which is convoluted are displayed with the same color. The input has dimension $w \times h \times c$ with $w = h = 12$ and $c = 3$. Each filter has dimensions $w \times h \times c$ with $w = h = 5$ and $c = 1$. (from [46])

2.1.4 Depth-wise Convolutions

In standard convolutions the filter is convolved over the input along its dimensions. The filter has the same channel dimension of the input and each kernel evaluation is performed over the whole channel dimension of the input. Depth-wise convolution instead is a type of convolution where each filter has channel dimension equal to 1 and the number of filter is equal to the number of channels of the input. Each filter is convoluted only with the corresponding channel of the input. A 2D Depth-wise convolution performed over a 2D input is presented in Fig. 2.4. In this case there are 3 filters and the input has the same channel dimension of the filter. The first filter is convoluted with the first channel of the input, the second filter is convoluted with the second channel of the input and so on, as displayed by the colors in the figure. Depth-wise convolution can be used to reduce the computational complexity of standard convolutions, on the other hand the channel dimension has to stay the same between input and output. For this reason they are frequently followed by an expansion (or reduction) layer in order to match the desired channel dimension of the output.

2.1.5 Squeeze-and-Excitation Blocks

A Squeeze-and-Excitation [15] block is a computational unit that operates upon a transformation $\mathbf{F}_{tr}$ that maps the input $\mathbf{X} \in \mathbb{R}^{H' \times W' \times C'}$ to an output $\mathbf{U} \in \mathbb{R}^{H \times W \times C}$. In the rest of the chapter, the transformation $\mathbf{F}_{tr}$ will represent a 2D convolutional operation.
The convolution is characterized by a an implicit and local modeling of the

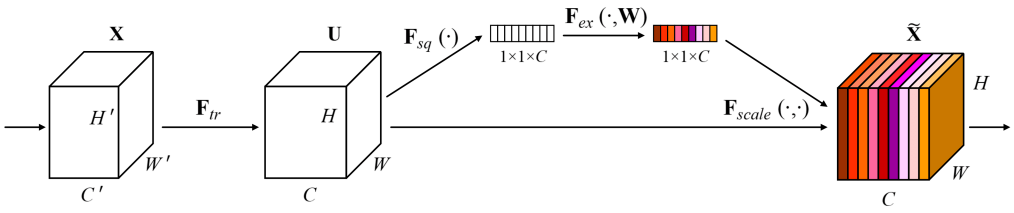


Figure 2.5: Squeeze and Excitation block [15].

channel dimension. Implicit because the channel information is contained only due to the summation during the kernel evaluation, local because every kernel evaluation has only the information coming from the subsample of the input that is using for the computation. The objective of the Squeeze-and-Exitation block is to provide a global explicit modeling of the channel dimension. Modeling the inter-channel dependencies can help the model focus on informative channel features, making it easier to learn from them without the noise coming from uninformative channel features. The Squeeze-and-Excitation block is composed of two different operation: *squeeze* and *excitation*.

Squeeze: Global Information Embedding

The modeling of the channel dimension in standard convolutions is limited by the fact that standard convolutions are characterized by a local receptive field, hence making impossible for the convolution to gather global information about the channel dimension. This problem is addressed by the *squeeze* operation.
The global channel information is gathered inside $\mathbf{z} \in \mathbb{R}^C$ by computing Global Average Pooling (GAP). For the c-th element of $\mathbf{z}$, the GAP operation can be expressed as such [15]:

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j) \tag{2.4}$$

where $\mathbf{u}_c$ indicates the c-th channel of the feature map $\mathbf{U}$.

Exitation: Adaptive Recalibration

The *squeeze* operation produces a global channel information vector $\mathbf{s}$. To fully exploit this information two main requirements have to be satisfied. The first one is that the information coming from $\mathbf{z}$ has to be processed by a non linear function capable of learning the interactions between the channels. The second one is that it has to be able to emphasize multiple channels, since the information will be likely distributed across different channels. A simple operation that is capable of meeting these criteria can be a two layer Feed Forward Neural Network with a sigmoid [11] activation function. The operation can be described as follows:

$$s = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})) \tag{2.5}$$

where σ represents the sigmoid function and δ the ReLU [32] function. The weights $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ and $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$ are, respectevely, a dimensionally reduciton layer with reduction parameter r and a dimensionally expansion layer that returns the channel dimension of the feature map $\mathbf{U}$. The parameter r can be chosen to make the operation more or less expensive. The final result of the block is obtained as:

$$\tilde{x_c} = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \mathbf{u}_c \tag{2.6}$$

where $\tilde{x_c}$ is the c-th element of the output $\tilde{X} = [\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_C}]$ and $\mathbf{u}_c \in \mathbb{R}^{H \times W}$. The overall structure is presented in Fig. 2.5.

Adaptation to 3D Convolutions

It's possible to extend the Squeeze and Excitation networks to operate upon a different transformation $\mathbf{F}_{tr}$ that maps the input $\mathbf{X} \in \mathbb{R}^{T' \times H' \times W' \times C'}$ to an output $\mathbf{U} \in \mathbb{R}^{T \times H \times W \times C}$. A transformation of this kind can be a 3D convolution. In order to make the *squeeze* transformation compatible it's enough to extend the global average pooling to 3 dimension as:

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{T \times H \times W} \sum_{k=1}^{T} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(k, i, j) \tag{2.7}$$

The *scale* transformation can be represented by the same formula:

$$\tilde{x_c} = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \mathbf{u}_c \tag{2.8}$$

Where now $\mathbf{u}_c \in \mathbb{R}^{T \times H \times W}$.

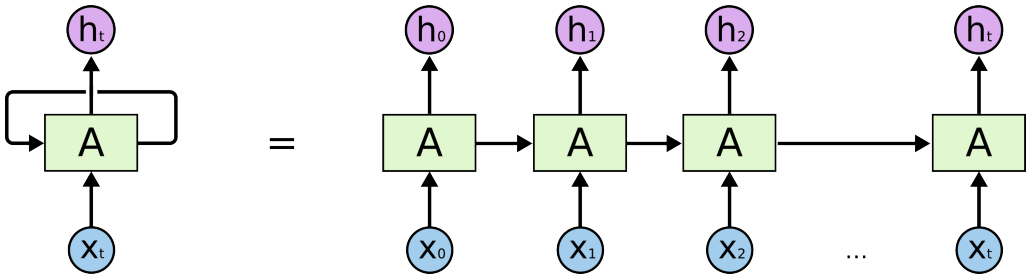


Figure 2.6: A traditional RNN layer. $x^{<t>}$ represents an element of the input sequence, where t represents the index of the element in the sequence. $a^{<t>}$ represents the output hidden state of the element $x^{<t>}$. $h^{<t>}$ represents the output of the element $x^{<t>}$. (from [33])

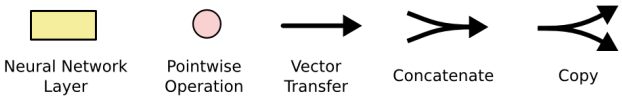


Figure 2.7: Notation used in the RNNs diagrams. (from [33])

2.2 RNN

A Recurrent Neural Network (RNN) is a particular type of Artificial Neural Network specialized for processing a sequence of values. In RNNs the output of the previous cell can be used as input for the next cell, allowing for information flow along the data sequence. [11] The overall structure is displayed in Fig. 2.6, where the cells are displayed in light blue, the input in green, the output in red.
The architecture of each cell can be described as follows:

$$a^{<t>} = tanh(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad (2.9)$$
$$h^{<t>} = a^{<t>} \quad (2.10)$$

where $tanh$ is the hyperbolic tangent function and W_{ax}, W_{aa}, b_a are parameters of the RNNs. $x^{<t>}$ represents an element of the input sequence, where t represents the index of the element in the sequence. $a^{<t>}$ represents the output hidden state of the element $x^{<t>}$. $h^{<t>}$ represents the output of the element $x^{<t>}$.
Each cell of the RNN layer shares the same parameters, for this reason this

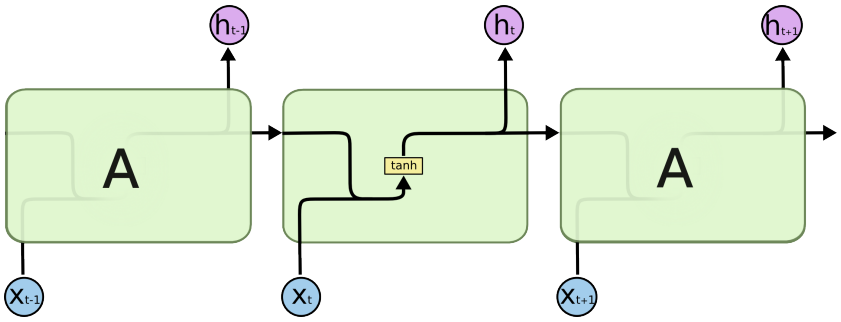


Figure 2.8: Architecture of a traditional RNN cell. (from [33])

type of network is called *Recurrent*. The architecture of a cell is displayed in Fig. 2.8. The notation used in this section and in the following is displayed in Fig. 2.7.

2.2.1 Advantages and Drawbacks

Thanks to the recurrent nature of RNNs the weights are reused at each time-step allowing to process inputs of any length without an increase in model dimension. RNNs are also capable to store information coming from the past elements, but they have difficulty in accessing information processed far in the past. Due to the iterative process RNNs don't allow for parallelization [11], making current GPUs and AI accelerators computationally inefficient. Traditional RNNs are also characterized by the problem of vanishing gradient [11]. This problem can be encountered when training Neural Networks with gradient-based methods and back propagation. During training, the parameters are updated proportionally to the partial derivative of the loss function. The partial derivative (gradient) may get smaller and smaller as it propagates through the layers, causing the Neural Network to train very slowly or to stop training completely.

2.2.2 LSTM

LSTM (Long Short-Term Memory) [13] is a particular type of RNNs introduced to solve the vanishing gradient problem, through the use of particular types of gates which provide at least one path for which the gradient does

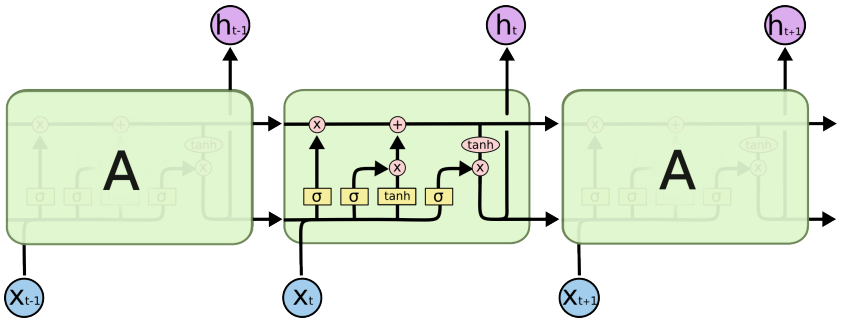


Figure 2.9: Architecture of a traditional LSTM cell. (from [33])

not vanish. The general formulation of a gate is the following:

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b) \tag{2.11}$$

where W, U, b are parameters of the model and σ is the sigmoid function. The t_{th} LSTM cell is also provided with a cell state in input indicated with c^{t-1}. Inside the LSTM architecture we have three types of gates: Γ_u the update gate, Γ_f the forget gate and Γ_o the output gate. Every gate outputs a vector with numbers between 0 and 1. The gate vector is multiplied element-wise with a second vector, which keeps the value if the gate value is 1 or changes the vector value to 0 if the gate values is 0.
The forget gate decides which values of the cell state to keep or to forget. The update gate decides which values of the input to use to update the cell state. The output gate decides which values of the cell state are relevant for the current output. The architecture of an LSTM can be described as follows:

$$\begin{aligned}
\tilde{c}^{<t>} &= tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) && (2.12)\\
c^{<t>} &= \Gamma_u \circ \tilde{c}^{<t>} + \Gamma_f \circ c^{<t-1>} && (2.13)\\
a^{<t>} &= \Gamma_o \circ tanh(c^{<t>}) && (2.14)\\
h^{<t>} &= a^{<t>} && (2.15)
\end{aligned}$$

The sign $\circ$ denotes the element-wise multiplication between two vectors. The sign $tanh$ represent the hyperbolic tangent function, W_c and b_c are parameters of the model. The overall architecture of an LSTM cell is displayed in Fig. 2.9.

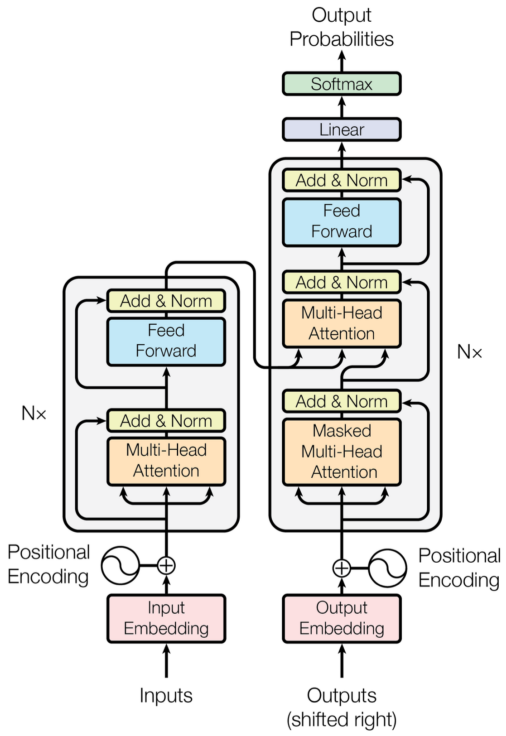


Figure 2.10: Transformer architecture (from [44])

2.3 Transformer

A transformer is a deep learning architecture that uses the mechanism of attention. The transformers were initially intended to be used for sequential data, for example in the field of Natural Language Processing (NLP) [44], but later the transformer has been used for Images and Video as a more general network. The innovative building block of the transformer architecture is the scaled dot-product attention.

2.3.1 Scaled dot-product attention

The input consists of queries keys and values, of dimension d_k, d_q and d_v, respectively, where $d_k = d_q$. For each query, the dot product with all the keys is computed. Considering the matrix of all keys vectors as K and the matrix of all queries vector as Q we could write the dot product as

a matrix multiplication QK^T. The result is then scaled by $\sqrt{d_k}$ and then passed through a softmax function before computing the multiplication with the matrix V containing all the values. The operation can be expressed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.16}$$

2.3.2 Multi-Head attention

In order to enhance parallelization, multiple attention are performed in parallel. Values, queries and keys are linearly projected h times to d_v, d_q and d_k dimensions, respectively. Each of these projections are used as input to perform attention in parallel. The output of the different h attention heads are then concatenated and linearly projected again to the dimension of the input.

2.3.3 Encoder and Decoder

The Transformer architecture composes of an encoder and a decoder. The encoder is used to encode the input $X_{1:n}$ into hidden states $H_{1:n}$, thus defining the mapping [45]:

$$f_{\theta_{enc}} : X_{1:n} \rightarrow H_{1:n} \tag{2.17}$$

The decoder is used to process the hidden states of the encoder and provide the appropriate output. Formally can be seen as an architecture that models the conditional probability of the target vector sequence $Y_{1:n}$ given the encoded hidden states $H_{1:n}$ [45]:

$$p_{\theta_{dec}}(Y_{1:n}|H_{1:n}) \tag{2.18}$$

The encoder is composed of N identical layers where N can be chosen arbitrarily. Each layer is generally composed of two blocks. The first block is composed of a Multi-Head attention, followed by a residual connection and a normalization operation. The second block is composed of a feed forward network and another residual connection and normalization operation.
The decoder is also composed of N identical layers. The decoder layer is slightly different, presenting in addition a masked Multi-Head Attention to ensure causality. The masked Multi-Head Attention attends to the values and

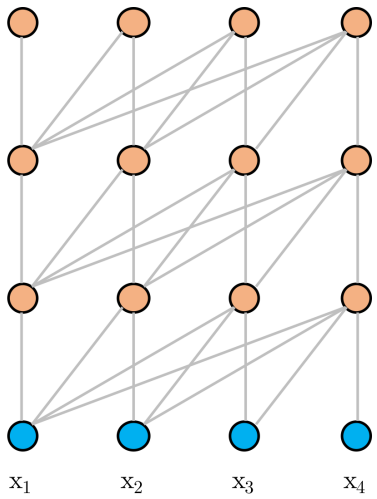


Figure 2.11: Causal Attention. The element x_t attends only to the input values x_{t_2} with $t_2 <= t$. (from [6])

keys coming from the encoder layers. The overall structure is represented in Fig. 2.10.

2.3.4 Positional encoding

The architecture described so far is a set architecture, meaning that it is not capable to make use of the order of a sequence. It is therefore necessary to inject into the input positional information such that the model is capable of infer the position of the values in the sequence. In the original transformer, the positional encoding are sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \quad (2.19)$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \quad (2.20)$$

Where *pos* is the position and i is the dimension. Each dimension corresponds to the function with a different frequency.

2.3.5 Causal Attention

Self-attention is defined as the attention where the keys and queries come from the same sequence in input. Causal attention is a particular type of

self attention where the queries are only allowed to look at preceding keys in input. This can be necessary in some particular task where the model is trying to predict the next value of a particular sequence. Causal attention is obtained by adding to the matrix multiplication a particular constant triangular matrix having zeros on the diagonal and below it, and minus infinity above the diagonal. This matrix can be considered as a Mask and It is denoted with M. Revisiting the formula (2.16) as discussed, we have:

$$CausalAttention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} + M)V \tag{2.21}$$

In practice, considering the matrix $\frac{QK^T}{\sqrt{d_k}}$, the values below the diagonal correspond to the interaction of the queries with the keys of the past, while the values above the diagonal correspond to the interaction of the queries with keys of the future. The mask allows to keep unchanged the interaction of queries with the past keys by adding the value zero. On the other hand, adding minus infinity to the interaction of the queries with keys of the future will produce an output of zero after the *softmax* computation.
The visual representation of causal attention can be seen in figure 2.11, where it's possible to observe that any given input element interacts only with elements of the past.

2.4 Human activity recognition architectures

This section will be focused on presenting some common architectures used today to solve the task of Human Activity Recognition, in particular:

- ConvNet-LSTM [7]: Convolutional network with a LSTM layer on top.
- Two-Stream Networks [40]: Networks provided with two streams that handle different data modalities.
- 3D ConvNets [22, 10]: Convolutional Networks with 3D filters.
- TSM [28]: Temporal Shift Module, a particular methodology that allows 2D convolutional Neural Networks to handle 3D input.
- Transformer architectures [2]: Transformer architecture adapted to handle 3D videos.

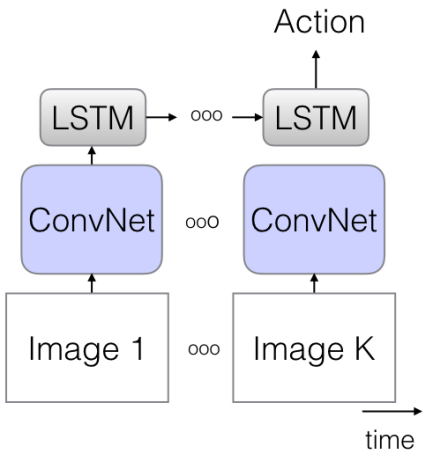


Figure 2.12: Convolutional network with LSTM head (from [21])

- 3D ConvNets - Transformers [20, 48]: Architectures that use both 3D ConvNets and Transformer architecture to take advantage of the features of both the architectures.

2.4.1 ConvNet-LSTM

One of the first approaches to video classification using deep learning was to adapt to videos the efficient architectures already used in image classification. This can be achieved by using recurrent neural networks, in particular very often LSTM, a particular type of recurrent neural network presented in Chapter 2.2.2. A ConvNet is used as a backbone to process the single RGB frames, while the LSTM is used to memorize information coming from the hidden states produced by the evaluation of the frames by the ConvNet (Fig. 2.12). This kind of approach is very convenient because it can build upon the improvements coming from the image recognition architectures, but it may fail to gather low-level information coming from the temporal dimension, since the network fuses the information of the images only very late in the architecture, just before the classification layer. It's also an inefficient approach, especially with the increase in the frame rate, since the image information from the video may be very redundant for frames close to each other.

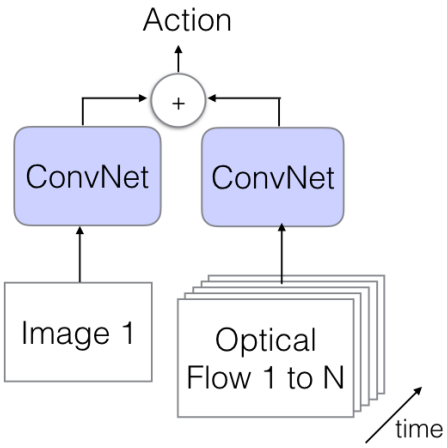


Figure 2.13: Two-stream networks. The network takes as input both RGB images and Optical Flow of the frames. The RGB frames contains crucial information about the object and the environment, the Optical Flow contains movement information. (from [21])

2.4.2 Two-Stream Networks

As mentioned in the Chapter 2.4.1, LSTMs may lack the ability to capture low-level motion details, due to the fact that they perform temporal fusion only very late in the architecture. To solve this problem, RGB 2D ConvNet is often combined with another 2D ConvNet which takes as input low-level motion features extracted from the video, e.g. optical-flow (figure 2.13). By providing information about the low-level movements, these models are able to perform better than their single stream version, with the trade-off of requiring more computational resources due to the addition of a second 2D ConvNet.

2.4.3 3D ConvNets

Another approach to Video Action Recognition using 3D convolutions (figure 2.14). Using 3D convolutions, the networks end up requiring more computational resources and having more parameters to train, while making it difficult to use the pretrained weights. Training a 3D ConvNet on image datasets like imagenet doesn't seem to provide the same benefits as it happens with 2D ConvNets. This problem was solved with the publication of the Kinetics

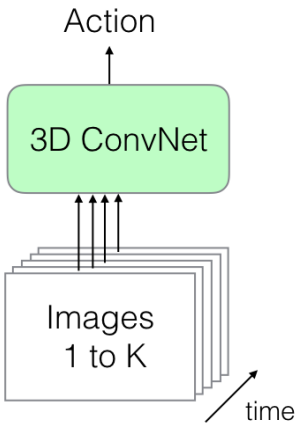


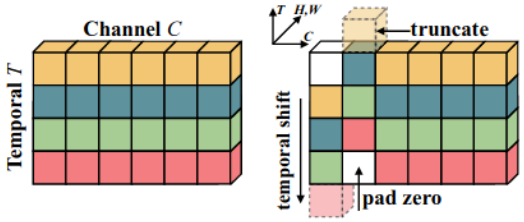
Figure 2.14: 3D convolutional neural networks. They take as input K RBG frames, and they perform evaluation of the action. (from [21])

dataset [21], now commonly used for pre-training 3D ConvNets. The Kinetics dataset is a large video dataset for human action classification, and thanks to its dimensions it is suited for pre-training Neural Networks with a large number of parameters. Due to the higher computational complexity, usually 3D CNNs are fed with short clips, typically 2-5 seconds [48]. Thanks to improvements to the architecture and to the ability to perform pre-training on the very large Kinetics datasets, now 3D ConvNets are state of the art in most benchmarks, only recently being challenged by Transformer-based architectures. Two of the most recent 3D ConvNets architecture are MoViNets [22] and X3D [10].

MoViNets

Mobile Video Networks (MoViNets) [22] are a family of computationally and memory efficient models that have the ability to work in an online environment. They are generated through the use of NAS (Neural Architecture Search) with the objective of creating the next standard for efficient video networks, deployable in real world applications. Two version of MoViNets are available:

- Base: suitable for efficient offline inference. The models take in input a clip of 50 frames (or more for more computationally expensive networks), and perform a single evaluation. Because of the high latency,



(a) The original tensor without shift.

(b) Offline temporal shift (bi-direction).

Figure 2.15: TSM (Temporal Shift Module). TSM module works by shifting the channel information along the temporal dimension, allowing to share information between neighboring frames. (from [28])

these models are not suited for online inference, but they are still deployable on mobile devices thanks to the low memory and computation requirements.

- Streaming: suitable for online inference. The models are able to take as input a stream of frames, one or more per evaluation, and perform evaluation in real time on mobile devices.

X3D

X3D [10] Networks are a family of computationally and memory efficient models released by Facebook AI Research. They were state of the art for efficiency before the release of MoViNets, and they are based purely on 3D convolutions. X3D Networks were reimplemented by the authors of the MoViNets paper which modified the training to allow for a higher number of frames in input, and they evaluated the models using a single clip per video. Single clip evaluation appears to be more efficient than multi-clip evaluation, making this model a stronger and more fair baseline for MoViNets.

2.4.4 TSM (Temporal Shift Module)

Conventional 2D CNNs are computationally efficient, but they fail to capture the temporal information. On the other hand, 3D CNNs are very good at capturing temporal components, especially at a low level, but they often lack

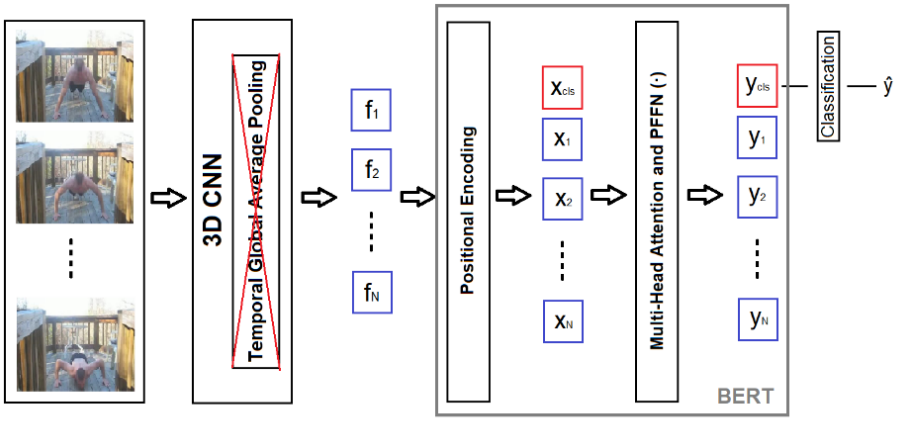


Figure 2.16: 3D ConvNets + Transformer architecture. The Temporal Global Average Pooling layer is eliminated and the activation is fed to a transformer architecture that handles the temporal modeling. (from [20])

the computational efficiency of 2D CNNs. TSM (Temporal Shift Module) [28], is a method to use highly efficient 2D CNNs for video classification. This is possible thanks to the fact that TSM allows the 2D models to reason about the temporal dimension. TSM is implemented simply by performing a shift of the channels along the temporal dimension like shown in Fig. 2.15. This implementation increases only marginally the computation required and it does not increase the number of parameters.

2.4.5 Transformers

Very recent approaches propose the use of Transformer-based architectures which already represent the state-of-the-art in sequence modeling tasks like machine translation and language understanding. At first they made their way to the field of image recognition with the famous VIT [8], and then to video understanding with ViViT [2]. They are generally more computationally expensive architecture, but they are nevertheless capable of reaching state-of-the-art results in common video understanding benchmarks [2].

2.4.6 3D ConvNets - Transformers

The transformer architecture has proven itself to be reliable in sequence modeling tasks. New approaches use the transformer architecture in order to model long term relationships between frames along the temporal dimension [20], [48]. The approach proposed in [20] is to use 3D convolutions as backbones which are capable of capturing low-level temporal features and are more efficient than current transformer architectures for videos. The authors remove the Temporal Global Average Pooling layer and they feed the output of the 3D CNN to the transformer architecture (Figure 2.16). The Temporal Global Average Pooling is a layer used in many 3D ConvNets just before the classification layers to reduce the input dimensionality by computing the average over the temporal dimension. By removing the Temporal Global Average Pooling layer, the temporal information is preserved and the output of the CNN is processed by the transformer as sequential input data, a field in which the transformer architecture has excelled, e.g. NLP.
The previous approach is capable of improving the capability of the network to capture information coming from the temporal dimension, but it does not solve the problem of being forced to handle short video sequences, a problem that afflicts 3D CNNs [48]. To solve this problem, a new approach [48] proposes to use the transformer to model the outputs of multiple short clip segments computed by a 3D CNN. This approach provides the model with a long temporal receptive field, enhancing the capability to remember details that happened a long time ago in the past. E.g. if someone has taken a pie from the fridge 20 seconds ago, the label eating a pie should be more likely, even in case the pie is not clearly visible anymore.

2.5 Efficient Networks: A Review

This Review is focused on models which are efficient and have low memory consumption. More computational expensive models perform better on benchmarks, but they are not of interest for this work, which focuses on architectures that can be used for inference on mobile devices.

Model	Params	FPS	Frames	Res	GFLOPS	Top-1
A0	3.1M	5	1x50	172	2.71	71.5
A0-stream	3.1M	5	1x50	172	2.73	70.3
MobileNetV3-S*	2.5M	5	1x50	224	2.80	61.3
MobileNetV3-S+TSM*	2.5M	5	1x50	224	2.80	65.5
X3D-XS*	3.8M	2	1x20	182	3.88	70.2
A1	4.6M	5	1x50	172	6.02	76.0
A1-stream	4.6M	5	1x50	172	6.06	75.6
X3D-S*	3.8M	4	1x40	182	7.80	73.4
X3D-S*	3.8M	5	1x50	182	9.75	74.3
A2	4.8M	5	1x50	224	10.3	77.5
A2-stream	4.8M	5	1x50	224	10.4	76.5
MobileNetV3-L*	5.4M	5	1x50	224	11.0	68.1
MobileNetV3-L+TSM*	5.4M	5	1x50	224	11.0	71.4
X3D-XS*	3.8M	2	30x4	182	23.3	72.3
X3D-M*	3.8M	5	1x50	256	19.4	76.9

Table 2.1: State-of-the-art comparison on kinetics-600. *Frames* indicates the number of frames in input with format *clips* × *frames*, *Res* the spatial resolution in input and *GFLOPS* the number of GFLOPS used in one video evaluation. *Top-1* indicates top-1 accuracy on Kinetics-600. *denotes a reproduced model (from [22])

2.6 State of the art comparison

When compared to images, processing videos can be extremely computationally expensive due to the presence of the extra temporal dimension. For this reason, outside the scope of beating state-of-the-art results, very large and heavy models rarely find an application to solve real-world problems. Considering the hardware at my disposal and the time for training very large models, the choice has been to focus on light and efficient models. As mentioned in the previous chapters, some of the best efficient networks in Video Action Recognition are: MoViNets [22], X3D [10] and 2D networks with TSM [28].
The comparison made by Kondratyuk et al. [22], includes the MoViNets family, the X3D family and MobileNetV3 with TSM. The comparison is shown in Table 2.1 where we can observe state-of-the-art results in relation to the

GFLOP necessary to perform evaluation of a single video of the Kinetics-600 [3] dataset. From top to bottom, the first section includes network with less than 5 GFLOPS, the second includes networks with GFLOP between 5 and 10, the third includes networks with GFLOP between 10 and 30. The X3D networks have been reimplemented in order to perform Single-clip evaluation, which appears to be more efficient than multi-clip evaluation. Single-clip evaluation is defined as the evaluation performed using only one clip of the same video, Multi-clip evaluation is defined as the evaluation using multiple clips of the same video.
MoViNets appear to be more efficient than the other models with which has been compared to. In the $0 - 5$ GFLOPS and in the $5 - 10$ GFLOPS range both A0 and A0-stream perform better than all the model proposed, while in the $10 - 30$ GFLOPS range the A2-stream model perform slightly worse than X3D-M*, while havivng around half the GFLOPS. The A2 model performs better than all the other models. From these results, MoViNets appears to be the current state of the art for efficient Video Action recognition, beating the TSM method applied to MobileNets and the X3D efficient networks mentioned before, for further details see Table 2.1. For these reasons the MoViNets family has been chosen as backbone for the architecture used during the experiments.

Chapter 3

Model Architecture

This chapter will cover topics related to the architectures that will be used to perform the experiments. The proposed architecture will be composed of a MoViNet [22] backbone and a transformer classification head. The approach used is to substitute the global average pooling layer with a transformer classification head, as proposed by [20] and discussed in section 2.4.6. The MoViNet architectures have been chosen as a backbone due to their efficiency, as explained in the previous chapter. The transformer architecture of choice is the Transformer-XL [6], the reasoning behind the choice of the transformer architecture will be explained in the following sections and his related to its high efficiency and low latency.
In this chapter the MoViNets architectures will be presented in details, as well as the Transformer-XL architecture that is being integrated with the MoViNets.

3.1 Transformer-XL Architecture

Transformer-XL [6] is a transformer architecture that enables learning beyond fixed length sequences and solves the problem of context fragmentation, further explained in Chapter 3.1.3. The overall architecture is composed by a recurrent mechanism, in particular the hidden states of each of the segments are saved and used to compute the hidden states of the next one, without recomputing them for each segment in input. The self-attention over the saved hidden state (produced by the previous evaluations) allows the model to capture very long-term relationships that go beyond a single segment.

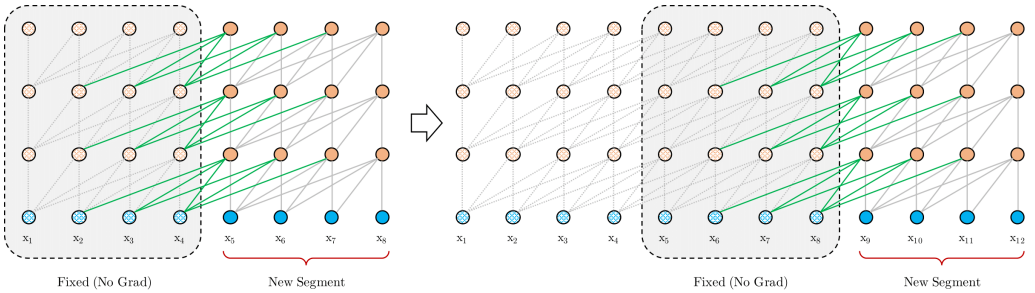


Figure 3.1: On the left, we can observe the first training step using the fixed activation and stopped gradient. On the right, the second training step is performed. The activation computed from the preceding segments are saved and used to compute the output of the following segment. The cached activations are displayed with a lighter color with respect to the activation computed in the current step. (from [6])

3.1.1 Segment-Level Recurrence with State Reuse

The recurrence and state reuse can be formally expressed as such [6]:
Let X^{seg} be the current segment and x_i^{seg} the i_{th} adjacent non overlapping segment of X^{seg}. The hidden states are indicated as h_i^l, the layer is indicated by the superscript l, where $l = 0$ indicates the input layer.
Then the l_{th} layer hidden state for segment x_{i+1}^{seg} is defined as h_{i+1}^l, and is computed as follows:

$$h_{i+1}^l = TransformerLayer(h_i^{l-1}, h_{i+1}^{l-1}) \tag{3.1}$$

where the computation of *keys*, *values* and *queries* is done as follows:

$$\tilde{h}_{i+1}^{l-1} = SG(h_i^{l-1}) \oplus h_{i+1}^{l-1} \tag{3.2}$$

$$q_{i+1}^l, k_{i+1}^l, v_{i+1}^l = h_{i+1}^{l-1} W_q^T, \tilde{h}_{i+1}^{l-1} W_k^T, \tilde{h}_{i+1}^{l-1} W_v^T \tag{3.3}$$

where the concatenation operation is indicated with $\oplus$ and SG indicates the operation of stop gradient. The overall structure of the recurrence methodology is illustrated in Fig. 3.1

3.1.2 Relative Positional Encoding

The idea presented so far is very appealing, but there is still a problem to be solved in order to make it possible to implement. From Chapter 2.3.4 we

remember that the transformer architecture operates on a set rather than on sequence data. In order to make it work on sequential data, it is necessary the use of positional encoding. Due to state reuse, standard sinusoidal positional encoding may result unable to provide enough positional information for the Transformer-XL architecture [6]. Formally, let x_i be the i-th input sequence of the model, $U_{1:L}$ the sinusoidal positional encoding from 1 to L where L is the length of the input sequence. Then we can express the computation of the hidden states of the first layer as follows [6]:

$$h_i = f(h_{i-1}, x_i + U_{1:L}) \tag{3.4}$$

$$h_{i+1} = f(h_i, x_{i+1} + U_{1:L}) \tag{3.5}$$

We can observe that x_i is associated with the same positional encoding as x_{i+1} making indistinguishable the position of $x_{i,j}$ and $x_{i+1,j}$ for any $j = 1, 2, 3, ..., L$. This would end up in a performance loss. In order to solve this problem, [6] proposes to use relative positional encoding and to inject them into the attention score of each layer, instead of statically injecting them into the embeddings of the first layer. By doing so, the positional information is computed for each layer and for each segment and hidden activation in input, allowing for accurate relative positional information. Furthermore, it is not necessary to know the absolute position of a token, it is enough to know the relative position of that particular token with respect to the query. This is particularly true in most application where the first token does not have any particular semantic meaning. While they introduce a new type of positional encoding, recent research suggests ALiBi [35] as a more suitable option, thanks to the important properties presented in Chapter 3.2. Particularly, ALiBi has shown a great ability to generalize to longer sequences than the ones used in training.

3.1.3 Vanilla Transformer Input Length Problem

The transformer improves over the RNN, Section 2.2, by solving the problems of vanishing gradient that afflicted RNN [34]. As a consequence, the transformer architecture is, in theory, capable of receiving as input an arbitrary long sequence. In practice, though, the Transformer model is characterized by a complexity $\mathcal{O}(n^2 \cdot d)$ [44], where n is the input length and d is the representation dimension, making it computationally expensive to increase the input length. For this reason a simple approach used for training

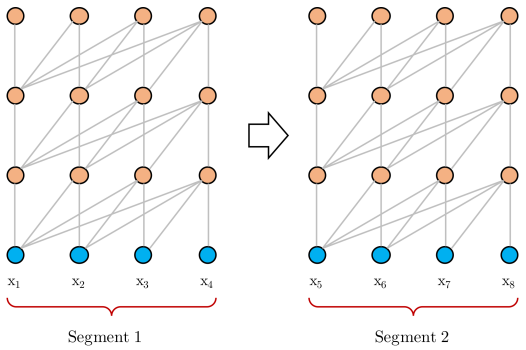


Figure 3.2: Training of vanilla transformer. A new segment is used to train at timestep two, without any information coming from segment one. The first tokens of each segment suffer from low context information. (from [6])

Attn Len	**How much vanilla transformer [1] is slower**
3,800	1,874x
2,800	1,409x
1,800	773x
800	363x

Table 3.1: Slowdown in terms of running time during evaluation. Evaluation is based on per-token time on one GPU. (From [6])

transformers is to split the input in arbitrary long sequences, and training the architecture without any information flow across sequences, ignoring the context information coming from adjacent segments of data. This idea is implemented by [1] and shown in Fig 3.2. There are two main limitation of this approach: the first one is that it's not possible to capture any long-term dependency that goes beyond the predefined input length, while the other one is that the sequences are split without taking in consideration any semantic relationships, causing a problem defined as context fragmentation [6]. This problem is caused by a reduced context information for the few first elements of the sequences, causing an inability of the network to infer correctly the labels, leading to inefficient optimization and inferior performances.

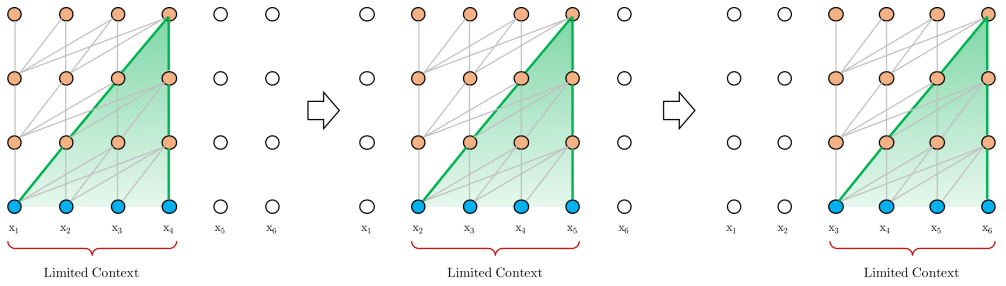


Figure 3.3: Vanilla transformer evaluation step. The input is shifted by only one token to provide maximum context information. The receptive field, represented in green, is limited by the current segment in input, represented with the color blue. The color yellow represents the activations. (From [6])

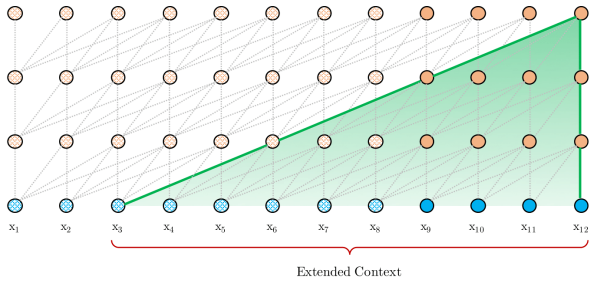


Figure 3.4: Transformer-XL evaluation step. The model uses the information computed in preceding steps to provide maximum context information for all the tokens. The context information is denominated as *Extended Context*. The cached activations are displayed in lighter colors. (from [6])

3.1.4 Inference

As described in the previous chapter, the vanilla transformer suffers from the context segmentation problem. For this reason during evaluation, at each step, the segment is shifted right by only one value and all the values have to be recomputed. This methodology is very computationally expensive, but it is necessary in order to have the maximum context during evaluation. The evaluation procedure is displayed in Fig. 3.3.
During evaluation the Transformer-XL, on the other hand, reuses the activation computed from the previous step allowing the model to provide maximum context information to all the tokens and achieve very fast evaluation time when compared with Vanilla Transformer. The process of evaluation performed with sequence length in input equal to 4 is illustrated in Fig. 3.4.
A comparison between evaluation latency of the Transformer-XL and Vanilla transformer can be found in Table 3.1

3.1.5 Training

During training the Transformer-XL takes as input the current sequence and the activations coming from the previous ones. The gradient is blocked and backpropagation is performed only for the activation of the new segments. The process is illustrated in Fig. 3.1.
As a comparison the Vanilla transformer only takes as input the current segment and it's not capable to gather any information from the previous ones. The process is illustrated in Fig. 3.2.

3.1.6 Receptive field

In vanilla transformers, the attention length is equal to the length of the segment itself, where attention length is defined as the number of elements that the transformer attends in one step. Due to the absence of cached activations, the attention mechanism is performed only on the current segment in input.
The Transformer-XL, on the other hand, could potentially have an attention length longer than the segment itself. It is in fact possible to expand the attention mechanism to attend directly to values multiple segments away from the current segments, which are cached. It is also possible to train the model

with a given attention length and use a longer length during evaluation, even though there are some concern about the ability of the model to generalize to longer input sequences. A discussion about the generalization problem and a promising solution can be found in Chapter 3.2.
Expanding the attention mechanism in evaluation is not the only way through which the Transformer-XL is capable of obtaining a wider receptive field. The receptive field of the Transformer-XL grows similarly to how it happens in convolutional neural networks, Chap. 3.4.1. In particular, the receptive field grows linearly with respect to the number of layers and the attention length, $\mathcal{O}(N \times L)$ where N is the number of layers and L is the attention length. This expansion of the receptive field happens faster than convolutions due to the fact that attention length is usually much larger than the kernel dimensions. The expansion of the receptive field is considerable, the extended receptive field is shown in Fig. 3.4. The receptive field of the vanilla transformer is limited by the segment length, and it is shown in Fig. 3.3.

3.2 ALiBi: Attention with Linear Biases

Attention with Linear Biases (ALiBi) [35] has been proposed in order to solve the problem of evaluating on longer sequences at test time, than the ones seen during training. ALiBi is a simple and efficient method which constists in adding a bias to the query-keys attention score proportional to the distance. Before continuing the analisys of the properties of ALiBi, few definitions are needed:
Press et al. (2021) define "extrapolation as the ability of the model to continue to perform well as the number of input tokens during validation increases beyond the number of tokens the model was trained on" [35].
Perplexity is defined as "a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample." [47].
One way to measure the ability to extrapolate is to measure perplexity with respect to the number of tokens used for evaluation. Transformers trained with ALiBi shows much higher ability to extrapolate with respect to existing methods. The improvement in perplexity on WikiText-103 [30] are displayed in Fig. 3.5.

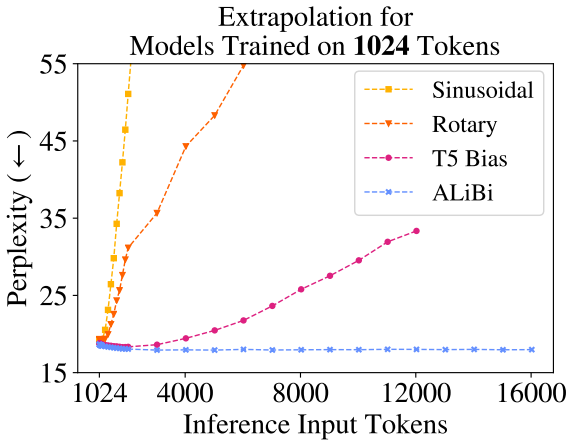


Figure 3.5: Extrapolation: as the number of input tokens at evaluation increases, Sinusoidal [44], Rotary [42] , T5 Bias [36] methods show a degrade in performances, while ALiBi shows no degrade in performances. On the x-axis the number of input tokens, on the y-axis the perplexity (lower is better). The models are trained on WikiText-103 [30] with sequence length of $L = 1024$. (from [35])

3.2.1 Architecture

Taking as a reference the attention formula 2.16, the computation of attention scores for a single query $q_i \in \mathbb{R}^{1\times d}$ can be defined as:

$$softmax(\frac{q_i K^T}{\sqrt{d_k}}) \tag{3.6}$$

where i indicates the i-th query of the input and $d = d_k = d_q$ where d_k and d_q are the key and query dimension, respectively. The matrix $K \in \mathbb{R}^{i\times d}$ containes the first i keys.
The ALiBi positional encoding can be expressed as a modification of the softmax argument with the addition of a bias term [35]:

$$softmax(\frac{q_i K^T}{\sqrt{d_k}} + m \cdot [0, -1, -2, ..., -(i-1)]) \tag{3.7}$$

Where the scalar m is a slope fixed before training, specific for each head. The original implementation suggests to set, for n heads, the slopes to the geometric sequence that starts at $2^{-2^{(-log_2 n+3)}}$ and uses the same value as its

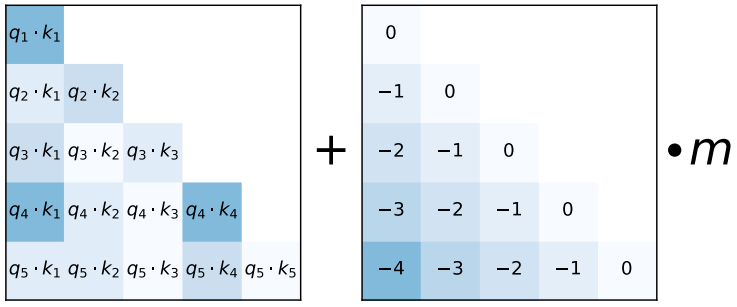


Figure 3.6: Computation of ALiBi positional encoding. Each attention score $(q_i \cdot k_j)$ is summed by a constant multiplied by a head-specific non-learnable parameter m. [35]

ratio. In particular for a model with 8 heads the slopes are the geometric sequence $\frac{1}{2^1}, \frac{1}{2^2}, \frac{1}{2^3}, ..., \frac{1}{2^8}$, that starts from $\frac{1}{2^1}$ and computes the next element by multiplying to $\frac{1}{2^1}$. Fig. 3.6 shows the computation of the attention scores with the ALiBi positional encoding.

3.3 MoViNets

Given the consideration reported in the State-of-the-Art results reported in section 2.6, the choice fell on the MoViNets [22] models. Particularly interesting are the streaming models that allow for online inference on mobile devices, with the tradeoff of a small loss in accuracy. The non-streaming versions of MoViNets are based on the use of standard 3D convolutional layers. The streaming version of the models is slightly different, since the 3D convolutional layers are substituted by (2+1)D layers, to obtain a lower inference time in case of mobile application using the TensorFlow lite framework. Excluding the architecture search results, the main architectural contributions made by this paper is the use of causal convolutions together with the stream buffers in the streaming version of these models. The concepts of causal convolution and stream buffer are presented in the section below.
The lighter MoViNets model is A0, the architectural description is provided in table 3.2. Under "Operation" is indicated in order: The kernel size in the format $(T \times H^2)$ where T represent the temporal dimension and H the height dimension, which is displayed with a power of two, representing that the same dimension is used also for the width. Then, always under "Opera-

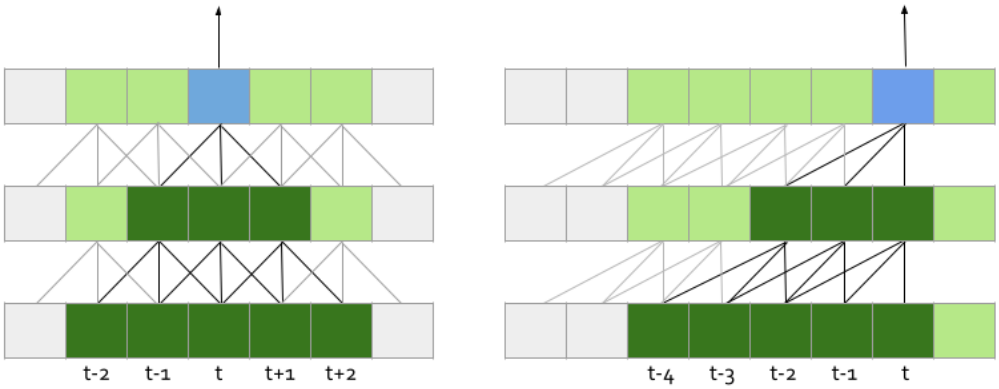


Figure 3.7: Standard Convolution on the left, Causal Convolution on the right. The kernel size is 3, padding is shown in white. Causal Convolutions are obtained by padding only the left side by the amount k-1, where k is the kernel size. Lighter color connections indicates the computation that is not involved in the current label prediction. [22]

tion" the base channel dimension and the expansion channel dimension are displayed. The base and the expansion channel refer to the block presented in Chapter 3.3.4.

3.3.1 Causal convolutions

Causal convolutions are a slight modification of standard convolution that ensures to process only information coming from the past, meaning that the prediction at time step t cannot depend on any of the future time steps. In Fig. 3.7 an example of the difference between standard convolution and causal convolution is displayed. In the example the standard convolution takes as input information coming from the past $(t-1, t-2)$, from the present (t) and from the future $(t+1, t+2)$. Causal convolution instead takes as input information coming only from the past $(t-1, t-2, t-3, t-4)$ and from the present (t). The causal convolution can be implemented by the use of padding to shift the input, such that the convolution considers only the input belonging to the past. Considering a kernel having temporal dimension k, it's possible to implement the causal convolution padding left by the amount $k-1$, corresponding to the operation of shifting the input by the same amount [22].

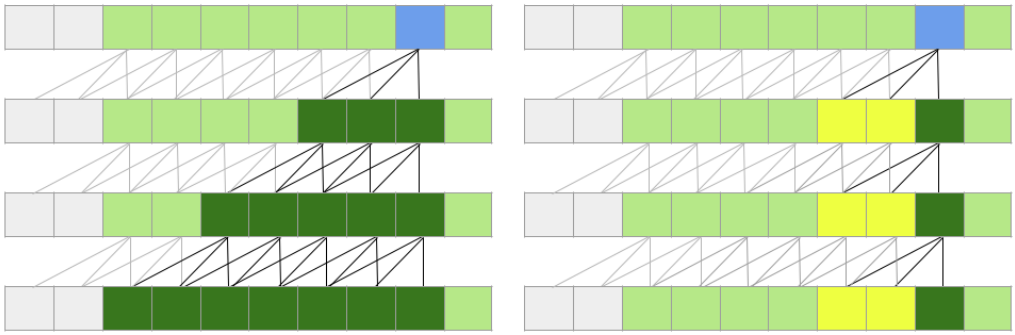


Figure 3.8: Causal Convolution on the left, Causal Convolution with stream buffer on the right. The kernel size is 3, the padding is shown in white, the stream buffer is show in yellow. The stream buffer is obtained by caching the activation obtained in the previous computations. Lighter color connections indicates the computation that is not involved in the current label prediction. (from [17])

3.3.2 Stream buffer

Causal convolutions enable the possibility to cache the activation previously computed and save computational resources. In MoViNets the activations are cached in the so-called stream buffer, illustrated in Fig. 3.8.
Formally, let X^{clip} be the current clip of video and x_i^{clip} the i-th adjacent non overlapping subclip of X^{clip}. Let B_i^l be the stream buffer for the i-th clip and l_{th} layer, initialized to zero for $i = 0$. For the input layer we have $l = 0$. Let b_l be the length of the temporal dimension of the stream buffer at layer l. It's possible to compute the activation of the first convolutional layer as follows [22]:

$$A_i^1 = f(B_i^0 \oplus x_i^{clip}) \tag{3.8}$$

And update the stream buffer for the next subclip:

$$B_{i+1}^0 = (B_i^0 \oplus x_i^{clip})[-b^0 :] \tag{3.9}$$

It's possible to extend this reasoning to the following layers:

$$A_i^{l+1} = f(B_i^l \oplus A_i^l) \tag{3.10}$$

And update the stream buffer for the next subclip:

$$B_{i+1}^l = (B_i^l \oplus A_i^l)[-b^l :] \tag{3.11}$$

where the selection of the last b^l elements is indicated as $[-b^l :]$, and the concatenation operation is indicated with $\oplus$. The model accumulates new information coming from the streaming video inside the stream buffer allowing for a variable number of frames in input. It is in fact possible to choose any amount of frames per subclip T^{clip}. This allows for lower or higher memory consumption, using a smaller or larger per subclip temporal dimension T^{clip}, respectively. Thanks to the stream buffer it is in fact possible to select a subclip having temporal dimension $T^{clip} = 1$. This choice reduces memory consumption and per-frame latency, since it allows to compute the output frame by frame. A comparison of peak memory consumption with and without the use of stream buffers (with $T^{clip} = 1$) shows a memory consumption 2 to 15 times lower using the architecture with stream buffers. For a complete overview see Table 3.5.

3.3.3 Fusing the concepts of stream buffer and state reuse

The MoViNets stream buffer and the Transformer-XL state reuse are conceptually very similar. Since in the following sections the two architectures will be used in the same model, a common view of the two modalities is provided.
Formally, let A^l be the activation of layer l, A^0 the input of the model, and A^l_i the i-th adjacent non overlapping segment of A^l. B^l_i is defined as the activation stored for the i-th segment and l-th layer. In the case of convolutional layers B^l_i is initialized to zero for $i = 0$. Let b_l be the number of maximum activation to store at layer l. It's possible to compute the activation of the next layer as follows:

$$A^{l+1}_i = f(B^l_i, A^l_i) \tag{3.12}$$

And update the activaiton stored for the next subclip as follows:

$$B^l_{i+1} = SG((B^l_i \oplus A^l_i)[-b^l :]) \tag{3.13}$$

where the selection of the last b^l elements is indicated as $[-b^l :]$, and the concatenation operation with $\oplus$. SG indicates the stop gradient operation. The function f can be expressed as such in the case of convolutional layer:

$$f(B^l_i, A^l_i) = ConvolutionLayer(B^l_i \oplus A^l_i) \tag{3.14}$$

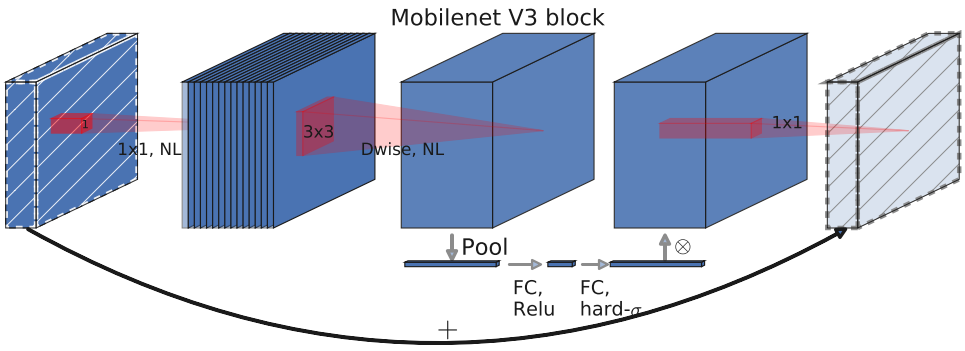


Figure 3.9: The MobileNetV3 convolutional block. It is characterized by an expansion block, a depth-wise convolution followed by a Squeeze-and-Excite layer and a reduction layer. [14]

or in the case of transformer layer:

$$f(B_i^l, A_i^l) = TransformerLayer(B_i^l, A_i^l) \tag{3.15}$$

Where the computation of *keys*, *values* and *queries* is done as follows:

$$\tilde{A}_i^l = B_i^l \oplus A_i^l \tag{3.16}$$

$$q_{i+1}^{l+1}, k_{i+1}^{l+1}, v_{i+1}^{l+1} = A_i^l W_q^T, \tilde{A}_i^l W_k^T, \tilde{A}_i^l W_v^T \tag{3.17}$$

From this chapter B will represent the stream buffer for both convolutional and transformer layers.

3.3.4 Convolutional Block

The MoViNets convolutional block is based on the MobileNetV3 [14] convolutional block. The original mobilenet block is characterized by a expansion layer, followed by a depthwise convolution, Section 2.1.4, a Squeeze-and-Excitation [15] layer and a projection layer to project the output to a lower dimension. The original MobileNetV3 block is modified by expanding the 2D convolution to 3D convolutions in order to deal with 3D video input, i.e. the 2D convolutions are substituted with 3D convolutions. The Squeeze-and-Excitation layer is modified as discussed in Section 2.1.5.

3.4 The Proposed Architecture

The transformer architecture has proven itself to be capable of handling very well sequence data. This ability has been exploited initially in the field of Natural Language Processing, reaching later successful results also in other fields. Recently the transformer architecture has proven itself very useful also in the field of Video Action Recognition. In [35] they replace the Temporal Global Average Pooling layer at the end of 3D ConvNets with a Transformer architecture, leading to important improvements in accuracy. In [48] thanks to the Transformer architecture they are capable of attending to the entire span of the video, reaching State-of-the-Art results. They use 3D convolutions and pooling to produce a representation of the clip, then they use the transformer to interact between the representation of the different clips of the same video. A fully transformer architecture has been proposed which is capable of achieving State-of-the-Art accuracy, even though it is computationally expensive. [2]
To the best of my knowledge, no previous work has been done to integrate transformer architecture in efficient models, in order to improve the model capabilities while keeping the architecture computationally efficient.
In this work, the efficiency of the MoViNets is combined with the ability of processing sequence data of the transformer architecture. Taking inspiration from [35] the Temporal Global Average Pooling layer at the end of MoViNets is replaced with a Transformer-XL architecture. The aim of this work is to improve the previous integration of the Transformer architecture in the field of Video Action Recognition, allowing for a better modeling of the temporal dimension while maintaining high efficiency and low latency. The proposed architecture with Transformer inner dimension $d_{model} = 192$ and MoViNet-A0 as backbone is displayed in Table 3.9.

3.4.1 Modeling the Temporal Dimension

Modeling correctly the temporal dimension can be a very important and challenging task for Video Action Recognition. The ideal architecture has to be able to capture:

- Short term relationships between frames: defined as the ability to capture information about frames that are close to each others.
- Long term relationships between frames: defined as the ability to cap-

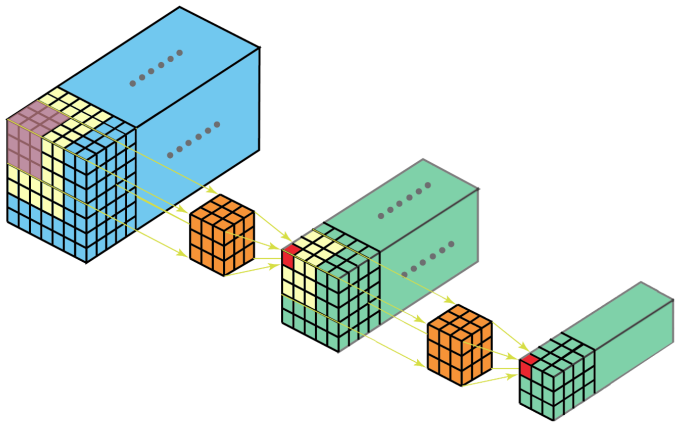


Figure 3.10: Locality in convolutions, The receptive field of convolutions grows linearly with the number of layers in the network and the kernel dimension.

ture information about frames that are many steps away from each others.

The ability of capturing short-term relationships can be interpreted as the ability to capture low-level information, as for example low-level motion information of the objects. This kind of features can be exploited through the use of kernels and convolutions or by low-level feature extractor like optical-flow.
Capturing long-term relationships between frames can be very important, since a particular action could be arbitrarily long or could need some information from a long time ago in the past. E.g. if someone has taken a pie from the fridge 20 seconds ago, the label eating a pie should be more likely even in case the pie is not clearly visible while the actor is eating it. This type of information can be captured through the use of the attention mechanism and its ability to handle long sequences of data.

Limitation of 3D convolutions: Local Receptive Field

Convolutions are capable of capturing low-level temporal features thanks to the property of locality and weight reuse [43]. While these properties do have many important and useful consequences, they also have limitations due to

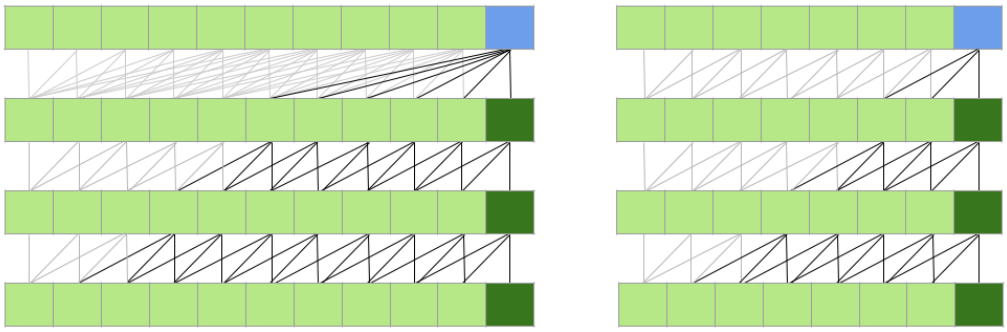


Figure 3.11: Difference in receptive field. On the left the receptive field of a convolution architecture with the last layer being a transformer layer. On the right, the receptive field of a convolutional architecture. The convolutional layers has been chosen to have kernel dimension $k = 3$ and the transformer architecture has *attention length* equal to 6.

the bias induced into the model itself.
The concept of locality allows the model to extend his receptive field only in the subsequent layers, which combine the features to obtain higher-level features. The property of locality and the corresponding receptive field are displayed in Fig. 3.10. In particular the receptive field of the model expands linearly with respect to the number of layers N and to the kernel dimension k, for each input dimension, i.e., $\mathcal{O}(N \times k)$.
Due to the usage of low values for the kernel dimension k, even on the latest layers, the model receptive field may not be large enough to include the all the input. The receptive field is usually expanded by the final Global Average Pooling layers commonly used in ConvNets.

Transformers: Expanding the Receptive Field

In the proposed architecture the Temporal Global Average pooling is replaced with few layers of the Transformer-XL architecture. As discussed earlier, the Transformer-XL has a low latency which makes it a good candidate for building efficient architectures. The Transformer-XL also has a large receptive field, making it suitable to replace the Global Average Pooling. In fact, although the average allows you to extend the receptive field at will, also makes the extrapolation of content belonging to individual frames more difficult for the network. Furthermore, computing an average over many

frames may result in an information loss due to the fact that every frame is weighted the same way. This is a suboptimal behavior for a layer responsible for modeling the temporal dimension, especially if we consider Fine-grained Action Recognition, where a single particular frames can be critical to the recognition of the task itself. On the contrary, the transformer, thanks to the attention mechanism, is capable of selecting the most important activations and use them for classification. The transformer architecture, in one time step, can attend to as many values as the value of the attention length of the model. It is not unusual to have models with an attention length of 512 or larger.
The larger attention length, with respect to 3D convolutions, ensures a very different behavior when the transformer architecture is combined with the stream buffer. In particular, we can observe a substantial difference in the receptive field of the network when combining convolutions with transformer layers. Even a single transformer layer can substantially increase the receptive field, as shown in Fig. 3.11.

3.4.2 Efficiency and Latency

Another objective is to obtain models which are lightweight such that they can be used to perform inference locally on mobile devices. For this reason, it has been chosen to use the MoViNets architectures as backbones for the proposed models, as they have demonstrated themselves to be very efficient. The proposed architecture is provided with two transformer layers which replace the GAP as discussed in chapter 3.4.1. The choice of the transformer architecture fell on the Transformer-XL, due to its properties, especially the faster inference time with respect to vanilla transformer. In fact, the Transformer-XL state reuse behaves similarly to the MoViNets Stream Buffer, meaning that it stores the activation so that it doesn't have to recompute them at every evaluation. This property of not having to recompute the activation is the key to the low latency of both the Transformer-XL and the MoViNets. The other property of the Transformer-XL is causality, necessary to work together with the streaming version of MoViNets.

3.4.3 Architecture Details

As already discussed, the new architecture will use MoViNets as backbones for a transformer classification head. The transformer architecture used is

a Transformer-XL with two layers. The "key" and "value" matrices of all attention mechanisms have an inner dimensionality of $d_{kv} = 64$ and all attention mechanisms have 6 heads. The model dimensions have a dimensionality of $d_{model} = 128$ displayed in Tab. 3.6, or $d_{model} = 192$ displayed in Tab. 3.7 or $d_{model} = 256$ displayed in Tab. 3.8, in order of increasing computational requirements. The following dense layer has dimensionality equal to four times d_{model}. These three classification heads created by changing the model dimensions are used to create three different versions for each MoViNet.
The proposed architecture with $d_{model} = 192$ and MoViNet-A0 as backbone is displayed in Table 3.9. The base backbone is a MoViNet-A0, while on top we have Transformer-XL layers followed by a classification fully connected layer. Comparing the model with the original MoViNet-A0 architecture in Table 3.2 with the transformer version in 3.9, we can see that the GAP layer is substituted with an average pooling layer with kernel of dimension $(1 \times 5 \times 5)$, where 1 is the temporal dimension, and 5 are the spatial dimensions. The temporal dimension of the activations is therefore kept the same, and the activations are fed to the transformer architecture as sequential data.

Summarizing, the new architecture has nice properties introduced by the stream buffer, such as:

- Possibility to learn long dependencies. The activation are saved and it's therefore possible to attend to frames that are very far in the past.
- Lightweight models and possibility of deployment on mobile devices for real-time application.

And properties introduced by the transformer architecture:

- Capturing long-term relationships between frames.
- Capable of attending to particular salient frames thanks to the attention mechanism.

In the following chapter we will discuss the benefits and the downsides of each of the architectures investigated, a comparison of the results obtained on the different architectures, and an analysis of the properties of the newly introduced models.

Stage	Operation	Output size
data	stride 5, RGB	50×172^2
conv_1	1×3^2, 8	50×86^2
block_2	[1×5^2, 8, 40]	50×43^2
block_3	$\begin{bmatrix} 5\times3^2, 32, 80 \\ 3\times3^2, 32, 80 \\ 3\times3^2, 32, 80 \end{bmatrix}$	50×21^2
block_4	$\begin{bmatrix} 5\times3^2, 56, 184 \\ 3\times3^2, 56, 112 \\ 3\times3^2, 56, 184 \end{bmatrix}$	50×10^2
block_5	$\begin{bmatrix} 5\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \end{bmatrix}$	50×10^2
block_6	$\begin{bmatrix} 5\times3^2, 104, 344 \\ 1\times5^2, 104, 280 \\ 1\times5^2, 104, 280 \\ 1\times5^2, 104, 344 \end{bmatrix}$	50×5^2
conv_7	1×1^2, 480	50×5^2
pool_8	50×5^2	1×1^2
dense_9	1×1^2, 2048	1×1^2
dense_{10}	1×1^2, 600	1×1^2

Table 3.2: **MoViNet-A0 Architecture.** Under *operation* for the different blocks we have, in order, the kernel size in the format $\text{k}_t\times\text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, the base channel and the expansion channel of the convolutional block. If only one value is provided after the kernel size, it is referred to the output channels. Output size is in the format $\text{T}\times\text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions.(from [22])

Stage	Operation	Output size
data	stride 5, RGB	50×172^2
conv_1	1×3^2, 16	50×86^2
block_2	$\begin{bmatrix} 1\times5^2, 16, 40 \\ 3\times3^2, 16, 40 \end{bmatrix}$	50×43^2
block_3	$\begin{bmatrix} 3\times3^2, 40, 96 \\ 3\times3^2, 40, 120 \\ 3\times3^2, 40, 96 \\ 3\times3^2, 40, 96 \end{bmatrix}$	50×21^2
block_4	$\begin{bmatrix} 5\times3^2, 64, 216 \\ 3\times3^2, 64, 128 \\ 3\times3^2, 64, 216 \\ 3\times3^2, 64, 168 \\ 3\times3^2, 64, 216 \end{bmatrix}$	50×10^2
block_5	$\begin{bmatrix} 5\times3^2, 64, 216 \\ 3\times3^2, 64, 216 \\ 3\times3^2, 64, 216 \\ 3\times3^2, 64, 128 \\ 1\times5^2, 64, 128 \\ 3\times3^2, 64, 216 \end{bmatrix}$	50×10^2
block_6	$\begin{bmatrix} 5\times3^2, 136, 456 \\ 1\times5^2, 136, 360 \\ 1\times5^2, 136, 360 \\ 1\times5^2, 136, 360 \\ 1\times5^2, 136, 456 \\ 3\times3^2, 136, 456 \\ 1\times3^2, 136, 544 \end{bmatrix}$	50×5^2
conv_7	1×1^2, 600	50×5^2
pool_8	50×5^2	1×1^2
dense_9	1×1^2, 2048	1×1^2
dense_{10}	1×1^2, 600	1×1^2

Table 3.3: **MoViNet-A1 Architecture.** Under *operation* for the different blocks we have, in order, the kernel size in the format $\text{k}_t \times \text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, the base channel and the expansion channel of the convolutional block. If only one value is provided after the kernel size, it is referred to the output channels. Output size is in the format $\text{T} \times \text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions. (from [22])

Stage	Operation	Output size
data	stride 5, RGB	50×224^2
conv_1	1×3^2, 16	50×112^2
block_2	$\begin{bmatrix} 1\times5^2, 16, 40 \\ 3\times3^2, 16, 40 \\ 3\times3^2, 16, 64 \end{bmatrix}$	50×56^2
block_3	$\begin{bmatrix} 3\times3^2, 40, 96 \\ 3\times3^2, 40, 120 \\ 3\times3^2, 40, 96 \\ 3\times3^2, 40, 96 \\ 3\times3^2, 40, 120 \end{bmatrix}$	50×28^2
block_4	$\begin{bmatrix} 5\times3^2, 72, 240 \\ 3\times3^2, 72, 160 \\ 3\times3^2, 72, 240 \\ 3\times3^2, 72, 192 \\ 3\times3^2, 72, 240 \end{bmatrix}$	50×14^2
block_5	$\begin{bmatrix} 5\times3^2, 72, 240 \\ 3\times3^2, 72, 240 \\ 3\times3^2, 72, 240 \\ 3\times3^2, 72, 240 \\ 1\times5^2, 72, 144 \\ 3\times3^2, 72, 240 \end{bmatrix}$	50×14^2
block_6	$\begin{bmatrix} 5\times3^2, 144, 480 \\ 1\times5^2, 144, 384 \\ 1\times5^2, 144, 384 \\ 1\times5^2, 144, 480 \\ 1\times5^2, 144, 480 \\ 3\times3^2, 144, 480 \\ 1\times3^2, 144, 576 \end{bmatrix}$	50×7^2
conv_7	1×1^2, 640	50×7^2
pool_8	50×7^2	1×1^2
dense_9	1×1^2, 2048	1×1^2
dense_{10}	1×1^2, 600	1×1^2

Table 3.4: **MoViNet-A2 Architecture.** Under *operation* for the different blocks we have, in order, the kernel size in the format $\text{k}_t \times \text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, the base channel and the expansion channel of the convolutional block. If only one value is provided after the kernel size, it is referred to the output channels. Output size is in the format $\text{T} \times \text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions. (from [22])

Model	Top-1	Res	Frames	FPS	gflops	Mem (MB)
MobileNetV3-L*	**68.1**	224	1×50	5	11.0	**23**
MoViNet-A0	**71.5**	172	1×50	5	2.71	173
MoViNet-A0-Stream	70.3	172	1×50	5	2.73	**71**
MoViNet-A1	**76.0**	172	1×50	5	6.02	191
MoViNet-A1-Stream	75.6	172	1×50	5	6.06	**72**
MoViNet-A1-Stream-Ens (x2)	75.9	172	1×25	2.5	6.06	**72**
MoViNet-A2	**77.5**	224	1×50	5	10.3	470
MoViNet-A2-Stream	76.5	224	1×50	5	10.4	**85**
MoViNet-A2-Stream-Ens (x2)	77.0	224	1×25	2.5	10.4	**85**
MoViNet-A3	80.8	256	1×120	12	56.9	1310
MoViNet-A3-Stream	79.6	256	1×120	12	57.1	**82**
MoViNet-A3-Stream-Ens (x2)	**80.4**	256	1×60	6	57.1	**82**
MoViNet-A4	81.2	290	1×80	8	105	1390
MoViNet-A4-Stream	80.5	290	1×80	8	106	**112**
MoViNet-A4-Stream-Ens (x2)	**81.4**	290	1×40	4	106	**112**
MoViNet-A5	82.7	320	1×120	12	281	2040
MoViNet-A5-Stream	82.0	320	1×120	12	282	**171**
MoViNet-A5-Stream-Ens (x2)	**82.9**	320	1×60	6	282	**171**
ResNet3D-50	**78.7**	224	1×250	25	390	3040
ResNet3D-50-Stream	76.9	224	1×250	25	390	**2600**
ResNet3D-50-Stream-Ens (x2)	78.6	224	1×125	12.5	390	**2600**

Table 3.5: **Base vs. Streaming Architectures** on Kinetics 600. We and report the inference resolution (res), number of clips × frames per clip (frames), and frame rate (fps) for each video. We measure the total GFLOPs per video across all frames. We denote "Stream" to be causal models using a stream buffer frame-by-frame, and "Ens" to be two ensembled models (with half the input frames so FLOPs are equivalent). Memory usage is measured in peak MB for a single video clip. * denotes reproduced models.

Stage	Operation	Output size
conv_7	1×1^2, 480	50×5^2
pool_8	1×5^2	50×1^2
$\text{transformerlayer}_9$	128	50×1^2
$\text{transformerlayer}_{10}$	128	1×1^2
dense_{12}	1×1^2, 512	1×1^2
dense_{13}	1×1^2, 600	1×1^2

Table 3.6: Classification head using the transformer architecture with $d_{model} = 128$ and A0-stream architecture. The operation indicates, in order, the kernel size in the format $\text{k}_t \times \text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, and the output channel. For the Transformer layers only the dimension of the model is displayed. Output size is in the format $\text{T} \times \text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions.

Stage	Operation	Output size
conv_7	1×1^2, 480	50×5^2
pool_8	1×5^2	50×1^2
$\text{transformerlayer}_9$	192	50×1^2
$\text{transformerlayer}_{10}$	192	1×1^2
dense_{12}	1×1^2, 768	1×1^2
dense_{13}	1×1^2, 600	1×1^2

Table 3.7: Classification head using the transformer architecture with $d_{model} = 192$ and A0-stream architecture. The operation indicates, in order, the kernel size in the format $\text{k}_t \times \text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, and the output channel. For the Transformer layers only the dimension of the model is displayed. Output size is in the format $\text{T} \times \text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions.

Stage	Operation	Output size
conv_7	1×1^2, 480	50×5^2
pool_8	1×5^2	50×1^2
$\text{transformerlayer}_9$	256	50×1^2
$\text{transformerlayer}_{10}$	256	1×1^2
dense_{12}	1×1^2, 1024	1×1^2
dense_{13}	1×1^2, 600	1×1^2

Table 3.8: Classification head using the transformer architecture with $d_{model} = 256$ and A0-stream architecture. The operation indicates, in order, the kernel size in the format $\text{k}_t \times \text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, and the output channel. For the Transformer layers only the dimension of the model is displayed. Output size is in the format $\text{T} \times \text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions.

Stage	Operation	Output size
data	stride 5, RGB	50×172^2
conv_1	1×3^2, 8	50×86^2
block_2	[1×5^2, 8, 40]	50×43^2
block_3	$\begin{bmatrix} 5\times3^2, 32, 80 \\ 3\times3^2, 32, 80 \\ 3\times3^2, 32, 80 \end{bmatrix}$	50×21^2
block_4	$\begin{bmatrix} 5\times3^2, 56, 184 \\ 3\times3^2, 56, 112 \\ 3\times3^2, 56, 184 \end{bmatrix}$	50×10^2
block_5	$\begin{bmatrix} 5\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \\ 3\times3^2, 56, 184 \end{bmatrix}$	50×10^2
block_6	$\begin{bmatrix} 5\times3^2, 104, 344 \\ 1\times5^2, 104, 280 \\ 1\times5^2, 104, 280 \\ 1\times5^2, 104, 344 \end{bmatrix}$	50×5^2
conv_7	1×1^2, 480	50×5^2
pool_8	1×5^2	50×1^2
$\text{transformerlayer}_9$	192	50×1^2
$\text{transformerlayer}_{10}$	192	1×1^2
dense_{12}	1×1^2, 768	1×1^2
dense_{13}	1×1^2, 600	1×1^2

Table 3.9: **MoViNet-A0-T192 Architecture.** Under *operation* for the different blocks we have, in order, the kernel size in the format $\text{k}_t\times\text{k}_s^2$, where t indicates the temporal dimension and s indicates the spatial dimensions, the base channel and the expansion channel of the convolutional block. If only one value is provided after the kernel size, it is referred to the output channels. For the Transformer layers only the dimension of the model is displayed. Output size is in the format $\text{T}\times\text{S}^2$, where T indicates the temporal dimension and S indicates the spatial dimensions.

Chapter 4

Experiments

The experiments in this study focus on three datasets: HMBD51 [24], UCF101 [41] and Something-to-Something [12]. The first two datasets have fewer clips, with $6,766$ clips and $13,320$ clips, respectively, while the third has a larger number of clips corresponding to $108,499$. The first two datasets have three different splits for evaluation, while the last one follows the standard division into training, validation and test set. Moreover, the first two can be considered as Coarse Action Recognition datasets, while the last one is a Fine-Grained Action Recognition dataset. The results are calculated on the three splits of HMDB51 and UCF101 as suggested by the respective papers. The results on Something-to-Something are performed on the validation dataset provided by the authors. Moreover, it is important to consider that the MoViNets backbones have been pre-trained on the Kinetics-600 [3] dataset.
In the following sections we will discuss the benefits and the downsides of each of the architectures investigated, a comparison of the results obtained on the different architectures, and an analysis of the properties of the newly introduced models.

4.1 Implementation Details

The experiments compare the results of the original MoViNets [22] models, both the streaming and non-streaming versions, with the newly proposed networks. The streaming versions are named A0-stream, A1-stream and A2-stream, while the non-streaming versions are indicated with A0, A1 and

A2. The architectures go from A0 to A2, representing increasingly larger models. The models chosen are the smallest of the MoViNets family, which are more suited for the training hardware at our disposal. Furthermore, the A2 non-streaming model hasn't been trained due to the impossibility of the model to fit the GPU memory. The comparison covers the different properties of the network such as GFLOPS, number of parameters, peak memory occupancy and Latency. For the scope of this work, the MoViNets models have been reimplemented in PyTorch, following the code released by the authors in TensorFlow [22], and using the same weights released by the authors in their original implementation. The results obtained by the reimplemented architectures are displayed with an asterisk *, e.g. A0-stream* indicates the reimplemented A0-stream model.
The architectures presented are characterized by different latency, number of parameters, GFLOPS, and peak memory occupancy. The properties are displayed in Tables 4.1, Table 4.3 and Table 4.2, for the models having 60, 30 and 20 frames in input, respectively.
The streaming versions of the models are indicated in what follows with a *-s-* or *-stream*, while the transformer versions are indicated with T. The models with transformer are also provided with a number indicating the dimension of the model. As an example, the transformer version having a $d_{model} = 128$ with a A0 backbone is indicated with A0T-s-128 or A0T-stream-128.

4.1.1 Architectures

As discussed in section 3.4, the modified architectures are composed by a MoViNet backbone and a Transformer-XL [6] classification head. The transformer head is characterized by 6 heads of dimension 64 each for a total dim of 384. The expansion layer of the feed forward network of the transformer block is set to two times the transformer inner dimension d_{model}, where d_{model} corresponds to the dimension in input and in output to both the self-attention and feed-forward layers. The classification feed-forward network instead is composed by two layers with the expansion layers dimension set to four times d_{model}. The transformer head is composed by two transformer layers.
The new models have been chosen to be lightweight and to have a low enough latency to allow them to be used for online inference. The model dimension d_{model} has been modified in order to obtain three different models with similar GFLOPS with respect to the original streaming version. The proposed transformer heads have different numbers of parameters due to the different

d_{model} dimension, on the other hand, the number of GFLOPS is very close to the original network.
The proposed architectures are three, with 128, 192 and 256 model dimension d_{model}. The attention length and the number of frames in input are different for each dataset, as one model may not gain in accuracy by an increase in the number of frames or attention length. This is especially true for HMDB51 and UCF101 datasets, which are considered Coarse-Action Recognition Datasets.
The proposed architectures are very fast and can be realistically used in real-time applications.

4.1.2 Training and Evalution

The models are fine-tuned with cosine learning rate with a warm-up period equal to 5 epochs of training, except for SSv1 which is trained with a warm-up period of 3 epochs. The optimizer of choice is Adam with learning rate (LR) equal to $1e-4$ and weight decay 0.0001 for all the models, except for SSv1, where the LR is equal to $7e-5$. Augmentation consisting of random resized crops and RandAugment [5] has been applied. The models are trained with batch size 16 and a frame rate of 5 FPS for HDB51 and UCF101, while for SSv1 a frame rate of 12 FPS is used. The number of frames in input at training time is kept constant also at evaluation time. In order to reduce the memory requirements, models are trained using FP16. MoViNet-A1 has been trained with batch size 15 in order to fit the GPU memory. On UCF101 and HMDB51, in order to reduce memory consumption during training, the models with stream buffer are trained with $T_{clip} = 10$ and back propagation is not performed through the stream buffer. On SSv1 due to the higher number of frames in input the models it has been chosen a $T_{clip} = 20$ for the A0 based models, for the A2 training we have instead $T_{clip} = 10$ due to the impossibility of fitting the model on the GPU memory when provided with a $T_{clip} = 20$.
The evaluation is always performed only using one clip of video, instead of using multiple clips of the same video. During evaluation, a resize to $1.2 \times d_{input}$ where d_{input} represent the dimension of the height and width of the input of the model, and then a center crop is performed to obtain the dimension desired dimension d_{input}. If not otherwise specified the d_{input} of the models is euqal to 174 for the A0 and A1, and 224 for A2. Along the temporal dimension we select the first frames of the video for HMDB51 and

UCF101 and the center frames of the video for SSv1.

4.1.3 Latency Testing

MoViNets A0, A1 and A2 would be the models that can realistically run on a mobile device. In particular, the streaming version has been realized with the intent of a possible deployment for real-time applications. This is mainly due to the lower latency that can be achieved thanks to the stream buffer. To understand the improvements in latency, the latency has been tested for both the streaming versions and the non-streaming versions of MoViNets in single frame evaluation. The tests include also the newly introduced models in order to observe whether the transformer architecture introduces an increase in latency.
The tests have been done using a randomly generated input of the same dimension of the input videos. 10 runs are completed to warm up the CPU, then 60 evaluations are performed, and the average latency is computed. The input of the non-streaming versions has dimension $(1 \times 3 \times t \times h \times w)$, while the input of the streaming version has dimension $(1 \times 3 \times 1 \times h \times w)$. The dimensions correspond, respectively, to the batch dimension, to the number of channels of the RGB video, to the time dimension and to the two spatial dimensions. The time dimension is kept equal to 1 for the streaming dimension, while it's equal to the total length of the video for the non-streaming version. As discussed in section 3.3.2 the streaming versions allow for a frame by frame inference ($T^{clips} = 1$) which improves the inference time, making these networks suitable for online inference.
The overall latency is shown are shown in Tables 4.2, 4.3 and 4.1, for 20, 30 and 60 frames in input, respectively. The results in the tables highlight a consistent increase in latency for the non-streaming models as the number of frames in input increases. Considering A0 the models experiences a latency of 81.57 ms, 118.41 ms and 270.46 ms considering 20, 30 and 60 frames in input, respectively. This is due to the fact that in order to compute the label for a given timestep it is necessary to provide the whole clip to the model. The streaming versions instead don't experience increase in latency thanks to the presence of the stream buffer that allows to store the past activation, instead of recomputing them. The increase in attention length also doesn't seem to provide a relevant difference in latency.
The tests have been performed on an x86 processor i9-9940x.

4.1.4 FLOPS estimation and parameters count

In order to obtain the FLOPS estimation and the parameters count of the model, a software developed by Facebook has been used [9]. In this case the input dimensions are constant for both the streaming and non-streaming version of the models, and they correspond to the entire input video. It is therefore measured the FLOPS necessary to compute the label of a single video.
The models are created in order to have comparable GLOPS between versions of the same backbone.

4.1.5 Peak Memory Occupancy

Peak memory occupancy is defined as the maximum amount of memory occupied by the network in inference or training. In particular, in this chapter, peak memory occupancy will refer to the maximum amount of memory occupied by the network during the inference of a single clip.
The non-streaming models results are reported by providing the whole clip to the model, while the streaming versions results are obtained by providing the clip frame by frame, as possible thanks to the presence of the stream buffer.
Due to the fact that non-streaming architectures take as input the whole clip, we can observe an increase in memory occupancy with the increase of the number of frames in input, considering the same architecture. The streaming version of the architectures instead show a peak memory occupancy that doesn't increase significantly with the increase in the number of total frames of the clip. Overall, the architectures with the transformer head have a higher peak memory occupancy with respect to the streaming versions, which increases with the increase of model dimension d_{model}. The peak memory occupancy of the transformer versions does not change significantly with the change in the number of frames in input. The results on peak memory occupancy can be found in Table 4.2 with 20 frames in input, Table 4.3 with 30 frames in input, Table 4.1 with 60 frames in input.

4.2 Experiment results

The first experiments have been done on the HMDB51 and UCF101 dataset. The HMDB51 dataset, with only $6,766$ clips, and the UCF101, with $13,320$

clips, are now considered very small datasets. The results of the new architectures on small datasets may fail to provide information about the performances of the architecture when scaling up to larger datasets. On the other hand, it may result time-consuming to test multiple architectures with very large dataset, if not provided with sufficient computational resources.
Something to Something V1, with 108,499 clips, is the biggest dataset used during the experiments, and, due to computational resources, only two architectures have been trained on this dataset. The chosen architectures are the A2T-stream-256 and the A2-stream in order to have a comparison for the performances. The architectures have been chosen based on the results on the smaller datasets and on the computational requirements and capabilities.

Model	Param	Att Len	Frames	GFLOPS	Lat(ms)	Mem(MB)
A0*	2.24M	-	1×60	3.56	270.46	107.49
A0-s*	2.88M	-	1×60	3.40	18.01	20.75
A0T-s-128	2.34M	60	1×60	3.44	19.13	30.78
A0T-s-192	2.89M	60	1×60	3.46	19.21	41.11
A0T-s-256	3.54M	60	1×60	3.49	19.36	52.64
A1*	3.82M	-	1×60	7.79	486.77	171.10
A1-s*	5.57M	-	1×60	7.56	29.55	38.87
A1T-s-128	4.81M	60	1×60	7.60	30.34	47.11
A1T-s-192	5.37M	60	1×60	7.62	30.36	57.48
A1T-s-256	6.03M	60	1×60	7.65	30.61	69.06
A2-s*	6.45M	-	1×60	14.60	36.36	49.94
A2T-s-128	5.61M	60	1×60	14.65	37.54	57.83
A2T-s-192	6.17M	60	1×60	14.67	37.83	67.69
A2T-s-256	6.84M	60	1×60	14.7	38.17	79.55

Table 4.1: Model architectures specifications used for the experiments with Something to Something V1. *Params* indicates the number of parameter of the model, *Att Len* indicates the length of the attention for the models that use a transformer architecture, *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS necessary for a single clip evaluation, *Lat* indicates the latency experienced in producing the label of the next frame in the video and *Mem* represent peak memory occupancy in evaluating a single clip.

4.2.1 HMDB51 Results

Model	Params	Att Len	Frames	GFLOPS	Lat(ms)	Mem(MB)
A0*	1.99	-	1×20	1.19	81.57	40.67
A0-s*	2.62	-	1×20	1.13	18.08	19.79
A0T-s-128	2.278	20	1×20	1.15	18.84	30.42
A0T-s-192	2.80	20	1×20	1.15	18.81	40.57
A0T-s-256	3.42	20	1×20	1.16	19.19	51.92
A1*	3.56	-	1×20	2.60	136.91	67.15
A1-s*	5.32	-	1×20	2.52	29.15	37.92
A1T-s-128	4.75	20	1×20	2.53	30.61	46.75
A1T-s-192	5.27	20	1×20	2.54	30.16	56.94
A1T-s-256	5.9	20	1×20	2.55	30.24	68.34
A2-s*	6.20	-	1×20	4.87	36.97	48.98
A2T-s-128	5.55	20	1×20	4.88	37.28	57.47
A2T-s-192	6.08	20	1×20	4.89	37.48	67.42
A2T-s-256	6.71	20	1×20	4.90	37.41	78.83

Table 4.2: Model architectures specifications used for the experiments with HMDB51. *Params* indicates the number of parameter of the model, *Att Len* indicates the length of the attention for the models that use a transformer architecture, *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS necessary for a single clip evaluation, *Lat* indicates the latency experienced in producing the label of the next frame in the video and *Mem* represent peak memory occupancy in evaluating a single clip.

As already pointed out, the HMDB51 dataset is a dataset of coarse action recognition. The model has been trained with 20 frames, due to the fact that the model accuracy either didn't improve or got worse with an increase in the number of frames in input. This can be explained by the fact that, being a dataset for Coarse Action Recognition, the actions can be easily captured by few frames of the videos.
The experiments performed provide a comparison between the accuracy of the different architecture. From Table 4.5 it's possible to observe that the streaming versions have a comparable accuracy with the non-streaming versions. Furthermore, the models with transformer provide comparable accu-

racy with respect to both the reimplemented streaming and non-streaming MoViNets.
Considering the A0 based models, A0-stream performs slightly better than the best performing transformer model with 69.74% accuracy and 69.41%, respectively. Considering instead the A1 based models, A1-stream performs slightly better than the best performing transformer model with 73.32% accuracy and 73.26%, respectively. Lastly, the best performing transformer A2 model, A2T-stream-192, performs slightly better than A2-stream with 74.62% accuracy and 74.56%, respectively.
The transformer versions have proved to be a valid alternative to standard MoViNets. In particular, the version with $d_{model} = 128$ has a considerable lower number of parameters when compared to the version without a transformer, while having a comparable accuracy, making it a suitable option in case of a memory constrain.

4.2.2 UCF101 Results

The UCF101 dataset is also a dataset of coarse action recognition. The number of frames in input is set to 30 because providing more frames either didn't improve or got the accuracy worse. This can be explained by the fact that being a dataset for Coarse Action Recognition, the action can be easily captured by few frames of video. We can also expect that UCF101 requires more information in the temporal dimension with respect to HMDB51 due to the fact that HMDB51 performances caps out at a number even lower of frames.
On UCF101, the architectures with transformers perform better than the original streaming versions. In particular, the models with $d_{model} = 256$ outperform all the corresponding reimplemented streaming version of the MoViNets, achieving the best results for A1 and A2 with a slight increase of 0.21% and 0.36% with respect to A1-stream and A2-stream, respectively. The A0T-stream-192 version with Transformer dimension $d_{model} = 192$ is the best performing A0 streaming model, with an increase of 0.74% with respect to A0-stream. The A0T-stream-192 slightly outperforms also the non-streaming version of A0. As discussed for the HMDB51 dataset, the models with transformer dimension $d_{model} = 128$ remain of particular interest, thanks to the lower number of parameters and comparable accuracy. The models with $d_{model} = 256$ are also interesting thanks to the consistent increase in accuracy while maintaining the GFLOPS of the model constant. For

detailed information about the results, see Table. 4.4.

Model	Params	Att Len	Frames	GFLOPS	Lat(ms)	Mem(MB)
A0*	2.09M	-	1×30	1.78	119.41	57.52
A0-s*	2.73M	-	1×30	1.70	18.08	20.19
A0T-s-128	2.3M	30	1×30	1.72	19.18	30.57
A0T-s-192	2.8M	30	1×30	1.73	19.08	40.79
A0T-s-256	3.47M	30	1×30	1.74	19.12	52.21
A1*	3.67M	-	1×30	3.90	227.72	93.60
A1-s*	5.43M	-	1×30	3.78	29.28	38.31
A1T-s-128	4.77M	30	1×30	3.80	30.26	46.90
A1T-s-192	5.31M	30	1×30	3.81	30.70	57.16
A1T-s-256	5.95M	30	1×30	3.82	30.49	68.63
A2-s*	6.30M	-	1×30	7.30	36.79	49.37
A2T-s-128	5.57M	30	1×30	7.32	38.16	57.62
A2T-s-192	6.12M	30	1×30	7.33	37.84	67.64
A2T-s-256	6.76M	30	1×30	7.35	37.89	79.13

Table 4.3: Model architectures specifications used for the experiments with UCF101. *Params* indicates the number of parameter of the model, *Att Len* indicates the length of the attention for the models that use a transformer architecture, *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS necessary for a single clip evaluation, *Lat* indicates the latency experienced in producing the label of the next frame in the video and *Mem* represent peak memory occupancy in evaluating a single clip.

4.2.3 Something to Something V1 Results

The Something to Something dataset is a considerably larger dataset than HMDB51 and UCF101 and, as explained in Section 1.4.3, can be considered a Fine-Grained action recognition dataset. A first experiments has been performed with A0-stream and A0T-stream-256, with an input size of $(112, 112)$ and $T^{clips} = 20$. A0-stream has been chosen to provide a comparison to the performances of the model, as it has comparable parameters and GFLOPS with respect to A0T-stream-256. The newly proposed model is capable of outperforming the standard MoViNets by a 2.91%, with 45.27% and 42.36%

accuracy for A0T-stream-256 and A0-stream, respectively. To the best of our knowledge this is a new State-of-the-Art result for models with comparable GFLOPS, providing strong evidence of the efficancy of the transformer architecture to model the temporal dimension in the field of RGB Human Action Recognition. Considering the larger dataset size, it seemed appropriate to test also on A2 for the experiment, being the larger architecture tested in this work. Also, to avoid bottlenecking the architecture, the model with dimension $d_{model} = 256$ has been chosen for the experiments. Due to the high memory resources necessary to perform training, the models have been trained with $T^{clips} = 10$, in order to fit the GPU memory. A2-stream has been chosen to provide a comparison to the performances of the model, as it has comparable parameters and GFLOPS with respect to A2T-stream-256. The experiments report that the original version of the MoViNets outperforms the Transformer version with 49.11% and 47.70% accuracy, respectively. This result is in contrast with the previously results, and may be due to the higher T^{clips} used in A0, which allows to backpropagate further in the temporal dimension, or it may be a result of an not sufficiently large Transformer head for the A2 architecture. Further experiments are necessary to better understand the reasons behind this outcome.
The results are reported in Table 4.6.

4.2.4 State-of-the-Art Comparison

In this section, the results of the proposed architectures are compared with State-of-the-Art architectures. The models selected for this comparison are models which are provided with information regarding the GLOPS, and are comparable with the architectures studied in this work, in terms of GFLOPS and parameters. It is relevant to report that only few architectures have a comparable number of GLOPS due to the high efficiency of MoViNets.
3D-MobileNetV2 1.0x [23], where 1.0x indicates the width multiplier of the network, is a conversion to 3D CNN of the efficient 2D MobileNetV2 architecture. Comparing the results on UCF101, Table 4.4, 3D-MobileNetV2 1.0x fails to achieve comparable accuracy, reporting $13,67\%$ lower accuracy with respect to A1T-stream-256, with similar GFLOPS. Ligar [18], is a more recent approach that proposed a new architecture, which is capable of providing more competitive results. The network can be compared with A1T-stream-256, having 4.47 and 3.82 GLOPS, respectively. Ligar reaches 93.63% accuracy, which represent a 1.64% drop in accuracy with respect to A1T-stream-

256. STM [19], instead, presents similar accuracy to A2T-stream-256, while requiring 3× more GFLOPS. R(2+1)D-BERT [20] is reported due to the fact that currently holds State-of-the-Art accuracy in UCF101. The model provides a $2,5\%$ increase in accuracy with respect to A2T-stream-256, while requiring more than 20× the GFLOPS. The real number of GFLOPS used for R(2+1)D-BERT is unknown due to the fact that they didn't report the number of clips used in evaluation.
Comparing the results on HMDB51, Table 4.5, STM [19], reports 72.20% accuracy, which is lower than the 73.26% accuracy reported by A1T-stream-256 while requiring more than 10× the GLOPS. MSNet-R50 [25] reports 1.18% increase in accuracy over A2T-stream-192, while requiring almost 70× more GFLOPS.
Comparing the results on SSv1, Table 4.6, A2-stream and A2T-stream-256 perform 1.79% and 3.11% worse in accuracy than MSNet-R50, respectively, but they require less than half the GLOPS. The A0T-stream-256 performs with similar accuracy to TSM (R50), while having more than 20× the GFLOPS. To the best of my knowledge no model with comparable GFLOPS to A0-stream, A0T-stream-256 , A2-stream and A2T-stream-256 has been trained on SSv1. The MoViNets and the transformer versions perform better than all the architectures reported with comparable GFLOPS. In particular, the proposed A0T-stream-256 performs 2.91% better than A0-stream, reaching a new State-of-the-Art result.

4.2.5 Generalization to Longer Sequences

The architectures can take as input a larger number of frames, with respect to the ones used in training. A study of the generalization abilities has been done on the A2-stream-T256 and A2-stream trained on SSv1. In this case the number of frames used in training corresponds to 60 and the dataset used is SSv1. From Table 4.7, we can observe that both the Transformer and the standard architecture are capable of generalizing to longer sequences, with a consistent performance improvement even when provided with 1.5× the frames used during training, corresponding to 0.28% and 0.59% increase in accuracy for the Transformer and standard architecture, respectively. When evaluating with 3× the frames used during training, we observe a +0.66% and +0.43% in accuracy for the Transformer and standard architecture, respectively, with respect to the evaluation performed using the same number of frames used in training. The results are obtained with an attention length

Name	Frames	GFLOPS	Params	Accuracy
A0*	1×30	1.78	2.09M	93.89
A0-stream*	1×30	1.70	2.73M	93.37
A0T-stream-128	1×30	1.72	2.3M	93.70
A0T-stream-192	1×30	1.73	2.8M	**94.11**
A0T-stream-256	1×30	1.74	3.47M	93.77
A1*	1×30	3.90	3.67M	**95.59**
A1-stream*	1×30	3.78	5.43M	95.06
A1T-stream-128	1×30	3.80	4.77M	94.87
A1T-stream-192	1×30	3.81	5.31M	94.85
A1T-stream-256	1×30	3.82	5.95M	**95.27**
3D-MobileNetV2 1.0x [23]	10×16	10×0.45	3.12M	81.60
LIGAR [18]	1×16	4.74	4.47M	93.63
A2-stream*	1×30	7.30	6.30M	95.83
A2T-stream-128	1×30	7.32	5.57M	96.00
A2T-stream-192	1×30	7.33	6.12M	96.07
A2T-stream-256	1×30	7.35	6.76M	**96.19**
STM [19]	1×8	33.3	24.0M	96.2
LIGAR [18]	10×16	10×4.74	4.47M	94.85
R(2+1)D-BERT [20]	NA×64	NA×152.97	66.67M	**98.69**

Table 4.4: State-of-the-Art performance comparison on UCF101. *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS used in evaluation,*Params* indicates the number of parameter of the model. In bold we can find models with the best accuracy, compared with models having similar GFLOPS. When the non streaming version performs better than the best streaming version, both the best performing streaming version and the non-streaming version are displayed in bold.

Name	Frames	GFLOPS	Param	Accuracy
A0*	1×20	1.19	1.99M	**69.87**
A0-stream*	1×20	1.13	2.62M	**69.74**
A0T-stream-128	1×20	1.15	2.28M	69.08
A0T-stream-192	1×20	1.15	2.80M	68.41
A0T-stream-256	1×20	1.16	3.42M	68.78
A1*	1×20	2.60	3.56M	**74.01**
A1-stream*	1×20	2.52	5.32M	**73.32**
A1T-stream-128	1×20	2.53	4.75M	72.87
A1T-stream-192	1×20	2.54	5.27M	72.45
A1T-stream-256	1×20	2.55	5.95M	73.26
A2-stream*	1×20	4.87	6.20M	74.58
A2T-stream-128	1×20	4.88	5.55M	74.16
A2T-stream-192	1×20	4.89	6.08M	**74.62**
A2T-stream-256	1×20	4.90	6.71M	74.05
STM [19]	1×8	33.3	24.0M	72.2
TSM† [25]	10×8	10×33	24.3M	71.9
MSNet-R50 [25]	10×8	10×34	24.5M	**75.8**

Table 4.5: State-of-the-Art performance comparison on HMDB51. † Represents a reimplemented model. *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS used in evaluation,*Params* indicates the number of parameter of the model. In bold we can find models with the best accuracy, compared with models having similar GFLOPS. When the non streaming version performs better than the best streaming version, both the best performing streaming version and the non-streaming version are displayed in bold.

Name	Frames	GFLOPS	Param	Accuracy
A0-stream*	1×60	1.443	2.875M	42.36
A0T-stream-256	1×60	1.534	3.414M	**45.27**
A2-stream*	1×60	14.60	6.45M	**49.11**
A2T-stream-256	1×60	14.70	6.84M	47.70
TRN [50]	NA×8	NA×16	18.3M	34.4
TSM (R50) [28]	1×8	33	24.3M	45.6
MSNet-R50 [25]	1×8	34	24.5M	50.9
X3D-M† (non-pretrained) [17]	6×16	6×6.4	3.3M	46.7
BQN [17]	6×48	6×9.7	6.6M	**53.7**

Table 4.6: State-of-the-Art performance comparison on Something-to-Something V1. † Represents a reimplemented model. *Frames* indicates the number of frames in input in the format *n-clips* × *n-frames-per-clip*, *GFLOPS* indicates the GFLOPS used in evaluation, *Params* indicates the number of parameter of the model. In bold we can find models with the best accuracy, compared with models having similar GFLOPS.

equal to the input length for the Transformer version.
It's also possible for the Transformer to extend or reduce the attention length, with respect to the one used in training. The experiments in Table 4.8 show that the network benefits from having an attention length equal to the amount of frames in input, even when larger than the attention length used in training, highlighting the ability of the Transformer attention mechanism to successfully generalize to longer sequences. A strong reduction in performance is instead observable when the attention length is set to 50% the number of frames in input, especially when provided with a low number of frames in input.

4.2.6 Clip centering

It is of particular interest the study of centering of the evaluation clip, since the models are provided with a single clip of the video. The choice of the centering of the clip can strongly influence performance, depending on whether the videos contain more information at the beginning, at the center or at the end of the videos. A study of this kind has been done on the A2-stream-T256 and A2-stream trained on SSv1. From Fig. 4.1, we can observe the models tend to perform better with center subclips when provided with few

Name	Frames Train	Frames Eval	Accuracy
A2-stream*	1×60	1×60	49.11
A2-stream*	1×60	1×90	**49.70**
A2-stream*	1×60	1×180	49.54
A2T-stream-256	1×60	1×60	47.70
A2T-stream-256	1×60	1×90	47.98
A2T-stream-256	1×60	1×180	**48.36**

Table 4.7: Capability of generalizing to longer sequences on SSv1. The models provide better accuracy when they receive in input a number of frames longer than the one they are trained on. The best performing versions for the A2-stream and A2T-stream-256 are displayed in bold.

Name	Frames Train	Frames Eval	Att Len	Accuracy
A2T-stream-256	1×60	1×180	1×90	48.14
A2T-stream-256	1×60	1×180	1×135	48.16
A2T-stream-256	1×60	1×180	1×180	**48.36**
A2T-stream-256	1×60	1×60	1×30	45.44
A2T-stream-256	1×60	1×60	1×45	47.50
A2T-stream-256	1×60	1×60	1×60	**47.70**

Table 4.8: Capability of generalizing to longer sequences on SSv1. The models provide better accuracy when provided with an attention length longer than the one they are trained on. Reducing the attention length provides a consistent drop in accuracy. The best performing versions are displayed in bold.

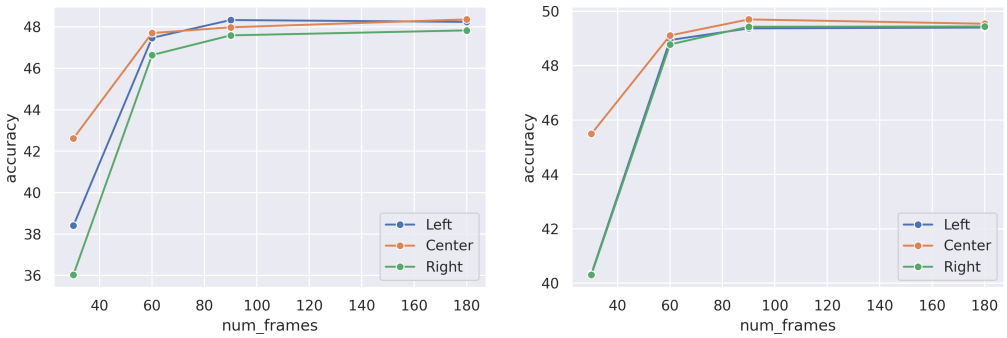


Figure 4.1: Change in accuracy with respect to the centering of choice. On the left the A2-stream-T256 model with attention length equal to the number of frames in input. On the right the A2-stream network. The chart represent the accuracy of the models when provided with the left, center, right frames of a clip, with respect to the number of frames in input.

frames, probably due to the more information contained at the center of the video. Once the number of frames in input gets close to the total number of frames of the video (most of the videos are contained within 72 frames), the differences between left, center, right becomes less important. Considering the results discussed in this section, as already anticipated, the accuracy on SSv1 are reported using centered clips.

Conclusions

In this thesis, the topic of Human Action Recognition has been investigated with the objective of proposing an architecture which is capable of modeling Long-Term relationships between frames and that is, at the same time, very efficient in order to be used for real time application on mobile devices. The overall architecture is composed of a MoViNet backbone combined with a Transformer-XL classification head. In order to test the capabilities of the newly proposed networks, the models have been trained on three datasets: HMDB51, UCF101 and Something-to-Something V1.
The results show that the Transformer architectures have comparable accuracy, for the dataset of HMDB51 and UCF101. On Something-to-Something V1, the largest and most challenging dataset for the temporal dimension, the experiments report that the Transformer version A0-stream-256 outperforms the original A0-stream improving the SOTA by 2.91% in accuracy, when compared with models with similar GFLOPS. A slight decrease in performance is instead observed for A2T-stream-256 when compared with A2-stream on SSv1. This may be due to different reason and more investigation has to be done to understand the applicability of the Transformer architecture to larger models.
Both the Transformer and the original architecture are also capable of generalizing to longer input sequences, with a performance improvement even when provided with 3× the frames used in training, +0.66% and +0.43% for the Transformer and original architecture, respectively.
Overall, the Transformer architecture is a very general architecture, reason for which it's important to keep working on the integration of the Transformer architecture to extend the current capabilities of the Action Recognition models. The integration of the Transformer architecture in the field of efficient action recognition has brought State-of-the-Art results, but further research is needed in order to explore the applicability to larger models.

Bibliography

[1] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *CoRR*, abs/1808.04444, 2018.

[2] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lucic, and Cordelia Schmid. Vivit: A video vision transformer. *CoRR*, abs/2103.15691, 2021.

[3] João Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. A short note about kinetics-600. *CoRR*, abs/1808.01340, 2018.

[4] Tanzeem Choudhury, Sunny Consolvo, Beverly Harrison, Jeffrey Hightower, Anthony LaMarca, Louis LeGrand, Ali Rahimi, Adam Rea, G. Bordello, Bruce Hemingway, Predrag Klasnja, Karl Koscher, James Landay, Jonathan Lester, Danny Wyatt, and Dirk Haehnel. The mobile sensing platform: An embedded activity recognition system. *Pervasive Computing, IEEE*, 7:32–41, 05 2008.

[5] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019.

[6] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*.

[7] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[9] Facebookresearch. $fvcore/flop_count.md$ at main · facebookresearch/fvcore, Apr 2021.

[10] Christoph Feichtenhofer. X3D: expanding architectures for efficient video recognition. *CoRR*, abs/2004.04730, 2020.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fründ, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and Roland Memisevic. The "something something" video database for learning and evaluating visual common sense. *CoRR*, abs/1706.04261, 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[15] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[16] Weiming Hu, Dan Xie, Zhouyu Fu, Wenrong Zeng, and Steve Maybank. Semantic-based surveillance video retrieval. *IEEE Transactions on Image Processing*, 16(4):1168–1181, 2007.

[17] Guoxi Huang and Adrian G. Bors. Video classification with finecoarse networks. *CoRR*, abs/2103.15584, 2021.

[18] Evgeny Izutov. Ligar: Lightweight general-purpose action recognition, 2021.

[19] Boyuan Jiang, Mengmeng Wang, Weihao Gan, Wei Wu, and Junjie Yan. STM: spatiotemporal and motion encoding for action recognition. *CoRR*, abs/1908.02486, 2019.

[20] M. Esat Kalfaoglu, Sinan Kalkan, and A. Aydin Alatan. Late temporal modeling in 3d CNN architectures with BERT for action recognition. *CoRR*, abs/2008.01232, 2020.

[21] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[22] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition, 2021.

[23] Okan Köpüklü, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. Resource efficient 3d convolutional neural networks. *CoRR*, abs/1904.02422, 2019.

[24] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[25] Heeseung Kwon, Manjin Kim, Suha Kwak, and Minsu Cho. Motionsqueeze: Neural motion feature learning for video understanding. *CoRR*, abs/2007.09933, 2020.

[26] T. Lane. Multi-channel convolutions explained with... ms excel! `https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108`, November 2018.

[27] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[28] Ji Lin, Chuang Gan, and Song Han. Temporal shift module for efficient video understanding. *CoRR*, abs/1811.08383, 2018.

[29] Weiyao Lin, Ming-Ting Sun, Radha Poovandran, and Zhengyou Zhang. Human activity recognition for video surveillance. In *2008 IEEE International Symposium on Circuits and Systems*, pages 2737–2740, 2008.

[30] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016.

[31] Sparsh Mittal and Vibhu . A survey of accelerator architectures for 3d convolution neural networks. *Journal of Systems Architecture*, 115, 01 2021.

[32] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[33] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Aug 2015.

[34] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[35] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2021.

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[37] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael Littman. Activity recognition from accelerometer data. volume 3, pages 1541–1546, 01 2005.

[38] Carl Rebman, Milam Aiken, and Casey Cegielski. Speech recognition in the human–computer interface. *Information & Management*, 40:509–519, 07 2003.

[39] I. Rodomagoulakis, N. Kardaris, V. Pitsikalis, E. Mavroudi, A. Katsamanis, A. Tsiami, and P. Maragos. Multimodal human action recognition in assistive human-robot interaction. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2702–2706, 2016.

[40] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[41] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[42] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021.

[43] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CoRR*, abs/1711.11248, 2017.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[45] Patrick von Platen. Transformer-based encoder-decoder models, Oct 2020.

[46] Chi-Feng Wang. A basic introduction to separable convolutions, Aug 2018.

[47] Wikipedia. Perplexity — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Perplexity&oldid=1022965742`, 2021. [Online; accessed 02-October-2021].

[48] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krähenbühl, and Ross B. Girshick. Long-term feature banks for detailed video understanding. *CoRR*, abs/1812.05038, 2018.

[49] Chuhan Zhang, Ankush Gupta, and Andrew Zisserman. Temporal query networks for fine-grained video understanding. *CoRR*, abs/2104.09496, 2021.

[50] Bolei Zhou, Alex Andonian, and Antonio Torralba. Temporal relational reasoning in videos. *CoRR*, abs/1711.08496, 2017.

[51] Yi Zhu, Xinyu Li, Chunhui Liu, Mohammadreza Zolfaghari, Yuanjun Xiong, Chongruo Wu, Zhi Zhang, Joseph Tighe, R. Manmatha, and Mu Li. A comprehensive study of deep video action recognition. *CoRR*, abs/2012.06567, 2020.