Video Classification from scratch

Martine Toering
11302925

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Tao HU

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 28th, 2018

Abstract

Every minute, 400 hours of video are uploaded to YouTube. Video being a popular content type today has positively impacted the development of algorithms that attempt to extract semantic information from video. One challenging task gaining relevance in this context is video classification. In recent years several deep learning architectures have been proposed for the task. However, models often require initial model weights that are obtained by pretraining on large-scale datasets. Pretraining is time-consuming and sets up an unnecessary constraint for model design. Another reason for further attention is the fact that sequential data contains a considerable amount of temporal information in contrast to images.

This thesis researches whether it could be viable to remove the process of pretraining and train the network from scratch while combining additional utilization of spatiotemporal information. The proposed architecture is a multi-branch architecture composed of three components: (1) video classification (2) video tracking task (3) video direction task. With methods (2) and (3), representations of the video are learned in a self-supervised manner in which labels are automatically obtained. This model uses data more efficiently, as no other data or annotations are needed. The combined training of self-supervised learning and video classification is a novel approach that contributes to progress in both fields. The architecture proposed in this thesis shows a significant improvement and favourable results on the HMDB-51 dataset in comparison with random initialization. An error analysis of the network is included.

Contents

1 Introduction

Developing the ability to interpret the content of video automatically is vital given the present day abundance of this medium. The various applications of the semantic analysis of video include search operations, captioning and summarization. Providing an automatic categorization of videos would contribute greatly to user experience. For instance, automatic recognition and identification of human actions would immensely benefit many applications based on human-computer interaction.

Classification of an action present in a given video has been a fundamental problem in computer vision for decades. Recent advances in deep learning for image processing has led to considerable progress in the field of video classification. Numerous successful attempts have been made to extend deep neural networks such as Convolutional Neural Networks (CNNs or ConvNets) to the domain of video processing. While some architectures combine the output of two separate CNNs for spatial and temporal information (Simonyan and Zisserman 2014, Feichtenhofer et al. 2016), others use a Recurrent Neural Network (RNN) component in order to capture long-term temporal information (Donahue et al. 2017, Lakhal et al. 2018). The 3D CNN first introduced in 2015 uses entire video fragments as its input and performs 3D convolution operations (Tran, Bourdev, et al. 2015).

Despite the advancements, current methods for video classification still suffer from multiple problems. Firstly, capturing the spatiotemporal context is difficult and the computational cost associated is huge. Moreover, the methods are often based on transfer learning. In transfer learning, weights obtained by training deep networks on large-scale datasets are transferred to the task at hand. Using pretrained models often result in an increase in performance, but this method is very time-consuming and requires large amounts of manually labelled data. Use of pretrained models also limits the choice of architecture.

The objective of this thesis is to optimize video classification from scratch by employing a method that learns visual representations from video. The inherent structure of video can be used to learn representations without the use of any labels. This approach is part of a novel paradigm called self-supervised learning. Self-supervised learning methods are ways to compose a supervisory signal using the abundance of information in sequential data. Additionally, it does not rely on annotated datasets which are sparse and costly, for videos in particular.

This free spatiotemporal information can be obtained in a number of ways. Self-supervised learning methods have been proposed that obtain representations by future frame prediction (Goroshin et al. 2015), frame order verification (Misra et al. 2016) and solving three-dimensional puzzles (Kim et al. 2019). One recently proposed method is to learn feature representations by tracking (X. Wang et al. 2019a). Videos display a certain consistency in duration between the start and end of a given segment. Tracking is performed by sampling backward, followed by sampling forward. The learning objective for the CNN is to identify correspondences across the frames in this cycle. Another method of learning video representations is learning the direction of the video in time (Pickup et al. 2014). The learning objective is to identify whether videos are playing either forwards or backwards.

The proposed architecture is a multi-head neural network consisting of three branches. The main branch performs video classification. The second and third branch perform a *video tracking* task and a *video direction* task. The three branches share the backbone of the neural network that is a 3D Convolutional Neural Network based on ResNet-50 architecture. The combined learning of additional spatiotemporal information in videos with classification can benefit performance as well as efficiency. The proposed method also achieves the goal of video classification from scratch, as no other annotated datasets or pretraining are needed. The results show a significant improvement in performance on dataset HMDB-51 (Kuehne et al. 2011) in comparison to training from scratch.

In this thesis, the following contributions are made:

- To our knowledge, we are the first to combine self-supervised learning methods with video classification in one model that is trained from scratch.
- The combined application of both techniques is shown to be advantageous for performance in terms of accuracy.
- An error analysis of the model provides deeper substantiation of the model.

The sections are organized as follows. First, background knowledge on CNN and image classification is introduced after which related work will be discussed. Next, a description of the proposed architecture with the three branches is given. Discussion of experiments conducted on dataset HMDB-51 follows, after which a model evaluation is conducted. The thesis concludes with a discussion of the prospects and possibilities for future work.

2 Related Work

2.1 Image Classification

Rapid progress has been made in deep learning and image classification. The ImageNet challenge introduced in 2010 has been a driving force behind this progress. The annual competition was centered around the ImageNet dataset consisting of more than one million images (Deng et al. 2009). In 2012, architecture AlexNet based on Convolutional Neural Networks (CNN) won the contest with an outstanding top-5 error of 17% (Krizhevsky et al. 2012). More breakthroughs followed with VGGNet (Simonyan and Zisserman 2015), GoogleLeNet (Szegedy et al. 2015) and ResNet (He, X. Zhang, et al. 2016). The advancements are mainly due to the effectiveness of Convolutional Neural Networks.

Convolutional Neural Networks Convolutional Neural Networks are feed-forward neural networks that consist of multiple hidden layers where convolution, non-linearity or pooling operations are applied. In convolution operations kernels are applied to images to produce feature maps. The spatial structure of images is thus preserved. Non-linearity operations, like for instance Rectified Linear Input (ReLU) (Nair and Hinton 2010), are activation functions for neurons. Pooling combines previous output, which reduces the dimensionality and forms resilience against local variation. To serve as a classifier, a fully connected layer completes the network.

Residual Networks The recently obtained capability of scaling toward deeper networks has been another significant factor in the effectiveness of deep learning and CNNs. In 2015, Residual Networks (ResNets) were introduced and achieved a 3.57% error on ImageNet, thereby winning the contest for 2015 (He, X. Zhang, et al. 2016). ResNets are Convolutional Neural Networks that consist of blocks that have residual layers. Residual layers allow for skip connections, as illustrated in Figure 2.1. ResNets opened the possibility of training neural networks that consist of hundreds of layers. Figure 2.2 provides an overview of some ResNet architectures, ordered from relatively shallow to very deep.

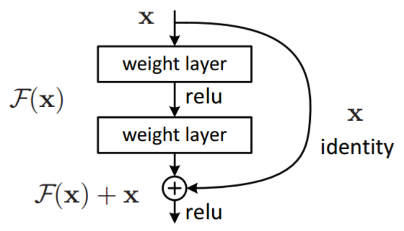
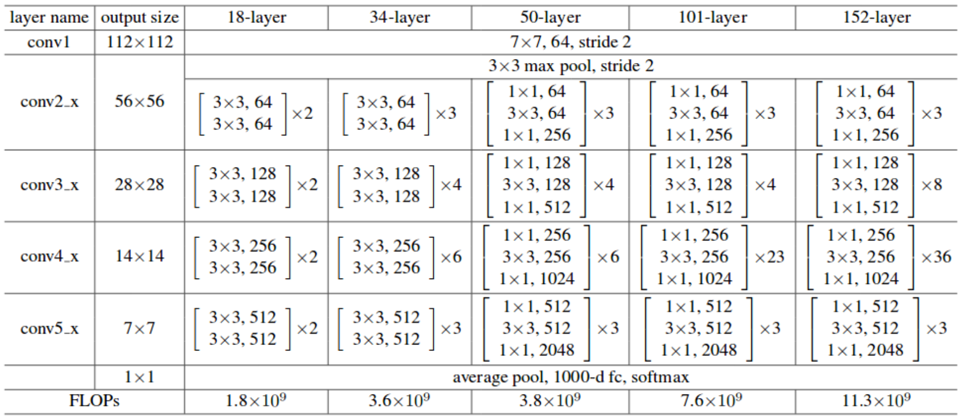


Figure 2.1: Residual Learning: a building block (He, X. Zhang, et al. 2016).



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.2: Architectures for ImageNet. Building blocks are shown in brackets, with the numbers of blocks stacked (He, X. Zhang, et al. 2016).

Neural Architecture Search Neural Architecture Search (NAS) has lately been given much attention (Zoph et al. 2018). Designing complex architectures manually is labour intensive. In this automated machine learning technique, the model is provided with sets of components and decisions. It searches for the best architecture within the search space. The architecture created by a Recurrent Neural Network (RNN) updates itself with its test results on a task (see figure 2.3). NAS results outperform state-of-the-art on several tasks including image classification (Elsken et al. 2018).

2.2 Video Classification

Video classification provides automatic labeling of clips according to their semantic content. Central to this field is the task of action recognition which involves recognizing human actions from videos. Video classification has been studied in computer vision for some time. Traditional approaches consist of computing local feature descriptors such as SIFT (Lowe 2004) and HOG (Dalal and Triggs 2005). Descriptors are quantized into bag-of-words representations (Csurka et al. 2004). Next, these representations are the input for classifiers like Support Vector Machines (SVMs) (Hearst 1998) or Random Forest (RFs) (Breiman 2001) algorithms.

A more recent approach is to use state-of-the-art Convolutional Neural Networks for images pretrained on ImageNet on video frames (Zha et al. 2015). Averages of these frame-level features are classified, for instance with SVMs. However, image-based deep learning methods do not utilize temporal information present in videos. Extension of traditional CNN architectures is required to make them more suitable for the domain of videos.

Current research on deep learning architectures for video classification can roughly be divided in two categories. Either the architectures use CNNs with 3D convolutions (single-

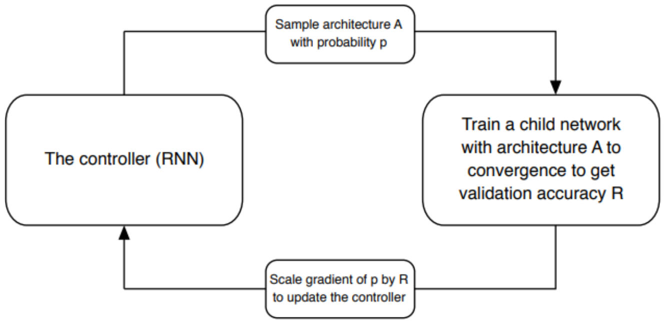

Sample architecture A with probability p
The controller (RNN)
Train a child network with architecture A to convergence to get validation accuracy R
Scale gradient of p by R to update the controller

Figure 2.3: Overview of Neural Architecture Search (Zoph et al. 2018).

stream architectures) (Tran, Bourdev, et al. 2015, Karpathy et al. 2014, Hara et al. 2018b) or temporal and spatial information are processed separately (two-stream architectures) (Simonyan and Zisserman 2014, Feichtenhofer et al. 2016, L. Wang et al. 2018). In the following, important findings and the current state-of-the-art on video classification are presented.

2.2.1 Two-stream architectures

In 2014, Simonyan and Zisserman introduced a two-stream CNN architecture that included two separate networks (Simonyan and Zisserman 2014). One network was aimed at spatial information and the other was aimed at temporal information. Optical flow, the pattern of the apparent motion in a scene, is modelled separately and is fed into the temporal network. Figure 2.4c provides an abstract visualization of a two-stream architecture. Modelling optical flow separately is an attempt to make optimal use of spatiotemporal information and motion features in video. Much of the subsequent work in the field has been built upon this type of architecture. Feichtenhofer *et al.* proposed fusing the two networks at the last convolutional layer instead of the softmax layer (Feichtenhofer et al. 2016). In accordance with the 2014 paper by Simonyan and Zisserman was the single RGB frame for the spatial stream and the optical flow vectors for the temporal stream. This network is visualized in figure 2.4d. Wang *et al.* introduced the Temporal Segment Network (TSN) architecture with a new sampling method for modelling long-range temporal structure (L. Wang et al. 2018). The Hidden Two-Stream architecture proposed by Zhu *et al.* generates optical flow implicitly within the network, as generating it separately beforehand is time-consuming (Zhu et al. 2017).

LSTM Others have tried to incorporate a Recurrent Neural Network (RNN) component in two-stream architectures in order to capture long-term temporal information (Donahue et al. 2017, Lakhal et al. 2018). A RNN is a neural network that allows for cycle connections. A Long short-term Memory (LSTM) is a RNN that can model long-term information and does not have vanishing gradient problems (Hochreiter 1998). Modelling long-term information can be beneficial as target actions generally consist of multiple smaller actions, for instance cooking and sports. These networks were further explored by comparing different pooling strategies (Yue-Hei Ng et al. 2015). A visualization of one

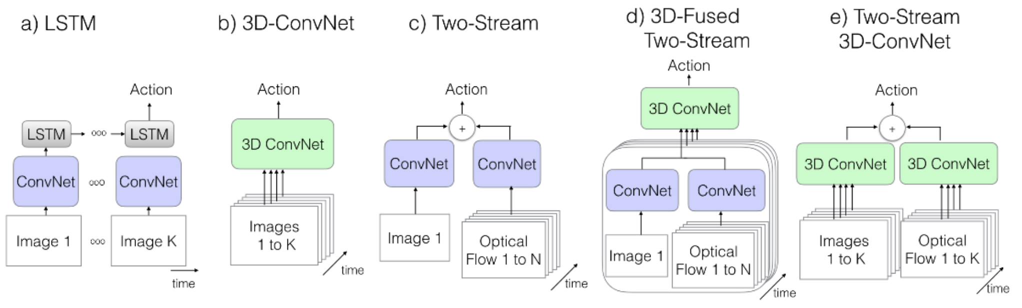


Figure 2.4: Video architectures (Carreira and Zisserman 2017).

possible architecture with a LSTM is shown in figure 2.4a.

Two-Stream 3D CNN One important proposal by Carreira *et al.* inflates a 2D CNN by expanding the kernels into 3D, enabling the use of pretrained image classification models (Carreira and Zisserman 2017). This I3D architecture improved considerably on the state-of-the-art and 98.0% accuracy on the benchmarking dataset UCF-101 (Soomro et al. 2012) was reported. Figure 2.4e shows such a network architecture. The T3D network was subsequently built on this research (Diba et al. 2017). It proposed a layer that models variable temporal convolution kernel depths to capture long-range information.

2.2.2 Single-stream architectures

In 2015, Tran *et al.* introduced the C3D architecture based on a 3D Convolutional Neural Network (Tran, Bourdev, et al. 2015). Unlike a 2D CNN that operates on a single image, a 3D CNN takes a complete stack of frames as input and performs 3D Convolution operations. Tran *et al.* found that 3D CNNs perform better in the capture of spatiotemporal information than 2D CNNs and that small $3 \times 3 \times 3$ convolution kernels perform the best. Another influential paper by Karpathy *et al.* made a comparison between methods of fusing temporal information on the Sports-1M dataset consisting of 1 million videos from YouTube (Karpathy et al. 2014).

The performance of 3D Convolutional Neural Networks lagged behind that of two-stream methods and handcrafted methods. The fact that 3D CNNs in particular are computationally expensive because of the substantial number of their parameters was an additional disadvantage. Because of the development of larger-scale benchmarking datasets like ActivityNet (28.000 videos) and Kinetics (300.000 videos), 3D CNNs now seem to start outperforming other methods (Tran, H. Wang, et al. 2018, Carreira and Zisserman 2017). One study examines different ResNet architectures for 3D CNNs from relatively shallow architectures to deep architectures. It was found that the size of the Kinetics dataset is sufficient for training deep neural networks (Hara et al. 2018b).

2.3 Video Representation Learning

The application of unsupervised learning in video classification would be highly desirable. Annotated datasets are sparse and the manual annotation of video is time-consuming as well as complex. Recently more research has been devoted to the design of deep learning architectures that are capable of learning video representations without using any labels. This method is referred to as self-supervised learning (Zisserman 2018, Doersch and Zisserman 2017, Fernando et al. 2017). Instead of annotations, the video itself forms the supervision.

There are multiple approaches to utilizing the raw spatiotemporal information in videos. Firstly, a neural network can be employed to do future frame prediction (Goroshin et al. 2015), Vondrick et al. 2016). For instance, one work proposes an encoder-decoder LSTM architecture that learns representations (Srivastava et al. 2015). Another approach is to let a neural network perform a sequential ordering task. Misra *et al.* propose a CNN based architecture that verifies the order of shuffled frames that are given as input (Misra et al. 2016). Another work presents a multi-branched network that identifies the wrong temporal order frame by using stacks of frame differences (Fernando et al. 2017). Lee *et al.* propose a network that learns representations by sorting frame sequences. It is also shown that this method can be used as a pretrained model for video classification (Lee et al. 2017).

Learning the direction of time in the video, in other words determining whether the video plays forwards or backwards, is another possible approach (Pickup et al. 2014). Various motion representations are used to learn the time direction. Wang *et al.* use the consistency in a cycle in time as free supervisory signal for learning visual representations (X. Wang et al. 2019a). Furthermore, a model can perform cubic puzzles (Kim et al. 2019) and the representations can be transferred to video classification. Solving jigsaw puzzles has also been studied (Noroozi and Favaro 2016). Gan *et al.* explore geometry for the video domain (Gan et al. 2018).

2.4 Training from scratch

State-of-the-art results for image classification and video classification rely heavily on transfer learning (Torrey and Shavlik 2010). In most studies, the goal of training a model from scratch is to make a comparison with proposed methods (Hara et al. 2018b, Carreira and Zisserman 2017, Diba et al. 2017, Tran, Bourdev, et al. 2015). These influential studies present methods that use pretrained models which have been trained on large-scale image or video datasets. Methods that learn video representations with self-supervised learning often train from scratch with the interest of establishing a baseline for comparison (Misra et al. 2016, Lee et al. 2017, Gan et al. 2018).

More research has on the other hand been devoted to removing the pretraining process in other tasks. It was found that for object detection and image segmentation, state-of-the-art results can be obtained on the COCO dataset (Lin et al. 2014) without any pretraining. For this, the model needs to go through a sufficient number of training iterations (He, Girshick, et al. 2018). These remarkable results challenge the conventional method of pretraining and fine-tuning. This aspect of training from scratch was again shown by Deeply Supervised Object Detector (DSOD) (Shen et al. 2017) based on DenseNet (Huang et al. 2017). Experiments on multiple datasets show that this model performed better than current methods on object detection. The Dense In Dense (DID) network trains image segmentation from scratch and gives favourable results on several datasets (Hu et al. 2018).

Here, we propose training from scratch in video classification. The goal is to show that video classification does not have to rely on extensive pretraining and that training from scratch can obtain competitive results in an efficient way.

2.5 Multi-task learning

Multi-task learning is the approach of combining multiple tasks and training them together in one model. This is often done for related tasks where the tasks share representations in one model. Multi-task learning has been applied in several areas of deep learning and supervised learning (Z. Zhang et al. 2014, Ranjan et al. 2017, Abdulnabi et al. 2015). Some works have been proposed in video classification (Zhi et al. 2017, Luvizon et al. 2018) and in self-supervised learning (Doersch and Zisserman 2017, Fernando et al. 2017). For video classification multi-task learning has so far received limited attention. More research on self-supervised learning could aid in expanding the subject and presently seems imminent.

An improvement in performance is always found as result of combining multiple self-supervised learning tasks into one trainable model via a multi-head architecture (Doersch and Zisserman 2017). This thesis proposes combining multiple self-supervised learning tasks and video classification. The belief is that training video classification and self-supervised learning simultaneously can improve performance yet further.

3 Approach

In this section the multi-branch network method with its three components is described. The underlying architecture is explained, after which details of the different methods are given.

Proposed model The network architecture is composed of four parts, namely one backbone and three branches. The backbone of the neural network is shared between the three branches. An abstract visualization of the multi-head model is shown in figure 3.1. The input is a sequence of frames of length s. This stack of frames is input to the neural networks backbone which consist of four convolutional layers. The output from the backbone (figure 3.1a) is subsequently fed into each of the three branches. The model has three different outputs. The loss of the three branches are combined with a weighted loss function

$$l = l_{classification} + w_1 \cdot l_{tracking} + w_2 \cdot l_{direction} \tag{3.1}$$

where w_1 is the weight for the video tracking branch and w_2 is the weight for the video direction branch. The video classification branch (figure 3.1b) performs video classification and the video direction branch (figure 3.1d) performs binary classification. Both branches output prediction vectors. The video tracking branch (figure 3.1c) outputs a patch feature from tracking.

3.1 Architecture

The proposed architecture is a ResNet-50 architecture based on the concept of Inflated CNNs.

ResNet-50 ResNet-50 is a model consisting of four Bottleneck blocks with 3, 4, 6 and 3 units (He, X. Zhang, et al. 2016). As seen in figure 2.2, each unit has three convolutional layers. After each of these layers in a unit, ReLU (Nair and Hinton 2010) and batch normalization (Ioffe and Szegedy 2015) layers are applied. We follow He *et al.* and refer to the blocks that contain these units as conv1, conv2, etc. The same concepts of a two-dimensional ResNet-50 extend to 3D-ResNets. Table 3.2 shows a regular 3D ResNet with a model depth of 50, such as proposed by Hara *et al.* (Hara et al. 2018b). The kernel sizes of the second convolutional layer typically are $3 \times 3 \times 3$ with the notation being $d \times h \times w$. The kernel sizes of first and third layers are $1 \times 1 \times 1$. The shortcut connections join the begin of the block to just before the ReLU of the third layer. Both Identity shortcuts (type A) or Projection shortcuts (type B) can be used (He, X. Zhang, et al. 2016).

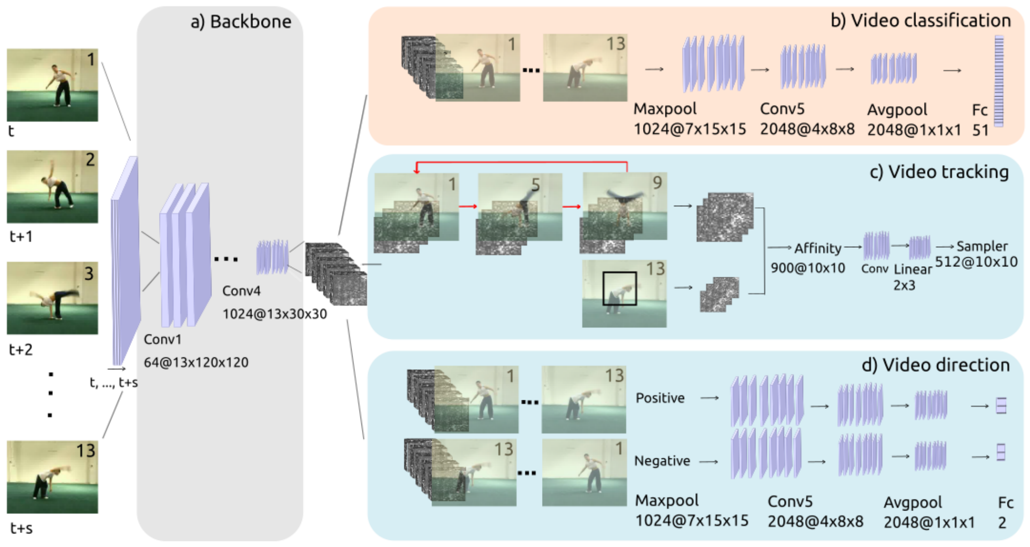


Figure 3.1: The proposed multi-head architecture with video tracking and video time direction branches next to the video classification branch. The input is a sequence of frames from t to $t + s$ where t is the starting frame and s is the sample duration. The output from the backbone is feature maps that are given to the three branches. The procedure for a single video from the batch is shown.

Inflated ResNet-50 Inflating 2D ResNets to 3D ResNets has recently been introduced (Carreira and Zisserman 2017). One popular use is the transfer of 2D ImageNet weights. Using this method, both the convolutional filters and the pooling kernels are provided with a temporal dimension. Carreira *et al.* inflate the Inception-v1 (Szegedy et al. 2015), or GoogLeNet, architecture. An inception submodule applies both $1 \times 1 \times 1$ and $3 \times 3 \times 3$ convolutions and the output of filters is concatenated before the next layer. Submodules use ReLU activation.

Our model uses a ResNet-50 architecture (He, X. Zhang, et al. 2016) that is inflated into a 3D-ResNet-50. Throughout the model, different filters and pooling kernels are applied between the backbone and the branches.

3.2 Backbone

Table 3.1 shows an overview of the architecture for the backbone. The backbone consists of the four blocks conv1, conv2, conv3 and conv4. These last three blocks contain 3, 4 and 6 units respectively. The kernel sizes in all second convolution layers in the blocks are $1 \times 3 \times 3$. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. The first $1 \times 7 \times 7$ convolutional layer conv1 outputs 64 channels and max pooling is applied with stride $1 \times 2 \times 2$. Next, conv2, conv3 and conv4 follow respectively. As can be seen in 3.1, downsampling is performed on the height and width dimension in conv2 and conv3. However, downsampling is not performed in the

stage	output	Backbone	
conv1	13×120×120	$1 \times 7 \times 7$, 64, stride 1,2,2	
conv2_x	13×60×60	$1 \times 3 \times 3$ maxpool, stride 1,2,2	
		$\begin{bmatrix} 1 \times 1 \times 1, 64 \\ 1 \times 3 \times 3, 64 \\ 1 \times 1 \times 1, 256 \end{bmatrix}$	$\times$ 3
conv3_x	13×30×30	$\begin{bmatrix} 1 \times 1 \times 1, 128 \\ 1 \times 3 \times 3, 128 \\ 1 \times 1 \times 1, 512 \end{bmatrix}$	$\times$ 4
conv4_x	13×30×30	$\begin{bmatrix} 1 \times 1 \times 1, 256 \\ 1 \times 3 \times 3, 256 \\ 1 \times 1 \times 1, 1024 \end{bmatrix}$	$\times$ 6

Table 3.1: The backbone of the proposed architecture based on inflated 3D-ResNet-50. Downsampling on the height and width dimension is performed at conv2_x and conv3_x.

stage	3D-ResNet-50	
conv1	$7 \times 7 \times 7$, 64, stride 1,2,2	
conv2_x	$3 \times 3 \times 3$ maxpool, stride 2,2,2	
	$\begin{bmatrix} 1 \times 1 \times 1, 64 \\ 3 \times 3 \times 3, 64 \\ 1 \times 1 \times 1, 256 \end{bmatrix}$	$\times$ 3
conv3_x	$\begin{bmatrix} 1 \times 1 \times 1, 128 \\ 3 \times 3 \times 3, 128 \\ 1 \times 1 \times 1, 512 \end{bmatrix}$	$\times$ 4
conv4_x	$\begin{bmatrix} 1 \times 1 \times 1, 256 \\ 3 \times 3 \times 3, 256 \\ 1 \times 1 \times 1, 1024 \end{bmatrix}$	$\times$ 6
conv5_x	$\begin{bmatrix} 1 \times 1 \times 1, 512 \\ 3 \times 3 \times 3, 512 \\ 1 \times 1 \times 1, 2048 \end{bmatrix}$	$\times$ 3
	avgpool, fc, softmax	

Table 3.2: A more standard ResNet-3D model based on the 50-layer architecture. Downsampling is performed at conv2_x, conv3_x, conv4_x and conv5_x.

depth dimension in any of the blocks, as opposed to more regular 3D CNNs such as the 3D-ResNet-50 shown in table 3.2. Furthermore, the stride of conv4 is reduced from 2 to 1, outputting the same output size as the block before. The output from the backbone is a feature map of 1024 channels with output size $13 \times 30 \times 30$. Next, these features are fed into each of the three branches.

3.3 Video classification

The top branch shown in figure 3.1b performs video classification. This branch is regarded as the main branch, as video classification is the core objective. This part of the model is added to the backbone and adapted to fit. The 13-dimensional feature maps are first put into the max pooling layer. Max pooling reduces the output size to $7 \times 15 \times 15$. The conv5 layer that follows performs convolutions that are identical to conv5 from 3D-ResNet-50 shown in table 3.2. Therefore, the output size reduces to $4 \times 8 \times 8$. This 2048-channeled output is fed into a average pool layer and a fully connected layer. For each video in a batch, the fully connected layer outputs a prediction vector with a size correlated to the number of classes. The loss for video classification is cross-entropy and is defined as

$$l_{classification} = -\sum_{i}^{N} q_i \, log(p_i) \tag{3.2}$$

where N is the number of classes, q is the binary target value for a class and p is the predicted probability for a class.

3.4 Video tracking

The tracking branch is based on a recently proposed self-supervised learning method from Wang *et al.* (X. Wang et al. 2019a). The model learns video feature representations by tracking a small patch (80×80) of a frame (240×240) backwards and forwards in a cycle. The goal of the tracking operation is to find similarity between patches in the video. The tracking loss is the inconsistency between the start and the end of the cycle, as seen in figure 3.2. While minimizing the consistency, the model learns representations which in turn improve the tracking.

As seen in figure 3.1c, the input to this branch is a short video with gaps in between the frames. The gaps are included in order to capture the long-term changes. The length of the video is determined by k, the number of past frames and the gap between the frames by f. In our case, f is set to 4 and k is set to 3, setting the tracking operation exactly within the sample duration of $s = 13$ as seen in the figure. Each step in the cycle, the affinity is computed from feature maps of a current patch (10×10) and feature maps of one target image (30×30). A localizer consisting of two convolutional layers and one linear layer takes this affinity matrix and then outputs localization parameters. These are input for the bilinear sampler which produces a new patch feature map.

The features of the initial patch and this newly produced patch are input to the loss function. The complete loss consists of the feature similarity between the two patch features, the loss of skipping in time and the tracking loss (X. Wang et al. 2019a):

$$l_{tracking} = \sum_{i=1}^{k} l^i_{sim} + \lambda \cdot l^i_{skip} + \lambda \cdot l^i_{long} \tag{3.3}$$

where $\lambda = 0.1$ and k is the number of cycles. The tracking loss l^i_{long} is the euclidean distance between the two patch features. The loss l^i_{skip} is based on the euclidean distance between the initial patch and the patch from a skip cycle that can leap to further frames. The negative Frobenius inner product (Amir-Moéz and Davis 1960) is calculated to obtain the feature similarity of the two patch features l^i_{sim}. For more information on this method, see Wang *et al.* (X. Wang et al. 2019a).

3.5 Video direction

The third branch is a self-supervised task in which the model learns the direction in time of the video. The learning objective is to find whether the video is playing forwards or

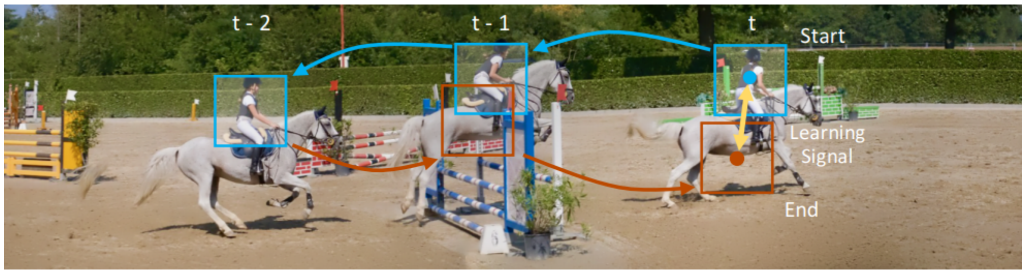


Figure 3.2: A Cycle in Time (X. Wang et al. 2019a).

backwards (Zisserman 2018). In this idea from Zisserman *et al.*, the procedure is proposed as a binary classification problem where positive examples are forward videos and negative examples are backward videos. The binary labels are composed in the task itself and it is therefore a self-supervised learning approach.

In the video direction branch, we attempt to incorporate this idea. The third branch (figure 3.1d) is a binary classification task where positive examples are video frame features ordered identical to the input t, $t+1$, ..., $t+s$, while negative examples are video features from frames $t+s$, $t+s-1$, ..., t. These feature maps are thus both the same size with $13 \times 30 \times 13$ and 1024 channels. The forward and backward versions of the feature maps are evenly divided over the batch size.

As can be seen in figure 3.1d, the feature maps from the backbone go through the video direction branch by an almost identical procedure as in the video classification branch. The feature maps are fed into max pooling layer, conv5, average pooling layer and a fully connected layer. The first three perform downsampling in all dimensions. The fully connected layer is again a 2048-dimensional fully connected layer, in this case outputting a binary classification for each video. The loss for the video direction branch is cross-entropy, see equation 3.2. Because the video direction branch is a binary classification problem, cross-entropy can also be formulated as

$$l_{direction} = -\left(q\,log(p) + (1-q)\,log(1-p)\right) \tag{3.4}$$

where q is again the binary target value for a class and p is the predicted probability for a class.

4 Experimental Results

In this section, the training setting is described and an attempt is made to evaluate the performance of the proposed architecture with experiments and comparisons. The section concludes with a discussion on parameters and an ablation study.

4.1 Dataset

The evaluation is performed on the dataset HMDB-51 (Kuehne et al. 2011). HMDB-51 has been adopted as a benchmarking dataset in video classification alongside UCF-101 (Soomro et al. 2012). The dataset is categorized into 51 human action classes with a minimum of 100 trimmed videos in each class. The total duration of the dataset is 6 hours and the clips are mostly taken from movies. HMDB-51 is considered a relatively challenging dataset due to large variation in viewpoint, background and camera motion. Furthermore, there are some particularly similar actions present. Examples are the actions sword, sword exercise and drawing sword, as well as the actions kick and kick ball.

Three splits are provided with the dataset, each including a train and test set. The training set consist of a number of 3.570 videos and the test set consists of 1.530 videos. The results are evaluated on the mean accuracy over the three splits of HMDB-51, following the official evaluation protocol for this dataset.

The model operates on a stack of 13 RGB frames ($s = 13$). The frame rate for the dataset is kept at the original 30 frames per second (fps).

4.2 Implementation details

The model is implemented using the PyTorch framework (Paszke et al. 2017). The implementation is based on different GitHub repositories, namely inflated ConvNets code for PyTorch (Hasson 2017), 3D-ResNets-PyTorch from Hara *et al.* (Hara et al. 2018a) and the TimeCycle repository from Wang *et al.* (X. Wang et al. 2019b).

Data augmentation The benefit of using data augmentation to avoid overfitting is shown in numerous studies (Krizhevsky et al. 2012, He, X. Zhang, et al. 2016, Huang et al. 2017). We adopt data augmentation in several ways. The starting frame t is randomly chosen from the video to obtain a sample of length 13. Next, we perform random crop of the frames at training time to get frames of 240×240. The input size for the network is therefore $13 \times 240 \times 240$. A random horizontal flip is applied with 50% probability as another form of data augmentation. At test time, a center crop is applied and no flip is performed.

Training and optimization Training is done via backpropagation through the network. The loss is calculated with the cross-entropy loss function. The model is optimized with Adam optimizer (Kingma and Ba 2015). Adam is an extension of mini-batch

Stochastic Gradient Descent (SGD) with momentum. The initial learning rate is set to 0.0002 and the weight decay is 0.0001. The settings for the momentum are $\beta_1 = 0.5$ and $\beta_2 = 0.999$. These values are used in computing running averages of gradients and the squares of gradients respectively (Kingma and Ba 2015). The batch size for the model is 4. The model is trained on 1 GPU (GTX 1080 Ti) for 100 epochs. Training the model takes around 30 hours.

Weight parameters We set our weight parameters in our loss function to $w_1 = 25$ and $w_2 = 2$. These values were approximated in order to balance the losses.

Learning rate schedule The learning rate is further reduced based on the results for three randomly sampled clips per video. The learning rate is reduced whenever the loss plateaus. We set the patience to 10, so that after 10 epochs with no improvement the learning rate is reduced.

Testing and evaluation Video-level predictions for testing are obtained by averaging over non-overlapping clips for each video. This method is also referred to as the sliding window method (Hara et al. 2018b, Tran, Bourdev, et al. 2015). The evaluation is based on the Hit@k method (Tran, Bourdev, et al. 2015, Karpathy et al. 2014). The value for Hit@k is the number of samples that contain at least one of the ground truth labels in the top k predicted labels.

4.3 Comparisons

In the following, our results are shown and the method is compared with baselines and other results from related work.

4.3.1 Comparison with baseline

Our baseline is training a model from scratch where the weights are randomly initialized. The baseline result is based on a 3D-ResNet-50 architecture from scratch using implementation from Hara *et al.* (Hara et al. 2018b). Training settings such as the number of epochs, the batch size and the input sizes are set identically to the settings of our proposed architecture. The model uses a SGD optimizer and following Hara *et al.*, an initial learning rate of 0.1 is set.

Table 4.1 shows the result for the baseline of training from scratch. The table shows an overview of the top-1 accuracy (Hit@1) over the three splits of HMDB-51 and the average over these three splits. The table also shows an overview of the results for our proposed method. The result for both of these methods are based on the best test result for alternating epochs. Our proposed method shows an improvement over the baseline of **7.0**% accuracy.

From the results in table 4.1, we can see that better performance can be achieved via our method than via merely training the network from scratch.

Method	Backbone	Pretraining	Supervision	Split (%) 1	2	3	Avg (%)
Misra et al.*	VGG-M-2048	-	-	14.8	-	-	13.3
Misra et al.	VGG-M-2048	UCF-101	-	-	-	-	15.2
Misra et al.	VGG-M-2048	-	Order verification	19.8	-	-	18.1
Hadsell et al.	VGG-M-2048	-	Invariant mapping	16.3	-	-	-
Mobahi et al.	VGG-M-2048	-	Temporal coherence	15.9	-	-	-
Gan et al.*	FlowNet	-	-	-	-	-	13.4
Gan et al.	FlowNet	-	Geometry (a)	-	-	-	22.6
Gan et al.	FlowNet	-	Geometry (b)	-	-	-	23.3
Lee et al.*	CaffeNet	-	-	-	-	-	16.3
Baseline*	3D-ResNet-50	-	-	18.0	17.8	15.8	17.2
Our model	3D-ResNet-50	-	Tracking + direction	24.1	24.7	23.9	**24.2**
Lee et al.	CaffeNet	UCF-101	Order verification	-	-	-	22.1
Lee et al.	VGG-M-2048	UCF-101	Order verification	-	-	-	23.8
Pretraining	3D-ResNet-50	Kinetics	-	55.2	53.7	54.1	54.4

Table 4.1: Top-1 accuracies over the three splits of HMDB-51 and the average accuracy for different initialization methods. Methods indicated with * are trained from scratch.

4.3.2 Comparison with pretraining

Next, we perform a comparison between our method and results from a model that uses conventional pretraining. The model used here is the same 3D-ResNet-50 as the baseline, except that we now finetune the model with an already pretrained model. The pretrained model is provided by the implementation of Hara *et al.* (Hara et al. 2018b) and is pretrained on the large video dataset Kinetics. The parameter settings are again as similar as possible to the settings from our model. The learning rate is set to 0.001 following Hara *et al.* and the highest result is once more taken from 50 tests.

Results of finetuning this pretrained model on HMDB-51 are shown in table 4.1. This improvement is significant when compared to training from scratch as well as compared to our method. However, there are several aspects to take into consideration. First, the Kinetics dataset (300.000 videos) is exceptionally large in comparison to HMDB-51 (7.000 videos). Second, the temporal input size of this pretrained model is larger than our sample duration $s = 13$ and is trained for a longer period.

Our improvement over random initialization does show a clear improvement. This is despite using a relatively small dataset, using a medium-sized ResNet and training the model for 100 epochs. When considered in combination with the additional advantages of not using pretraining, our results seem to indicate that combined training self-supervised learning with video classification is a reasonable and promising approach.

4.3.3 Comparison with other self-supervised methods

We show a comparison of the performance of our proposed method and other work on self-supervised learning in table 4.1. Each of these works use self-supervised learning methods as pretraining for video classification. The weights for video classification are in these cases initialized with the weights from models trained on a self-supervision task. Methods indicated with * are trained from scratch.

One research trains from scratch VGG-M-2048 (Simonyan and Zisserman 2014), a Two-stream model. This is compared with the same model initialized with weights from their pretrained network (Misra et al. 2016). This self-supervised network learns by temporal order verification. A tuple of frames is extracted from a video and is then used to determine whether they are in the correct order. The improvement on performance is 4.7% accuracy. The result is in accordance with the result for pretraining the model on UCF-101 without self-supervision, which leads to worse performance.

Misra *et al.* have also trained other self-supervised methods in order to provide more comparisons for their method. The methods also verify the order of frames and additionally perform temporal smoothing over frames. One (Hadsell et al. n.d.) uses manhattan (l_1) distance, the other (Mobahi et al. 2009) uses euclidean (l_2) distance. The results obtained where only a minor improvement over the baseline. Our model both outperforms the proposed order verification method and the comparison methods.

Feature representations were learned from solving geometry tasks and were then transferred to video classification (Gan et al. 2018). Results for two different geometry methods were compared with their baseline from scratch. Our model outperforms these tasks although their improvement over baseline is slightly higher.

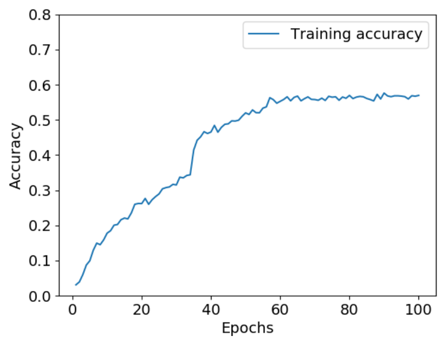
Verifying the temporal order of frames is used as supervision in another work (Lee et al. 2017). Transferring a pretrained self-supervised model to video classification on HMDB-51 achieves an improvement of 6.4% over the baseline. However, this model is pretrained (supervised) on the UCF-101 dataset. This means that our model outperforms a model that both uses conventional pretraining as well as self-supervised learning. The potential of training from scratch is supported by this case.

4.4 Analysis

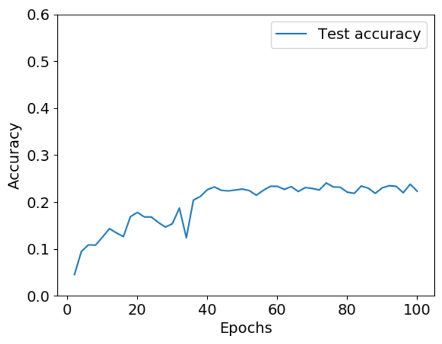
Here, we will perform model evaluation and error analysis of the model.

Accuracy Figure 4.1a provides an overview of the training accuracy for our proposed model in the training phase. The accuracy is plotted as a function of the number of epochs. The curve is based on experiments conducted on HMDB-51 split 1. The training accuracy reaches around 60% and only marginally improves after approximately 60 epochs. The plot in figure 4.1b shows the test accuracy over 50 test results for alternating epochs.

Loss Figure 4.2 shows how the loss is minimized in the training phase for the model on HMDB-51 split 1. The loss overall decreases well and the sudden drop indicates where



(a) Training accuracy



(b) Test accuracy

Figure 4.1: Curve of the training accuracy and test accuracy for split 1 of HMDB-51.

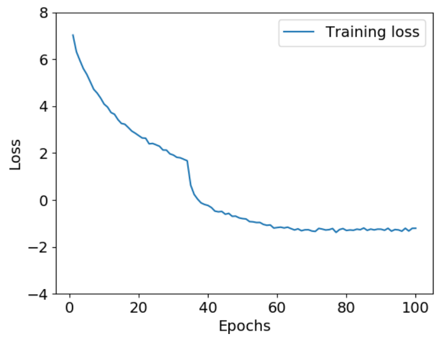


Figure 4.2: Curve of the training loss for split 1 of HMDB-51.

the learning rate is decreased.

Results A confusion matrix for the predictions is shown in figure 4.3. The ground truth labels are placed on the vertical axis and predicted labels on the horizontal axis. The classes are arranged according to the group types provided by the dataset (Kuehne et al. 2011). The group types are indicated in red and are facial actions (F), body movements (B), body movements with object interaction (BO) and body movement for human interaction (HI). The confusion matrix shows that the category body movement for human interaction (HI) is the group of categories with the least performance. This set of categories includes classes such as hug, kiss and punch. Overall, the correctly classified classes are noticeably visible on the diagonal axis.

Samples Figure 4.4 shows some video samples and their predicted labels with confidences. The correct label is indicated in blue. The two samples at the top left are samples that are classified correctly with a high confidence. The samples at the top right are correctly classified by a narrow margin. The bottom row are misclassified samples.

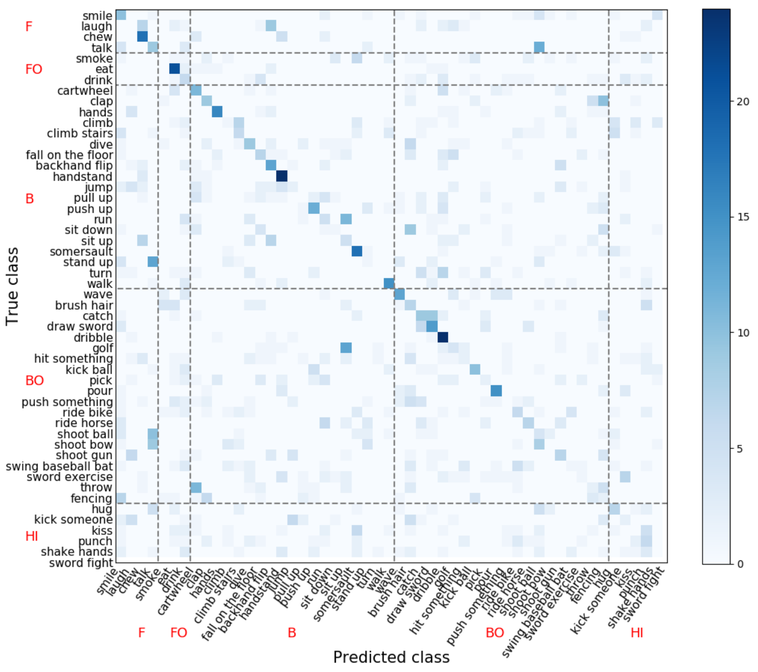


Figure 4.3: Confusion matrix for prediction results on split 1 of HMDB-51. The classes are grouped according to the five types where F is facial actions, FO is facial actions with object manipulation, B is body movements, BO is body movements with object interaction and HI is body movement for human interaction (Kuehne et al. 2011).

In this row, only the samples at the bottom left have the correct prediction in their top 5 predictions. We can observe that at times the network makes mistakes between similar looking actions such as between shake hands and shoot gun as well as between hit and hug.

4.4.1 Discussion on parameters

Input sizes Increasing the input size of a 2D CNN can greatly improve performance (Wu et al. 2015). This is certainly expected when the temporal input size of a 3D CNN is increased, as more frames and thus more information is provided. Increasing input sizes was however limited by available GPU resources. We chose to lower the batch size in order to allow for a temporal input of $s = 13$. This temporal duration could then exactly fit the video tracking branch with $k = 3$ and $f = 4$, as mentioned before. An attempt was made to increase the duration of the input by another method. We converted the dataset from 30 fps to 15 fps to allow for a frame gap of $f = 2$. This allows for two improvements. We next could either create a longer distance between the past frames and the target image, as the method from Wang *et al.* originally suggests (X. Wang et al. 2019a), or we could simply perform the tracking on a longer sample by increasing k. Initial experiments were conducted on the model with only the video classification and video tracking branch in place. These indicated that lowering the fps did not result in a

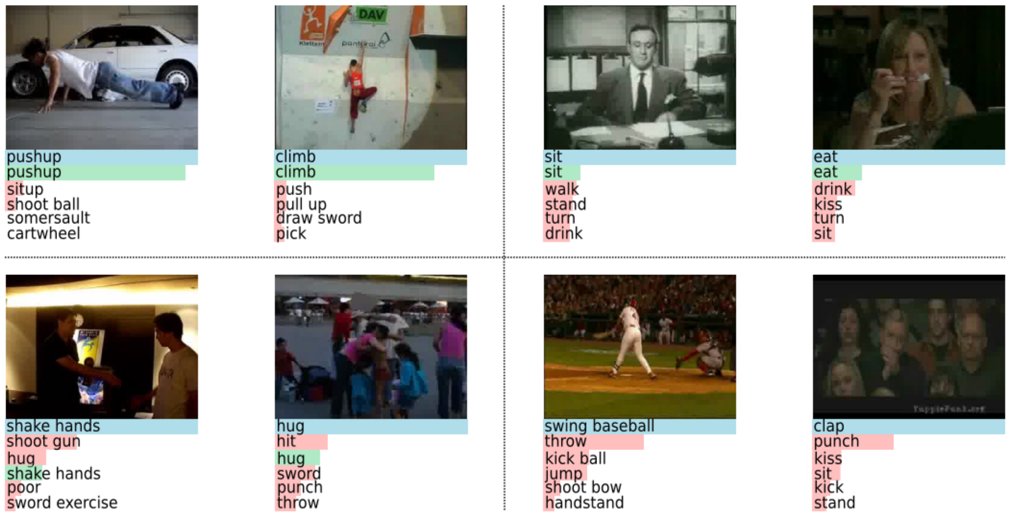


Figure 4.4: Predictions for several samples from the HMDB-51 dataset with ground truth label in blue, correct prediction in green and incorrect prediction in red. Confidences are indicated by bars. Top row shows samples that were correctly classified. Bottom row shows samples that were incorrectly classified.

w_1, w_2	**25, 2**	**50, 8**	**100, 8**
HMDB-51 (split 1)	24.1	25.4	19.2

Table 4.2: Effect of different weights on the best test result for HMDB-51 split 1.

considerable improvement.

Determining the weights The loss function is a combined loss function with $w_1 = 25$ and $w_2 = 2$. The aim is to analyze the impact of different weights on accuracy. Table 4.2 shows an overview of the test results for these different weights. Figure 4.5 shows a plot comparison of the test accuracy. The accuracy for our chosen weights $w_1 = 25$ and $w_2 = 2$ initially increases at a faster rate. However, table 4.2 shows that the best result over 100 epochs on split 1 is obtained by using $w_1 = 50$ and $w_2 = 8$. Presumably, using these weights and weighing the self-supervised methods by a larger weight than the video classification would obtain a better result over a longer period.

4.4.2 Ablation studies

Here, one study will be performed specifically on the third branch, the video direction branch. The motivation for this ablation study is that the third branch did not seem to improve as much as the second tracking branch. One cause could be that the HMDB-51 dataset, which consist of trimmed videos with small variation within videos, is simply not suitable for this method. Another cause could be that this method does not fit this

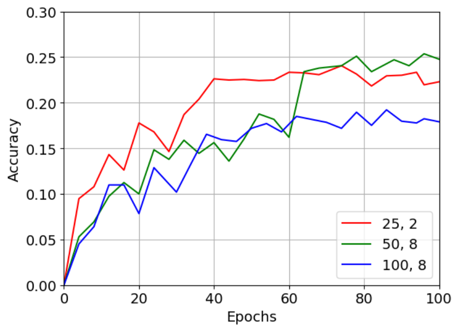


Figure 4.5: Comparison of weights used in the loss function $l = l_{classification} + w_1 \cdot l_{tracking} + w_2 \cdot l_{direction}$. Weights $w_1 = 25$ and $w_2 = 2$ result in the largest initial increase in test accuracy.

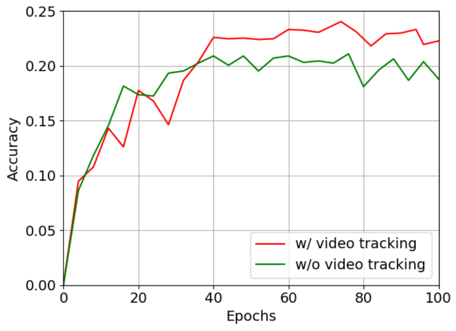


Figure 4.6: Study of the impact of removing the video tracking branch from the proposed model on test accuracy. Comparison is shown with model with all three branches in place.

architecture and the features from the backbone stage are not suitable for the task.

To understand whether this branch is critical for performance we remove all branches except the third branch. We now directly input the videos both forwards and backwards to the backbone. Removing the time tracking branch and the video classification branch reveals that the video direction branch still does not perform as the loss barely decreases. An experiment with shuffled frames delivered the same result. This indicates that the lack of performance is possibly due to videos of the dataset lacking clear temporal direction. One recent work has found that shuffling the frames in the dataset UCF-101 resulted in the same performance on their model (Zhou et al. 2018). The same experiment on the Something-Something dataset (Goyal et al. 2017) resulted in a significant difference. UCF-101 is a similar dataset to HMDB-51 in terms of the classes present, the source of the videos and the size. Thus, it seems likely that the reason for the video direction branch not performing well could be attributed to the dataset not including many temporal relations.

In addition, we removed only the video tracking branch to study the impact of the video tracking task on the model. A plot comparison is shown in figure 4.6. The test accuracy as a function of the number of epochs is shown. The test plot for the complete architecture with all three branches is included as well. The test plot suggest that the video tracking task contributes to performance visibly.

5 Conclusion

In this thesis, the benefit of training video classification from scratch is studied in order to reduce the need for costly and laborious data annotation. The model combines self-supervised learning methods with video classification. The multi-branched architecture is composed of one video tracking branch, one video time direction branch and one main branch that performs video classification.

Experimental results on HMDB-51 show that this approach improves performance from scratch significantly. We outperform results from related works that use self-supervised learning as pretraining for video classification. We also outperform work that uses additional supervised pretraining on UCF-101. The potential of combining video classification with self-supervised learning is demonstrated. Moreover, this method makes extensive pretraining unnecessary and only uses the annotated dataset at hand. The method therefore allows for even less total training time.

The time scope of the thesis being two months combined with the training time per experiment caused us to not perform extensive analysis. The main focus was integrating self-supervised methods in a model trained from scratch and showing the advantage over training from random initialization. The analysis of the model suggests that by using larger datasets these results can easily be surpassed. Usage of more resources could allow the results to gain a greater advantage over methods that use conventional pretraining. The proposed architecture so far shows favourable results and reduces the human labour needed for annotating large datasets.

There are numerous possible directions for future work. First of all, the self-supervised learning methods could be improved. The video tracking branch currently extracts patches for tracking randomly. By first finding the region of interest in a frame and then extracting this patch, the performance could benefit. Additionally, the video tracking method is applied in an identical manner on all videos. The method does not take into account whether an action takes place slowly or happens quickly. When the duration of actions in videos is taken into consideration, the frame gap in the video tracking branch can be adapted. Further research on suitable architectures for learning the direction of time in video would add to the possibilities for improvement of the video direction branch.

Next, the underlying architecture can more extensively be studied. In particular, a better backbone of the neural network can be found by applying Neural Architecture Search (Zoph et al. 2018). Applying this present-day technique reduces the time spent on manual design considerably. Further improvements on the architecture can be found in increasing the depth of the model and the training time. The impact of several other parameters on the performance of the model could be studied as well. One such parameter could be increasing the temporal size of input. Extra temporal information could either be provided by self-supervision tasks or by additionally utilizing the optical-flow information, such as implemented in two-stream models. Another possibility could perhaps be found in visualizing the filters in the model in order to gain insight in the learning of self-supervised tasks.

With only minor modifications the model could possibly be applied in other tasks. Examples could be untrimmed video classification, video activity localization and video temporal proposal. The emerging field of self-supervised learning has much potential to be combined with other video understanding tasks.

References

Abdulnabi, A. H., Wang, G., Lu, J., and Jia, K. (2015). “Multi-Task CNN Model for Attribute Prediction”. In: *CoRR* abs/1601.00400.

Amir-Moéz, A. R. and Davis, C. (1960). “Generalized Frobenius inner products”. In: *Mathematische Annalen* 141.

Breiman, L. (2001). “Random Forests”. In: *Machine Learning*. DOI: 10.1023/A:1010933404324.

Carreira, J. and Zisserman, A. (2017). “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2017.502.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004). “Visual categorization with bags of keypoints”. In: *European Conference on Computer Vision*.

Dalal, N. and Triggs, B. (2005). “Histograms of oriented gradients for human detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2005.177.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). “ImageNet: A large-scale hierarchical image database”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2009.5206848.

Diba, A., Fayyaz, M., Sharma, V., Karami, A. H., Arzani, M. M., Yousefzadeh, R., and Gool, L. van (2017). “Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification”. In: *CoRR* abs/1711.08200.

Doersch, C. and Zisserman, A. (2017). “Multi-Task Self-Supervised Visual Learning”. In: *International Conference on Computer Vision*. DOI: 10.1109/ICCV.2017.226.

Donahue, J., Hendricks, L. A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K., and Darrell, T. (2017). “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/tpami.2016.2599174.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research*.

Feichtenhofer, C., Pinz, A., and Zisserman, A. (2016). “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/cvpr.2016.213.

Fernando, B., Bilen, H., Gavves, E., and Gould, S. (2017). “Self-Supervised Video Representation Learning with Odd-One-Out Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2017.607.

Gan, C., Gong, B., Liu, K., Su, H., and Guibas, L. J. (2018). “Geometry Guided Convolutional Neural Networks for Self-Supervised Video Representation Learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2018.00586.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and Lecun, Y. (2015). “Unsupervised Learning of Spatiotemporally Coherent Metrics”. In: *International Conference on Computer Vision*. DOI: `10.1109/ICCV.2015.465`.

Goyal, R., Kahou, S. E., Michalski, V., Materzynska, J., Westphal, S., Kim, H., Haenel, V., Fruend, I., Yianilos, P., Mueller-Freitag, M., Hoppe, F., Thurau, C., Bax, I., and Memisevic, R. (2017). “The “Something Something” Video Database for Learning and Evaluating Visual Common Sense”. In: *International Conference on Computer Vision*. DOI: `10.1109/iccv.2017.622`.

Hadsell, R., Chopra, S., and LeCun, Y. (n.d.). “Dimensionality Reduction by Learning an Invariant Mapping”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/cvpr.2006.100`.

Hara, K., Kataoka, H., and Satoh, Y. (2018a). *3D ResNets for Action Recognition*. URL: `https://github.com/kenshohara/3D-ResNets-PyTorch` (visited on 05/08/2019).

— (2018b). “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/CVPR.2018.00685`.

Hasson, Y. (2017). *Inflated I3D models with ImageNet weight transfer in PyTorch*. URL: `https://github.com/hassony2/inflated_convnets_pytorch` (visited on 06/14/2019).

He, K., Girshick, R. B., and Dollár, P. (2018). “Rethinking ImageNet Pre-training”. In: *CoRR* abs/1811.08883.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/cvpr.2016.90`.

Hearst, M. A. (1998). “Support Vector Machines”. In: *IEEE Intelligent Systems*. DOI: `10.1109/5254.708428`.

Hochreiter, S. (1998). “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6. DOI: `10.1142/S0218488598000094`.

Hu, T., Wang, Y., and Yang, P. (2018). “Dense In Dense: Training Segmentation from Scratch”. In: *Asian Conference on Computer Vision*. DOI: `10.1007/978-3-030-20876-9_29`.

Huang, G., Liu, Z., Weinberger, K. Q., and Maaten, L. van der (2017). “Densely connected convolutional networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/CVPR.2017.243`.

Ioffe, S. and Szegedy, C. (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). “Large-Scale Video Classification with Convolutional Neural Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/cvpr.2014.223`.

Kim, D., Cho, D., and Kweon, I.-S. (2019). “Self-Supervised Video Representation Learning with Space-Time Cubic Puzzles”. In: *CoRR* abs/1811.09795.

Kingma, D. P. and Ba, J. (2015). “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. DOI: 10.1145/3065386.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. (2011). “HMDB: A large video database for human motion recognition”. In: *International Conference on Computer Vision*. DOI: 10.1109/iccv.2011.6126543.

Lakhal, M., Clapés, A., Escalera, S., Lanz, O., and Cavallaro, A. (2018). “Residual Stacked RNNs for Action Recognition”. In: *European Conference on Computer Vision*. DOI: 10.1007/978-3-030-11012-3_40.

Lee, H.-Y., Huang, J.-B., Singh, M., and Yang, M.-H. (2017). “Unsupervised Representation Learning by Sorting Sequences”. In: *International Conference on Computer Vision*. DOI: 10.1109/iccv.2017.79.

Lin, T.-Y., Maire, M. B., J., S., Bourdev, L. D., Girshick, R. B., Hays James Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). “Microsoft COCO: Common Objects in Context”. In: *European Conference on Computer Vision*. DOI: 10.1007/978-3-319-10602-1_48.

Lowe, D. G. (2004). “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision*. DOI: 10.1023/B:VISI.0000029664.99615.94.

Luvizon, D., Picard, D., and Tabia, H. (2018). “2D/3D Pose Estimation and Action Recognition Using Multitask Deep Learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2018.00539.

Misra, I., Zitnick, C. L., and Hebert, M. (2016). “Shuffle and Learn: Unsupervised Learning using Temporal Order Verification”. In: *European Conference on Computer Vision*. DOI: 10.1007/978-3-319-46448-0_32.

Mobahi, H., Collobert, R., and Weston, J. (2009). “Deep learning from temporal coherence in video”. In: *International Conference on Machine Learning*. DOI: 10.1145/1553374.1553469.

Nair, V. and Hinton, G. E. (2010). “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *International Conference on Machine Learning*.

Noroozi, M. and Favaro, P. (2016). “Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles”. In: *European Conference on Computer Vision*. DOI: 10.1007/978-3-319-46466-4_5.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). “Automatic differentiation in PyTorch”. In:

Pickup, L. C., Pan, Z., Wei, D., Shih, Y., Zhang, C., Zisserman, A., Scholkopf, B., and Freeman, W. T. (2014). “Seeing the Arrow of Time”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/cvpr.2014.262.

Ranjan, R., Patel, V. M., and Chellappa, R. (2017). “Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/tpami.2017.2781233.

Shen, Z., Liu, Z., Li, J., Jiang, Y.-G., Chen, Y., and Xue, X. (2017). “DSOD: Learning Deeply Supervised Object Detectors from Scratch”. In: *International Conference on Computer Vision*. DOI: 10.1109/ICCV.2017.212.

Simonyan, K. and Zisserman, A. (2014). “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: *Advances in Neural Information Processing Systems*.

Simonyan, K. and Zisserman, A. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556.

Soomro, K., Zamir, A. R., and Shah, M. (2012). "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". In: *CoRR* abs/1212.0402.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. R. (2015). "Unsupervised Learning of Video Representations Using LSTMs". In: *International Conference on Machine Learning*.

Szegedy, C., Liu, W., Jia Yangqing ad Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/CVPR.2015.7298594`.

Torrey, L. and Shavlik, J. (2010). "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). "Learning Spatiotemporal Features with 3D Convolutional Networks". In: *International Conference on Computer Vision*. DOI: `10.1109/iccv.2015.510`.

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). "A Closer Look at Spatiotemporal Convolutions for Action Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/CVPR.2018.00675`.

Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). "Generating videos with scene dynamics". In: *Advances in Neural Information Processing Systems*.

Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., and Gool, L. van (2018). "Temporal Segment Networks for Action Recognition in Videos". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: `10.1109/tpami.2018.2868668`.

Wang, X., Jabri, A., and Efros, A. A. (2019a). "Learning Correspondence from the Cycle-Consistency of Time". In: *CoRR* abs/1903.07593.

— (2019b). *Learning Correspondence from the Cycle-consistency of Time*. URL: `https://github.com/xiaolonw/TimeCycle` (visited on 05/18/2019).

Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. (2015). "Deep Image: Scaling up Image Recognition". In: *CoRR* abs/1501.02876.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). "Beyond Short Snippets: Deep Networks for Video Classification". In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/cvpr.2015.7299101`.

Zha, S., Luisier, F., Andrews, W., Srivastava, N., and Salakhutdinov, R. R. (2015). "Exploiting Image-trained CNN Architectures for Unconstrained Video Classification". In: *British Machine Vision Conference*. DOI: `10.5244/c.29.60`.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2014). "Facial Landmark Detection by Deep Multi-task Learning". In: *European Conference on Computer Vision*. DOI: `10.1007/978-3-319-10599-4_7`.

Zhi, H., Yu, H., Li, S., and Gao, C. (2017). "DMMLN: A deep multi-task and metric learning based network for video classification". In: *2017 IEEE Symposium Series on Computational Intelligence*. DOI: `10.1109/SSCI.2017.8285296`.

Zhou, B., Andonian, A., and Torralba, A. (2018). "Temporal Relational Reasoning in Videos". In: *European Conference on Computer Vision*. DOI: `10.1007/978-3-030-01246-5_49`.

Zhu, Y., Lan, Z., Newsam, S., and Hauptmann, A. G. (2017). “Hidden Two-Stream Convolutional Networks for Action Recognition”. In: *Asian Conference on Computer Vision*. DOI: 10.1007/978-3-030-20893-6_23.

Zisserman, A. (2018). *Self-Supervised Learning*. URL: https://project.inria.fr/paiss/files/2018/07/zisserman-self-supervised.pdf (visited on 07/10/2019).

Zoph, B., Vasudevan, V., Shlens, J., and V. Le, Q. (2018). “Learning Transferable Architectures for Scalable Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/cvpr.2018.00907.