**Hochschule**
**Bonn-Rhein-Sieg**
*University of Applied Sciences*

**Fachbereich Informatik**
*Department of Computer Science*

# Deep Learning for Recognition of Daily-Living Actions

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Autonomous Systems

Maximilian Schöbel
Mat. Nr. 9027059

*Advisors:*
Prof. Dr. Erwin Prassler
Prof. Dr. Paul G. Plöger

October 1st, 2017

# Abstract

With the world's population growing older and caring facilities having reached their capacities already today, the need for assistive (robotic) systems is specifically critical in elderly care. In order to aid, an autonomous assistive system first needs to know in what form and when help is required. One way to achieve this is to recognize the currently performed actions by a human user and offer assistance accordingly.

This work studies *temporal order verification*, a quasi unsupervised pre-training method for improving the recognition of daily living actions from video data, using the *C3D* deep convolutional neural network. In particular, we evaluate *temporal order verification* as a means to incorporate motion sensitivity in 3D convolutional neural networks, which otherwise treat the temporal evolution of a video and the spatial dimension of video frames equally. Since pre-training a network in an unsupervised way does not require labelled data, this method can be specifically beneficial for recognition tasks where large amounts of labelled data are sparse. We focus on the Charades dataset, which features a unique yet small amount of mundane daily-living action videos and therefore enables the design of vision-based systems, which can be deployed in real-world applications such as assistive robotics.

Our *C3D* implementation yields an accuracy of 44.57% on the UCF101 standard action recognition benchmark and a mean average precision of 9.01% for recognizing daily-living actions of the Charades dataset, when trained from randomly initialized weights. By pre-training the model with *temporal order verification*, we were not able to improve classification results, in fact the pre-trained weights turned out to impair the network's performance. This result underlines the impact of weight initializations on the ability of a network to effectively learn a classification task and fuels the future search for potentially beneficial initialization methods.

## Statement of Originality

I, Maximilian Schöbel, declare that this thesis has not been previously submitted to this or any other university and that it is my own work except where explicitly stated otherwise.

Date: 30.09.2017

Signature:

# Contents

# 1. Introduction

## 1.1. Motivation

Assistive robots in domestic environments have the potential to aid persons with special needs in their daily living, thereby helping them to maintain a quality of life that would otherwise be lost. Robotic assistance can be beneficial to a variety of domains and users: the care and assistance of the elderly at home or in a nursing environment, assistance of persons with physical impairments and persons with cognitive or social deficits such as dementia or autism spectrum disorder.

The Department of Economic and Social Affairs of the United Nations estimates that the number of people in Germany above an age of 70 years will leap from 13 to 19 million, i.e. increase by around 46% over the next 20 years (see figure 1). With the world's population growing older and caring facilities having reached their capacities already today, the need for assistive systems is specifically critical in elderly care. Domestic robots may provide the freedom to keep living in the own home, prolong the time before living in a nursing environment becomes necessary and thereby relieve take the load off such institutions.

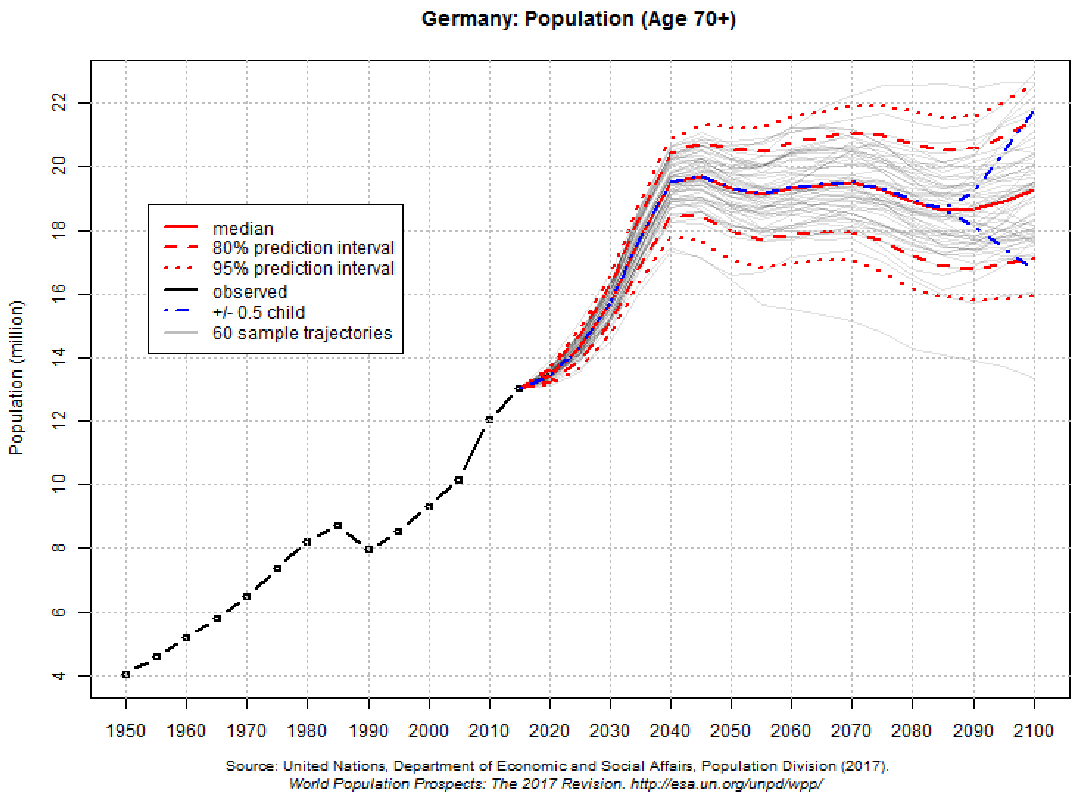


*Figure 1:* Estimation of the population of age 70+ in Germany

Contact assistive robotics, i.e. robots with the ability to manipulate objects in their environment offer to perform simple every-day tasks for the physically impaired, the elderly and people in general need of help. One of the biggest concerns in assistive robotics is inadvertently harming a user[1]. Robots equipped with manipulators, which represent additional powerful hardware, may raise the concern of possible harm and therefore struggle against widespread acceptance in elderly care or private homes.

In contrast, the field of socially assistive robotics aims at assisting persons through social rather than physical interaction[2]. Examples include robotic animal toys, which intends to improve the physiological as well as psychological fitness of elderly persons through robotic interaction. A prominent example is the robotic seal PARO[3], which targets at providing the social benefits of a pet without the usual occurring responsibilities. Socially assistive robots have since been extended to a variety of helpful functions, such as intelligent reminding, cognitive stimulation and mobile videophony[4].

This work is carried out in the context of the RoboLand project[1], which aims at increasing social interactions for elderly persons with dementia in rural regions through telepresence. Telepresence is an extended form of videophony through a remote controlled robot. The market research institute *Wainhouse Research* defines telepresence as "a videoconferencing experience that creates the illusion that the remote participants are in the same room with you"[5]. The underlying hypothesis of the RoboLand project is, that telepresence can have the biggest social impact in rural environments, where relatives live far from family members with cognitive impairments and visiting regularly is difficult.

Current telepresence robots span a wide spectrum of capabilities. The initial two models that were selected for practical evaluation in the RoboLand project are:

**Double**

A remote controlled self-balancing platform with an iPad attached to a telescopic extendable shaft.[2]

**Amy**

A ROS-based[3] robotic platform, designed for telepresence and social assistance. In contrast to *Double*, *Amy* is equipped with a variety of additional sensors such as ultrasonic sensors, infrared sensors, an inertial measurement unit, a 2D camera and a 3D camera.[4] Amy is therefore equipped to perform autonomous robotics tasks such as navigation, mapping and recognition of persons or objects.

[1] `www.roboland-projekt.de`
[2] `www.doublerobotics.com`
[3] `www.ros.org`
[4] `www.amyrobotics.com`

## 1.2. Scope of this Work

In order to provide assistance in a domestic environment, an autonomous assistive system first needs to know in what form and when assistance is required. This can either be achieved by registering direct commands from the user, e.g. through voice recognition and touchscreen inputs or by autonomous recognition of visual stimuli from the system's environment. By autonomously recognizing situations in which assistance is required, a robotic system is able to then offer a set of actions which could be executed to assist. This provides a form of social immersion of the robot, since it is actively able to engage the user and offer assistance even if it might not actually be required.

The underlying hypothesis of this work is that currently performed actions by a human user are the best indicator of whether assistance is required or human-robot communication should be initiated. This work therefore aims at providing a basis for improving assistive robotics through human action recognition from video data by an adapted deep learning approach, specifically targeted against the recognition of daily-living actions.

Since video cameras are a widespread and cheap technology, available in every mobile phone and naturally available in telepresence robots, a vast amount of video data for training action recognition algorithms is easily obtainable from online video platforms such as YouTube. An action recognition system that processes 2D video-data can therefore be easily deployed, since regular 2D cameras form the standard equipment of most robots. In the context of the RoboLand project, we aim at extending the deployed robots beyond the telepresence functionality towards social assistance through human action recognition.

Action recognition is a classification task. An action recognition system is presented with a video, usually a short clip that contains the performance of a single action, and then outputs the corresponding action class according to a finite and predefined set of recognizable actions. In learning systems, this set of recognizable actions is defined by the labelling of the dataset that was used to train the system. A properly labelled and big dataset is therefore vital for successfully training an action recognition system.

Deep learning approaches have shown remarkable results in a multitude of vision tasks. These include handwritten digit recognition[6], object detection and recognition[7][8] and scene segmentation[9]. In our previously conducted literature survey[10], the performance of deep learning approaches in action recognition has been compared to the classical method of using hand-crafted features. In that work we concluded that deep learning approaches do not yet outperform state-of-the-art hand-crafted feature methods, but show promising and competitive results. We therefore aim at increasing the performance of a deep learning approach for action recognition and specifically train a system for recognizing daily-living actions, which could be deployed on an assistive robotic system in the real-world.

## 1.3. Challenges

The practical value of an action recognition system heavily depends on the data, that was used for training it, since the dataset defines the action classes that can later be recognised; therefore, a sufficiently large dataset of daily-living actions is necessary to obtain a classifier that can be used for recognising daily-living actions in a real-world setting. The common disadvantage of deep neural networks, namely that they require a lot of labelled data in order to generalize effectively, has to be addressed. Since manually collecting and labelling video data is expensive, action recognition datasets are usually sampled from sources like movies, television broadcasts or online video platforms[11]. These videos however are often biased towards unrealistic or entertaining videos in staged and uncommon settings. Traditionally, sports video have been used for general action recognition benchmarking datasets, since sports videos are widely distributed and a distinct label is easily obtainable.

Fortunately, the work of Sigurdsson et al. [12] already addressed the challenge of obtaining a dataset with *boring* daily-living actions. Since these types of actions are hard to sample from movies or the internet, a series of crowdsourced tasks was deployed to compile the dataset called Charades[12]. The complete process of creating actions scripts, recording the videos and annotating them with action labels was done by 267 individuals from three different continents. The videos in the Charades dataset were recorded in regular homes by a decently large number of private actors. The dataset thus contains a decent amount of variety in action performances in real-world scenes. In particular 9,848 videos with 66,500 individually labelled and temporally localized action instances are contained in the dataset, as it is specifically designed to enable robotic applications in real-world environments.

The size of 66,500 action instances is extremely small for a deep learning dataset, since deep learning architectures usually require more data. In comparison, the two currently biggest action recognition datasets are Sports-1M[13] (around one million noisily labelled sports videos) and YouTube-8M[14] (around eight million noisily labelled YouTube videos). The practical benefit of these datasets however is questionable, since the labels are noisy and the contained actions are not temporally localized. Recently, the Kinetics dataset was published, which addresses the issues of action recognition datasets being too small by providing videos. The actions in Kinetics are comparably well localized, since each video has a fixed length of 10 seconds which contains the action. Even these datasets are magnitudes smaller compared to common image classification datasets, such as ImageNet[15] with a total number of $14,197,122$ images or the 80 million tiny images dataset[16] with $79,302,017$ images.

In addition to generally smaller dataset sizes, processing video data yields further challenges not present in the domain of image processing. Since multiple frames are being processed per training step, a single video input has a higher dimensionality than an image input by a factor according to the number of processed frames per iteration. This leads to bigger sizes of individual inputs and therefore smaller batch sizes, i.e the number of inputs processable per training step, need to be incorporated to not exceed memory limitations.

Furthermore, videos are encoded in order to decrease file-sizes for distributing them. Since pre-decoding and storing raw RGB frames would have exceeded our storage capacities, a partial decoding step has to be incorporated into the data-preprocessing pipeline for network input generation. This adds to the time until an input is ready to be further processed and potentially, if not handled otherwise, delays the overall training process.

## 1.4. Methodology

In deep learning for action recognition, two main approaches can be identified:

1. Extending the convolutional kernels in regular (2D) CNNs to the temporal dimension and thereby enabling them to process 3D video volumes that result from stacking consecutive video frames. One of the first publications that incorporate 3D CNNs for action recognition is [17].
2. Incorporation of two separated processing streams of 2D CNNs into the overall network architecture. A spatial recognition stream processes RGB video frames and an additional temporal recognition stream receives optical flow inputs to incorporate explicit motion information. The prototypical publication that describes this approach is [18].

Our previously conducted survey[10] concludes that the two-stream approach in general performs better, than single-stream 3D convolutional networks. The major deficiency of 3D convolutional neural networks is that they treat the temporal and spatial domain equally: a video volume of stacked input frames is treated as a regular 3D object. However, the temporal dimension is not inherently equal to the spatial dimension since adjacent frames correlate by obeying the laws of physics, i.e. shown objects cannot instantaneously disappear but instead move with finite velocities, so that the amount of change between frames is limited.

To address the above stated challenges, we incorporate an unsupervised pre-training strategy called *temporal order verification*, which has previously shown to increase performance of 2D CNNs in action recognition. During pre-training with *temporal order verification*, a model has to determine whether a presented input is in correct or incorrect order. These inputs can be sampled from source videos by creating network inputs from the video frames and possibly permuting them. This is in the strictest sense a supervised pre-training strategy; however, since the labels *correct temporal order* and *incorrect temporal order* are automatically determined, it is treated as an unsupervised method. The obtained model weights are then used as initialization for training on the actual labelled dataset.

Pre-training a model with *temporal order verification* has the following beneficial properties, which makes it particularly interesting for our work.

- since it is a quasi unsupervised pre-training strategy, large amounts of possibly unlabelled video data can be incorporated in the pre-training process.
- Misra, Zitnick, and Hebert [19] describe the benefit of *temporal order verification* in that the network needs to reason about the movement in the presented inputs, in order to determine the temporal order. The learned representations, therefore need to encode motion of the inputs, which is highly informative for the presented actions.

In this work, we investigate the transfer of this pre-training strategy to a 3D CNN model, namely the *C3D* convolutional network as described in [20]. Our working hypothesis is that *temporal order verification* is especially beneficial for a single-stream 3D CNN in the domain of daily-living action recognition because:

1. There is not much labelled training data available that features *boring* daily-living actions in real-world environments and incorporating an unsupervised pre-training strategy addresses this problem. Specifically, the benefit of using the Kinetics dataset for pre-training is evaluated.
2. *Temporal order verification* has shown to result in input representations that focus on human motion by comparing the temporal relations between inputs.

To address the problem of pre-processing times, we implement a high performant multi-threaded input pipeline, that decodes and prepares network inputs in parallel. According to the available hardware, an arbitrary number of threads can feed the prepared network inputs into a FIFO-queue for training. To address the problem of small batch-sizes, we incorporate a multi-GPU setup for training in which several GPUs process individual batches. Averaging of the gradients per GPU results in an effectively bigger batch size.

# 2. Related Work

The field of action recognition from video can broadly be divided into two main approach classes:

**Hand-crafted feature based approaches**

These video classification algorithms extract video/image features around previously identified areas of motion from an input video. These features are aggregated into a fixed-length representation of the video and then classified using an arbitrary classifier. The performance of these approaches is heavily influenced by *where* features are extracted and *what* kind of features are being extracted. Interest-point detectors and feature extractors are carefully hand-crafted for that purpose. Although hand-crafted feature based approaches yield competitive results in action recognition, the results of Wang et al. [21] indicate, that there is no universally best manually engineered set of features for all tasks.

**Deep Learning based approaches**

(Deep) Learning approaches eliminate the need of manual feature engineering by learning to extract descriptive features and video representation directly from training data. Deep models are end-to-end trainable, i.e. once a model architecture is constructed it requires a large amount of labeled training videos to learn the direct correspondence between action label and input data.

## 2.1. Hand-Crafted Feature-based Approaches

The research interest in human action recognition, fuelled by the amount of possible applications, has produced a decent number of comprehensive survey articles about human action recognition approaches with hand-crafted features and deep learning [22, 23, 24, 25, 26, 27]. This section provides a very brief overview of the main ideas behind using hand-crafted feature based methods. For a detailed review we refer the reader to the article of Poppe [22].

In the class of *space-time volume* approaches, a global video representation is built from the complete input video. The input video is thereby interpreted as a 3D space-time volume, which results from stacking the individual video frames. These approaches use template matching to compare the representation of an to be classified input video with a previously created template that represents the input class.

Bobick and Davis [28] create video representations by overlying the foreground silhouettes of the person performing action. Treating each silhouette as equally important results in the binary motion-energy image. Letting the silhouettes decay over time results in the scalar valued motion-histogram image, as displayed in figure 2.

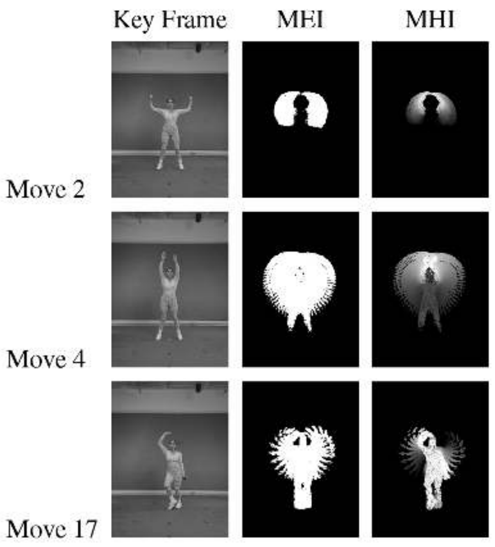


*Figure 2: (left)* Example frames of three videos showing different ballet moves. *(middle)* Motion-energy image (MEI). *(right)* Motion-history image (MHI) [28]

The main idea of *trajectory-based* approaches is that the trajectories of a persons tracked joints provide informative to recognize the displayed action. The resulting joint trajectories can either be compared directly by template matching or trajectory can be extracted to be classified. Sheikh, Sheikh, and Shah [29] track 13 joint positions and use the resulting trajectories for action recognition. The approach is illustrated in figure 3.

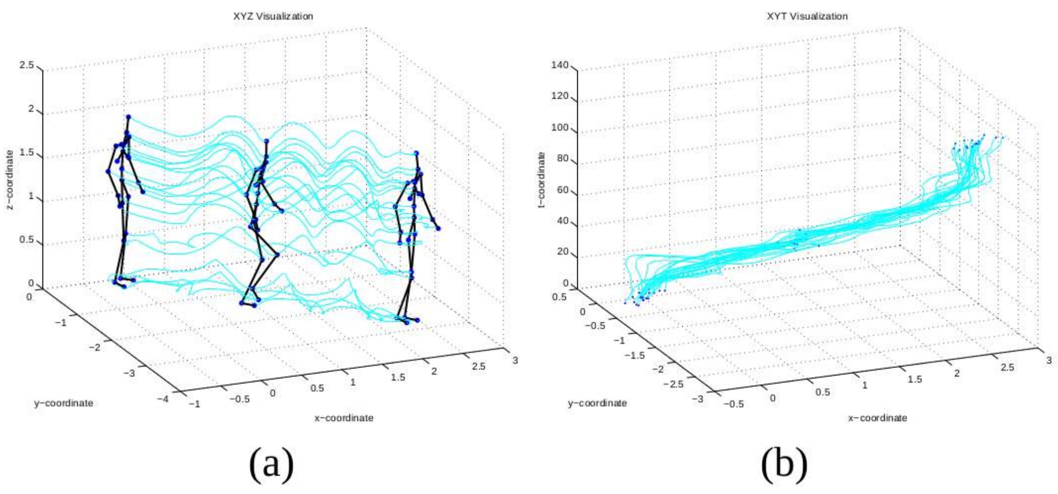


*Figure 3:* Tracking a person's joint positions to obtain motion trajectories for action recognition. *(a)* XYZ space. *(b)* XYT space. [29]

Another class of hand-crafted feature approaches uses *local space-time features.* These approaches have emerged as the standard approach in video classification with manually engineered feature extractors [13]. Local space-time features are descriptive video attributes, that are extracted from a local region of an input video, where sudden changes in motion appear (also called spatio-temporal interest points [22]). These points are considered highly informative for action recognition.

Generally local spatio-temporal feature approaches can be divided into three stages: [13][23]

1. **Feature Extraction**: Describes the process of determining where regions of motion are present and then transforming the raw RGB values around that region into a descriptive feature representation. Feature extraction can be further divided into the two sub stages interest-point detection and local description [22].
2. **Feature Encoding**: Encoding all of the previously extracted features into a fixed-size video representation. Common representations are created using the bag-of-visual-words paradigm, the Fisher Vector[30] or the Vector of Locally Aggregated Descriptors[31].
3. **Classification**: Learns the correspondence between the video representations and the action class it contains.

Prominent examples of approaches using local spatio-temporal features were proposed by Laptev [32] and Dollár et al. [33]. The approach of [33] incorporates a feature detector, that focuses on periodic motion. A detected interest point is then described as a cube of video pixels called a *Cuboid.* The raw local pixel values of each cuboid are then further processed and aggregated into a global video representation. Figure 4 illustrates this approach.

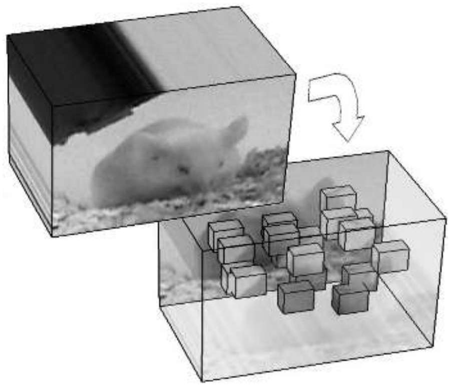


*Figure 4:* [33]

A hybrid approach of using local spatio-temporal feature around previously sampled trajectories is proposed by Wang et al. [34]. The approach is called *Dense Trajectories* and is illustrated in figure 5. Instead of computationally costly feature detection, a dense number of points is sampled from the video's frames. These points are then tracked along the temporal evolution of the video frames to capture the illustrated motion. Local spatio-temporal feature are then extracted around the resulting trajectories and aggregated for action recognition.

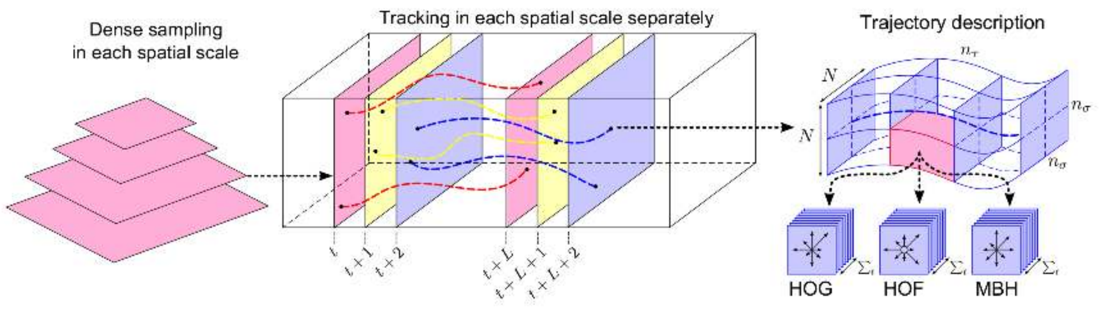


*Figure 5:* Description of densely extracted trajectories [34]

Since background motion constitutes to the sampling of trajectories as well but does not convey any information about the performed action in the video, the approach was later extended to filter background motion by camera motion estimation[35]. The *Improved Dense Trajectories* approach yielded state-of-the-art performance in action recognition by the time it was published.

## 2.2. Deep Learning Approaches

Applying deep learning techniques to action recognition is motivated by the results that were achieved on other vision tasks. In contrast to hand-crafted feature-based methods, no manual feature engineering is required, because extracting features and encoding an input video into a descriptive representation can be learned from large amount of labelled training data.

Besides discriminative models such as convolutional neural networks, generative models are adopted for action recognition as well. Examples include Convolutional Deep Belief Networks[36] and Recurrent LSTM Auto-encoders[37]. Generative models are particularly interesting for action recognition because they can be trained in an unsupervised way, i.e. with unlabelled input data. Action recognition video datasets are generally harder to obtain, store and label than images, because they have bigger file sizes and an action in a video needs to be identified and temporally localized.

Using convolutional neural networks in action recognition can broadly be divided into two classes:

1. 3D CNNs are the natural extension from regular 2D CNNs to using three dimensional convolutional filters and thereby able to directly process spatio-temporal

data, i.e. video volumes.

2. Two-stream network use regular 2D CNNs but process inputs in two separate streams. A spatial stream processes raw RGB input frames, while an additional temporal stream processes direct motion information in the form of optical flow.

### 2.2.1. 3D Convolutional Neural Networks

An early approach to incorporate three dimensional convolutional kernels for action recognition has been proposed by Ji et al. [17]. The architecture still incorporates an initial hard-wired layer, i.e. fixed transformations to extract greyscale frames, frame gradients and optical flow along the horizontal and vertical axis. Later models eliminate the first hard-wired through trainable layers and are considerably deeper. This architecture, as illustrated in figure 6 contains $295,458$ trainable parameters in total.[5]. The approach showed competitive results, but did not outperform the state-of-the-art at that time.

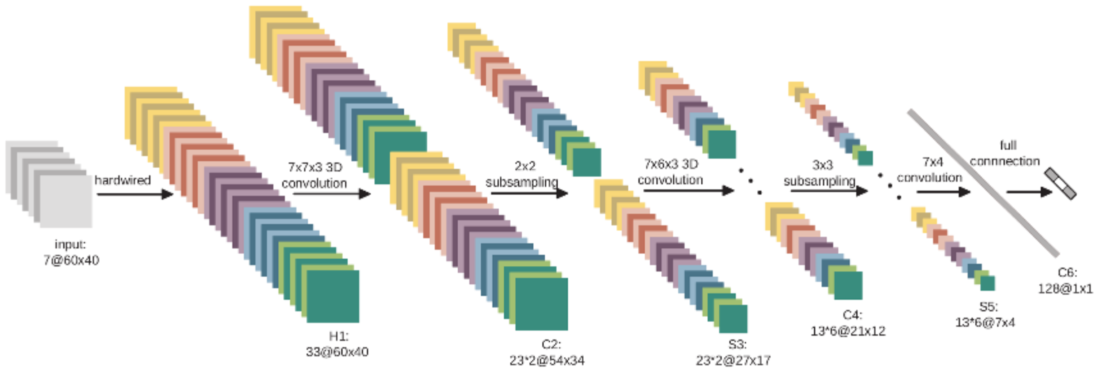


*Figure 6:* Early 3D CNN for human action recognition [17]

Building on the work of Ji et al. [17], follow-up approach of Baccouche et al. [38] addresses two main previous deficits. That is *(1)* the previous model still relies on a hand-crafted processing and *(2)* only 15 frames were processed. Therefore a 3D CNN without fixed pre-processing layers is used as a feature extractors and an LSTM was used to aggregate resulting clip representations to perform action recognition.

[5]For comparison: The *C3D* model used in this work contains around $78,000,000$ parameters

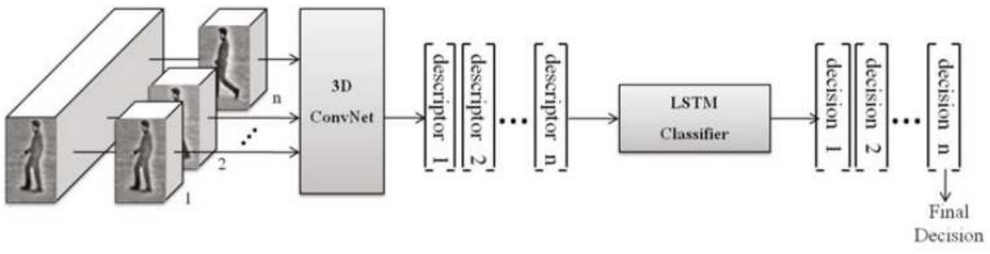


*Figure 7:* 3D CNN + LSTM architecture. [38]

This approach performs comparably to the one of Ji et al. [17]. Aggregating 3D CNN activations with an LSTM only results in a minor increase in accuracy, i.e. from 91.04% to 92.17% on the KTH dataset[39].

Karpathy et al. [13] perform a comprehensive evaluation on how to incorporate temporal information in convolutional neural network architectures in general. The naive approach of simply classifying individual video frames and then averaging the results is compared to multiple different fusion techniques. Using *Late Fusion* two 2D CNNs are applied to two video frames with a fixed temporal distance between them. An *Early Fusion* approach incorporates 3D convolutions only in the first network layer and the *Slow Fusion* approach is equivalent to general 3D CNNs. The results show, that the *Slow Fusion*, i.e. 3D convolutions in all network layers, performs best. This work thereby backs up the use of 3D convolutions, which has among others motivated the development of the *C3D* model (see section 2.3).

The work of Varol, Laptev, and Schmid [40] addresses similar to Baccouche et al. [38] a common deficit in 3D convolutional architectures for action recognition, namely that they process short, fixed-length video sub-sequences only. Common temporal extends being learned from only span around 1-16 input frames at a time [17][13][20]. The authors provide a systematical evaluation of the influence of temporal input length in 3D convolutional networks. The used network architecture for evaluating temporal input lengths of $T = \{20, 40, 60, 80, 100\}$ frames is illustrated in figure 8.

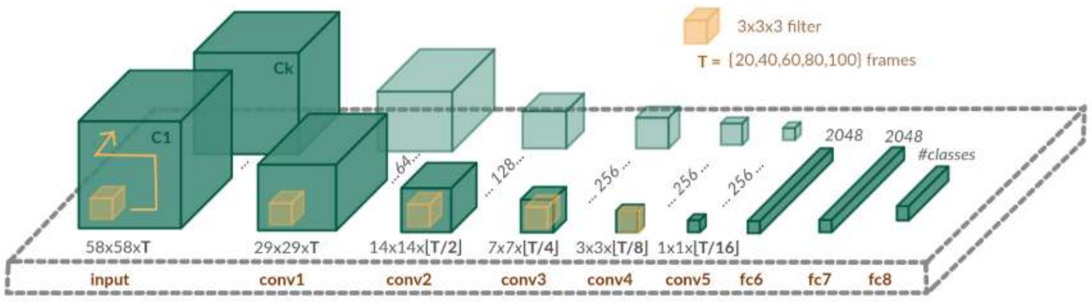


*Figure 8:* 3D CNN architecture for evaluating the influence of the number of input frames on action recognition accuracy. [40]

In order to process temporally longer inputs and not exceed computational limitations, the spatial resolution of inputs is decreased to either $58 \times 58$ or $71 \times 71$ pixel. The results show, that incorporating temporally longer inputs for action recognition heavily improves the performance of the model. On UCF101 an increase of +9% recognition accuracy was achieved by extending the temporal input length from 16 to 60 frames.

### 2.2.2. Two Stream Networks

Two-stream convolutional neural networks for action recognition have to our best knowledge first been proposed by Simonyan and Zisserman [18]. Due to their state-of-the-art performance, they have quickly been adapter by the action recognition community. Two-stream networks are inspired by the human visual cortex, which according to [41] contains two separate processing paths: One for static object recognition, and another independent one for guiding actions towards the recognized objects.

Simonyan and Zisserman [18] adapt this hypothesis in their neural network architecture for action recognition by implementing a spatial and a temporal stream. The spatial stream processes raw RGB video frames, while the temporal stream receives stacked optical flow images in different channels. The combined architecture is illustrated in figure 9.

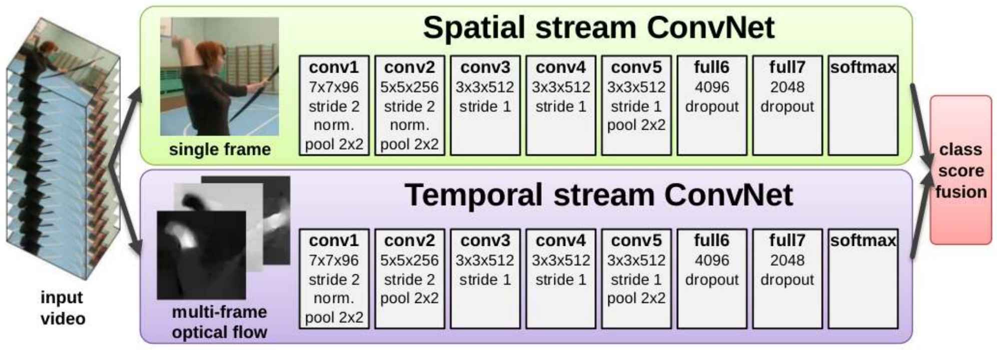


*Figure 9:* Spatial stream and temproal stream network in the two-stream action recognition architecture by Simonyan and Zisserman [18]

Both streams are implemented as 2D CNNs. The authors identify the advantage in separating the two streams in being able to pre-train the spatial network on large pre-existing image datasets such as ImageNet [15], since it is basically an image recognition architecture. Both streams are fused late, after each network produced an individual class prediction. The authors found fusing the class scores by an SVM works best and results in an accuracy of 88.0% on UCF101 and 59.4% on HMDB51.

As Baccouche et al. [38] and Varol, Laptev, and Schmid [40] addressed the deficit of short temporal input lengths in 3D CNNs, Ng et al. [42] address the same issue in the two-stream approach. They aim at constructing a video representation in a two-stream setup from inputs with lengths in the order of tens of seconds. The main idea is to apply multiple 2D CNNs to inputs in parallel, pooling the results and then merging them with the temporal stream.The number of applied CNNs thereby determines the processed input lengths. To pool the CNN outputs, LSTMs or additional pooling are evaluated. The approach is illustrated in figure 10 and outperforms Simonyan and Zisserman [18] the original two-stream network by a small margin.

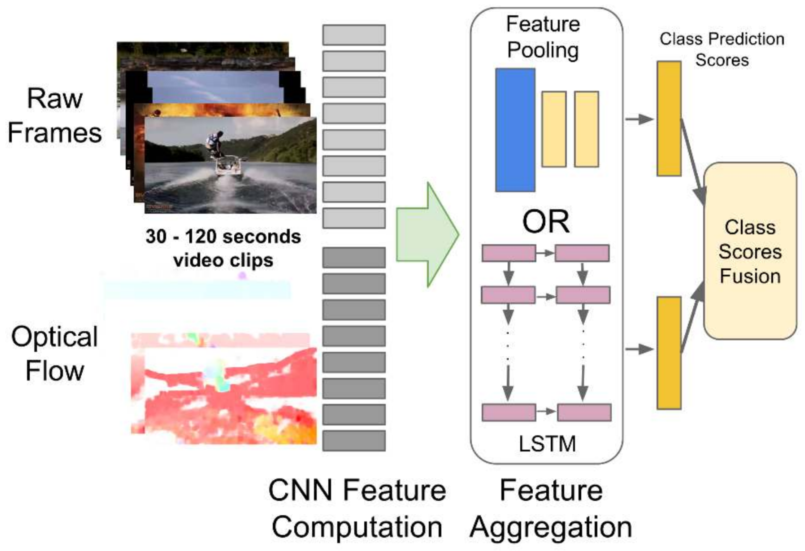


*Figure 10:* Approach to incorporate longer temporal inputs in a two-stream action recognition setup. [42]

Feichtenhofer, Pinz, and Zisserman [43] aim at improving the two-stream architecture by incorporating more sophisticated fusion strategies. These include earlier fusion of the streams and keeping one of the streams after fusion, to re-fuse it again in the final layer. These approaches are illustrated in figure 12.

Additionally several approaches of temporal fusion, i.e. incorporating the results of several temporally distant network inputs are being evaluated. The authors identify an advantage of early network fusion: the number of parameter can be reduced by around a factor of 2. The best performing temporal and spatial fusion approach results in an increase in performance of +3.7% with an significantly decreased number of network parameters.

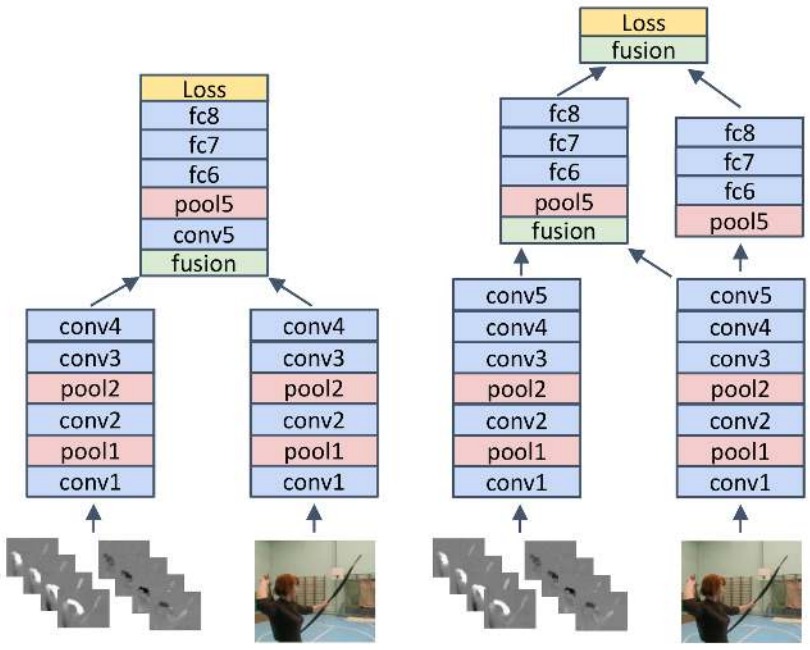


*Figure 11:* [43]

*Figure 12:* Placement of fusion layers between the spatial and temporal stream in a two-stream architecture. (*left*) Only the merged stream is kept. (*right*) The spatial stream is maintained after first fusion operation and merged before the final layer. [43]

### 2.2.3. Hybrid Approaches

Wang, Qiao, and Tang [44] extend the hand-crafted feature-based approach using *Dense Trajectories* by Wang and Schmid [35] with deeply learned features of two-stream CNNs [18]. The approach is called *Trajectory-pooled Deep Convolutional Descriptors (TDD)*. A two-stream CNN is first trained on a large dataset as in [18]. Dense trajectories are then extracted from input videos [35] to identify regions of motion. The previously trained two-stream CNN functions as a generic feature extractor instead of using hand-crafted feature extractors, to obtain discriminative features around the sampled trajectories. These features are then aggregated over the whole video using the Fisher Vector representation [45] and classified with an SVM.

Another successful hybrid approach of combining a two-stream architecture with 3D convolutions was recently published by Carreira and Zisserman [46]. The authors use a pre-trained 2D CNN, i.e. the Inception-V1 model [47] pre-trained on ImageNet and extend it to process spatio-temporal input data. Extending is done by *inflating*Carreira and Zisserman [46] the model, i.e. each 2D convolution and pooling kernel is replicated along an additional third dimension. The authors note, that the inflated 3D model is already properly initialized from the pre-trained weights of the 2D kernels.

## 2.3. C3D

The work of Tran et al. [20] provides a prominent example of a 3D convolutional architecture, which does not require the explicit calculation of optical flow. Optical flow has previously been incorporated in an additional temporal network stream by Simonyan and Zisserman [18]. The network in [20] uses a volume of stacked video frames as input and therefore performs action recognition on raw RGB inputs only. By training the network on the very large noisily labelled dataset Sports-1M [13], the authors were able to train a generic spatio temporal feature extractor, which they call *C3D*. That is, the activations of the network's last fully connected layer can be used as a generic feature representation of the input. An arbitrary classifier can then be trained on this input-representation to perform several different video-based vision tasks such as action recognition, scene classification or object detection.

The work of Karpathy et al. [13] motivated the incorporation of 3D convolutions in *C3D* by providing a comparative evaluation on different fusion strategies, i.e. on how to combine temporal and spatial information in different neural network architectures. Tran et al. [20] additionally conclude, the advantageous property of 3D convolutions, that temporal information is not immediately but gradually condensed along the layer of a 3D CNN.

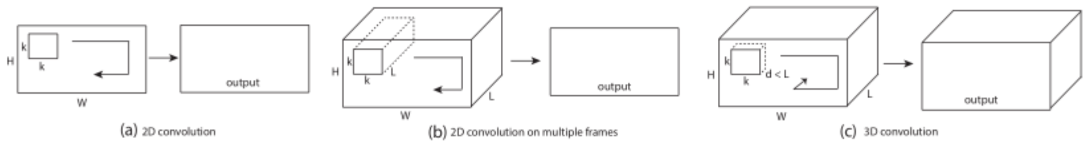


*Figure 13:* Preservation of the temporal dimension by 3D convolutions: *(a)* and *(b)* 2D convolutions on a single or multiple frames results in a 2D feature map without temporal dimension. *(c)* 3D convolutions on multiple input frames preserves the temporal dimension and results in a 3D feature map. [20]

Tran et al. [20] initially perform experiments on the smaller action recognition dataset UCF101 [48], to find the optimal size of kernels for the *C3D* architecture. Results indicate, that kernels of size $3 \times 3 \times 3$ perform best. The final architecture consists of 8 3D convolutional layers, 5 max-pooling layers, two fully-connected layers and a softmax output layer. The architecture and layer sizes are illustrated in figure 14.

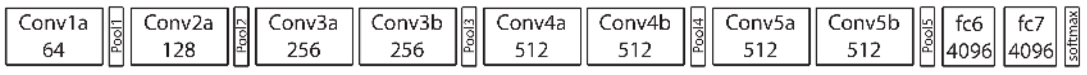


*Figure 14:* C3D architecture. Number of filters is given in each box. [20]

Final results when using a single *C3D* network yield an accuracy of 82.3% on UCF101 [48]. Note that these results were obtained from classifying the activations of the network's last fully-connected layer with an SVM. The network was trained on the very large Sports-1M dataset [13] for this evaluation. Only training the SVM on top of the networks activations uses the UCF101 dataset.

Due to its competitive performance, the *C3D* network is used as a prominent example of a 3D convolutional neural network using only raw RGB inputs. Carreira and Zisserman [46] use it as main architecture to compare their own approach against. Although the results of *C3D* are competitive, the prototypical two-stream approach by Simonyan and Zisserman [18] outperforms *C3D* by yielding 88.0% accuracy on UCF101. Two-stream approaches are therefore considered the more successful architectures in action recognition [44], since they treat the spatial and temporal dimension differently.

## 2.4. Temporal Order Verification

The work of Misra, Zitnick, and Hebert [19] is most related to ours. The authors incorporate *Temporal Coherency* between video frames to learn image representations, which are highly discriminative towards motion and therefore well suited for Action Recognition. Our work is the logical continuation from learning these action representations with 2D CNNs as in [19] to 3D CNNs.

*Temporal Coherency* describes an inherent relation between consecutive video frames. More precisely, consecutive frames are semantically and dynamically correlated [26]. This means in terms of action recognition: Two consecutive frames are highly likely to present the same action and are limited by the laws of physics in the magnitude of change that can occur between them.

*Temporal Coherency* can be used as a weak form of supervision in training deep networks, in contrast to strong supervision regular semantic labels. Misra, Zitnick, and Hebert [19] propose an auxiliary quasi-unsupervised pre-training method which incorporates this *Temporal Coherency* by first training a network on a binary classification task. More precisely: The Network is presented with a sequence of input frames, which may have been randomly permuted, and has to determined if the sequence is in correct temporal order or not. This pre-training task is called *temporal order verification*.

This is, in the strictest sense, not an unsupervised training task, but since obtaining the label (*correct temporal order* or *incorrect temporal order*) is free, the authors attribute it as unsupervised [19]. This has the advantage, that easily obtainable unlabelled data can be used for pre-training the network.

Misra, Zitnick, and Hebert [19] evaluate their approach by presenting three video frames to a triplet siamese CNN with 2D convolutions (see figure 15). Their results show, that using four or five frames did not result in an increase of performance. Datasets utilized for training and evaluation are UCF101 [48] and HMDB51[49].

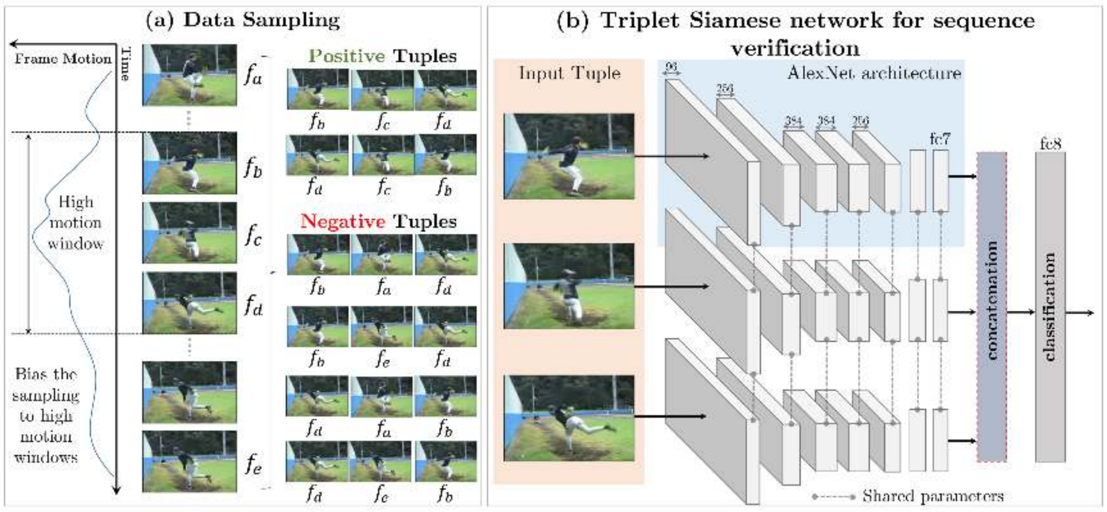


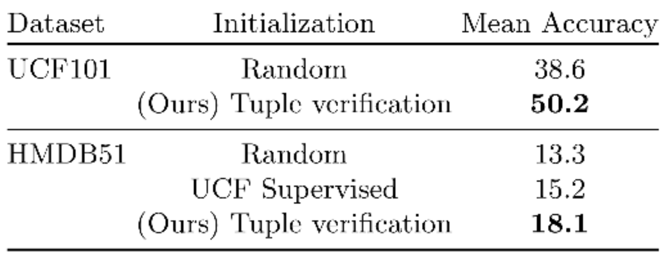
*Figure 15: (Left)* Sampling of input sequences during temporal order verification. Sampling was biased towards regions with motion present in order to not obtain too similar frames. *(Right)* Triplet siamese convolutional neural network, with shared weights. [19]

The authors use optical flow between two frames [50] to identify a high motion window in the input video and construct positive examples (*correct temporal order*) and negative examples (*incorrect temporal order*) from the frames contained in it. This ensures a certain magnitude of motion between the sampled frames and results in frame tuples, which are clearly distinguishable when permuted.

More precisely: Five frames $\{f_a, f_b, f_c, f_d, f_e\}$ are sampled from an input video, with $a < b < c < d < e$. Out of these $f_a, f_b, f_c$ need to lie in the high motion window. $f_b$ and $f_d$ are taken as start and end of the input sequence. To construct a negative example (*wrong temporal order*), $f_e$ or $f_a$ are inserted as middle frame to complete the input sequence. The authors note, that it is vital to keep $f_b$ and $f_d$ as beginning and end of the input, i.e. the correct frames of the middle triple. A mere inversion, i.e. $f_d, f_c, f_b$ is used as positive examples.

The incorporated network is available in the Caffe-Framework [51] and is called CaffeNet, an adapted version of the AlexNet model [52]. The CNN model is arranged as a siamese triplet, with weight sharing. That is: three identical copies of the network model each process one input frame, the results activations of the fully-connected layer are concatenated and then classified by an additional fully connected layer. The individual network models share weights, i.e. each of them receive the same weight updates.

Results are obtained by training the composite model on UCF101 and HMDB51. The model is either trained from scratch (weights are randomly initialized), pre-trained with *temporal order verification* (weights are initialized from a pre-trained model) or pre-trained on UCF101 for the evaluation on HMDB51.



| Dataset | Initialization | Mean Accuracy |
|---|---|---|
| UCF101 | Random | 38.6 |
| | (Ours) Tuple verification | **50.2** |
| HMDB51 | Random | 13.3 |
| | UCF Supervised | 15.2 |
| | (Ours) Tuple verification | **18.1** |

*Table 1:* Perfomance gain in mean accuracy when incorporating temporal order verification as weight-initialization method on UCF101 and HMDB51 [19]

Table 1 shows the increase in final performance after fine-tuning the model on either UCF101 or HMDB51. The increase in performance of +11.6% is most prominent on UCF101. On HMDB51 initializing the model using *temporal order verification* achieves an increase in accuracy of +4.8%, which is smaller than on UCF101 but still significant. Notably initialization from *temporal order verification* outperforms initializing the network from regular pre-training on labeled data.

**Implementational details:**

- The authors did not use any additional unlabeled data for pre-training, but sample around 900k triples for *temporal order verification* from the labeled datasets itself.
- Weights are randomly initialized for the *temporal order verification* pre-training.
- Pre-training is conducted for 100k iterations with a fixed learning rate of $10^{-3}$ and a batch-size of 128 tuples (This results in around 14 epochs in total for a pre-training set of 900k triples).
- Best results were obtained with 25% positive and 75% negative triples.
- Batch normalization [53] is used during training.
- A single CaffeNet model is then initialized from the weights obtained from pre-training the siamese triplet. It is trained for 20k iterations with a batch size of 256 frames on UCF101 and HMDB51.
- The learning rate used for fine-tuning the model $10^{-2}$, which is decayed to $10^{-3}$ after 14k iterations.
- Evaluation results are obtained from averaging random crops and flips from each test-video. First 25 frames are uniformly sampled per video, 5 input-sized regions are cropped out and flipped. This results in a total of 250 inputs for averaging per video (25 frames $\times$ 5 crops $\times$ 2 flips).

# 3. Action Recognition Datasets

The training of action recognition systems, which are applicable to real-world tasks, e.g. by adding vision system to robots or by processing existing video streams from cameras heavily relies on the kind and amount of data that is available for training. This section describes the most relevant datasets for this work. For a more general overview of available dataset in action recognition we refer the reader to the comprehensive survey published by Chaquet, Carmona, and Fernández-Caballero [24] and [10].

## 3.1. UCF101

The UCF101 dataset [48], released by the University of Central Florida in 2012, is the third iteration of action recognition datasets, following UCF50 from 2010 [54] and UCF11 from 2009 [55]. For the compilation of these datasets action videos were sampled from YouTube and therefore contain a variety of different recording conditions such as lighting, backgrounds and scales.

The UCF101 dataset contains $13,320$ action videos, each featuring a single action with a fixed resolution of $320 \times 240$ pixels @ 25fps. The actions are grouped into 101 different action classes, which are illustrated in figure 16 and grouped in the following categories:

- Blue: Human-object interaction
- Red: Body-motion only
- Purple: Human-human interaction
- Orange: Playing musical instrument
- Green: Sports

The UCF101 dataset has besides HMDB51[49] emerged as standard benchmark for general action recognition algorithms. Kay et al. [56] criticise however, that all 13,320 action clips are extracted from only 2,500 sampled source videos and that the dataset therefore lacks the extent and variety modern action recognition datasets require.

## 3.2. Kinetics

The kinetics dataset has recently been released by Kay et al. [56] from Google's DeepMind in 2017. It is promoted as the successor to UCF101 and HMDB51 as it contains $306,245$ action videos sampled from YouTube and therefore provides the previously criticised lack of size and variation.

In contrast to very big action recognition datasets such as Sports-1M[13] and YouTube-8M[14], the actions in kinetics are temporally localized as each video clip is approximately

10 seconds long. The dataset is distributed via YouTube, i.e. annotations can be downloaded which include the action class, a temporal interval and a YouTube ID. The actual videos can then be downloaded from YouTube.

At the time of this writing around 150, 000 videos are obtained from the dataset.

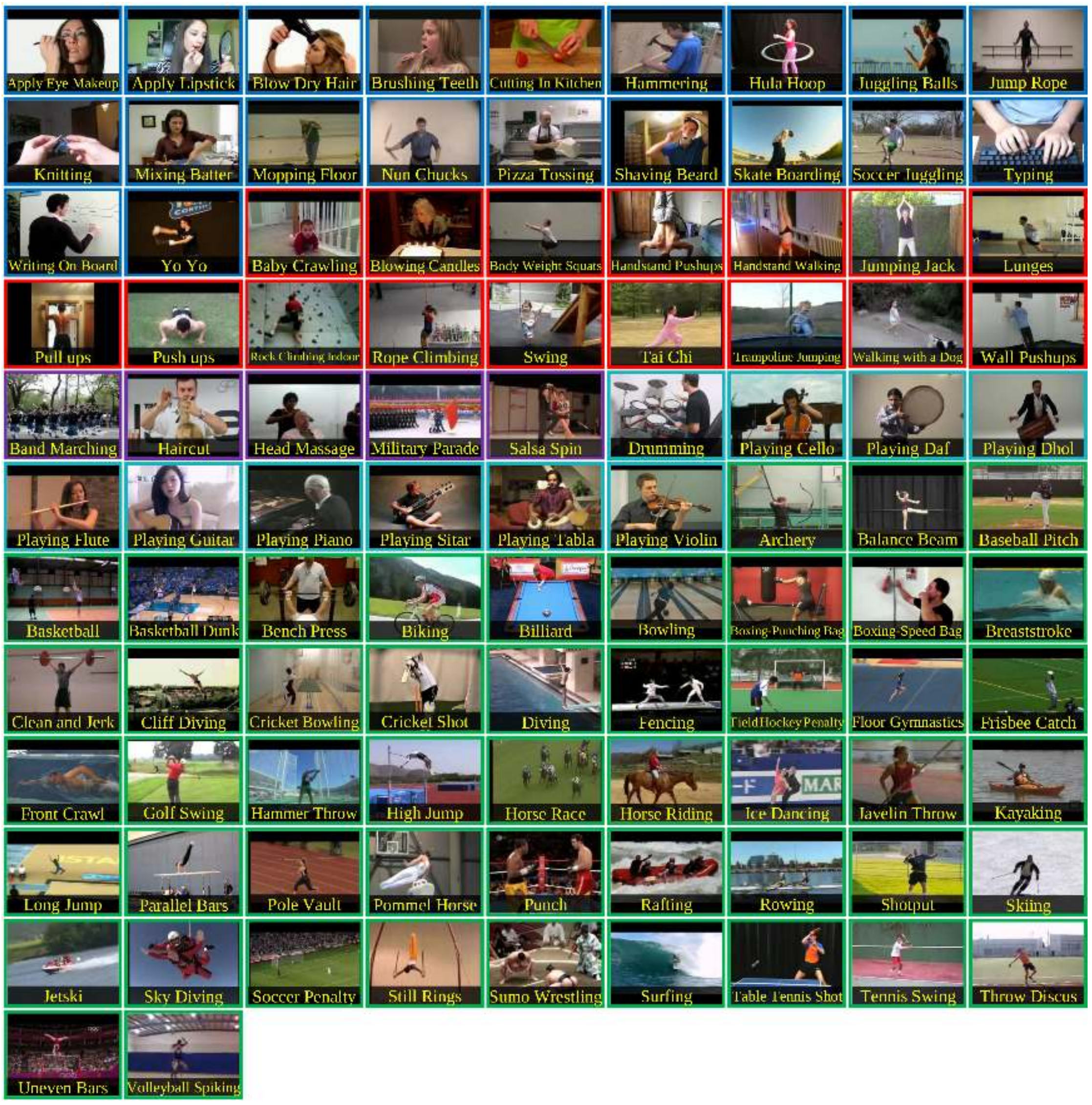


*Figure 16:* The 101 action classes of the UCF101 dataset.[48]

## 3.3. Charades (Activities of Daily-Living)

The creation of the Charades dataset [12] by the Allen Institute of Artificial Intelligence in 2016 addressed common flaws in existing action recognition datasets. As our earlier survey revealed, publicly available action recognition datasets stem from three common sources:

1. Videos are created in controlled environments by filming a finite set of actors performing given actions.
2. Videos are sampled from movies or television broadcasts, and then labeled either manually or using video-metadata.
3. Videos are gathered from the internet, i.e. from video platforms such as YouTube.

Datasets from (1.) usually lack the desired variety needed for enabling proper generalization of machine learning approaches and are comparably small, due to the cost of capturing and labeling such data through a single institution.

Datasets from (2.) and (3.) are heavily biased towards entertaining videos, and therefore provide action samples in rather atypical settings and executions.

The authors note, that it is nearly impossible to find enough videos of people performing *boring* everyday actions to create a vision system applicable in real-world environments.

The charades dataset was therefore created and annotated using a crowdsourcing strategy, which the authors call *Hollywood in Homes*. It relies on Amazon Mechanical Turk (AMT) workers scripting, filming and annotating the videos for the dataset.

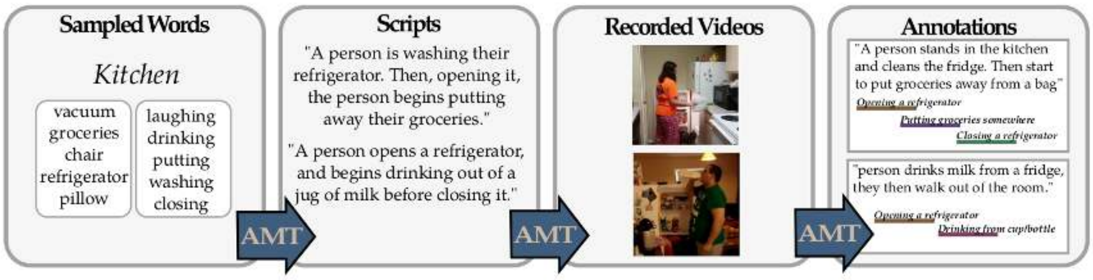

Sampled Words
Kitchen
vacuum
groceries
chair
refrigerator
pillow
laughing
drinking
putting
washing
closing
AMT
Scripts
"A person is washing their refrigerator. Then, opening it, the person begins putting away their groceries."
"A person opens a refrigerator, and begins drinking out of a jug of milk before closing it."
AMT
Recorded Videos
AMT
Annotations
"A person stands in the kitchen and cleans the fridge. Then start to put groceries away from a bag"
Opening a refrigerator
Putting groceries somewhere
Closing a refrigerator
"person drinks milk from a fridge, they then walk out of the room."
Opening a refrigerator
Drinking from cup/bottle

*Figure 17:* Processing stages to create the Charades dataset. `AMT` denotes the incorporation of worker from the Amazon Mechanical Turk service. [12]

Figure 17 illustrates creation of the dataset in multiple stages:

1. **Sampled Words → Scripts**
   By analyzing a total of 549 Movies focussing on the 15 most common indoor rooms of domestic households, the authors derived a list of 40 objects and 30 actions which occurred most often in these scenes. The names of five randomly selected objects, five randomly selected actions and one randomly selected room were given to an AMT worker to combine into realistic activities scripts, that could occur in this room.

2. **Scripts → Recorded Videos**
   The previously created activity scripts are given to AMT workers to act out and record the described series of actions in an approximately 30 seconds long video. Analyzing the previously created activity scripts, the most frequent (verb, proposition, noun) triplets could be grouped into 157 action classes.

3. **Recorded Videos → Annotations**
   AMT workers annotate the created videos in multiple ways: Short description of the observed actions in text, identification of the present action classes, temporal localization of the observed action classes and identification of the present objects.

This crowdsourcing-based approach results in 9,848 videos created by 267 different people over three continents. In total, the datasets contains $66,500$ temporally localized individual action instances. Thereby each video contains on average 6.8 videos.

Figure 18 shows example frames from the charades dataset in comparison to frames from YouTube videos. It demonstrates the atypical and entertaining nature of examples for action classes on YouTube.

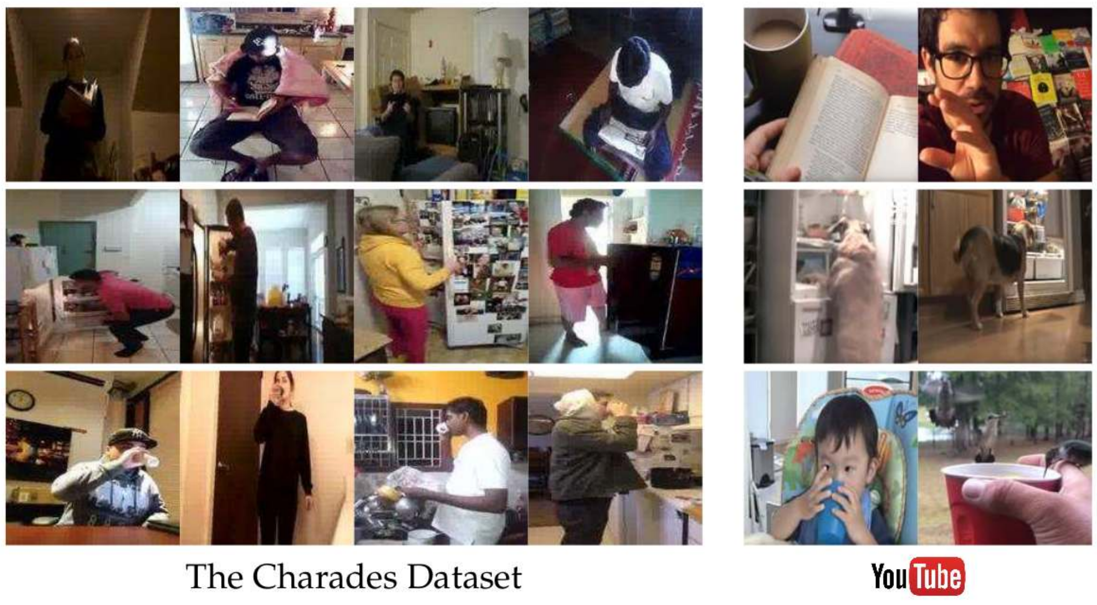


*Figure 18:* Rows show example frames of the actions *reading*, *opening fridge* and *drinking*. Left: Frames from videos in the charades dataset. Right: Frames from sampled YouTube-videos. This illustrates the bias of YouTube videos towards entertaining actions and unrealistic settings. [12]

### 3.3.1. Baseline Approaches on Charades

Additionally to releasing the dataset, several approaches for action recognition were implemented by the authors and evaluated on the Charades dataset. These experimental results were obtained to provide baselines for comparing future approaches on the dataset. The following approaches were implemented:

**Random** Presented with an input video, the classifier outputs a random action label.

**C3D** The authors utilize the *C3D* feature extractor introduced and published by Tran et al. [20]. It is obtained by training a deep 3D convolutional network on the Sports-1M video dataset [13] and using the activations of the last convolutional network as a generic video-feature extractor. These features are then classified using one-versus-rest SVMs.

**AlexNet** Deep 2D CNNs, AlexNet [52] and VGG-16 [57] are being trained on a large collection of object images and extract frame-wise features over 30 equidistant frames of an input video. These features are averaged, normalized and then classified with one-versus-rest SVMs.

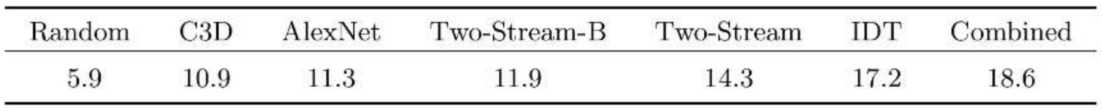
**Two-Stream** The VGG-16 architecture is incorporated in a two-stream setup as introduced by Simonyan and Zisserman [18]. The spatial-stream network is pre-trained on ImageNet [15] and the temporal-stream network is pre-trained on UCF101 [48]. Both streams are then fine-tuned by training *conv4*, *conv5* and the fully-connected layers only.

**Two-Stream-B** To handle class imbalance in the dataset, training of the previously described two-stream approach is adapted by ensuring that each training mini-batch of 256 examples contains at least 50 unique classes.

**IDT** The authors implement the *Improved Dense Trajectory (IDT)* approach [35], which relies on hand-crafted feature extractors. Spcifically MBH, HOG and HOF descriptors are being extracted around sampled trajectories in the input video. These descriptors are being encoded with Fisher Vectors [58], which are then classified using one-versus-rest SVMs.

**Combined** Represents the result of combining all previous approaches by late fusion [13].

Because multiple action labels can be present per time-segment in a video of the dataset (see next section 3.3.3), the authors evaluate the performance of their implemented baseline approaches in terms of mean average precision (mAP). For a discussion of mean average precision, which is a performance measure usually used in information retrieval, we refer the reader to [59]. The results are shown in figure 19.



| Random | C3D | AlexNet | Two-Stream-B | Two-Stream | IDT | Combined |
|---|---|---|---|---|---|---|
| 5.9 | 10.9 | 11.3 | 11.9 | 14.3 | 17.2 | 18.6 |

*Figure 19:* Mean average precision of baseline approaches evaluated on the Charades dataset [12]

### 3.3.2. Recent Advances on Charades

Alongside the publication, that describes the Charades dataset itself and the performance of various baseline approaches [12], the authors additionally published an approach, proposing a fully-connected temporal Conditional Random Field (CRF) for action and intention recognition [60]. The incorporates various aspects of a human activity by modeling a video as a sequence of random variables, which capture the overall action category, present objects, the currently performed atomic action, the overall progress and the scene. Human intention is incorporated as another unobserved latent variable. For each frame in the input video, a two-stream CNN predicts the CRFs potentials. The authors were thereby able to beat the baseline approaches by yielding an mAP of 22.4%.

Accompanying the conference on Computer Vision and Pattern Recognition (CVPR) in July 2017, the Charades challenge for action recognition and action localization was posed. Winner in both the localization and recognition track were TeamKinetics, lead by João Carreira, using their I3D approach [46]. Their approach yields 34.41% mAP for action recognition with 29.74% and 28.11% being the second and third best results. (Results obtained from `http://vuchallenge.org/charades.html`)

### 3.3.3. Labelling of the Charades Dataset

The Charades dataset is richly labeled, in order to enable researcher to perform a variety of tasks on the dataset. These tasks include scene/action/object recognition, action localization and action description. The labels are provided in the form of comma separated values in two files (one for training, one for testing). The example in table 2 corresponds to a single line in the training-labels file.

The Charades dataset is available for download in different resolutions (480p and original resolution as provided by the AMT workers), decoded as RGB-frames and as optical flow images. The different versions of the dataset are directly available for download at

`http://allenai.org/plato/charades/`

| | |
|---|---|
| id | RXLKF, |
| subject | P6LJ, |
| scene | Living room, |
| quality | 4, |
| relevance | 5, |
| verified | Yes, |
| script | A person is reading a book while snuggling on the sofa. They put the book down and begin standing while holding a cup of coffee, |
| objects | blanket;book;coffee;couch;cup;glass;sofa;tissue, |
| descriptions | "Person walks to couch wearing blanket, grabs book, place on lap, straighten blanket, opens book, reads it, closes book, place on couch, gets up, grabs tissue, clean face, place tissue back;The girl is walking to the couch with a blanket around her.  She sits down, picks up a book, and begins reading.  Then, she puts down the book and stands, takes a drink, puts it down, and walks back to the camera.", |
| actions | c025 23.80 28.70; |
| | c028 25.50 30.20; |
| | c106 29.50 36.30; |
| | c026 4.30 30.20; |
| | c029 15.00 26.20; |
| | c032 6.40 28.70; |
| | c107 27.00 37.30; |
| | c070 0.00 39.00; |
| | c027 5.40 15.20; |
| | c030 0.00 3.30; |
| | c123 1.80 9.40; |
| | c110 27.00 32.70; |
| | c109 27.60 32.70; |
| | c151 2.60 9.20; |
| | c072 0.00 4.10, |
| length | 38.08 |

*Table 2:* Example labelling for a video of the training-set in the Charades dataset. [12]

Annotations and evaluation scripts in MATLAB to calculate the mean average precision (mAP) for the outputs of a model are provided as well. We note, that this form of providing the dataset is most comfortable, as it is not needed to download individual videos from YouTube as with ActivityNet[61], Kinetics[56] and YouTube-8M.

The documentation of the dataset describes the individual fields of the annotations as follows:

| | |
|---|---|
| `id` | Is a unique identifier for each video. |
| `subject` | Describes an unique identifier for each subject, i.e. the person performing the action, in the dataset |
| `scene` | One of the 15 domestic indoor scenes, i.e. rooms in the dataset. Examples are kitchen, dining room, living room or basement. |
| `quality` | Indicates the quality of the video judged by an AMT annotator (7-point scale, 7=high quality). |
| `relevance` | Indicates the relevance of the video to the script judged by an AMT annotator (7-point scale, 7=very relevant). |
| `verified` | Describes if an annotator successfully verified that the video matches the script. |
| `script` | The human-generated script, given to AMT workers to generate the video. |
| `descriptions` | Semicolon-separated list of descriptions by annotators watching the video. |
| `actions` | Semicolon-separated list of (class,start,end) triples for each action in the video. For example in table 2: `c025` describes the action *closing a book*, which lasts from second `23.80` to second `28.70` in the video. |
| `length` | The length of the video in seconds. |

*Table 3:* Descriptions of fields in the video annotations for the Charades dataset [12]

As can be seen in table 2, annotated actions may overlap in videos of the charades dataset. This is due to the facts, that humans are able to multitask and some actions naturally happen concurrently such as *sitting on a chair* and *working at a table.* In this sense, the Charades dataset deviates from general action recognition benchmarking datasets, such as UCF101[48] HMDB51[49] and Kinetics[56]. These provide unambiguous video clips, which are labelled with a single action to be recognised. Charades is therefore more realistic, by providing less but longer video-files that contain multiple actions which can overlap.

Input video-clips for training a classifier can therefore be labelled in different ways: Either by providing only a single label with each input clip (one-hot-encoding) or by providing multiple labels for each clip (n-hot-encoding). The latter would result from sampling an input clip from a region of overlapping actions.

Multiple labels being possible for the same input clip results in a challenge when evaluating the performance of the network, since usually only a single output (the one with the biggest activation value) is taken as the network's prediction. Therefore simply calculating the classifiers accuracy with a single target label per input is not an adequate measure of performance any more. The mean average precision [59] was therefore established as the standard performance measure on the dataset.

# 4. Experimental Setup

The main question this work aims to answer is: can pre-training with temporal order verification improve action recognition recognition performance from raw RGB inputs without explicit calculation of optical flow in a single stream 3D convolutional neural network. Specifically we try to develop a temporal order verification pre-training method for 3D CNNs, similar to the original approach in [19]. This section describes the overall details of the implementation of the *C3D* model [20].

## 4.1. Hardware Setup

We perform experiments on a single PC with two GPUs.

| | |
|---|---|
| Processor | Intel Core i7 950 with 3.07 Ghz clock speed |
| Cores / Threads | 4 / 8 (Hyperthreading) |
| Main memory | 12 Gb DDR3 RAM with 1067Mhz |
| Mass storage | 1 Tb (HDD) |
| GPU | 2× Nvidia GeForce GTX 1070 |
| GPU Memory | 2× 8Gb |

*Table 4:* Overview hardware setup

## 4.2. Model Architecture

Our implementation of the *C3D* model follows the architectural design of the original *C3D* network, as described in [20] and in the published Caffe implementation (see `https://github.com/facebook/C3D`). We implemented our model in Python using Google's deep learning framework *Tensorflow* [62]. The *C3D* network contains eight 3D convolutional layers, with max-pooling layers after the first, second, fourth, sixth and eighth layer. An illustration of the network model is given in figure 20. After the last pooling layer, the temporal information is fully collapsed and the feature maps two-dimensional. These are then flattened into a one-dimensional vector to be further processed by the fully-connected layers.

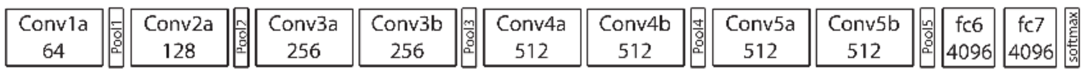


*Figure 20:* Illustration of the C3D architecture. An additional flattening operation is performed after *Pool5*, which is not illustrated in this figure. [20]

### 4.2.1. Architectural Details

According to the experiments conducted by Tran et al. [20], 3D convolutional kernels of size $3 \times 3 \times 3$ perform best in the *C3D* model. We therefore follow this design decision in our implementation as well. The network processes inputs of spatial dimension $112 \times 112$ pixels and temporal length of 16 frames in 3 RGB channels. The dimension of a complete input volume therefore is $16 \times 112 \times 112 \times 3$. The 3D filters in the convolutional layers *conv1a* - *conv5b* are applied to their inputs with stride 1 and zero-padding is applied. The convolutional layers therefore do not reduce the dimensionality of the inputs in neither spatial nor temporal dimension.

All pooling layers, except the first one, perform $2 \times 2 \times 2$ (*width* $\times$ *height* $\times$ *temporal depth*) 3D max-pooling with strides two. The pooling layers thereby reduce the spatial and temporal dimension of an input by 2. The first pooling layer only performs spatial pooling, i.e. with a pooling kernel of dimension $2 \times 2 \times 1$, in order to not collapse the temporal information too early as done in [20]. Table 5 holds the output dimensions of each layer along with the number of trainable weights and biases.

| Layer Name | Output Dimension | Trainable Parameters |
|---|---|---|
| **Conv1a** | $16 \times 112 \times 112 \times 64$ | $3 \cdot 3 \cdot 3 \cdot 3 \cdot 64 + 64 = 5,248$ |
| **Pool1** | $16 \times 112 \times 112 \times 64$ | 0 |
| **Conv2a** | $16 \times 56 \times 56 \times 128$ | $64 \cdot 3 \cdot 3 \cdot 3 \cdot 128 + 128 = 221,312$ |
| **Pool2** | $8 \times 28 \times 28 \times 128$ | 0 |
| **Conv3a** | $8 \times 28 \times 28 \times 256$ | $128 \cdot 3 \cdot 3 \cdot 3 \cdot 256 + 256 = 884,992$ |
| **Pool3** | $4 \times 14 \times 14 \times 256$ | 0 |
| **Conv4a** | $4 \times 14 \times 14 \times 512$ | $256 \cdot 3 \cdot 3 \cdot 3 \cdot 512 + 512 = 3,539,456$ |
| **Conv4b** | $4 \times 14 \times 14 \times 512$ | $512 \cdot 3 \cdot 3 \cdot 3 \cdot 512 + 512 = 7,078,400$ |
| **Pool4** | $2 \times 7 \times 7 \times 512$ | 0 |
| **Conv5a** | $2 \times 7 \times 7 \times 512$ | $512 \cdot 3 \cdot 3 \cdot 3 \cdot 512 + 512 = 7,078,400$ |
| **Conv5b** | $2 \times 7 \times 7 \times 512$ | $512 \cdot 3 \cdot 3 \cdot 3 \cdot 512 + 512 = 7,078,400$ |
| **Pool5** | $1 \times 4 \times 4 \times 512$ | 0 |
| **Flattening Layer** | 8192 | 0 |
| **fc6** | 4096 | $8192 \cdot 4096 + 4096 = 33,558,528$ |
| **fc7** | 4096 | $4096 \cdot 4096 + 4096 = 16,781,312$ |
| **softmax** | #classes in dataset | $4096 \cdot$ #classes + #classes |

*Table 5:* Output dimension and number of trainable parameters per layer in our *C3D* model. The flattening layer collapses the feature maps in **pool5** into a one dimensional vector.

The total number of parameters in the network adds up to $77,995,776$ plus the parameters residing in the last softmax output layer, whose amount depends on the number of output classes in the used dataset. We report final results on the UCF101 and Charades dataset. The number of total parameters for these datasets are given in the following table 6:

| Dataset | #classes | #Parameters (last layer) | #Parameters (total) |
|---|---|---|---|
| UCF101 | 101 | $4096 \cdot 101 + 101 = 413,797$ | **78,409,573** |
| Charades | 157 | $4096 \cdot 157 + 157 = 643,229$ | **78,639,005** |

*Table 6:* Total number of parameters in our implemented *C3D* model according to the number of classes in the target dataset.

As can be seen in table 6, the number of total parameters is two orders of 10 bigger than the number of parameters in the output layer. Most of the networks parameters, namely around 64% are located in the last two fully-connected layers. This is illustrated in figure 21 and common in architectures with fully-connected layers, as they naturally contain more parameters than convolutional layers. In recent image classification approaches with CNNs [63], the number of parameters is heavily reduces by substituting last fully connected layers. We keep the architecture unchanged in order to compare our results, more precisely the effect of *temporal order verification* on 3D convolutional networks, to the results of Tran et al. [20] and Carreira and Zisserman [46].

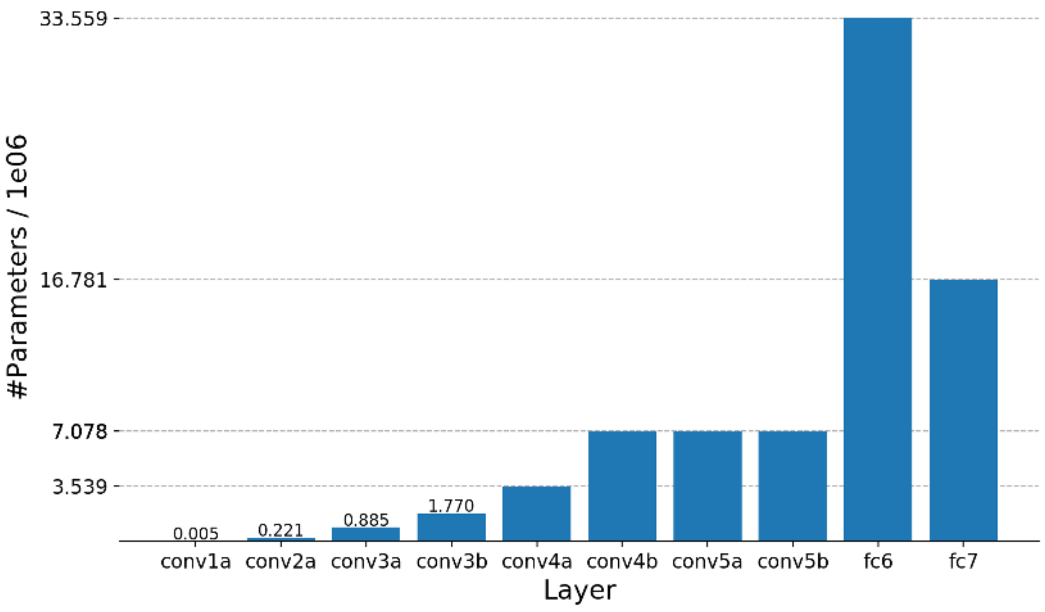


*Figure 21:* Distribution of parameters in our implementation of the *C3D* model. Parameters in the output layer are not included, because they depend on the number of output-classes.

### 4.2.2. Regularization Methods

As around half of the model parameters are located in the last fully-connected layers, these are particularly prone to overfitting. We therefore follow the original implementation of the *C3D* model and incorporate Dropout[64] with a dropout rate of 0.5 in those last two layers. Dropout has not been used in the convolutional layer, since the number of parameters in these layers is significantly smaller. In order to provide regularization for the other parameters as well we incorporate L2 regularization [65]. Batch Normalization [66] has been shown to be non-trivial to implement in a multi-GPU setup and incorporating it would have exceeded to scope of this work. The original implementation of the *C3D* model did not use it, Carreira and Zisserman [46] however implemented it a multi-GPU setup to compare their own approach with *C3D*.

The main data-augmentation technique used in our experiments is random cropping of inputs (see section 4.5).

## 4.3. Input Pipeline

A single network training step in our multi-GPU setup, i.e. processing an input batch and updating the model parameters as described in section 4.4, takes longer than preparing a single input batch.

This is due to the following reasons:

- Common sizes of action recognition video-datasets make it impossible to completely keep them in main memory. Therefore source videos need to be loaded from slower mass storage, in order to be processed further.
- Videos are encoded to reduce their file-size and need to be decoded to obtain the frame-by-frame RGB-values. Pre-decoding would heavily increase the dataset size and exceed the storage limit of our hardware.
- Source videos usually have a higher spatial resolution than necessary as network input and need to be rescaled.
- It is common practise to perform data-augmentation, i.e. to artificially increase the number of unambiguous network inputs by manipulating the source data, usually through cropping, flipping or rescaling.

Therefore, even if the input-batch generation and model training would run in parallel in two different threads, the GPUs would not receive enough data to fully unlock the potential training speed-up of a multi-GPU setup. To mitigate this problem, we implement a multi-threaded input pipeline, which can easily be scaled to more powerful CPUs (more processing cores) and is illustrated in figure 22.

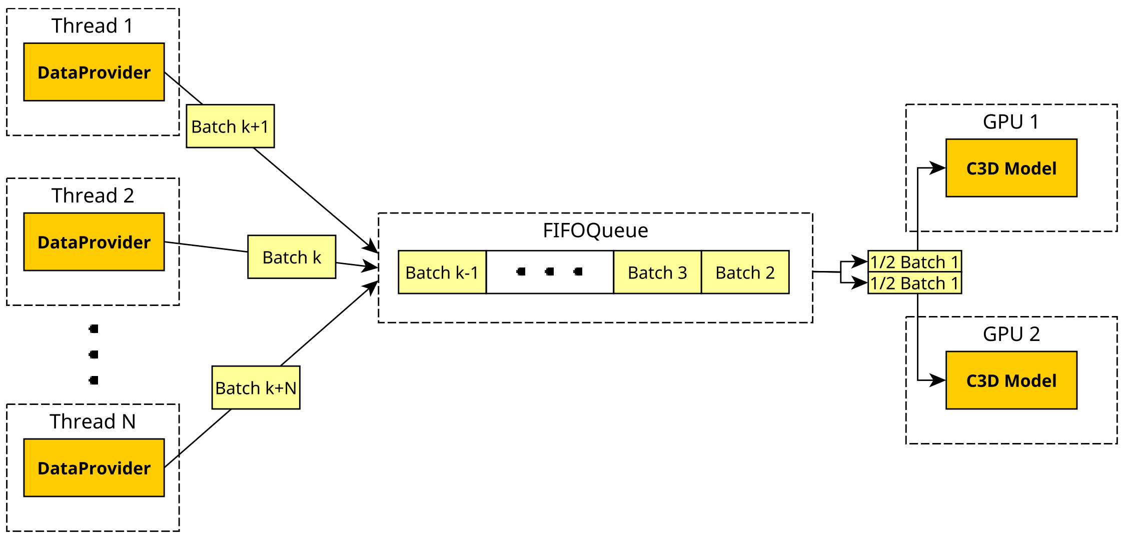


*Figure 22:* Illustration of the multi-threaded input pipeline. A `DataProvider` instance prepares input batches in parallel to provide enough data for the GPUs. Finished batches are enqueued asynchronously into a FIFO queue which then feeds the GPUs. Batches contain input video clips as well as training labels.

An arbitrary number of threads can be created to asynchronically preprocess input batches. The maximum reasonable number of threads depends on the number of CPU cores available on the target hardware. Each thread has a reference to a single `DataProvider` instance, which encapsulates and provides the necessary pre-processing operations to produce an input batch in a single method (`get_next_training_batch()`). This method accesses a thread-lock to ensure synchronization between the threads, i.e. no thread will pre-process the same video during an epoch[6]. As soon as the end of an epoch is reached, that is when one thread encounters the end of the training set, all other threads wait until the last batch is processed. The training set is then randomly shuffled and the next epoch begins.

The batches get enqueued into a single FIFOQueue which resides in the main memory. An input batch is then dequeued from the queue whenever necessary, i.e. after the previous training step completed on the GPUs. Several kinds of queues as well as appropriate mechanisms to asynchronically enqueue and dequeue data are supported in the *Tensorflow* framework [62].

The FIFOQueue represents as a buffer between the mass storage and the data-processing GPUs. The currently enqueued number of batches thereby provides an accurate measure of the data flow, i.e. if more than one batch is enqueued during the overall training process, then the GPUs never had to wait for data. Only in this case does the training process fully benefit from a multi-GPU setup.

[6]An epoch describes a complete iteration through the training set.

## 4.4. Multi-GPU Gradient Descent

Incorporating multiple GPUs for parallel training of a deep neural network is not a trivial task and there are different ways to do so. Parallelising network training with multiple GPUs can be divided into the following main approaches:

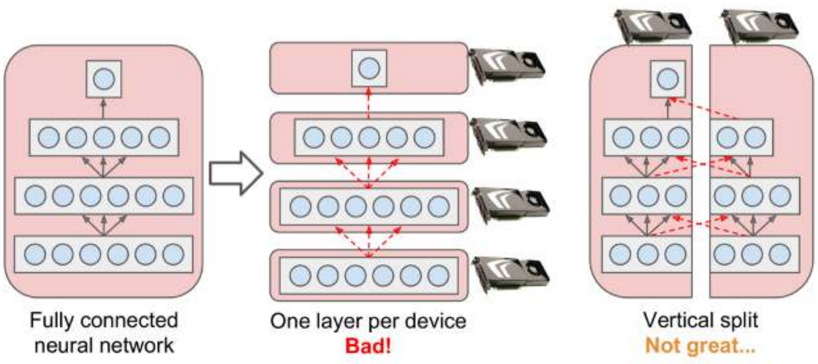
1. Model Parallelism
2. Data Parallelism
    a) Asynchronous model updates
    b) Synchronous model updates

### 4.4.1. Model Parallelism

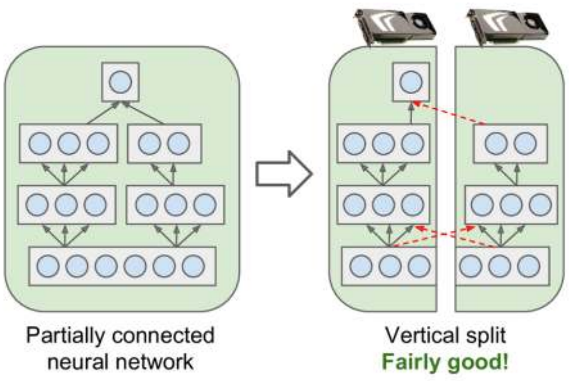
Model parallelism describes dividing the to be trained network model into multiple parts and calculating each of these parts on a separate device. Parallelizing a neural network that way turns out to be difficult, since the model needs to be split in a way to reduce cross-GPU communication as much as possible. Particularly in fully-connected networks, splitting the model results in a lot of cross-device communication and generally no speed-up can be obtained from this approach. The advantage of model parallelism therefore heavily depends on the network architecture itself. Convolutional neural networks contain locally connected patches of neurons and are therefore easies to split, i.e. with less resulting cross-device communication.

The most intuitive way of splitting a network model horizontally, results in no parallelization speed-up at all, because the layers on GPU $n$ always have to wait for the outputs of GPU $n-1$.

Figure 23 illustrates the above discussed alternatives to parallelize a single neural network model.



*(a)* Splitting a fully-connected network either layer-wise (*middle*) or vertically (*right*).



*(b)* Splitting a partially-connected network vertically.

*Figure 23:* Different configurations to parallelize a single neural network model across multiple GPUs [67]

### 4.4.2. Data Parallelism

Another way to use multiple GPUs for network training is to split the input batch rather than the network model, and process these sub-batches by replicated network models across multiple GPUs in parallel. This approach is called data parallelism and each available GPU hosts an exact replica of the model that is to be trained. An additional set of model weights is kept in the main memory to provide synchronization for the models before each new training step. Figure 24 provides an illustration of data parallelism in a multi-GPU setup.

Data parallelism hence yields additional computational overhead, because the shared model weights in the main memory need to be updated. Since each model processes a sub-batch individually on its host GPUs, each model also calculates an individual set of weight updates, which needs to be communicated and merged with the weights in the main memory. Since each GPU only processes a smaller sub-batch, the initial batch size can be increased. This can be done in two ways: *synchronically* or *asynchronically.*

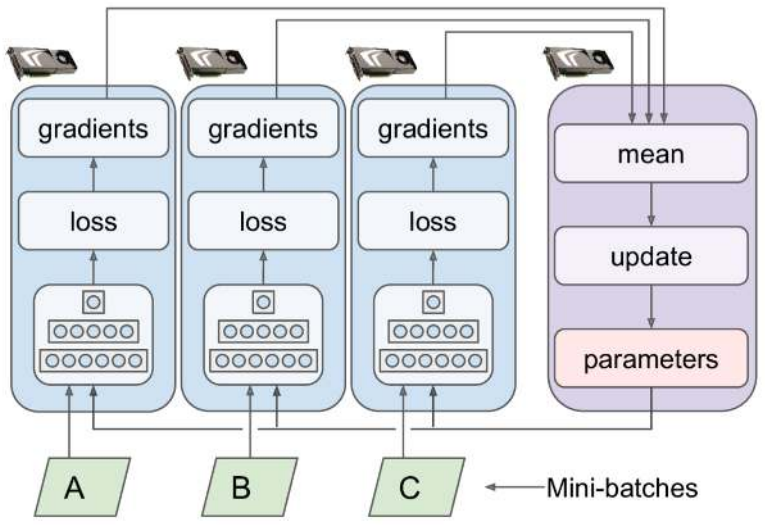


*Figure 24:* General overview of multi-GPU training with data parallelism. [67]

When updating the shared model weights *synchronically*, the averaging step waits for all GPUs to finish the calculation of their individual gradients. The averaged weight updates are then applied to the shared model weights, which are afterwards communicated back to the GPUs and thus synchronize the models to a homogeneous weight-state. The models then proceed to process the next set of sub-batches.

Synchronous weight updates have the disadvantage, that the overall weight update has to wait for the slowest GPU to finish its computation.

Another option is to perform *asynchronous* updates of the shared model parameters. In that case, the gradients of a model are immediately used for updating as soon as a GPU finished its computation. This averts the averaging delay and waiting for the slowest GPU. However asynchronous updates yield the problem of "stale" gradients [67], i.e. whenever an update according to a GPU's gradients is issued, the shared weight may have been updated in the meantime and the current updates may not be beneficial anymore.

Benchmarking experiments conducted by Google Brain [68] indicate, that data parallelism with synchronous weight updates performs best. Since our *C3D* model contains locally connected convolutional layers as well as large fully-connected layers, it is difficult to split, i.e. to incorporate model parallelism for training. Additionally, video-inputs are naturally bigger than images. Since the amount of available memory on the GPUs limits the size of the input batches, training with data-parallelism is tempting. We therefore decide to perform data parallel mini-batch gradient descent.

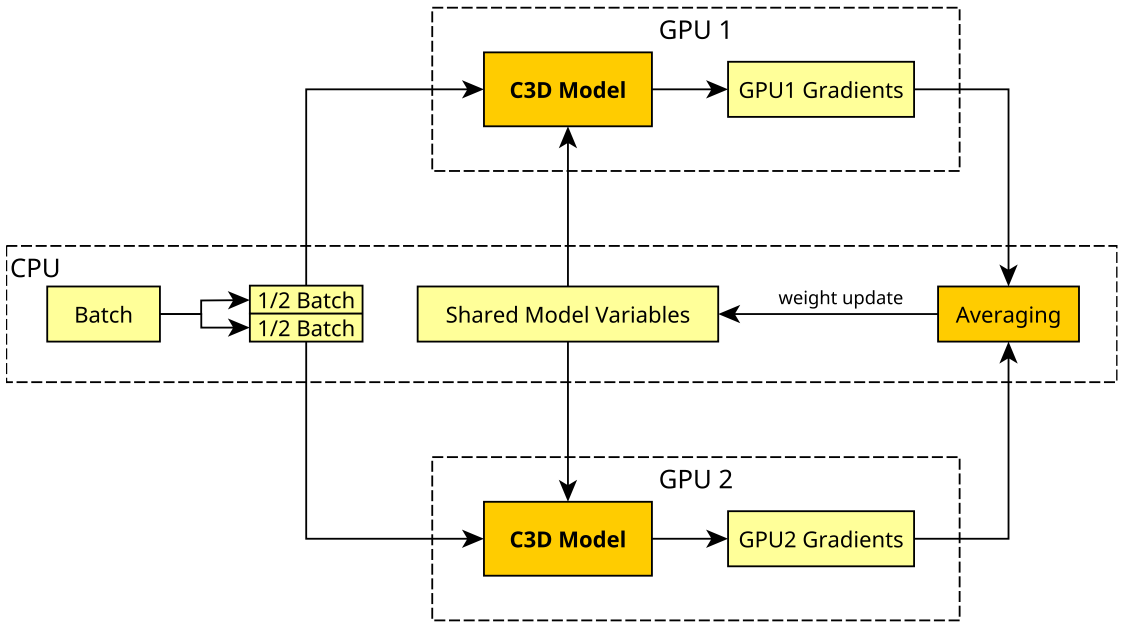


*Figure 25:* Diagram of a single weight update step during data parallel synchronous gradient descent.

## 4.5. Input Sampling

Our *C3D* model trains on input clips with a length of 16 frames per step, yet most source videos provide more frames, i.e. have a longer temporal duration. A sampling strategy to provide the network with properly sized inputs ($112 \times 112 \times 16$ pixels) is therefore needed. This primarily includes temporal cropping, that is selecting a 16 frame long sub-clip from the full temporal extend of the video. We additionally incorporate spatial cropping: Source videos are initially rescaled to a larger spatial resolution and frames with the necessary width and height are cropped out to produce properly sized network inputs.

### 4.5.1. UCF101 Input-Clips

Videos in the UCF101 dataset are provided in a fixed resolution of $320 \times 240$ pixels (width $\times$ height) with an average length of $7.21s$ [48]. Each video, in the training- as well as in the test-set, are annotated with a single action label per video. During training we iterate over the list of training videos and sample one input clip for each of those training-videos. The number of finished training epochs therefore corresponds to the overall number of processed input-clips per source video.

To reduce pre-processing time per training step, only a 16 frame long time-interval is decoded from the source training video per sample. For training on UCF101, this interval is randomly sampled from the complete training video, since it only contains one single action. The resulting 16 frames are then rescaled to a fixed spatial resolution of $160 \times 120$ pixels, i.e. the spatial resolution is reduced by a factor of 2. The final network input is then cropped out by randomly selecting a $112 \times 112$ pixel wide region in the first frame and cropping the 15 following frames accordingly.

Following this procedure a multitude of different network inputs can be generated from a single source video. We use a batch size of 40 per training step, which results in 20 clips per GPU during training. This was found to be the biggest possible number of inputs to not exceed memory limitations of the available GPUs.

### 4.5.2. Charades Input-Clips

The videos in the Charades dataset are provided in multiple different resolutions. To reduce the download time and dataset size, the $480p$-version of the dataset was downloaded, which contains videos with a maximal height of 480 pixels. In contrast to UCF101 a single source video is annotated with multiple action instances, which possibly can overlap (see section 3.3.3). To incorporate training on the Charades dataset in our implemented pre-processing pipeline, we treat each annotated action instance as a uniquely labelled video, from which input-clips can be sampled.

During training on the Charades dataset we iterate over the list of temporally localized action instances and sample one input-clip per instance and epoch. To reduce pre-processing time, only a 16 frame long interval is decoded from each action instance in the training set. Analogously to the input sampling procedure on UCF101 this interval is selected randomly. Since the videos have different spatial resolutions, we rescale the shorter side of the previously sampled interval to 120 pixels and spatially crop consecutive $112 \times 112$ pixel wide regions.

The batch size of 40 input-clips per step, i.e. 20 inputs per GPU is left unchanged for training on the Charades dataset.

### 4.5.3. 3D Temporal Order Verification

Since obtaining the labels *correct temporal order* and *incorrect temporal order* requires no manual data labelling (see 2.4), *temporal order verification* can be seen as an unsupervised pre-training method. Therefore also a large amount of unlabelled data can be used in the pre-training process, as long it contains videos of (preferably human) motion.

Promising candidate datasets are ActivityNet, because it contains a big number human actions, which unfortunately are noisily annotated and not temporally localised as well as Kinetics, which provides a big number of temporally localized actions. These datasets however are only distributed via YouTube, i.e. each video needs to be downloaded individually. At the time of this writing about half of the training-set of the Kinetics dataset, i.e. around $150,000$ videos were obtained. To harvest the advantages of incorporating more data than the target dataset contains, the partially downloaded Kinetics videos were used for pre-training under the assumption that benefits will already be apparent and can be improved by incorporating the full dataset later-on.

Successful pre-training with *temporal order verification* requires providing inputs, which are clearly distinguishable when permuted randomly as described by Misra, Zitnick, and Hebert [19]. This means in practise for our *C3D* model, that a sampled input clip needs to contain a certain amount of motion in order to be beneficial for pre-training. A video clip with only a small magnitude of motion looks nearly the same, when permuted.

We investigate three different methods to transfer the sampling of positive and negative inputs from the original *temporal order verification* approach as illustrated in figure 15 to our 3D convolutional network.

### Method 1

A regular network input of 16 frames is sampled from a video as described above. The frames are then divided into three similarly sized chunks $a$, $b$ and $c$. One of the chunks contains six video frames, which one is determined randomly. Figure 26 illustrates the approach with chunk $b$ containing 6 frames.

In order to provide a certain amount of motion in the sampled frames, the mean frames of each chunk are compared. The difference between mean frames is used as a measure of change in the 16 frame long clip. We empirically determine a threshold to filter out clips with a small amount of motion. To generate a negative example (*incorrect temporal order*) the order of chunks is permuted. For example: $(b, a, c)$ corresponds to a negative example. To generate a positive example (*correct temporal order*) the order of chunks is left unchanged.

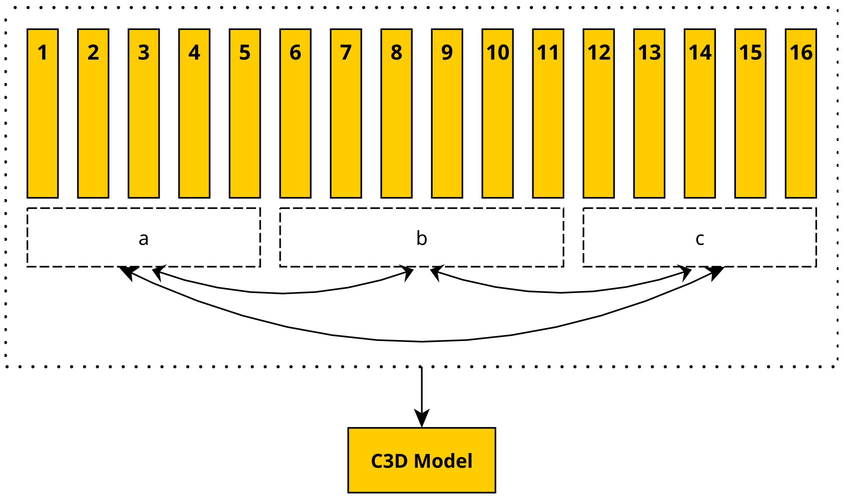


*Figure 26:* Grouping and permutation of input frames for generating positive and negative examples during pre-training of the *C3D* model.

**Method 2**

We hypothesize that the previously described *Method 1* may be flawed, since non-continuous frame transitions, i.e. cuts from permuting the frame chunks only occur at distinct frame regions. A network trained to recognize these permutations could therefore overfit to just focus on these regions.

To alleviate this problem, another sampling method that allows cuts between every frame transition has been designed. The authors of the original *temporal order verification* approach [19] keep the first and last frame in place and only change the middle frame to generate a negative example in *incorrect temporal order*. To incorporate this, a number of frames $a$ in the beginning and end of the input clip is uniformly sampled between 1 and 3. These frames are kept unchanged, as illustrated in figure 27.

The remaining middle $16 - 2 \cdot a$ frames are divided into $n$ chunks, where $n$ is uniformly sampled between 2 and 5. These $n$ chunks are then randomly permuted to create a negative example.

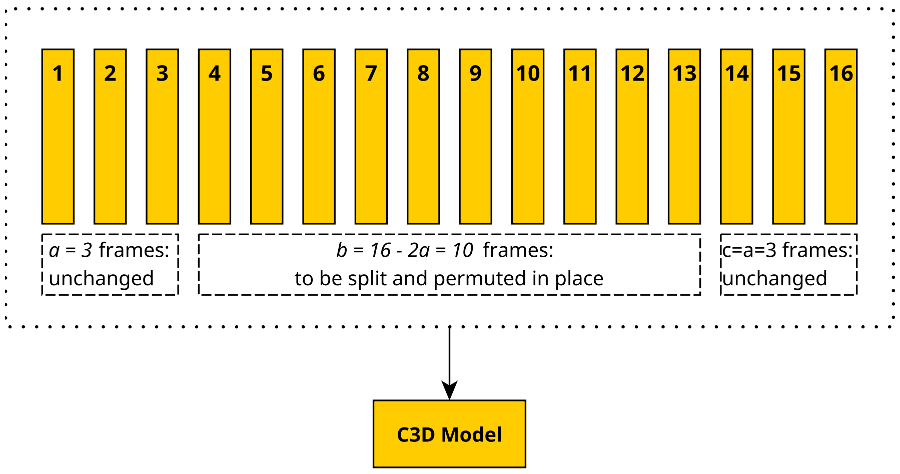


*Figure 27:* Permutation of input frames to generate a possible cut between every input frame.

**Method 3**

A third approach we evaluate is most analogous to the original approach by Misra, Zitnick, and Hebert [19]. Instead of feeding a single permuted clip into a single *C3D* model, three input clips whose temporal relation amongst each other is permuted are fed into a triplet siamese *C3D* model. This triplet model is generated by incorporating three *C3D* models with shared weights and no individual output layers. The activations of the last fully-connected layers are concatenated and then classified by a final softmax output-layer into the two-classes *correct temporal order* and *incorrect temporal order*.

In total 64 consecutive frames, i.e. four complete *C3D* inputs are sampled from a training video. In figure 28 these four 16-frame inputs are denoted as $a$, $b$, $c$ and $d$. To create a negative example the four individual inputs are left unchanged and their overall order is permuted before being fed in the triplet siamese network.

Specifically, a negative input can be generated by choosing either $(a, d, c)$ or $(b, a, d)$ as input for the network triplet, i.e. the first and last input are again in their correct place. Positive inputs can be generated by choosing $(a, b, c)$ or $(b, c, d)$.

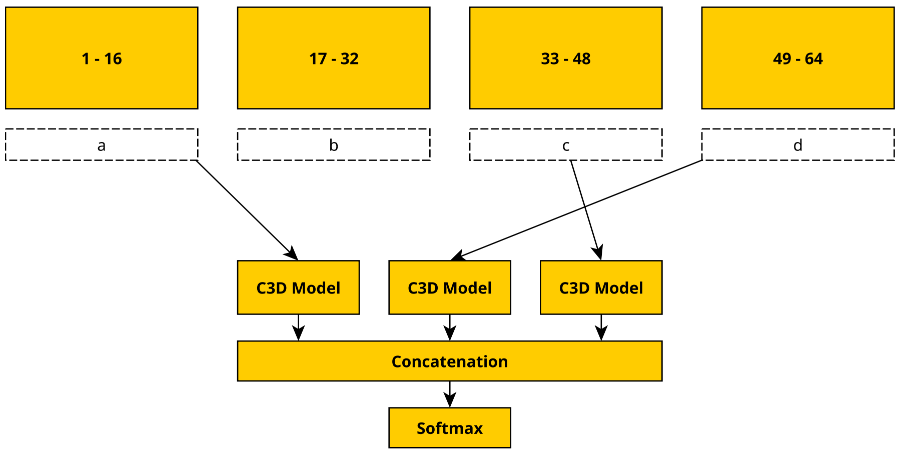


*Figure 28:* Pre-training with a triplet Siamese *C3D* model equivalent to [19]

# 5. Evaluation

The following evaluated training processes have been carried out on two GPUs as described in section 4.1. In order to make optimal use of the available GPUs, the training and evaluation process have been performed independently. More specifically, all trained models are saved after each finished epoch in order to load it on a separate machine and evaluate it. That way training can seamlessly continue while the last saved model is loaded and evaluated in parallel on the regarding training set.

## 5.1. Input Pipeline with multiple GPUs

To evaluate the training speed up by using multiple GPUs with an asynchronous input pipeline, we measure the number of training input clips that can be processed per second in different setups. Specifically, processing an input during training includes calculating the forward pass through the network, calculating the weight updates and applying them to the model. In the case of multiple GPUs, the calculated weight updates of the individual GPUs need to be averaged to obtain a global model update after each training step as described in section 4.4. For the incorporated *C3D* model an input clip denotes a $16 \times 112 \times 112 \times 3$ video volume, i.e. 16 frames of resolution $112 \times 112$ pixel in 3 channels (RGB).

During training without input pipeline (denoted as *no IP* in table 7), input processing and training are executed sequentially: an input is first pre-processed and then fed into the model for training in the same thread. This yields the disadvantage, that the GPUs need to wait for the input clips to be ready for further processing. This time however depends on the dataset, since e.g. video decoding time depends on the video quality. Training with input pipeline (denoted as *IP* in table 7) incorporates a FIFO Queue which is fed from four threads asynchronously as described in 4.3. That way an input clip is ready whenever needed, if the pre-processing threads are faster than the GPUs.

All measurements are taken from training on UCF101, which has shown to be the fastest dataset to be decoded in comparison to Kinetics and Charades. Speed measurements on the CPU were only obtained for a batchsize of 10 without input pipeline, since the result already indicates the infeasibility of training on the CPU and bigger batchsizes were not processable due to memory limitations.

**Training Speed in clips/second**

| | **batch = 10 clips** | | **batch = 20 clips** | | **batch = 40 clips** | |
|---|---|---|---|---|---|---|
| | no IP | IP | no IP | IP | no IP | IP |
| CPU | 0.08 ± 0.01 | | | | | |
| GPU | 14.04 ± 0.93 | 13.59 ± 1.54 | 14.94 ± 0.66 | 14.74 ± 0.92 | | |
| 2 × GPU | 8.39 ± 1.29 | 16.70 ± 0.71 | 13.66 ± 0.61 | 22.36 ± 0.58 | 15.07 ± 0.73 | 26.32 ± 0.67 |

*Table 7:* Training speed for different batchsizes with and without multi-threaded queue-based input pipeline (IP and no IP) measured for 100 training steps each.

The results in table 7 indicate:

1. Training on the CPU only is infeasible.
2. For smaller batch-sizes, i.e. 10 or 20 clips per batch, the overhead from averaging the gradients of multiple GPUs and communication the updates hurts training speed to a point where a single GPU is more effective.
3. A sophisticated input pipeline is not necessary for batchsizes of 10 and 20 clips. We note that this is in part due to the low resolution of UCF101 videos which results in fast decoding and therefore short pre-processing times.
4. Batchsizes of 40 clips are only processable with two GPUs due to the doubled amount of GPU memory.
5. Two GPUs significantly benefit from an asynchronous input pipeline and perform best in this setup.
6. By training on two GPUs 1.76 times more inputs can be processed per second compared to using a single GPU, which significantly decreases training time.

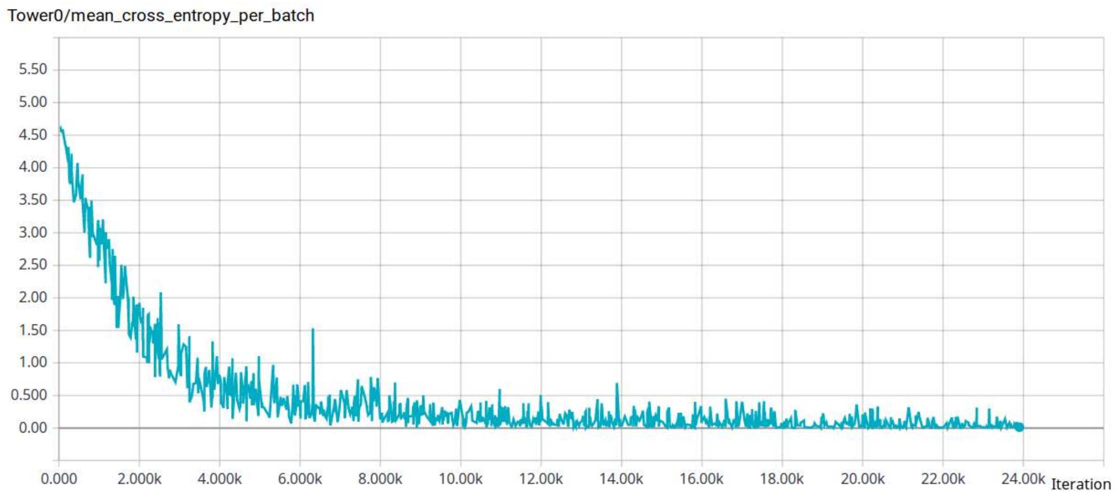
## 5.2. UCF101 Classification without Pre-Training

To provide a baseline for comparing the effects of pre-training with *temporal order verification*, a *C3D* model is trained on UCF101 (split 1) from randomly initialized weights. More precisely, weights are initialized from a normal distribution with mean 0 and standard deviation of 0.01.[7] The model is trained for 23.9k iterations, which corresponds

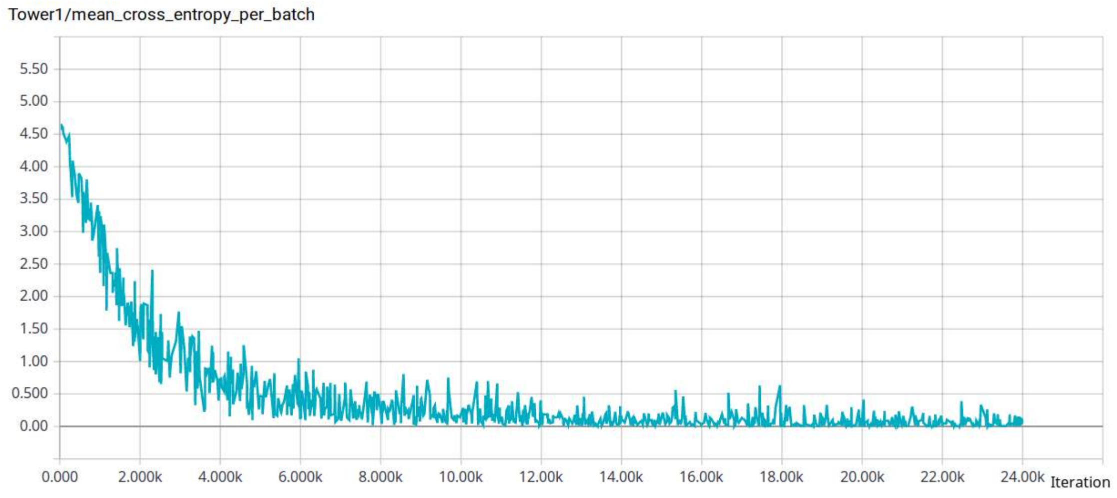
[7]Whenever weights are initialized randomly in the following evaluations, exactly this distribution is meant and used.

to 100 finished epochs at a batch-size of 40 clips per iteration and 9,537 videos in the training set of UCF101. The overall training process takes 12 hours 20 minutes and 12 seconds. A learning rate of $10^{-5}$ was found to be the biggest possible learning rate without resulting in diverging loss.

Figure 29 shows the **loss** per batch for each training-step of both GPUs. The training process clearly converges, indicated by a degrading loss towards 0 and a slope of approximately 0 during the last iterations.



*(a)* Training loss of GPU0 (Tower0)



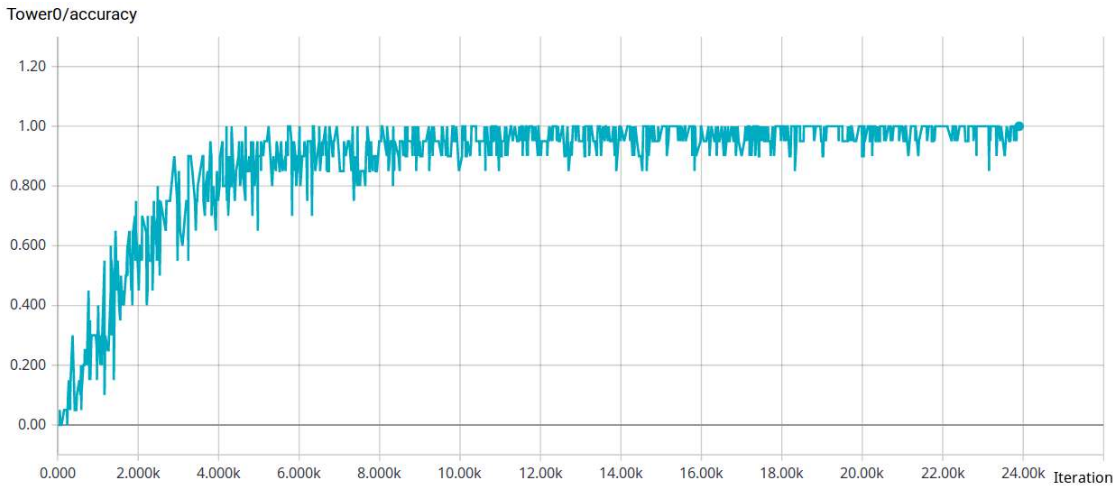
*(b)* Training loss of GPU1 (Tower1)

*Figure 29:* Training loss of the *C3D* model per batch while training from randomly initialized weights on split 1 of UCF101.

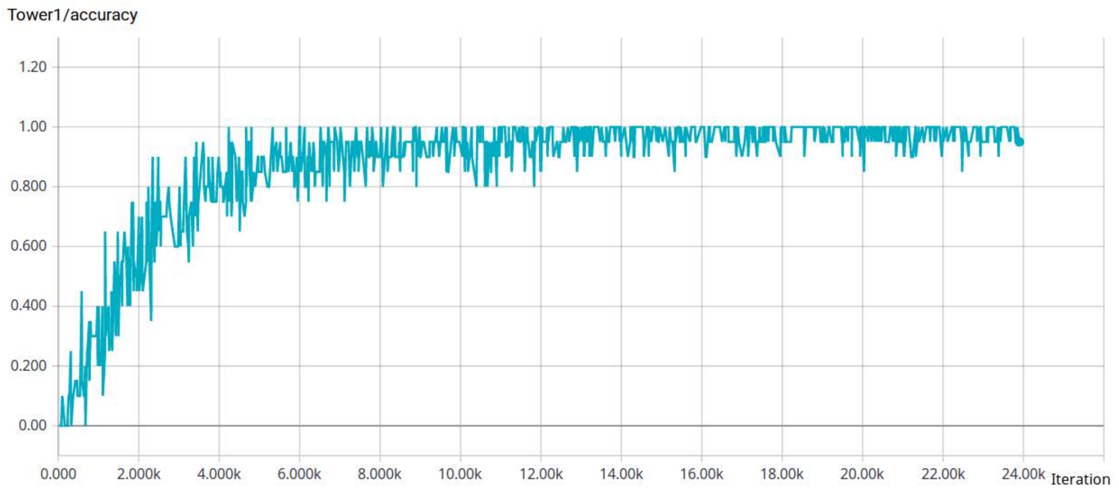
Figure 30 shows the **accuracy** per batch for each training-step of both GPUs while training on UCF101. The training accuracy rises to near optimum during convergence of the training process. This indicates overfitting, as we are training a deep model on a comparably small dataset (see section 3).

To evaluate the performance of our model on the testset of UCF101 we incorporate the evaluation strategy proposed in [46]. Test accuracy is measured by cropping consecutive center clips from the test video, feeding them to the model and averaging the individual outputs. This way the complete temporal evolution of the video can be taken into account for calculating the test accuracy, since a variable number of clips can be averaged to cover different video lengths. The cropped center clips can possibly overlap, i.e. an offset between the beginnings of consecutive clips can be varied. An offset of $\omega = 16$ corresponds to no overlap, an offset of $\omega < 16$ results in $16 - \omega$ overlapping frames and an offset of $\omega > 16$ results in gaps between consecutive clips. For our evaluation we incorporate no overlap, i.e. $\omega = 16$, as a compromise between evaluation speed and video coverage.

**Result:**
Epoch 65 results in the best performing model on the test-set of UCF101 with an accuracy of 44.57%. The early maximum of the test accuracy and difference between training and test accuracy confirms the conjecture, that the model is overfitting to the data. Nevertheless, our result is comparable to the reported performance of [46] (51.6% accuracy) and [40] (51.9% accuracy). Our resulting accuracy is slightly lower, since no hyperparameter-tuning has been performed as we are for now only interested in a functioning baseline to evaluate the difference against pre-training the same model with the same hyperparameters using *temporal order verification*[19].



*(a)* Training accuracy of GPU0 (Tower0)



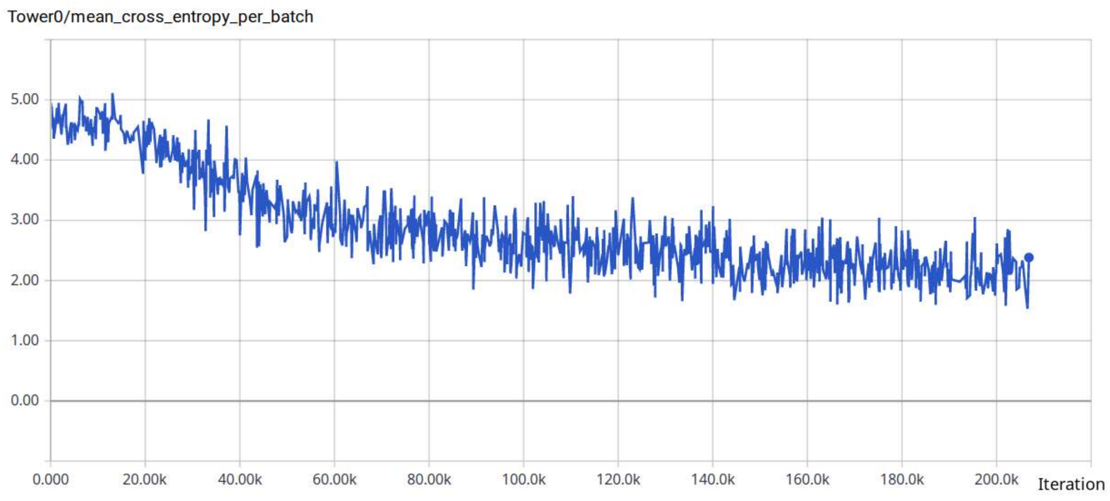
*(b)* Training accuracy of GPU1 (Tower1)

*Figure 30:* Training accuracy of the *C3D* model per batch while training from randomly initialized weights on split 1 of UCF101.

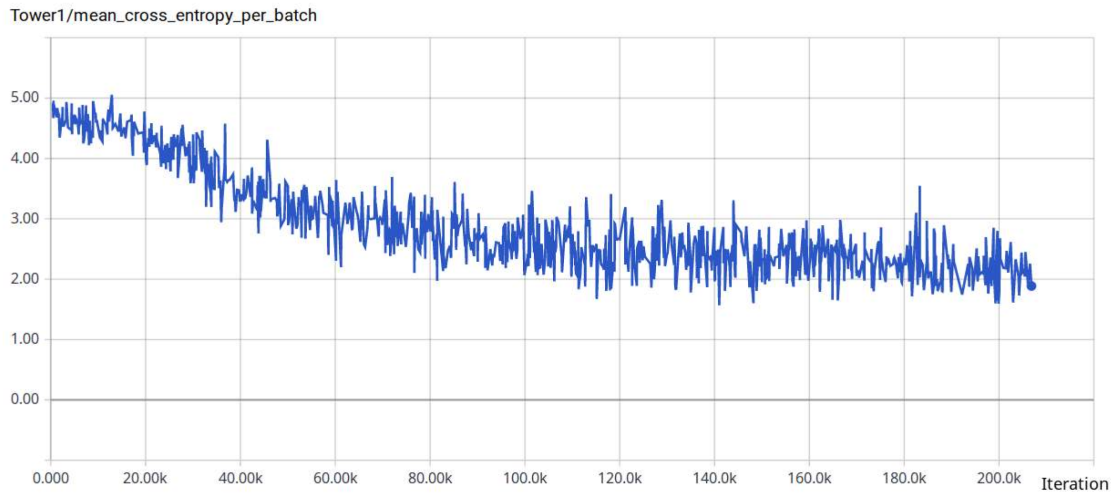
## 5.3. Charades Classification without Pre-Training

Similar to obtaining a baseline for the UCF101 dataset, a *C3D* model is trained on the Charades dataset from randomly initialized weights. The weights are again sampled from a normal distribution with mean 0 and standard deviation of 0.01. The model is trained for roughly 206.8k iterations, which corresponds to 166 epochs at a batch-size of 40 clips per iterations and $49,809$ temporally localized actions in the training-set of Charades. The training process takes 7 days 11 hours 11 minutes and 14 seconds. A learning rate of $10^{-5}$ was found to be the biggest possible learning rate without resulting in diverging loss.

Figure 31 shows the loss per batch for each training-step of both GPUs. The training process did not fully converge as there is still a noticeable slope present in the last 40k iterations. As computation time is limited and this model is trained as a baseline, the training was stopped at this point after roughly eight days.



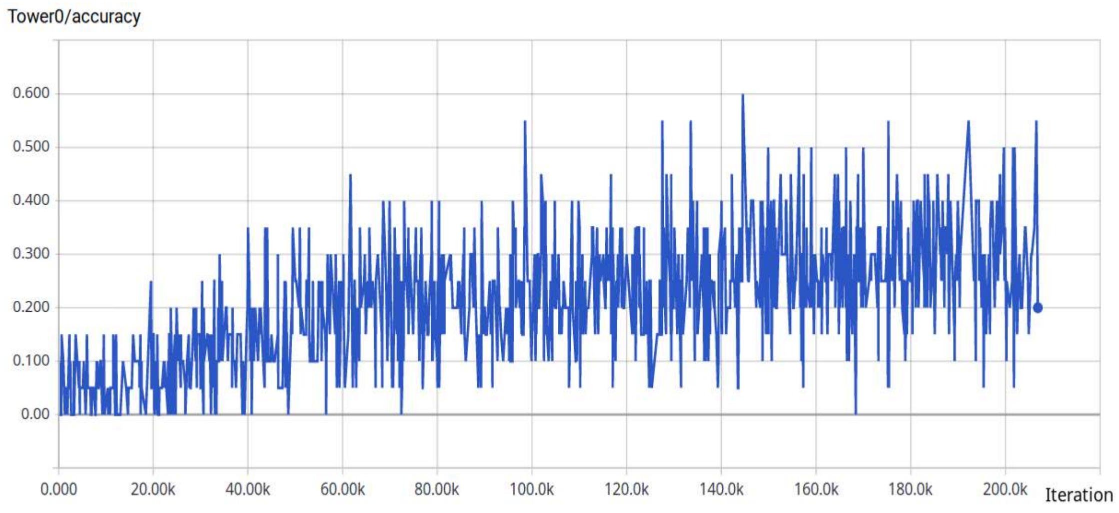
*(a)* Training loss of GPU0 (Tower0)



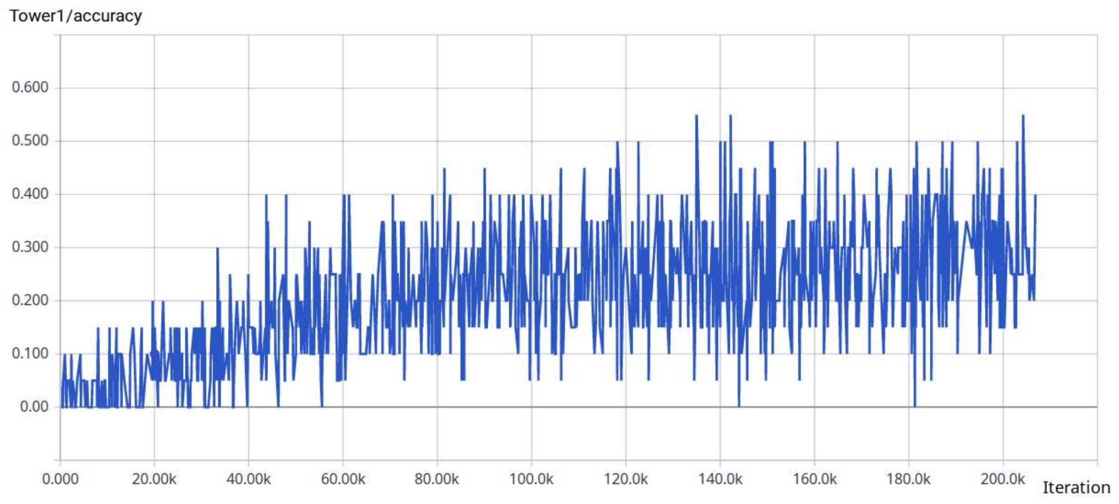
*(b)* Training loss of GPU1 (Tower1)

*Figure 31:* Training loss of the *C3D* model per batch while training from randomly initialized weights on the Charades dataset.

Figure 32 shows the accuracy per batch for each training-step of both GPUs on the Charades dataset. The training accuracy starts to rise significantly after around 35k iterations, which corresponds to $24h$ of training. The evolution of the accuracy curve indicates, that this is a more difficult learning task than UCF101 classification. This is partly due to the fact, that labels are ambiguous on the Charades dataset since they can overlap for a single video as described in section 3.3.3.



*(a)* Training accuracy of GPU0 (Tower0)



*(b)* Training accuracy of GPU1 (Tower1)

*Figure 32:* Training accuracy of the *C3D* model per batch while training from randomly initialized weights on the Charades dataset.

Since labels are ambiguous, test accuracy is not an appropriate performance measure any more. Therefore *mean average precision*[59] was established as standard performance measure on the charades dataset. To evaluate the performance of our model on the test-set of Charades we again incorporate the evaluation strategy proposed in [46]. Network outputs per test video are obtained by cropping consecutive center clips from the test video, feeding them to the model and averaging the individual outputs. The mean average precision (mAP) for classifying Charades videos is calculated from the network outputs for all test videos using the provided evaluation script (see section 3.3.3).

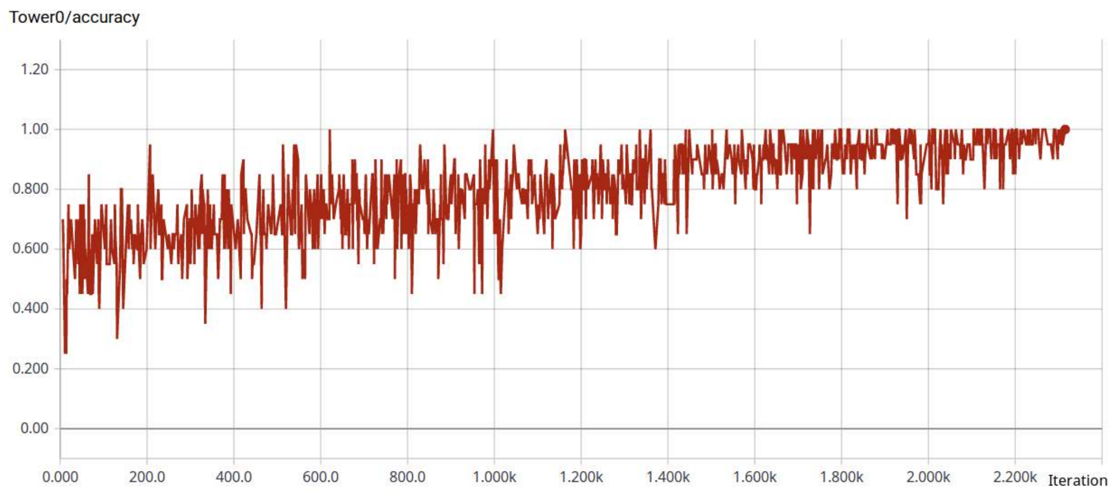
**Result:** The *C3D* network trained from randomly initialized weights yields a mAP of 9.01% after epoch 146. This result is comparable to the reported mAP (10.9%) in the original Charades publication by Sigurdsson et al. [12], as described in section 3.3.1. Sigurdsson et al. [12] use a *C3D* model pre-trained on Sports-1M[13] while ours is trained from random weights, which explains the difference in performance.

## 5.4. Pre-Training on Kinetics

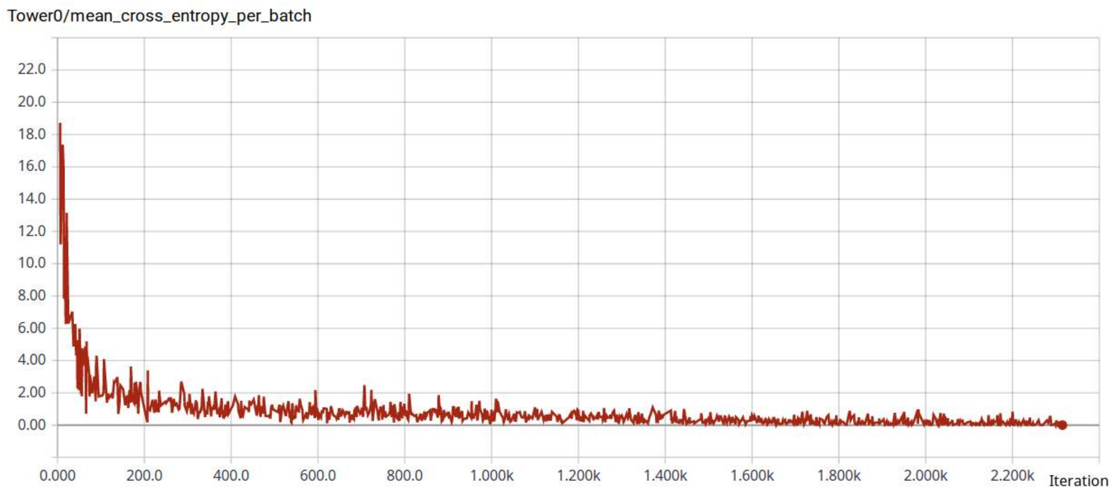
This section evaluates *temporal order verification* with the previously introduced methods of sampling positive (*correct temporal order*) and negative (*incorrect temporal order*) input clips from the Kinetics dataset. A *C3D* model is pre-trained on these three methods, which are further called *method 1, 2 and 3*.

**Method 1**
Pre-training of a *C3D* model is performed with sampled input clips, that are split in three equal parts which are then randomly permuted to generate a negative input sample or kept in place to generate a positive input sample.

Figure 33 illustrates the pre-training process by showing the training accuracy and training loss for each input batch per training step on GPU0. In the following evaluation results for GPU1 are omitted to not unnecessarily lengthen this report.



*(a)* Training accuracy of GPU0 (Tower0)



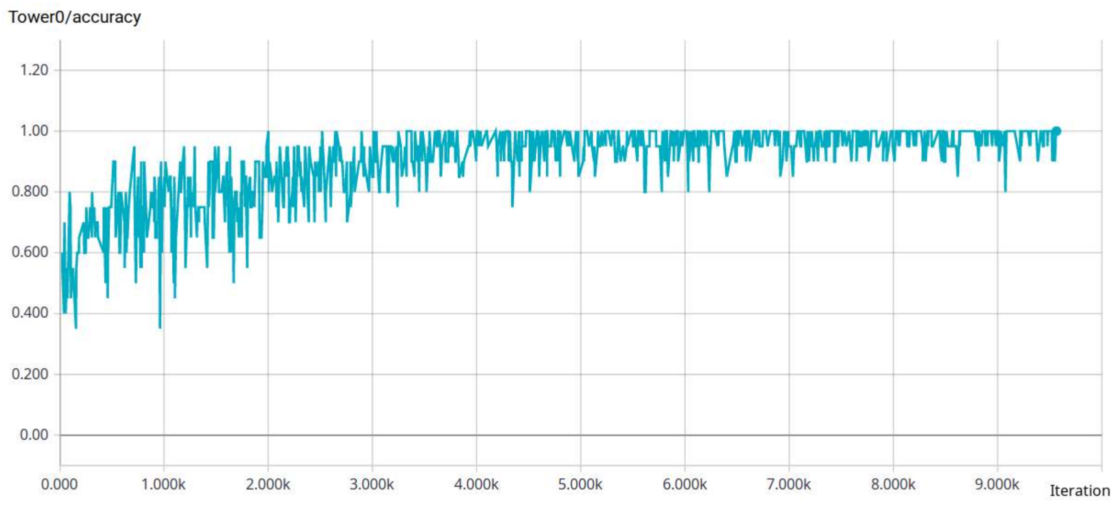
*(b)* Training loss of GPU0 (Tower0)

*Figure 33:* Pre-training process by *temporal order verification* using input sampling method 1.

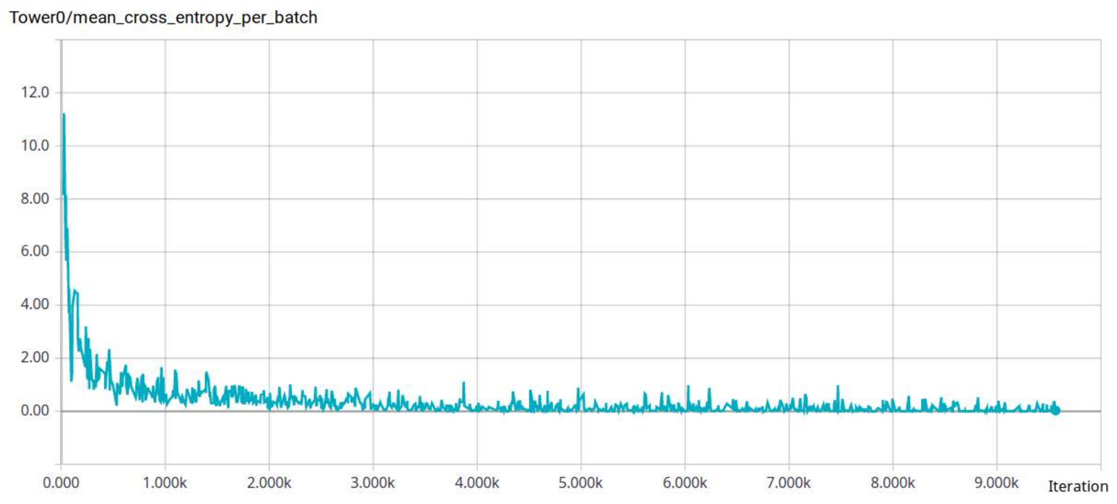
As figure 33 illustrates, the training process converges after only 2.2k iterations, which is extremely fast considering previous training times. This goes against the observation in the original *temporal order verification* publication[19], which reports pre-training for 100k iterations. Since here a more complex 3D CNN is trained, we assume even longer pre-training times. This indicates that the model learned to just focus on the regions where cuts can occur, as diving dividing the input clip in only three equal parts generate cuts at distinct locations. Due to limited computation time and this obvious flaw in *method 1*, we therefore abandon this method and focus on *method 2* and *3*.

**Method 2**

This input sampling improves over the previous sampling *method 1* in that a cut from permuting the input clip can potentially occur between any two frames. Figure 34 illustrates the pre-training process by showing the training accuracy and loss for each batch per training step on GPU0. Results on GPU1 are again similar enough to be omitted in this report.



*(a)* Training accuracy of GPU0 (Tower0)

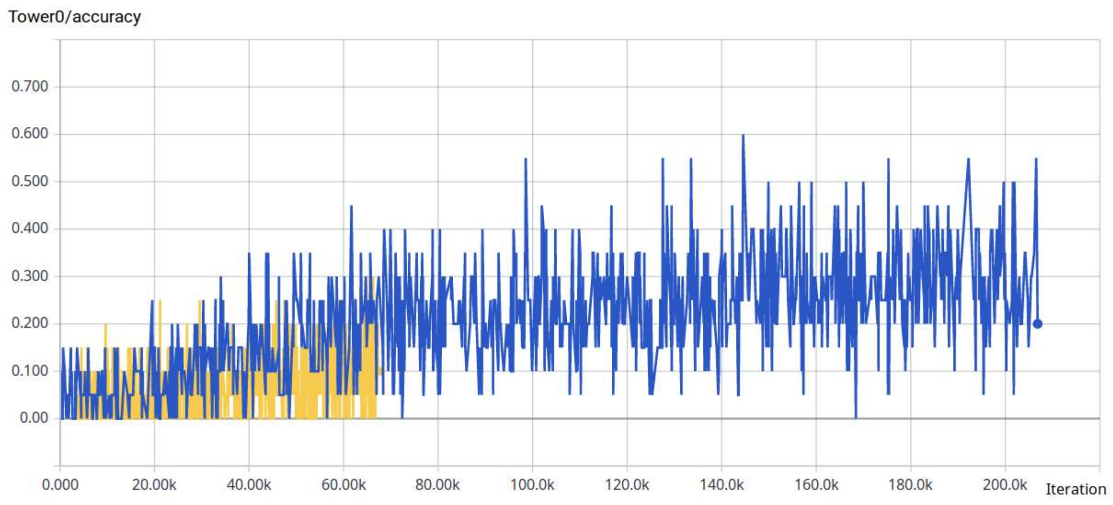


*(b)* Training loss of GPU0 (Tower0)

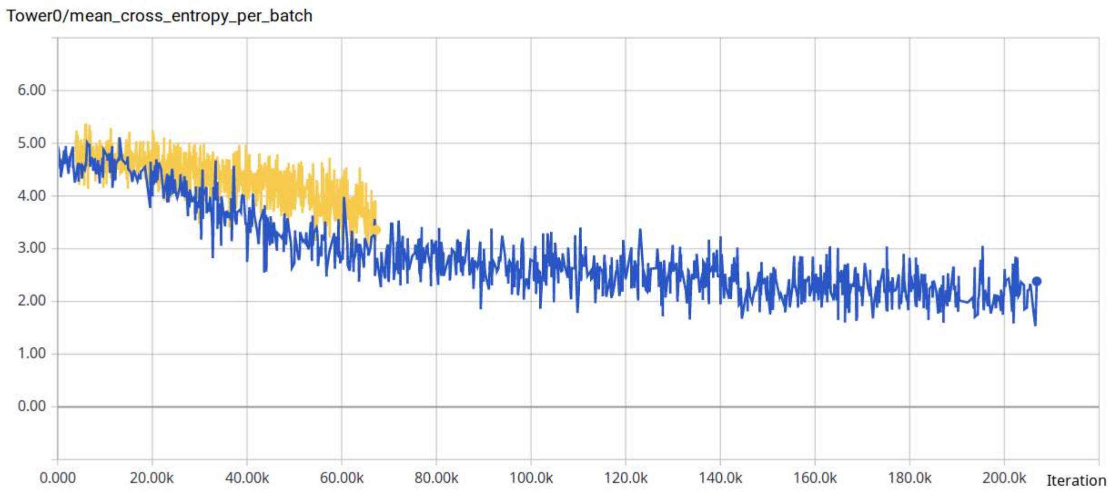
*Figure 34:* Pre-training process by *temporal order verification* using input sampling method 2.

The pre-training process again converges fast in comparison to the pre-training times reported by Misra, Zitnick, and Hebert [19]. In comparison to input sampling *method 1*, training takes roughly three times longer. This indicates, that sampling method 2 poses a more challenging learning task, since potential cuts need to be detected between every frame. We therefore evaluate the quality of pre-training with *method 2* by initializing a *C3D* model from the pre-trained weights and train it on the Charades dataset. Figure

35 shows training accuracy and loss per batch and training step on the Charades dataset for randomly initialized weights (blue) and training from pre-trained weights (yellow).



*(a)* Training accuracy of GPU0 (Tower0)



*(b)* Training loss of GPU0 (Tower0)

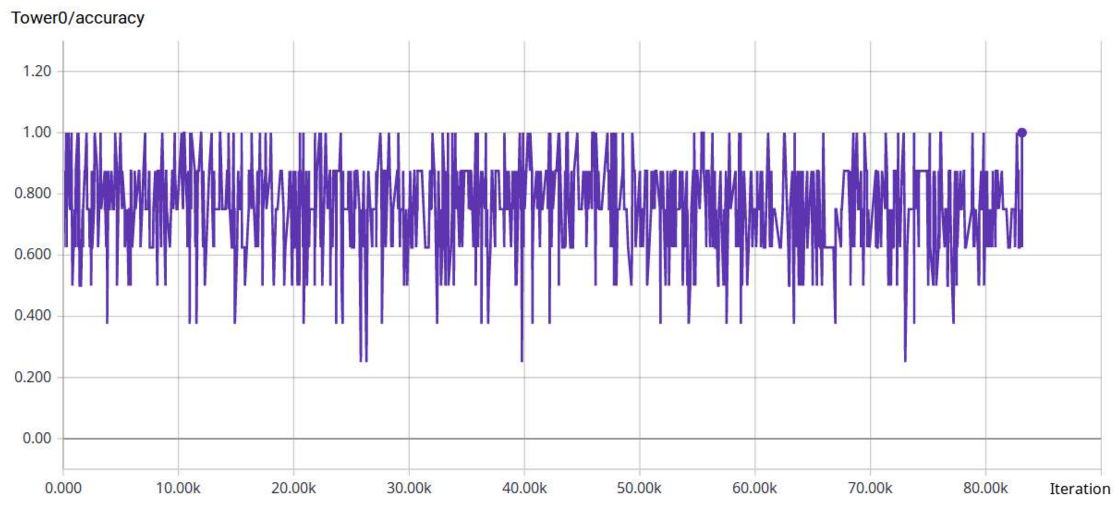
*Figure 35:* Training a *C3D* model on the Charades dataset from randomly initialized weights (*blue*) and from pre-trained weights obtained through *temporal order verification* (*yellow*).

The progression of training accuracy and loss in figure 35 shows, that both models initially perform equally well. However, the pre-trained model performs significantly worse after iteration 15k. Specifically, the training loss of the pre-trained model is consistently higher than from the model without pre-training and the training accuracy is accordingly inferior. This goes against the observation of pre-training in [19], as the pre-trained model should require considerably less training steps to perform equally well or better during training and outperform the randomly initialized model during testing. After a training time of roughly 2 days, when the performance deficit was significantly noticeable, the training was stopped. We therefore concluded pre-training with input sampling *method 2* to be non beneficial for action recognition and focus on *method 3*.

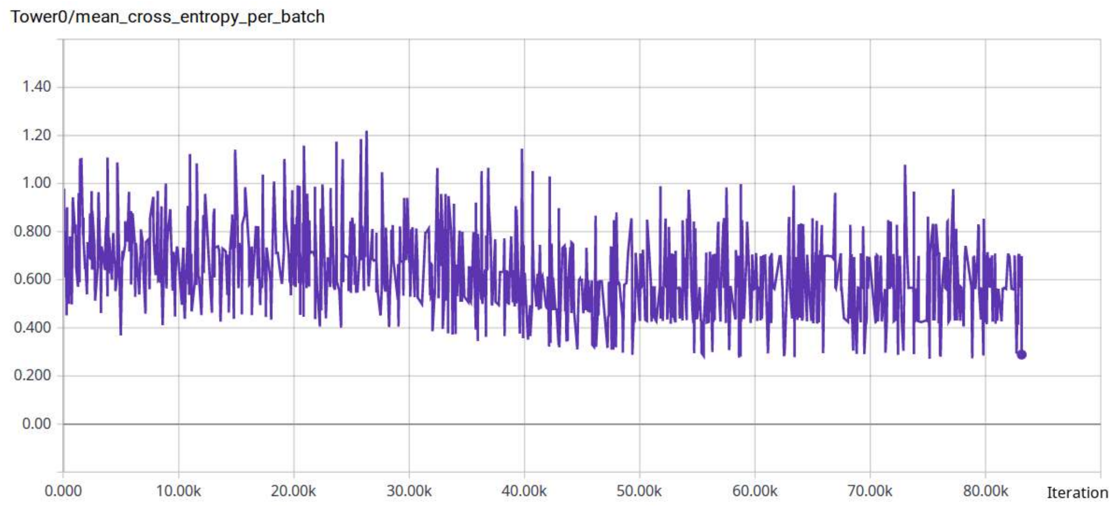
**Method 3**

This sampling method is, compared to the previous two, most related to the original approach of Misra, Zitnick, and Hebert [19]. The *C3D* model is replicated three times in a triplet network setup, which also increases the size of network inputs by a factor of three. This necessitates a decrease of the batchsize, in our case to 16 input examples per training step, to not exceed the GPU memory.

The triplet *C3D* model is pre-trained for 83.13k iterations, which corresponds to 2 days 20 hours and 56 minutes. A learning rate of $10^{-5}$ was found to be the biggest possible learning rate without resulting in diverging loss. The training accuracy quickly adjusts to a mean training accuracy of around 75%, yet does not change noticeably over the pre-training process.



*(a)* Training accuracy of GPU0 (Tower0)



*(b)* Training loss of GPU0 (Tower0)

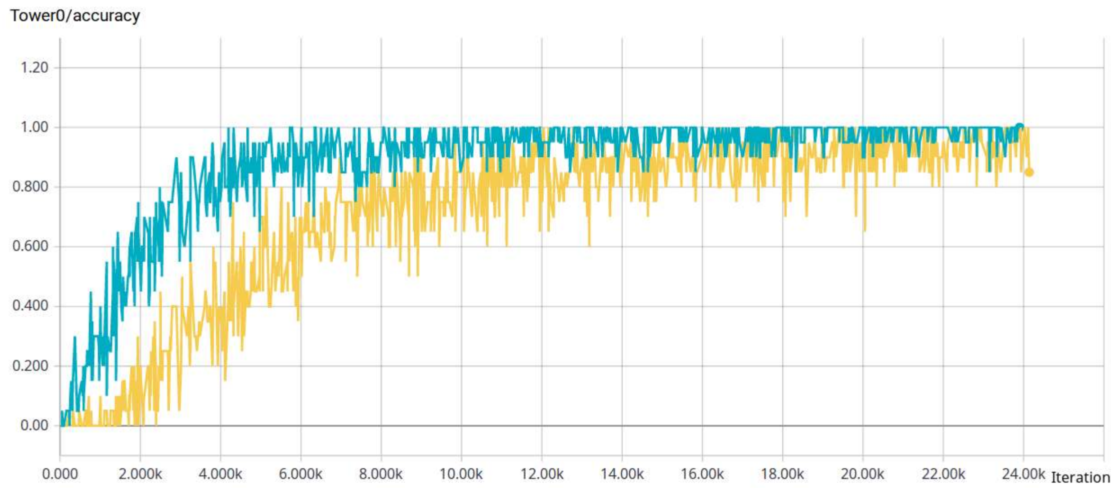
*Figure 36:* Pre-training process by *temporal order verification* using input sampling method 3.

The fact, that the training accuracy does not noticeably improve during the training, might raise the concern that no weight updates are taking place and the training process is inherently flawed. However, the apparent decrease in training loss indicates weight updates, which additionally have been confirmed manually by checking the parameters of multiple saved models. Misra, Zitnick, and Hebert [19] report an accuracy of 72.1% for *temporal order verification*, which is comparable to our observation. We therefore evaluate the last saved model from this pre-training run for supervised fine-tuning on UCF101 and Charades.

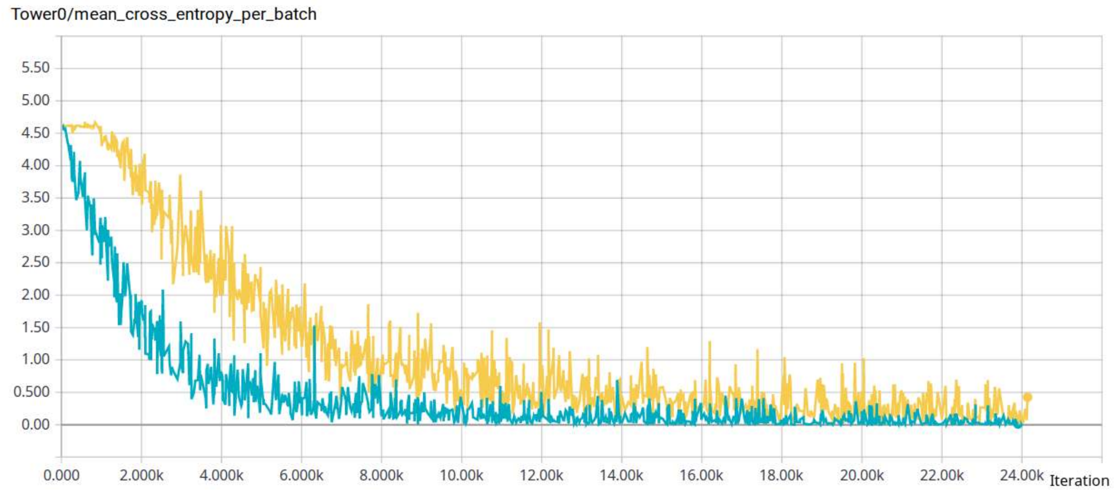
## 5.5. UCF101 Classification with Pre-Training

To evaluate the pre-trained weights obtained with *method 3*, a *C3D* model is initialized on these weights and trained for 24.14k iterations, which corresponds to 100 epochs on UCF101 at a batch-size of 40 input clips per iteration (20 inputs per GPU). The overall training process takes 17 hours and 6 seconds.

Figure 37 illustrates the training process for the pre-trained model (yellow) in comparison to the previously trained model from randomly initialized weights (cyan).



*(a)* Training accuracy of GPU0 (Tower0)



*(b)* Training loss of GPU0 (Tower0)

*Figure 37:* Training process of a *C3D* model from randomly initialized weights (cyan) in comparison to training a *C3D* model from pre-trained weights obtained by pre-training method 3.

The progression of the training accuracy and loss curve indicates a slower training process compared to training the model without pre-training. The curves obtained from pre-training converge close to the optima, yet not as strictly as before. Since the model without pre-training overfit to the training data, this might indicate a regularization effect of pre-training the model.

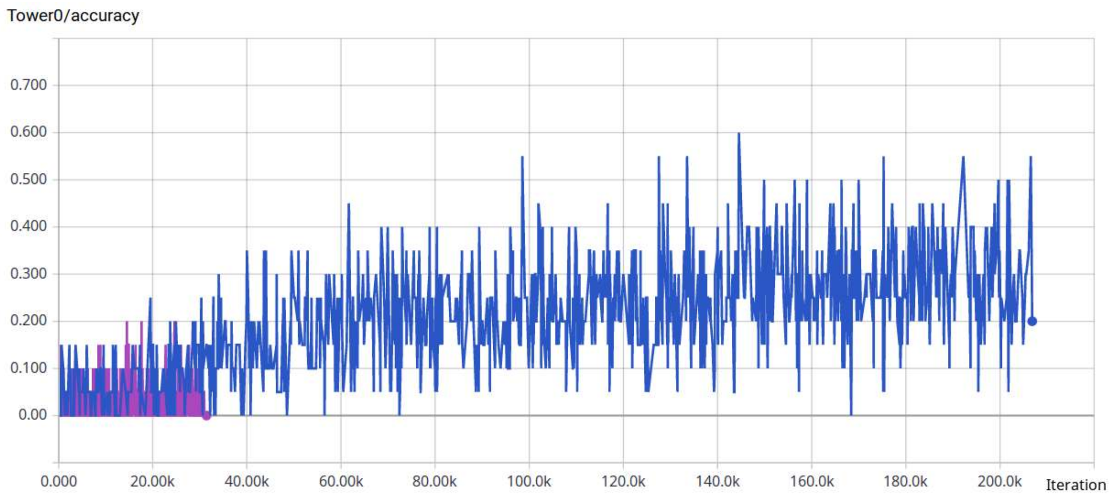
**Result:**
We evaluate the pre-trained model on the test set of UCF101. Epoch 80 results in the best performing model, which yields an accuracy of 30.05%. This result is significantly worse compared to training the model from randomly initialized weights, which yields an accuracy of 44.57%. The previously conjectured regularization effect is therefore not present and pre-training the model actually impairs the performance of the model. The loss curve of the pre-trained model in figure 37 (b) shows a plateau at the beginning of training. This is most likely due to the disadvantageous initialization and the optimizer first needs to adjusts the initial weights without any change in training loss.

## 5.6. Charades Classification with Pre-Training

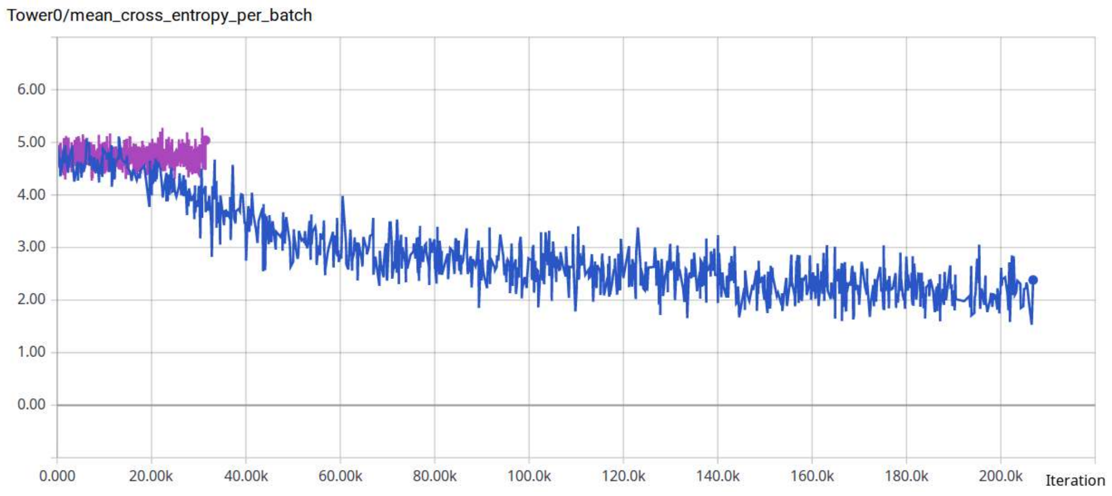
The effect of the pre-trained weights obtained with *method 3* is additionally evaluated on the Charades dataset. A *C3D* model is initialized on the previously obtained weights and trained for 31.15k iterations, which corresponds to 24 finished iterations. Figure 38 illustrates the training process compared to training on the charades dataset from randomly initialized weights. The initial plateau in the training loss, previously observed on UCF101, is clearly present again. After 1 day 2 hours and 56 minutes the training process was stopped due to time constraints. However the time-span is long enough to also confirm the negative effect of pre-trained weights on the Charades dataset.

## 5.7. Discussion

As these results disprove our initial hypothesis, that one of the above evaluated input sampling strategies can be used to beneficially transfer *temporal order verification* to 3D CNNs, we investigate probable causes. Upon manually investigating the Kinetics source videos, which were used to sample positive and negative examples for *Temporal Order Verification*, we identified several of these to contain cutscenes. Since the magnitude of contained motion is evaluated by splitting an input clip into three parts and comparing the similarity of between each chunk's mean frame, a pre-existing cutscene would identify the input as a high motion video clip. This would then compromise the labelling during *temporal order verification*, because an non permuted input clip with a cutscene is labelled as *correct temporal order* although being de-facto temporally discontinuous.



*(a)* Training accuracy of GPU0 (Tower0)



*(b)* Training loss of GPU0 (Tower0)

*Figure 38:* Training of a *C3D* model on the Charades dataset from pre-trained weights (purple) compared to training from randomly initialized weights (blue).

# 6. Conclusion and Future Directions

In this work, we studied the quasi unsupervised pre-training method called *temporal order verification* for improving the recognition of daily living actions from video data with the *C3D* deep convolutional neural network. *Temporal order verification* has shown promising results in learning movement-focused representations from still video frames with 2D CNNs. We therefore evaluated it as a means to incorporate motion sensitivity in 3D convolutional neural networks, which otherwise treat the temporal evolution of a video equally to the spatial dimension of video frames. Two-stream CNNs address this issue by explicitly incorporating optical flow, which is computationally expensive to obtain. Since pre-training a network with *temporal order verification* does not require labelled data, it is especially suited to improve recognition tasks where large amounts of labelled data are sparse. We specifically focused on the Charades dataset, which features a unique amount of mundane daily-living action videos and therefore enables vision-based systems, which can be deployed in real-world applications such as assistive robotics.

In this work we devised an asynchronous input pipeline for multiple GPUs to make deep neural network training for action recognition from video data feasible on commercial hardware. We trained the *C3D* model for action recognition using this pipeline on multiple GPUs in a data-parallel way, which yields competitive performance in recognizing daily-living actions. We additionally studied three different input sampling methods to transfer the procedure of *temporal order verification* from 2D CNNs into 3D CNN pre-training: the first two methods focus on permuting single network inputs for a single *C3D* model, while the third method adapts the replication of network models and processes three network inputs, which are individually kept unchanged but whose temporal order amongst each other is permuted.

By incorporating multiple GPUs in the training process and feeding input data through our input pipeline we were able to process 1.76 times more input clips per second compared to training on a single GPU. Our *C3D* implementation yields an accuracy of 44.57% on the UCF101 action recognition standard benchmark and a mean average precision of 9.01% for recognizing daily-living actions of the Charades dataset. The longest training duration in the course of this work took 7 days and 11 hours on two GPUs, which would have been prolonged to 13 days and 3 hours by training on a single GPU only. We conclude that efforts to reduce training time are vital for action recognition research, since models for processing video data are more complex, inputs have a higher dimensionality and training therefore naturally takes longer than in the image processing domain. Reducing training time through parallelization results in being able to perform more training runs and evaluations in a given amount of time.

By pre-training the *C3D* network with *temporal order verification*, we were not able to improve the action recognition performance with any of the three evaluated input sampling methods: In fact the pre-trained weights turned out to significantly impair the network's performance. This result underlines the impact of weight initializations on the ability of a network to effectively learn a classification task and fuels the future search for proper initialization methods to improve a network's performance. We hypothesize that the observed decrease in performance may be due to pre-existing cuts in the source videos, which were not introduced from permuting inputs and therefore compromise the generation of positive samples with *correct temporal order.* This could be addressed by first detecting pre-existing cuts in a video and sampling *temporal order verification* input clips only from continuous regions of the video.

In the course of this work, we identified additional aspects to improve action recognition performance in the future. For instance a human detector could be incorporated to provide the learning system with more relevant training clips during training instead of sampling input clips randomly from a training video; as this does not heavily influence training on pre-processed datasets, which contain clips of a single human performing a single action, it would provide better training inputs for real-world training datasets. We initially chose the *C3D* model for this work, because its deficits are potentially addressed by *temporal order verification* and given that it is a prominent example of a single stream 3D CNN, the results can be compared to other publications in action recognition that picked up the model. However, several recent advances such as strided convolutions to substitute the pooling layers for dimensionality reduction or skip-connections could be incorporated to improve the model itself. Most importantly, a longer temporal extent, i.e. temporally longer inputs, should be processed by the model, as this was found to heavily influence action recognition performance. To stay computationally feasible, the final fully-connected layer would then need to be reduced in size.

## A. List of the 157 action classes in Charades

Holding some clothes
Putting clothes somewhere
Taking some clothes from somewhere
Throwing clothes somewhere
Tidying some clothes
Washing some clothes
Closing a door
Fixing a door
Opening a door
Putting something on a table
Sitting on a table
Sitting at a table
Tidying up a table
Washing a table
Working at a table
Holding a phone/camera
Playing with a phone/camera
Putting a phone/camera somewhere
Taking a phone/camera from somewhere
Talking on a phone/camera
Holding a bag
Opening a bag
Putting a bag somewhere
Taking a bag from somewhere
Throwing a bag somewhere
Closing a book
Holding a book
Opening a book
Putting a book somewhere
Smiling at a book
Taking a book from somewhere
Throwing a book somewhere
Watching/Reading/Looking at a book
Holding a towel/s
Putting a towel/s somewhere
Taking a towel/s from somewhere
Throwing a towel/s somewhere
Tidying up a towel/s
Washing something with a towel
Closing a box
Holding a box
Opening a box
Putting a box somewhere
Taking a box from somewhere
Taking something from a box
Throwing a box somewhere
Closing a laptop
Holding a laptop
Opening a laptop
Putting a laptop somewhere
Taking a laptop from somewhere
Watching a laptop or something on a laptop
Working/Playing on a laptop
Holding a shoe/shoes
Putting shoes somewhere
Putting on shoe/shoes
Taking shoes from somewhere
Taking off some shoes
Throwing shoes somewhere
Sitting in a chair
Standing on a chair
Holding some food
Putting some food somewhere
Taking food from somewhere
Throwing food somewhere
Eating a sandwich
Making a sandwich
Holding a sandwich
Putting a sandwich somewhere
Taking a sandwich from somewhere
Holding a blanket
Putting a blanket somewhere
Snuggling with a blanket
Taking a blanket from somewhere
Throwing a blanket somewhere
Tidying up a blanket/s
Holding a pillow
Putting a pillow somewhere
Snuggling with a pillow
Taking a pillow from somewhere
Throwing a pillow somewhere
Putting something on a shelf
Tidying a shelf or something on a shelf
Reaching for and grabbing a picture
Holding a picture
Laughing at a picture
Putting a picture somewhere
Taking a picture of something
Watching/looking at a picture
Closing a window
Opening a window
Washing a window
Watching/Looking outside of a window
Holding a mirror

Smiling in a mirror
Washing a mirror
Watching something/someone/themselves in a mirror
Walking through a doorway
Holding a broom
Putting a broom somewhere
Taking a broom from somewhere
Throwing a broom somewhere
Tidying up with a broom
Fixing a light
Turning on a light
Turning off a light
Drinking from a cup/glass/bottle
Holding a cup/glass/bottle of something
Pouring something into a cup/glass/bottle
Putting a cup/glass/bottle somewhere
Taking a cup/glass/bottle from somewhere
Washing a cup/glass/bottle
Closing a closet/cabinet
Opening a closet/cabinet
Tidying up a closet/cabinet
Someone is holding a paper/notebook
Putting their paper/notebook somewhere
Taking paper/notebook from somewhere
Holding a dish
Putting a dish/es somewhere
Taking a dish/es from somewhere
Wash a dish/dishes
Lying on a sofa/couch
Sitting on sofa/couch
Lying on the floor
Sitting on the floor
Throwing something on the floor
Tidying something on the floor
Holding some medicine
Taking/consuming some medicine
Putting groceries somewhere
Laughing at television
Watching television
Someone is awakening in bed
Lying on a bed
Sitting in a bed
Fixing a vacuum
Holding a vacuum
Taking a vacuum from somewhere
Washing their hands
Fixing a doorknob
Grasping onto a doorknob
Closing a refrigerator
Opening a refrigerator
Fixing their hair
Working on paper/notebook
Someone is awakening somewhere
Someone is cooking something
Someone is dressing
Someone is laughing
Someone is running somewhere
Someone is going from standing to sitting
Someone is smiling
Someone is sneezing
Someone is standing up from somewhere
Someone is undressing
Someone is eating something

## References

[1] Maja J. Matarić and Brian Scassellati. “Socially Assistive Robotics”. In: *Springer Handbook of Robotics*. Springer, 2016. URL: http://link.springer.com/chapter/10.1007/978-3-319-32552-1_73.

[2] David Feil-Seifer and Maja J. Mataric. “Defining Socially Assistive Robotics”. In: *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on*. IEEE, 2005. URL: http://ieeexplore.ieee.org/abstract/document/1501143/.

[3] K. Wada et al. “Analysis of Factors That Bring Mental Effects to Elderly People in Robot Assisted Activity”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2002.

[4] H.-M. Gross et al. “Progress in Developing a Socially Assistive Mobile Home Robot Companion for the Elderly with Mild Cognitive Impairment”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011. URL: http://ieeexplore.ieee.org/abstract/document/6094770/.

[5] A. W. Davis and I. M. Weinstein. “Telepresence 2007–Taking Videoconferencing to the Next Frontier”. In: *Crevision 2.0, Wainhouse Research*. 2007. URL: http://www.wainhouse.com/images/reports/wr_telepres07.pdf.

[6] Yann LeCun et al. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems*. 1990. URL: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[7] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.

[8] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems*. 2015. URL: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.

[9] Liang-Chieh Chen et al. “Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected Crfs”. In: *arXiv preprint arXiv:1606.00915* (2016). URL: https://arxiv.org/abs/1606.00915.

[10] Maximilian Schöbel, Erwin Prassler, and Paul Plöger. “Evaluation of Current Approaches for Situation-Awareness in Autonomous Systems from Action Recognition in Video Data”. In: *Unpublished* (2017).

[11] Tal Hassner. “A Critical Review of Action Recognition Benchmarks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, June 2013. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6595882.

[12] Gunnar A. Sigurdsson et al. “Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding”. In: *arXiv preprint arXiv:1604.01753* (2016). URL: http://arxiv.org/abs/1604.01753.

[13] Andrej Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2014/html/Karpathy_Large-scale_Video_Classification_2014_CVPR_paper.html.

[14] Sami Abu-El-Haija et al. “Youtube-8m: A Large-Scale Video Classification Benchmark”. In: *arXiv preprint arXiv:1609.08675* (2016). URL: https://arxiv.org/abs/1609.08675.

[15] Jia Deng et al. “Imagenet: A Large-Scale Hierarchical Image Database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5206848.

[16] Antonio Torralba, Rob Fergus, and William T. Freeman. “80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* (2008). URL: http://ieeexplore.ieee.org/abstract/document/4531741/.

[17] Shuiwang Ji et al. “3D Convolutional Neural Networks for Human Action Recognition”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (2013). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6165309.

[18] Karen Simonyan and Andrew Zisserman. “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: *Advances in Neural Information Processing Systems*. 2014. URL: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.

[19] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. “Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification”. In: (Mar. 28, 2016). arXiv: 1603.08561 [cs]. URL: http://arxiv.org/abs/1603.08561.

[20] Du Tran et al. “Learning Spatiotemporal Features With 3D Convolutional Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015. URL: http://www.cv-foundation.org/openaccess/content_iccv_2015/html/Tran_Learning_Spatiotemporal_Features_ICCV_2015_paper.html.

[21] Heng Wang et al. “Evaluation of Local Spatio-Temporal Features for Action Recognition”. In: *BMVC 2009-British Machine Vision Conference*. BMVA Press, 2009. URL: https://hal.inria.fr/inria-00439769/.

[22] Ronald Poppe. “A Survey on Vision-Based Human Action Recognition”. In: *Image and vision computing* (2010). URL: http://www.sciencedirect.com/science/article/pii/S0262885609002704.

[23] Jake K. Aggarwal and Michael S. Ryoo. “Human Activity Analysis: A Review”. In: *ACM Computing Surveys (CSUR)* (2011). URL: `http://dl.acm.org/citation.cfm?id=1922653`.

[24] Jose M. Chaquet, Enrique J. Carmona, and Antonio Fernández-Caballero. “A Survey of Video Datasets for Human Action and Activity Recognition”. In: *Computer Vision and Image Understanding* (2013). URL: `http://romisatriawahono.net/lecture/rm/survey/computer%20vision/Chaquet%20-%20Human%20Activity%20Recognition%20-%202013.pdf`.

[25] Martin Längkvist, Lars Karlsson, and Amy Loutfi. “A Review of Unsupervised Feature Learning and Deep Learning for Time-Series Modeling”. In: *Pattern Recognition Letters* (June 2014). ISSN: 01678655. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0167865514000221`.

[26] Samitha Herath, Mehrtash Harandi, and Fatih Porikli. “Going Deeper into Action Recognition: A Survey”. In: *arXiv prePrint:1605.04988* (2016). URL: `http://arxiv.org/abs/1605.04988`.

[27] Soo Min Kang and Richard P. Wildes. “Review of Action Recognition and Detection Methods”. In: *arXiv prePrint:1610.06906* (Oct. 21, 2016). URL: `http://arxiv.org/abs/1610.06906`.

[28] Aaron F. Bobick and James W. Davis. “The Recognition of Human Movement Using Temporal Templates”. In: *IEEE Transactions on pattern analysis and machine intelligence* (2001). URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=910878`.

[29] Yaser Sheikh, Mumtaz Sheikh, and Mubarak Shah. “Exploring the Space of a Human Action”. In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. IEEE, 2005. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1541250`.

[30] Florent Perronnin and Christopher Dance. “Fisher Kernels on Visual Vocabularies for Image Categorization”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4270291`.

[31] Hervé Jégou et al. “Aggregating Local Descriptors into a Compact Image Representation”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5540039`.

[32] Ivan Laptev. “On Space-Time Interest Points”. In: *International Journal of Computer Vision* (2005). URL: `http://link.springer.com/article/10.1007/s11263-005-1838-7`.

[33] Piotr Dollár et al. “Behavior Recognition via Sparse Spatio-Temporal Features”. In: *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*. IEEE, 2005. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1570899`.

[34] Heng Wang et al. “Action Recognition by Dense Trajectories”. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE, 2011. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5995407.

[35] Heng Wang and Cordelia Schmid. “Action Recognition with Improved Trajectories”. In: *Proceedings of the IEEE International Conference on Computer Vision.* 2013. URL: http://www.cv-foundation.org/openaccess/content_iccv_2013/html/Wang_Action_Recognition_with_2013_ICCV_paper.html.

[36] Petar Palasek and Ioannis Patras. “Action Recognition Using Convolutional Restricted Boltzmann Machines”. In: *Proceedings of the 1st International Workshop on Multimedia Analysis and Retrieval for Multimodal Interaction.* New York, NY, USA: ACM, 2016. URL: http://doi.acm.org/10.1145/2927006.2927012.

[37] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. “Unsupervised Learning of Video Representations Using LSTMs”. In: *Proceedings of The 32nd International Conference on Machine Learning.* 2015. URL: http://jmlr.org/proceedings/papers/v37/srivastava15.html.

[38] Moez Baccouche et al. “Sequential Deep Learning for Human Action Recognition”. In: *International Workshop on Human Behavior Understanding.* Springer, 2011. URL: http://link.springer.com/chapter/10.1007/978-3-642-25446-8_4.

[39] Christian Schüldt, Ivan Laptev, and Barbara Caputo. “Recognizing Human Actions: A Local SVM Approach”. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on.* IEEE, 2004. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1334462.

[40] Gül Varol, Ivan Laptev, and Cordelia Schmid. “Long-Term Temporal Convolutions for Action Recognition”. In: *arXiv preprint arXiv:1604.04494* (2016). URL: https://hal.inria.fr/hal-01241518/.

[41] Melvyn A. Goodale and A. David Milner. “Separate Visual Pathways for Perception and Action”. In: *Trends in neurosciences* (1992). URL: http://www.sciencedirect.com/science/article/pii/0166223692903448.

[42] Joe Yue-Hei Ng et al. “Beyond Short Snippets: Deep Networks for Video Classification”. In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on.* IEEE, 2015. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7299101.

[43] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *arXiv preprint arXiv:1604.06573* (2016). URL: http://arxiv.org/abs/1604.06573.

[44] Limin Wang, Yu Qiao, and Xiaoou Tang. “Action Recognition with Trajectory-Pooled Deep-Convolutional Descriptors”. In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on.* IEEE, 2015. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7299059.

[45] Jorge Sánchez et al. "Image Classification with the Fisher Vector: Theory and Practice". In: *International journal of computer vision* (2013). URL: http://link.springer.com/article/10.1007/s11263-013-0636-x.

[46] Joao Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *arXiv preprint arXiv:1705.07750* (2017). URL: https://arxiv.org/abs/1705.07750.

[47] Christian Szegedy et al. "Going Deeper With Convolutions". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html.

[48] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". In: (2012). URL: http://crcv-web.eecs.ucf.edu/papers/UCF101_CRCV-TR-12-01.pdf.

[49] Hildegard Kuehne et al. "HMDB: A Large Video Database for Human Motion Recognition". In: *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6126543.

[50] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Scandinavian Conference on Image Analysis.* Springer, 2003. URL: http://link.springer.com/chapter/10.1007/3-540-45103-X_50.

[51] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM International Conference on Multimedia.* ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2654889.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems.* 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-w.

[53] Sergey Ioffe. "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models". In: (Feb. 10, 2017). arXiv: 1702.03275 [cs]. URL: http://arxiv.org/abs/1702.03275.

[54] Kishore K. Reddy and Mubarak Shah. "Recognizing 50 Human Action Categories of Web Videos". In: *Machine Vision and Applications* (2013). URL: http://link.springer.com/article/10.1007/s00138-012-0450-4.

[55] Jingen Liu, Jiebo Luo, and Mubarak Shah. "Recognizing Realistic Actions from Videos "in the Wild"". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 2009. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5206744.

[56] Will Kay et al. "The Kinetics Human Action Video Dataset". In: *arXiv preprint arXiv:1705.06950* (2017). URL: https://arxiv.org/abs/1705.06950.

[57] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *arXiv preprint arXiv:1409.1556* (2014). URL: https://pdfs.semanticscholar.org/45c6/a85a359be655f459516919138a46ae516621.pdf.

[58] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. "Improving the Fisher Kernel for Large-Scale Image Classification". In: *European Conference on Computer Vision*. Springer, 2010. URL: http://link.springer.com/chapter/10.1007/978-3-642-15561-1_11.

[59] Mu Zhu. "Recall, Precision and Average Precision". In: *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* (2004).

[60] Gunnar A. Sigurdsson et al. "Asynchronous Temporal Fields for Action Recognition". In: (Dec. 19, 2016). arXiv: 1612.06371 [cs]. URL: http://arxiv.org/abs/1612.06371.

[61] Fabian Caba Heilbron et al. "Activitynet: A Large-Scale Video Benchmark for Human Activity Understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Heilbron_ActivityNet_A_Large-Scale_2015_CVPR_paper.html.

[62] Martın Abadi et al. "Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1603.04467* (2016). URL: https://arxiv.org/abs/1603.04467.

[63] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html.

[64] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* (2014). URL: http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf.

[65] Andrew Y. Ng. "Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: ACM, 2004. URL: http://doi.acm.org/10.1145/1015330.1015435.

[66] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *PMLR*. International Conference on Machine Learning. June 1, 2015. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[67] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc, 2017.

[68] Jianmin Chen et al. “Revisiting Distributed Synchronous SGD”. In: (Apr. 4, 2016). arXiv: 1604.00981 [cs]. URL: http://arxiv.org/abs/1604.00981.