# HUMAN ACTION CLASSIFICATION USING 3-D CONVOLUTIONAL NEURAL NETWORK

Deepak Pathak - 10222
Kaustubh Tapi - 10346
Mentor : Dr. Amitabha Mukerjee
Dept. of Computer Science and Engineering
IIT Kanpur

{deepakp,ktapi,amit} @ iitk.ac.in

April 15, 2012

**Abstract**

Our objective is to implement human action recognition in video streams through learning models. In this paper, we propose 3D convolution neural network model which can learn spatio-temporal features and classify human actions without any prior knowledge. Experimental results obtained on applying 3-D CNN over Weizmann dataset containing ten classifications gives comparable accuracy with recent research in this field.

## 1 Introduction

Action classification has always been an active area of research in computer science but its main approaches are using image processing which use manually engineered motions and texture descriptors calculated around STIPs (spatio-temporal interest points). In real-world scenarios like intelligent video surveillance, customer attributes, shopping behaviour analysis etc., the choice of feature is highly problem-dependent and it is rarely known which features are important for the task at hand.

We developed our 3D CNN model based on the idea explained by M. Baccouche [1]. In this project, we look at this problem using neural networks which automatically build high

level representation of raw input without any pre-processing. As deep learning models, Convolutional Neural Network (CNN) architecture have been applied on 2-D images and they have yielded very competitive performance in many image processing tasks, but their application in video stream classification is still an open and unexplored area. In our project, we successfully applied CNNs in 3-D to effectively incorporate action/motion classification in video analysis. We extended the 2-D CNN algorithm applied on MNIST image database [Mike O' Neill implementation[1]] of handwritten digits to 3-D CNN where third dimension corresponds to time frames. We propose to perform 3D convolution in the convolutional layers of CNNs so that discriminative features along both spatial and temporal dimensions are captured.

We evaluated the developed 3D CNN model on the Weizmann Dataset containing human action video dataset of ten classifications and it achieved a reasonably good accuracy and competitive performance without depending on manually engineered features demonstrating that 3-D CNN model is more effective for real world environments.

## 2 Convolution Neural network (2-D)

Convolutional neural networks are also known as "shared weight" neural networks introduced by LeCun [2]. ConvNets are the adaptation of multilayered neural deep architectures to deal with real world data.

This is done by the use of local receptive fields (better known as kernels) whose parameters are forced to be identical for all its possible locations of input array, a principle called weight sharing. The idea is that a small kernel window is moved over each node from a prior layer. In the CNN architecture, the sharing of weights over processing units reduces the number of free variables, increasing the generalization performance of the network. Weights are replicated over the input image, leading to intrinsic insensitivity to translations in the input.

A typical convolution framework is shown in Figure-1. Multiple planes are usually used in each layer (called Features maps (FMs)) so that multiple features can be detected. These layers are called convolutional layers. The network is trained with the usual backpropagation gradient-descent procedure. 2-D CNNs are applied on image dataset to classify them and extract spatio features.

[1]Refer: http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi

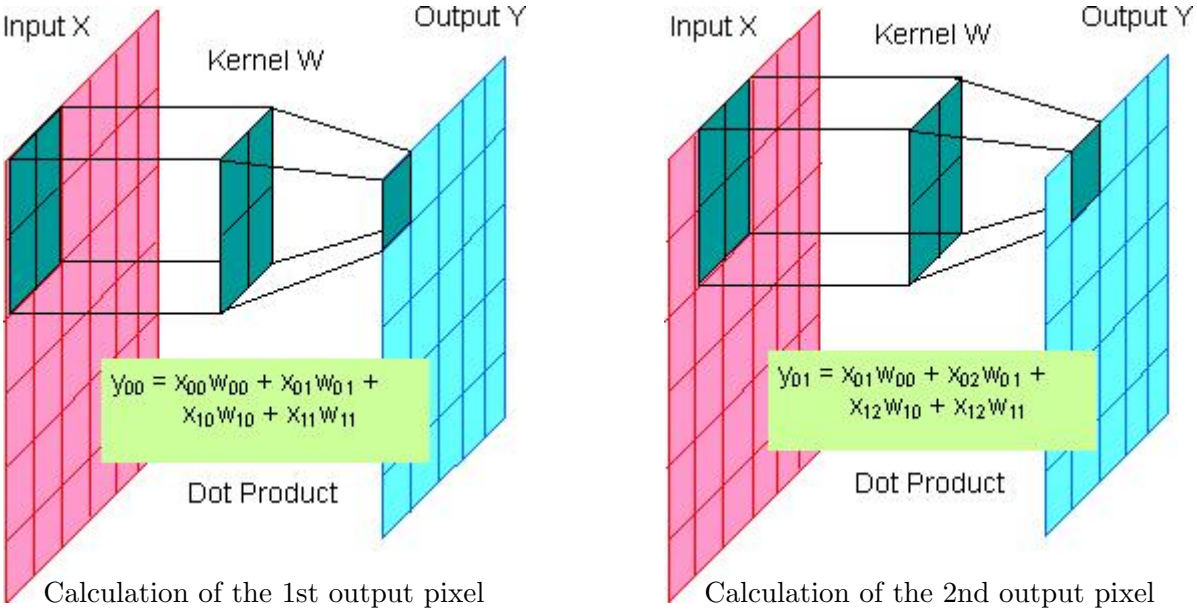



Figure-1: Image showing convolution on one Feature Map with (2x2) Kernel.
[CREDIT: Ishtiaq Rasool Khan's[2] implementation of 2-D CNN]

## 3 Spatio-temporal feature extraction using 3D Conv Nets

In 2D CNNs, convolutions are applied on the 2D feature maps to compute features from the spatial dimensions only. But for human action recognition in videos along with spatial features it is also desirable to capture the motion information encoded in multiple contiguous frames. To effectively incorporate the motion information in video analysis, we propose to perform 3D convolution in the convolutional layers of CNNs so that discriminative features along both spatial and temporal dimensions are captured. 3D convolution is achieved by convolving a 3D kernel to the cube formed by stacking multiple contiguous frames together. By this construction, the Feature-maps in the convolution layer are connected to multiple contiguous frames in the previous layer, thereby capturing motion information.

A 3D convolutional kernel can only extract one type of features from the frame cube, since the kernel weights are replicated across the entire cube. A general design principle of CNNs is that the number of feature maps should be increased in late layers by generating multiple types of features from the same set of lower-level feature maps. Similar to the case

[2]Available at : http://www1.i2r.a-star.edu.sg/ irkhan/conn1.html

of 2D convolution, this can be achieved by applying multiple 3D convolutions with distinct kernels to the same location in the previous layer.

Now we describe the architecture of 3-D CNN we developed for human action recognition on WEIZMANN Dataset [4].

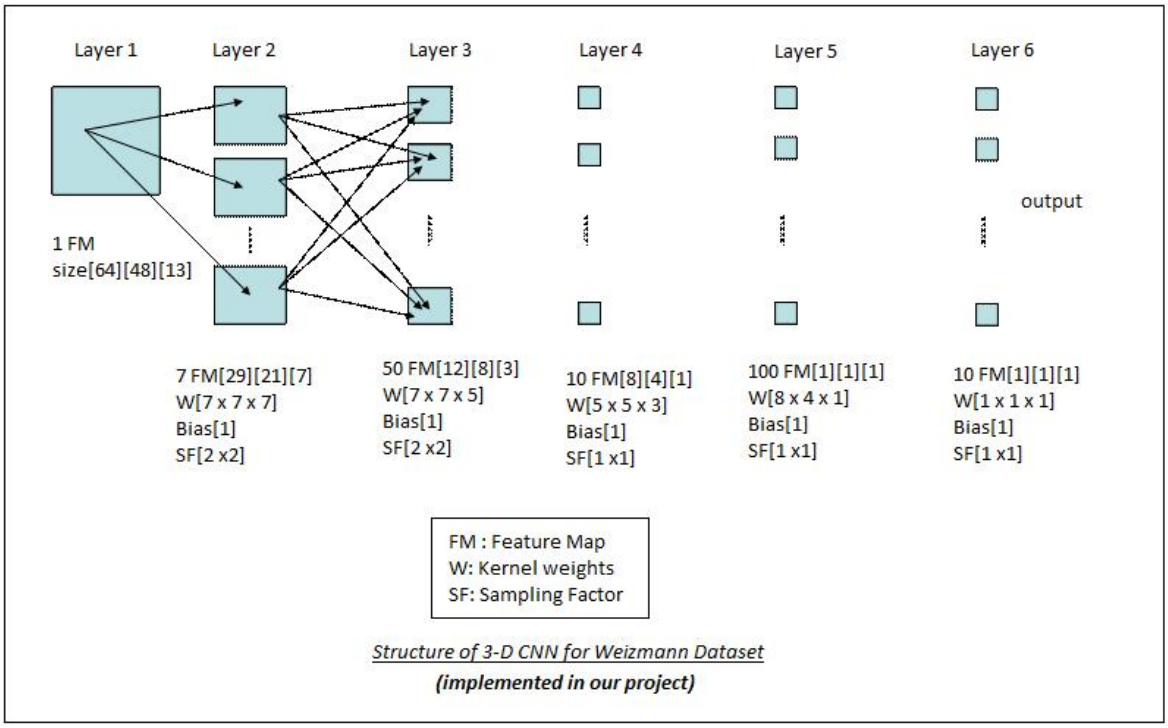



Figure-2: Structure of 3-D CNN constructed by us for Weizmann Dataset.

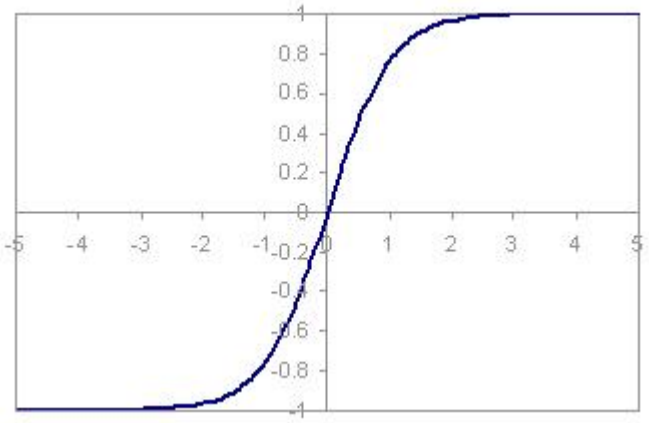
**Our CNN architecture constructed for Weizmann dataset :** In this architecture we consider 13 consecutive frames of size 64 x 48 as input to the 3-D CNN (input is [64 x 48 x 13] array). Our architecture consists of six layers including the input layer. After Input layer, next two layers are Convolution layers (C1 and C2) in which each layer involves two steps i.e. convolution followed by sub-sampling. This is followed by a 3rd convolution layer (C3) where no sub-sampling takes place (sub-sampling factor=1). This is further followed by two neuron layers (N1 and N2) containing the output layer. The last three layers are fully-connected layers with sub-sampling factor as unity. First convolution layer C1 consists of 7 feature maps of size 29 x 21 x 7(29 x 21 in spatial dimensions and 7 in temporal dimension). This layer is obtained by applying 7 different 3D kernels of size (7 x 7

x 7) followed by sub-sampling of factor 2. Sub-sampling makes our model resistant to small spatial distortions. As a result of this, the subsequent layers perform pattern recognition at progressively larger spatial scales, with lower resolution. Thus a CNN with several sub-sampling layers enables processing of large inputs, with relatively few free weights. Second convolution layer C2 consists of 50 feature maps of size 12 x 8 x 3. This layer is obtained by applying 3D kernels of size 7x7x5 followed by sub-sampling of factor 2. Third Convolution layer C3 consists of 10 feature maps of size 8 x 4 x 1 obtained by applying 3D kernels of size 5x5x3 on layer C2. At this stage by applying multiple stages of convolution and sub-sampling we are able to extract spatio - temporal features from the input. 13 consecutive input frames have been converted into a 320D ((8x4x1) x 10) feature vector capturing the motion information in the input frames after all the convolutional layers. Now the neuron layers (N1 and N2) act as a classical Multilayer perceptron classifier on the 320D input. Layer N1 consists of 100 nodes of size 1 x 1 and last layer N2 (output layer) consists of 10 nodes corresponding to different actions. We have used back-propagation algorithm for training the model.

**Comparison with M. Baccouche's [1] architecture :** Our CNN architecture is deeper and extensive than the one proposed by M. Baccouche [1] for KTH dataset. The number of feature maps in each layer in our construction is larger than their construction. The number of feature maps in each layer in their architecture are 7,35,5,50 and 6 in layers C1,C2,C3,N1,N2 respectively. Moreover, the kernel sizes in our architecture are larger as we deal with larger fearure maps as compared to their architecture.
Also, we have trained our 3-D CNN model using back-propagation with hessian learning to reduce the number of epochs required for the weights to converge. Our model operates on larger input size (64X48X13) as compared to (34X54X9).

When input data is normalized, then the performance is significantly improved after each convolution. Generally, the activation(normalisation) function should be symmetric, and the neural network should be trained to a value that is lower than the limits of the function. We have used the hyperbolic tangent as the activation function instead of classical sigmoid function.

$$x=F(y)=\tanh(y)$$




This function is a good choice because it's completely symmetric, as shown in the graph. The activation function used in the code is a scaled version of the hyperbolic tangent. Scaling causes the function to vary between $\pm 1.7159$, and permits us to train the network to values of $\pm 1.0$.

## 4 Learning through Back-propagation

Our 3D Convolution model learns by standard back-propagation. Back-propagation is an iterative process that starts with the last layer and moves backwards through the layers until the first layer is reached. Assume that for each layer, we know the error in the output of the layer. If we know the error of the output, then it is not hard to calculate changes for the weights, so as to reduce that error. The problem is that we can only observe the error in the output of the very last layer.

Back-propagation gives us a way to determine the error in the output of a prior layer given the output of a current layer. The process is therefore iterative: start at the last layer and calculate the change in the weights for the last layer. Then calculate the error in the output of the prior layer.

$$E_n^A = \tfrac{1}{2} \cdot \sum (x_n^i - T_n^i)^2 \qquad \text{(equation [i])}$$

$E_n^A$ is the error due to a single action A at the last layer n;

$x_n^i$ is the target output at the last layer (i.e., the desired output at the last layer); and

$T_n^i$ is the actual value of the output at the last layer.

Given equation [i], then taking the partial derivative yields:

$$\frac{dE_n^A}{dx_n^i} = x_n^i - T_n^i \qquad \text{(equation [ii])}$$

We use the numeric values for the quantities on the right side of equation [ii] in order to calculate numeric values for the derivative. Using the numeric values of the derivative, we calculate the numeric values for the changes in the weights, by applying the following two equations [iii] and then [iv]:

$$\frac{dE_n^A}{dy_n^i} = G(x_n^i) \cdot \frac{dE_n^A}{dx_n^i} \qquad \text{(equation [iii])}$$

where $G(x_n^i)$ is the derivative of the activation function.

$$\frac{dE_n^A}{dw_n^{ij}} = x_{n-1}^j \cdot \frac{dE_n^A}{dy_n^i} \qquad \text{(equation [iv])}$$

Then, using equation [ii] again and also equation [iii], we calculate the error for the previous layer, using the following equation [v]:

$$\frac{dE_{n-1}^A}{dx_{n-1}^k} = \sum_i w_n^{ik} \cdot \frac{dE_n^A}{dy_n^i} \qquad \text{(equation [v])}$$

Now take the numeric values obtained from equation [v], and use them in a repetition of equations [iii], [iv] and [v] for the immediately preceding layer. Now update the value of each weight in current layer n according to the formula:

$$(w_n^{ij})_{new} = (w_n^{ij})_{old} \quad - \quad eta.(\frac{dE_n^A}{dw_n^{ij}}) \qquad \text{(equation [vi])}$$

where eta is the "learning rate".

The learning rate is slowly decreased during training, as the neural network learns, so as to allow the weights to converge to some final value. At present, we start with a learning rate of 0.0005 and multiply the current learning rate by a factor of half, which results in a continuously decreasing learning rate.

In his "Efficient BackProp" article, Dr. LeCun [5] proposes a second order technique that he calls the "stochastic diagonal Levenberg-Marquardt method". According to his comparisons, he concludes that for stochastic diagonal Levenberg-Marquardt] convergence is about three times faster than a carefully tuned stochastic gradient algorithm. So in keeping with the advice our model implements this technique through propagating second order derivative (Hessian matrix) for maximised error. After implementing Diagonal Hessian approach, number of epochs required for weights to converge has reduced considerably.

## 5 Implementation details

The implementation of the project involves use of matlab and C++. Initially implementation in matlab involves getting frames out of video dataset then applying bounding box algorithm [3] on each frame i.e. focussing on the area of interest in each frame which is basically surrounding the person in Weizmann dataset. The background is then subtracted to obtain silhouetted dataset of frames. These pixel values of all the frames per video are then written into different text files in order to maintain randomness in training and testing.

Now 3-D Convolutional neural network is implemented in C++. The code is basically implemented in command line which automatically reads input values from the text files. After each epoch , individual weights of all kernels are saved into a text file and they get automatically updated after training over each epoch. This implementation is generalised one in which we can easily vary the number of feature maps in each layer and number of layers. Moreover this implementation can be evaluated on any dataset with some minor changes.

## 6 Experiment on Weizmann dataset

[3]Refer: Ankit Gupta - http://sites.google.com/site/ankit0370/

We trained and tested the extended 3-D CNN on the standard WEIZMANN dataset [4]. It contains 90 video clips from 9 different subjects. Again, each video clip contains one subject performing a single action. There are 10 different action categories: walking, running, jumping, gallop sideways, bending, one-hand-waving, two-hands waving, jumping in place, jumping jack, and skipping. This dataset is then divided into total 226 subsequences containing 10 actions performed by different persons in different context. We applied cross-validation on this dataset with training over 181 sequences and testing over 45 sequences containing all the classifications.

Now each subsequence is passed as sequence of 13 frames (64 x 48 x 13 as input) with 12 frame overlap. In this way, 3-D ConvNet is trained to extract spatio-temporal features even for partial actions i.e. any continuous sequence of frames which is multiple of 13.

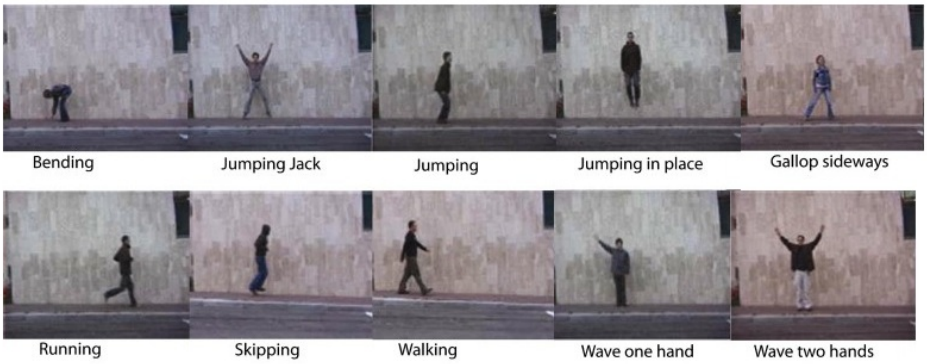



Figure-3: Different Actions performed in the Weizmann Dataset
[CREDIT: WEIZMANN dataset[4][4]]

## Experimental Results

The 3-D CNN model is trained over training dataset for several epochs. The weights appeared to converge after 18 epochs, after which the accuracy became almost constant. The variation of error with number of epochs is depicted in the following graph.

[4]Available at: http://www.wisdom.weizmann.ac.il/ vision/SpaceTimeActions.html

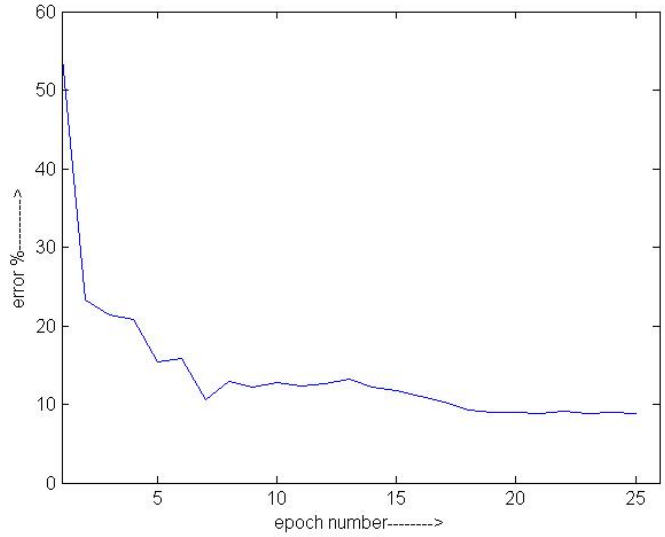



Figure-4: Graph showing variation of error v/s no. of epochs trained.

| Dataset | Config1 | Config2 | Config3 | Config4 | Average(%) |
|---|---|---|---|---|---|
| Weizmann | 91.11 | 88.8 | 93.3 | 91.11 | 91.07 |

Table : Summary of Results for different configurations

Fully trained 3D Convolutional Neural Network when tested on WEIZMANN Dataset(divided into 181 training videos and 45 testing videos) gives an accuracy of 91.07% for 10 classifications.

| Dataset on which accuracy measured | Accuracy(%) |
|---|---|
| Accuracy over videos(Voting) | 91.07 |
| Accuracy over subsequences of 13 consecutive frames | 88.26 |

These results are comparable to the model proposed in M. Baccouche [1] which was evauated on KTH dataset containing 6 classifications with accuracy of 91.04%.

## Confusion matrix

Confusion Matrix depicting the mis-classifications has been shown in following table-

| Recognized | Bend | Jack | Jump | Pjump | Run | Side | Skip | Walk | Wave1 | Wave2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Actual** | | | | | | | | | | |
| **Bend** | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Jack** | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Jump** | 0 | 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Pjump** | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Run** | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| **Side** | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| **Skip** | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 |
| **Walk** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 |
| **Wave1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| **Wave2** | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 |

Confusion matrix: Rows correspond to actual action(label) and columns correspond to the action recognized.

## 7 Conclusion

We developed a 3D CNN model for human action recognition on WEIZMANN Dataset. Our model learns and extracts both spatial and temporal features by performing 3D convolutions. The developed deep architecture extracts multiple channels of information from adjacent input frames and then performs convolution and sub-sampling separately in each channel. The final feature representation is computed by combining information from all channels. We use Multilayer Perceptron classifier to classify these feature representations. According to M. Baccouche [1], learned feature maps in 3-D CNN seem to capture visually relevant information (person/background segmentation, limbs involved during the action, edge information. . . ).
This fully automated learning model produces accuracy that is comparable to the recent works in this field.

## 8 Further Work

In future we would like to use LSTM (Long Short Term Memory) as a classifier for the spatio-temporal features extracted from the 3D Convolution .To improve the accuracy, neuron layer N1 can be replaced with one layer of LSTM (Long Short Term Memory) blocks between layer C3 and N2(output) layer. The main challenge in doing this is incorporating a single learning algorithm for both Convolution and LSTM. We would also like to test our

model on more recent datasets like- YouTube Action Dataset, Hollywood-2 Dataset and LIRIS Human Dataset.

## References


[1] Baccouche M., Mamalet F., Wolf C., Garcia C., Baskurt A. : "Sequential Deep Learning for Human Action Recognition". In : Salah A.A., Lepri B. (eds.) HBU 2011. LNCS, vol. 7065, pp. 2939. Springer, Heidelberg [2011].

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition" Proceedings of the IEEE, v. 86, pp. 2278-2324, [1998].

[3] S. Ji, W. Xu, M. Yang, and K. Yu. "3D convolutional neural networks for human action recognition". In ICML, 3362, 3366 [2010].

[4] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. PAMI, 29(12):22472253, December [2007].

[5] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller. "Efficient backprop". In G. B. Orr and K.-R. Muller, editors, Neural Networks: Tricks of the Trade, pages 950. Springer-Verlag, [1998].