Sports Video Action Recognition

A Thesis Submitted in the Partial Fulfilment
of the Requirements for the Degree of

Master of Technology

in

Computer Science

by

Santanu Datta

Under Guidance of

Prof. Kumar Sankar Ray

Electronics and Communications Unit
Indian Statistical Institute, Kolkata
India

June 30,2019

Certificate

This is to certify that the thesis entitled, **Sports Video Action Recognition** and submitted by **Santanu Datta**, Roll No. **CS1706** in partial fulfillment of the requirements of **Master of Technology in Computer Science** embodies the work done by him under my supervision.

Prof. Kumar Sankar Ray
ECSU, ISI Kolkata
Date:

Acknowledgement

I would like to thank my supervisor **Prof. Kumar Sankar Ray** for the kind guidance he has provided throughout the dissertation work.

Abstract

From playing games to driving cars, deep learning has achieved great success in the recent past.In this dissertation, we apply deep learning to recognize sports videos. We have implemented state of the art VGG3D model on different challenging state of the art video datasets. In this paper , we communicate our findings.

Contents

Chapter 1

Introduction

From the advent of computer, researchers have always wondered about making it intelligent so that it can do our work. Over the past few decades, artifical intelligence was a interesting topic and many activities have been tried to teach the computer.From winning chess against grandmaster Garry Kasparov to answering questions, artificial intelligence showed a way to fulfilling the dream. But due to lack of computational power and lack of data, it was not being used in much in real life scenario.

In the last 20 years, internet era and progress in computational technologies broke those barriers. Now terabytes of data is being generated everyday and computational facilities such as GPU computing, Cloud computing are available to researchers.This encouraged researchers to apply deep learning, a section of artificial intelligence to real world problems. Within a few years, deep learning based algorithms showed immense success in most of the Machine learning tasks. Specially in computer vision, deep neural network based algorithms won the prestigious **Imagenet** competition. Not only in image recognition, segmentation, localization, deep learning showed promising results in other domains also. In this thesis, we apply deep learning in videos, and we show how it is providing good results to a challenging video action recognition task.

1.1 What is action recognition?

Action recognition is a computer vision task involves the identification of different actions from video clips (a sequence of 2D frames) where the action may or may not be performed throughout the entire duration of the video.

Action recognition is a important topic having a great many benefits.Sports action recognition can help us build a software that automatically recognizes an uploaded sports video and index it so that it will come up during appropriate query.
Though it seems similar to image recognition task, over the years image recognition has achieved immense success, while video action recognition is not.Some of the difficulties are :

- Huge Computational Cost A simple convolution 2D net for classifying 101 classes has just approx 5M parameters whereas the same architecture when inflated to a 3D structure results in approx 33M parameters.
- Capturing long context Action recognition involves capturing spatio temporal context across frames. Additionally, the spatial information captured has to be compensated for camera movement.
- Designing classification architectures Designing architectures that can capture spatiotemporal information involve multiple options which are non-trivial and expensive to evaluate.

1.2 Objective

Our objective is to develop a deep neural network architecture than can recognize a given sports video in one of the given classes. To show the robustness of the network, we will train and test the architecture on several standard dataset. At the end, we compare our findings with other techniques.We also conduct some analysis to explain our findings.

1.3 Outline

In the next chapter we briefly go through the topics of deep learning we will be using in our thesis. In chapter 3, we present a detailed presentation of the architecture we are using. In the subsequent chapter, we describe the datasets that we are using. Chapter 5 comprises of implementation details. Chapter 6 conveys the results that have been found by us. In the next chapter, we compare our finding to other works. Lastly, in chapter 8, we conclude the thesis.

Chapter 2

Deep Learning

We provide brief introduction to deep learning.A good resource is the book written by Goodfellow et al [2].This will be helpful to understand the model architecture. It will also explain the reason we choose the architecture.

2.1 Perceptron

Perceptron [6] was the simplest model of neural network.It was proposed by Minsky and Papert in 1969.It consists of only one computational neuron. It takes inputs $x_1, x_2, \ldots, x_n$ with labels $0, 1$ and outputs y which is a function of weighted sum of inputs. The goal is to learn the weights so that it can classify them accurately. Notice that perceptron model can correctly classify only the datapoints that are linearly separable.

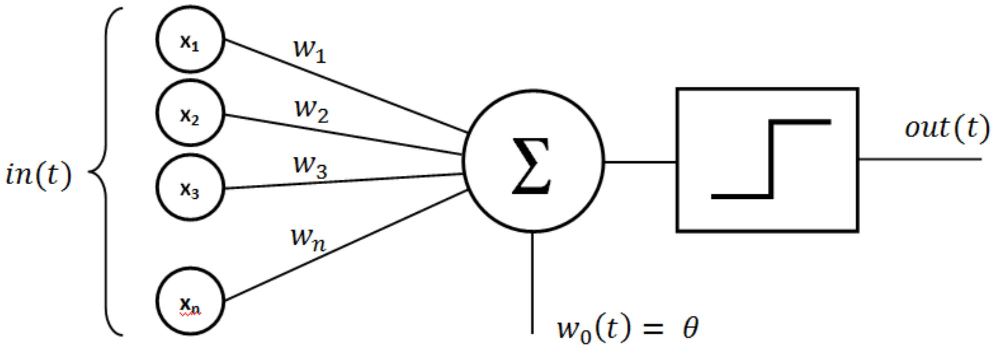
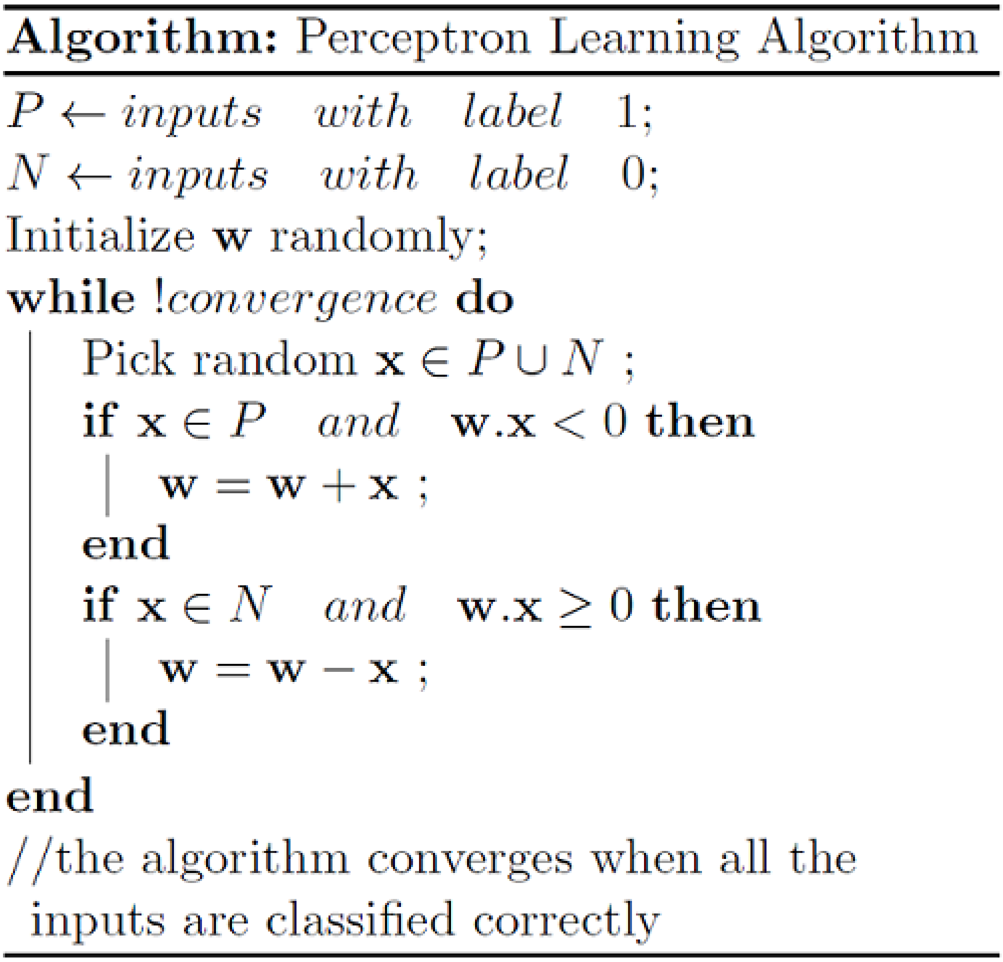


Figure 2.1: Perceptron Model

The perceptron weights are learned via the following algorithm:



```
Algorithm: Perceptron Learning Algorithm
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

Figure 2.2: Learning Algorithm

2.2 Multi Layer Perceptron

It was noticed in the same article [6] that perceptron cannot even learn XOR.So, in search of more advanced architecture, multilayer perceptron model(MLP), or which we know by the name of neural networks, was found. The main principle is backpropagation algorithm, which was discovered by Geofrey Hinton in 1986.

The main idea is that the input goes through a multiple layers of neurons and provides an output. Then there is a loss function which calculates the error. The error is then backpropagated to the neurons where weights are adjusted using gradient descent update rule. This whole process is called one epoch. The algorithm stops when error is within predefined tolerance

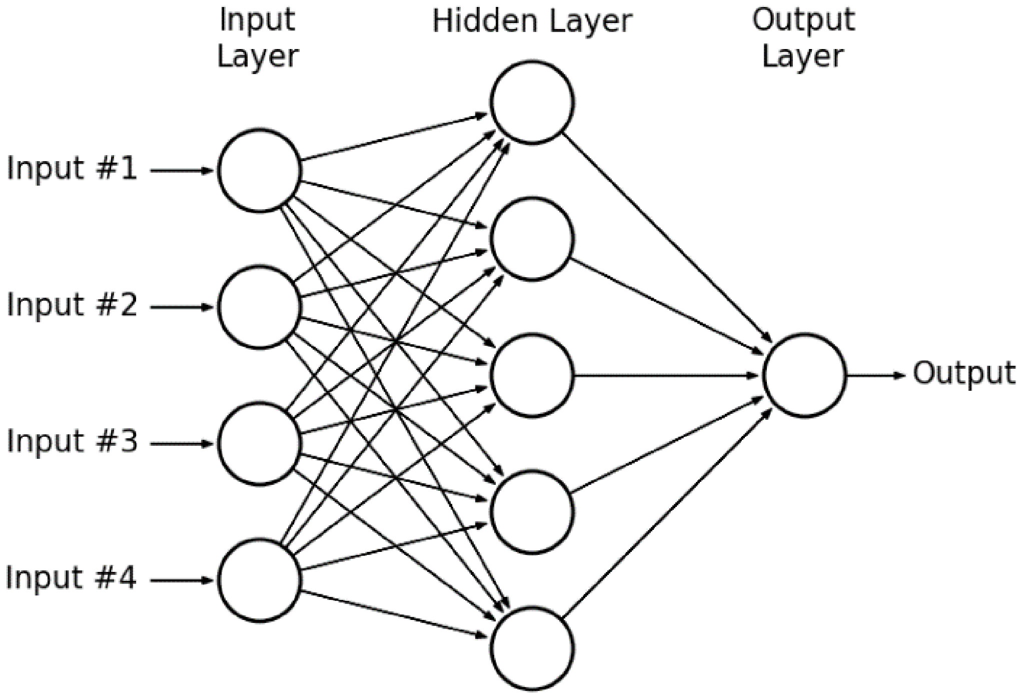


Figure 2.3: MLP

level or a predefined number of epochs has been passed or the network is has stopped learning.

2.3 Deep Neural Network

By the discovery of the **Universal Approximation Theorem** [3], it was shown that any given function can be approximated by neural network with sufficient number of neurons.This encouraged the researchers to go for more complicated networks. The layers between input layer and output layer are called hidden layers in MLP. When the number of layers are large, the network is called **deep neural network**.

2.3.1 Limitations

The main limitation was the requirement of huge computational resource needed to train those network.

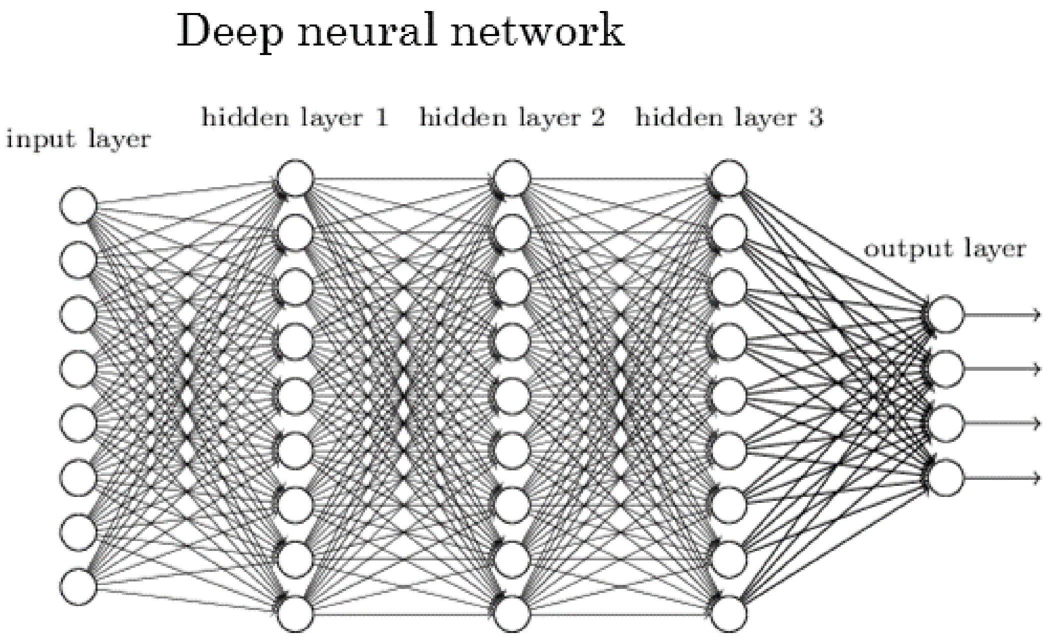


Figure 2.4: Deep neural Network

2.4 Convolutional Neural Network

The convolutional neural networks was invented to solve the problem. Th idea is to use multiple filters and convolve with the input to learn representations of data capturing the underlying principle. The convolutional neural network has two advantages :

- **Parameter Sharing :** A filter is used over all of the parts of the input. For example, a filter which detects vertical edge can be used in all of the picture to detect vertical edge.
- **Sparsity of Connections :** In each of the layers, a neuron in connected to selected neurons from the previous layer, where in DNN, each neuron is connected to all the neurons in previous layer.

2.5 Dropout

Dropout is a training technique invented by Hinton et al [15]. It works during training as follows :

- Choose a number p between 0 and 1, generally 0.5 is chosen.

- In each layer, p fraction of neurons are randomly chosen and given 0 weight so that they do not take part in learning.
- During test time, dropout is not used but the output of the neurons are multiplied by 1 - p, since it is the expected time that neuron took part in training.

2.5.1 Why dropout ?

Dropout forces the neurons not to rely on other neurons, thus forces to learn the hidden representation. Also dropout implements ensemble of different neural networks without high computational cost. Dropout thus prevents overfitting and gives way to learn.

2.6 Transfer Learning

Transfer learning is the process of using an already learned network to learn a similar task. This is useful in mainly two cases :

Less Computational Resource : The transfer learning technique provides already some expertise to the network in task, which means network needs fewer training to be done.

Less Data : If the data is scarce for the particular task, then using transfer learning, network inherits some of the underlying representations already.

Chapter 3

Architecture

3.1 Developing Architecture

After reading a few research papers regarding video action recognition, we pointed out two main underlying principles :

- Increasing number of layers on CNN, which is one of the main philosophy behind VGGNet [12].
- Using a pretrained model on image dataset(available online).

Since , we also have computational constraints and storage limitations, we decided to use an architecture which enjoys the advantages of transfer learning. We avoided heavy computation based algorithms such as incorporating optical flow. Also, we wanted the main underlying principle behind the architecture to be simple, so we have avoided LSTM or RNN based algorithms for now.

Based on those underlying principles, we decided to go with the following architecture [4].

3.2 Architecture Description

The architecture can be divided into 3 parts.

3.2.1 VGG16

VGG is the model developed by Karen et al [12]. The architecture of the VGG model is a specific combination of convolutional layers, fully connected

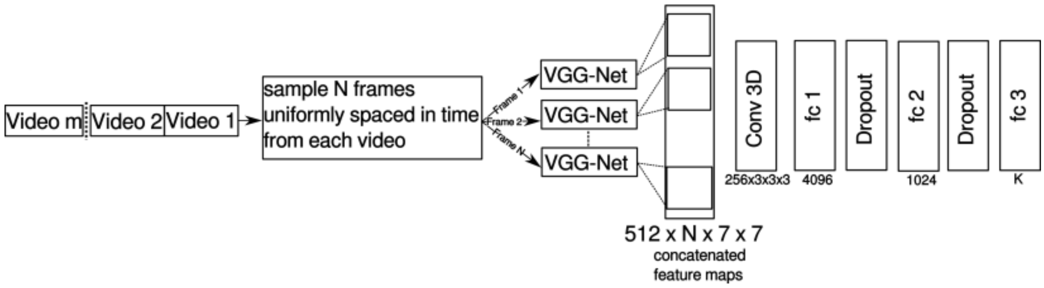


Figure 3.1: VGG3D

layers. This is the architecture of VGG :

This is the first part of the architecture. We feed extracted frame to VGG16 model. We remove the last 7 layers of VGG. The reason is that after passing through this modified VGG we will get a representation of the image as a vector.

3.2.2 Concatenation

In this step, we concatenate all the frames representation vector together. This concatenated vector represents one video to the last part of the deep neural network.

3.2.3 FC Layers

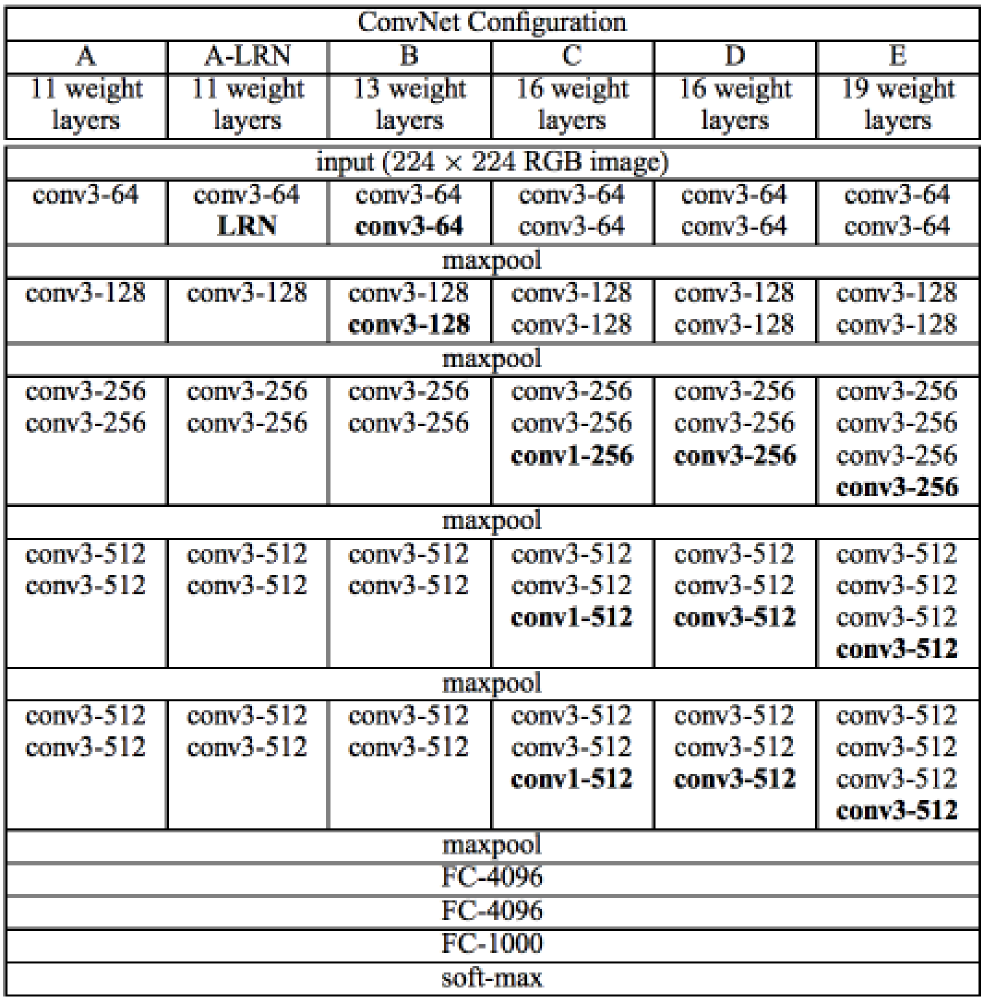
In the third stage, the architecture contains a series of convolution layer, two fully connected layer each followed by dropout. Finally, there is a fully connected layer of size K for multiclass classification.

3.3 Architecture Methodology

The architecture works as follows :

- Take a video.
- Sample N frames from it.
- Feed them through different vgg16 models and get a representation.
- Concatenate those representations.

- Pass them through conv3D layer of size $256 \times 3 \times 3 \times 3$
- Pass them through fully connected layers of size 4096 and 1024.
- Finally pass through output layer with K nodes, where K is the number of classes.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.2: VGG Architecture

Chapter 4

Datasets

4.1 UCF-101

UCF-101 dataset is an action recognition dataset collected from YouTube. It was developed in University of Central Florida [13]. The dataset contains 13320 videos from 101 action classes, making it quite a large dataset to work with.Not only the action classes are diverse, but also the dataset has large variance in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions etc. So, it is a challenging dataset.

UCF -101 is the base dataset where authors of the architecture trained the network.

4.2 KTH

KTH [11] is an old sports video dataset.The summary of KTH dataset is :

- There are six types of human actions :walking, jogging, running, boxing, hand waving and hand clapping.
- Actions are performed several times by 25 subjects in four different scenarios: outdoors, outdoors with scale variation, outdoors with different clothes and indoors.
- There are 2391 sequences in the database. All sequences were taken over homogeneous backgrounds with a camera with 25fps frame rate.
- The sequences were downsampled to the spatial resolution of 160×120 pixels. The video lengths are four seconds in average.

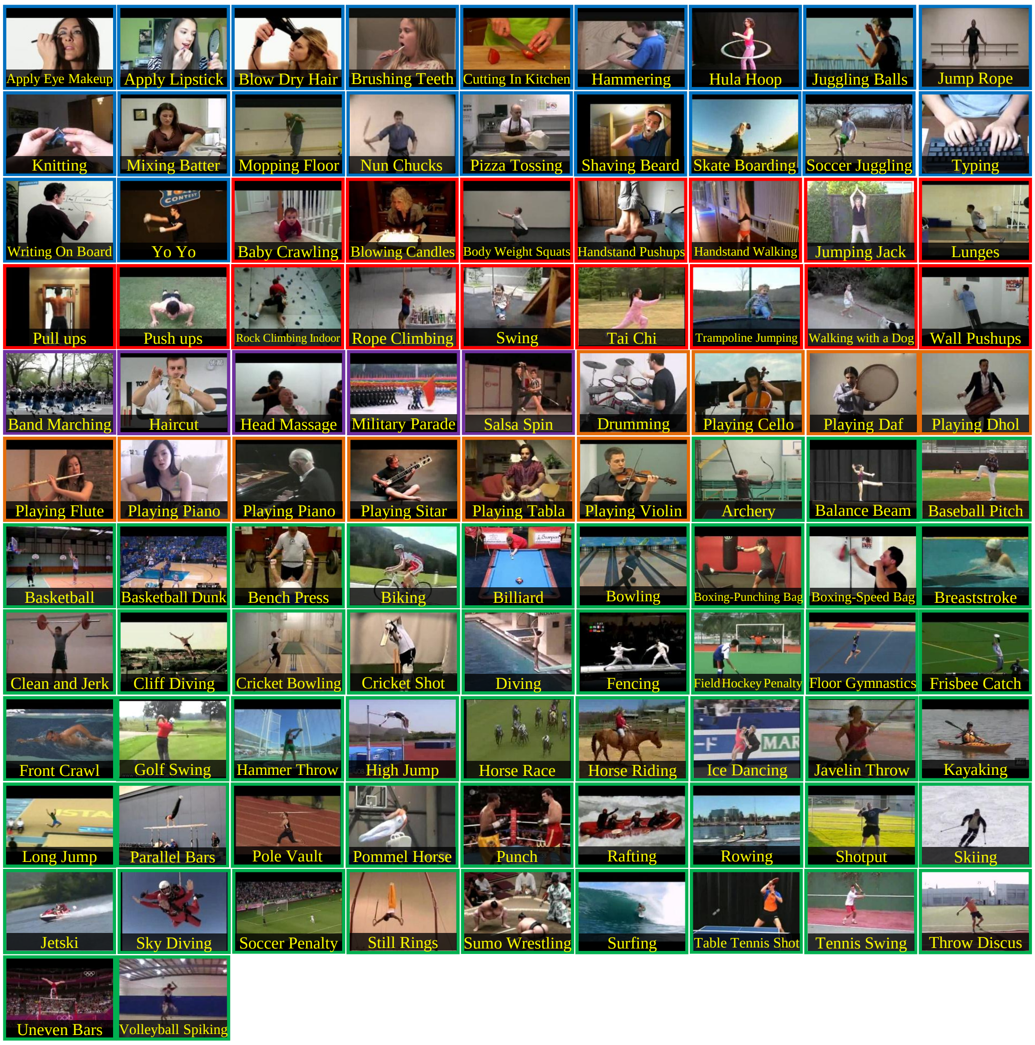
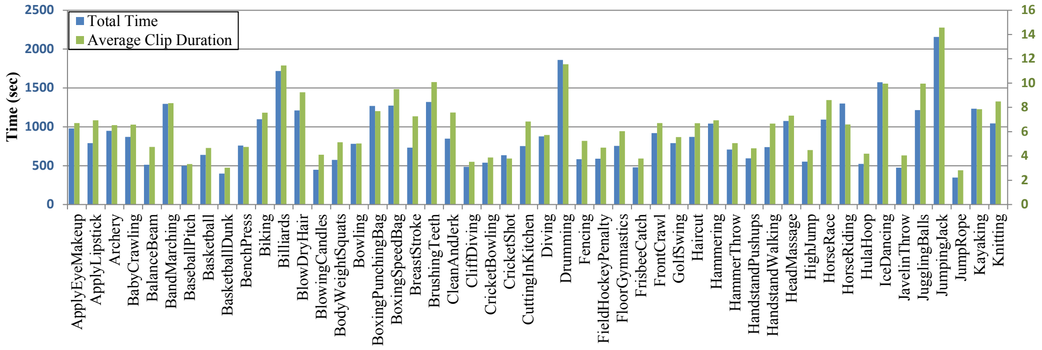
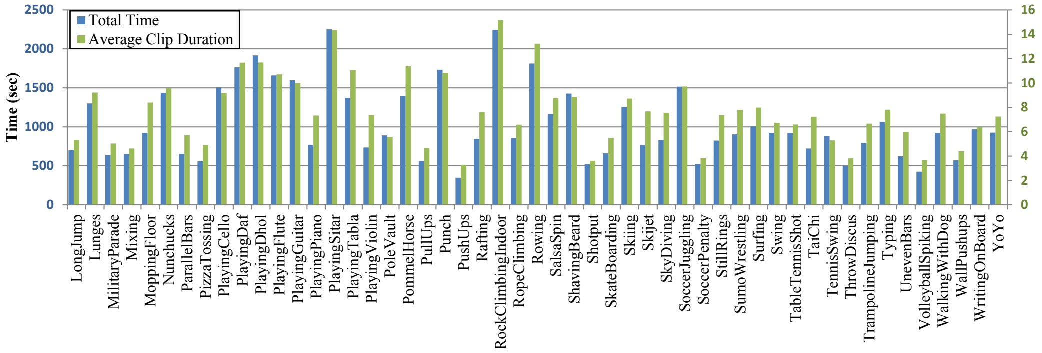
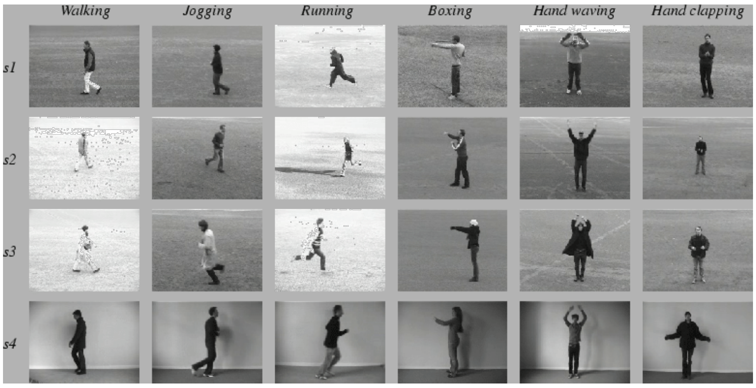


Figure 4.1: UCF-101





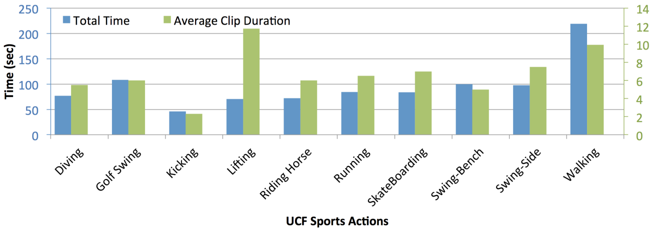


4.3 UCF Sports

UCF Sports dataset [14] [9] has the following features :

- It contains 10 sports action classes.
- The dataset includes a total of 150 sequences with the resolution of 720×480.

Actions	10	Total duration	958 s
Clips	150	Frame rate	10 fps
Mean clip length	6.39 s	Resolution	720 × 480
Min clip length	2.20 s	Max num. of clips per class	22
Max clip length	14.40 s	Min num. of clips per class	6

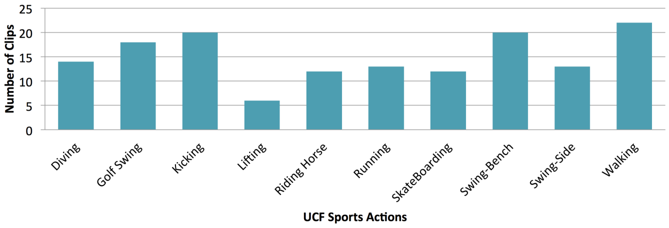


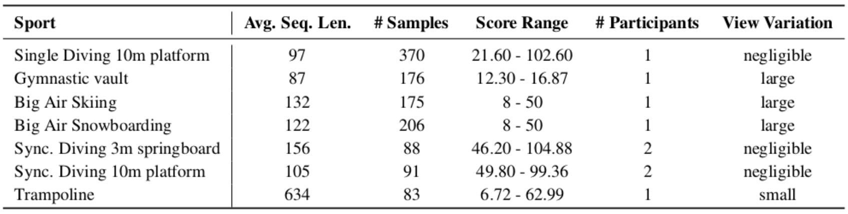
- The collection represents a natural pool of actions featured in a wide range of scenes and viewpoints.
- The dataset has been used for numerous applications such as: action recognition, action localization, and saliency detection.

4.4 Action Quality Assessment

Action quality assessment [7] is yet another useful dataset for sports action recognition.

- This is developed by Real-Time Intelligent Systems (RTIS) Laboratory.
- Contains 7 type of actions : singles diving-10m platform, gymnastic vault, big air skiing, big air snowboarding,synchronous diving-3m springboard, synchronous diving-10m platform, and trampoline.
- There are 1106 samples.





Sport	Avg. Seq. Len.	# Samples	Score Range	# Participants	View Variation
Single Diving 10m platform	97	370	21.60 - 102.60	1	negligible
Gymnastic vault	87	176	12.30 - 16.87	1	large
Big Air Skiing	132	175	8 - 50	1	large
Big Air Snowboarding	122	206	8 - 50	1	large
Sync. Diving 3m springboard	156	88	46.20 - 104.88	2	negligible
Sync. Diving 10m platform	105	91	49.80 - 99.36	2	negligible
Trampoline	634	83	6.72 - 62.99	1	small

Table 1: **Characteristics of AQA-7 dataset.**

4.5 Sports Videos in the Wild

Sports Videos in the Wild [10] or SVW has the following properties :

- SVW contains 4200 videos captured using smartphones by users of Coach's Eye smartphone app, a leading app for sports training developed by TechSmith corporation.
- SVW includes 30 categories of sports and 44 different actions.
- Due to imperfect practice of amateur players and unprofessional capturing by amateur users, SVW is very challenging for automated analysis.
- SVW can be used in : genre categorization, action recognition, action detection, and spatio-temporal alignment.

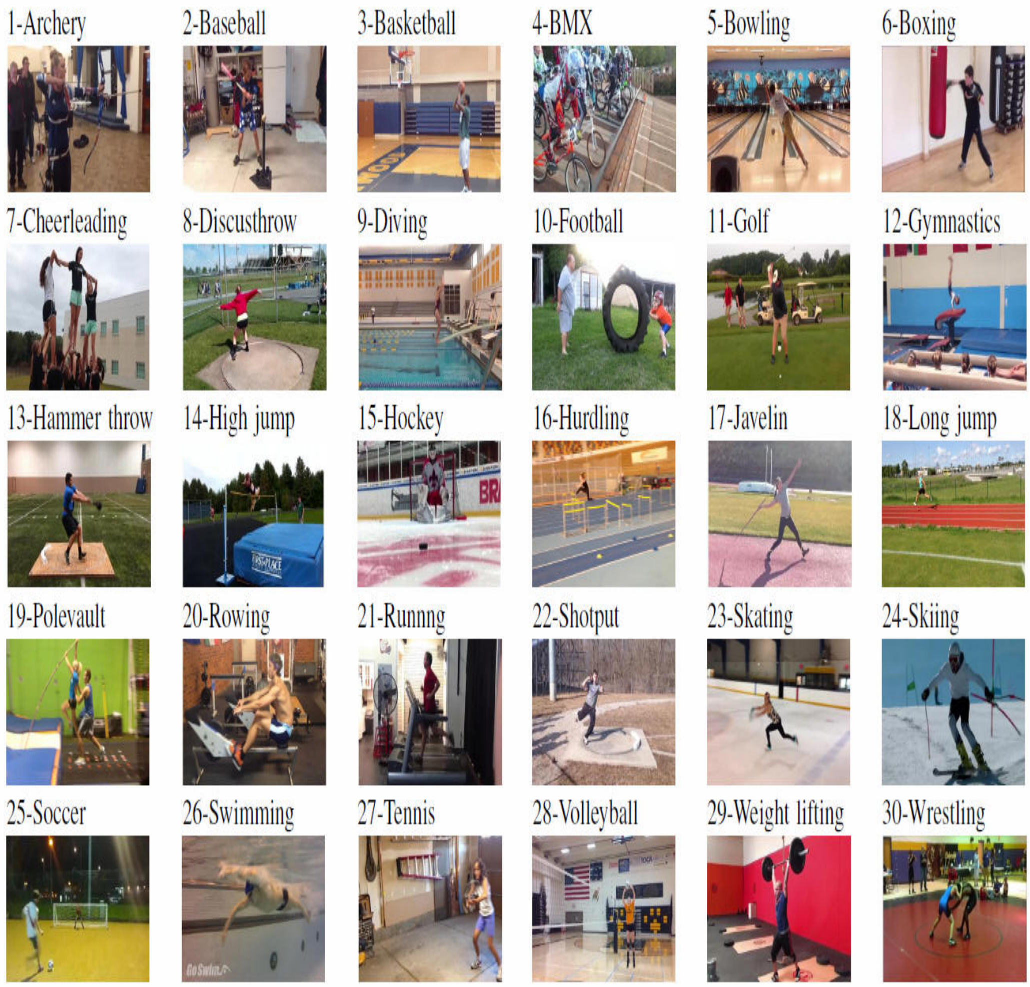
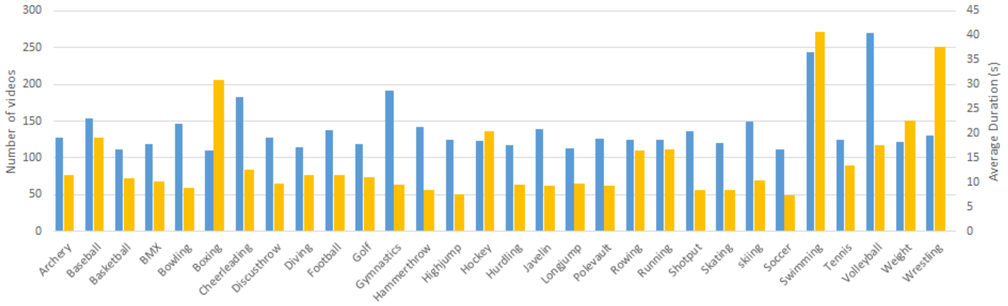
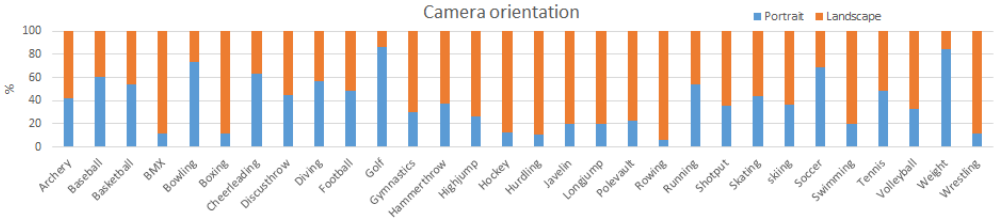
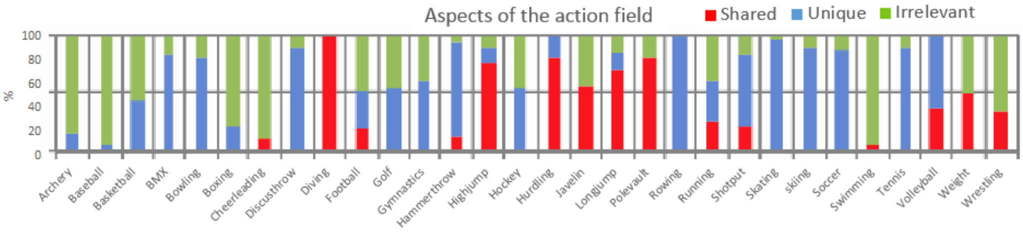


Figure 4.2: SVW Classes




Camera orientation
Portrait
Landscape
100
80
60
40
20
0
%
Archery
Baseball
Basketball
BMX
Bowling
Boxing
Cheerleading
Discusthrow
Diving
Football
Golf
Gymnastics
Hammerthrow
Highjump
Hockey
Hurdling
Javelin
Longjump
Polevault
Rowing
Running
Shotput
Skating
skiing
Soccer
Swimming
Tennis
Volleyball
Weight
Wrestling


Aspects of the action field
Shared
Unique
Irrelevant
100
80
60
40
20
0
%
Archery
Baseball
Basketball
BMX
Bowling
Boxing
Cheerleading
Discusthrow
Diving
Football
Golf
Gymnastics
Hammerthrow
Highjump
Hockey
Hurdling
Javelin
Longjump
Polevault
Rowing
Running
Shotput
Skating
skiing
Soccer
Swimming
Tennis
Volleyball
Weight
Wrestling

Chapter 5

Implementation

In this chapter we carefully provide detailed training and testing methodology.

5.1 Train Test Split

UCF-101 provides train - test split file, so we have used them.For other datasets, we decide the ratio of train test split to be $70 - 30$ or $80 - 20$. For each class, we randomly split the videos into train and test folder according to the ratio. Scikit-Learn's traintestsplit package was extremely useful.

5.2 Preprocessing

We have resized every frame to 224×224 since vgg16 accepts input of the same size. For preprocessing, we transformed every pixel value within range of $0 - 1$ by dividing them by 255.

5.3 Training

The training procedure aims to optimize the CrossEntropy loss with stochastic gradient descent. We have limited ourselves with$N = 4$ for computational limitations, that is, we sampled 4 frames uniformly from each video.The learning rate is kept at 0.001.The Dropout ratio is kept at 0.5.

We used pretrained vgg16 networks, which provides us with a strong starting point. After each epoch of training, we monitor the test accuracy. We stop training when we observe the accuracy on both training and testing is nonincreasing.

5.4 Testing

For testing, we use top-1 accuracy method.For each video, we select N frames uniformly,resize them to 224×224, then pass them through our trained model,consider argmax of the probabilities and compare with the correct label.

5.5 Computational Details

We have implemented the model in python using PyTorch framework.We have used the CSSC computational GPU server for training and testing.Also, in the preprocessing stage, we have extracted frames beforehand to save time and memory space during execution of training process. Depending on the dataset, training time ranges from 1 hr to 30 hr using single NVIDIA GPU. Due to unavailability of GPU memory in most of the time, we ran training process on CPU also, which significantly increased the training time by at least $10x - 20x$.

Chapter 6

Result

6.1 Evaluation Metric

We have used accuracy as the evaluation metric for every model, since accuracy is the standard metric in deep learning community.

6.2 UCF-101

We have run 15 epochs with $N = 4$, $lr = 0.001$ using SGD.
Training accuracy : 99.21% and test accuracy 59.74%

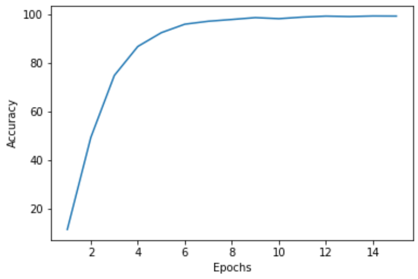


Figure 6.1: UCF-101 Train

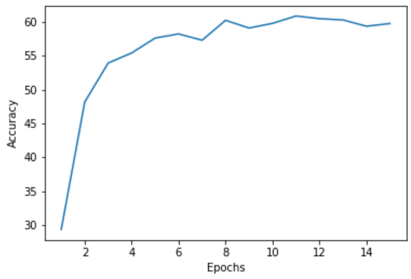


Figure 6.2: UCF-101 Test

6.3 KTH

We have run 20 epochs with $N = 4$, $lr = 0.001$ using SGD.
Training accuracy : 70.15% and test accuracy 60%

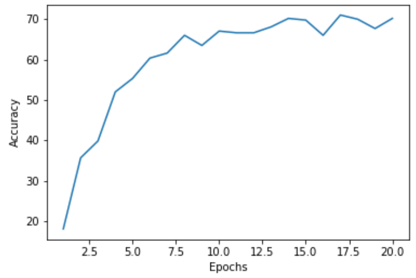


Figure 6.3: KTH Train

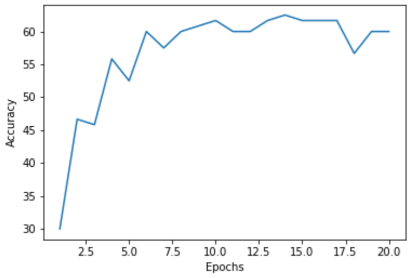


Figure 6.4: KTH Test

6.4 UCF-Sports

We ran for 25 epochs with the same hyper-parameters and algorithm. Training accuracy : 100%, test accuracy : 68.97%

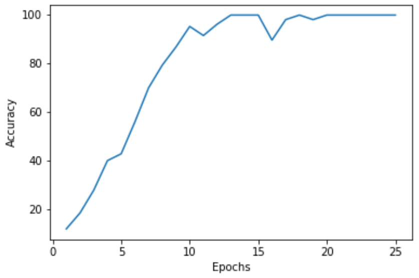


Figure 6.5: UCF-Sports Train

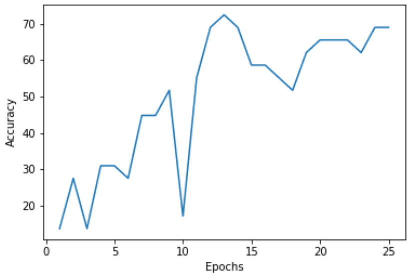


Figure 6.6: UCF-Sports Test

6.5 Action Quality Assessment Performance

We ran for 20 epochs with the same hyper-parameters and algorithm. Training accuracy : 100%, test accuracy : 97.51%

6.6 Sports Videos in the Wild Performance

We ran for 25 epochs with the same hyper parameters and algorithm. Training accuracy : 100%, test accuracy : 74.56%

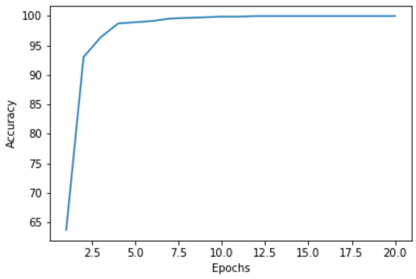


Figure 6.7: AQA Train

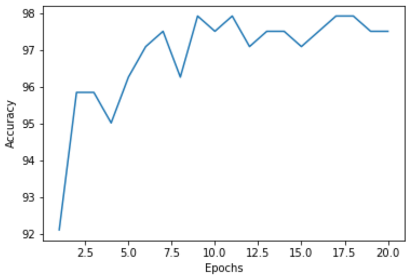


Figure 6.8: AQA Test

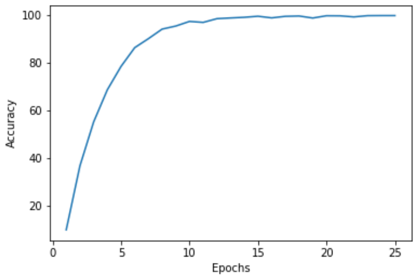


Figure 6.9: SVW Train

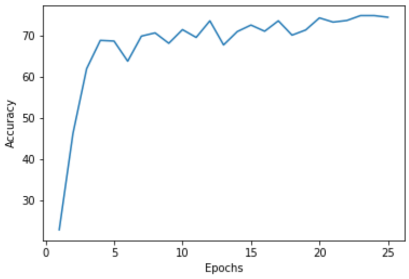


Figure 6.10: SVW Test

Chapter 7

Related Work

7.1 Comparison

We now start comparing our model with others. A few points regarding this :

- For each of the dataset, we find some papers.

- Find and compare the results they have obtained.

- Since people have used different metrics for evaluating their models, it is difficult to decide whether their model is actually better or it is due to the evaluation metric.

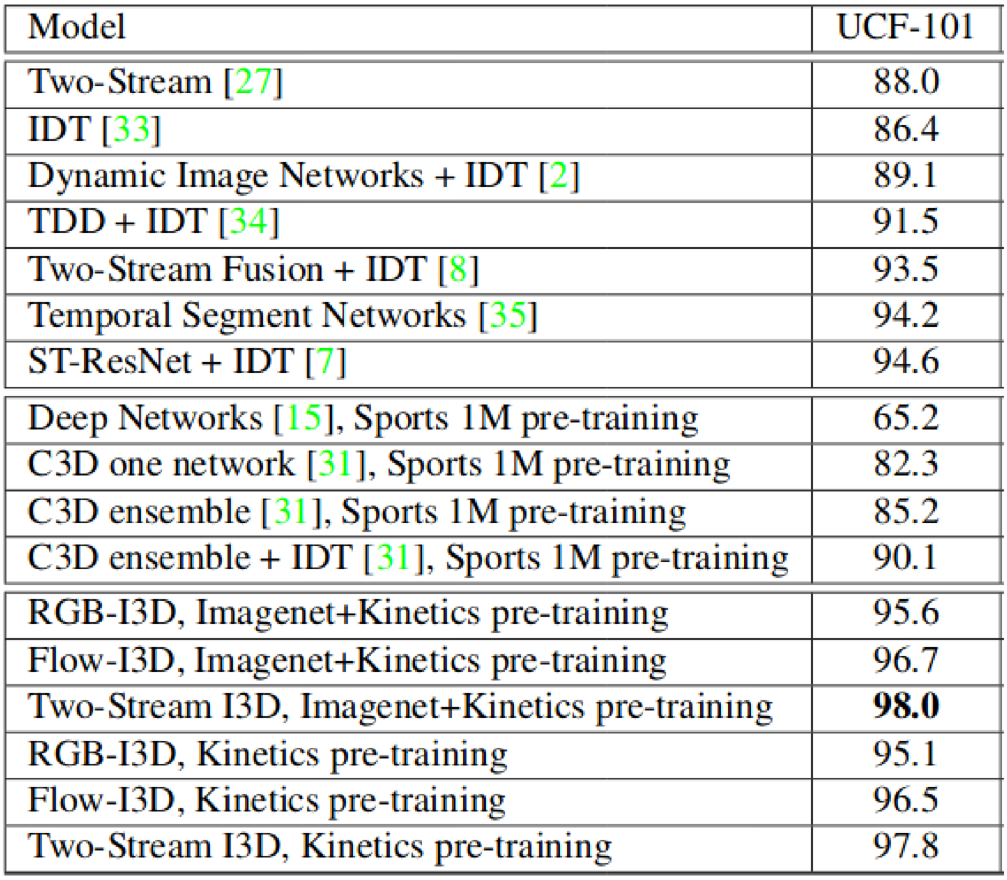
- We only report top papers that we have came across while searching. The sources of the informations are referenced.

7.2 UCF-101

We found the following comparison chart provided by [1]. Our Approach: Test accuracy 59.74%

7.3 KTH

We have come across with the following chart [16] : Our Approach: Test accuracy 60%



Model	UCF-101
Two-Stream [27]	88.0
IDT [33]	86.4
Dynamic Image Networks + IDT [2]	89.1
TDD + IDT [34]	91.5
Two-Stream Fusion + IDT [8]	93.5
Temporal Segment Networks [35]	94.2
ST-ResNet + IDT [7]	94.6
Deep Networks [15], Sports 1M pre-training	65.2
C3D one network [31], Sports 1M pre-training	82.3
C3D ensemble [31], Sports 1M pre-training	85.2
C3D ensemble + IDT [31], Sports 1M pre-training	90.1
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**
RGB-I3D, Kinetics pre-training	95.1
Flow-I3D, Kinetics pre-training	96.5
Two-Stream I3D, Kinetics pre-training	97.8

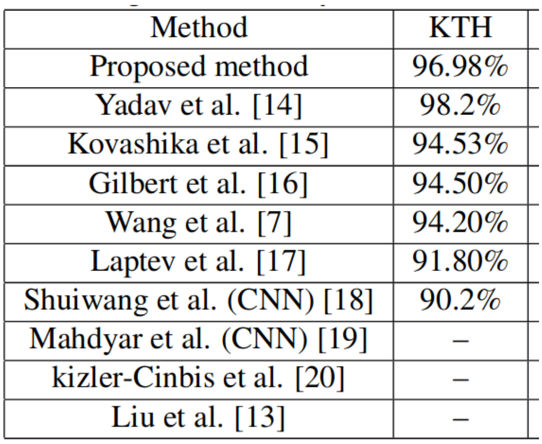
Figure 7.1: UCF-101 Comparison

7.4 UCF Sports

The following result is from the paper [5]. Our Approach: Test accuracy : 68.97%

7.5 Action Quality Assessment

This dataset is very recent and people haven't applied it for action recognition. The main paper [8] gives the following table : Our Approach: Test accuracy : 97.51%



Method	KTH
Proposed method	96.98%
Yadav et al. [14]	98.2%
Kovashika et al. [15]	94.53%
Gilbert et al. [16]	94.50%
Wang et al. [7]	94.20%
Laptev et al. [17]	91.80%
Shuiwang et al. (CNN) [18]	90.2%
Mahdyar et al. (CNN) [19]	–
kizler-Cinbis et al. [20]	–
Liu et al. [13]	–

Figure 7.2: KTH

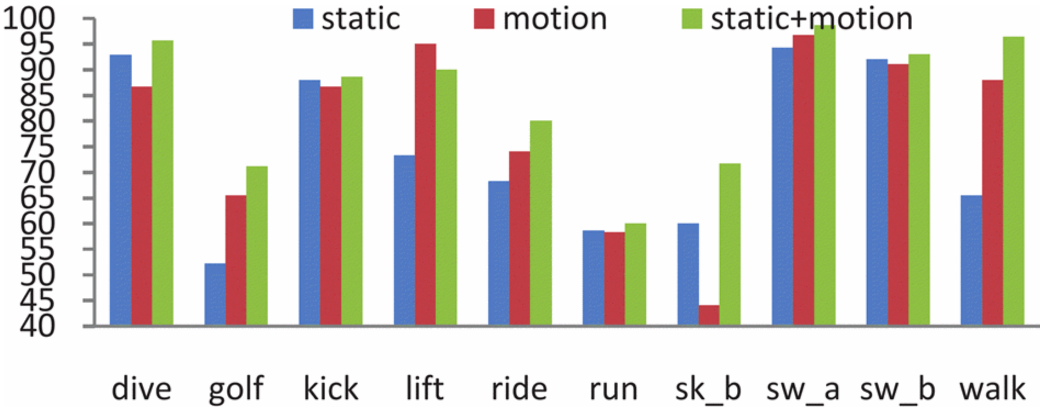


Figure 7.3: The average accuracy for static, motion and static+motion experimental strategy is 74.5%, 79.6% and 84.5% respectively.

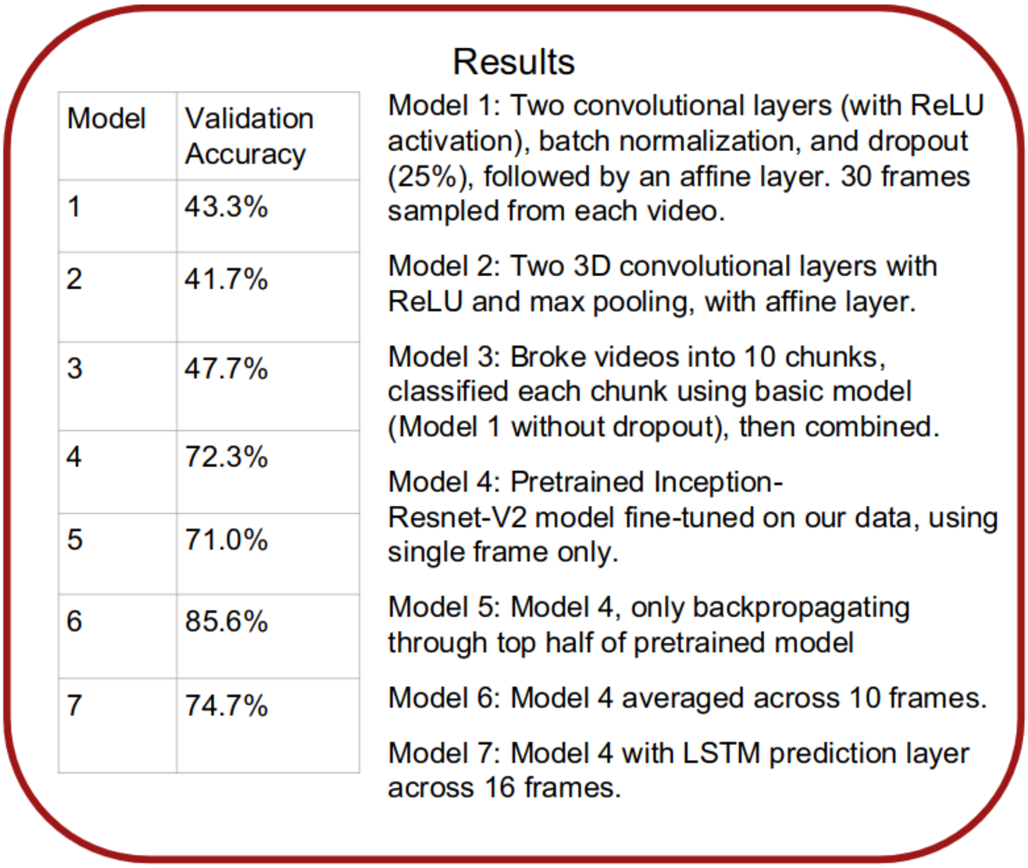
7.6 Sports Videos in the Wild

The main paper [10] who prepared the dataset reports highest accuracy of 61.53% .The following result is from Stanford :

Our Approach: Test accuracy : 74.56%

Unseen action class / Training action class	Diving	Gym-vault	Skiing	Snow-board	Sync-Dive 3m	Sync-Dive 10m	Avg. Corr.
Random Wts./Ini.	0.0590	0.0280	-0.0602	-0.0703	-0.0146	-0.0729	-0.0218
Diving	**0.6997**	-0.0162	0.0425	0.0172	0.2337	0.0221	0.0599
Gymvault	0.0906	**0.8472**	0.0517	0.0418	-0.1642	-0.3200	-0.0600
Skiing	0.2653	-0.1856	**0.6711**	0.1807	0.1195	0.2858	0.1331
Snowboard	0.2115	-0.2154	0.3314	**0.6294**	0.0945	0.1818	0.1208
Sync. Dive 3m	0.1500	-0.0066	-0.0494	-0.1102	**0.8084**	0.0428	0.0053
Sync. Dive 10m	0.0767	-0.1842	0.0679	0.0360	0.4374	**0.7397**	0.0868
Multi-action	0.2258	0.0538	0.0139	0.2259	0.3517	0.3512	**0.2037**

Table 3: **Zero-shot AQA**. Performance comparison of randomly-initialized model, single-action models (for *e.g.*, first row shows the results of training on diving action measuring the quality of the remaining (unseen) action classes), and multi-action model (all-action model trained on five action classes) on unseen action classes. In multi-action class, the model is trained on five action classes and tested on the remaining action class (column-wise). In single-action model rows, diagonal entries show results of training and testing on the same action. Avg. Corr. shows the result of average (using Fisher's z-score) correlation across all columns.



Results

Model	Validation Accuracy
1	43.3%
2	41.7%
3	47.7%
4	72.3%
5	71.0%
6	85.6%
7	74.7%

Model 1: Two convolutional layers (with ReLU activation), batch normalization, and dropout (25%), followed by an affine layer. 30 frames sampled from each video.

Model 2: Two 3D convolutional layers with ReLU and max pooling, with affine layer.

Model 3: Broke videos into 10 chunks, classified each chunk using basic model (Model 1 without dropout), then combined.

Model 4: Pretrained Inception-Resnet-V2 model fine-tuned on our data, using single frame only.

Model 5: Model 4, only backpropagating through top half of pretrained model

Model 6: Model 4 averaged across 10 frames.

Model 7: Model 4 with LSTM prediction layer across 16 frames.

Chapter 8

Conclusion

8.1 Performance

We find that though in some cases our results are not in par with current state of the art, our results are quite satisfactory in comparison with other Machine Learning/Deep Learning models. The main reason is computational capacity, which bottlenecks our architecture. But, with this limited source of computational facility, our architecture is able to perform good in datasets such as Sports Videos in The Wild, which is a good achievement.

8.2 Future Work

In future, we plan to extend our architecture and experiment with larger datasets.

Chapter 9

Appendix

9.1 Train Test Split Code

```

# coding: utf-8

# In[20]:


import os
from sklearn.model_selection import train_test_split


# In[21]:


PATH = 'SVW/Videos/'


# In[22]:


os.makedirs('Train')
os.makedirs('Test')

list_of_labels = os.listdir(PATH)
video_path = os.path.join(os.getcwd(),PATH)

X = []
y = []
for label in list_of_labels:
    os.makedirs('Train/' + label)
    os.makedirs('Test/' + label)
    path_to_label = os.path.join(video_path,label) + '/'
```

```python

    #print(path_to_label)
    list_of_labelled_video = os.listdir(path_to_label)
    for video in list_of_labelled_video:
        path_to_video = os.path.join(path_to_label, video)
        print(path_to_video, label)
        X.append(path_to_video)
        y.append(label)


# In[23]:


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,stratify=y)


# In[24]:


for i in range(len(y_test)):

    file_name = X_test[i].split('/')[-1]
    copy_to_path = os.getcwd() + '/' + 'Test/' + y_test[i] + '/' + file_name
    print(X_test[i],copy_to_path)
    os.rename(X_test[i],copy_to_path)


# In[25]:


for i in range(len(y_train)):

    file_name = X_train[i].split('/')[-1]
    copy_to_path = os.getcwd() + '/' + 'Train/' + y_train[i] + '/' + file_name
    print(X_train[i],copy_to_path)
    os.rename(X_train[i],copy_to_path)
```

9.2 PreProcessing Code

```python
import os
import shutil
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pickle

def extractFrames(pathIn, pathOut):
    """
    This code takes absolute path of the video(pathIn) and returns the frames of the video in the folder pathOut.
    If the folder is not present, it will be created.
    """
    os.makedirs(pathOut, exist_ok=True)

    cap = cv2.VideoCapture(pathIn)
    count = 0

    cap.read()
    while (cap.isOpened()):

        # Capture frame-by-frame
        ret, frame = cap.read()

        if ret == True:
            #print('Read %d frame: ' % count, ret)
            cv2.imwrite(os.path.join(pathOut, "{:d}.jpg".format(count)), frame)  # save frame as JPEG file
            count += 1
        else:
            break

    # When everything done, release the capture
    cap.release()
    cv2.destroyAllWindows()


# In[6]:


def extract_dataset(folder_name = '/user1/student/mtc/mtc2017/cs1706/dissertation/',frame_dir = '/user1/student/mtc/mtc2017/cs1706/dissertation/Extracted_Frames_test/',N=4):
    """
    folder_name contains the path to training folder.
    frame_dir contains the folder where the extracted frames of the videos will be stored.
    N is the number of frames we need from each video.
    """
    list_ = []
    list_ = os.listdir(folder_name) #contains name of all the labels
    #print('list_',list_)
    dict_of_labels = {} #stores the path to the extracted frames of an video as key and the label as value.
    #list stores class names
    for i in list_:
        tmp = folder_name + '/' + i # i is the label of video
        #print('i = ',i)
        _list = os.listdir(tmp) # stores the name of the videos in the class.
        for vid in _list:
            pathIn = tmp + '/' + vid
            #print('tmp - vid ',tmp,vid)
            pathOut = frame_dir + i + '_' + vid + '_jpg'
            dict_of_labels[pathOut] = i
            #print('pathin-out',pathIn,pathOut)
            # Extracting frames from the video and storing to the required destination

            extractFrames(pathIn,pathOut)
            # To select the frames we need
            list_of_files = os.listdir(pathOut)
            num_frames = len(list_of_files) # counts the number of frames
            selected_frame_indices = np.linspace(start=0,stop=num_frames,num=N+1,dtype=np.int)[:-1]
            selected_frame_names = [str(x) + '.jpg' for x in selected_frame_indices]
            #print(selected_frame_names)
            # Deleting the unnecessary frames
            for file in list_of_files:
                if file in selected_frame_names:
                    print('the following file remains', file)
                else:
                    #print('this should be deleted:', file)
                    os.remove(os.path.join(pathOut,file))

        #print(_list)
    return dict_of_labels


# In[8]:


def dict_save(framelist, path = '/user1/student/mtc/mtc2017/
```

```python
cs1706/dissertation/',file = 'dict.save'):
    """
    Utility function To save the dict_of_labels in a file for future use.
    """
    with open(path+file , 'wb') as f:
        pickle.dump(framelist , f)

def dict_load(path = '/user1/student/mtc/mtc2017/cs1706/dissertation/', file = 'dict.save'):
    """
    Utility function To load the dict_of_labels from a file for future use.
    """
    with open(path+file , 'rb') as f:
        framelist = pickle.load(f)
    return framelist

def get_numeric_labels(path='Action/Test/'):
    """
    Provides numeric labels for each of the class.The path to dataset is input.
    Outputs a dict containing the string labels as keys and numeric labels as values.
    """
    list_of_labels = os.listdir(path)
    label_dict = {}
    i = 0;
    for label in list_of_labels:
        label_dict[label] = i
        i += 1

    for key,item in label_dict.items():
        print(key,item)
    return label_dict

label_dict = get_numeric_labels()
PATH = os.getcwd() + '/'
dict_save(label_dict,path = PATH, file = 'dict_of_labels.save')
dict_labels = dict_load(PATH,'dict_of_labels.save')

# In[9]:


train_folder_name = os.path.join(os.getcwd(),'Action/Train/')
train_frame_dir = os.path.join(os.getcwd(),'Extracted_Frames_train/')
print(train_folder_name,train_frame_dir)
test_folder_name = os.path.join(os.getcwd(),'Action/Test/')
```

```python
test_frame_dir = os.path.join(os.getcwd(),'Extracted_Frames_test/')
print(test_folder_name,test_frame_dir)


# In[11]:
dict_test = extract_dataset(test_folder_name,test_frame_dir)
dict_save(dict_test,os.getcwd() + '/',file='dict_test.save')
print('Test dataset successfully preprocessed')

dict_train = extract_dataset(train_folder_name,train_frame_dir)
dict_save(dict_train,os.getcwd() + '/',file='dict_train.save')
print('Train dataset successfully preprocessed')
```

9.3 Training and Evaluation Code

```python
import os
import shutil
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pickle
import torch
import torchvision.models as models



def dict_save(framelist, path = '/user1/student/mtc/mtc2017/cs1706/dissertation/',file = 'dict.save'):
    """
    Utility function To save the dict_of_labels in a file for future use.
    """
    with open(path+file, 'wb') as f:
        pickle.dump(framelist, f)

def dict_load(path = '/user1/student/mtc/mtc2017/cs1706/dissertation/', file = 'dict.save'):
    """
    Utility function To load the dict_of_labels from a file for future use.
    """
    with open(path+file, 'rb') as f:
        framelist = pickle.load(f)
    return framelist


# Assuming N = 4, we create 4 vgg16 models
mod1=models.vgg16(pretrained=True)
mod2=models.vgg16(pretrained=True)
mod3=models.vgg16(pretrained=True)
mod4=models.vgg16(pretrained=True)


# In[17]:


# Taking out the last 7 layers
mod1.classifier=mod1.classifier[:-7]
mod2.classifier=mod2.classifier[:-7]
mod3.classifier=mod3.classifier[:-7]
mod4.classifier=mod4.classifier[:-7]

```

```


# In[18]:


output_list = []
models = [mod1, mod2, mod3, mod4] #putting models to a list


# In[19]:


class PartC(torch.nn.Module):
    def __init__(self, num_frames, n_classes=10):
        super(PartC, self).__init__()

        self.num_frames = num_frames
        kernel_size = 3
        fc_input = 256 * (self.num_frames - kernel_size + 1) * 5 * 5
        self.conv3d = torch.nn.Conv3d(512, 256, kernel_size)
        self.relu1 = torch.nn.ReLU()
        self.fc1 = torch.nn.Linear(fc_input, 4096)
        self.relu2 = torch.nn.ReLU()
        self.dropout1 = torch.nn.Dropout()
        self.fc2 = torch.nn.Linear(4096, 1024)
        self.relu3 = torch.nn.ReLU()
        self.dropout2 = torch.nn.Dropout()
        self.fc3 = torch.nn.Linear(1024, n_classes)
        #self.softmax = torch.nn.Softmax(dim=-1)


    def forward(self, x):
        x = self.conv3d(x)
        x = self.relu1(x)
        x = x.view(1, -1)
        x = self.fc1(x)
        x = self.relu2(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu3(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        #x = self.softmax(x)
        return x


# In[20]:


```

```
class VGG3d(torch.nn.Module):
      def __init__(self, A, C):
        super(VGG3d, self).__init__()

        self.A = torch.nn.ModuleList(A)
        self.C = C

      def forward(self, video):
        output_list = []

        for i in range(len(self.A)):
            out = self.A[i](video[i].unsqueeze(0))
            output_list.append(out)

        B = torch.cat(output_list).transpose(1, 0)     # Concatenation
        final_output = self.C(B.unsqueeze(0))
        return final_output


# In[21]:


device = 'cuda:2'
cuda1 = torch.device(device)


# In[22]:


#Instanciation of the model. .cuda(cuda1) is added to move the model into GPU memory.
models = [mod1.features.cuda(cuda1), mod2.features.cuda(cuda1), mod3.features.cuda(cuda1), mod4.features.cuda(cuda1)]
C = PartC(num_frames=4, n_classes=30)
vgg3d = VGG3d(models, C).cuda(cuda1)


# In[10]:


def image_resize(filename, shape=(224,224)):
    """
    Utility function to resize an image to (224,224,3) which is the input size needed to feed into the model
    """
    image = cv2.imread(filename)
    new_img = cv2.resize(image,shape)
    return new_img
```

```


# In[11]:


def get_frame_from_one_video(folder_path):
    """
    This utility function loads frames of an video, after resizing them to (224,224,3) format
    Input is path to folder where the frames of the video is stored.
    Returns a numpy array of size (N,3,224,224)
    """
    frame_list = []
    list_of_files = os.listdir(folder_path)

    for frame_name in list_of_files:
        temp_path = os.path.join(folder_path,frame_name)
        temp_img = image_resize(temp_path)
        temp_img = np.array(temp_img,np.float32)
        frame_list.append(temp_img.T)
    return np.array(frame_list)


# In[12]:


def training(vgg3d,epochs=1):

    criteria = torch.nn.CrossEntropyLoss().cuda(cuda1)
    optimizer = torch.optim.SGD(vgg3d.parameters(), lr=0.001)
    saved_list = dict_load('/user1/student/mtc/mtc2017/cs1706/dissertation/','dict_train.save')
    saved_list_test = dict_load('/user1/student/mtc/mtc2017/cs1706/dissertation/','dict_test.save')
    get_label = dict_load('/user1/student/mtc/mtc2017/cs1706/dissertation/','dict_of_labels.save')
    #epochs = 10
    for epoch in range(epochs):
        correct = 0
        total = 0
        vgg3d.train()
        l = np.random.permutation(len(saved_list))
        for pos in l:
            key,item = list(saved_list.items())[pos]
            #print(video)
            if len(os.listdir(key)) >= 4:
                total += 1 # for training accuracy
                path_to_video = key
            #print(path_to_video,item)
            temp_list = get_frame_from_one_video(path_to_video)
            frame_list = []
            #print(frame_list.max(), frame_list.min())
            #print('current epoch = ',epoch)
            for i in range(temp_list.shape[0]):
                temp = temp_list[i].astype(float)/255.0
                frame_list.append(temp)
                #print(i,frame_list[i].dtype)
            frame_list = np.array(frame_list,np.float64)
            inp = torch.from_numpy(frame_list).type(torch.FloatTensor)
            inp = inp.cuda(cuda1)#for running in gpu

            #print('len_frame:', frame_list.shape)
            #print('inp_shape:', inp.shape)

            #print('inp_0_shape:', inp[0].shape)
            vgg3d.zero_grad()
            prediction = vgg3d(inp).cuda(cuda1)
            #print(prediction.shape)
            target = get_label[item]
            #For training accuracy
            predicted_label = prediction.argmax()
            #print(predicted_label.item(),target,correct,total)
            if predicted_label == target:
                correct += 1
            target = torch.tensor(target)
            target = target.unsqueeze(0).type(torch.LongTensor).cuda(cuda1)
            #print('prediction target',prediction.shape,target.shape,type(prediction),type(target))
            #print(prediction.argmax(), target)
            loss = criteria(prediction, target)
            loss.backward()
            optimizer.step()
        #else:
        #    print('has less than 4 frames', key)

    print('train accuracy after epoch is ',epoch, correct/total)


    correct_test = 0
    total_test = 0
    vgg3d.eval()
    for key,item in saved_list_test.items():
            if len(os.listdir(key)) >= 4:
                total_test += 1 # for training accuracy
                path_to_video = key
                temp_list = get_frame_from_one_video(path_to_video)
                frame_list = []
                for i in range(temp_list.shape[0]):
                    temp = temp_list[i].astype(float)/255.0
                    frame_list.append(temp)
                        #print(i,frame_list[i].dtype)
                frame_list = np.array(frame_list,np.float64)
                inp = torch.from_numpy(frame_list).type(torch.FloatTensor)
                inp = inp.cuda(cuda1)#for running in gpu

                prediction = vgg3d(inp).cuda(cuda1)
                target = get_label[item]
                predicted_label = prediction.argmax()
                if predicted_label == target:
                    correct_test += 1
                #else:
                #    print('has less than 4 frames', key)
                #print(predicted_label.item(),target,correct_test,total_test)
        print('testing accuracy after epoch is ',epoch,correct_test/total_test)

    return vgg3d


# In[13]:


vgg3d = training(vgg3d, 25)


# In[14]:


PATH = os.getcwd() + '/saved_gpu_dict.pth'
torch.save(vgg3d.state_dict(),PATH)
```

References

[1] Joao Carreira and Andrew Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *arXiv e-prints*, arXiv:1705.07750 (May 2017), arXiv:1705.07750. arXiv: `1705.07750 [cs.CV]`.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[3] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(91)90009-T`. URL: `http://www.sciencedirect.com/science/article/pii/089360809190009T`.

[4] F. Husain, B. Dellen, and C. Torras. “Action Recognition Based on Efficient Deep Feature Learning in the Spatio-Temporal Domain”. In: *IEEE Robotics and Automation Letters* 1.2 (July 2016), pp. 984–991. ISSN: 2377-3766. DOI: `10.1109/LRA.2016.2529686`.

[5] J. Liu, Jiebo Luo, and M. Shah. “Action recognition in unconstrained amateur videos”. In: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. Apr. 2009, pp. 3549–3552. DOI: `10.1109/ICASSP.2009.4960392`.

[6] Allen Newell. “Perceptrons. An Introduction to Computational Geometry. Marvin Minsky and Seymour Papert. M.I.T. Press, Cambridge, Mass., 1969. vi + 258 pp., illus. Cloth, 12; *paper*,4.95”. In: *Science* 165.3895 (1969), pp. 780–782. ISSN: 0036-8075. DOI: `10.1126/science.165.3895.780`. eprint: `https://science.sciencemag.org/content/165/3895/780.full.pdf`. URL: `https://science.sciencemag.org/content/165/3895/780`.

[7] Paritosh Parmar and Brendan Tran Morris. “Action Quality Assessment Across Multiple Actions”. In: *arXiv e-prints*, arXiv:1812.06367 (Dec. 2018), arXiv:1812.06367. arXiv: `1812.06367 [cs.CV]`.

[8] Paritosh Parmar and Brendan Tran Morris. “Action Quality Assessment Across Multiple Actions”. In: *arXiv e-prints*, arXiv:1812.06367 (Dec. 2018), arXiv:1812.06367. arXiv: `1812.06367 [cs.CV]`.

[9] Mikel Rodriguez. *SPATIO-TEMPORAL MAXIMUM AVERAGE CORRELATION HEIGHT TEMPLATES IN ACTION RECOGNITION AND VIDEO SUMMARIZATION*. 2010.

[10] Seyed Morteza Safdarnejad et al. “Sports Videos in the Wild (SVW): A Video Dataset for Sports Analysis”. In: *Proc. International Conference on Automatic Face and Gesture Recognition*. Ljubljana, Slovenia, May 2015.

[11] Christian Schuldt, Ivan Laptev, and Barbara Caputo. *Recognizing Human Actions: A Local SVM Approach*. 2004.

[12] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv e-prints*, arXiv:1409.1556 (Sept. 2014), arXiv:1409.1556. arXiv: `1409.1556 [cs.CV]`.

[13] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild”. In: *arXiv e-prints*, arXiv:1212.0402 (Dec. 2012), arXiv:1212.0402. arXiv: `1212.0402 [cs.CV]`.

[14] Khurram Soomro et al. *Chapter 9 Action Recognition in Realistic Sports Videos*.

[15] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html`.

[16] G. K. Yadav and A. Sethi. “Action recognition using spatio-temporal differential motion”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. Sept. 2017, pp. 3415–3419. DOI: `10.1109/ICIP.2017.8296916`.