TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

IIKKA TEIVAS

VIDEO EVENT CLASSIFICATION USING 3D CONVOLUTIONAL NEURAL NETWORKS

Master of Science thesis

Examiners: Prof. Joni Kämäräinen,
Univ. lect. Heikki Huttunen
Examiners and topic approved by the
Faculty of Computing and Electrical
Engineering
on 7th December 2016

ABSTRACT

IIKKA TEIVAS: Video event classification using 3D convolutional neural networks
Tampere University of Technology
Master of Science thesis, 49 pages
April 2017
Master's Degree Programme in Information Technology
Major: Data Engineering
Examiner: Prof. Joni Kämäräinen, Univ. lect. Heikki Huttunen
Keywords: Deep learning, convolutional neural networks, video classification

The objective of this thesis is to study the capabilities of 3D convolutional neural networks (CNN) in human action classification. By doing 3D convolutional operations through a stack of adjacent video frames, motion can be captured in the resulting features. These features are called spatio-temporal features, which take the spatial video frame data and motion into account, hence the name. For example, in a golf video the model recognizes the typical swinging motion in addition to the player itself.

Here, we present a 3D CNN model for action recognition and test it with four different 10 class datasets consisting of sports and daily human interactions. We compare this fairly new method with an image based 2D CNN method, where only spatial frame data is considered. In addition, we analyze and visualize the 3D CNN features and discuss the strengths and weaknesses.

The achieved video accuracies were lower than expected and we learned that neural networks architecture design and scaling is difficult and time consuming. Still, the relative results were comparable to previous studies. On average, 3D CNN performs around 10% better than 2D CNN and offers compact features. The resulting spatio-temporal features work well with strong motions like sports and struggle with separating human interactions that consist of small movement e.g. eating and speaking.

TIIVISTELMÄ

IIKKA TEIVAS: Videotapahtumien luokittelu konvoluutiohermoverkoilla
Tampereen teknillinen yliopisto
Diplomityö, 49 sivua
Huhtikuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Datatieteet
Tarkastajat: Prof. Joni Kämäräinen, Yliopistonlehtori Heikki Huttunen
Avainsanat: Syväoppiminen, konvoluutiohermoverkot, videoluokittelu

Semanttinen videotapahtumien tunnistus on tällä hetkellä yksi tutkituimmista konenäköön liittyvistä haasteista. Videotapahtuman tunnistamisella tarkoitetaan videossa tapahtuvan liikkeen ymmärtämistä ja sen perusteella luokittelemista.

Tämän työn tavoitteena on tutkia videotapahtumien tunnistusta 3D konvoluutiohermoverkoilla. Metodi perustuu automaattisesti tuotettuihin piirrevektoreihin, joihin kaapataan sekä spatiaali- että aikatietoa lyhyestä videosta. 3D-laskuoperaatiossa liike saadaan kaapattua konvoloimalla pieniä kuva-alueita yhtä aikaa peräkkäisille videon kuville, jolloin kaikki kuvat vaikuttavat operaation lopputulokseen.

Työssä esittelemme 3D konvoluutiohermoverkkomallin, jolla pyrimme luokittelemaan erilaisia tapahtumia toisistaan. Verkon opettamiseen ja testaamiseen käytämme neljää kymmenen luokan kokoelmaa, jotka sisältävät erilaisia urheilulajeja ja jokapäiväisiä ihmistoimintoja kuten syöminen tai hampaiden pesu. Tuloksia vertaamme vastaavaan hermoverkkoon, joka ei käytä hyväkseen aikaa, vaan perustaa piirrevektorinsa vain tilatason tietoon. Lisäksi visualisoimme ja analysoimme tuotettuja piirteitä ja pohdimme niiden vahvuuksia sekä heikkouksia.

Saavutetut tulokset eivät vastanneet täysin odotuksia ja opimme, että syvän hermoverkkoarkkitehtuurin suunnittelu on monimutkaista ja aikaa vievää. Ongelmista huolimatta suhteelliset tulokset vastaavat edellisten tutkimusten tuloksia. Kolmiulotteisilla konvoluutio-operaatioilla tuotettu liikedata parantaa videoluokittelun tuloksia noin 10 prosenttia verrattuna pelkkään kuvadataan. 3D piirteet toimivat testiympäristössä hyvin selkeille liikkeille kuten urheilulajeille, mutta epäonnistuu pienillä kuva-alueilla tapahtuvien liikkeiden kuten syömisen ja puhumisen erottelussa.

PREFACE

I wish to thank my supervisors Joni Kämäräinen and Heikki Huttunen for all the help during this work. In addition, I'd like to thank Antti Hahto for the useful tips and tricks.

Tampere, 04.04.2017

Iikka Teivas

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ANN	Artificial neural networks
BN	Batch normalization
BoW	Bag of visual words
CE	Cross-Entropy
CNN	Convolutional neural networks
CRCV	Center for Research in Computer Vision
CUDA	Compute Unified Device Architecture
DNN	Deep neural networks
FC	Fully connected layer
FPS	Frames per second
GPU	Graphics Processing Unit
HOF	Histograms of optical flow
HOG	Histograms of oriented gradients
iDT	Improved dense trajectories
LSTM	Long short-term memory
MBH	Motion boundary histogram
MIFS	Multi-skip feature stacking
MLP	Multi layer perceptron
RANSAC	Random sample consensus
ReLU	Rectified linear units
RNN	Recurrent neural networks
SFV	Stacked Fisher vector
SGD	Stochastic gradient descent
STIP	Space time intrest points
SURF	Speeded up robust features
SVM	Support vector machine
TDD	Trajectory-pooled deep convolutional descriptor
t-SNE	t-Distributed Stochastic Neighbor Embedding

1. INTRODUCTION

Convolutional neural networks (CNN) have been studied and proven as an effective tool for image content analysis and classification [7]. Currently CNN provide state-of-the-art results for image recognition, detection and segmentation. Inspired by the good results in images, the same tools are being studied for video recognition.

Semantic video based event recognition is currently one of the interesting challenges in machine learning and computer vision research fields. Semantic event recognition means, that a series of human motions in a video is classified as a type of action, e.g. playing tennis. Especially, human action recognition has become an important research area due to various application possibilities like video surveillance, customer behavior analysis and clinical seizure detection. Action recognition aims to detect certain pre-trained events or activities from a video stream and classify it accordingly.

Video analysis is a difficult area due to the fact that it is actually a mix of multiple different problems. In videos there are numerous variations already inside one class such as motion, backround clutter and camera position changes. Also, the action recognition itself depends on multiple factors such as the human pose and objects being interacted with. In addition, there are multiple human actions that may look similar even for a human, e.g. putting on lipstick vs. eating and diving vs. cliff jumping. In many situations, the difference between two totally different actions is the scene where the action is performed.

Commonly above types of machine learning problems are solved by handcrafting a set of features or a bag of visual words that describe the type of movement best. These features are then used to train a classifier. The issue behind this type of methodology is the huge amount of work needed to create a good feature set and these typically rely on certain assuptions about the circumstances of the video scenario. In a real world applications, relying on assumptions that narrow the problem space is not preferable or tempting.

Deep neural networks (DNN) have been showing promising results also on video classification problems. In essence, idea behind deep neural networks is to automatically find the parameters or features for the given training data in order to classify it accordingly to the known type. It has been proven, that CNNs can provide general and robust models for image or video recognition problems with minimal manual work and can be easily extended to different kinds of scenarios.[30]

In this thesis, the focus is on CNNs and more precisely 3-dimensional convolutional neural networks. The usage of 3D convolutions allows to capture spatial information in time from the video data stream by taking consecutive video frames into account. The purpose is to find out how well a 3D CNN can perform in comparison to more classical manual techniques and to truly understand the way it works.

The rest of the paper is organised as follows. Theory section covers related work and basic theory behind the neural networks. Methods section explains how the practical work has been done and what were the building blocks. Evaluation section consists of the results and visualization helping to understand them. In addition the decisions towards the final model design is explained and analyzed. Finally, we draw some conclusions about the task at hand and discuss about the possible future development.

2. VIDEO EVENT CLASSIFICATION

2.1 Background

Video event classification is a supervised learning problem meaning, that the system is trained with labelled data and after training it tries to assign input to some of the pre-defined classes. In this context, labelling data means assigning a data sample with an unambiguous class label that acts as a ground truth. The two phases of supervised learning, training and deployment, are visualized in Figure 2.1.

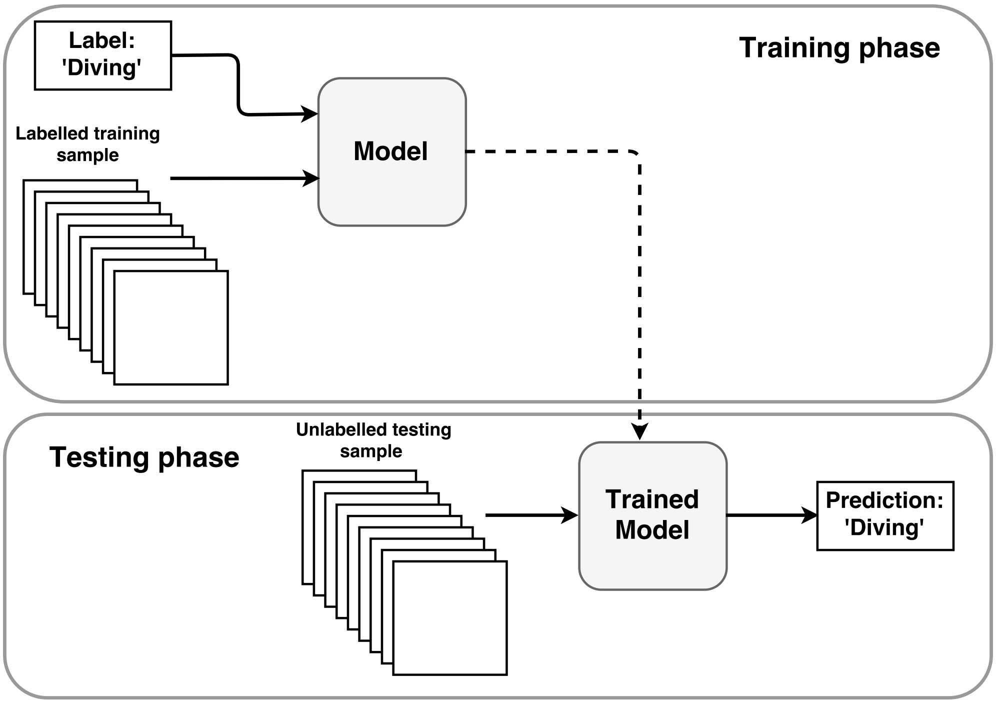


Figure 2.1 *Supervised learning flow. Upper left graph represents the training phase and the lower right testing phase. This example mimics a video classification problem, so here one training and testing sample is a stack of adjacent video frames.*

In contrast to image classification, there are more aspects to consider when using video data. The semantic understanding becomes more complex for example when the camera moves and the backround changes in time. The task is computationally very demanding due to the fact that even a short video has hundreds of frames. If a normal video has 30 frames per second (FPS) and the movement is slow or subtle, most of the frames look exactly the same and do not offer much additional information compared to adjacent ones (see Figure 2.2). This is one of the reasons why it is extremely unefficient to analyse video data according to raw data. Due to this fact, feature extraction becomes more and more important so the complex context information in a video clip can be represented without using all of the data thus making the processing faster and easier.

Figure* *2.2 *Four consecutive video frames of a man jumping into a pool.*

2.2 Datasets

Availability of labelled large-scale video datasets has made video analysis easier to research. At the moment, the video analysis field is higly focused on human actions as can be seen from the dataset content. Here are some of the major benchmark datasets currently used in video event analysis.

UCF-101: One of the most popular large scale video dataset which has short, around 10 second video clips from 101 different human actions. The dataset is provided by Center for Research in Computer Vision (CRCV) and it has been one of the benchmark datasets in human action recognition field. The videos include a broad range of different sports and activities. The major action type categories are sports, playing musical instrument, human-object interaction, body-motion only and human-human interaction. Example classes can be seen in the Figure 2.3

The videos have been gathered from Youtube and every class has around 100 video clips. Videos are 30 fps and the original resolution is 320x240. The clips in one class are split into groups according to the original video they are extracted from.

The dataset offers training and test splits so that the groups are not mixed and distortion of evaluation results is prevented. [27]

***Figure* 2.3** *Sample video classes of UCF-101. [27]*

HMDB-51: A large video database for Human motion recognition consisting of 51 different movement types and a total of 7000 video clips. The set is provided by Brown University Research Group. The dataset is similar to UCF-101 when it comes to the classes. It consists of body movement types like walking, pushups and waving. In addition, the dataset has more subtle facial expression types like chew, eat, laugh and smile, making it harder to classify with current technology as can be seen from the state-of-the-art results in the next section. The videos are also separated into different training and testing splits, similarly to UCF-101. The video resolution is the same as in UCF-101 in major of the video clips. [16]

Hollywood 2: A dataset of human action videos collected from 69 different Hollywood videos. The dataset consists of 12 human action classes and 10 scene classes from a total of 3669 video clips. The action classes include, for example, answering phone, eating and kissing, while the scene classes consists of the types like inside a house, inside an office and on a road.[19]

Sports-1M: One of the largest video dataset to date with total of 1,133,158 videos belonging to 487 classes. The videos are automatically collected from Youtube by an algorithm and thus are weakly labelled. The dataset was gathered and introduced by Google Research and Computer Science Department in Stanford University [14].

2.3 Related work

Convolutional neural networks have been studied for a couple of decades. With the current state of the art image classification models, images can be classified having good confidence and low classificaton error [7]. A natural continuation is to move from images to videos, which are essentially stacks of images. In image recognition, one tries to classify objects in images, which is a purely static situation compared to videos where the object moves in time or in temporal space.

Although the results are promising, they are far from perfect or even satisfactory for the more trickier datasets like HMDB-51 and Hollywood 2, consisting of fine movement like eating, smiling and smoking a cigarette. For UCF-101 and Sports 1M the best results are already decent with close to 90 % accuracy [30, 34]. Different sports are easier examples consisting of more unambiguous and large movement, meaning more pixel differences between adjacent frames and more distinctive separability.

2.3.1 Traditional features

Video recognition is based on detecting certain type of movement in a video. This is done using spatial features from adjacent frames and creating an understanding of the movement by tracking them in time. The combination of these features are called spatio-temporal features and many different methods have been used for collecting those. Most of them are based on 3D convolutions or 2D convolutional networks mixed with recurrent layers as a memory. One popular method of collecting the temporal data is optical flow representation [32, 33]. Optical flow is a pattern of apparent motion happening in a video. In short, it is calculated from the displacement of pixels in two adjacent frames of a video [3]. Optical flow offers dence trajectories and the basis for bag of words type of feature sets.

Bag of (visual) Words (BoW) presentation and local features have been popular in the field of action recognition; the reason can be seen from good results in THUMOS 2013 Action Recognition Challenge [12]. BoW framework pipeline consists of five steps: feature extraction, feature pre-processing, codebook generation, feature encoding, pooling and normalization. The last step is to feed the global representation into a classifier like linear SVM.

H. Wang et al.[32] have current state of the art results in the action recognition field. (2011) They have started with more classical machine learning approach utilizing local BoW features and nonlinear SVM. Their idea was to calculate dense point trajectories from every frame and collect a best possible set of features. Before this study dense point trajectories have not been used for action recognition. The motion trajectories were calculated by tracking dense sampled points using Farnebäcks optical flow algorithm and the bag-of-features consisting histograms of optical flow (HOF), histograms of oriented gradients (HOG), motion boundary histogram (MBH) and the trajectiories as such. This method yielded 58.3 % accuracy on Hollywood 2 action dataset.

Two years later in 2013 H. Wang and Schmid[33] further improved the method and results by removing distracting motion trajectories caused by camera movement. In human action videos the camera is usually focused on the human performing the action moving in a similar fashion. Their method was to estimate and then remove camera motion trajectories by matching features between adjacent frames using algorithms like random sample consensus (RANSAC) and speeded up robust features (SURF). In order to find the matching backround trajectories they also used HOG features to detect the action performing human in every frame. These local foreground trajectories were then used to create similar BoW features as mentioned previously. The technique is now commonly known as imporved dense trajectories (iDT). In addition to BoW another feature encoding method called Fisher vector was utilized. Contrary to BoW, Fisher vector considers the occurences of the features and additionally encodes information about the feature distribution. The improved method with Fisher vector encoding achieved 91.2 % accuracy on the UCF50 and 64.3 % on the Hollywood 2 dataset and 57.2 % on HMDB-51 dataset.[3]

2014 Peng et al.[22] improved the dense trajectory method by extending the BoW representation using larger set of descriptors, fusing different coding methods and exploring different pooling and normalization methods. Descriptors are based on two popular local features namely Space time intrest points (STIP) and iDT. The studied coding methods belong to three categories which are voting based, reconstruction based and super vector based encoding. The extensive BoW model study results were then used to create a hybrid representation by combining multiple BoW model outputs. The hybrid model reached 61.1 % accuracy for HMDB5 and 87.9 % for UCF101.

Later Peng et al.[23] proposed Stacked Fisher Vector (SFV) representation for their BoW representation. SFV is a multi-layer nested Fisher vector encoding method, that hugely improved the previous results on HMDB-51 achieving accuracy of 66.8 % which is currently the best published result.

2015 Lan et al.[17] introduced Multi-skIp Feature Stacking (MIFS) technique which was designed to remove the problem of coarse scale feature generation while using Gaussian Pyramids for scale-invariant feature enhancing. The way of recapturing information allows the system to match human motion at different speeds and ranges. For evaluation purposes the rest of the implementation is identical to [33]. MIFS improved the state-of-the-art results for Hollywood 2 with the accuracy of 68.0 % and UCF101 with the accuracy of 89.1 % and almost achieved the same results as Peng et al. on HMDB-51 with accuracy of 65.1 %.

2.3.2 Deep features

Good results on action recognition have been achieved also with deep neural networks. In 2014 Karpathy et al.[14] trained a deep 2D convolutional neural network with 1 million sports videos from Youtube. This was the main study to introduce the Sports 1M. The used CNN architecture is based on 2012 ImageNet challenge winning model AlexNet [15]. Since 2D convolutions give only spatial information Karpathy introduced three fusion models to utilize temporal information in the network. Early fusion means doing 3D convolutions in the first convolutional layer by extending the filters by one dimension. Late fusion model combines two single frame models taking the same input multiple frames apart and then fusing the outputs in the last layer, thus getting local movement information. The third model, called slow fusion, is a way of getting more global movement descriptors by carrying out the temporal data through the whole network in order to compute activations. This means extending the dimensions of kernels in every layer similarly to the 3D convolution model presented by Ji et al.[11] in 2013.

The results presented by Karpathy et al.[14] showed that CNNs gives strong, robust and performant features for weakly labelled data compared to hand crafted Bag-of-Words methods. The trained model was also used to test other similar datasets like UCF-101 and the so called transfer learning experiment showed promising results even though the dataset includes other activities than just sports. The 3-fold accuracy for UCF-101 was 65.4% after using just 50 samples per every class to further train the model.

Intrigued by the previous findings, Simonayan and Zisserman[25] trained a two stream deep 2D CNN for action recognition and presented a new way of creating spatiotemporal features with CNNs. In addition to single frames the other stream takes multi-frame optical flow output as an input. This local motion information is then fused with the spatial information from the first stream by averaging and using pre-trained linear SVM. The classifier uses softmax layer output as features. Using both streams and SVM fusion, authors achieved an accuracy of 88.0 % on UCF-101 and 57.9 % on HMDB-51. Compared to state of the art BoW methods, UCF-101 result is marginally better (+0.1 %) but with HMDB51 the result is significantly worse (-7.4 %).

Tran et al.[30] extended the study of the capabilities of 3D CNN as a spatiotemporal feature extractor for action recognition. Main findings were that 3D CNNs are more suitable for spatio temporal feature extraction than 2D CNNs and that 3D CNN features with simple linear classifier is outperforming other state of the art methods. Authors mention, that 3D CNN has all the properties of a good descriptor meaning it is generic, compact, simple and high-performance. The author emphasizes the simplicity compared to other current technologies.

Compared to the model Karpathy et al.[14] used, the presented model in [30] consists of 5 convolution and pooling layers and two fully connected layers. Alone with the 3D CNN features Tran et al.[30] achieved great results on UCF-101 dataset with the accuracy of 85.2 %. Author then adds, that combining different descriptors is beneficial in order to achieve state of the art results. Results show, that by combining 3D CNN features with iDT and classifying those with linear SVM the model is capable of classifying UCF-101 dataset with 90.1 % accuracy. The reason for this is that these two descriptors are highly complementary to each other. iDT is based on low-level gradient histograms and 3D CNN represents higher level abtract information, so the benefit of combining these two methods is clear.

Tran et al.[30] compared 3D CNNs with other spatiotemporal neural network solutions like Long short-term memory network (LSTM), which is a one implementation of RNN. The results show, that 3D CNN outperforms LSTM clearly with 10-20 % difference using human action datasets.

If the 3D CNN capabilities are compared against image based 2D CNN architectures the results show, that the addition of one dimension makes a difference. Single net 3D CNN presented by Tran et al.[30] achieved 82.3 % accuracy on UCF-101 while H. Wang et al.[33] achieved 73 % accuracy on the same dataset using only spatial information and 2D CNNs.

Similar results have been achieved using multi-frame optical flow fields as inputs for 2D CNNs. This is a popular method to extract temporal data from video stream and was used by Simonyan and Zisserman[25] in 2014. The results show, that optical flow method gives similar results compared to 3D CNNs with pure video frames.

In 2015 L. Wang et al.[34] built a new kind of video representation called trajectory-pooled deep convolutional descriptor (TDD). TDD utilizes both hand crafted features and deep learned ones in order to make more robust descriptors. The handcrafed features are based on local descriptors like HOF, HOG and MBH and the deep CNN architecture is built from two nets considering spatial data and temporal optical flow data. The two stream CNN alone gives 84.7 % accuracy on UCF-101 and the final TDD net gives 90.3 %. If iDT features are mixed with the TDD, the result increases to 91.5 % which is one of the highest scores for this data set. In addition the HMDB-51 data set results were exceptional with the accuracy of 65.9 % using TDD and iDT.

2.3.3 Method summary

The best feature extraction methods can be clearly categorized into three: handcrafted BoW based solutions, purely CNN based solutions and a mixture of these two. The earlier mentioned results are shown in Table 2.1. Currently the old fashioned way of hand-crafting a set of optimal features is better according to the results but it's clear, that CNNs and especially 3D convolutions are promising feature extractors for action recognition. Although CNNs alone are marginally worse for the use case, they are performant, robust, easy to implement and require little work compared to BoW methods. This makes them an interesting study subject for this Master's thesis.

Author	HMDB51	UCF101	Hollywood 2	UCF50
H. Wang and Schmid [33]	57.2		64.3	91.2
Peng et al.[23]	**66.8**	87.9		92.3
Lan et al. [17]	65.1	89.1	**68.0**	**94.4**
Simonayan and Zisserman [25]	59.4	88.0		
L. Wang et al. [34]	65.9	**91.5**		
Tran et al. [30]		90.4		

Table 2.1 *Summary of the current state-of-the-art results. Best score for every dataset is highlighted.*

3. NEURAL NETWORKS

In this chapter, the concept of neural networks is briefly explained. The focus is on a few models which are used in the implementation part of this thesis.

3.1 Feed forward neural networks

Artificial neural network (ANN) is a network of simple computing units called neurons or perceptrons. In general terms, ANN is an approximator that learns from a large amount of labelled input data. The term neural network and the structure are inspired by the central nervous system in the brain of animals. A neural network consists of layers of neurons. A layer is a set of parallel neurons that are not connected to each other and which output together a set of weights for one input. These layers are stacked a number of times which defines the depth of the network.

A simple fully connected two layer networks structure can be seen in the Figure 3.1. The input layer is not considered as a layer as such, as it only takes the input data and passes it to the next hidden layer without altering. Thus it is not counted towards the total layer count when comparing network architectures. [1]

The simplest ANN model is a feedforward *multi layer perceptron* (MLP) network, which means that every neuron is connected to each of the previous and the following layer neurons. This form is also known as *fully connected* network. The name feedforward comes from network functionality where the data flows through the system from input x to output y without any feedback loops. If the models are extended with feedback loops, they are called recurrent neural networks (RNN). The focus of this thesis is on the former type. [1]

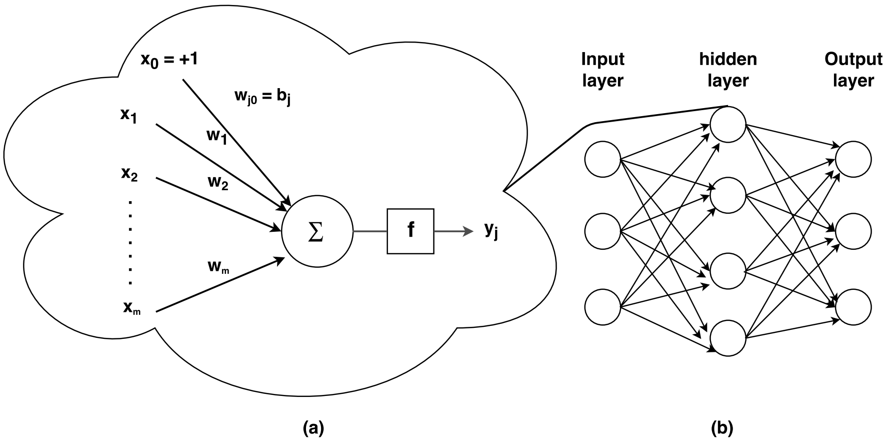


Figure 3.1 *Typical ANN structure: (a) A visualization of one neuron (b) Two layer neural network with one hidden layer and output layer*

3.1.1 Perceptron

A perceptron, also known as neuron, (Figure 3.1) is a function that takes a vector $\mathbf{x}$ $\{x_0, ..., x_m\}$ as an input and computes a dot product with the weight vector $\mathbf{w_j}$ $\{w_{j0}, ..., w_{jm}\}$ and outputs a scalar $v_j(n)$ as:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)x_i(n)) = \mathbf{w}_j^T\mathbf{x}, \tag{3.1}$$

where j is the neuron index, n is the current iteration and m is the number of inputs. The weight w_{j0} equals the bias b of the neuron j. After the activation function $f_j : \Re \rightarrow \Re$ we get the neuron output $y_j(n) = f_j(v_j(n))$.

The activation function produces a nonlinear decision boundary for the weighted input and transforms the score to probability. The function is commonly implemented as sigmoidal nonlinearity and one form is *Logistic sigmoid function* and can be generally defined with the function:

$$f_{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

Another activation function is called *Hyberbolic tangent function* and is defined as:

$$f_{tanh}(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{3.3}$$

which is, in essence, a rescaled and biased version of the first form. Currently, the most popular and recommended activation function for deep neural networks is a *rectifier* and it is defined as follows:

$$f_{ReLU}(x) = max(0, x). \tag{3.4}$$

The units with rectifier activations are then called *Rectified Linear Units* or ReLUs. The reason for popularity is because ReLUs make deep neural network training multiple times faster due to the simple operations needed to calculate it. At the same time, it is the most different of the three. In contrast to sigmoidal nonlinearities, a rectifier is not a saturating function. ReLUs non-saturating form has been proven to accelerate gradient based optimization functions like gradient descent (explained later in Section 3.1.2.[15, 1]

For output layer activations, the situation is different compared to hidden layers. For binary classification the preferred activation funtion for output layer neurons is the logistic sigmoid which can be referred as *logit* model. For multiclass classification problems the model can be implemented as *multinomial logit* model which uses the *softmax* nonlinearity which is defined component wise for vector $\mathbf{x}$ as:

$$f_{softmax}(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \tag{3.5}$$

where the K is the total number of different classes. Softmax regression is a multi-class generalization of the logistic regression model. It outputs one-hot coded class probablities and it is preferred if the classes in question are mutually exclusive. [6][4]

3.1.2 Network training

In essence, the neuron training process is an optimization problem. The goal is to minimize the error $e_j(n)$ between the desired output d_j and the prediction $y_j(n)$

according to the loss function or objective function $\mathcal{L}$. For example the loss $\mathcal{L}(n)$ using squared loss is defined as:

$$\mathcal{L}(n) = \frac{1}{2}\sum_{j \in C}(e_j(n))^2 = \frac{1}{2}\sum_{j \in C}(d_j(n) - y_j(n))^2, \tag{3.6}$$

where C is the set of output neurons.[6]

In order to minimize the loss function, one needs to find optimal weights $\mathbf{w}_j$ with an optimization function. For feedforward networks this is commonly done by using a gradient based solutions, which iteratively update the weights towards the negative gradient of the non-convex loss function in order to reach its global minimum. One of the most popular gradient based learning techniques is called stochastic gradient descent (SGD) and it is defined as:

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n), \tag{3.7}$$

In order to update weights for the next iteration $n+1$ with the algorithm, loss output $\mathcal{L}(n)$ is required for the gradient calculation. To efficiently do this, a technique called backpropagation or backward pass is used. First in the forward pass phase the input data is carried through the whole network to produce an output with the current weights. In backpropagation the error between calculated output and the known target is passed to the optimization function in previous layers so the weights can be updated accordingly using, for example, the above mentioned SGD function. The implementation of the backward pass is essentially based on chain-rule and in this example can be calculated as:

$$\frac{\partial \mathcal{L}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{L}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}, \tag{3.8}$$

where j is an output node.[6]

According to the delta rule, the weight correction procedure can be defined as:

$$\Delta w_{ij}(n) = -\eta \frac{\partial \mathcal{L}(n)}{\partial w_{ij}(n)}, \tag{3.9}$$

where $\eta > 0$ is the step size or learning rate and $\frac{\partial \mathcal{L}(n)}{\partial w_{ij}(n)}$ is the gradient. The minus sign here means that the function seeks the direction for weights that reduces the value of $\mathcal{L}(n)$, hence the name gradient *descent.*[4, 6]

Hidden layer nodes don't have explicit desired output so the backpropagation algorithm have to be extended for recursive operations through the whole network. The hidden layer neuron error signal is calculated according to the error signals from all of the neurons the hidden layer neuron is connected to.

In order to extend the backpropagation to hidden layers we must define a *local gradients* $\delta_j(n)$ for output neuron j and $\delta_k(n)$ for hidden neuron k. The local gradient for output neuron j is defined as:

$$\delta_j(n) = -\frac{\partial \mathcal{L}(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{L}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}, \tag{3.10}$$

which defines the product of the error signal $e_j(n)$ and derivate of the activation output $y_j(n) = f_j(v_j(n))$ i.e. the required weight changes.

The hidden neuron local gradient $\delta_k(n)$ is then defined as:

$$\delta_k(n) = -\frac{\partial \mathcal{L}(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)}, \tag{3.11}$$

where the loss $\mathcal{L}(n)$ is calculated using the error signals from the output nodes defined in equation 3.6. Thus the partial derivate $\frac{\partial \mathcal{L}(n)}{\partial y_k(n)}$ can be calculated as:

$$\frac{\partial \mathcal{L}(n)}{\partial y_k(n)} = \sum_j e_j \frac{\partial e_j(n)}{\partial y_k(n)}. \tag{3.12}$$

The backpropagation weight correction formula can be then summarized for any neuron j as follows:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n), \tag{3.13}$$

where $\delta_j(n)$ is the local gradient of j and $y_i(n)$ is the input signal of neuron j.[6]

3.1.3 Regularization

Due to the huge amount of weights in a deep ANN, overfitting becomes an unavoidable issue. Overfitting means, that the network learns the training data so explicitly that it becomes biased and cannot generalize enough for unseen data. This is one of the core problems in deep learning and there are several ways to reduce this. *Regularization* is one methodology to avoid this.

One of the regularization techniques is called *dropout* which was developed for improving the generalization of large ANNs by minimizing complex co-adaptation. Dropout adds a fixed amount of noise to the hidden network layer activations meaning that with some given probability the neuron activation is dropped off. For example, if the dropout ratio is 0.4 then with a 40 % chance a hidden unit is dropped from the calculations towards the output. Dropout can be seen as a way of combining different subnetworks to create a one larger one. This adds noise to the learning process which reduces overfitting and increases robustness, but also makes the learning process slower. [2] [8]

Training a deep ANN typically requires multiple iterations and if some regularization methods like dropout is used, the process extends even more in time. This happens mostly due to changing parameters of the previous layer, which is again caused by changing distribution of input data between iterations. Due to this fact the training becomes difficult with saturating activation functions like *tanh*. The issue is called *internal covariate shift* and it can be reduced via correct parameter initialization or normalization of the layer input. [10]

One of the most popular methods for input normalization is called *batch normalization* (BN) and its main purpose is to reduce the *gradient exploding* and *vanishing* issue in deep ANN. Vanishing gradient is a phenomenon that is caused by saturating activation functions in deep network architectures. Saturating activation functions bound the gradients usually between (-1,1) or [0,1) and when these values are used in the backpropagation chain rule, the error signal decreases exponentially according to the network depth and the gradient may become nearly zero. Opposite reaction to vanishing is a gradient explosion, where activation derivates explode close to infinity due to large values.[21]

The idea behind BN is to normalize every layer input in mini-batches contrary to global dataset normalization, hence the name. Furthermore, BN works as a regularizer which in some cases makes dropout unecessary [5]. According to recent systematic parameter studies BN is recommended to use with ReLU non-linearity [20].

3.2 Convolutional neural network (CNN)

In this section, we will introduce the relevant concepts of convolutional neural networks.

3.2.1 2D convolutional neural networks

Convolutional neural network (CNN) is essentially very similar to fully connected MLP. Both are networks of neurons that have weights and biases and the core functionality is a dot product between input and neuron weights. The main difference is that CNN layer has a 3D network structure and the neurons are connected locally to a small region of input data in a grid contrary to fully connected MLP structure [18].

$$y_{xy} = f(\sum_i \sum_j w_{ij} v_{(x+i)(y+j)} + b), \tag{3.14}$$

where y_{xy} is a feature map value at (x,y), w_{ij} is a kernel weight and $v_{(x+i)(y+j)}$ is an input value at (x+i,y+j).[9]

Scalability is an issue restricting image processing with fully connected networks [13]. One fully connected neuron would have as much weights as the number of pixel values in the input image, times the number of color channels. For example if the input image has a size of 200x200x3 (width, height and depth respectively) the amount of weights in one neuron would be 120 000. This means that when constructing a full size MLP network with multiple neurons, the number of weights quickly exceeds the limit where overfitting starts to occur. In addition this requires a relatively huge amount of memory so MLP is not suitable for the use case.

CNN architecture typically consists of convolutional layers, pooling layers and fully connected layers, see Figure 3.2. The core functionality of CNN happens in convolutional layers, hence the name. As mentioned earlier, the neurons in convolutional layer are not fully but locally connected to a small region of the input data at a time. The region or area that that one neuron connects to is called a *receptive field* referring to ganglio cell functionality in eye's retina [13].

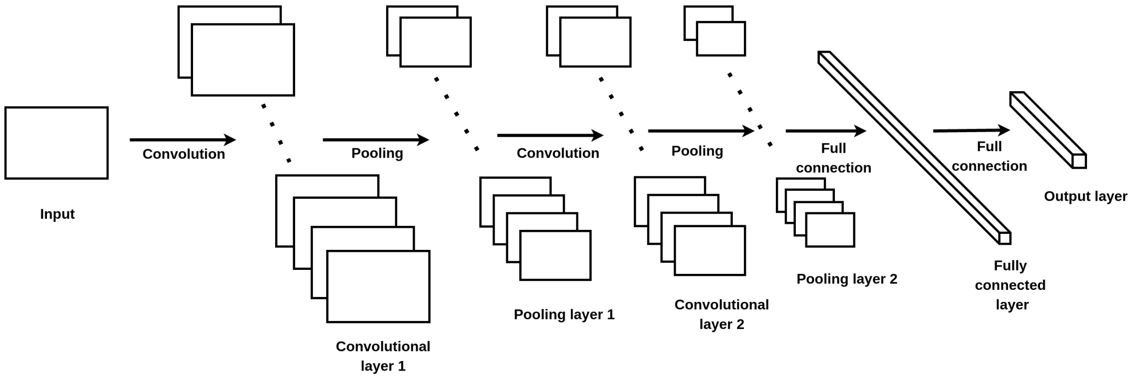


Figure* *3.2 *2D CNN structure. A typical 2D convolutional neural networks architecture with several convolutional layers followed by pooling and at the end one fully connected layer followed by softmax output layer.*

In one layer, each neuron calculates a dot product between its weights and the small input region. The number of neurons is defined such that the whole area of the image is covered by the receptive fields of every neuron by sliding the layer weights. The resulting convolution output is then spatially the same size as the input image (200x200x1). It is important that the receptive field depth is the same as the number of input color channels (in this case 3) in order to get all the color information from the image.

The resulting convolution output is called a *feature map* and the number of these maps is defined by the layer depth K [18]. A set of above mentioned convolution processes are computed in a stack which makes the network structure three dimensional. The layer output is then a stack of these feature maps and K is defined accordingly to the use case.

One feature map is a collection of neuron outputs and each neuron for every K has the same amount of weights. To limit this amount, convolutonal layer uses a method called parameter sharing, which means that every neuron in each K network layers share the same parametes. Each K feature map then needs just one set of weights which reduces the memory requirements significantly. The set of these weights are also often called filters or *convolution kernels*. Due to the shared weights, the same filter is applied or, more precisely, convolved across the whole spatial space of the input followed by some non-linearity. The result is a two dimensional response map of the applied filter at every position of the input. The relatively small number of weights is one of the main reasons why CNNs are good at creating general features.

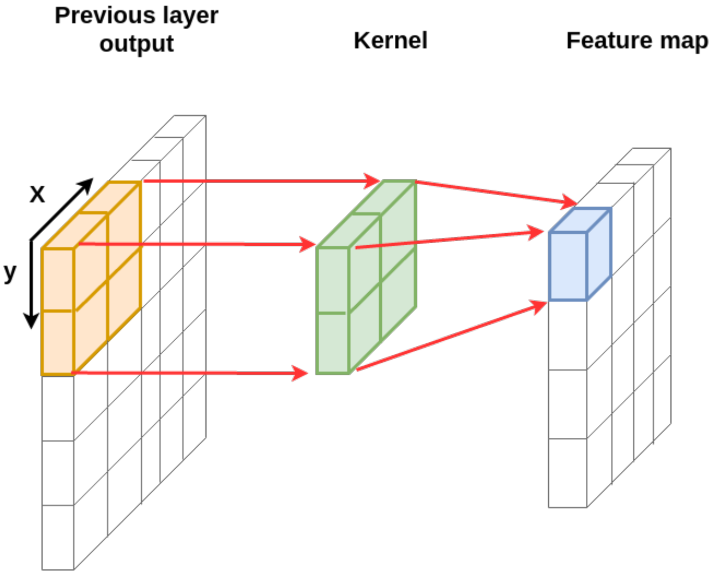


Figure 3.3 *2D convolution visualization. A previous layer feature map (left), a 2x2x1 kernel (middle) and the output (right). X and Y are the spatial dimensions of the layer output (image). In this example zero padding P is 0 and stride S is 1 so the feature map dimensions change in the process.*

The final amount of neurons on one layer is controlled by three hyperparameters which are the earlier mentioned depth K, stride S and zero-padding P. Stride defines how much the neuron receptive fields are overlapping. For example, if S=1 the filter is moved 1 pixel at a time taking into account majority of the field information multiple times. [13] Depending on the situation it is somethimes beneficial to pad zeros around the input volume. By extending the dimensions by adding zeros to the borders one can control the amount of dot product operations per input thus adjusting the amount of neurons in that layer. The effect of not using zero padding can be seen in Figure 3.3. Here, if the 2x2x1 filter is used to calculate the output in every part of the input layer feature map, the final feature map is one pixel smaller in both dimensions. The reduction of output size is enhanced when a larger stride is used.

The amount of neurons needed for a layer can also be calculated. This can be done with the formula:

$$(W - F + 2P)/S + 1, \tag{3.15}$$

where W is the input volume axis and F is the kernel size[13]. Lets assume an input image with the size of 200x200x3, a kernel size of 10x10x3 and feature map number K=20. Stride S=5 and padding P=0. The formula is then (200-10+2x0)/5+1 = 39. In this case, the size of the convolutional layer would be 39x39x20=30420 neurons and the total amount of unique weights 20x10x10x3+20=6020.

Convolution is followed by activation function, applying a non-linearity to the feature maps like in the fully connected MLP. The resulting output is then an activation for every feature map from the previous convolutional layer. Typically after convolution operation comes *pooling*. Pooling layer is essentially downsampling in order to control the size of the network and at the same time limiting the computational requirements and overfitting. Most importantly, pooling makes the feature map representation invariant to small positional changes of the input [1]. The pooling filter size defines how much downsampling is to be done. A common filter size is 2x2 which then downsamples the image to a fourth of its original size. Commonly the operation is maxpooling which takes a maximum value inside the pooling filter.

Although almost every popular CNN architecture includes pooling layers the true need for them has been disputed. A deep CNN can be implemented entirely without pooling if the convolution kernel strides are increased in some layers. This has been proven to work as well as state-of-the-art networks with pooling. [28]

3.2.2 State-of-the-art in image classification

CNNs have been studied for about 10 years and there are several implementations that have become standards in the field. AlexNet, developed by Alex Krizhevsky et al. was the first popular CNN structure. The network won the ImageNet ILSVRC challenge in 2012. The reason to the good results were the large, deep structure of the model in addition to stacked convolutional layers without pooling in the midde.[15]

In 2014 the ILSVRC winner was a CNN from Google developed by Szegedy et al. The GoogLeNet introduced a module called inception that reduced the number of parameters in the network by doing multiple convolutions with different sized kernels in the convolutional layer [29]. This way the network needs around 4 million parameters compared to 60 million in AlexNet.

In 2015 Kaiming He et al. developed a CNN architecture called ResNet which won the ILSVRC the same year [7]. The ResNet introduced a way of adding identity mapping to the convolutional layer stack outputs in order to address the accuracy degradation problem in deep CNNs. The way called residual learning allows a shortcut connection to the feedforward networks structure without increasing the computational complexity. The residual network is now state-of-the art and can be considered as the golden standard for CNN applications.

3.2.3 3D convolutional neural networks

Movement in videos can be visualized by showing multiple images in a short time frame. Human brain needs to see multiple seconds of movement to be able to recognize it with certainty. The same requirements apply to machines. There are many neural network architectures with some kind of memory like recurrent neural networks mentioned in Section 3.1. RNNs use feedback loops to mix earlier data with current input thus creating understanding of movement. CNNs are fully feedforward networks so the movement features extraction needs other measures like 3D convolutions.

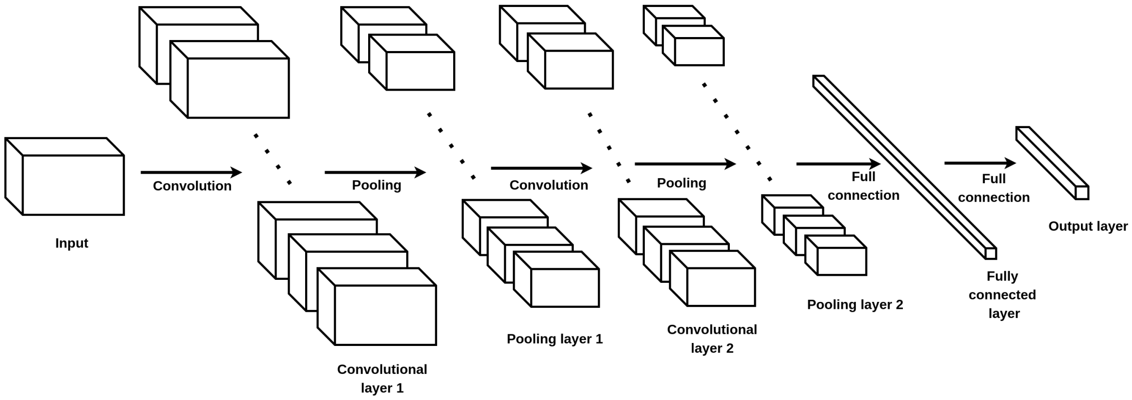


Figure 3.4 *3D CNN structure. A typical 3D convolutional neural networks architecture with several convolutional layers followed by subsampling and at the end one fully connected layer followed by softmax output layer. Input rectangle represents a stack of consecutive video frames.*

3D convolution is essentially an extension to the 2D convolution, described in the previous section. In 2D convolutional layer the spatial data of an input is considered towards final features. In 3D convolution, the mixture of spatial and temporal data is utilized in order to learn movement. The 3D convolution kernel is a 3 dimensional cube which considers local spatial region in adjacent frames. For example if the kernel is size 2x2x2x1 the dot product considers receptive field of 2x2 in two consecutive frames (Figure 3.5). The 4th dimension is the color channel (which in this case is 1 due to usage of gray scale image). The 3D convolution formula is defined as:

$$y_{xyt} = f(\sum_i \sum_j \sum_k w_{ijk} v_{(x+i)(y+j)(t+k)} + b), \tag{3.16}$$

where y_{xyt} is a feature map value at (x,y,t), f is the activation function, w_{ijk} is a kernel weight and $v_{(x+i)(y+j)(t+k)}$ is an input value at (x+i,y+j,t+k).[9] This way every scalar in the feature map has the information from multiple frames in that particular spatial region.

3D CNN architecture is similar to the 2D version, but the convolution kernels are extended as mentioned. Also the model input has to be modified into a stack of consecutive frames. Commonly the sample size in action recognition is around tens of consecutive frames which means a fraction of a second if every frame is considered in standard 30 fps video. Rest of the model is identical.

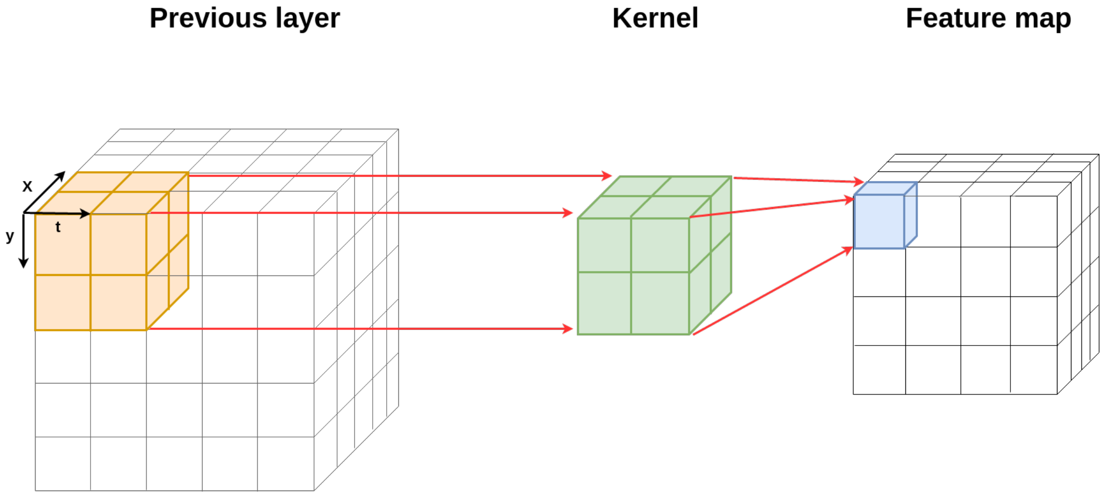


Figure 3.5 *3D convolution. A previous layer feature map (left), a 2x2x2x1 kernel (middle) and the output (right). X and Y are the spatial dimensions of the layer output and t is the temporal dimension i.e. frames/time. In this example zero pooling P is 0 and stride S is 1 so the feature map dimensions change in the process.*

4. METHODS

In this chapter, we go through the implementation related techniques and methods. First we review what is required for pre-processing followed by the final network architecture details. Lastly, we will present the workflows used for training and evaluation of the designed model.

The main dataset used in this thesis is the UCF-101 human actions dataset presented in the section 2.2. Due to the computationally heavy and time consuming training operations, the network is taught from scratch using four smaller subsets of this large dataset (see Table 4.1). Each subset consists of 10 classes of different actions from the same subclass. The subsets *Sports1* and *Sports2* includes only sports activities like surfing and skydiving. The other two sets *Interactions1* and *Interactions2* consists of interactions between a human and object like playing guitar and brushing teeth.

No.	**Sports1**	**Sports2**	**Interactions1**	**Interactions2**
1	IceDancing	RopeClimbing	ApplyLipstick	Knitting
2	Rafting	PullUps	BlowDryHair	PlayingViolin
3	Skijet	BoxingSpeedBag	Typing	PlayingSitar
4	Rowing	WallPushups	BlowingCandles	MoppingFloor
5	Skiing	TrampolineJumping	ShavingBeard	Hammering
6	CliffDiving	Kayaking	WritingOnBoard	PlayingDhol
7	Diving	SalsaSpin	ApplyEyeMakeup	HeadMassage
8	SkyDiving	TaiChi	PlayingFlute	YoYo
9	JumpRope	VolleyballSpiking	PlayingGuitar	Haircut
10	Surfing	BasketballDunk	BrushingTeeth	JugglingBalls

Table 4.1 *UCF-101 subsets used in this work and the classes belonging to them.*

4.1 Pre-processing

The beauty of neural networks is that it requires little to none pre-processing in order to work properly. The learning algorithms are responsible for the learning process and they do it automatically from the original data.

The training process in this thesis was relatively simple. The video data was gathered frame by frame and then split into overlapping samples of 16 consecutive frames. We picked 16 frames by following the findings in [24] and [30]. The studies show, that the most prominent motions can be detected even with small 1-7 frames samples in 30 FPS videos. Intuitively, longer clips futher enhance the results giving better separability. The overlap is 8 frames meaning that samples were gathered using a 16 frame sliding window with a 50% stride. A large sample overlap is an easy way to increase the dataset size and to make sure that movement at every moment is fed to the model. The sample overlap is visualized in the Figure 4.1. In addition, we further augmented the data by mirroring frames for every video which doubles the training set size.

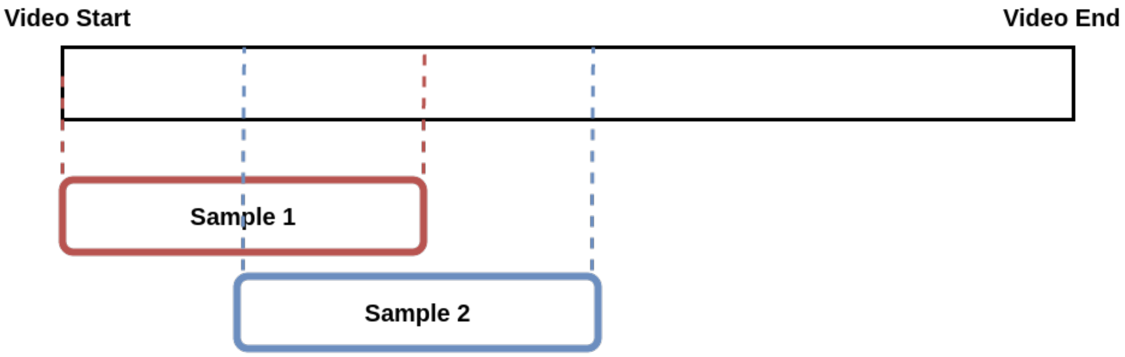


Figure 4.1 *Video clip sampling visualization with 50% overlap.*

CNNs learn well high level abstract features using spatial information from each frame. In this thesis, we ephasized the effects of temporal dimension by using grayscale frames. By doing this, the network does not rely too much on typical colors of the background, which means that temporal dimension might add more value to the training.

Due to the heavy memory requirements of 3D CNNs, the frame size had to be reduced. The original UCF-101 frame size is 320x240 but for the model, the size was shrinked to half of the original (160x120). Despite the small frame dimensions, the model was fully capable of getting the necessary image information for this task.

After resize we randomly cropped a 96x96 sized area of the video frames in order to add some jitter for the training process and to make the model usable for different aspect ratios. Here we assumed that the action of the video happens usually in the center of the frame. In some cases the action may be missed, but due to the iterative process this is an acceptable approach as the results show. Lastly the frame data was scaled between a range from 0 to 1 for the SGD algorithm to converge faster. Data normalization is a good practice in any kind of statistical analysis but here it is enough to just scale. We know that grayscale frame values range from 0 to 255 so scaling is done simply by dividing every frame value by 255.

4.2 Network architecture

In this section the chosen network architecture in reviewed. Designing neural networks is time consuming and difficult because of its complex set of hyper parameters and the synergy between them. The optimal network architecture may be different for different datasets and even for different subsets as seen later in the results. Due to these facts, the architecture is chosen here based on various best practices from state of the art studies and own experiments with the data in hand.

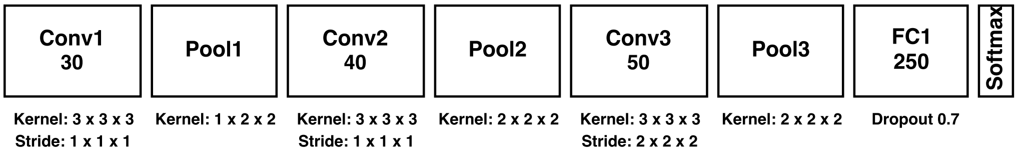


Figure 4.2 *10 class 3D CNN architecture. Network has three convolutional and max-pooling layers and one fully connected layer before softmax output. In each of the convolution layers the kernels are 3x3x3 with with stride 1 in every dimension excluding the last convolutional layer which has stride 1x2x2. Each pooling layer has 2x2x2 kernels, except the first one which has 1x2x2. The fully connected layer has 250 nodes with 70% dropout.*

The final network architecture used for the result section caluculations is presented in Figure 4.2. In this model, a zero padding layer works as an input layer. The input sample dimensions are 16x96x96x1 (16 gray scale frames) and we start by adding one zero around every dimension in order to get the same dimensions out of the first convolutional layer. The convolutional layers 1-3 have 30, 40 and 50 filters, respectively. The convolution itself is done with small kernels of size 3x3x3. This decision is based on the findings in 2D CNNs [26], where results prove small kernels to be efficient with deep architectures.

Each of the convolutions is followed by max-pooling layer reducing the spatial and temporal dimensions by half. After the first convolutional layer, only the spatial dimensions are reduced preserving the temporal information for the deeper layers. The last convolutional layer and the following pooling uses the stride of 2x2x2 in order to reduce the data into two dimensional 5x5 frame per filter. This way the we have reduced all of the temporal depth from 16 to one and the spatial dimensions from 96 to 5. The resulting 5x5x50 output is flattened into a 1250 long feature vector and then fed to fully connected layer for the actual classification part. In fully connected layer the feature vector lenght is reduced into one fifth and heavily regularized with high dropout ratio of 0.7 to overcome overfitting.

For comparison purposes, a 2D CNN was also implemented. The 2D network takes single frames as an input so temporal dimension is totally discarded. The 2D model follows the same architecture than the 3D version but only with 2D convolutions instead of 3D. The 2D model is presented in the Figure 4.3.

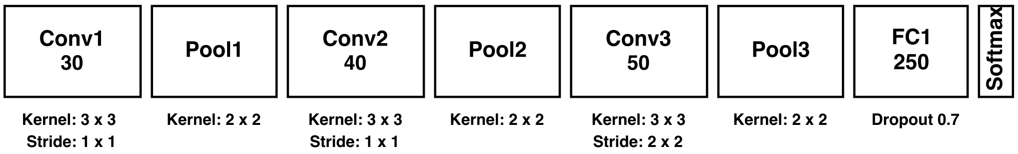


Figure 4.3 *10 class 2D CNN architecture. Network has three convolutional and max-pooling layers and one fully connected layer before softmax output. In each of the convolution layers the kernels are 3x3 with with stride 1 in every dimension excluding the last convolutional layer which has stride 2x2. Each pooling layer has 2x2 kernels. The fully connected layer has 250 nodes with 70% dropout.*

Neuron activation for each hidden layer is done using ReLU. According to the best practices study by D. Mishkin et al. [20], exponential linear unit (ELU) and ReLU perform almost equally well. ReLU was finally chosen due to its overall popularity.

Output layer activation is implemented with softmax as it is the correct approach for K>2 classification tasks as mentioned in Section 3.1.2.

The loss function in this model in implemented as cross-entropy (CE) which is defined as follows:

$$E(n) = \frac{1}{m} \sum_{j=0}^{m} [d_j \ln(y_j(n)) + (1 - d_j) \ln(1 - y_j(n))], \tag{4.1}$$

where m is the number of neurons. d_j and y_j are the desired and estimated outputs, respectively. CE is used due to its natural fit for classification tasks.

SGD works as an optimizer for the network as already mentioned in Section 3.1.2. We start by using the default learning rate of 0.1 which is divided by 10 after each 10 full epochs. Due to the custom batch process, the decay is left 0.

4.3 Training and evaluation

The main programming language for this thesis is Python due to its simplicity and capabilities for fast prototyping. Also many popular neural networks libraries are written in Python so the choice was clear.

In this paper, Theano is used as a processing backend and Keras as the main neural network library. Theano is a numerical computation library for Python that enables also CUDA (Compute Unified Device Architecture) processing with Nvidia GPUs (Graphics Processing Units). Keras is a modular neural networks library for Python and can be run on top of Theano or Google's TensorFlow. In addition, being up to date with current ANN research, Keras supports convolutional neural networks up to three dimensions.

The main dataset used for this study is UCF-101 which has short, around 10 second video clips from 101 different human actions. The clips in one class are split into groups according to the original video they are extracted from. The dataset offers 3 different training and test splits so that the groups are not mixed and distortion of the evaluation results is prevented.

The full UCF-101 dataset takes 7,2 GB of hard drive space. This amount of data rarely fits computer cache so the training is done here using a customized batch process. In order to neural networks to learn in Keras, it has to see a statistically meaningful chunk of data with every iteration. This means that we prefer to give it data from every possible class every iteration.

In this work the data is read one video at a time for every class. This means that the model sees one video from each of the classes during one iteration. We can call this 10 video set a mini batch. The dataset has roughly 100 videos per class, from which 75% is used for training. In order to get through all of the training data, we iterate through the mini batches and treat that as a full epoch in our code. The batch process was implemented due to the low memory on the local test machine.

In Keras the fit function, which handles the training, has a batch size as an input. This number means the amount of samples that goes to the optimization algorithm. Here we can call this a micro batch. High micro batch size requires a lot of video memory from the GPU so the micro batch size is set to 25. Normally this should be something between 10-500 depending on the hardware. It has to be noted, that the learning rate is then proportional to this value. According to the studies, the learning rate should be lowered for small micro batch sizes [20]. In this case it has a marginal effect and only slows the training process, so the learning rate is left to its default starting value.

During the training, no automatic evaluation split was used. By default in Keras the validation accuracy and loss is calculated by using the training data, describing how well the model has learnt the training data. Normally, training is continued as long as validation accuracy increases or if some pre-defined amount of epochs has been reached.

In this project, during every full epoch all of the training data was fed to the network. After each epoch, a test with the main testing set was performed (defined by the UCF-101 train-test-split 1) in order to visualize the learning as a function of epochs. The training was stopped after 45 epochs, which was enough for the networks validation accuracy to fully converge even with harder subsets. Estimations were done by averaging the sample results per video. This means we assigned the whole video clip to estimated class after seeing it. The averaging here means taking the majority vote from every sample of one video clip. With this information we calculated top 1 video accuracy for the dataset, which means that the estimation was considered correct only if the highest probability is the correct class.

The gains of temporal dimension was analyzed by comparing 2D and 3D CNN results. Every eight frames were given to the 2D model to match the amount of samples of 3D model.

5. EVALUATION

5.1 Visualization

One could say that CNN can see or understand the given input image or set of frames, but in reality that is not quite the case. In practice, the network understands a decomposition of the input space as a hierarchical network of convolution kernels and a probabilistic mapping between them. In Figure 5.1 can be seen the corresponding outputs for each network layer. The output image stacks were created with random kernels. Although the input data is gray scale, the layer outputs are coloured for visualization purposes.

To clarify the concept of feature extraction, the 3D CNN takes a stack of consecutive grayscale frames as an input and creates 50 small feature maps which are then transformed into a final one dimensional feature vector (see Figure 5.2).

In Figure 5.3 some of the 30 first convolution layer outputs are visualized. The colors help to understand how the different convolution filters affect the output. Every filter is different so the different aspects of the video clip is enhanced in the output of each frame. The actual operations are in 3D and the consecutive frames are used for convolution calculations. For visualization purposes only one frame per filter is shown, so the full affect of the movement between frames can not be visualized clearly. In the Figure 5.4 some of the corresponding first layer activation outputs can be seen. By analyzing the activation outputs, it is easier to see how different convolution kernels highlight the input.

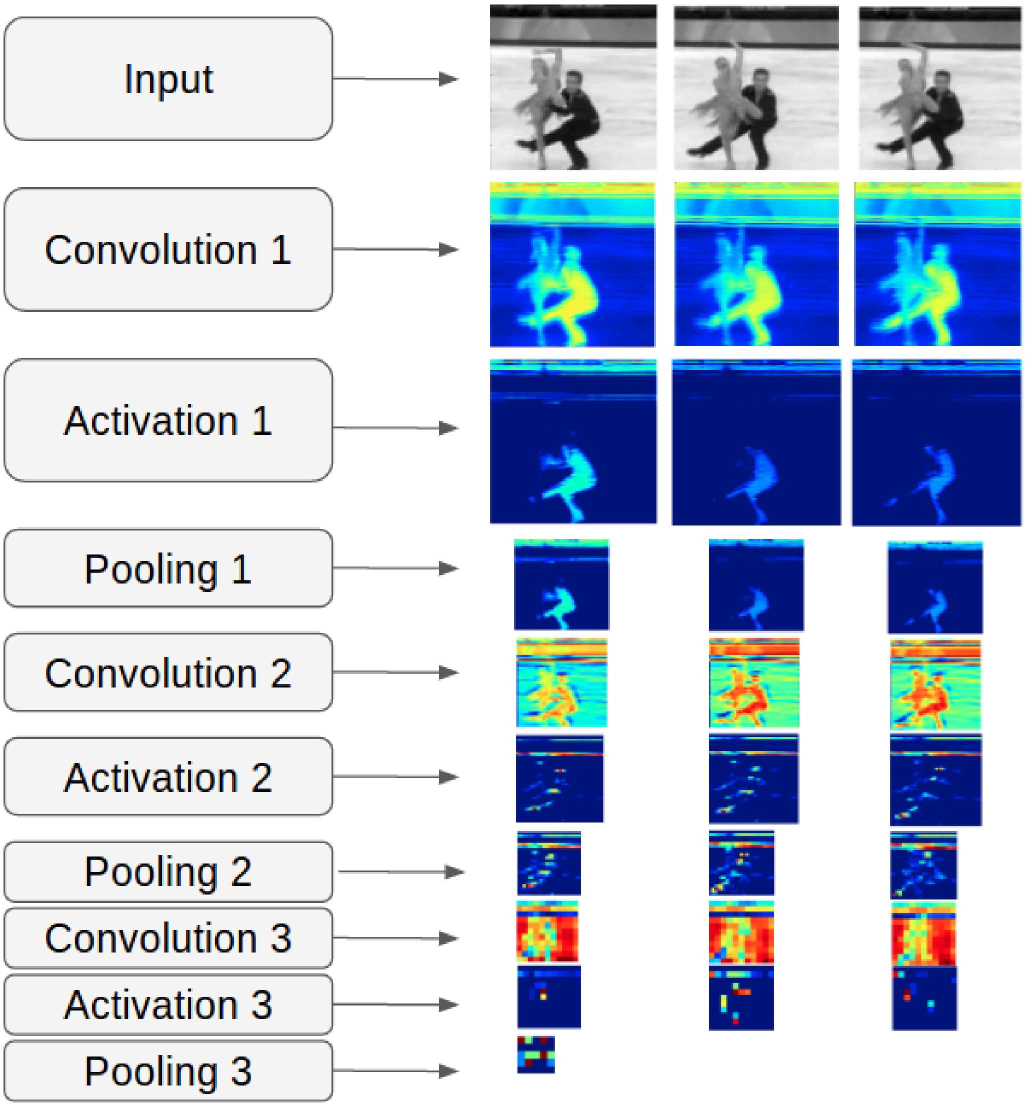


***Figure* 5.1** *Intermediate 3D CNN layer outputs. The first three frames of one video sample is visualized after every network layer. All of the outputs are products of one randomly chosen convolution filter. Note that with this network architecture we finally end up with only two dimensional output after the last pooling layer. Colors are added for visualization purposes.*

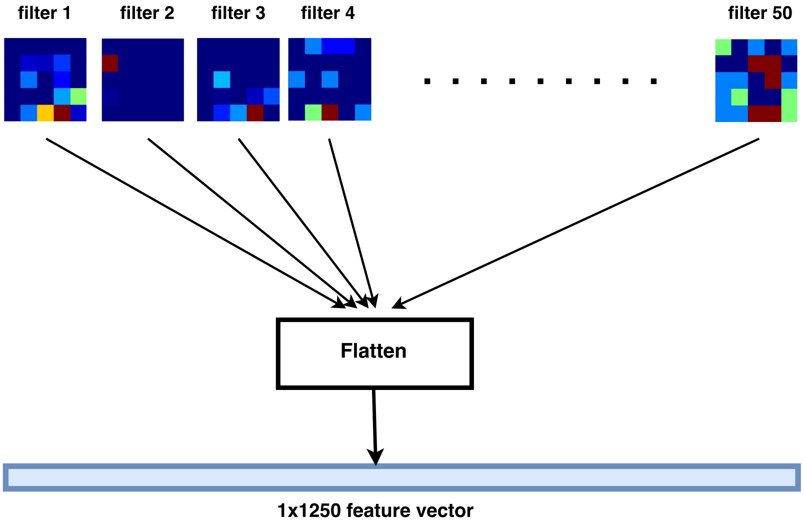


***Figure* 5.2** *Feature vector creation. The vector is concatenated from the last pooling layer putputs and then fed to the fully connected layers for classification.*

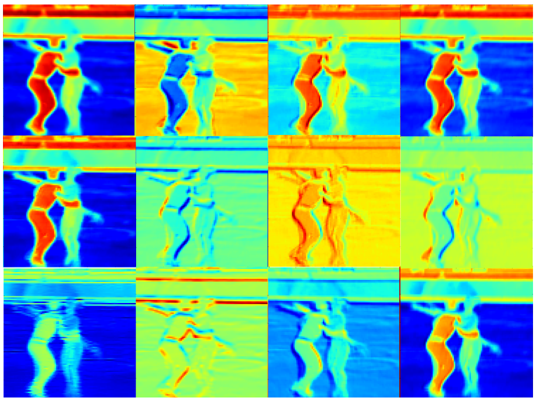


***Figure* 5.3** *Convolution filter outputs. Some of the 30 filter outputs of the first convolution layer. Colors are added for visualization purposes.*

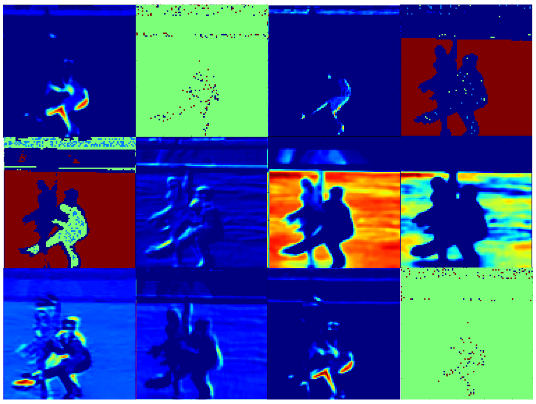


Figure 5.4 *Activation outputs. Some of the 30 activation outputs of the first convolution layer. Colors are added for visualization purposes.*

5.2 Finding a good architecture

In this section, we present implementation defining details and justification for them.

5.2.1 Optimization function parameters

In Figure 5.5 one can see two sets of test and validation accuracy graphs from an example network training. One network is trained with default parameters (left figure) and the other network is trained with fine tuned parameters (right figure). The x-axis is the number of full epochs and y-axis is the video accuracy. In the left figure, it is important to notice, that although the top 1 video accuracy quickly converges around 75% there is a clear zigzag around that value. In this case, Keras SGD default learning rate of 0.01 with 0 decay and momentum (see equation 3.9) causes the accuracy to oscillate during the later training. High learning rate works as a multiplier to the network weight changes, which results this oscillation in the later training when a network has reached its capacity to learn.

Essentially, we want test accuracy to reach its maximum and stay close to it for the rest of the training until the learning is stopped. One popular way to fix the accuracy oscillation is to reduce the learning rate over time [30]. In this study the learning rate is divided by 10 after every 10 full epochs starting from the value $\eta = 0.01$. The right graph in Figure 5.5 shows the effect of decaying learning rate.

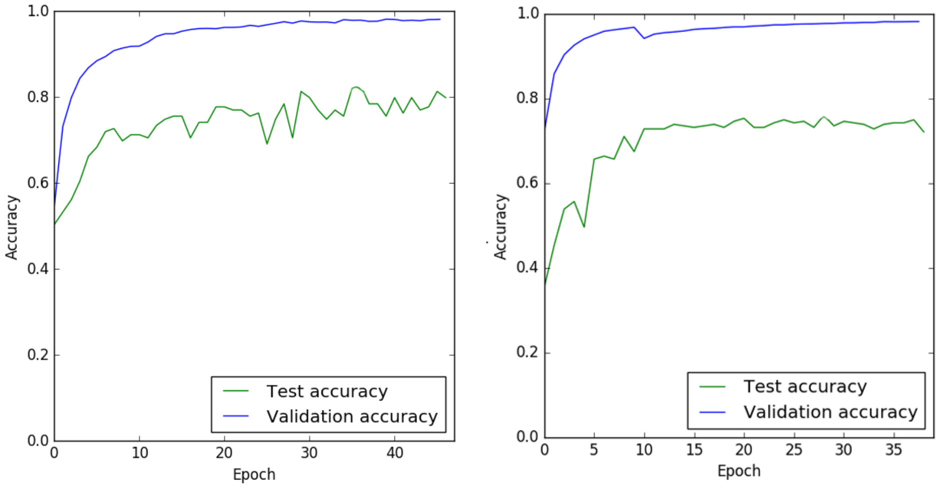


Figure 5.5 *Test and validation accuracies for example network using default SGD parameters (left). Another example network with decreasing leraning rate (right).*

5.2.2 Generalization

Batch normalization was also tested with the model. For sports1 subset it did not provide any added value but made the training slower and required lot more video memory. With this kind of not-so-deep architecture, there is no risk of gradient explosion or vanishing so it is better to leave out altogether. With deeper and larger networks it is still wise to add batch normalization if the training keeps halting or if validation accuracy suddenly drops close to zero.

Due to the fact, that the network is trained from a scratch, we use relatively high dropout ratio in the last fully connected layer. In the 10 class case, using lower than 70% dropout ratio caused severe overfitting.

5.2.3 Number of layers

As mentioned in Section 3.2, CNNs have usually couple of convolution layers followed by subsampling. Finding the right amount of layers and kernels is difficult and highly data dependent, but generally neural networks require a lot more data than some other machine learning methods. In order to decide the suitable amount of layers for the data in hand, the intermediate layer outputs can be extracted and analyzed. Extracting different pooling layer outputs and clustering them for example with t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm gives an approximation of the feature separability [31]. In short, t-SNE is dimensionality reduction algorithm for high dimensional data. The algorith is developed by L.J.P. van der Maaten and can be also found from *sklearn* library for Python. Here t-SNE gives us an idea about the class separability in two dimensions which helps to understand the feature strenght for classification.

In the beginning, we started with just three classes to find the first working model. In the following Figure 5.6, the t-SNE results can be seen for different intermediate maxpooling layer outputs for three class test subset. The model had three convolutional layers each followed by maxpooling.

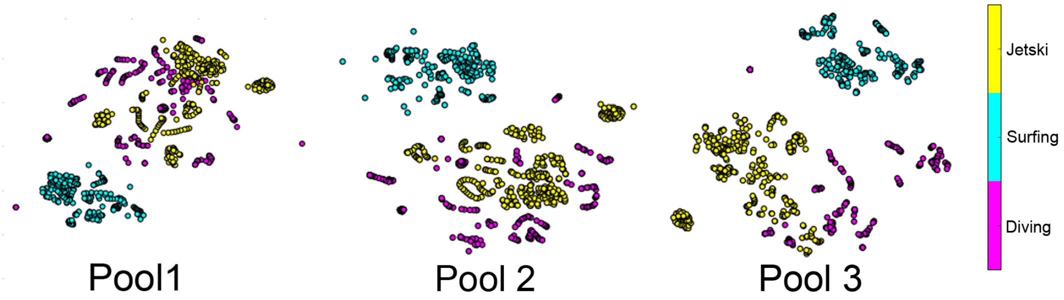


Figure 5.6 *Embedded t-SNE features for different layer outputs.*

The t-SNE results are clear with the small three class subset. Features after third maxpooling are clearly the easiest to separate. Already after one layer, the surfing class features can be separated from the others, but jetski and diving classes are clearly separable only after the third layer. The test were done also with 4 convolutional layers, but the model had already troubles to learn at all so we ended up with three layers. Similar test results for the 10 class subset are shown in the table 5.1 from which we can again see, that three layers causes highes top 1 video accuracy. Other hyper parameters might affect the results so that 4 layers would give better results, but in this work we ephasized on training speed due to slow GPU. Adding a layer slows the training drastically.

Layers	**Accuracy**
2	0.66
3	0.76
4	0.73

Table 5.1 *Sports1 accuracy with different amount of convolution layers.*

5.3 Performance

In this section, the final network performance metrics are presented and analyzed.

5.3.1 Video accuracy

The top 1 video accuracy results for different subsets and methods are presented in the Table 5.2. Even though results do not reach the accuracy of state of the art, they offer a good example how the 3D CNN performs on this kind of problem. The results follow the findings in the previous studies. D. Tran et al. achived 83% accuracy with the 3D CNN and linear SVM on UCF-101 [30] and Simonyan and Zisserman [25] 73% accuracy with 2D CNN. Although they used the full 101 class set, the results in this thesis can be compared if we look the relative difference between 2D and 3D methods.

Model	**Sports1**	**Sports2**	**Interactions1**	**Interactions2**
3D CNN	0.76	0.71	0.44	0.49
2D CNN	0.63	0.40	0.33	0.39

Table 5.2 *Top 1 video accuracies. Test results for each subset using the split 1.*

The results of previous works and performed tests show, that the spatial information creates the most dominant features for classification and temporal dimension offers around 10% gain. Learning curves of both 2D and 3D CNNs can be observed from the Figure 5.7. From the graphs one can say that most of the learning is done during the first 10 full epochs after which test accuracy starts to converge to its maximum.

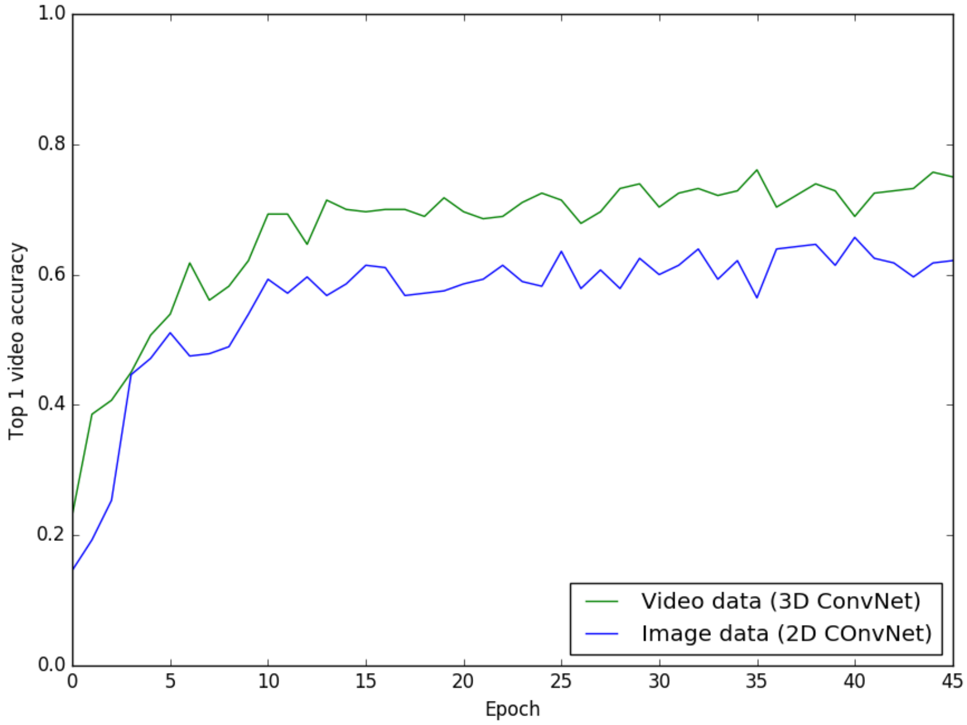
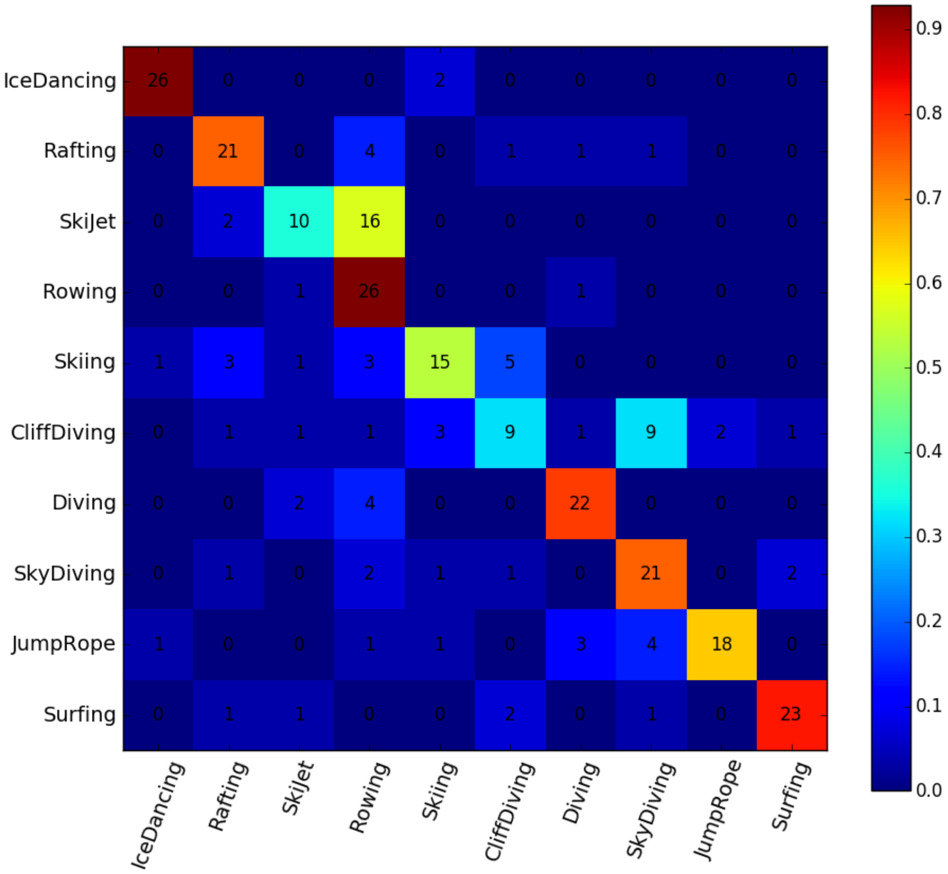


Figure 5.7 *3D CNN vs 2D CNN. Sports1 test result comparison between grayscale image and video data*

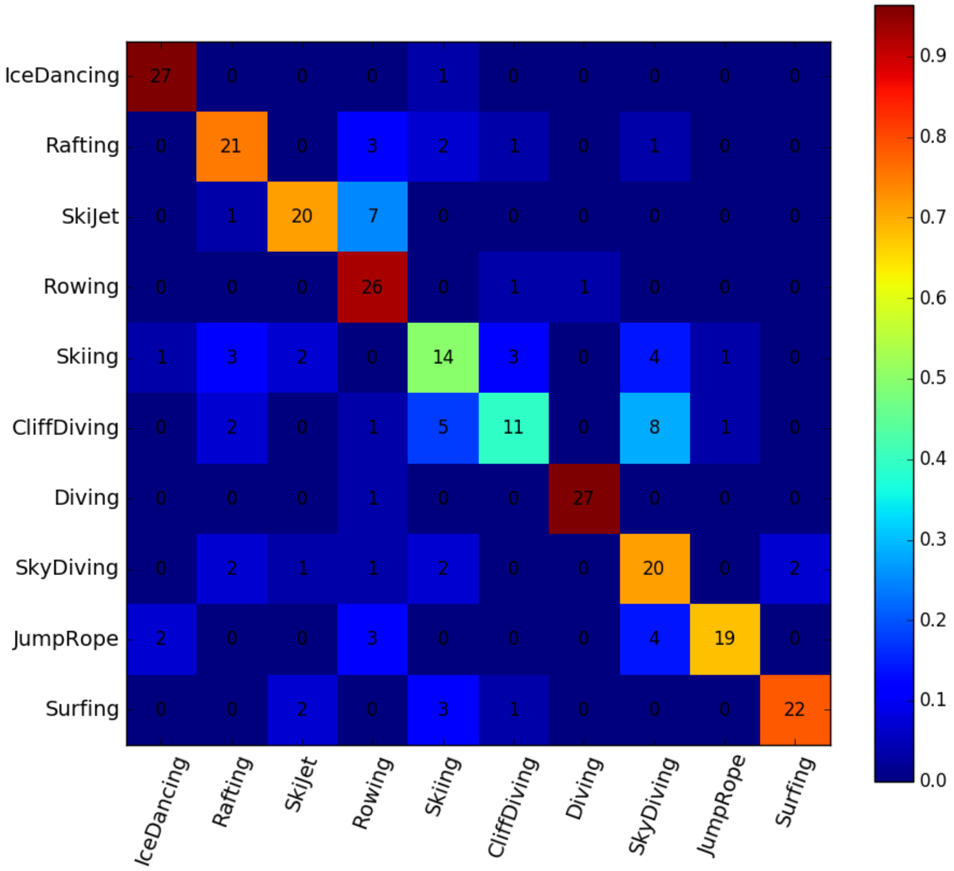
Even though the difference between spatial and spatio-temporal network results is small, it is easy to see that the video data offers more consistent results. In Figure 5.8, one can see that for example the skijet class is poorly classified and in Figure 5.9 the skijet estimates are distributed more sparsely between skijet and rowing.

The way 3D convolutions work in this case is intuitive for humans. Skijet and rowing classes have some vehicle gliding on the water which makes the pure movement similar compared to the other trained classes. Frame based 2D network mixes these two classes probably due to the similar background and the dominant tones in it. Similar results can be seen when looking at the surfing class results. 2D CNN confuses that with similar looking classes with water backrounds, while 3D CNN estimates it being skiing because the movement is again very similar to surfing.



	IceDancing	Rafting	SkiJet	Rowing	Skiing	CliffDiving	Diving	SkyDiving	JumpRope	Surfing
IceDancing	26	0	0	0	2	0	0	0	0	0
Rafting	0	21	0	4	0	1	1	1	0	0
SkiJet	0	2	10	16	0	0	0	0	0	0
Rowing	0	0	1	26	0	0	1	0	0	0
Skiing	1	3	1	3	15	5	0	0	0	0
CliffDiving	0	1	1	1	3	9	1	9	2	1
Diving	0	0	2	4	0	0	22	0	0	0
SkyDiving	0	1	0	2	1	1	0	21	0	2
JumpRope	1	0	0	1	1	0	3	4	18	0
Surfing	0	1	1	0	0	2	0	1	0	23

***Figure* 5.8** *2D CNN confusion matrix. Sports1 confusion matrix for gray scale image data. Y-axis is the actual class and the X-axis is the network estimate count.*



	IceDancing	Rafting	SkiJet	Rowing	Skiing	CliffDiving	Diving	SkyDiving	JumpRope	Surfing
IceDancing	27	0	0	0	1	0	0	0	0	0
Rafting	0	21	0	3	2	1	0	1	0	0
SkiJet	0	1	20	7	0	0	0	0	0	0
Rowing	0	0	0	26	0	1	1	0	0	0
Skiing	1	3	2	0	14	3	0	4	1	0
CliffDiving	0	2	0	1	5	11	0	8	1	0
Diving	0	0	0	1	0	0	27	0	0	0
SkyDiving	0	2	1	1	2	0	0	20	0	2
JumpRope	2	0	0	3	0	0	0	4	19	0
Surfing	0	0	2	0	3	1	0	0	0	22

Figure 5.9 *3D CNN confusion matrix. Sports1 confusion matrix for gray scale video data. Y-axis is the actual class and the X-axis is the network estimate count.*

5.3.2 3D CNN feature analysis

If the top 1 video accuracy in Figure 5.10 is analyzed in respect to number of samples, one can see that already one sample gives fairly high results (68 %) over the sports classes. This supports the fact that the spatial information is the most dominant and the temporal information gives just marginally better certainty to the final video classification. The maximum video accuracy is reached after seeing 25 samples of every clip. This translates to roughly 8.5 seconds of video.

In the small 10 class case we also noticed, that 3D CNN creates very powerful and compact features. In the Figure 5.11 one can see the sample accuracy as the function of feature dimension. By sample accuracy, we mean the sample level

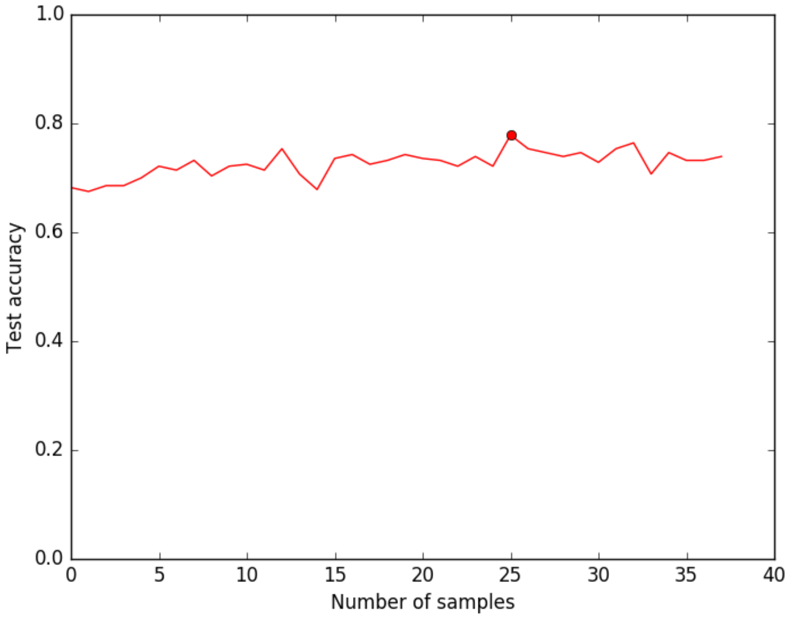


Figure 5.10 *3D CNN accuracy with different number of samples. Sports1 classes top 1 video accuracy in respect to number of samples. The maximum sample accuracy is highlighted.*

accuracy instead of the video accuracy, which is calculated by majority voting sample level classifications. The feature vector is extracted from the FC layer output and then fed to linear SVM classifier. The classifier is implemented in sklearn library and here the default parameters were used. The full 250 long feature vector offers 69% sample accuracy and the results show that we almost achieve that results by using only the first 15 values. Figure 5.11 was created by simply adding values randomly starting from 1. Picking the most important values may reduce that even more, but this test already demonstrates the power of 3D CNN features.

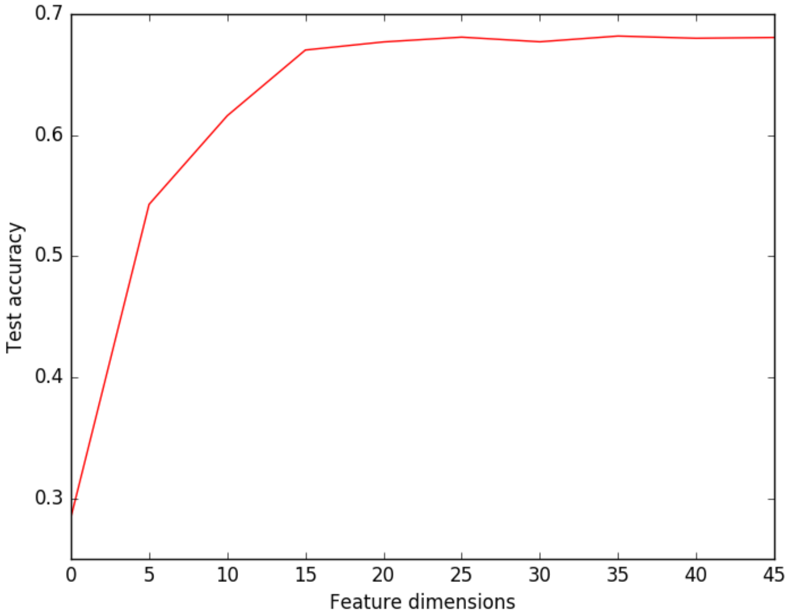


Figure 5.11 *3D CNN feature strength. Sports1 sample accuracy using SVM and different number of sample dimensions.*

6. CONCLUSIONS

In this thesis, we studied the capabilities of 3D CNNs in human action classification. The comparison was made with image based networks in order to analyze spatio-temporal feature strengths. For testing the networks, four small human action datasets were used.

The achieved results were lower than expected after studying the field. We noticed that large movements are easier to classify than subtle movements in a small area of the frame. With the computing power at hand and within the time frame of this thesis we managed to achieve mediocre results with a relatively shallow network. Despite the overall lower accuracies, results follow the findings in previous works. Relative differences between spatial and spatio-temporal features are distinct. Adding a 3rd dimension to convolutional operations offers a higher accuracy in video classification. The two sports dataset accuracies were 76% and 71% while the human interactions dataset accuracies were 41% and 49%. For each of the dataset, the 3D CNN results were approximately 10% higher than 2D CNN results.

3D CNN creates generic abstract features that are also highly compact and simple. As opposed to previous studies, we found that creating a generic, high accuracy deep network architecture is not simple. The amount of DNN hyperparameters and the correlations between them is huge, which makes finding a good general model time consuming and cumbersome. This means that the means have to be thoroughly analyzed before choosing between BoW and DNN methods because both require a lot of manual work. In addition, 3D convolutions are computationally highly demanding, which was one of the main reasons why we had difficulties with deeper network architectures. Another problem was scaling in the beginning of the research. Great results were achieved with small class numbers, but increasing the class number decreased the overall accuracy even when the network was scaled both deeper and wider. Because of the time consuming scaling issues, we decided to continue with fixed amount of labels.

With the designed model, spatio-temporal 3D CNN features alone lack the descriptive power needed for small subtle movement separation. In this thesis the human interactions subsets were difficult to classify and the same trend can be seen for example with the HMDB-51 dataset results in previous studies.

Depending on the specific task at hand, the usage of 3D CNNs should be disputed due to its hardware requirements. It has been proved in the previous studies, that 3D CNN can offer high accuracy with a correct model architecture, but requires a lot of computing power and weeks of training time. In some cases 2D CNNs perform equally well and takes just a fraction of time to train. Performance wise, a good choice would be to extract the high level features with 2D CNN and combining them with some BoW methods to further increase the accuracy.

Despite the mediocre results, 3D CNN features seem powerful and compact. With a proper hardware and a deep network architecture, 3D CNNs offer high performing features that are perfect for general models as can be seen from the state-of-the-art results. For more specific tasks, these features should be coupled with more local features and fine tuned in order to get reliable results.

For further studies, 3D CNN features could be powerful for recurrent LSTM network training. This combination would add interesting global video level features, if the whole video would be classified. Another suggestion would be to give optical flow trajectory data to a 3D CNN. Here, optical flow would work as a motion filter before feeding the data into a 3D CNN. One sample would be then a stack of optical flow representations.

There is a lot of room for improvement considering the action classification problem but the strong features make 3D CNNs an interesting subject for further research and development in this field.

BIBLIOGRAPHY

[1] I. G. Y. Bengio and A. Courville, “Deep learning,” 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[2] G. E. Dahl, T. N. Sainath, and G. E. Hinton, “Improving deep neural networks for LVCSR using rectified linear units and dropout,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8609–8613.

[3] G. Farnebäck, *Two-Frame Motion Estimation Based on Polynomial Expansion.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370. [Online]. Available: http://dx.doi.org/10.1007/3-540-45103-X_50

[4] A. Graves, *Neural Networks.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 15–35. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24797-2_3

[5] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, “Recent advances in convolutional neural networks,” *CoRR*, vol. abs/1512.07108, 2015. [Online]. Available: http://arxiv.org/abs/1512.07108

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[9] J. Huang, W. Zhou, H. Li, and W. Li, “Sign language recognition using 3d convolutional neural networks,” in *2015 IEEE International Conference on Multimedia and Expo (ICME)*, June 2015, pp. 1–6.

[10] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[11] S. Ji, W. Xu, M. Yang, and K. Yu, “3D convolutional neural networks for human action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan 2013.

[12] Y. Jiang, J. Liu, A. R. Zamir, I. I. Laptev, M. Piccardi, M. Shah, and M. Sukthankar, “Thumos challenge: Action recognition with a large number of classes.” 2013. [Online]. Available: http://crcv.ucf.edu/ICCV13-Action-Workshop/

[13] A. Karpathy, “Convolutional neural networks, convolutional neural networks for visual recognition.” Available (cited 24.11.2016): http://cs231n.github.io/convolutional-networks/, Stanford Computer Science.

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1725–1732.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

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

[17] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj, “Beyond gaussian pyramid: Multi-skip feature stacking for action recognition,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 204–212.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[19] M. Marszałek, I. Laptev, and C. Schmid, "Actions in context," in *IEEE Conference on Computer Vision & Pattern Recognition*, 2009.

[20] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of CNN advances on the imagenet," *CoRR*, vol. abs/1606.02228, 2016. [Online]. Available: http://arxiv.org/abs/1606.02228

[21] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." *International Conference on Machine Learning (ICML)*, vol. 28, pp. 1310–1318, 2013.

[22] X. Peng, L. Wang, X. Wang, and Y. Qiao, "Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice," *Computer Vision and Image Understanding*, vol. 150, pp. 109–125, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2016.03.013

[23] X. Peng, C. Zou, Y. Qiao, and Q. Peng, *Action Recognition with Stacked Fisher Vectors*. Cham: Springer International Publishing, 2014, pp. 581–595. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10602-1_38

[24] K. Schindler and L. J. V. Gool, "Action snippets: How many frames does human action recognition require?" in *Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2008. [Online]. Available: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2008.html#SchindlerG08

[25] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 568–576. [Online]. Available: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.pdf

[26] ——, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[27] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human action classes from videos in the wild," in *CRCV-TR-12-01*, 2012. [Online]. Available: http://crcv.ucf.edu/papers/UCF101_CRCV-TR-12-01.pdf

[28] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, “Striving for simplicity: The all convolutional net,” *CoRR*, vol. abs/1412.6806, 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3D convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 4489–4497.

[31] L. van der Maaten and G. Hinton, “Visualizing high-dimensional data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[32] H. Wang, A. Kläser, C. Schmid, and C. L. Liu, “Action recognition by dense trajectories,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, June 2011, pp. 3169–3176.

[33] H. Wang and C. Schmid, “Action recognition with improved trajectories,” in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 3551–3558.

[34] L. Wang, Y. Qiao, and X. Tang, “Action recognition with trajectory-pooled deep-convolutional descriptors,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 4305–4314.