VideoAttentionCapsules: Matrix EM Capsules for Video Action Recognition

Shoaib Ahmed

Advisor: **Dr. Shyju Wilson**

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati

This dissertation is submitted for the degree of
Bachelors of Technology

IIIT Guwahati April 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Shoaib Ahmed
April 2019

Acknowledgements

I would like to express my deep gratitude to Dr. Shyju Wilson, my research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Dr. Dip Sankar Banerjee, for his help in providing and setting up the GPU server.

Abstract

Video-based human action recognition has become one of the most popular research areas in the field of computer vision and pattern recognition in recent years. It has a wide variety of applications such as surveillance, robotics, health care, video searching, and human-computer interaction. Therefore, the problem of Human Action Recognition becomes a very crucial step in the process of engineering Visual Intelligence in machines and Artificial Intelligence in general. Existing techniques rely heavily on 2D and 3D CNNs. Although highly accurate on current datasets, CNN-based techniques lack a sense of a spatial-hierarchical relationship between parts and the whole. Capsules, on the other hand,model the spatial-hierarchical relationships between lower level features and higher level features i.e not only do they check for the presence of features but also check if the lower level features are properly aligned with respect to higher level features. A recent type of Capsules: Matrix Capsules capture the viewpoint-invariant relation between parts and their wholes(lower level features and higher level features), which means that the model has the capability to handle viewpoint changes in the video implicitly. We propose a new architecture for Action Recognition in Videos, called VideoAttentionCapsules, that extends the viewpoint-equivariant properties of Matrix Capsules from Images to Videos. The architecture also uses Soft Attention Models to help achieve better accuracy and faster training time by helping us focus on the regions in the frame where the action is actually happening. We implement our model on the KTH Action Recognition dataset.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

CNN Convolutional Neural Networks

ConvNets Convolutional neural Networks

DBN Deep belief networks

DNN Deep neural networks

LRCN Long-term Recurrent Convolutional Networks

LSTM Long short-term Memory

RGB Red Green Blue

SOTA State-of-the-art

Chapter 1

Introduction

Recognizing human actions from a video stream is a challenging task and has received significant attention from the computer vision research community recently[23]. Analyzing a human action is not merely a matter of presenting patterns of motion of different parts of the body, rather, it is also a description of a person's intention, emotion, and thoughts. Hence, it has become a crucial component in human behavior analysis and understanding, which are essential in various domains including surveillance, robotics, health care, video searching, human-computer interaction, etc. Different from still image classification, video data contains temporal information which plays an important role in action recognition. There are many challenges involved in human action recognition in videos, such as cluttered backgrounds, occlusions, viewpoint variation, execution rate, and camera motion.

Among the early state-of-the-art approaches[13][21][18] for human action recognition, all of these investigations use motion and texture descriptors calculated based on the spatio-temporal interest points, which are built manually. Subsequently, they compute features from raw video frames and classifiers are trained based on the features obtained. Thus, even the features can be fully extracted automatically, and these hand-crafted features are used for specific problems. Therefore, the main drawback of these approaches is that they are problem-dependent, which is challenging to apply in the real world, even though they may achieve high performance in action recognition.

Over the last few years, deep learning-based approaches[2][1][19] have become a very popular solution to the video-based human action recognition problem as they have the ability to learn features from multiple layers hierarchically and build a high-level representation of the raw inputs automatically. Therefore, unlike traditional approaches, the feature extraction process is fully automated. The main advantage of deep learning approaches compared with traditional approaches is their ability to recognize high-level activities with complex structures. Hence, researchers prefer to use deep learning methods to incorporate

the representation of features, such as time-space interest points, frequency, local descriptors and body modeling from video, depth video or RGB video datasets. The promising performance, robustness in feature extraction and the generalization capabilities of deep learning approaches are the major reason behind their increasing popularity and success. Almost all of the popular deep learning methods employed today for the task of Image/Video classification are CNN-based. In essence, CNNs work by detecting the presence of features in an image, and using the knowledge of these features to predict whether an object exists. However, CNNs only detect the existence of features. Therefore, an image that contains a misplaced eye, nose, and ear, would be considered a face because it consists of all the necessary features. This problem arises due to the operation of max pooling, wherein the spatial information about the detected feature is lost. Capsule Networks attempt to address this problem of CNNs by introducing the concept of part-whole relationships. The capsules in Capsule Networks essentially try to capture the spatial position, orientation, and other information about a feature. In this paper, we use this model and try to extend this technique to videos, with a final aim of Human Action Recognition.

Attention is defined as the “active direction of the mind to an object". The word describes the mind’s ability to allocate consideration unevenly across a field of sensation, thought and proprioception, to focus and bring certain inputs to the fore while ignoring or diminishing the importance of others. We use the Visual Soft Attention Model to reduce network size and improve accuracy as Matrix Capsule Networks can usually be memory-consuming and training-time heavy.

The next chapter outlines the related work done in the field of Action Recognition, Capsule Networks, and Attention Networks. Chapter 3 discusses the Proposed Capsule-based architecture for the purpose of Action Recognition in Videos. Chapter 4 describes the experimental setup and results of the implementation of VideoAttentionCapsules. We conclude with Discussions and Future Work in Chapter 5.

Chapter 2

Related Work

Over the years, there has been a shift in the architectures for video-based human action recognition, from pre-deep learning architectures that used hand-crafted feature extraction techniques to deep learning techniques that perform the action recognition task end-to-end. Some of the deep learning based architectures work on frames extracted from videos. Some other techniques[19] use an additional stream of Optical Flow images to model motion features from frames. While this technique performs reasonably well, it fails to extract and model continuous temporal data from videos. To overcome this, 3DConvNets[5] were introduced, which basically used 3-dimensional kernels to perform convolution on a contiguous stack of frames in all three dimensions - Time, Height and Width. The number of trainable parameters and the training time of 3DConvNets increased exponentially as compared to 2D CNNs. The state-of-the-art architecture, i3D[1], uses inflated 3D Inception modules that have been pre-trained on the Kinetics-600[7].

2.1 LRCN

The high performance of image classification networks makes it appealing to try to reuse them with as minimal change as possible for video. This can be achieved by using them to extract features independently from each frame then pooling their predictions across the whole video. This is in the spirit of bag of words image modeling approaches [12]; but while convenient in practice, it has the issue of entirely ignoring temporal structure (e.g. models can't potentially distinguish opening from closing a door).
In theory, a more satisfying approach is to add a recurrent layer to the model, such as an LSTM, which can encode state, and capture temporal ordering and long range dependencies.

T individual frames are inputs into T convolutional networks which are then connected to a single-layer LSTM with 256 hidden units. The authors use convolutional networks

which learn filters based on a stack of N input frames. The LRCN[2]- Long-term Recurrent Convolutional Networks predicts the video class at each time step and these predictions are averaged for the final classification. At test time, they extract 16 frame clips with a stride of 8 frames from each video and average across clips.

The net is pretrained on the 1.2M image ILSVRC-2012 classification training subset of the ImageNet[16] dataset, giving the network a strong initialization to facilitate faster training and prevent over-fitting to the relatively small video datasets. Each frame is first classified by the convolutional network and then the LSTM does classification based on average scores across all frames. The LRCN shows clear improvement over the baseline single-frame system [6] and approaches the accuracy achieved by other deep models.

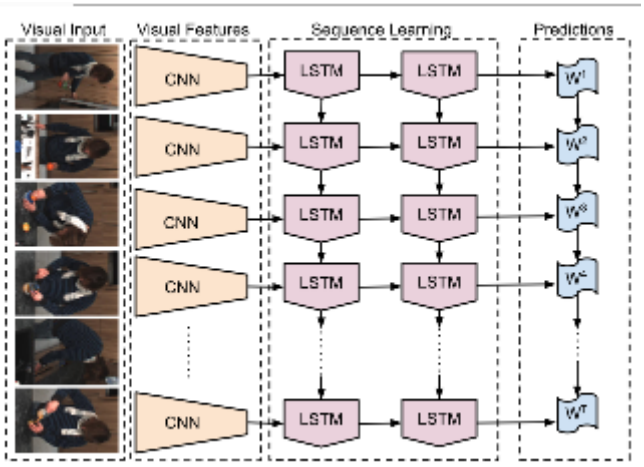

Fig. 2.1 LRCN (LSTM + CNN)

2.2 Two-Stream Networks

LSTMs on features from the last layers of ConvNets can model high-level variation, but may not be able to capture fine low-level motion which is critical in many cases. It is also expensive to train as it requires unrolling the network through multiple frames for backpropagation-through-time. A different, very practical approach[19], introduced by Simonyan and Zisserman, models short temporal snapshots of videos by averaging the predictions from a single RGB frame and a stack of 10 externally computed optical flow frames, after passing them through two replicas of an ImageNet pre-trained ConvNet.

Video can naturally be decomposed into spatial and temporal components. The spatial part, in the form of individual frame appearance, carries information about scenes and objects depicted in the video. The temporal part, in the form of motion across the frames, conveys

the movement of the observer (the camera) and the objects. The authors devise their video recognition architecture accordingly, dividing it into two streams, as shown in Fig. 3. Each stream is implemented using a deep ConvNet, softmax scores of which are combined by late fusion.

The flow stream has an adapted input convolutional layer with twice as many input channels as flow frames (because flow has two channels, horizontal and vertical), and at test time multiple snapshots are sampled from the video and the action prediction is averaged. This was shown to get very high performance on existing benchmarks, while being very efficient to train and test.

The static appearance of the frame in the spatial stream by itself is a useful clue, since some actions are strongly associated with particular objects. Unlike the ConvNet models used in Spatial stream, where the input is a single frame, the input to the temporal stream is formed by stacking optical flow displacement fields between several consecutive frames. Such input explicitly describes the motion between video frames, which makes the recognition easier, as the network does not need to estimate motion implicitly

A recent extension[3] fuses the spatial and flow streams after the last network convolutional layer, showing some improvement on HMDB[9] while requiring less test time augmentation (snapshot sampling).

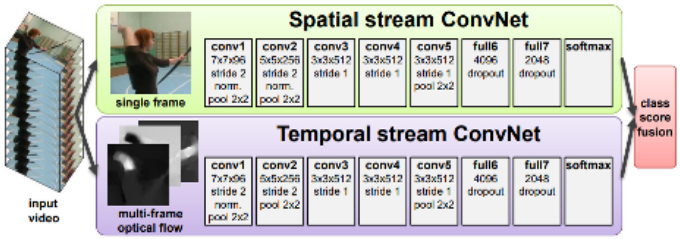

Fig. 2.2 Two-stream architecture for video classification

2.3 3D ConvNets

As a class of deep models for feature construction, CNNs have been primarily applied on 2D images. This model can simply be extended to videos by treating individual video frames as images and performing 2D convolution on them. However, such an approach does not consider the motion information encoded in multiple contiguous frames. To effectively incorporate the motion information in video analysis, this architecture performs 3D convolution in the convolutional layers of CNNs so that discriminative features along both the spatial and the temporal dimensions are captured.

The 3D convolution[5] 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 is connected to multiple contiguous frames in the previous layer, thereby capturing motion information. Formally, the value at position (x,y,z) on the jth feature map in the ith layer is given by:

$$v_{ij}^{xyz} = tanh(b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} \sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}) \tag{2.1}$$

where R_i is the size of the 3D kernel along the temporal dimension, w_{ijm}^{pqr} is the (p,q,r)th value of the kernel connected to the mth feature map in the previous layer.

One issue with these models is that they have many more parameters than 2D ConvNets because of the additional kernel dimension, and this makes them harder to train. Also, they seem to preclude the benefits of ImageNet pre-training, and consequently previous work has defined relatively shallow custom architectures and trained them from scratch.

The inputs to 3D CNN models are limited to a small number of contiguous video frames due to the increased number of trainable parameters as the size of input window increases. On the other hand, many human actions span a number of frames. Hence, it is desirable to encode high-level motion information into the 3D CNN models. To this end, motion features are computed from a large number of frames and are used to regularize the 3D CNN models by using these motion features as auxiliary outputs. We encourage the CNN to learn a feature vector close to the feature generated by the auxilliary output by connecting a number of auxiliary output units to the last hidden layer of CNN.

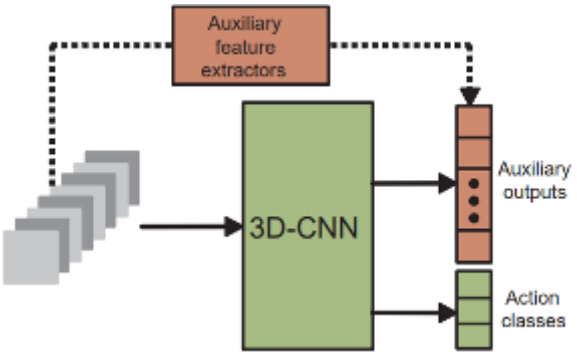

Fig. 2.3 Regularized 3D CNN

2.4 Two-Stream Inflated 3D ConvNets (i3D)

With this architecture, we see how 3D ConvNets can benefit from ImageNet 2D ConvNet architectures and, optionally, from their learned parameters. A two-stream configuration is also adopted here. While 3D ConvNets can directly learn about temporal patterns from an RGB stream, their performance can still be greatly improved by including an optical-flow stream. This method is the current state-of-the-art technique for video action recognition and has achieved SOTA results on UCF-101[20], Kinetics-400 and HMDB-51.

Inflating : A number of very successful image classification architectures have been developed over the years, in part through painstaking trial and error. Instead of repeating the process for spatio-temporal models it is proposed to simply convert successful image (2D) classification models into 3D ConvNets. This can be done by starting with a 2D architecture, and *inflating* all the filters and pooling kernels – endowing them with an additional temporal dimension. Filters are typically square and we just make them cubic – $N \times N$ filters become $N \times N \times N$.

Bootstrapping 3D filters from 2D Filters: Besides the architecture, one may also want to bootstrap parameters from the pre-trained ImageNet models. To do this, we observe that an image can be converted into a (boring) video by copying it repeatedly into a video sequence. The 3D models can then be implicitly pre-trained on ImageNet, by satisfying what we call the boring-video fixed point: the pooled activations on a boring video should be the same as on the original single-image input. This can be achieved, thanks to linearity, by repeating the weights of the 2D filters N times along the time dimension, and re-scaling them by dividing by N. This ensures that the convolutional filter response is the same. Since the outputs of convolutional layers for boring videos are constant in time, the outputs of pointwise non-linearity layers and average and max-pooling layers are the same as for the 2D case, and hence the overall network response respects the boring-video fixed point.

Two 3D Streams: While a 3D ConvNet should be able to learn motion features from RGB inputs directly, it still performs pure feedforward computation, whereas optical flow algorithms are in some sense recurrent (e.g. they perform iterative optimization for the flow fields). Perhaps because of this lack of recurrence, experimentally it was still valuable to have a two-stream configuration with one i3D network trained on RGB inputs, and another on flow inputs which carry optimized, smooth flow information. The two networks are trained separately and averaged their predictions at test time.

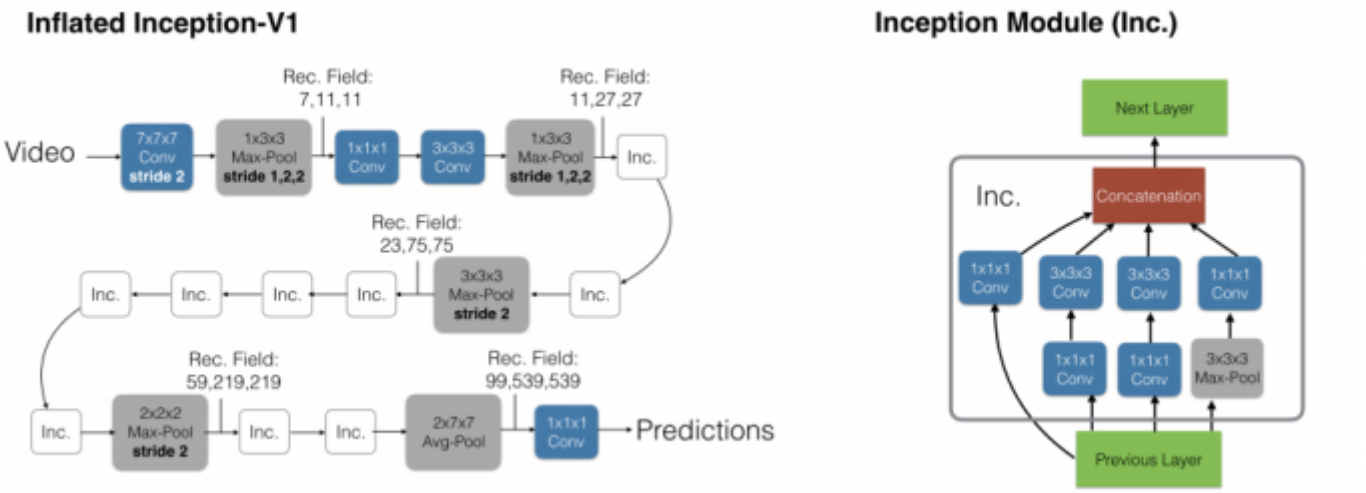

Fig. 2.4 Inception-V1 module used in i3D

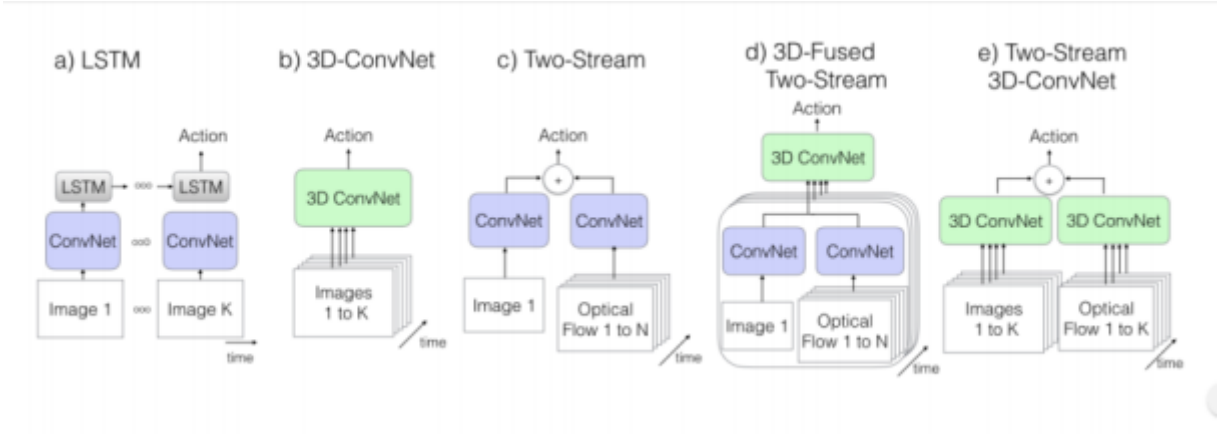

Fig. 2.5 Video architectures considered

Table 2.1 Comparison of various architectures present in literature

Models	UCF-101 Accuracy
Two-Stream	88.0
IDT	86.4
3D CNN + Sports 1M pre-training	82.3
3D CNN ensemle + IDT + Sports 1M pre-training	90.1
Two-Stream I3D + Kinetics pre-training	97.8
RGB I3D + ImageNet+Kinetics pre-training	95.6
Flow I3D + ImageNet+Kinetics pre-training	96.7
Two-Stream I3D + ImageNet+Kinetics pre-training	**98.0**

2.5 Matrix Capsules

A capsule[17][4] is a group of neurons whose outputs represent different properties of the same entity. Each layer in a capsule network contains many capsules. Matrix Capsule Network[4] is a version of capsules in which each capsule has a logistic unit to represent the presence of an entity and a 4x4 matrix which could learn to represent the relationship between that entity and the viewer (the pose), called the **pose matrix**. A capsule in one layer votes for the pose matrix of many different capsules in the layer above by multiplying its own pose matrix by trainable viewpoint-invariant transformation matrices that could learn to represent part-whole relationships. Each of these votes is weighted by an assignment coefficient. These coefficients are iteratively updated for each image using the Expectation-Maximization algorithm such that the output of each capsule is routed to a capsule in the layer above that receives a cluster of similar votes. The transformation matrices are trained discriminatively by backpropagating through the unrolled iterations of EM between each pair of adjacent capsule layers.

A fast iterative process called “routing-by-agreement”[4] updates the probability with which a part is assigned to a whole based on the proximity of the vote coming from that part to the votes coming from other parts that are assigned to that whole. An important difference between capsules and standard neural nets is that the activation of a capsule is based on a comparison between multiple incoming pose predictions whereas in a standard neural net, it is based on a comparison between a single incoming activity vector and a learned weight vector. Each capsule in the higher-layer corresponds to a Gaussian and the pose of each active capsule in the lower-layer (converted to a vector) corresponds to a data-point (or a fraction of a data-point if the capsule is partially active). For finalizing the pose parameters and activations of the capsules in layer L + 1 the authors run the EM algorithm for few iterations

(normally 3) after the pose parameters and activations have already been finalized in layer L. The non-linearity implemented by a whole capsule layer is a form of cluster finding using the EM algorithm, so it is called EM Routing.

2.5.1 Spread Loss

In order to make the training less sensitive to the initialization and hyper-parameters of the model, the authors use "spread loss" to directly maximize the gap between the activation of the target class a_t and the activation of the other classes. If the activation of a wrong class, a_i , is closer than the margin, m, to at then it is penalized by the squared distance to the margin:

$$L_i = (max(0, m - (a_t - a_i))^2, L = \sum_{i \neq t} L_i \tag{2.2}$$

Spread loss is equivalent to squared Hinge loss with m = 1.

This algorithm has attention in the opposite direction of typical attention mechanisms. The competition is not between the lower-level capsules that a higher-level capsule might attend to. It is between the higher-level capsules that a lower-level capsule might send its vote to.

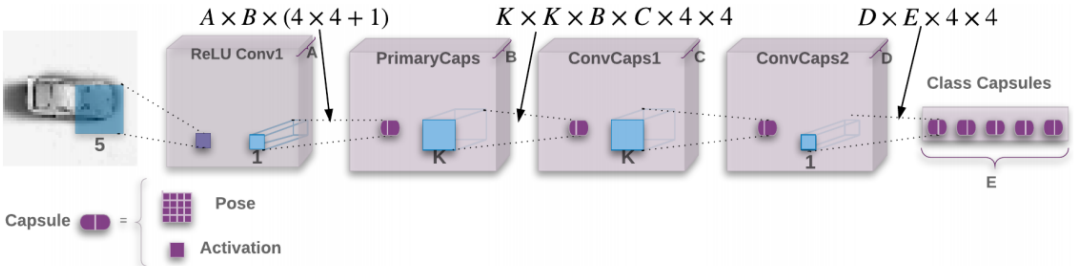

Fig. 2.6 Matrix Capsule Architecture by Hinton et al.

The model starts with a 5x5 convolutional layer with 32 channels (A=32) and a stride of 2 with a ReLU non-linearity. All the other layers are capsule layers starting with the primary capsule layer. The 4x4 pose of each of the B=32 primary capsule types is a learned linear transformation of the output of all the lower-layer ReLUs centered at that location. The primary capsules are followed by two 3x3 convolutional capsule layers (K=3), each with 32 capsule types (C=D=32) with strides of 2 and one, respectively. The last layer of convolutional capsules is connected to the final capsule layer which has one capsule per output class.

2.6 Hierarchical Attention Networks

Human Actions are layered , giving rise to the need for modeling video temporal structure with multiple granularities. A Visual attention model aims to capture the property of human perception mechanism by identifying the interesting regions in the images. Long-short term memory (LSTM), which has the ability to preserve sequence information over time and capture long-term dependencies, has become a very popular model for sequential modeling tasks such as speech recognition, machine translation, and program execution. Recent advances in computer vision also suggest that LSTM has potentials to model videos for action recognition. However, directly applying LSTM cannot capture the property of multiple granularities in human actions. To fully capture the video's temporal structure, a hierarchical LSTM is developed in [22]. An illustration of the hierarchical LSTM is shown in Figure 2. This hierarchical LSTM is composed of two layers – the first layer accepts the appearance feature, extracted using a ConvNet, of each frame as the input and the output of the first layer LSTM is used as the input of the second layer LSTM. To capture the dependencies between different sub-actions, for every k encoded features a feature is skipped as input to the first LSTM layer and is used as the input to the second layer. In addition to capturing the video temporal structure, another advantage of layered LSTM is that it increases the learning capability of LSTMs. By adding another layer in LSTM, LSTMs are made to learn higher level and more complex features, which is a common practice proven to work well in other deep architectures such as CNN, DNN, and DBN.

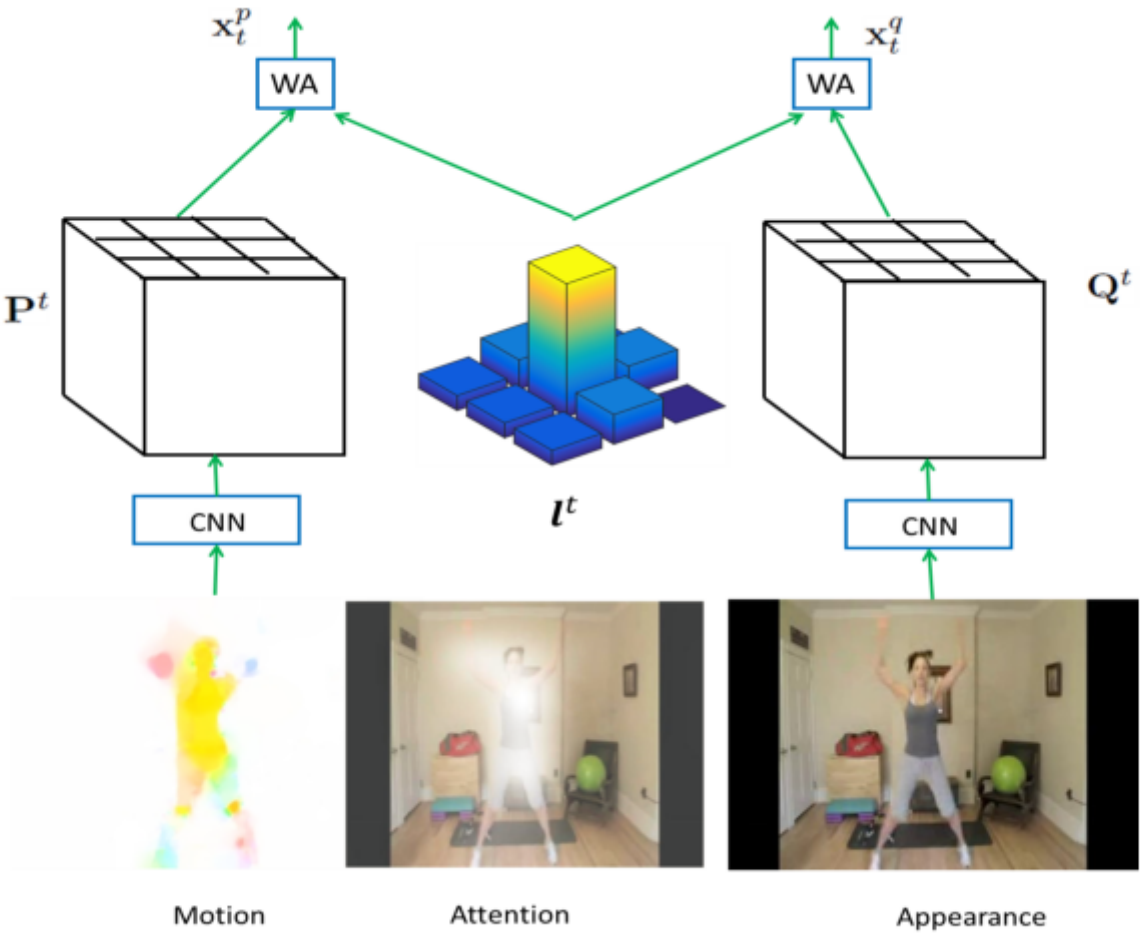

Fig. 2.7 Hierarchical Attention Network

Chapter 3

Proposed Architecture

3.1 VideoAttentionCapsule

Our architecture uses averaged frames (described in Section 3.2.2) extracted from the video at regular intervals as input. The architecture uses N Matrix Capsule Networks, each processing a different averaged frame of the same video. These networks have similar architecture to the one proposed in the original paper. We modify the last layer of each of these networks to output a fixed number of capsules. After having processed all of the N averaged frames, through N distinct networks, each of these predictions are then averaged for the final classification. To reduce the training time of the architecture and boost the accuracy of the model, we employ a soft-attention module that helps the successive networks recognize and focus on the areas in the frame that have motion. The attention weights are obtained from the output of the first Convolutional Capsule layer of a frame's network and are fed as input to the second Convolutional Capsule layer of the next frame's network.

3.2 Experiments and Results

We first build the Matrix Capsule Network using PyTorch[15], which uses the Adam Optimizer and Spread Loss function(Defined in Section 2.5.1).

3.2.1 Preliminary Experiments

The Matrix EM Capsules were proposed for the purpose of image recognition and classification. The proposed architecture was only tested on the MNIST dataset[10] and the smallNORB dataset[11]. Although the smallNORB dataset is a very good starting-point due to its simplistic nature and viewpoint variance, it lacks many of the features present in

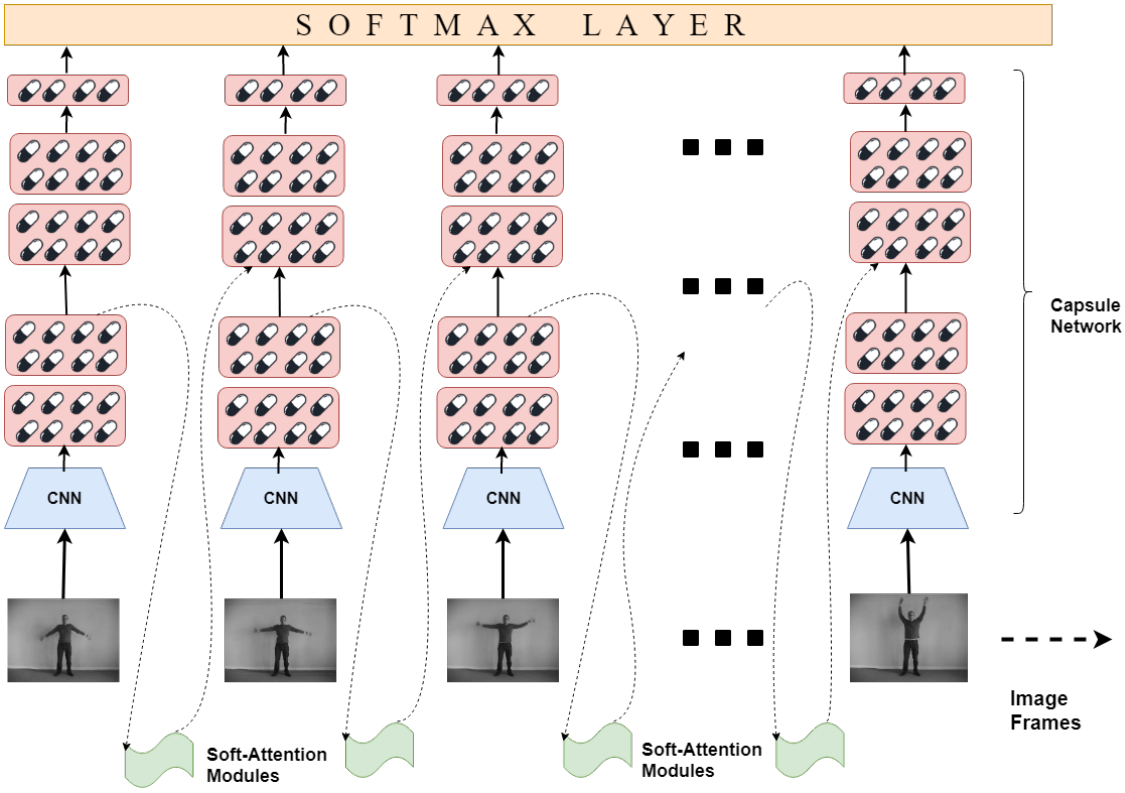

Fig. 3.1 Proposed VideoAttentionCapsule Architecture

images in the wild. To test the performance of this model on typical images in the wild, we run the EM Matrix Capsule Network on a more general image dataset such as the CIFAR-10 [8] by tuning the model parameters to incorporate RGB channels and to enlarge the size of the model. The results obtained from these tests can be seen in the table below.

Table 3.1 Performance of EM Matrix Capsules on CIFAR-10

Model Parameters	**Epochs**	**Accuracy**
A=B=C=D=32	10	62.48
A=64 B=8 C=16 D=16	10	60.17
A=32 B=24 C=32 D=24	35	70.11
A=B=C=D=32	35	71.1600
A=64 B=8 C=16 D=16	35	71.52

The experiments were conducted on Tesla K80. Specific setting is learning rate = 0.01, batch size = 20, weight decay = 0, Optimizer : Adam optimizer, Loss : Spread Loss and Number of EM - itertions = 2.

3.2.2 Dataset

The **KTH Human Action Recognition Dataset** contains six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in four different scenarios: outdoors, outdoors with scale variation, outdoors with different clothes, and indoors. All sequences were taken over homogeneous backgrounds with a static camera with 25fps frame rate. The sequences were downsampled to the spatial resolution of 160x120 pixels and have an average length of four seconds. Due to the relative simplicity of this dataset as compared to other, more complex and larger action recognition datasets, we choose this dataset. The fact that the frames are black and white also reduces the complexity of the network. In addition to this, the actions performed in the videos of the dataset are simple, short and repetitive, making the dataset an ideal choice for preliminary experiments.

Pre-processing : We pre-process the Video frames to obtain a stack of averaged frames that are fed to the VideoAttentionCapsules. We obtain averaged frames by taking a stack of 6 contiguous frames from a certain point in a video and average all the 6 Numpy[14] arrays to a single averaged array. We repeat this process at fixed intervals of time to extract N number of averaged frames from the video. We then stack these extracted averaged frames. Each of

these stacks represent a summary of the corresponding video action. We store these arrays in Numpy files as it is not possible to store images of more than 4 dimensions.

Fig. 3.2 Sample frames of Classes in KTH

3.2.3 Experiment

The pre-processing step of Section 3.2.2 creates numpy files corresponding to each video in the dataset. We now input the corresponding slice of the numpy file to its corresponding Capsule Network from an ensemble of N Capsule Networks (N=5 in our experiments). Each of the Capsule Network in the ensemble produces a 6x1 prediction vector containing negative logits corresponding to each input file. Each of the N networks' last layer is then averaged to produce the final classification.

The training was done on a Tesla K80 using a training batch size of 7, a learning rate of 0.01, 2 EM iterations on 50 epochs. The Loss function used is Spread Loss and the optimizer used is Adam Optimizer. The architecture of the Capsule Network used is A = 64, B = 8, C = 16, D = 16. The testing was done with a batch size of 10. Each epoch of training was completed in approximately 17 minutes.

3.2.4 Results

The best Testing Accuracy achieved was **61.02**. The number of EM iterations used were 2. Increasing the number of iterations to 3 produces poor quality results. The batch size for training was also controlled as larger batch sizes require larger memory. Test batch of 7 required 11GB GPU memory. A decaying learning rate is also undesirable as the Adam Optimizer takes care of the learning rate of our Capsule Networks.

Table 3.2 Performance of VideoAttentionCapsules on KTH Action Recognition Dataset

Model Parameters	**Epochs**	**Test Accuracy**
A=B=C=D=32	30	50.2
A=64 B=8 C=16 D=16	30	55.74
A=32 B=24 C=32 D=24	50	57.2
A=B=C=D=32	50	58.93
A=64 B=8 C=16 D=16	50	61.02

The figures below show the Epochs vs Accuracy Graph and the Epochs vs Testing Loss graph for VideoAttentionCapsules with Parameters : A=64 B=8 C=16 D=16, Epochs = 50, batch size = 6, EM iterations = 2.

Fig. 3.3 Testing Accuracy and Loss for VideoAttentionCapsules

Chapter 4

Discussion

The invention of Capsule Networks is a breath of fresh air in this era of computationally-turgid architectures which feel increasingly similar to much-hated brute-force algorithms. The solid mathematical foundation of Matrix Capsules is something that is highly intriguing. Although the accuracy and scalability of Matrix Capsules are nowhere near the state-of-the-art, it is a robust and mathematically-found architecture that solves the problem of Image Classification. The extension of Matrix EM Capsules to Videos requires a better Ensemble technique than the current averaging method. The pre-processing step that generates the averaged frames lacks in capturing spatio-temporal features of movement between contiguous frames. A better technique to generate averaged frames could be the use of LSTMs to capture the time dependant relations. The performance of VideoAttentionCapsules on KTH Action Recognition Dataset gives approximately the same accuracy as baseline CNN-based models on the same dataset. Although there is a lot of scope for improvement in VideoAttentionCapsules, the results are quite impressive for such a young architecture.

References

[1] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733.

[2] Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*.

[3] Feichtenhofer, C., Pinz, A., and Zisserman, A. (2016). Convolutional two-stream network fusion for video action recognition. *CoRR*, abs/1604.06573.

[4] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with EM routing. In *International Conference on Learning Representations*.

[5] Ji, S., Xu, W., Yang, M., and Yu, K. (2010). 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:221–231.

[6] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classication with convolutional neural networks. In *Proceedings of International Computer Vision and Pattern Recognition (CVPR 2014)*.

[7] Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M., and Zisserman, A. (2017). The kinetics human action video dataset. *CoRR*, abs/1705.06950.

[8] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

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

[10] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[11] LeCun, Y., Huang, F., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2.

[12] Li, Y. and Kuai, Y. (2012). Action recognition based on spatio-temporal interest points. pages 181–185.

[13] Liu, J. and Shah, M. (2008). Learning human actions via information maximization. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[14] Oliphant, T. (2006–). NumPy: A guide to NumPy. USA: Trelgol Publishing. [Online; accessed <today>].

[15] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

[16] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2014). Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575.

[17] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 3859–3869, USA. Curran Associates Inc.

[18] Sadanand, S. and Corso, J. J. (2012). Action bank: A high-level representation of activity in video. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1234–1241. IEEE.

[19] Simonyan, K. and Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 568–576, Cambridge, MA, USA. MIT Press.

[20] Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402.

[21] Wang, H. and Schmid, C. (2013). Action recognition with improved trajectories. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 3551–3558, Washington, DC, USA. IEEE Computer Society.

[22] Wang, Y., Wang, S., Tang, J., O'Hare, N., Chang, Y., and Li, B. (2016). Hierarchical attention network for action recognition in videos. *CoRR*, abs/1607.06416.

[23] Wu, D., Sharma, N., and Blumenstein, M. (2017). Recent advances in video-based human action recognition using deep learning: A review. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2865–2872.

Appendix A

Running the code

The code for VideoAttentionCapsules can be found on Github at :
https://github.com/ahmedshoaib/em_caps_attn_kth
The steps to run the code are :

1. Download/Clone the code from the github repository given above.
2. Download all the KTH action recognition dataset classes from http://www.nada.kth.se/cvap/actions/, unzip and store them in the format :

```
em_caps_attn_kth
|->kth
 |-->walking
  |--->person01_walking_d1_uncomp.avi
  |--->person01_walking_d2_uncomp.avi
  |--->person01_walking_d3_uncomp.avi
 .
 .
 |-->jogging
 .
 ...
```

3. Run the pre-processing script using :

```
python pre_process.py
```

4. This creates a folder named '/processed' which has the processed .numpy files for each video used by the VideoAttentionCapsules.

5. Finally train the VideoAttentionCapsule on the kth dataset by running :

```
python train.py
```

6. Upon completion of training, this creates a file that saves the weights of the best model weights named model_epochs.pth in the /snapshots folder, which can later be used for testing.