International Journal of Computer Vision and Image Processing
Volume 9 • Issue 2 • April-June 2019

Accelerating Deep Action Recognition Networks for Real-Time Applications

David Ivorra-Piqueres, University of Alicante, Alicante, Spain
John Alejandro Castro Vargas, University of Alicante, Alicante, Spain
Pablo Martinez-Gonzalez, University of Alicante, Alicante, Spain

ABSTRACT

In this work, the authors propose several techniques for accelerating a modern action recognition pipeline. This article reviewed several recent and popular action recognition works and selected two of them as part of the tools used for improving the aforementioned acceleration. Specifically, temporal segment networks (TSN), a convolutional neural network (CNN) framework that makes use of a small number of video frames for obtaining robust predictions which have allowed to win the first place in the 2016 ActivityNet challenge, and MotionNet, a convolutional-transposed CNN that is capable of inferring optical flow RGB frames. Together with the last proposal, this article integrated a new software for decoding videos that takes advantage of NVIDIA GPUs. This article shows a proof of concept for this approach by training the RGB stream of the TSN network in videos loaded with NVIDIA Video Loader (NVVL) of a subset of daily actions from the University of Central Florida 101 dataset.

KEYWORDS

Action Recognition, Action Understanding, Deep Learning, GPU Acceleration, Machine Learning, Optical Flow, Real-Time, Recurrent Networks, Video Decoding

1. INTRODUCTION

Although in recent years the task of activity recognition has witnessed numerous breakthroughs thanks to the development of new methodologies and the rebirth of deep learning techniques, the natural course of events has not always been like this. As for many years, despite of being tackled from multiple perspectives, the problem of constructing a system that is capable of identifying which activity is being performed in a given scene has been barely solved. In the state of the art we can find different approaches based on handcrafted traditional methods and machine learning approaches:

- **Handcrafted features dominance**. The first approximations were motivated by fundamental algorithms such as optical flow (Horn and Rhunck, 1981), the Canny edge detector (Canny, 1986), Hidden Markov Model (HMM) (Rabiner and Juang, 1986) or Dynamic Time Warping (DTW) (Bellman and Kalaba,1959). Several of these methods have been reviewed in (Gavrila, 1999), for hand and the whole-body movements, which can be used to obtain relevant information for the recognition of activities.
- **Machine learning approaches**. More modern methods use optical flow (Efros et al., 2003) to obtain temporal features over the sequences, in addition to using automatic learning algorithms

DOI: 10.4018/IJCVIP.2019040102

such as Support Vector Machine (SVM) (Schüldt, Laptev and Caputo, 2004) to classify spatiotemporal features.

- **Deep learning**. The CNN networks allow to obtain robust visual features on 2D images (Chéron and Laptev, 2015), but more specifically its version adapted to work with data defined in three dimensions offers the ability to obtain spatial and temporal features when working with sequences of images. In this way, furthermore of two spatial dimensions (height and width), we have a third dimension defined by time (frames) (Ji et al., 2013) (Simonyan and Zisserman, 2014).

2. APPROACH

In this section we review the most modern action recognition works carried out in the past three years.

Online Inverse Reinforcement Learning (Rhinehart and Kitani, 2017) is a novel method for predicting future behaviors by modeling the interactions between the subject, objects, and their environment, through a first-person mounted camera. The system makes use of online inverse reinforcement learning. Thus, making it possible to continually discover new long-term goals and relationships. Also, a similar approach to that of the hybrid Siamese networks, has been shown (Mahmud, Hasan and Roy-Chowdhury, 2017) that is possible to simultaneously predict future activity labels and their starting time. It does so by taking advantage of features of previously seen activities and currently present objects in the scene.

Thanks to the use of Single Shot multi-box Detectors (SSDs) CNNs, the system proposed in (Singh et al., 2017) is capable of predicting both action labels, and their corresponding bounding boxes in real-time (28FPS). Moreover, it can detect more than one action at the same time. All of this is accomplished by using RGB image features combined with optical flow ones (with a decrease in the optical flow quality and global accuracy) extracted in real-time for the creation of multiple action tubes.

In (Kong, Tao and Fu, 2017), for predicting action class labels before the action finishes, authors make use of features extracted from fully observed videos processed at train time, for filling out the missing information present in the incomplete videos to predict. Furthermore, thanks to this approach their model obtains a great speedup improvement when compared to similar methods.

A model that is capable of performing visual forecasting at different abstraction levels is presented in (Zeng et al., 2017). For example, the same model can be trained for future frame generation as well as for action anticipation. This is accomplished by following an inverse reinforcement learning approach. Also, the model is enforced to imitate natural visual sequences from pixel level.

The model developed in (René et al., 2017) is capable of predicting in real-time future activities labels on RGB-D videos. This is accomplished by making use of soft regression, for jointly learning both the predictor model and the soft labels. Moreover, real-time performance (around 40FPS) is obtained by including a novel RGB-D feature named Local Accumulative Frame Feature (LAFF). Moreover, a TCN Encoder-Decoder system is built for performing the mentioned tasks. After training, it is able to surpass current similar approaches. Furthermore, the system presents a better performance than Bidirectional Long short-term memory networks (Bi-LSTMs) networks.

In (Buch et al., 2017), a system that is capable of presenting temporal action proposals on a video with only one forward pass is presented. Thus, there is no need to create overlapped temporal sliding windows. Moreover, the system can work with long untrimmed videos of arbitrary length in a continued fashion. Finally, by combining the system with action classifiers, temporal action detection performance is increased.

A new convolutional model is present in (Carreira and Zisserman, 2017), known as Two-Stream Inflated 3D convolutional neural network (I3D), which is used as a spatio-temporal feature extractor. After this, authors pre-train I3D based models on the Kinetics dataset, showing that with this approach, action classification performance on well-known datasets is noticeably increased.

In (Feichtenhofer, Pinz and Wildes, 2017), a fully space-time convolutional two-stream network (named STResNet) is proposed for the task of action recognition in videos. The first stream is fed with RGB data while the second, with optical flow features. The main particularity of this model is the existing interconnections between both streams. Moreover, for learning long-term relationships, identity mapping kernels are injected through the network. All of this allows the network to predict on a single forward pass.

News recurrent neural network approaches are presented in (Dave, Russakovsky and Ramanan, 2017), which are used for solving the problem of action detection in videos obtaining satisfactory results. In its basis the model: (1) Focuses on changes between frames, (2) predicts the future, (3) makes corrections upon it by observing what truly happens next.

Authors of (Sigurdsson et al., 2017) propose a model that is capable of detailedly reason about aspects of an activity, i.e, for each frame the model is capable of predicting the current activity, its action and object, the scene, and the temporal progress. This is accomplished by making use of Conditional Random Fields (CRFs) that are fed by CNN feature extractors. Moreover, for being able to train this system in and end-to-end-manner, an asynchronous stochastic inference algorithm is developed.

In (Wang et al., 2017) authors propose a CNN framework for the recognition of actions in videos, both trimmed an untrimmed and in (Aliakbarian et al., 2017) is proposed a multi-stage Long short-term memory network (LSTM) architecture combined with a novel loss function, that is capable of predicting action class labels in videos, even when only the first frames of the sequence have been shown. The model takes advantage of action-aware and context-aware features for succeeding in this task.

2.1. TSN Framework

Temporal Segment Networs (TSN) (Wang et al., 2017) is a CNN framework for the recognition of actions in videos, both trimmed an untrimmed. Along with it, a series of guidelines for properly initialize and operate such deep models for this task are proposed. The framework aims to tackle four common limitations when using Convolutional Neural Networks on videos. First, the difficulty of using long-range frame sequences, due to high computational and memory space costs, which can lead to miss important temporal information. Second, most of the systems focus on trimmed videos instead of untrimmed ones (several actions may happen in a video). Adapting to these would mean to properly localize actions and avoid background (irrelevant) video parts. Third, meanwhile deep models become complex, still many datasets are small in number of samples and diversity, lacking enough data for properly train them and avoid overfitting. Fourth, time consumed for optical-flow extraction can become a delay for both, using large-scale datasets and using the model on real-time applications. Figure 1 shows a schematic view of such network.

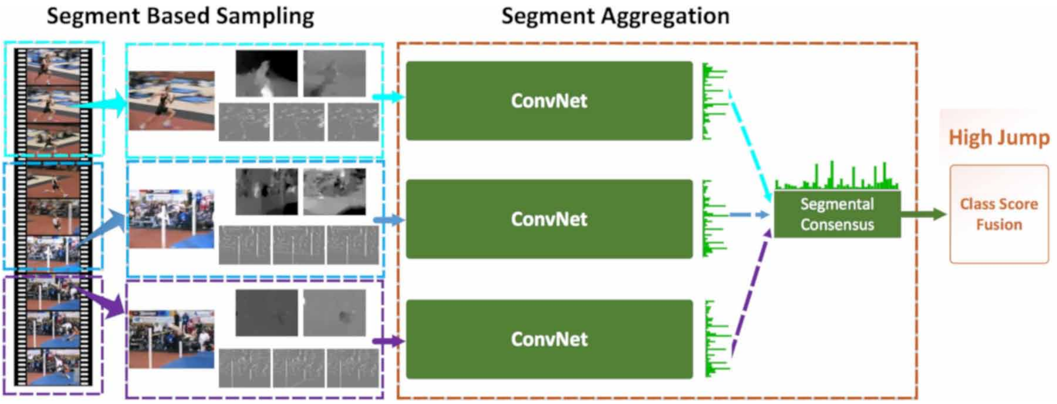
2.2. GPU Video Decoding

Since the beginning of the modern deep learning era, data storing and loading times have always been a bottleneck in the pipeline. Although recently we are witnessing great speedups thanks to new hardware technologies like SSD for storing, or data transferring devices (between CPU to GPU, and vice-versa) such as NVLINK, the issue persists.

Many of the research areas where this problem aggravates more are the ones which work with videos as the main dataset source. These include: predictive learning, video understanding, question answering, activity recognition, and super-resolution networks, between many others.

The main approach when tackling this problem in those areas is to first extract all the frames for each video of the dataset, for example by using FFmpeg, and save them in a high-quality image format, rather than one with possible lossy compression and artifact generations, in order to properly train the network. This comes with an increasing need of storage space, since the more information willing to be kept, the larger in size our converted image dataset will be.

Figure 1. Representation of TSN framework. First a snippet is extracted from each of a fixed number of segments that equally divide the video, Then, features such as optical flow or RGB-diff (top and bottom images of the second process column) are extracted. After passing through the corresponding stream, an aggregation function joins the individual snippet class probabilities. Then, softmax is applied for obtaining the final video action class.



In Figure 2 we can see the effects of storing the University of Central Florida 101 (UCF101) dataset (Soomro, Zamir and Shah, 2012), composed of only 13320 videos in different formats. For the case of JPEG (image), it occupies 63 GiB, while in AVI format (video) it occupies 9.25 times less, 6.8 GiB. If it is transformed to the proper MP4 format, needed by NVIDIA Video Loader (NVVL) (Casper, Barker and Catanzaro, 2018) with the corresponding number of frames, it occupies 14.2 GiB, still 4.44 times less. If we take this into a fine-grained level, such as frames, we can see that the storage differs by a large margin.

In order to alleviate this problem, a useful solution is to directly load video files into memory, decode the necessary frames, prepare them, and finally feed them to the network. Actually, APIs that can manage the first two steps exist: FFmpeg libraries itself, and higher abstraction modules like PyAV or PIMS, which both load data into the CPU. On the other hand, the (beta) Hwang project, also supports NVIDIA GPUs through the use of their specific hardware decoder units. Furthermore, those designed with machine learning tasks in mind, which can provide all the mentioned steps have been recently developed. Two are currently known: Lintel (Duke, 2018), and NVVL (Casper, Barker and Catanzaro, 2018). The first focuses on CPU loading (uses FFmpeg as backend), while the second targets GPU devices. Indeed, although being written in C++, it offers off-the-shelf PyTorch modules (dataset and loader). Moreover, another wrapper for CuPy arrays has been created.

Figure 2. Storage comparison between frames and video formats for the UCF101 dataset

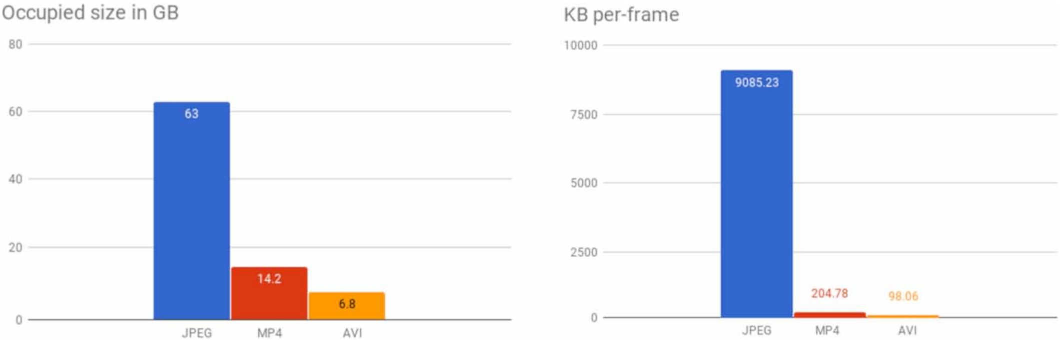
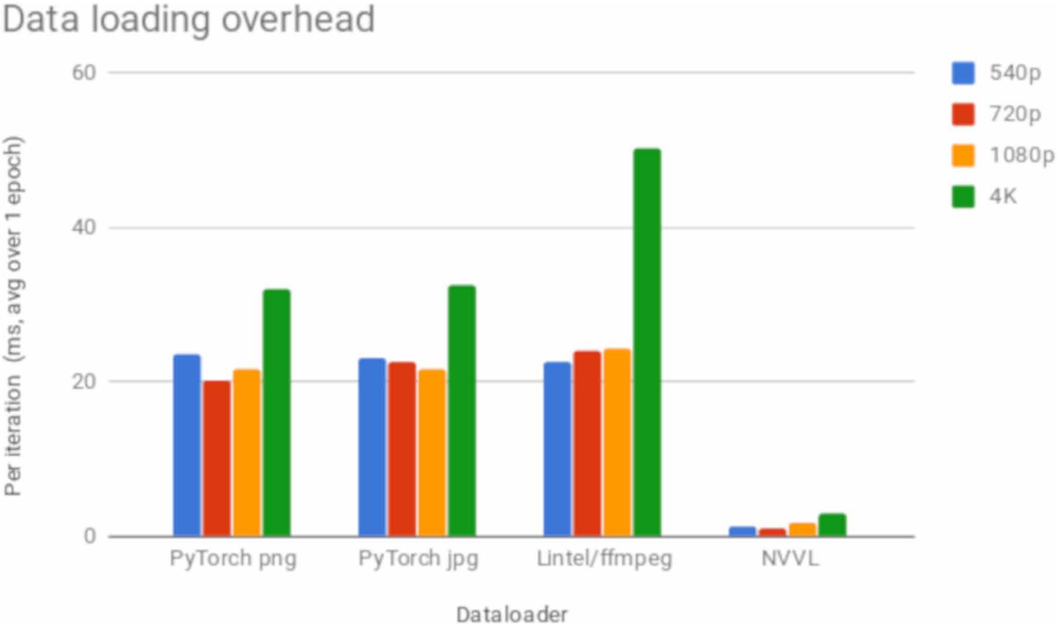


Figure 3. Average loading time (milliseconds) that 32-Floating Point PyTorch tensors take to be available in the GPU. The experiment was run on an NVIDIA V100 GPU over one epoch with batches of size 8. Figure extracted from (Casper, Barker and Catanzaro, 2018).



Regarding performance, we can see that NVVL reduces by a large margin the I/O processing times, as it can be appreciated in Figure 3. More benchmarks that take into account memory usage and CPU loads can also be found in the blog post, while an even more detailed evaluation is located on GitHub[1]. Regarding data, loading behaves like a sliding window of stride one, where frame sequences of a previously fixed length are subsequently loaded and returned as a single tensor. On the other hand, we can apply different transformations to these sequences: data type (float, half, or byte), width and height resizing and scaling, random cropping and flipping, normalizing, color space (RGB or Y: Luminance; Cb: Chrominance-Blue; and Cr: Chrominance-Red (YCbCr)), and frame index mapping. For performance, flexibility, and completeness reasons, we decided to use NVVL as our main tool to accelerate the TSN framework.

2.3. Hidden Two-Stream Convolutional Networks for Action Recognition

Another side for approaching the question of real-time action recognition can be found in (Zhu, Y et al, 2017), where the use of a convolutional network for automatically compute optical flow is presented.

More in detail, in a first phase, a CNN denoted as MotionNet is trained in an unsupervised manner for the task of optical-flow estimation. After obtaining acceptable results similar to optimal traditional methods, the network is attached to a conventional CNN as the temporal stream part of the whole model, being the spatial stream similar in architecture to the other one. Then, the network is trained (including MotionNet) on the task of action recognition from frame sequences. The approach enables the optical flow generator to be adapted to the characteristics of the task and further finding a suitable motion representation.

3. EXPERIMENTATION

In this project we have experimented using the approaches discussed with dataset ucf101.

3.1. Dataset

(Soomro, Zamir and Shah, 2012) Given the limited number of RGB action datasets that included realistic scenes (without actors or prepared environments) and a wide range of classes until 2012, authors of this paper propose a new large-scale datasets of user-uploaded videos (YouTube). These present a much diverse type of challenges than the ones of previous datasets, since recordings can contain different lighting configurations, image quality degradation, cluttering, movement of the camera, and occluded scenes.

In regard to the size of the dataset, 13320 videos are divided into 101 classes that cover five action groups: Human-Human Interaction, Sports, Playing Musical Instruments, Human-Object Interaction, and Body-Motion only. The actions contained in the first and fourth groups can be observed in Figure 4[2].

Furthermore, this dataset marked a milestone in what large scale action recognition datasets refers. It made possible to establish a well-known starting testbed to be improved as well as for benchmarking. Moreover, deep learning competitions where established around it, such as the different modalities of the THUMOS Challenge, which was run for three years in a row. After that, other large-scale datasets appeared, expanding the characteristics of the UCF101. For this, marking the start of an ever-growing number of diverse large-scale action recognition datasets, the UCF101 dataset is also worth of.

3.2. GPU Video Decoding Experiments

In order to test our GPU video decoding algorithm, we can compare the difference between the original frames and the ones loaded through NVVL. For this task, we are going to use the SSIM index between two pictures, usually used in the video industry for measuring the visual difference we can perceive when comparing frames of an original and downsampled video. It ranges from 0 to 1, where 1 is given for two identical pictures and 0 for two completely different ones. For example, given the two frames obtained from the UCF dataset (Diving class) that can be observed in Figure 5, we can notice a green band on the right extreme of the NVVL loaded image.

Apart from this, we cannot perceive any other substantial degradation in quality. Indeed, the SSIM obtained is 0.992, indicating that this artifact is probably due to a bug rather than a low-quality video processor. For reassuring this fact, we can compute the SSIM heat map, in order to locate other possible missed artifacts (Figure 6):

Thus, we can assume that there will be no harm at the time of incorporating this tool in a neural network training pipeline.

Figure 4. Classes for the Human-Object Interaction (blue) and Body-Motion Only (red) action groups from the UCF101 dataset. Figure extracted from (Soomro, Zamir and Shah, 2012).

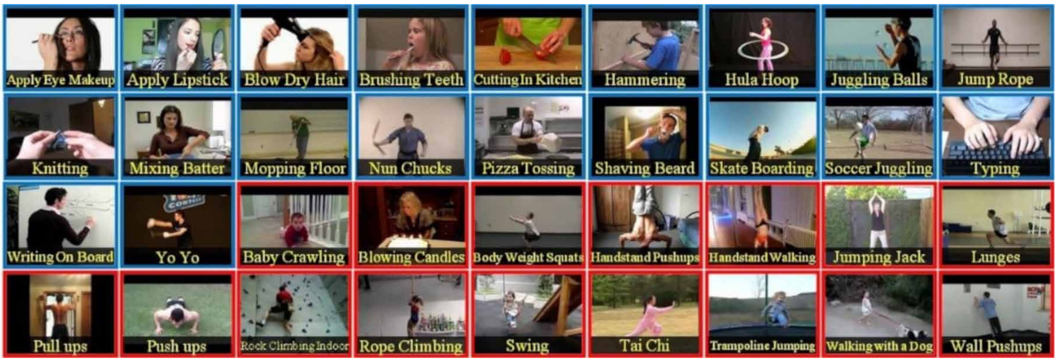


Figure 5. Original frame (left) and NVVL obtained frame (right). The frames pertain to a sample of the diving class in the ucf101 dataset.

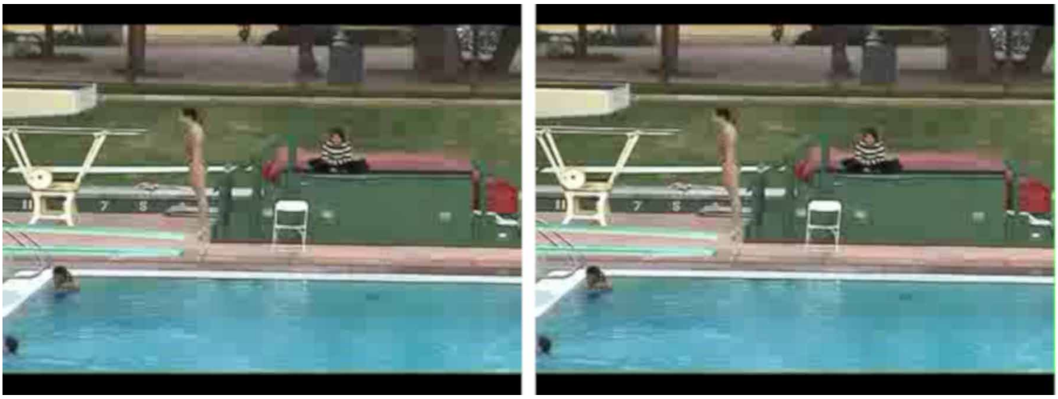
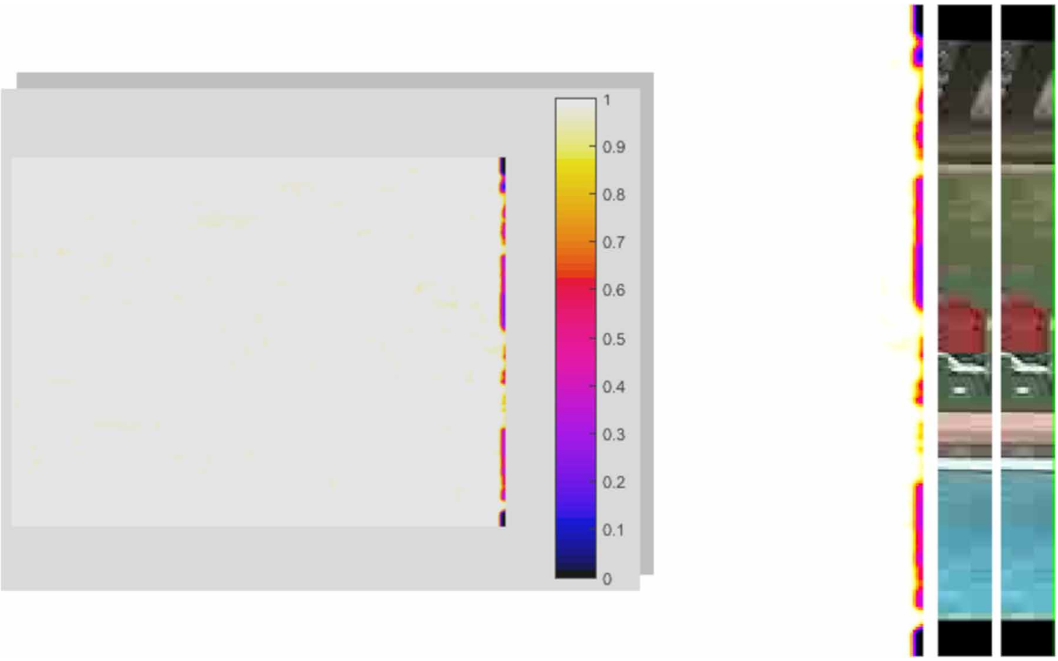


Figure 6. Heat map of above frames, the lighter the color, the closer to the original frame each pixel is



Now, we should pay attention to knowing which the current time speedup is we can obtain from replacing the image loading system of the TSN framework by a NVVL pipeline. For this, after adapting the frame-index generation functions, and integrating the video loader into it, we can perform the following:

1. Obtain a list of videos, get the total number of videos and the mean number of frames per video.
2. Extract all the frames from the videos, also convert them into the required NVVL video format.
3. Select the number of frames per video that are going to be loaded. For NVVL, all the frames have to be loaded.

4. Measure how much time it takes for extracting the told number of frames (into the GPU) on each occasion. For NVVL this only needs to be done once.
5. Obtain mean times and trend for the previous process and compare the results.

Step 1

We will use the first 450 videos of the UCF split-1 train list obtained with the data tools provided by the TSN framework. This list is formatted with a row for each video. In each row, the path to the video, the number of frames the video has and its class index. Due to this, the total number of frames can be obtained just by summing the second element of each row over the whole list. The resulting number is 87501. So, the mean number of frames in a video is approximately 194.

Step 2

For completing this step, we can simply follow the instructions and commands provided in the repositories of each project. We have to take into account that the extraction process can take a quite greater amount of time than the video converting process.

Step 3

In this case, we are going to load even number of frames, starting from 3 and finishing in 25, a total of 12 different instances. This has been selected since the authors of TSN test the model with 3, 5, 7, and 9 frames per video.

Step 4

For obtaining an accurate measurement, we are going to repeat each execution 29 times. For computing the time we have used Python *time.time* function. Also, in order to free all the resources in each run, we are going to loop inside a bash script instead of inside the Python executing script itself, thus having the process killed automatically.

Step 5

In this step we computed the mean values for each number of frames. The time taken for loading all the videos with NVVL is approximately 24.18 seconds. On the other hand, we can plot the results obtained from loading sole frames:

We can notice that the trend follows a lineal growth with respect to the number of frames loaded. Since we computed the equation defining the trend line (shown in the lower-right part of Figure 7), we can obtain a more precise approximation of the speedup achieved when using NVVL. For this, since we know the number of frames loaded with NVVL is the same as the mean number of frames obtained in Step 1, we just need to substitute it in the equation (X variable), obtaining a mean value of approximately 458.74 seconds or 7.65 minutes. We have achieved an improvement on loading time performance, leading to 18.97 times speedup when using NVVL.

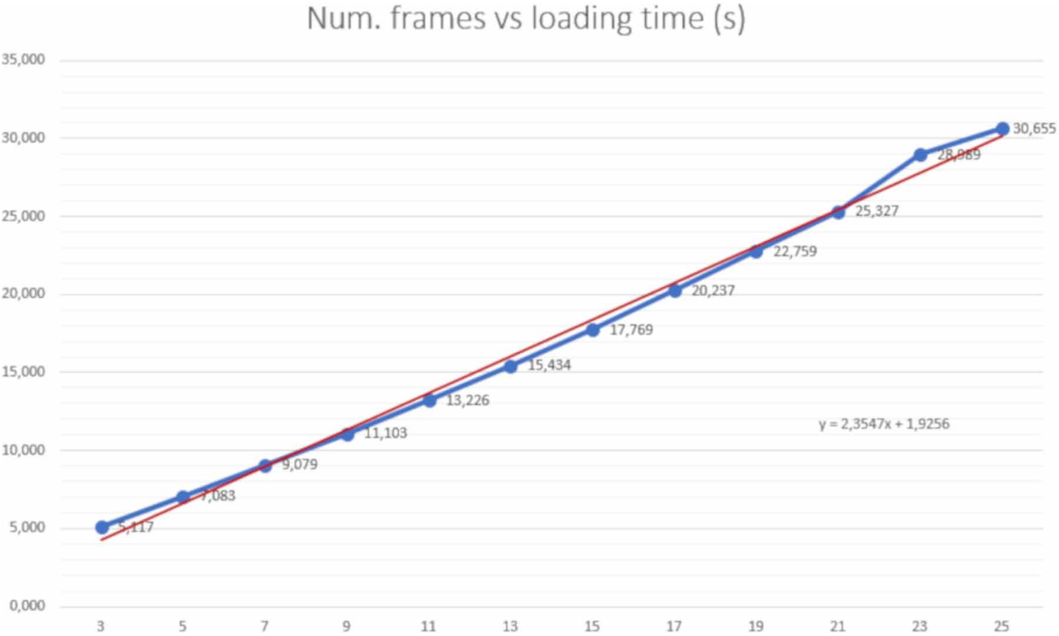
3.3. Training RGB TSN+HTS with NVVL

So far we have shown how useful incorporating NVVL into a video-consuming deep learning pipeline can be, it allows us to reduce both the storage and data transfer costs at the same time we do not suffer degradation in image quality. Now, what only remains is to incorporate this tool into a common action recognition scenario, where we train and test a network for learning to categorize human actions.

Such a network is going to be TSN, since it has demonstrated a superior performance in the task at hand. Moreover, we propose to make use of the converted HTS Caffe model and weights, in order to avoid pre-computing the optical flow and being able to use NVVL also in this stream, focusing the resulting pipeline for real-time applications. In spite of the dataset we are going to use, memory limitations detailed below, and time constrains, we are going to focus the following experiment only for the RGB stream.

Before starting, we need to prepare the data into a format that is compatible with NVVL. As referred in the GitHub repository[1], we need videos with either H.264 or HEVC (H.265) codec, and yuv420p pixel format, also they can be in any container that FFmpeg parser supports.

Figure 7. Mean loading time in seconds of each number of frames executed (blue). Trend line of from the obtained data (red). Y axis represents the loading time in seconds, while the X axis shows the number of frames used.



Moreover, we have to take into account the number of keyframes each video will have, i.e., a codec only contains a subset of all the frames that we see in a video, these are the keyframes. At the time of decoding, the rest of the frames are obtained by algorithmically inferring them through the keyframes. For this reason, when loading sequences that can start and end at any frame (similar to what we can do with NVVL), the system has to seek the nearest keyframe, which can be far before or after the starting frame. This can result into an underperforming execution, and for this reason when converting the videos, we have to indicate the frequency of keyframes per frame we want to have (Figure 8).

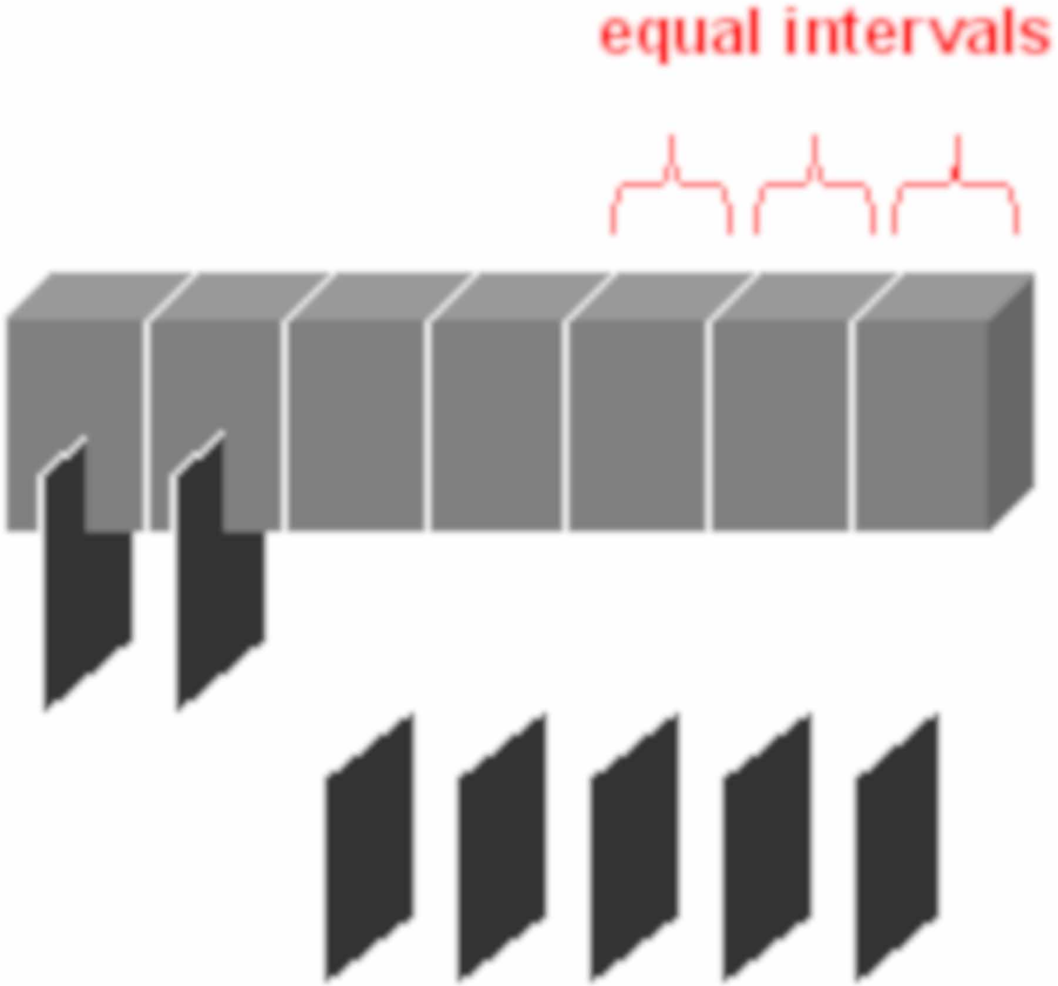
Developers of the video loader suggest to set one keyframe in intervals that correspond to the length of the sequences we are going to load. For example, if we are going to load sequences of length 7, then every 7 frames there will be a keyframe. Furthermore, they also provide the required commands to carry out this conversion with FFmpeg.

For our case, we are going to set every frame in the video to be a keyframe, this is due to the fact that currently the PyTorch wrapper (the C++ API seems more flexible) is intended for loading multiple frame sequences for each video with a sliding window approach of a fixed length. Although this length could be equal to the number of frames in the video, thus loading only a sequence per video, this would only work if all the videos had the same length, since this parameter, the sequence length, is global for the whole dataset.

For iterating over the dataset, we are going to use the data loader provided by NVVL PyTorch wrapper, where in each iteration it will load a batch of frame sequences. Since now, each frame is a sequence of length one, we need to set the batch size also to one. In this way we can easily know when the loader has fully output a video, add it to a list, and when we have enough videos, group them in a batch of the size we want for providing it to the network. Furthermore, for accomplishing this we also need to set to false the shuffle option in the loader.

Although we are ready for training our network, an impediment arises at the time of writing this work. Whether the videos have not been properly converted, or there is a code issue, the data loader

Figure 8. Representation of how keyframes can be evenly inserted into a video stream



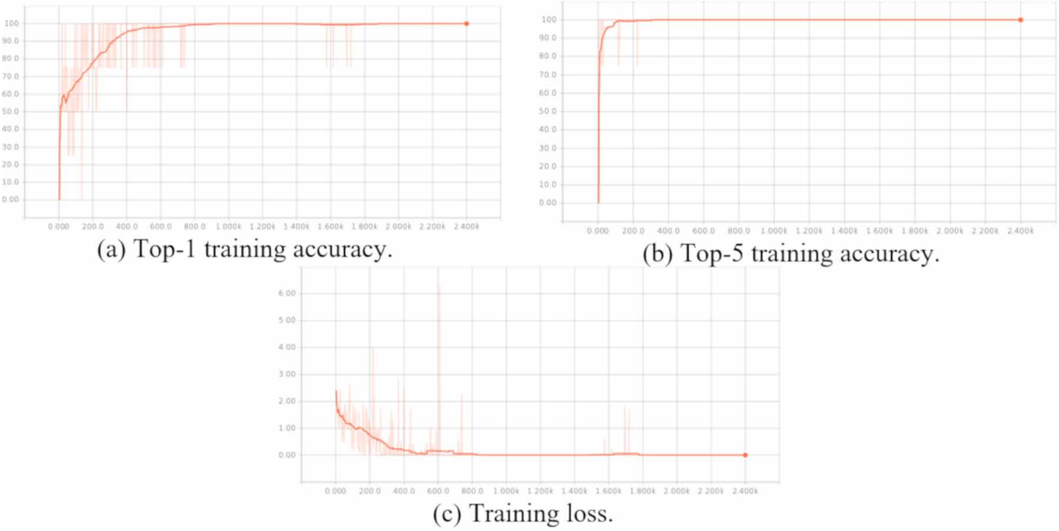
seems to get silently stuck when loading some videos. For solving this, one circumvention is to create a loader for each video instead of having one for the whole dataset.

So far this works, but what happens next is that GPU memory is not properly freed, thus limiting the size of our dataset to the space available on the graphic card at the moment. For Asimov, this concurs in having around 240 videos for training and 160 for validation (only the Titan card supports NVVL).

For this, following the same lines of motivation proposed at the beginning of the document, we are going to select daily actions for the reduced dataset we can work with. Specifically, it is composed of eight classes from the Human-Object Interaction group of the UCF101 dataset: *Apply Eye Makeup*, *Apply Lipstick*, *Blow Dry Hair*, *Brushing Teeth*, *Cutting In Kitchen*, *Mopping floor*, *Shaving Beard*, and *Typing*. The training set contains 30 videos for each action, while the validation one has 20 of them.

Regarding the training hyper-parameters, we are going to use the ones set by default for the TSN with the only exception of the batch size and number of epochs. For the former, we have set it to 4 dues to the limited memory, for the later, we will perform 40 epochs, which is enough for the model to converge with this dataset. For the metrics, we will keep track of the loss and top-1 and top-5 accuracies for both the training and validation sets.

Figure 9. Training curves for 40 epochs, 60 iterations per epoch



(a) Top-1 training accuracy.

(b) Top-5 training accuracy.

(c) Training loss.

Once all the epochs have been completed, we finish our training with in a common situation, 100% of top-1 (and top-5) accuracy and zero loss. Clearly, our network has overfitted. This is due to the scarce amount of data and its limited variability. Moreover, such deep networks (BN Inception) are prone to overfit, since they have more flexibility (bigger number of parameters) for adjusting to the data they are consuming while in training. However, we obtain validation accuracies of 76.25% and 98.125% for the top-1 and top-5 versions respectively, with a loss of approx. 1.52. Taking into account the data we are working with; these results are quite promising in comparison to what has happened in the training phase.

Now, we can get more insights if looking on how the training and validation have evolved. In the Figure 9:

Here we can take note of two facts. First, the top-5 accuracy converges much faster (iter. approx. 200) than the top-1 accuracy (iter. approx. 800). Clearly, this is something that can be foreseen, since it takes more time to learn the label of a video rather than guess it among five samples.

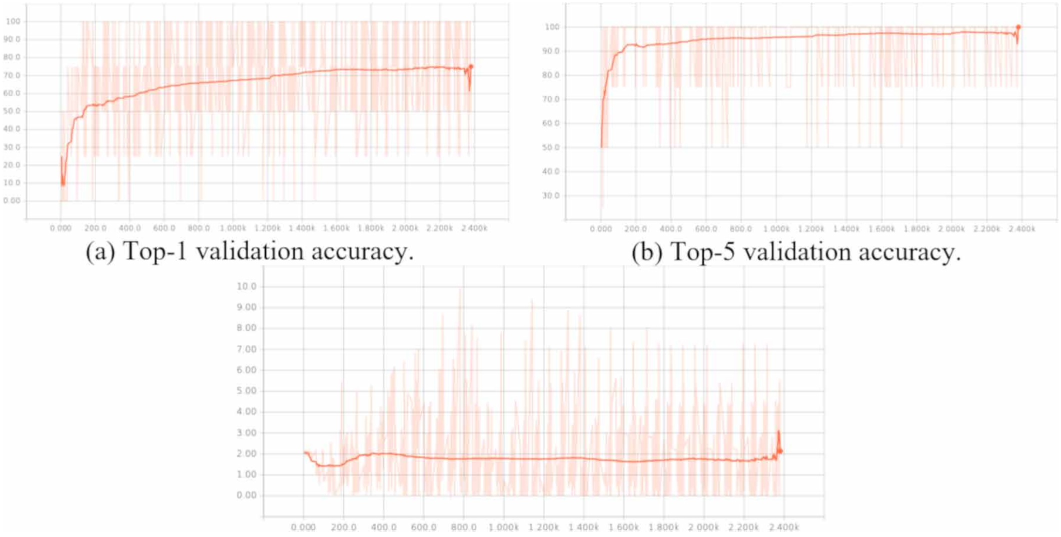
Secondly, we see that the unsmoothed curve (shaded red) bounces between higher and lower values (accuracy and loss) among the first iterations. This effect can be better seen in the validation curves (Figure 10):

This happens as a consequence of the small batch size we have previously set. The smaller is the batch size, the more weight updates we will perform. If it is too small, we could find the following:

- Instability: The frequent updates will cause the metrics to wander, going continually up and down.
- Not meaningful updates: The reduced number of samples makes it to contain less information about the error (negative gradient) direction in each update, thus needing a major number of epochs for converging into the same accuracy than with a bigger batch size. This can be summarized as longer training times.
- Hit a local minimal: Also known as plateau, and commonly induced by the previous statements, a small batch size can make that the network gets stuck on a non-optimum (nor sub-optimum) minimum of the loss function, obtaining insufficient performance results.

As intuition, we can take a look at the Figure 11, where on the left, the evolution of three types of batch size loss curves (arriving to the minimum) are plotted. The blue one, represents a batch of

Figure 10. Validation curves for 40 epochs, 60 iterations per epoch



(a) Top-1 validation accuracy.

(b) Top-5 validation accuracy.

the same size of the dataset, thus making only one update per epoch, a smother curve with a much less noisy evolution. Although it seems the best approach, the detail is in the time and space it takes to update the weights, since we have a large number of samples, we have to compute a vast amount of operations. Moreover, commonly is impracticable that a complete dataset fits into a modern GPU memory.

The purple curve is for the case where we realize one-sample updates, something that reflects, on extreme, what happened during the training of our network. Finally, the green curve shows the daily situation of most deep learning trainings, where the batch size is found in balance with the number of updates per epoch. Although there are frequent updates, they are not too much for firing divergence, at the same time a reasonable time is taken for finding the error.

In order to better visualize where the network guesses right or wrong, we can make use of the training and validation confusion matrices 12, where in each cell we can see the percentage of true positives for the class in the cell row. For example, in the validation matrix, 35% percent of the times we see the class *Brushing teeth*, the network sees it as *Shaving Beard*. Moreover, we can note that by obtaining the trace of a confusion matrix (summation over the diagonal) and dividing by the number of classes, we retrieve the final accuracy (Figure 12).

Easily, we notice what we determined before, the training set is overfitted, since for all diagonal cells the confusion matrix reports a 100% value (normalized between 0 and 1). On the other hand, when analyzing validation matrix, we can see that the network mostly fails when the classes are very similar. For example:

- *Apply Eye Makeup* is confounded 15% of the times with Apply Lipstick, since both use some kind of hand stick and cover zones of the face vertically near between each other, is logic to think that they are more difficult to differentiate.
- *Apply Eye Makeup* and *Shaving Beard* follow a similar error pattern, since in both actions there is hand movement over the zone of the mouth and arm movement around the whole face.

In other cases, the contrary can happen, when the action is easily differentiable from others, this mostly happens with two actions, *Mopping the floor* which usually happens in a room, and *Cutting in kitchen* where the camera focuses on the knife and the cutting table area (Figure 13).

Figure 11. Effects of batch sizes when training

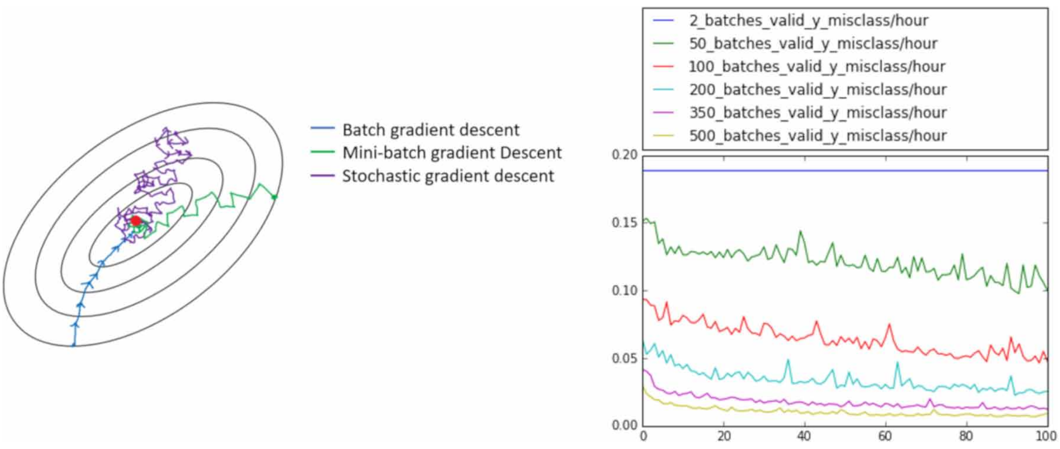
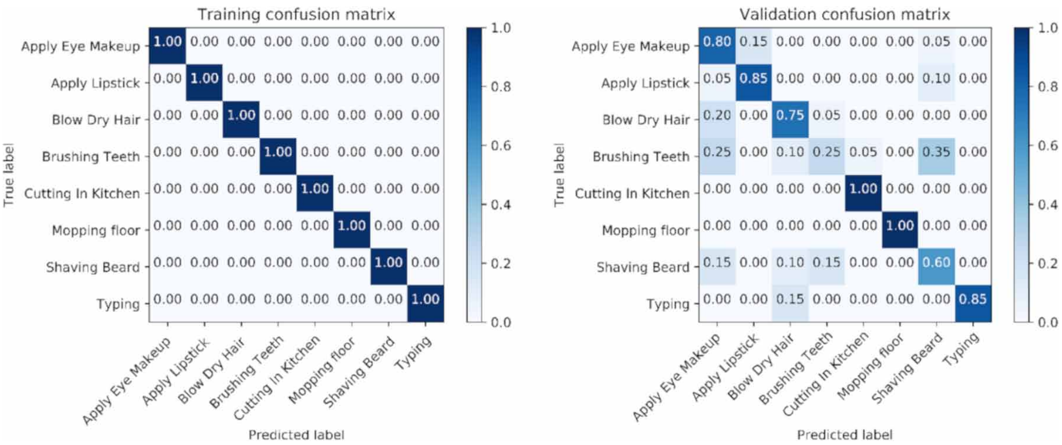


Figure 12. Confusion matrices for the proposed dataset



Training confusion matrix

	Apply Eye Makeup	Apply Lipstick	Blow Dry Hair	Brushing Teeth	Cutting In Kitchen	Mopping floor	Shaving Beard	Typing
Apply Eye Makeup	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Apply Lipstick	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Blow Dry Hair	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
Brushing Teeth	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
Cutting In Kitchen	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Mopping floor	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Shaving Beard	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Typing	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

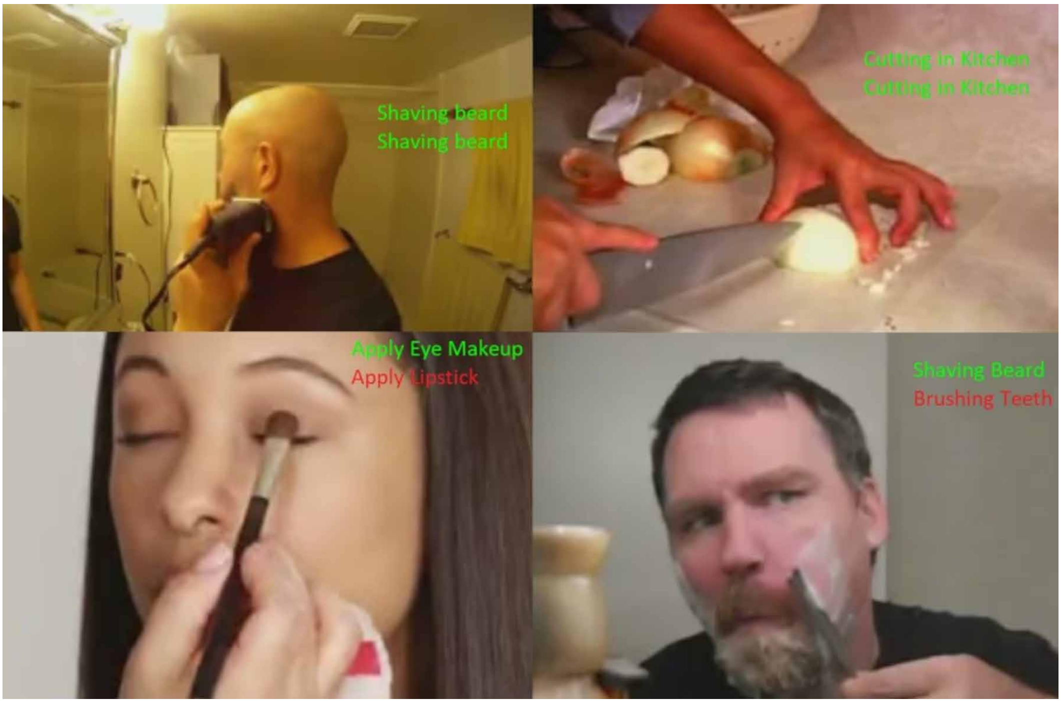
Validation confusion matrix

	Apply Eye Makeup	Apply Lipstick	Blow Dry Hair	Brushing Teeth	Cutting In Kitchen	Mopping floor	Shaving Beard	Typing
Apply Eye Makeup	0.80	0.15	0.00	0.00	0.00	0.00	0.05	0.00
Apply Lipstick	0.05	0.85	0.00	0.00	0.00	0.00	0.10	0.00
Blow Dry Hair	0.20	0.00	0.75	0.05	0.00	0.00	0.00	0.00
Brushing Teeth	0.25	0.00	0.10	0.25	0.05	0.00	0.35	0.00
Cutting In Kitchen	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Mopping floor	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Shaving Beard	0.15	0.00	0.10	0.15	0.00	0.00	0.60	0.00
Typing	0.00	0.00	0.15	0.00	0.00	0.00	0.00	0.85

4. CONCLUSION

In this work, we have focused our attention on different ways for accelerating the training and inference processes of a modern video-based action recognition pipeline. First, the use of a TSN framework, since it requires small amounts of data as an input. Secondly, the use of MotionNet from the HTS work, in order to achieve real-time optical flow computation times, adapt its representation for action recognition. Third, the use of the recent NVVL for reducing the cost of IO operations, save storage space, and speedup the whole pipeline by directly decoding videos on the GPU.

Figure 13. Class labels and network predictions: First line is correct label, second line is the predicted one, green if correct or red if not



REFERENCES

Aliakbarian, M. S., Saleh, F. S., Salzmann, M., Fernando, B., Petersson, L., & Andersson, L. (2017, October). Encouraging lstms to anticipate actions very early. In *IEEE International Conference on Computer Vision (ICCV)*. 10.1109/TPAMI.2018.2868668

Bellman, R., & Kalaba, R. (1959). On adaptive control processes. *I.R.E. Transactions on Automatic Control*, *4*(2), 1–9. doi:10.1109/TAC.1959.1104847

Buch, S., Escorcia, V., Shen, C., Ghanem, B., & Niebles, J. C. (2017, July). Sst: Single-stream temporal action proposals. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 6373-6382). IEEE. 10.1109/CVPR.2017.675

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8*(6), 679–698. doi:10.1109/TPAMI.1986.4767851

Carreira, J., & Zisserman, A. (2017, July). Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4724-4733). IEEE.

Casper, J., Barker, J., & Catanzaro, B. (2018). NVVL: NVIDIA Video Loader.

Chéron, G., Laptev, I., & Schmid, C. (2015). P-cnn: Pose-based cnn features for action recognition. In *Proceedings of the IEEE international conference on computer vision* (pp. 3218-3226). 10.1109/ICCV.2015.368

Dave, A., Russakovsky, O., & Ramanan, D. (2017, April). Predictive corrective networks for action detection. In *Proceedings of the Computer Vision and Pattern Recognition*.

Duke, B. (2018). Lintel: Python video decoding.

Efros, A. A., Berg, A. C., Mori, G., & Malik, J. (2003, October). Recognizing action at a distance. In null (p. 726). IEEE.

Feichtenhofer, C., Pinz, A., & Wildes, R. P. (2017, July). Spatiotemporal multiplier networks for video action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7445-7454). IEEE. 10.1109/CVPR.2017.787

Gavrila, D. M. (1999). The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, *73*(1), 82–98. doi:10.1006/cviu.1998.0716

Horn, B. K., & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*(1-3), 185–203. doi:10.1016/0004-3702(81)90024-2

Ji, S., Xu, W., Yang, M., & Yu, K. (2013). 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(1), 221–231. doi:10.1109/TPAMI.2012.59

Kong, Y., Tao, Z., & Fu, Y. (2017, July). Deep sequential context networks for action prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1473-1481). 10.1109/CVPR.2017.390

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2017, July). Temporal convolutional networks for action segmentation and detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1003-1012). IEEE.

Mahmud, T., Hasan, M., & Roy-Chowdhury, A. K. (2017, October). Joint prediction of activity labels and starting times in untrimmed videos. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 5784-5793). IEEE. 10.1109/ICCV.2017.616

Rabiner, L. R., & Juang, B. H. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, *3*(1), 4–16.

Rhinehart, N., & Kitani, K. M. (2017, October). First-person activity forecasting with online inverse reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 3696-3705). 10.1109/ICCV.2017.399

Schuldt, C., Laptev, I., & Caputo, B. (2004). Recognizing human actions: a local SVM approach. In *Proceedings of the 17th International Conference on Pattern Recognition ICPR 2004* (Vol. 3, pp. 32-36). IEEE.

Sigurdsson, G. A., Divvala, S. K., Farhadi, A., & Gupta, A. (2017, July). *Asynchronous Temporal Fields for Action Recognition* (Vol. 6, p. 8). CVPR.

Simonyan, K., & Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems (pp. 568-576).

Singh, G., Saha, S., Sapienza, M., Torr, P., & Cuzzolin, F. (2017, October). Online real-time multiple spatiotemporal action localisation and prediction. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 3657-3666). IEEE. 10.1109/ICCV.2017.393

Soomro, K., Zamir, A. R., & Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. arXiv:1212.0402

Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., & Van Gool, L. (2018). Temporal segment networks for action recognition in videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zeng, K. H., Shen, W. B., Huang, D. A., Sun, M., & Niebles, J. C. (2017, August). Visual forecasting by imitating dynamics in natural sequences. *In IEEE International Conference on Computer Vision (ICCV)* (Vol. 2). 10.1109/ICCV.2017.326

Zhu, Y., Lan, Z., Newsam, S., & Hauptmann, A. G. (2017). Hidden two-stream convolutional networks for action recognition. arXiv:1704.00389

ENDNOTES

[1] https://github.com/NVIDIA/nvvl/tree/master/pytorch/test

[2] http://crcv.ucf.edu/data/UCF101.php

John A. Castro-Vargas is a PhD student at the University of Alicante. His areas of interest are: Robotics, DeepLearning, gesture recognition and action recognition. He has participated in the nationally funded projects "Multi-sensorial robotic system with dual manipulation for human-robot assistance tasks" and "COMBAHO: COMe BAck HOme system for enhancing autonomy of people with acquired brain injury and dependent on their integration into society".