University of Padua

Department of Mathematics "Tullio Levi-Civita"

Action Recognition in Low-Resolution Videos

Master Thesis in Computer Science

Supervisor
Lamberto Ballan

Master Candidate
Alex Dametto

Co-supervisors
Guglielmo Camporese
Elena Izzo

Academic Year 2021-2022

Abstract

The current state of the art on recognizing actions as a computer vision problem focuses primarily on high quality video where the action is clearly visible. The models that are currently available are therefore not designed for low resolution inputs and their performance is not satisfactory in the presence of constraints such as resolution or duration of the video. This type of environment is very common in video surveillance, where we have low-resolution video captured with many of these constraints. In this work we are going to propose three different *Multi-Scale* architectures that try to adapt to these constraints and we will also propose some tricks that used in the training phase can significantly improve the performance of the models facing low resolution video. One of the proposed models, the *FPN AD-ResNet50*, manages to improve the *ResNet18* baseline scores with an improvement of +9.2% on F1-Score, +9% on Precision and +8.3% on Recall, using the low-resolution *TinyVIRAT-v2* action recognition benchmark.

Contents

List of Figures

List of Tables

List of acronyms

AD-ResNet Adapted ResNet.

AI Artificial Intelligence.

APL Attention Pooling Layer.

CNN Convolutional Neural Network.

FPN Feature Pyramid Network.

MViT Multi-scale Vision Transformer.

ResNet Residual Network.

TTA Test Time Augmentation.

ViT Vision Transformer.

Chapter 1

Introduction

In recent years, the problem of understanding actions from videos is experiencing an increase in interest from the scientific community. The reason for this interest can be assimilated to the increase in the fields of applications of this task, just think that in recent years more and more car manufacturers are implementing autonomous driving in their vehicles. Even simply in the field of security cameras, this task can be used to analyze the actions that are happening at that moment. Other very common applications of this task can be human - robot interaction, gaming or entertainment [50].
Specifically, this task is called **action recognition** and it aims at understanding the actions that are happening in the video and assign a label to each action [40]. In figure 1.1 you can see some examples of actions that can be recognized in video recognition task.

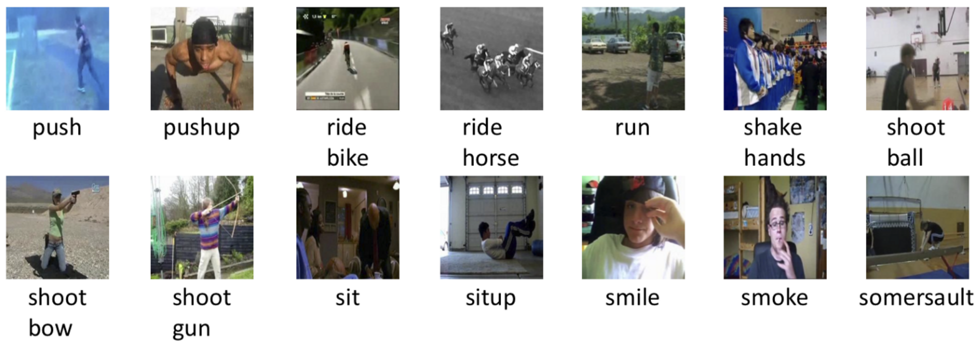


Figure 1.1: Examples of actions that can be recognized in the action recognition task.

In recent years, the increase in research interest in this task has led to the emergence of many datasets to address this problem. The availability of these large-scale, high-quality datasets has resulted in a significant improvement in this type of task [42].
In particular, datasets with multiple actors and multiple actions for each video, such as *UCF-101* [39], *Kinetics* [38], *AVA* [13], *YouTube-8M* [1] and *Moments-in-time* [26] allow you to have a huge amount of data with a large variety to be able to train neural networks [42]. This has allowed the development of various action recognition models, capable of being very effective in recognizing the correct actions. Some examples of these models can be *C3D* [44], *I3D* [3], *ResNet-3D* [14] and *R2+1D* [43].
The main problem with these datasets is that they ignore a large portion of real life videos, such as videos from low-resolution security cameras where the actions are not as clearly visible as in a high-resolution videos. The current state of the art of this task focuses only on high quality videos where the actions are clearly visible [42].
Existing approaches for this task re-scale high resolution video to create low resolution video. The problem with doing this however is that these videos don't exactly mirror the low-resolution videos. For example, low-resolution real-world videos suffer from camera sensor noise and other factors that are not present in down-scaled video [42].
Considering this context, the purpose of this work is to try to address the action recognition task applied to low resolution videos. In order to do this we will use the *TinyVIRAT* [8] dataset, a benchmark dataset containing natural low resolution videos taken by security cameras [42].
Addressing this type of problem with a model that is capable of recognizing actions in low-resolution videos could be beneficial for autonomous driving or, even better, in the video-security field, where it's very common to have low-resolution cameras.
The thesis is organized as follows:

- Chapter 2 introduces the action recognition task, talking about how it was investigated during the years and what are the main approaches to address this task, especially on low-quality videos;

- Chapter 3 explains our approach and what we have investigated to handle this particular task;
- Chapter 4 shows the ablation studies and the experiments that we have carried out in this context;
- Chapter 5 reports the concluding remarks.

Chapter 2

Low-quality Action Recognition in Computer Vision

Action recognition is the task of identifying the actions that are taking place in a video. The problem is even more challenging in low-resolution videos, since the action can be not clearly visible.
This chapter aims at illustrating this task and how it has been investigated in computer vision during past years. It is also showing some approaches to cope with low-resolution videos.
This chapter is divided into:

- Section 2.1 is illustrating the Action Recognition task, how it was investigated during the years and main approaches;

- Section 2.2 describes the Action Detection task, that is built upon Action Recognition task;

- Section 2.3 is illustration the Super-Resolution technique, a tool for transforming low-resolution tasks into high-resolution ones;

- Section 2.4 instead is illustrating the Deep Learning Multi-scale approach and some implementations.

2.1 Action Recognition on Videos

One of the most representative task in the field of video understanding is the video action recognition task. It is about understanding the actions that are taking place in the videos. This type of task has many applications in the real world, such as gaming, entertainment, human-robot interaction and also in the field of video surveillance [50].
This section aims at illustrating how this task has been investigated over the years and what are the current main approaches.

2.1.1 Hand-crafted features

The first approach for video and action recognition task is using hand-crafted features.
In particular, 2 hand-crafted features dominated the video understanding literature until 2015, bringing high accuracy and good robustness:

- Improved Dense Trajectories (IDC) [47];
- Stacked Fisher Vectors [29].

The first, *Improved Dense Trajectories (IDC)* [47] is a descriptor that is based on motion boundary histograms, which is robust to camera motions.
The second instead, *Stacked Fisher Vectors* [29], is a multi-layer nested Fisher vector encoding, that allows abstracting semantic information in a hierarchical way.

2.1.2 Rise of Deep Learning: Convolutional Neural Network

With the rise of deep learning [19], researchers started to adapt CNNs to video problems.
The first approaches, like the seminal work *DeepVideo* [17], proposed to use a single 2D CNN model on each video frame independently. *DeepVideo* [17] found that a model to which every single frame of the video is the input performs equally well when the input is changed to a stack of frames and this observation indicates that the learnt spatio-temporal features did not capture the motion well.

A conceptually easy way to understand a video is to the use a 3D tensor with two spatial and one time dimension. This leads to the usage of 3D CNNs as a processing unit to model the temporal information in a video.
The first work about using 3D CNNs for action recognition [16] was using a neural network that was not deep enough to show its potential.
Tran *et al.* [44] extended it to a deeper network, called *C3D*, that follow a modular design of [37], which could be thought of as a 3D version of *VGG16* [37] network. Its performance on standard benchmarks is not satisfactory [50], but shows strong generalization capability and can be used as a generic feature extractor for various video tasks [49].

2.1.2.1 ResNet-3D

2D CNNs enjoy the benefit of pre-training brought by the large-scale of image datasets, such as *ImageNet* [9].
One model that works very well on these large-scale datasets is the **ResNet**. Thanks also to the large-scale datasets, this is one of the models that is more accurate and generalize better in the Image Recognition task.
ResNet-3D [14] is the result by taking a standard 2D ResNet [15] and replace all the 2D convolutional filters with 3D kernels. This idea was inspired by the fact that by using a 3D CNNs together with a large-scale datasets one can exploit the success of 2D CNNs on *ImageNet*.
ResNet, that stands for *Residual Networks*, is a classic neural network used as backbone for many computer vision tasks and also, this model, was the winner of *ImageNet* challenge in 2015.
ResNet 2D [15] was the first model to introduce the concept of **skip connection**. On figure 2.1 you can see an example of skip connection. On the left part of the figure there is an illustration of a standard stack of convolution layers. On the right part of the figure, instead, it is illustrated what we consider a skip connection. In this case we also add the original input to the output of the convolution block.

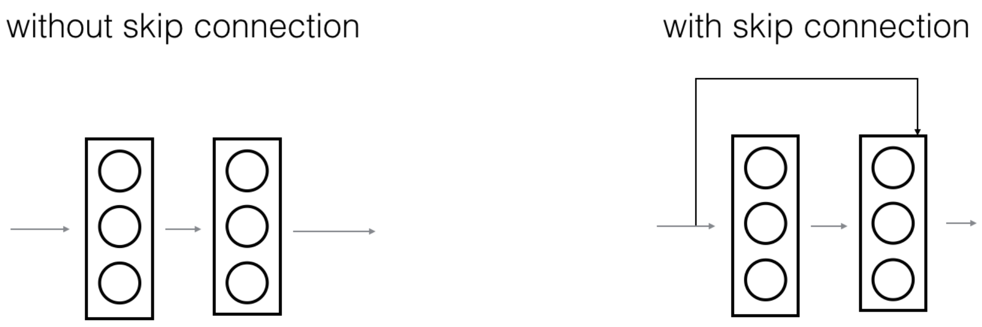


Figure 2.1: Illustration of the concept of "skip connection".

One important point is that you can perform the addition in the skip connection only if the original input and the output of the convolution block has the same shape.

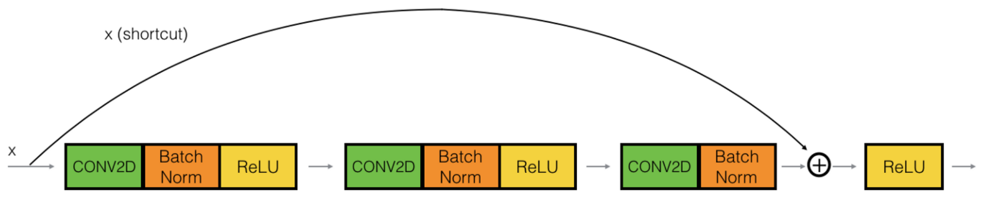


Figure 2.2: Skip connection when input and output of convolution block have the same shape.

If the convolution block is done in a way that the output shape is the same, then we can simply add them, as shown in figure 2.2.

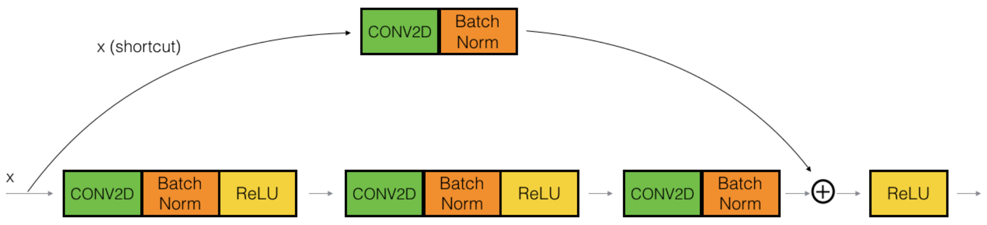


Figure 2.3: Skip connection when input and output of convolution block do not have the same shape.

Instead, if the convolution block is going to produce an output of a different shape, the original input goes through a convolution block in order to obtain the

same dimension as the output of the convolution block (figure 2.3).
There are 2 main reasons about why skip connections are very good:

- this alternative shortcut path is going to mitigate the problem of vanishing gradient, because it allows the gradient to flow through;
- it allows the model to learn an identity function which ensures that higher layer will perform at least as good as the lower layer

There are various types of ResNet based on the size of it, which in the case of ResNet is called *depth*, such as 18, 34 or 50. The architecture, however, always has the same structure, that is:

- a convolution block called *stem*, composed by a convolution and a max-pooling using 7×7 and 3×3 kernel sizes respectively;
- 4 convolution blocks, called *stages*, where each stage is identical to the others;
- a *classification* block, composed by an Average Pooling Layer followed by a fully connected layer.

Depending on the depth of the ResNet, the architecture of the stages will change, increasing the number of network parameters.
The complete architecture of the ResNet 18 can be seen in the figure 2.4.

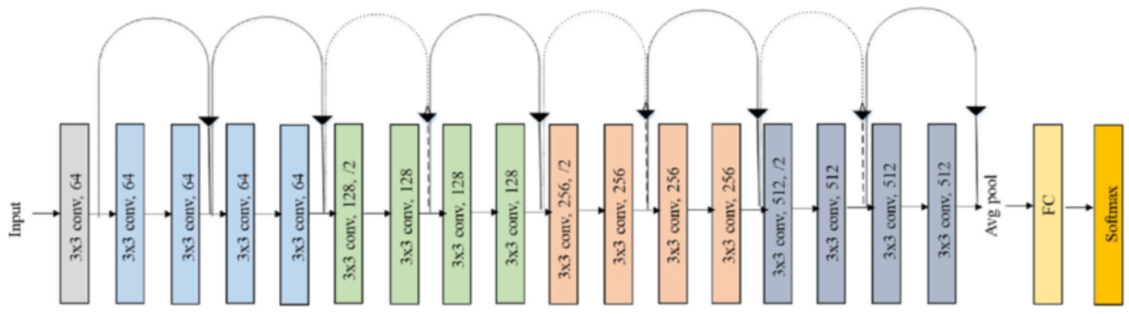


Figure 2.4: ResNet 18 architecture

2.1.2.2 R2+1D

3D networks are very hard to optimize. To train a 3D convolutional filter well, people need large-scale datasets with a variety of video contents and action categories.

To reduce the complexity of 3D network training, a lot of models like P3D [34] and R2+1D [43] explore the idea of *3D factorization*.
To be specific, 3D factorization is when a 3D kernel (e.g. $3 \times 3 \times 3$) can be factorized to two separate operations, a 2D spatial convolution (e.g. $1 \times 3 \times 3$) and a 1D temporal convolution (e.g. $3 \times 1 \times 2$). You can see an illustration of the factorization in figure 2.5.

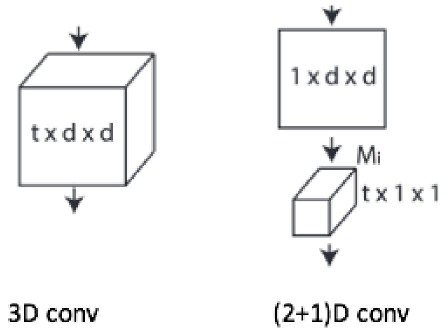


Figure 2.5: Difference between a 3D convolution block and a (2+1)D convolution block.

The R2+1D [43] (also referred as R(2+1)D) is composed of five (2+1)D convolution blocks (architecture shown in figure 2.6).
Using this type of factorization leads to two major benefits:

- the non-linearity of a (2+1)D convolution block is twice compared to the 3D convolution block, thanks to the fact that in the (2+1)D convolution block we have two computation phase;
- the optimization is easier in a (2+1)D CNN with respect to a 3D CNN block with the same number of blocks.

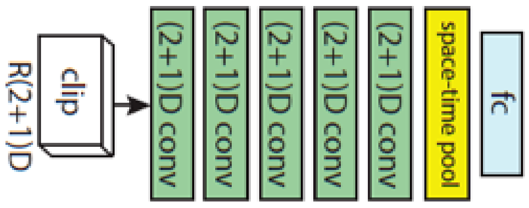
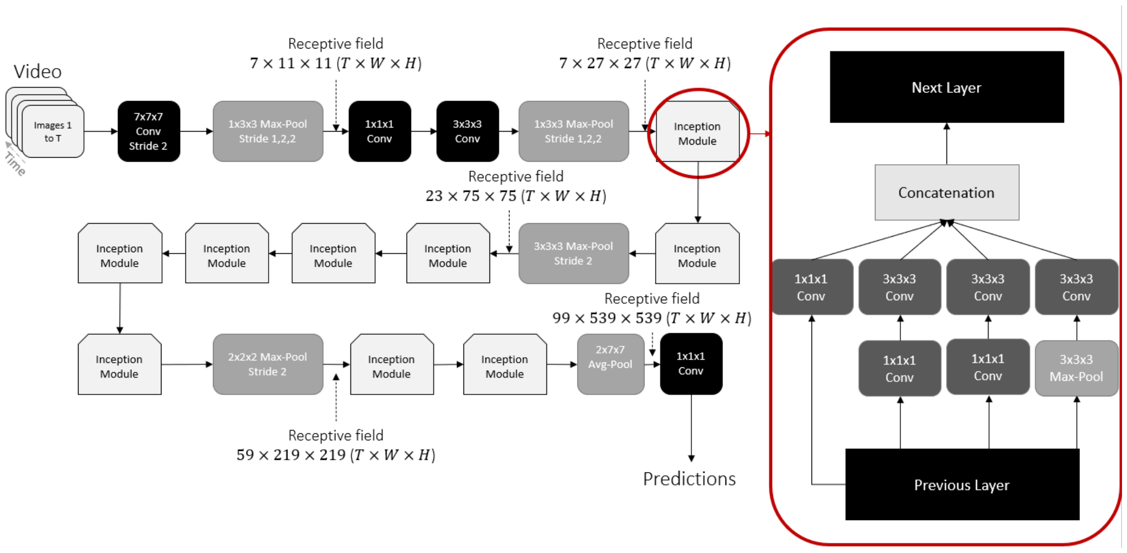


Figure 2.6: R(2+1)D architecture.

2.1.2.3 I3D

The architecture of the I3D [3] model was born from the idea of taking a 2D architecture and modifying all filters and kernel pools to add an additional dimension, making the filters that were previously $N \times N$ become $N \times N \times N$.
This leads to an interesting fact: you can re-use the weights of the original 2D filters by repeating N times its weights along the time dimension.



Video
Images 1 to T
Time
7x7x7 Conv Stride 2
1x3x3 Max-Pool Stride 1,2,2
1x1x1 Conv
3x3x3 Conv
1x3x3 Max-Pool Stride 1,2,2
Inception Module
Receptive field $7 \times 11 \times 11$ ($T \times W \times H$)
Receptive field $7 \times 27 \times 27$ ($T \times W \times H$)
Receptive field $23 \times 75 \times 75$ ($T \times W \times H$)
3x3x3 Max-Pool Stride 2
Receptive field $99 \times 539 \times 539$ ($T \times W \times H$)
2x2x2 Max-Pool Stride 2
2x7x7 Avg-Pool
Receptive field $59 \times 219 \times 219$ ($T \times W \times H$)
Predictions
Next Layer
Concatenation
1x1x1 Conv
3x3x3 Conv
3x3x3 Max-Pool
Previous Layer

Figure 2.7: I3D architecture.

Another modification to consider is the receptive field of pooling and convolutional layers. A *receptive field* in a convolutional neural network is the part of the image that is visible to one filter at a time. The receptive field will increase by adding more and more layers.
2D convolutions and pooling focus on the height and width of the image and therefore are symmetrical (e.g. 7×7 kernel or 3×3 kernel).
When a temporal dimension is included, it is important to find the optimal receptive field, which is dependent on the frame rate and frame dimensions.
By the authors [3], if the receptive field grows too quickly in time relative to space, it may conflate edges from different objects, breaking early feature detection. If receptive field grows too slowly, it may not capture scene dynamics as well. For this reason, kernels in I3D are not symmetrical because of the additional time dimensions.

In figure 2.7 you can see the architecture of the I3D model. As you can see, the beginning of the network uses asymmetrical filters for max-pooling, maintaining time while pooling over the spatial dimension. When the model run convolutions and pooling with the time dimension, filters becomes symmetrical filters.

2.1.3 Vision Transformers

The *Transformers* [46] is an architecture that has out-rivaled the competing Natural Language Process (NLP) models after its release.
This model can be generalized also to other application, like Computer Vision. When applying Transformers to Computer Vision, we talk about **Vision Transformers** (ViT) [10]. In this case, the image is splitted into multiple patches and the image patches are considered as words. In this case, ViT provides embeddings of the patches to the transformer.
The architecture is illustrated in figure 2.8. Input image is sliced into patches of size $P \times P$. Each patch is flattened and linearly mapped into a D dimension vector, called *embedding stage* of the ViT. Position embeddings from the original transformer and class tokens are added to the patch embedding. The position is considered as a single number (e.g., in figure 2.8, from 1 to 9) because a pair of (x, y) position was not helpful for the model. Then the whole process is going to convert image patches into tokens, like in the NLP tasks, in fact the encoder transformer has not been modified.

ViT can also incorporate CNNs for further improving performance.

2.2 Action Detection on Videos

On videos you could have two types of action detection on videos: temporal and non-temporal action detection [45]. Both of these tasks are built upon the action recognition task, which aims simply at classifing the categories of a video clip.
Temporal action detection aims at localizing the action instances in time and recognize their categories. This task is very similar to segmentation, but done in time. Instead, *non-temporal action detection* is very similar to object detection. These

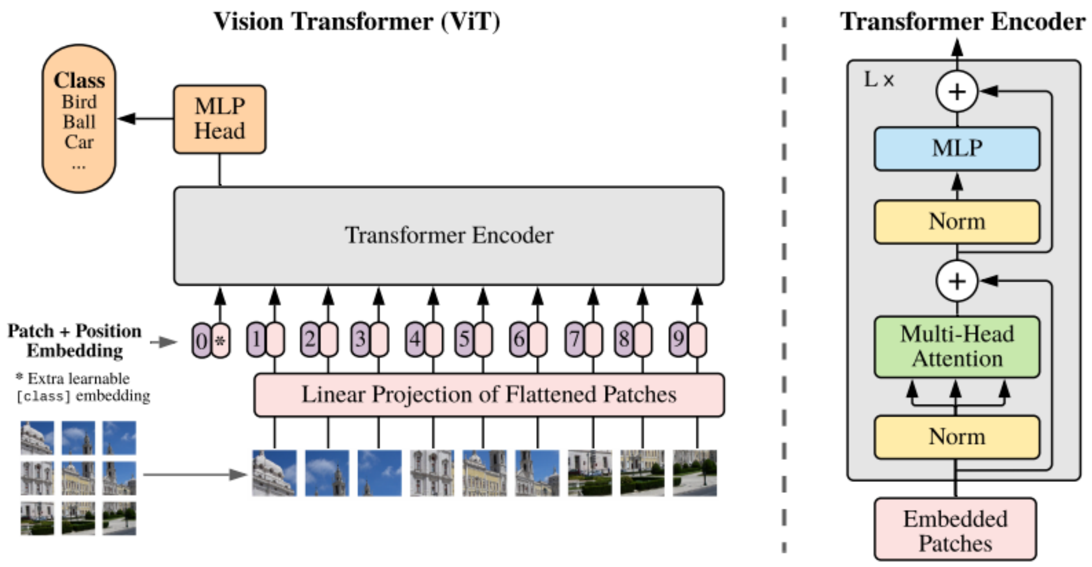


Figure 2.8: Vision Transformer (ViT) architecture.

problems aim at localizing objects/actions of interest in the spatial context.
Action detection has drawn much attention in recent years and has broad applications in video analysis tasks. For example, in video surveillance area, where an action could appear in a short period of time and the whole video that has been recorded from the camera is very long. This task is very time-consuming for an human, so having an automatic detection has a great advantage. This task has also another application that in recent years is growing so much: autonomous driving.

2.3 Turn a low resolution task into an high resolution one

A practical approach to adapt every state of the art model for a video understanding task that works with high-resolution videos to low-resolution videos is simply trying to increase the video resolution [23] (figure 2.9).
The *Super-Resolution* technology, either on images or video, has progressed a lot in recent years.

There are various methods to do super resolution of images or videos [22]. They mainly falls in two categories: traditional methods and deep learning based methods.

Figure 2.9: Basic idea of the Super Resolution technology.

In the first one the idea is to estimate motions by affine models. Instead, in the second one, the idea is to use deep learning models.
Some examples of video super resolution deep learning methods can be *BasicVSR* [4] (used also in [23] with TinyVIRAT dataset), *TecoGAN* [5] and *SOF-VSR* [48].

2.4 Multi-scale Recognition

In Section 2.3 we have illustrated the basic idea of Video Super-Resolution technology. Using this technique we could obtain an higher resolution video in order to train a state of the art model.
This section, instead, is going to present how to cope with different-scale dataset, where you have objects at different scale. In particular, this section presents the Multi-scale models idea and some (different) implementations.

2.4.1 Multi-Scale Models

Multi-scale modelling [28] is a style of modelling neural network where multiple models at different scales are used simultaneously to describe a system.
The need for multi-scale modelling comes usually from the fact that the available macro-scale models are not accurate enough and the micro-scale models are not efficient enough [11]. Combining both these models the hope is to arrive to a reasonable compromise between accuracy and efficiency.

2.4.2 Feature Pyramid Networks

One approach to address object detection at different scales is to use a *pyramid of image* at different scale. Processing multiple scale images is time consuming and the memory demand is too high to be trained simultaneously.
Alternatively, we can create a pyramid of features maps closer to the image layer composed of low-level structures. On figure 2.10 you can see an illustration of pyramid of images and pyramid of feature maps.

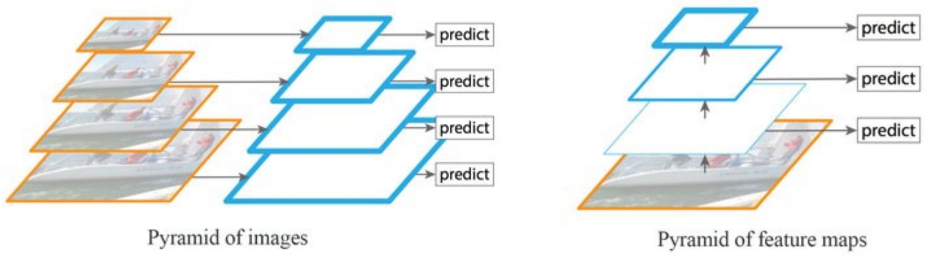


Figure 2.10: Illustration of pyramid of images (left) and pyramid of features maps (right).

Feature Pyramid network (FPN) [21] is designed for such pyramid concept in order to have both accuracy and speed. In replaces feature extract of detectors like Faster R-CNN [35] and generates multiple feature map layers (*multi-scale feature maps*) with better quality information than regular feature pyramid for object detection.

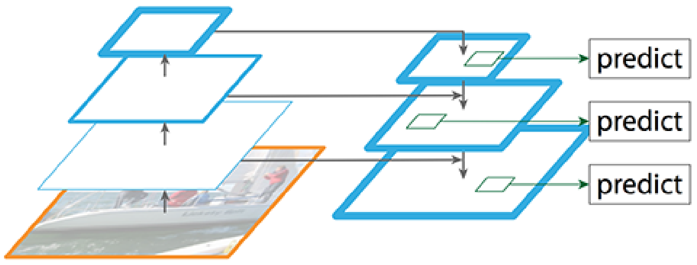


Figure 2.11: Feature Pyramid Network (FPN) architecture.

The architecture is illustrated in figure 2.11. It is composed of a bottom-up and a top-down pathway. The first one is, usually, a convolution neural network for feature extraction (e.g. ResNet). By going up, the spatial resolution decreases

while the semantic value increases.
In the top-down pathway instead the idea is to construct higher resolution layers from a semantic rich layer. The reconstructed layers are semantically strong, but the location of objects are not precise after all the down-sampling and up-sampling. To cope with this we add a lateral connection between reconstructed layers and the corresponding feature maps, in order to help the detector to predict the location better. This also acts as skip connection to make the training easier (like the ResNet, section 2.1.2.1).

2.4.3 MViT: Multi-scale Vision Transformer

The **Multi-scale Vision Transformer** architecture [12] is a *Transformer* [46] architecture for representation learning from visual data, such as images and videos. It incorporate the seminal concept of hierarchical representations into the Transformer architecture.
In figure [12] you can see the illustration of the models' architecture. The key idea for the hierarchical representation is that you apply transformer multiple times, where each time the number of patches is divided by two because patches are grouped by pairs.

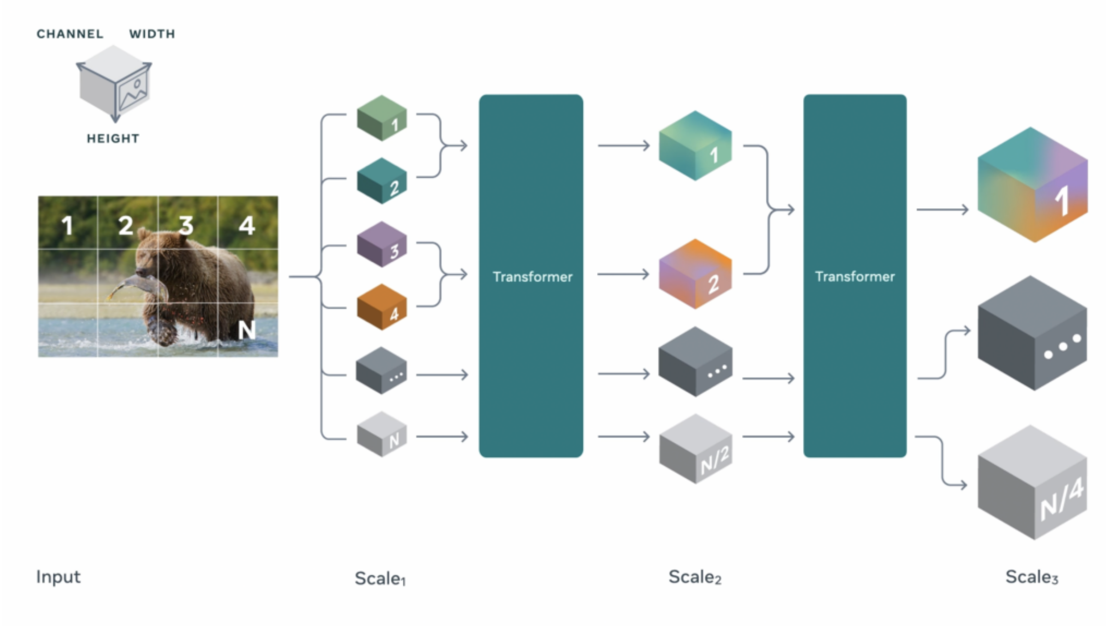


Figure 2.12: Multi-scale Vision Transformer (ViT) architecture.

Chapter 3

Methodology

Chapter 2 explains the task of action recognition on low resolution video, illustrating why this task is important and what are the main approaches of the literature in recent years.

This chapter contains a description of the working method and what approaches has been adopted to be able to cope with this type of task. This chapter is divided into the following sections:

- Section 3.1 explains two different methods to extract fixed-length clips from a video;
- Section 3.2 illustrates a simple procedure to adapt ResNet architecture in order to work with inputs of smaller scale than the standard one (224×224);
- Section 3.3 describes a set of data augmentation technique that has been used in this work;
- Section 3.4 illustrates different test time augmentation techniques;
- Section 3.5 indicates the importance of choosing a good positive-negative threshold;
- Section 3.6 shows a multi-scale implementation of ResNet model;
- Section 3.7 proposes a *Feature Pyramid Network* using ResNet as backbone;

- Section 3.8 illustrates a model that is the fusion of the one described in 3.6 and the one described in 3.7;
- Section 3.9 describes how to add *attention* to a ResNet;
- Section 3.10 proposes an implementation of Spatial-Temporal Discriminative Filter Banks [24] to a *ResNet-3D* and to a *Feature Pyramid Network*;
- Section 3.11 contains a description of two different loss functions that can be used in a multi-label classification task.

3.1 Clip Sampling

This section aims at illustrating two different techniques to sample a *clip* from a video. A *clip* is a fixed-length sub-video extracted from the original video.
The *clip* is then used as input for the models.

3.1.1 Random Clip Sampling

In the *random clip sampling* technique the clip is sampled starting from a frame that is chosen randomly between all the frames in the clip.
We firstly select (randomly) one frame between all the frames in the video and then we select the $N-1$ (usually $N=16$ or $N=32$) following frames, in order to obtain N total frames. On figure 3.1 you can see an illustration of the process.

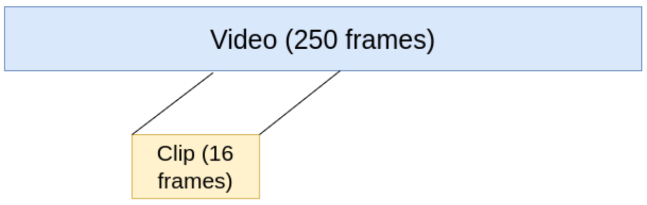


Figure 3.1: Illustration of Random Clip Sampling. After choosing the initial frame, all the following $N-1$ frames are selected.

3.1.2 Segment Based Sampling

When having a dataset where the number of frames is not so big, it could be useful to use *segment based clip sampling*.
In this technique we divide the frames in N (where N is the number of frames that we want to sample) disjoint chunks. After having N frame chunks, we sample one frame per chunk in order to obtain exactly N frames.
This technique can be very powerful when the action is visible temporally in the whole video or when the average number of frames in the dataset is very low. Instead, this technique is not very good when you have a long video and the action is visible only in a small part of it.

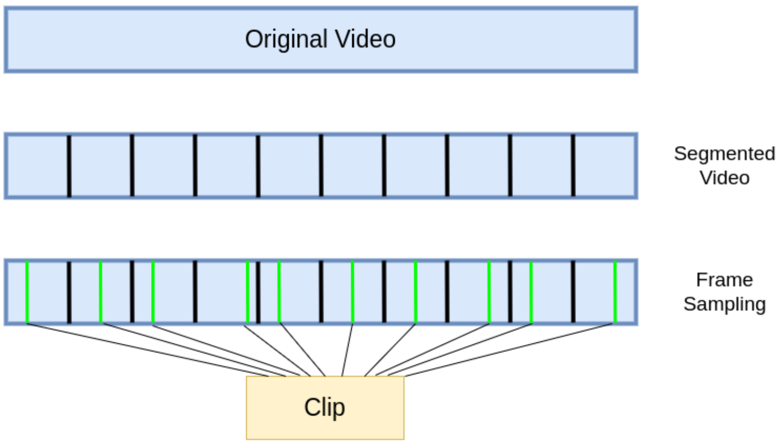


Figure 3.2: Illustration of Segment Based clip Sampling, with $N = 10$. In this example the frames (green lines) that are extracted from the segments are chosen randomly.

We can employ two types of frame sampling in a single chunk:

- random: select the frame randomly according to a uniform distribution;
- center: select always the central frame of the chunk.

Random frame selection should be used in training phase, in order to train the model on similar but different clips, while the center frame selection should be used for validation and test phase. Figure 3.2 is illustrating the (random) segment based sampling using $N = 10$.

3.2 Using very small inputs for standard ResNet

ResNet [14] has been described in the section 2.1.2.1. The standard architecture of the ResNet is designed to work with an input with spatial resolution of 224×224. Since we are in a low-resolution task, videos are probably of a resolution that is lower of 224×224. If we upsample the video to 224×224 we would get a very grainy video which certainly would lead to bad results.
So if we want to use, for example, 112×112 as input size instead of 224×224 we have to change the architecture to ensure that the output of the last stage is always 7×7, making the architecture more robust on these very small input sizes.
The best thing that we could modify in order to obtain this is the *stride* of the various pooling and convolutional layers. Using a value of strides of 2 means that the output will be half the size of the same layer with stride 1. ResNet architecture has many layers with stride 2 and this is not good for small input size since it will instantly half the information contained in the video. This is not a problem for high-resolution video, because if you half the spatial size the video it is still recognizable.
This means that if we want to use an input size of 112×112, to adapt the architecture we should change the first layer that has a stride of 2 to a stride of 1. Similarly, if we want to use an input size of 56×56 we need to change the first two layers that has a stride of 2 to a stride of 1.
This approach can be very effective, but it has a limitation: we need to have that

$$\log_2 \left(\frac{224}{\text{input size}} \right) \text{ to be an integer}$$

If this constraint is met, then the output of the last stage will be 7×7, like in the original architecture. In practice, the available input sizes are 14×14, 28×28, 56×56 and 112×112.

3.3 Advanced data augmentation

Data augmentation is a set of techniques that are used to increase the amount of data simply by adding slightly modified copies of existing data. It is a regularization

technique and it helps reduce overfitting [36].
Previous works has demonstrated the effectiveness of data augmentation [30].
For this document a combination of different augmentation techniques has been used. In particular, in order:

1. random short side scale;
2. random crop;
3. random horizontal flip;
4. RandAugment.

Some of these techniques will be explained on the images, but the idea is replicable on videos.

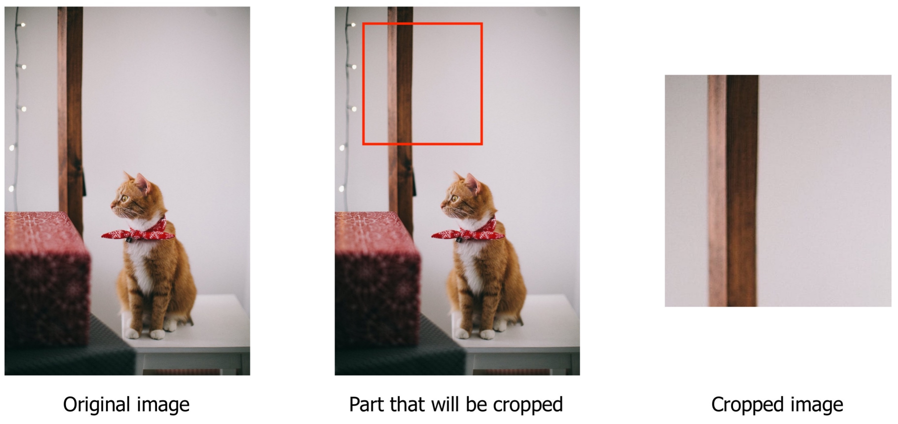


Figure 3.3: Example of image random cropping.

3.3.1 Random Short Side Scale

Random short side scale is a data augmentation technique. In this technique the idea is to do a resize of the frames based on the short side of the frames.
It is called random because you need to define a range of values and the algorithm is going to choose the size randomly in this range. Also, the aspect ratio is maintained. For example, if we have a 60×80 and we define a range of $100 - 130$, then a number between 100 and 130 is chosen. Let, for example, that the random number is 120. Then the short side (so 60) is scaled to 120 and the other scale (80) is scaled

proportionally. The output then will be a 120×160.
Since the dataset is composed of squared videos (i.e. both sides has the same dimension), this is easier and we will obtain again squared videos.

3.3.2 Random Crop

Random cropping is a data augmentation technique where the video are cropped in a random position.
The only parameter that this augmentation technique have is the output size. The output, for simplicity, is squared.
Figure 3.3 is illustrating an example of image cropping.
Choosing a good output size is crucial. If the output size is very small with respect to the input size, then a lot of information will be lost. Instead, if the output size is similar to the input size, then cropping is pretty useless. The perfect size should be a trade-off where most of the information is maintained and the cropping is changing the point of view of the input.

3.3.3 Random Horizontal Flip

Random horizontal flip is an augmentation technique where you randomly flip horizontally the input (figure 3.4).

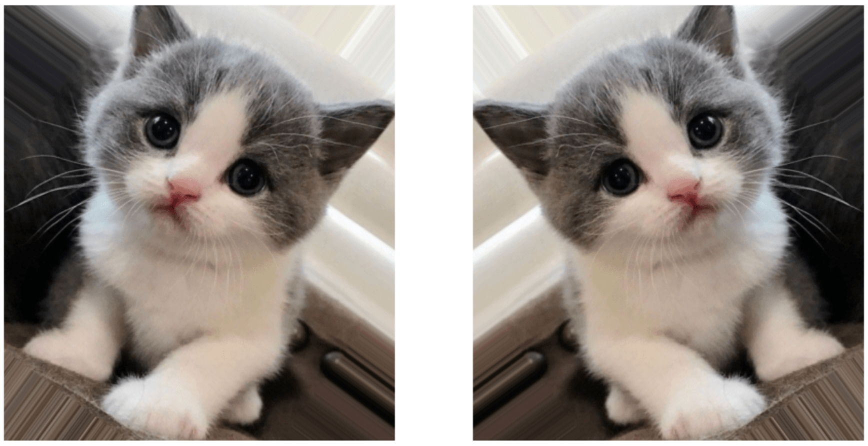


Figure 3.4: Example of horizontal flip.

It is called random because the only parameter of this data augmentation

technique is the probability of flipping. The idea behind this augmentation technique is that the model will learn the correct information also if you flip the input.
This technique cannot be used when you have actions that are indicating a left or right direction. For example, if the dataset has an action "pushing to the left", then flipping will result in an action that is "pushing to the right", but the model will wrongly learn "pushing to the left".

3.3.4 RandAugment

RandAugment [7] is a recent data augmentation technique. It is an automated data augmentation method where you have a search space with only two hyperparameter: N and M.
N is the number of augmentation transformations to apply sequentially, instead M is the magnitude for all the transformations.

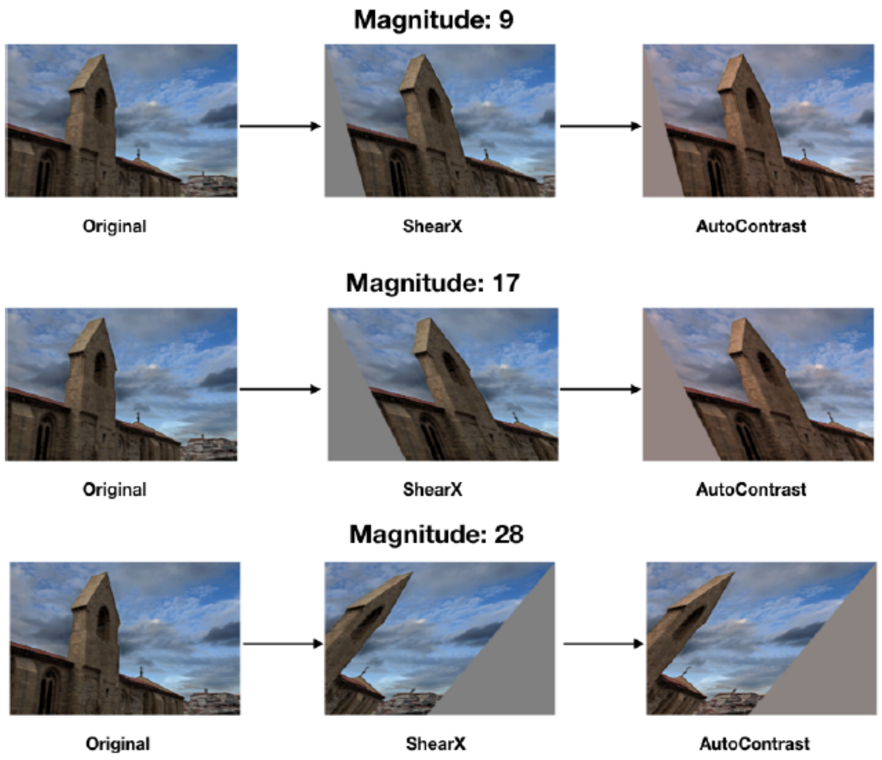


Figure 3.5: Example of *Rand Augment* with $N = 2$ and different values for magnitude M.

Each of the single transformation that can be applied is constrained by a probability value. Transformations that can be applied include identity transformation, auto contrast, equalize, rotation, solarization, color jittering, posterizing, changing contrast, changing brightness, changing sharpness, shear-x, shear-y, translate-x and translate-y.

3.4 Applying Test Time augmentation

This section is going to illustrate two Test Time Augmentation (TTA) technique that has been exploited in this work: *Horizontal Flip* and *K-clips*.

3.4.1 Horizontal Flip

The **Horizontal Flip** Test Time Augmentation (TTA) technique is done by sampling one clip, the horizontal flipped ones from the initial clip and do model ensemble with these clips.
By doing this procedure and getting the max value for each class is going to make the final prediction more robust.
Since this augmentation is very similar to the *random horizontal flip augmentation* described in section 3.3.3, this technique is applicable only if the dataset does not contain actions characterized by right or left directionality.

3.4.2 K-clips

The idea of **K-clips** Test Time Augmentation (TTA) technique is to sample K clips from the original video and do model ensemble with these clips.
Similarly to the *Horizontal Flip TTA*, the idea is that this is going to make the final prediction more robust.
One example for a value of K is 5, that is used in a lot of works.
This technique can be easily used with the *Random Clip Sampling*. Instead, if we are using the *Segmented Based Sampling*, we need to use the *random frame extraction* instead of the *central* one, otherwise the K clips will be K copies of the same clip.

3.5 Positive-Negative threshold

The model outputs are values between 0 and 1, which indicates the percentage of memberships in that class.
Considering that labels are defined only of 0s and 1s, we need to define a threshold, called **positive-negative threshold**, where

$$output[x] = \begin{cases} 1 & output[x] > threshold \\ 0 & output[x] \leq threshold \end{cases}$$

otherwise it is not possible to compare the output of the models with the actual labels and it is not possible to calculate losses and scores.
A rational choice of a correct threshold value could be

$$threshold = 0.5$$

This seems reasonable, since if the percentage of membership to a class is greater than 50% then this should be considered part of that class.
However, this is not always the best choice. Different threshold values, depending on the model, dataset and training procedure, can lead to better results than the standard 0.5. This can be seen in plot 3.6, where you can see the scores obtained by using different threshold values. As you can see, 0.5 is not corresponding to the optimal scores. If you consider $F1 - Score$ as your preferred score, then 0.1 is a good threshold value, instead for $Precision$ 0.4 is good.

For this reason it is necessary to make an accurate choice of a correct threshold value.

3.6 Multi-scale ResNet-3D Model

Using low-resolution dataset can be very tricky when you want to fix a scale for the model. For example, if you have a dataset that has videos that goes from 10×10 to 128×128, what kind of resolution should you consider for the inputs of your model?
Downsampling clips results in losing a lot of details from the largest clips in the

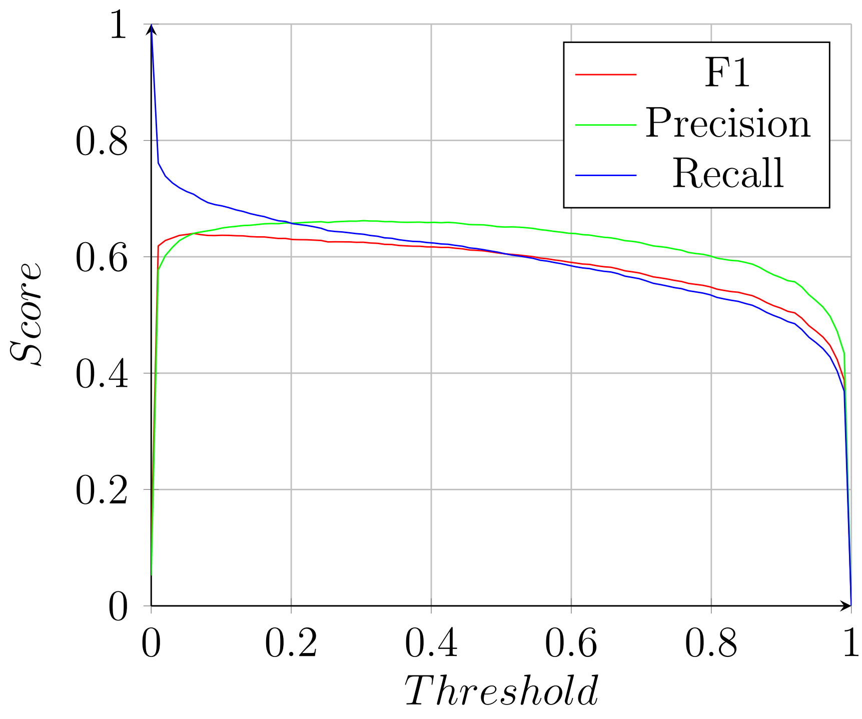


Figure 3.6: Scores obtained using different values of threshold on an *ResNet18* model on *TinyVIRAT-v2* dataset validation split.

dataset. Instead if you upsample 10×10 clips to 128×128 you are going to have a clip that doesn't have any details in it, because it will look a lot grainy.

In this cases using a **multi-scale model** can be a good choice. You could define more models for different scale, in order to not upsample and downsample the videos. Multi-scale models have already been illustrated in section 2.4.1.

This work proposes an implementation of a multi-scale version of the ResNet-3D.

As you can see from the architecture in figure 3.7, the idea is to use N different ResNet-3D. At the output of stage 1, 2 and 3 of each ResNet-3D, the embeddings flows from each architecture to the ones with smaller scale. This connection is called **lateral connection** (in figure 3.7 are colored in red) and its purpose is to enrich the semantic information that has the lower scale output with the semantic information that has the higher scale output. The *lateral connection* is implemented with a concatenation in the channels dimension.

After the last stage, the stage 4, the outputs are summed together in order to have a final single classifier.

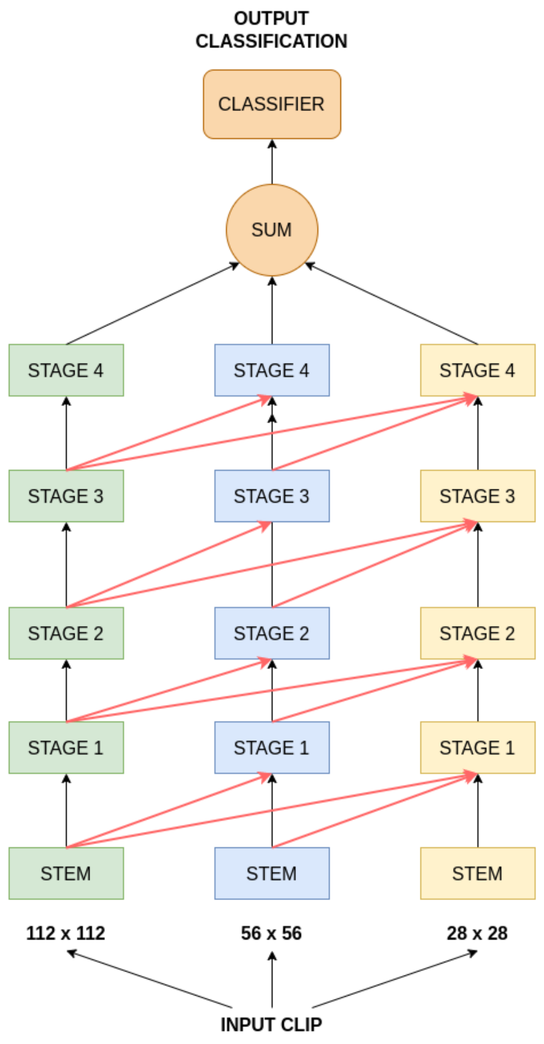


Figure 3.7: Multi-scale ResNet architecture. In this particular example we have considered 3 different input sizes, 28×28, 56×56 and 112×112.

3.7 ResNet-3D Feature Pyramid Network

Another type of multi-scale models are the *Feature Pyramid Networks*, explained in section 2.4.2.

This work proposes an implementation of a **ResNet-3D Feature Pyramid Network**, which architecture can be seen in figure 3.8.

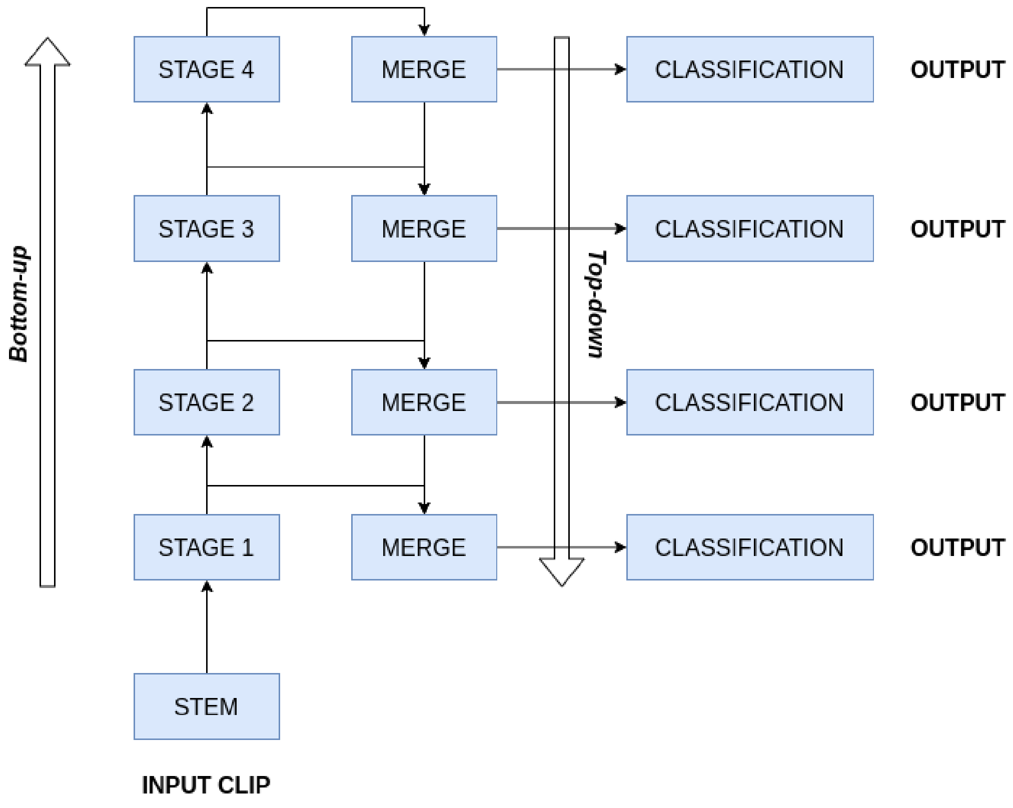


Figure 3.8: ResNet-3D Feature Pyramid Network architecture.

This is a particular version of the *Feature Pyramid Network* where the CNN backbone (bottom-up pathway) is the ResNet-3D. When going into top-down pathway, the lateral connections are coming from the outputs of the stage 1, 2 and 3 of the ResNet-3D. Then the classification is done for each of the different scales feature maps and the different classifications are combined together to have the final outcome.

3.8 Multi-scale ResNet-3D Feature Pyramid Network

This work also proposes a fusion between the *Multi-Scale ResNet-3D* model (section 3.6) and the *ResNet-3D Feature Pyramid Network* (section 3.7).

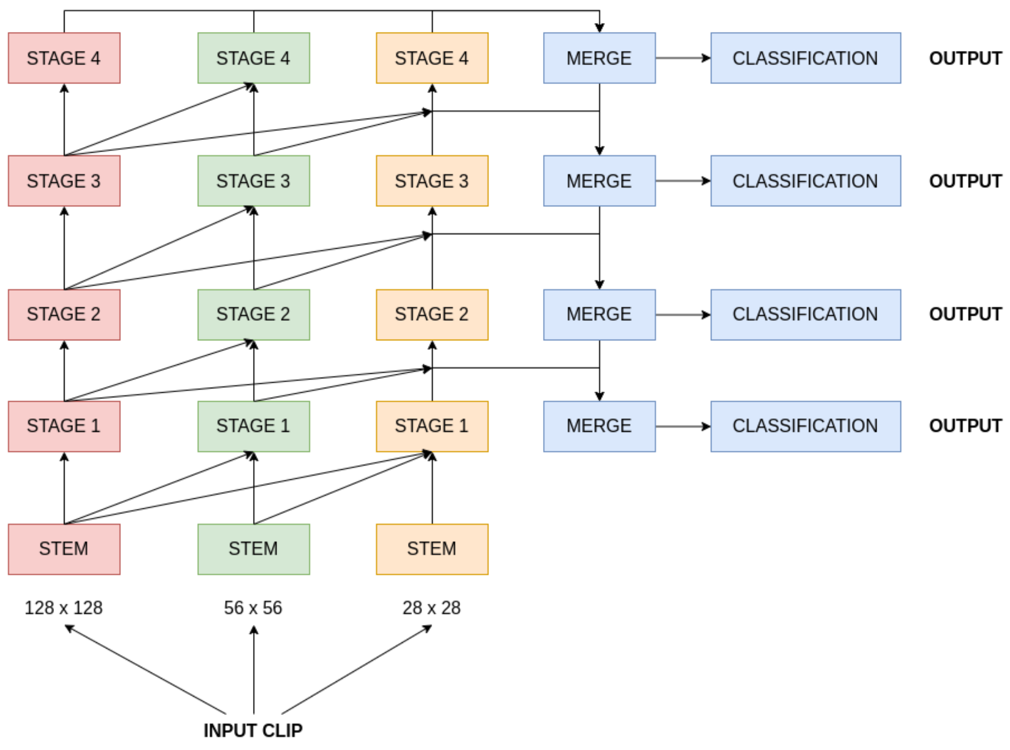


Figure 3.9: ResNet-3D Multi-Scale Feature Pyramid Network architecture. In this particular example we have considered 3 different input sizes, 28×28, 56×56 and 112×112.

The architecture of the *ResNet-3D Multi-Scale Feature Pyramid Network* is illustrated in figure 3.9.
The idea of the model is to have a *Feature Pyramid Network* where the bottom-up approach is composed of a *Multi-Scale ResNet-3D*.
By doing this, the strengths of the two models should come together.
As you can see from the architecture, the bottom-up pathway is done by a *Multi-scale ResNet-3D*. In the top-down pathway you have again the lateral connection that are coming from the bottom-up path. The particularity is that the lateral

connection of the *Feature Pyramid Network* are coming from the concatenation of the various outputs of the *Multi-scale ResNet-3D*, so only after the lateral connection of the *Multi-scale ResNet-3D*.
Then you have the same classifiers, one for each different scale feature maps. Each output is then combined together to get the final output.

3.9 Applying Attention Pooling Layer (APL) in a ResNet-3D

Attention is a technique that mimics cognitive attention. Also, the effect enhances some parts of the input data while diminishing other parts [46], based on the area of interest of the image/video.
This work proposes a modification of the standard ResNet-3D in order to use attention. This is done by changing the Average Pooling Layer at the end of the network with an **Attention Pooling Layer**.
An *Attention Pooling Layer* is composed of two parts: a *multi head attention layer* followed by an *average pooling layer*.
The idea behind this modification is that the *Attention Pooling Layer* will enhance some part of the input and then the prediction will be more accurate.

3.10 Using Spatial-temporal discriminative filter

Action recognition has seen a performance improvement in the last yaers. State-of-the-art literature aims at improving performance through changes to the CNN network or they explore different trade-offs between computational efficiency and performance.
Almost all of these works maintains the same last layers of the network: global average pooling followed by a fully connected layer.
An interesting improvement could be done in this phase [24]. We could apply a different classification block, that explores the finer details of the input.

In figure 3.10 you can see an illustration of the architecture.

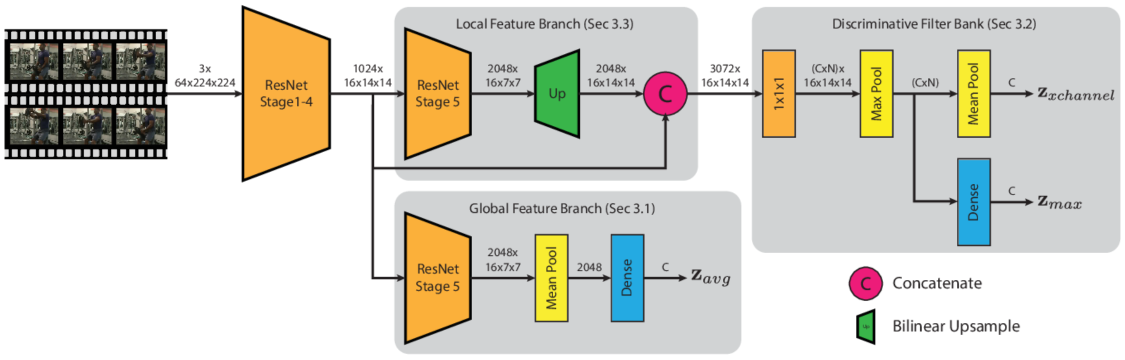


Figure 3.10: Architecture diagram of discriminative filter banks. *Global feature branch* is identical to the baseline. The approach is going to improve the baseline with a bank of discriminative filters that specialize on localized cues and a local feature extraction branch, that produces feature maps tuned to be sensitive to local patterns [24]

We have three branches. The first one, the *Global Feature Branch*, is the classic baseline branch with average pool and fully connected layer. Here we have also another branch, the *Local Feature Branch*, that is going to specialize the finer details. After this branch there is the *Discriminative Filter Bank* that is going to focus on these key local regions and to classify correctly the overall input.
Finally, to combine the three classifier outputs into a single prediction, we sum the three values.
Since this is a ResNet (2D or 3D) architecture proposal, we have applied it on the standard ResNet-3D architecture. Also this thesis proposes an additional *Feature Pyramid Network* where the classifier part on the top-down pathway is done using this approach.

3.11 Loss Functions

Loss functions plays an important role in neural network model because they define an objective which the performance of the model is evaluated. Then, parameters learned by the model are determined by minimizing a chosen loss function.
Changing the loss function can therefore lead to a performance improvement.

3.11.1 Binary Cross Entropy

Binary Classification is a problem where we have to segregate our observations in any of the two labels on the basis of the features.

Binary Cross Entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value.

To be precise, *Binary Cross Entropy* is the negative average of the log of corrected predicted probabilities.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

This loss function is going to use the **Entropy**, that is a measure of the uncertainty associated with a given distribution $q(y)$.

This loss function is one of the most used in *multi-label classification task.*

3.11.2 Asymmetric Loss

A work from the 2021 TinyAction Challenge has shown that the positive-negative imbalance nature in the multi-label datasets may hurt the optimization process [41].

Common practices in multi-label classification adopt a binary cross-entropy loss. To handle the problem of positive-negative imbalance, we can consider what is called **Asymmetric Loss (ASL)** [2]. It is defined as

$$X(m, n) = \begin{cases} L_+ = (1 - p)^{\lambda_+} + \log(p) \\ L_- = (p_m)^{\lambda_-} + \log(1 - p_m) \end{cases}$$

where $p = \delta(z)$ is the network's output probability.

Asymmetric loss contains two mechanisms of asymmetric focusing and probability shifting, that are integrated into a unified formula using soft thresholding, via the focusing parameter λ, and hard thresholding, based on the probability margin m. In the asymmetric loss, both mechanisms are used for reducing the contribution of easy negative samples to the loss function.

Since in the low-resolution action recognition task we would probably handle multi-label datasets, this loss function could be useful.

Chapter 4

Experiments

Chapter 3 proposed the approach that has been used to address the *Action Recognition* task on Low-Resolution videos. It proposes models, clip sampling methods and other training settings.
This chapter is gonna present all the experiments carried out in order to obtain the best result on a low-resolution action recognition dataset.
This chapter is divided into:

- Section 4.1 describes the setup used for experiments, illustrating the tools that has been used, the *TinyVIRAT* [8] dataset, the evaluation metrics and the training setup;
- Section 4.2 illustrates the baseline model used as point of reference for all the experiments;
- Section 4.3 contains all the ablation studies carried out in this work;
- Section 4.4 reports the final experimental results by comparing the best result with the baseline model.

4.1 Setup

This section describes the tools used, the *TinyVIRAT* dataset and the evaluation metrics used in this work. It also describes the training details, like parameters initialization and learning algorithm.

4.1.1 Tools

This section will present all the tools that has been used to develop each experiment that is reported in this document.

4.1.1.1 Python

Python [31] is a high-level, interpreted and general-purpose programming language used to build websites, softwares, automate tasks and conduct data analysis. It is dynamic and free open source. It has efficient high-level data structures and a simple but effective approach to object-oriented programming.

Figure 4.1: Python [31] logo.

Python is a great choice for developing machine learning algorithms a models. There are various reasons to use python in machine learning:

- it has a huge number of **libraries** and **frameworks**: python comes with many libraries and frameworks that helps the developer coding;
- **simplicity**: python code is readable and concise;
- massive **online support**: as python is an open-source programming language, it has a very excellent support from many resources and documentation worldwide;
- **fast development**: its syntax is very easy to understand and friendly and together with libraries and frameworks the application's develop time will be very short;
- **flexible integration**: a python project can be integrated with projects written with a different programming language;

- **extensible and portable**: you can write a portion of your code in other languages (like C++) and also you can move to code to any machine and, without making any change, you can run it;

- **visualization tools**: some of python libraries offer good visualization tools and this is very important for AI, Machine Learning and Deep Learning where you need to present data in a human-readable format.

4.1.1.2 PyTorch

PyTorch [32] is an open source machine learning framework written in Python (section 4.1.1.1). It is used for applications such as computer vision and natural language processing (NLP) and it was primarily developed by Facebook's AI Research lab (FAIR).

Figure 4.2: PyTorch [32] logo.

A lot of deep learning software are developed using PyTorch, like Tesla Autopilot and Uber's Pyro.
It has the following features:

- **distributed training**: it is scalable to distributed training;

- **robust ecosystem**: a lot of tools and libraries extends PyTorch;

- **cloud support**: it is well supported on major cloud platforms, providing friction-less development and easy scaling.

4.1.1.3 PyTorchVideo

PyTorchVideo [33] is a deep-learning library with a focus on video understanding work. It's based on PyTorch (section 4.1.1.2) and it provides reusable, modular and efficient components that a developer can use to accelerate the video understanding research. It supports different deep-learning video components like video models, datasets and video-specific transforms.

Figure 4.3: PyTorchVideo [33] logo.

The key features of PyTorchVideo are:

- **based on PyTorch**: it's easy to use all of the PyTorch-ecosystem components;
- **reproducible model zoo**: it contains a variety of the state of the art pre-trained video models with the associated benchmarks and it also contains a variety of datasets;
- **efficient video components**: it has some video-focused fast and efficient components and it supports accelerated inference on hardware.

4.1.2 Dataset

In this section we present the *TinyVIRAT* [8] dataset and it's extension, *TinyVIRAT-v2*, that is the dataset used in this document.

4.1.2.1 TinyVIRAT dataset

TinyVIRAT [8] is a dataset, based on *VIRAT* [27] dataset, for real-life tiny action recognition problems.
VIRAT [27] dataset contains a variety of different actor sizes and it is very complex because actions can happen any time in any spatial position. In order to have a low-resolution action recognition problem, the *TinyVIRAT* dataset is obtained by cropping small action clips from *VIRAT* videos and it's restricted to outdoor videos [42].
TinyVIRAT [8] dataset has 7663 training and 5166 testing videos with 26 action labels.

4.1.2.2 TinyVIRAT-v2 dataset

TinyVIRAT-v2 [42] is an extension to *TinyVIRAT* where also the *MEVA* [6] dataset has been used.
Similarly to *TinyVIRAT* (section 4.1.2.1) this dataset is based on security videos. A key difference of this dataset with respect to the standard *TinyVIRAT* is that it has also indoor scenes which makes this problem more challenging.
TinyVIRAT-v2 has 16950 videos in train, 3308 videos in validation and 6097 videos in test split.
In table 4.1 you can see a statistics comparison from *TinyVIRAT*, *TinyVIRAT-v2* and several other popular datasets.

Dataset	*Resolution*	*ANF*	*ML*	*NC*	*Train*	*Val*	*Test*
UCF-101 [39]	320x240	186.50	No	101	9537	-	3783
HMDB-51 [20]	320x240	94.49	No	51	3570	1530	-
AVA [13]	264x440 to 360x640	127081.66	Yes	80	210,634	57,371	117,441
TinyVIRAT [8]	10x10 to 128x128	93.93	Yes	26	7663	-	5166
TinyVIRAT-v2 [42]	10x10 to 128x128	76.14	Yes	26	16950	3308	6097

Table 4.1: Dataset statistics [42]. In this table AFN: average number of frames, ML: multi-label and NC: number of classes.

For this challenge we have used only *TinyVIRAT-v2* as benchmark dataset, since it has also indoor scenes and it will generalize better.

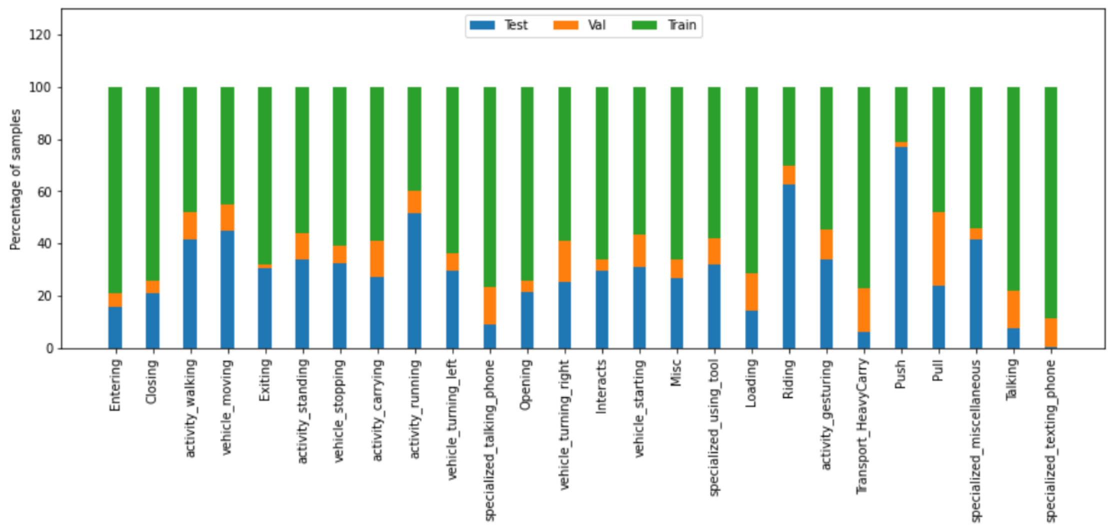


Figure 4.4: Number of samples per action class across the train, validation and test split of the *TinyVIRAT-v2* dataset [42].

In figure 4.4 [42] you can see the number of samples for each action class across the train, validation and test split. Instead, in figure 4.5 [42] you can see the class wise sample distribution by resolution. In table 4.2 you can see the number of videos for each class in training and validation set.

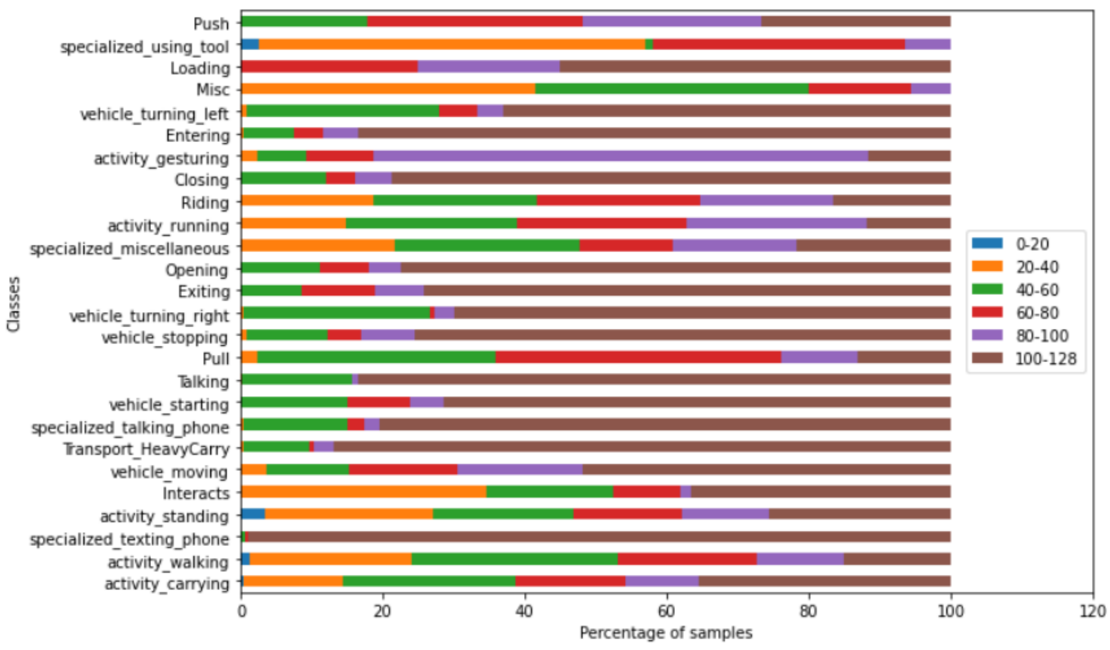


Figure 4.5: Class wise sample distribution by resolution in *TinyVIRAT-v2* dataset. Samples has been grouped into six groups based on their resolution (0-20, 20-40, 40-60, 60-80, 80-100, 100-128) for each class. [42].

4.1.3 Evaluation Metrics

This section aims at illustrating the three performance metrics that has been used to evaluate the different experiments that are proposed in this work. The three performance metrics are *Precision*, *Recall* and *F1-Score*.

4.1.3.1 Precision

The **Precision** of a model is a score that measures the proportion of positively predicted labels that are actually correct. It is also known as the *positive predictive value* and its minimizing the false negatives.
This type of score is affected by the class distribution. If there are more samples in the minority class, than precision will be lower.
The precision score is a useful measure of the success of prediction when the classes are very imbalanced. It represents the ratio of true positive (TP) to the sum of true positive (TP) and false positive (FP)

Class	**Training samples**	**Validation samples**
Opening	225	12
Interacts	501	29
Pull	85	50
activity carrying	2947	707
Entering	202	15
vehicle moving	689	154
Exiting	139	3
Loading	13	3
Talking	7072	1323
activity running	27	6
vehicle turning left	217	22
vehicle stopping	226	26
Riding	57	13
Closing	178	11
activity walking	3782	862
Push	12	1
specialized using tool	162	28
vehicle starting	166	36
specialized miscellaneous	13	1
activity standing	1552	279
Transport HeavyCarry	938	216
activity gesturing	24	5
vehicle turning right	152	42
specialized talking phone	668	123
specialized texting phone	3673	440
Misc	47	5

Table 4.2: Number of samples for each class in *TinyVIRAT-v2* training and validation set.

$$precision = \frac{TP}{TP + FP}$$

As you can see the number of false positive would impact the precision score.

4.1.3.2 Recall

Recall score represents the ability of the model to correctly predict the positives out of actual positives.
It is a score that is often paired with the *Precision* in order to do a trade-off between false positives and false negatives. With an high recall, you are minimizing the false positives.
Mathematically, it represents the ratio of true positives (TP) to the sum of true positive (TP) and false negative (FN)

$$recall = \frac{TP}{TP + FN}$$

The formula is very similar to *Precision*, but here the number of false negative would impact the recall score.

4.1.3.3 F1-Score

The **F1-Score** is a score that combines both *Precision* and *Recall* scores. It is a metric that gives equal weight to both the *Precision* and the *Recall*. It is used for optimizing both scores.
Mathematically, the F1-Score represents an harmonic mean of precision and recall score:

$$f1 = \frac{2 * precision * recall}{precision + recall}$$

Since this metric is a trade off between *precision* and *recall*, we would focus on optimizing this score.

4.1.4 Training setup

All the experiments carried out use the same configuration of training parameters. All numerical results are obtained training the models for 200 epochs. The setup used for all the training procedures is described hereafter:

- **Dataset**: the chosen dataset is the *TinyVIRAT-v2*;
- **Randomization**: the seed for all the random number generations is 1.
- **Initialization**: weights of the models are partially or completely initialized using pre-trained weights coming from other dataset. The weights that are not coming from a pre-trained model have been initialized randomly. In particular, for each model:
  - *ResNet*: pre-trained weights taken from *Kinetics-700* [38] both for depth 18 and 50 *ResNet*s.
  - *Multi-Scale ResNet*: only the weights of the backbone (*ResNet18* or *ResNet50*) are pre-trained from *Kinetics-700* [38].
  - *FPN ResNet*: only the weights of the backbone (*ResNet18* or *ResNet50*) are pre-trained from *Kinetics-700* [38].
  - *Multi-Scale FPN ResNet*: only the weights of the backbone of the *Multi-Scale ResNet* model (*ResNet18* or *ResNet50*) are pre-trained from *Kinetics-700* [38].
- **Weights updating**: the weight updating during the training process follows the *back-propagation* technique.
- **Learning Rate**: the learning rate starting value depends on the model. It is updated using a *Multi-Step Scheduler* where, at epoch 50, 100 and 150, it is divided by 10 (starting from 0.001, at epoch 50 it is changed to 0.0001, at epoch 100 it is changed to 0.00001 and at epoch 150 it becomes 0.000001). For *ResNet18*, *FPN ResNet18* and *Multi-Scale ResNet18* the learning rate starts from 0.001, instead for *FPN ResNet50* and for *Multi-Scale FPN ResNet18* it starts from 0.00001.

- **Optimizer**: the optimizer used in the experiments is *ADAM* [18] with a weight decay equal to 0.

- **Batch size**: the batch size depends on the model. For *ResNet18* we have a batch size of 96, instead for *Multi-Scale ResNet18*, *FPN ResNet18*, *FPN ResNet50* and *Multi-Scale FPN ResNet18* we have a batch size of 24.

- **Input sizes**: the temporal size (i.e., number of frames) per clip is always 16. The spatial size of the input clips depends on the model. In particular, for *ResNet18*, *FPN ResNet18* and *FPN ResNet50* we have 112×112 as input size, and for *Multi-Scale ResNet18* and *Multi-Scale FPN ResNet18* we have three different input sizes (because of the multi-scale approach) 14×14, 56×56 and 112×112.

4.2 Baseline

In order to compare the results of our studies we need to have a baseline score as point of reference.
To do this we have applied one recent state-of-the-art architecture, the $ResNet18$ model, on the action recognition *TinyVIRAT-v2* dataset.

Method	**F1-Score**	**Precision**	**Recall**
ResNet18	33.0 %	33.6 %	37.4 %

Table 4.3: Baseline scores calculated on the *TinyVIRAT-v2* dataset.

The results of the baseline model are shown in table 4.3, where we have evaluated the score of the standard $ResNet - 3D$ architecture with a depth of 18 and we have calculated F1-Score, Precision and Recall. On figure 4.6 you can see the scores during the training phase.

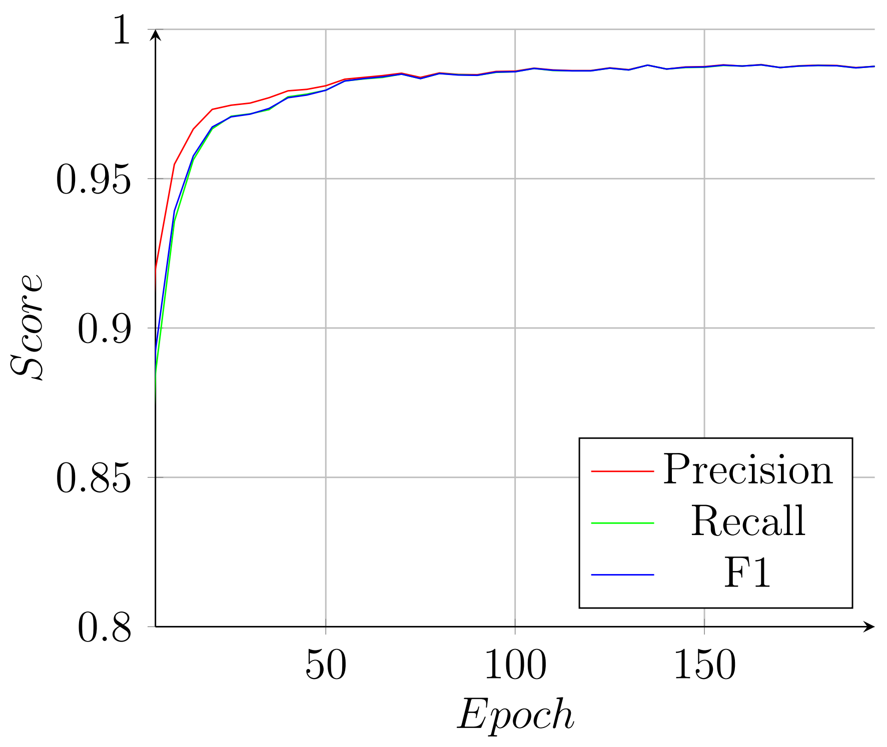


Figure 4.6: Training scores of the *ResNet18* baseline model during epochs.

4.3 Ablation studies

This section illustrates the ablation studies that has been done to evaluate the performances of our models with respect to the baselines. In each experiment we report also the score of the baseline, in order to compare the results that the experiment is exposing.

4.3.1 Model ablation

The first ablation study that this work proposes is the model ablation study. The idea of this experiment is to try different models in order to see the different performances.
In this experiment we compare the baseline score or the *ResNet18* model with the scores obtained by using a *Multi-Scale ResNet18* model, a *Feature Pyramid Network ResNet18* model and a *Multi-Scale FPN ResNet18* model.

The *Multi-Scale* architecture has been described in section 3.6. For this experiment we have used a *Multi-Scale* with a *ResNet18* as backbone. This model uses

Method	F1-Score	Precision	Recall
ResNet18	33.0 %	33.6 %	37.7 %
Multi-Scale ResNet18	33.6 %	33.8 %	38.3 %
FPN ResNet18	33.1 %	32.7 %	**40.4 %**
Multi-Scale FPN ResNet18	**34.9 %**	**34.6 %**	40.0 %

Table 4.4: Results for the action recognition task on the *TinyVIRAT-v2* dataset with different models. The *ResNet18* model is the baseline described in section 4.2, so it uses 112×112 as input size. The *Multi-Scale ResNet18* instead is using 14, 56 and 112 as input sizes. *FPN ResNet18* uses 112×112 and *Multi-Scale FPN ResNet18* is using again 14, 56 and 112 as input sizes.

14, 56 and 112 as input sizes.

The *Feature Pyramid Network* architecture has been described in section 3.7. As backbone, a *ResNet18* has been used and the input size is 112×112.

The last model, the *Multi-Scale Feature Pyramid Network* model, has been described in section 3.8. A *Multi-Scale ResNet18* model is used as backbone using 14, 56 and 112 as input sizes.

Results are reported in table 4.4. As you can see from the results, the model that is performing better is the *Multi-Scale FPN ResNet18* model, that obtains a +1.9% in the F1-Score, +1.0% in precision and +2.6% in the recall. However, the one that is performing better in the recall is the *FPN ResNet18*, obtaining a +0.1% in F1-Score, −0.9% in precision an +3% in the recall.

The *Multi-Scale ResNet18* is improving performances of +0.6% in F1-Score, +0.2% in precision an +0.9% in recall.

To analyze better the results we should consider the following things:

- the *Multi-Scale ResNet18* model is performing a little bit better of the baseline but the model complexity is ~ 3 times the complexity of the baseline (it uses three *ResNet18* instead of one);

- the *Multi-Scale Feature Pyramid Network ResNet* model is performing well, but it also increases the complexity of the model.

For these reasons this work will focus on the *Feature Pyramid Network* model. This model is very nice because it proposes a multi-scale model without duplicating the backbone. Hence, this model is giving very nice results without increasing a lot the model complexity.

4.3.2 Training procedure ablation

In this experiment we want to try different configuration for the training procedure in order to see if any changes can improve the performances of the model.
We compare the performances of the *ResNet18* baseline and of the *Feature Pyramid Network ResNet18* that we have seen in section 4.3.1 with the performance of a *Feature Pyramid Network* where we change some settings in the training procedure. The input size of the model is always 112×112.
We decide to change the training procedure in an incremental way. The changes are:

1. updating the strides of the network in order to work with low-resolution inputs, as described in 3.2;
2. changing the clip sampling method from the *random clip sampling* to *segment based clip sampling*, both described in section 3.1;
3. adding data augmentation (section 3.3);
4. applying *Horizontal Flip TTA* (Test Time Augmentation) described in section 3.4.1);
5. tuning the *positive-negative threshold*, exposed in section 3.5
6. using *5-clips TTA* (Test Time Augmentation) (section 3.4.2)

As you can see from table 4.5 the incremental changes are very effective and, from the initial setup of the standard *Feature Pyramid Network ResNet18*, we obtain a final +8.1% on F1-Score, +11.3% on precision and +3.8% on Recall.

The only training configuration that is not increasing the F1-Score is the one when we have changed the sampling method. However, this last training configuration is

Method	SM	AG	TTA	TH	F1	Precision	Recall
ResNet18	Random	No	No	No	33.0 %	33.6 %	37.7 %
FPN ResNet18	Random	No	No	No	33.1 %	32.7 %	40.4 %
FPN AD-ResNet18	Random	No	No	No	36.7 %	38.8 %	38.6 %
FPN AD-ResNet18	Segment	No	No	No	36.3 %	34.3 %	**46.0 %**
FPN AD-ResNet18	Segment	Yes	No	No	38.8 %	36.1 %	45.4 %
FPN AD-ResNet18	Segment	Yes	H.F.	No	40.6 %	40.0 %	45.5 %
FPN AD-ResNet18	Segment	Yes	H.F.	Yes	41.0 %	44.7 %	40.8 %
FPN AD-ResNet18	Segment	Yes	5-C.	Yes	**41.2 %**	**44.9 %**	41.2 %

Table 4.5: Results of the Training Procedure ablation study, a comparison between different training procedures on a *Feature Pyramid Network ResNet18* model with 112×112 as input size. In this table SM = Sampling Method, AG = Augmentation, TTA = Test Time Augmentation, H.F. = Horizontal Flip, 5-C = 5-Clips, TH = Positive-Negative Threshold Tuning.

increasing a lot the *Recall* obtaining the base score in this performance measure (+7.4% with respect to the previous configuration). Also, considering the plot in figure 4.7 that is showing the Validation F1-Score using *Random Clip Sampling* and *Segment Based Sampling*, we can see that the scores are very similar. For this reasons losing 0.4% on the F1-Score on the test set is worth for getting an incredible boost in the Recall score.

4.3.3 Using Attention

This ablation study will try to change the architecture in order to use *Attention* [46] in the model. For this purpose, in particular, we want to try to change the standard *ResNet18* architecture in order to use an *Attention Pooling Layer* instead of the *Average Pooling Layer* at the end of the model. The full procedure is described in section 3.9.

We compare the performances of the adapted *ResNet18* model (updated strides) using *Segment Based Sampling* and *Horizontal Flip TTA* with the adapted *ResNet18*

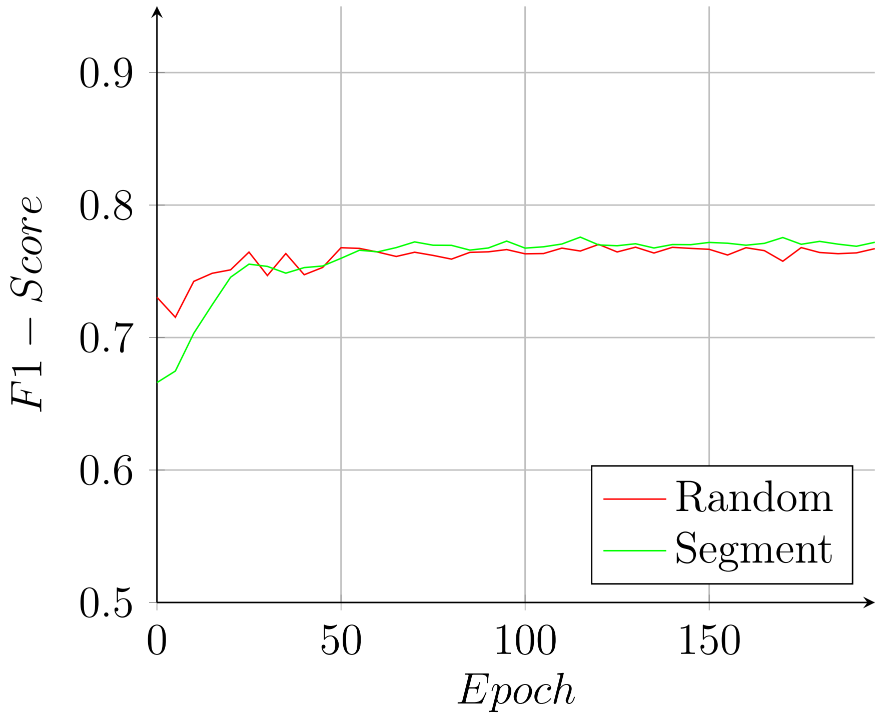


Figure 4.7: Validation F1-Score of the *FPN AD-ResNet18* with *Random Clip Sampling* and *Segment Based Clip Sampling*.

using the *Segment Based Sampling*, *Horizontal Flip TTA* and the *Attention Pooling Layer* instead of the *Average Pooling Layer*. For both models we have used an input size of 112 × 112.

Method	Pooling type	F1-Score	Precision	Recall
ResNet18	Average	33.0 %	33.6 %	37.7 %
AD-ResNet18	Average	**36.5 %**	**36.5** %	**40.7 %**
AD-ResNet18	Attention	34.3 %	33.2 %	38.7 %

Table 4.6: Comparison between using an Attention Pooling Layer instead standard Average Pooling Layer. *AD-ResNet18* stands for *Adapted ResNet18*, where you change the strides according to the description in section 3.2.

As you can see from the results in table 4.6, the results of the *Attention Pooling Layer* are not as good as using *Average Pooling Layer*, since with the *Attention Pooling Layer* we obtain a −2.2% on the F1-Score, −3.3% on the Precision and −2% on the Recall.

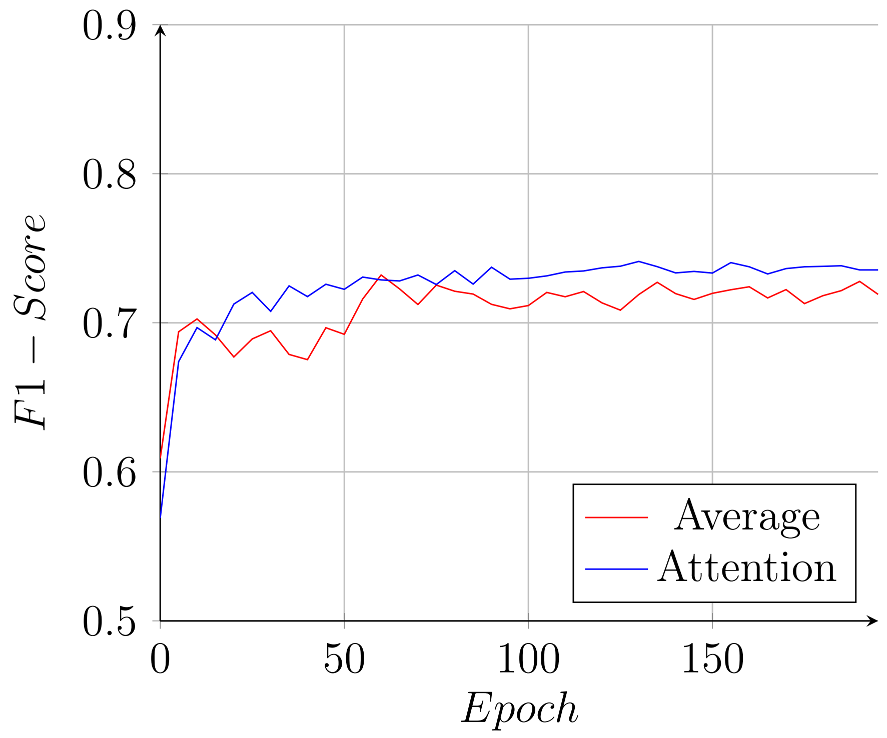


Figure 4.8: Validation F1-Score of the *AD-ResNet18* with *Average Pooling Layer* (red) and *Attention Pooling Layer* (blue).

The interesting thing to note is that by looking at the plot in figure 4.8 that is plotting the F1-Score during the validation phase, it seems that the *Attention Pooling Layer* is better than the *Average Pooling Layer*.
The same results can be seen by comparing two *Feature Pyramid AD-Network ResNet18*, using an adapted *ResNet18* as backbone and with *Augmentation* and **Horizontal Flip TTA**. In one model we use the adapted *ResNet18* with the *Average Pooling Layer* and in the other one we use the adapted *ResNet18* with the *Attention Pooling Layer*.

In fact, table 4.7 is showing that the *Average Pooling Layer* is still performing a little bit better than the *Attention Pooling Layer* on F1-Score and Precision (+1.8% on F1-Score, +6.2% on the Precision and −6.1% on the Recall).

Method	Pooling type	F1-Score	Precision	Recall
ResNet18	Average	33.0 %	33.6 %	37.7 %
FPN AD-ResNet18	Average	**41.0 %**	**44.7** %	40.8 %
FPN AD-ResNet18	Attention	39.2 %	38.5 %	**46.9 %**

Table 4.7: Comparison between using an Attention Pooling Layer instead standard Average Pooling Layer on a *Feature Pyramid Network* using an adapted *ResNet18* as backbone.

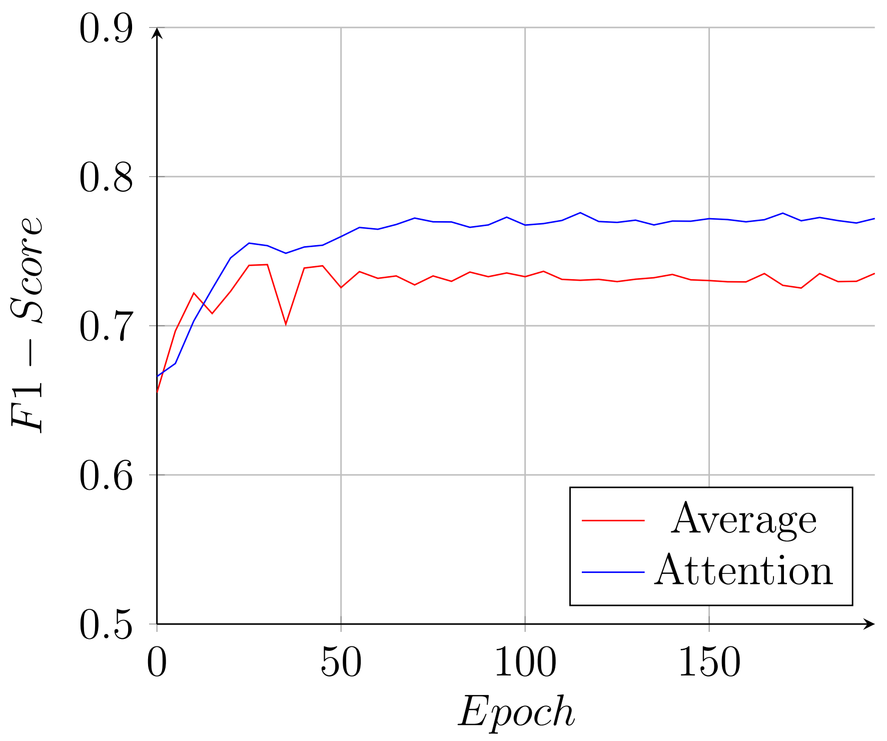


Figure 4.9: Validation F1-Score of the *FPN AD-ResNet18* with *Average Pooling Layer* (red) and *Attention Pooling Layer* (blue) in the *ResNet18* backbone.

If we look at the plot in figure 4.9 that is showing the validation F1-Score of the *FPN AD-ResNet18* with *Attention Pooling Layer* and *Average Pooling Layer* during the training we can see that it seems, again, that the *Attention Pooling Layer* is better.

Method	**Discriminative filter**	**F1-Score**	**Precision**	**Recall**
ResNet18	No	<u>33.0 %</u>	<u>33.6 %</u>	<u>37.7 %</u>
AD-ResNet18	No	**36.6 %**	**36.3 %**	40.3 %
AD-ResNet18	Yes	36.1 %	35.0 %	**40.8 %**

Table 4.8: Comparison between using Spatial-temporal discriminative filter [24] instead of classic *Fully Connected Layer* on *AD-ResNet18* model.

Method	**Discriminative filter**	**F1-Score**	**Precision**	**Recall**
ResNet18	No	<u>33.0 %</u>	<u>33.6 %</u>	<u>37.7 %</u>
FPN AD-ResNet18	No	**41.0 %**	**44.7 %**	40.8 %
FPN AD-ResNet18	Yes	37.7 %	36.5 %	**42.3 %**

Table 4.9: Comparison between using Spatial-temporal discriminative filter [24] instead of classic classic *Fully Connected Layer* on *FPN AD-ResNet18* model.

4.3.4 Applying Spatial-temporal discriminative filters

This experiment will study if there is any benefit by using spatial-temporal discriminative filters [24], that are described in section 3.10.
The idea is changing the classifier part of the network in order to detect finer actions.
To do this experiment we compare the adapted *ResNet18* model with and without the spatial-temporal discriminative filters, using *Augmentation* and *Horizontal Flip TTA*. We also try this method on the *Feature Pyramid Network*.

As you can see from the results reported in table 4.8, the *AD-ResNet* model without the discriminative filter is performing better in the F1-Score (+0.5%) and on the Recall (+1.3%), instead is performing worse in the Recall (−0.5%).
Also here it is interesting to see that in plot 4.10 the validation F1-Score of the *AD-ResNet18* with the *Spatial-Temporal Discriminative Filters* is higher than the one with *Fully Connected Layer*.
The results are similar on the *FPN AD-ResNet18*, using again an adapted *ResNet18* as backbone, *Augmentation* and *Horizontal Flip TTA*.

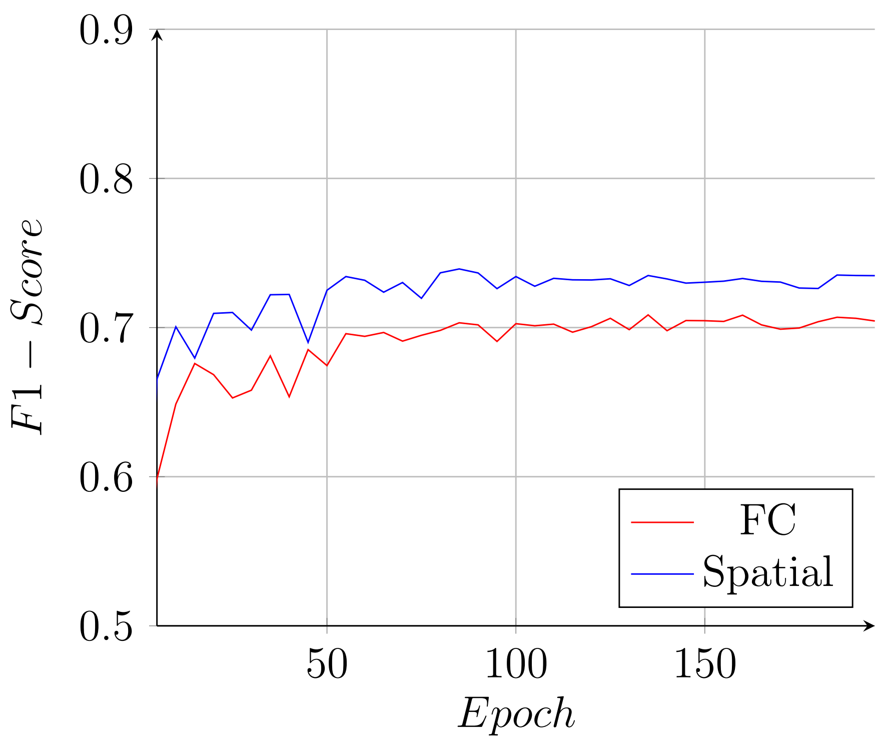


Figure 4.10: Validation F1-Score of the *AD-ResNet18* with *Fully Connected Layer (FC)* (red) and *Spatial-Temporal Discriminative Filters* (blue) in the classification stage of the *AD-ResNet18* model.

In fact, table 4.9 shows that *Fully Connected Layer* is better than *Spatial-Temporal Discriminative Filters*, obtaining +3.3% on F1-Score, +8.2% on Precision and −1.5% on Recall.

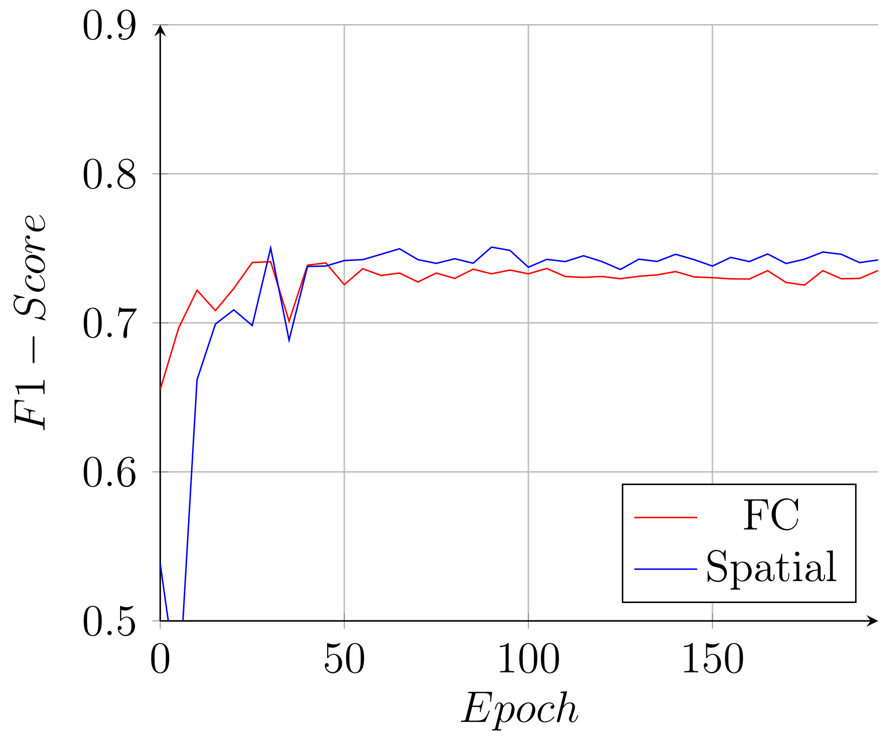


Figure 4.11: Validation F1-Score of the *FPN AD-ResNet18* with *Fully Connected Layer (FC)* (red) and *Spatial-Temporal Discriminative Filters* (blue) in the classification stage of the *AD-ResNet18* model.

Looking at plot 4.11 we can see that, similarly to the *AD-ResNet18* model, the *Spatial-Temporal Discriminative Filters* are a little better than the *Fully Connected Layer* on the validation set.

4.3.5 Loss Function ablation

The idea of this experiment is to see if changing the loss function is going to increase the performances.
In order to do this we compare the performance of two *FPN AD-ResNet18* with the same training configuration, changing only the Loss Function. We use 112×112 as input size, we have used an adapted *ResNet18* in order to work with smaller inputs, we have used *Augmentation*, *Horizontal Flip TTA* and *Positive-Negative Threshold Tuning*.

Method	Loss function	F1-Score	Precision	Recall
ResNet18	Binary cross-entropy	33.0 %	33.6 %	37.7 %
FPN AD-ResNet50	Binary cross-entropy	**42.2 %**	42.6 %	**45.7 %**
FPN AD-ResNet50	Asymmetric Loss	40.5 %	**52.2 %**	35.3 %

Table 4.10: Performances obtained using Binary Cross-Entropy and Asymmetric Loss in *FPN AD-ResNet50* model.

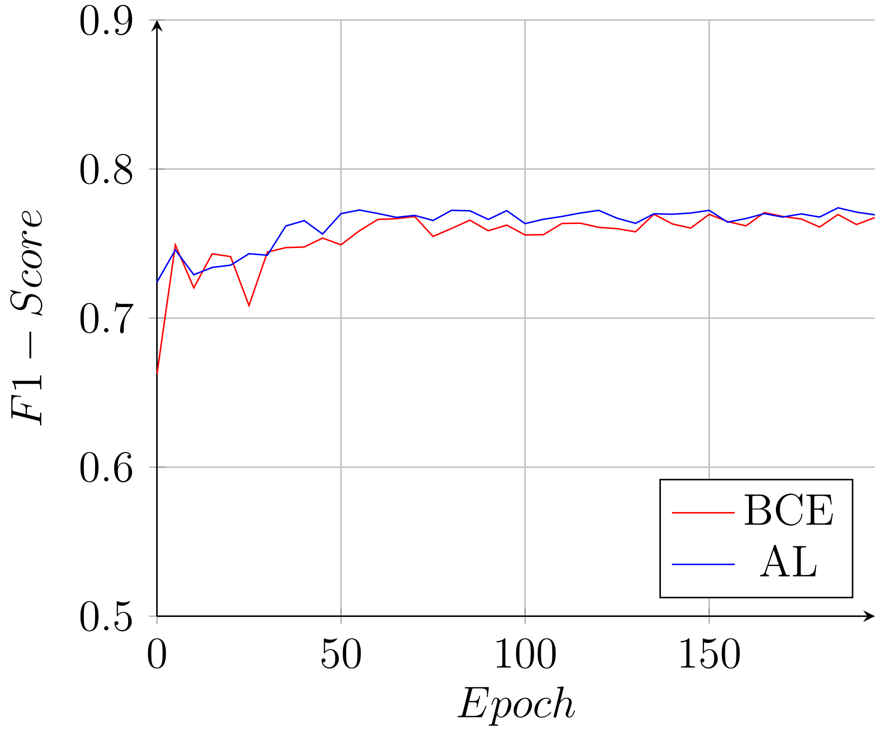


Figure 4.12: Validation F1-Score of the *FPN AD-ResNet50* with *Binary Cross-Entropy (BCE)* (red) and *Asymmetric Loss (AL)* (blue) as loss function.

Table 4.10 is reporting the results. As you can see, one is using standard *Binary Cross-Entropy* and the other one is using *Asymmetric Loss*, both described in section 3.11. The *Binary Cross-Entropy* is performing better on F1-Score, with an increment of 1.7%, and on Recall, with a +10.4%, but worse on Precision (−9.6%). On plot 4.12 we can see that actually these two loss function, on Validation set, are very similar and there is not a predominant loss function.

It is interesting to see this experiment done to *Multi-Scale FPN ResNet18*

Method	Loss function	F1-Score	Precision	Recall
ResNet18	Binary cross-entropy	33.0 %	33.6 %	37.7 %
Multi-Scale FPN ResNet18	Binary cross-entropy	34.9 %	34.6 %	**40.0 %**
Multi-Scale FPN ResNet18	Asymmetric Loss	**36.2 %**	**40.9 %**	34.9 %

Table 4.11: Performances obtained using Binary Cross-Entropy and Asymmetric Loss in *Multi-Scale FPN ResNet18* model.

model. We have done the same experiment using 14, 56 and 112 as input sizes, using standard *ResNet18* as backbone and without using any data augmentation technique or test time augmentation.

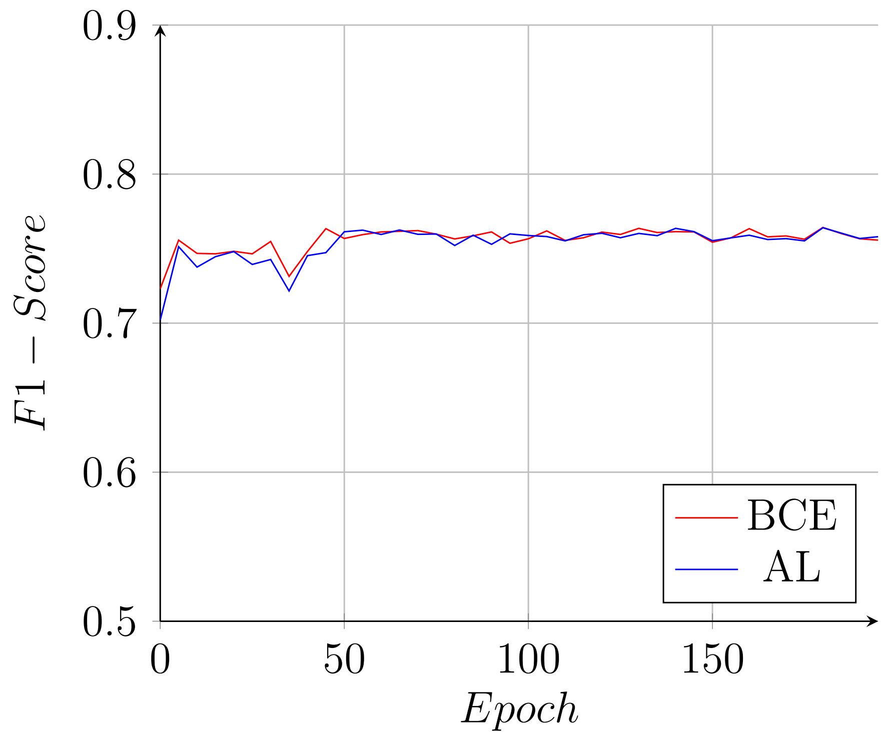


Figure 4.13: Validation F1-Score of the *Multi-Scale FPN ResNet18* with *Binary Cross-Entropy (BCE)* (red) and *Asymmetric Loss (AL)* (blue) as loss function.

Table 4.11 is showing results obtained using this model. As you can see, in this experiment, *Asymmetric Loss* is performing better than *Binary Cross-Entropy* on F1-Score (+1.3%) and Precision (+6.3%), but it loses on Recall (−5.1%). Similarly to the *FPN ResNet50*, plot 4.13 is showing that there is not a predominant Loss

Function on the validation set.

4.4 Results on TinyVIRAT-v2 action recognition benchmark

This section illustrates the improvement over the standard *ResNet18* model baseline evaluating on the *TinyVIRAT-v2* action recognition benchmark.
The best score that we have obtained is by using a *Feature Pyramid Network* with and adapted *ResNet50* as backbone. We have used the *Segment Based Sampling*, *Augmentation* and *Horizontal Flip TTA* using a *Positive-Negative threshold* of 0.35.

Method	**F1-Score**	**Precision**	**Recall**
ResNet18	33.0 %	33.6 %	37.7 %
FPN AD-ResNet50	**42.2 %**	**42.6 %**	**45.7 %**
Improvement	+9.2 % $\uparrow$	+9 % $\uparrow$	+8.3 % $\uparrow$

Table 4.12: Scores obtained on *TinyVIRAT-v2* action recognition benchmark with *FPN AD-ResNet50* model.

The table 4.12 is showing that the *FPN AD-ResNet50* model improves the standard *ResNet18* model baseline on all the performance metrics, obtaining a +9.2% on F1-Score, +9% on the Precision and +8.3% on the Recall.

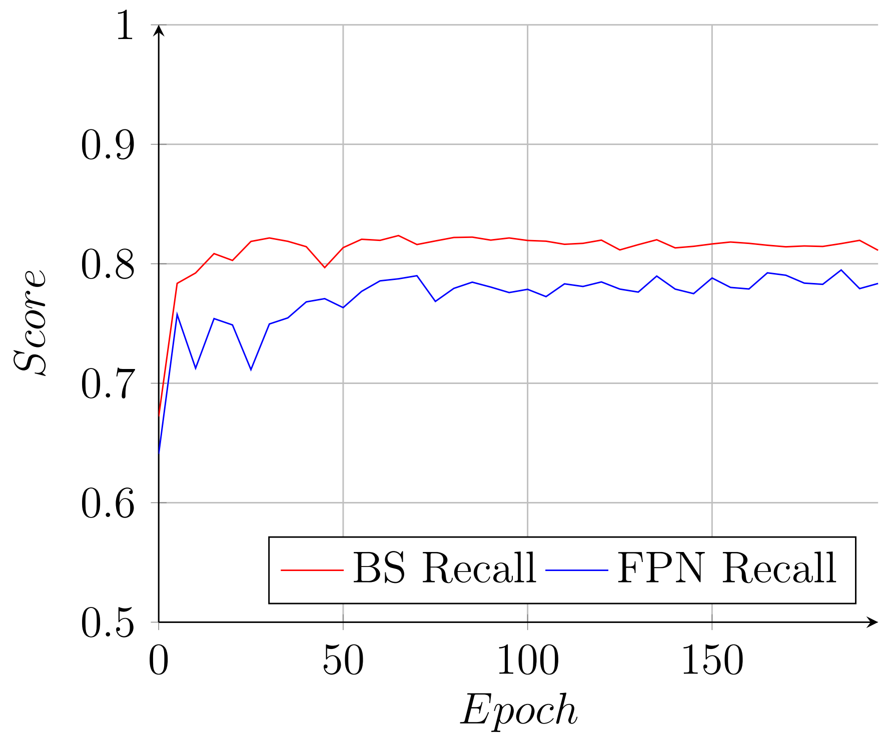


Figure 4.14: Validation Recall score of the *Baseline* model (red) and *FPN AD-ResNet50* (blue).

It is interesting to see, from plot 4.14, that the Recall of the baseline model on validation set is better than the one obtained by the *FPN AD-ResNet50* but, on the test set, *FPN AD-ResNet50* outperforms the baseline by +8.3%.

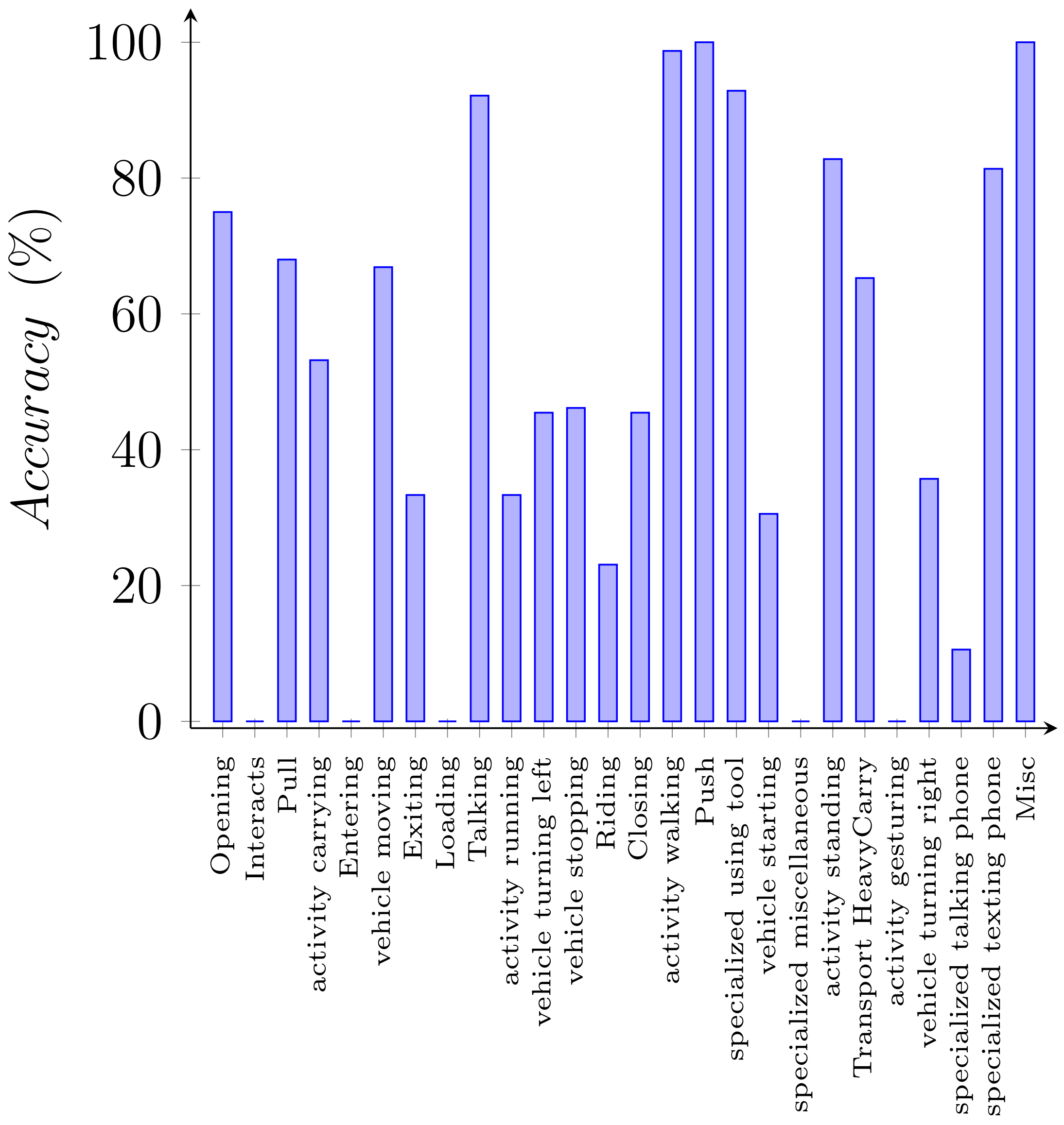


Figure 4.15: Validation per-class accuracy analysis on the *FPN AD-ResNet50* model.

Plot 4.15 is showing the classes of actions that are classified correctly and incorrectly. Plot is revealing that exists classes that are never recognized correctly and classes that are recognized correctly most of the time. In particular, by comparing this plot with table 4.2 that is showing the number of samples for each class in the training and validation set, you can see that the classes that are never recognized correctly (like action *Loading*) has a small number of elements in training and validation set (in this case 13 in the training and 3 in the validation).

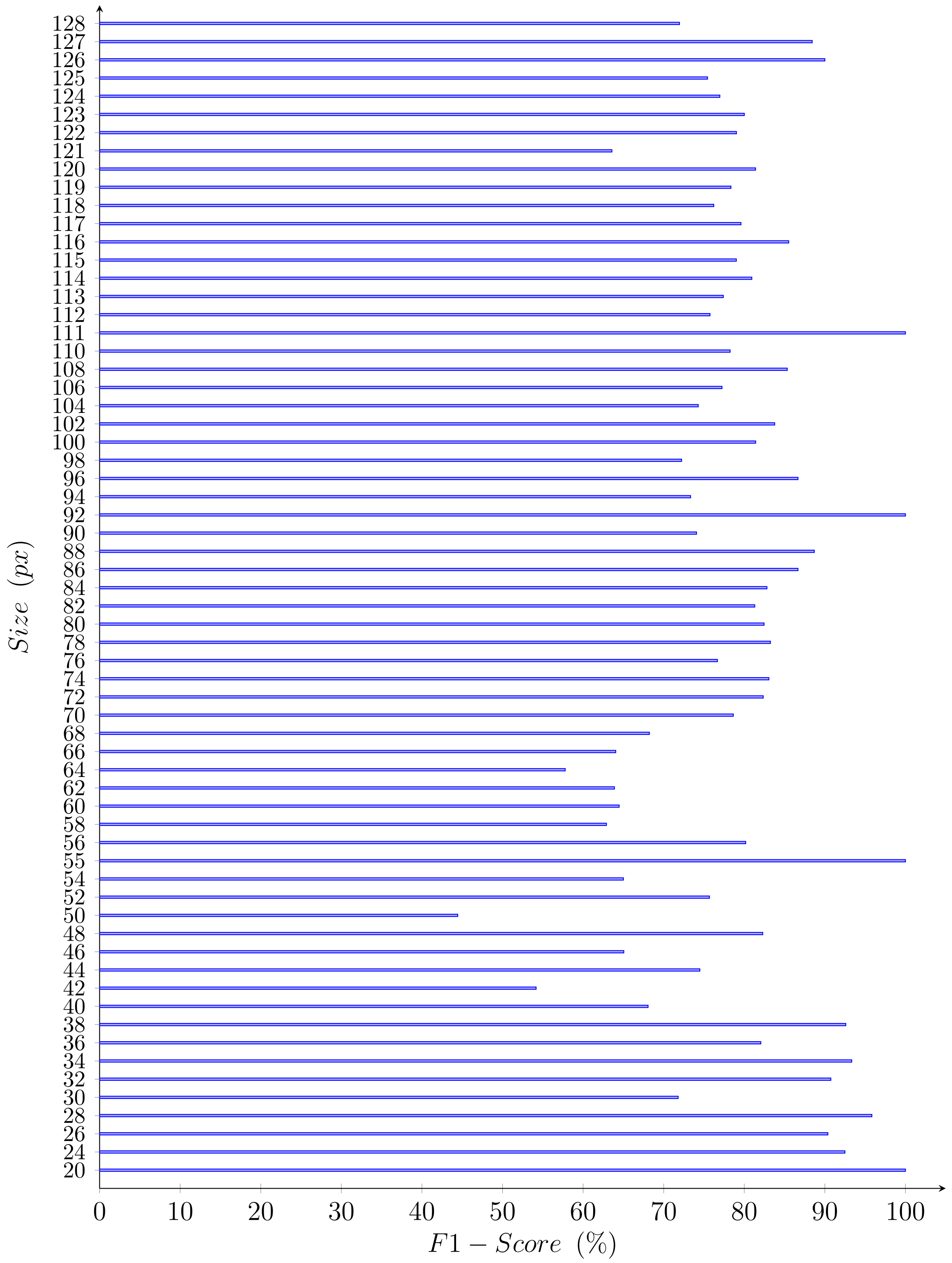


Figure 4.16: F1-Score for each video size in the validation set using the *FPN AD-ResNet50* model.

Instead plot 4.16 is showing F1-Score for each original clip size of our dataset.

As you can see the multi-scale mechanism (top-down pathway) of the *FPN AD-ResNet50* model is working pretty well since we obtain good F1-Scores in almost all the different input sizes, going from the smaller to the larger ones.

Chapter 5

Conclusions

This thesis focuses on applying the *action recognition* task on *low-resolution videos*. Our approach has led to trying various solutions, changing both the model to be trained and the way in which we pre and post process data. This particular type of task plays a key role in research, since the current state of the art does not consider these types of datasets, in which the videos are very short and have a very low spatial resolution. These datasets, such as the *TinyVIRAT*, allows people to train models in cases where these constraints are very common, such as in the context of video surveillance.

In this work we have seen how pre and post-process of the data matters a lot in the model training phase on low-resolution data. In fact, simply by changing the *random clip sampling* to the *segment based clip sampling* is going to boost up to +7% on the *Recall*. And also using *Test Time Augmentation (TTA)* techniques led up to a +1.8% on *F1-Score*, +3.9% on the *Precision* and +1% on the *Recall*.

Then, since the standard *ResNet-3D* architecture works with a spatial dimension of 224×224, we have proposed an modification of the architecture in order to work with inputs of 14×14, 56×56 and 112×112, since in the majority of the case datasets has smaller clips.

After that, we have proposed three different *Multi-Scale* architectures, the *Multi-Scale ResNet*, *FPN ResNet* and *Multi-Scale FPN ResNet*, in order to handle the problem of choosing the correct spatial size for the clips. In fact, selecting a size that is too small will lead to the loss of too much detail for the larger clips when

we go to reduce their size. Instead, by selecting a larger size we will get that the smaller clips, when enlarged, will be very grainy. Using this kind of model has increased the scores obtained by the proposed state-of-the-art solution up to +1.9% on the *F1-Score*, +1% on the *Precision* and +2.7% on the *Recall*.
In the end we were able to increase the results of the *ResNet18* baseline model by +9.2% in the *F1-Score*, +9% in the *Precision* and by +8.3% in the *Recall* (as reported in table 4.12) by using an *FPN AD-ResNet50*.

5.1 Future Works

One idea for a future work is to use more *low-resolution* benchmark datasets in order to have data coming from different sources and also to increase the amount of data. Comparing the performances obtained on the *TinyVIRAT* dataset with performances obtained on other low-resolution dataset is going to give us a generalized measure on the quality of our results, also considering that *TinyVIRAT* dataset has a shift between validation and test split (a model that obtain like 0.75 *F1-Score* on the validation set is getting like 0.30/0.35 *F1-Score* on the test set).
It is also interesting to see if using other *backbone* model, different from the *ResNet* model, is going to boost the performances. Also, the idea of using some *model ensemble* methods is a future work, since this method could be used in order to classify correctly all the classes of actions that, at the moment, our final result is not classifying correctly (section 4.4, plot 4.15). Another possible future work that can be done is to use the *Video Super-Resolution*, that is a different approach to handle *low-resolution* tasks used to transform a low-quality video to an high-resolution video and handle it as a common *high-resolution action recognition* task.

References

[1] Sami Abu-El-Haija et al. *YouTube-8M: A Large-Scale Video Classification Benchmark*. 2016. DOI: 10.48550/ARXIV.1609.08675. URL: https://arxiv.org/abs/1609.08675 (cit. on p. 2).

[2] Emanuel Ben-Baruch et al. *Asymmetric Loss For Multi-Label Classification*. 2021. arXiv: 2009.14119 [cs.CV] (cit. on p. 32).

[3] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2017. DOI: 10.48550/ARXIV.1705.07750. URL: https://arxiv.org/abs/1705.07750 (cit. on pp. 2, 11).

[4] Kelvin C. K. Chan et al. *BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond*. 2020. DOI: 10.48550/ARXIV.2012.02181. URL: https://arxiv.org/abs/2012.02181 (cit. on p. 14).

[5] Mengyu Chu et al. “Learning temporal coherence via self-supervision for GAN-based video generation”. In: *ACM Transactions on Graphics* 39.4 (Aug. 2020). DOI: 10.1145/3386569.3392457. URL: https://dl.acm.org/doi/10.1145/3386569.3392457 (cit. on p. 14).

[6] Kellie Corona et al. *MEVA: A Large-Scale Multiview, Multimodal Video Dataset for Activity Detection*. 2020. DOI: 10.48550/ARXIV.2012.00914. URL: https://arxiv.org/abs/2012.00914 (cit. on p. 39).

[7] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. 2019. DOI: 10.48550/ARXIV.1909.13719. URL: https://arxiv.org/abs/1909.13719 (cit. on p. 23).

[8] Ugur Demir, Yogesh S Rawat, and Mubarak Shah. *TinyVIRAT: Low-resolution Video Action Recognition*. arXiv, 2020. DOI: 10.48550/ARXIV.2007.07355. URL: https://arxiv.org/abs/2007.07355 (cit. on pp. 2, 35, 38, 39).

[9] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 7).

[10] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on p. 12).

[11] W. E and J. Lu. “Multiscale modeling”. In: *Scholarpedia* 6.10 (2011). revision #91540, p. 11527. DOI: 10.4249/scholarpedia.11527 (cit. on p. 14).

[12] Haoqi Fan et al. *Multiscale Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2104.11227. URL: https://arxiv.org/abs/2104.11227 (cit. on p. 16).

[13] Chunhui Gu et al. *AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions*. 2017. DOI: 10.48550/ARXIV.1705.08421. URL: https://arxiv.org/abs/1705.08421 (cit. on pp. 2, 39).

[14] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. *Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?* 2017. DOI: 10.48550/ARXIV.1711.09577. URL: https://arxiv.org/abs/1711.09577 (cit. on pp. 2, 7, 20).

[15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 7).

[16] Shuiwang Ji et al. “3D Convolutional Neural Networks for Human Action Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013), pp. 221–231. DOI: 10.1109/TPAMI.2012.59 (cit. on p. 7).

[17] Andrej Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732. DOI: 10.1109/CVPR.2014.223 (cit. on p. 6).

[18] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980 (cit. on p. 45).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386 (cit. on p. 6).

[20] H. Kuehne et al. “HMDB: A large video database for human motion recognition”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543 (cit. on p. 39).

[21] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. DOI: 10.48550/ARXIV.1612.03144. URL: https://arxiv.org/abs/1612.03144 (cit. on p. 15).

[22] Hongying Liu et al. *Video Super Resolution Based on Deep Learning: A Comprehensive Survey*. 2020. DOI: 10.48550/ARXIV.2007.12928. URL: https://arxiv.org/abs/2007.12928 (cit. on p. 13).

[23] Yunbo Peng Liu Cen and Yue Lin. *Along*. 2021 (cit. on pp. 13, 14).

[24] Brais Martinez et al. *Action recognition with spatial-temporal discriminative filter banks*. 2019. DOI: 10.48550/ARXIV.1908.07625. URL: https://arxiv.org/abs/1908.07625 (cit. on pp. 18, 30, 31, 53).

[25] Rajat Modi et al. *Video Action Detection: Analysing Limitations and Challenges*. arXiv, 2022. DOI: 10.48550/ARXIV.2204.07892. URL: https://arxiv.org/abs/2204.07892.

[26] Mathew Monfort et al. *Moments in Time Dataset: one million videos for event understanding*. 2018. DOI: 10.48550/ARXIV.1801.03150. URL: https://arxiv.org/abs/1801.03150 (cit. on p. 2).

[27] Sangmin Oh et al. "A large-scale benchmark dataset for event recognition in surveillance video". In: *CVPR 2011*. 2011, pp. 3153–3160. DOI: 10.1109/CVPR.2011.5995586 (cit. on p. 38).

[28] Grace C. Y. Peng et al. *Multiscale modeling meets machine learning: What can we learn?* 2019. DOI: 10.48550/ARXIV.1911.11958. URL: https://arxiv.org/abs/1911.11958 (cit. on p. 14).

[29] Xiaojiang Peng et al. "Action Recognition with Stacked Fisher Vectors". In: *ECCV*. 2014 (cit. on p. 6).

[30] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. DOI: 10.48550/ARXIV.1712.04621. URL: https://arxiv.org/abs/1712.04621 (cit. on p. 21).

[31] *Python*. URL: https://www.python.org/ (cit. on p. 36).

[32] *PyTorch*. URL: https://www.pytorch.org/ (cit. on p. 37).

[33] *PyTorchVideo*. URL: https://www.pytorchvideo.org/ (cit. on pp. 37, 38).

[34] Zhaofan Qiu, Ting Yao, and Tao Mei. *Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks*. 2017. DOI: 10.48550/ARXIV.1711.10305. URL: https://arxiv.org/abs/1711.10305 (cit. on p. 10).

[35] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://arxiv.org/abs/1506.01497 (cit. on p. 15).

[36] Connor Shorten and Taghi Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (July 2019). DOI: 10.1186/s40537-019-0197-0 (cit. on p. 21).

[37] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556 (cit. on p. 7).

[38] Lucas Smaira et al. *A Short Note on the Kinetics-700-2020 Human Action Dataset*. 2020. DOI: 10.48550/ARXIV.2010.10864. URL: https://arxiv.org/abs/2010.10864 (cit. on pp. 2, 44).

[39] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV] (cit. on pp. 2, 39).

[40] Zehua Sun et al. *Human Action Recognition from Various Data Modalities: A Review*. 2020. DOI: 10.48550/ARXIV.2012.11866. URL: https://arxiv.org/abs/2012.11866 (cit. on p. 1).

[41] Jinbao Wang Teng Wang Tiantian Geng and Feng Zheng. *Sustech&hku submission to tinyaction challenge 2021*. 2021 (cit. on p. 32).

[42] Praveen Tirupattur et al. *TinyAction Challenge: Recognizing Real-world Low-resolution Activities in Videos*. arXiv, 2021. DOI: 10.48550/ARXIV.2107.11494. URL: https://arxiv.org/abs/2107.11494 (cit. on pp. 2, 38–41).

[43] Du Tran et al. *A Closer Look at Spatiotemporal Convolutions for Action Recognition*. 2017. DOI: 10.48550/ARXIV.1711.11248. URL: https://arxiv.org/abs/1711.11248 (cit. on pp. 2, 10).

[44] Du Tran et al. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2014. DOI: 10.48550/ARXIV.1412.0767. URL: https://arxiv.org/abs/1412.0767 (cit. on pp. 2, 7).

[45] Elahe Vahdani and Yingli Tian. *Deep Learning-based Action Detection in Untrimmed Videos: A Survey*. 2021. DOI: 10.48550/ARXIV.2110.00111. URL: https://arxiv.org/abs/2110.00111 (cit. on p. 12).

[46] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (cit. on pp. 12, 16, 30, 49).

[47] Heng Wang et al. “Action recognition by dense trajectories”. In: *CVPR 2011*. 2011, pp. 3169–3176. DOI: 10.1109/CVPR.2011.5995407 (cit. on p. 6).

[48] Longguang Wang et al. *Deep Video Super-Resolution using HR Optical Flow Estimation*. 2020. DOI: 10.48550/ARXIV.2001.02129. URL: https://arxiv.org/abs/2001.02129 (cit. on p. 14).

[49] Li Yao et al. *Describing Videos by Exploiting Temporal Structure*. 2015. DOI: 10.48550/ARXIV.1502.08029. URL: https://arxiv.org/abs/1502.08029 (cit. on p. 7).

[50] Yi Zhu et al. *A Comprehensive Study of Deep Video Action Recognition*. 2020. DOI: 10.48550/ARXIV.2012.06567. URL: https://arxiv.org/abs/2012.06567 (cit. on pp. 1, 6, 7).

Acknowledgements

Prima di tutto, desidero ringraziare il professore Lamberto Ballan, nonchè relatore di questa tesi, per avermi dato l'opportunità di svolgere questo lavoro. Inoltre desidero ringraziare i miei correlatori, Guglielmo ed Elena, per avermi sempre assistito e supportato in questo progetto, non avrei mai raggiunto questi risultati senza di voi.

Ringrazio inoltre tutta la mia famiglia, genitori, fratello, sorella, nonni e parenti per avermi sempre appoggiato nel mio percorso ed in questi anni di studio.

Desidero ringraziare anche tutti i miei amici, per le uscite in compagnia e per tutti i momenti di svago. Grazie per aver reso questo percorso più leggero e spensierato.

Ringrazio inoltre tutti i miei colleghi di lavoro che mi hanno permesso di svolgere questi studi in parallelo al percorso lavorativo e per l'appoggio che hanno sempre saputo darmi, cercando sempre di agevolare il più possibile il tutto.

Inoltre un grazie speciale va a te, Giada, che durante questo percorso non mi hai mai fatto mancare tutto il tuo supporto ed il tuo incoraggiamento. Hai sempre creduto in me dall'inizio, mi hai sempre aiutato e spero di renderti felice tanto quanto hai reso felice me da quando fai parte della mia vita.