Self-Supervised Video Representation Learning by Recurrent Networks and Frame Order Prediction

Sai Shashidhar Nagabandi
sn6582@rit.edu

Self-Supervised Video Representation Learning by Recurrent Networks and Frame Order Prediction

by

Sai Shashidhar Nagabandi

June 2020

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of
Master of Science in Computer Engineering
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology

Approved By:

____*Date:*_______________
Dr. Raymond Ptucha

Primary Advisor – R.I.T. Dept. of Computer Engineering

____*Date:*_______________
Dr. Alexander Loui

Committee Member – R.I.T. Dept. of Computer Engineering

____*Date:*_______________
Dr. Andres Kwasinski

Committee Member – R.I.T. Dept. of Computer Engineering

ABSTRACT

The success of deep learning models in challenging tasks of computer vision and natural language processing depend on good vector representations of data. For example, learning efficient and salient video representations is one of the fundamental steps for many tasks like action recognition and next frame prediction. Most methods in deep learning rely on large datasets like ImageNet or MSCOCO for training, which is expensive and time consuming to collect. Some of the earlier works in video representation learning relied on encoder-decoder style networks in an unsupervised fashion, which would take in a few frames at a time. Research in the field of self-supervised learning is growing, and has shown promising results on image-related tasks to both learn data representations as well as pre-learn weights for networks using unlabeled data. However, many of these techniques use static architectures like AlexNet, which fail to take into account the temporal aspect of videos. Learning frame-to-frame temporal relationships is essential to learning latent representations of video. In our work, we propose to learn this temporality by pairing static encodings with a recurrent long short term memory network. This research will also investigate applying different methods of encoding architecture along with the recurrent network, to take in a range of different number of frames. We also introduce a novel self-supervised task in which the neural network has two tasks; predicting if a tuple of input frames is temporally consistent, and if not, predict the positioning of incorrect tuple. The efficacy is finally measured by using these trained networks on downstream tasks like action recognition on standard datasets UCF101 and HMDB51.

Contents

List of Figures

List of Tables

Acronyms

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

LSTM

Long short-term Memory

GRU

Gated recurrent Unit

C3D

3D convolutional network

SS learning

Self-supervised learning

N

Input number of frames

Chapter 1

Introduction

With the advent of convolutional networks and deep learning, great strides have been made in the field of image and video understanding. Convolutional Neural Networks (CNNs) have helped to solve several challenging tasks in the visual domain. CNNs have become the de-facto method of trying to solve problems in visual domain like image classification, object localization, and semantic segmentation [28,30,31]. These networks usually work on labelled data with annotations generated by humans. This field of learning is called supervised learning. The annotations/labels take a lot of time and resources, and are prone to human error. CNNs are loosely modelled after taking cues from the human brain, but the neurons are oversimplified variants of their biological counterparts, and it is unlikely the current training method of using labelled data with backpropagation reflects learning in the mammalian brain. The human brain is inherently unsupervised in nature [32], and active research is being done in this area. Self-supervised learning is a subset of unsupervised learning, where we use unlabeled data to 'pre-train' the network which would help our supervised training. The aim of this method is to leverage and use the huge amounts of available unlabeled data [23,34], and then fine-tune with limited supervised data. In self-supervised learning, we generate 'pseudo-labels' which are computer generated. These labels are generated by asking the network to solve a task, which is usually based on the structure of the data [35,39,40]. Self-supervised learning can be on different domains – image/audio/video and geometry (pose etc.).

Self-supervised learning on videos is of interest because videos are inherently challenging to annotate. In videos there is a lot of temporal movement, including temporal occlusions and objects coming into and out of the frame field of view. Such a dynamic scene makes it difficult to label each frame by a single class for training. Segmentation maps in a video sequence might be helpful, but creating such datasets is very time consuming and costly. Also, there are a lot of unlabeled videos available [23], which can be used as a pre-training in a self-supervised fashion. In this work, we specifically aim to leverage the temporal relation between frames in a video as a self-supervised learning signal to train our networks.

We model our task primarily as representation learning, and evaluate this on downstream task in video classification - action recognition of videos. We test this on two standard datasets of action recognition- UCF101 [24] and HMDB 51 [25] datasets.

1.1 Contributions

The main contributions of this thesis can be summarized as follows:

- Develop a self-supervised pre-text task for video representation learning, using frame order prediction as a learning signal.
- Develop and compare different encoding architectures, which can subsequently work on different number of frames *N*.
- Test the efficiency of model on a downstream task – action recognition in videos.

Chapter 2

Background

2.1 Convolutional Neural Networks

A typical CNN is shown in Fig 2.1. In CNN the major layers are as follows

- Convolutional layer
- Pooling layer
- Dense layer (Fully connected layer)

CNNs typically work on images, and have lots of parameters (>100M parameters) [34]. The convolutional layer of a CNN takes in an image and applies a filtering operation on it. Each convolutional layer has several filters, and their parameters or 'weights' are learned over several iterations. In the pooling layer, the size of the activation maps is reduced. This reduction is based on the type of the pooling, and the pool size. Finally, the dense layer of a CNN is used to take the output from a pooling layer to the classification layer. Between layers there are activation units. These induce non-linearity into the model so that it can learn complex functions. We usually use ReLU as an activation function, but there are many alternatives such as leaky ReLU, PReLU (Parametric ReLU), sigmoid and RR-ReLU (Randomized leaky ReLU). [13,15].

The parameters are learned by adjusting them over several iterations, using gradient descent and backpropagation. There are optimizers like Adam [27] which are also used to aid the training process.

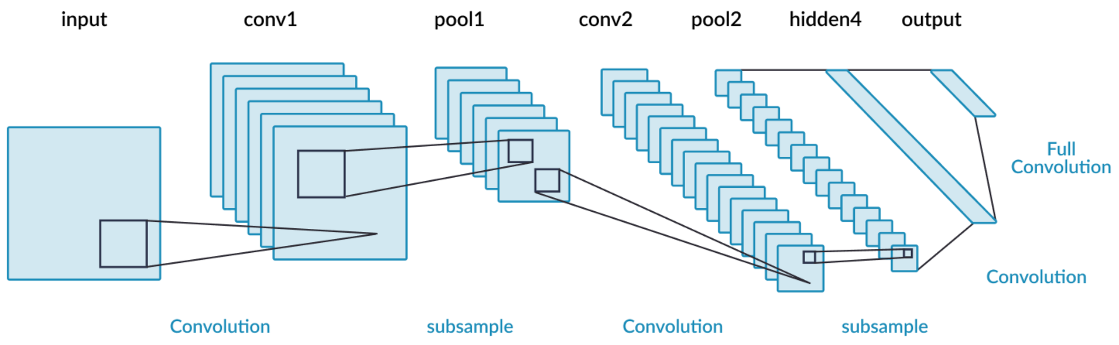


Fig 2.1 Convolutional neural network architecture [10].

2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network which are designed to work on temporal sequences. RNNs are used extensively to solve problems which have a sequential and causal nature. They include language translation, time series prediction, speech recognition, and video understanding.

RNNs usually have an input state and a hidden state. The hidden unit is the output of the previous time step. The units of an RNN block are usually Long Short-Term Memory (LSTM) unit [41] or a Gated Recurrent Unit (GRU) [42]. Different configurations of RNNs are shown in Fig 2.2.

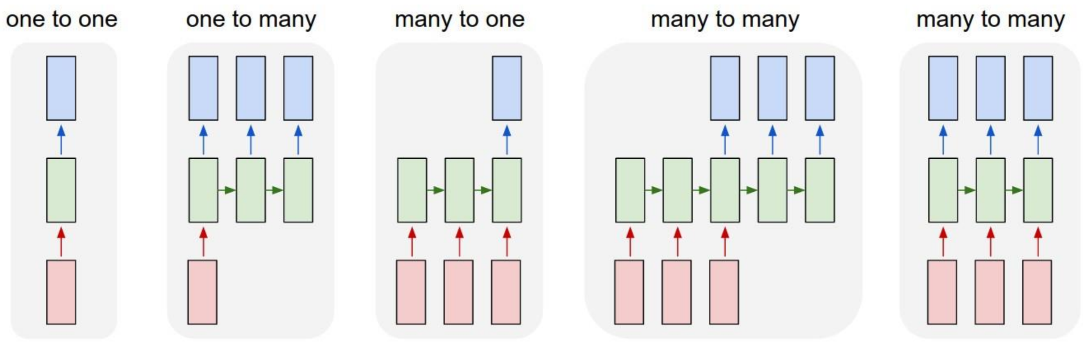


Fig 2.2 Different styles of recurrent neural network architecture [43].

2.3 Long short-term Memory unit (LTSM)

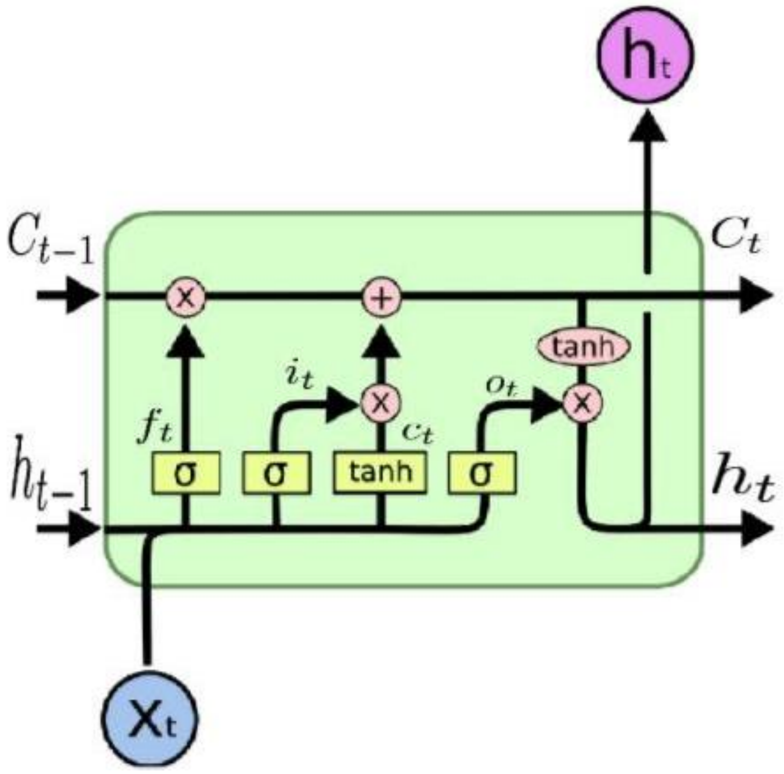


Fig 2.3 Overview of a Long short-term Memory cell [7].

There are three gates in an LSTM cell – an input gate, output gate and forget gate. These gates are primarily used to write to the memory cell in the LSTM. In Fig 2.3, c_t is the state of cell at time step *t*. The LSTM cell gets inputs at four of its terminals every time step. There are two primary sources for it, one of them is the current frame x_t and the second one is h_{t-1}. h_{t-1} is the previous hidden state from all LSTM units. The cell c also has another input which is c_{t-1} which is the input from its previous state.

The gates have sigmoid activation function, and switch between 0 and 1. Their job is to either send the data forward to be written in the memory cell or not. The inputs to the gates are summed up with bias before sending them. The resulting activation is multiplied by input gate.

Then the cell state is multiplied by forget gate f_t and added this term. Finally, we multiply the output gate o_t with the last output from hidden state h_t and pass this through a *tanh* non-linearity. The updates of operations happening in an LSTM cell are outlined as below from (2.3.1) to (2.3.5):

$$i_t = \sigma(W_{xi}X_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \quad (2.3.1)$$

$$f_t = \sigma\left(W_{xf}X_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right), \quad (2.3.2)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}X_t + W_{hc}h_{t-1} + b_c\right), \quad (2.3.3)$$

$$o_t = \sigma(W_{xo}X_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \quad (2.3.4)$$

$$h_t = o_t \tanh(c_t), \quad (2.3.5)$$

The major advantage of using an LSTM cell is that using regular RNNs, backpropagation after few time steps suffers from vanishing gradients. This is because of repeated multiplications of gradients (which are already <1). Now with the addition of sigmoidal gates and memory cell, this problem is solved.

There also exist modalities like videos which encompass both the visual and temporal features in their structure. Thus, videos present challenges which required to combine both CNNs and RNN. Some salient challenges in the video domain include action recognition, pose estimation, video classification and next frame prediction.

2.4 3D convolutional encoder

In this thesis, we use a 3D convolutional encoder called C3D [26] as the backbone architecture with a few changes made. C3D uses 3D convolutional blocks instead of 2D convolutions. 2D convolutions apply the filtering operation only spatially whereas in a 3D model we can filter across time as well. 3D convolutions thus are not limited by appearance representations, and can extract both temporal and spatial features. 3D pooling is done in a similar fashion with pooling in both the spatial and time dimension. In Fig 2.4.1 we show a comparison between 2D and 3D pooling.

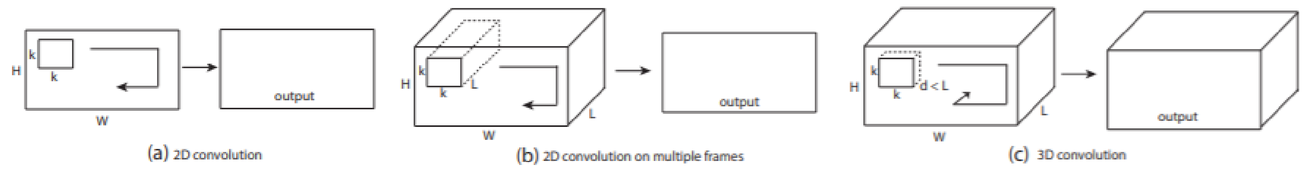


Fig 2.4.1 Comparison between 2D and 3D convolutions [26].

As seen in Fig 2.4.1(a), 2D convolution only filter across the image spatially, or across the channels of an image. But 3D convolution can filter over different images (frames) along with their channels. The default C3D model can take in 16 frames at a time and outputs a 4096 vector. The C3D model is also shown to generate features which are unique to videos, better than the process of sending frames via Siamese CNNs to generate a single encoding for a video sequence [2].

Using this model, the C3D architecture can be seen in Fig 2.4.2. It has eight convolutional layers interspersed by five max pooling layers. All kernels are 3 × 3 × 3 with stride one.

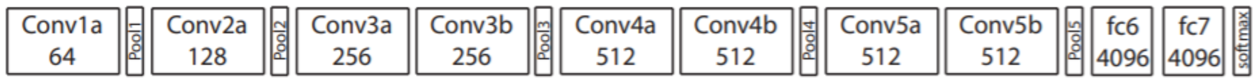


Fig 2.4.2 3D Convolutional Neural Network architecture [26].

Chapter 3

Related Work

There has been a lot of work done in video representation learning. Unsupervised learning has primarily used networks such as Boltzmann machines, autoencoders, and deep belief networks. These works usually rely on generating latent representations for the data, and also induce sparsity [2]. But the problems with the traditional unsupervised techniques is that they do not take into account the temporality of videos, and work on single frames.

Srivastava et al. [1] was one of the first works in learning video representations. The network consists of two sub-modules, each module designed to solve a specific task. The first module is an LSTM autoencoder model which is an encoder-decoder style RNN. The task of the network is to regenerate the input sequence. But the drawback of this method is that the network can simply learn an identity mapping, instead of trying to learn the underlying data semantics. To prevent this, they add another model called the Future predictor model. The task of this network is to predict the future frames of the video. The drawback is that predicting pixels for a simple 256 x 256 image would have $256^{2*3*256}$ hypotheses, and is not very constrained. Due to memory constraints, the network could fit only a few frames, and further, as the sequence grew, the predicted frames would be blurry [12], as the network is unsure of the next action.

Mathieu et al. [12] tried to overcome this by replacing the loss function to generate better images. They used an image gradient loss function instead of the traditional Mean Squared Error (MSE) loss. Along with this, they also used adversarial training and Generative Adversarial Networks (GANs) for next frame prediction. By introducing these techniques, the future frame predictions were less blurry. Also, for measuring the image quality, they used metrics like Signal to Noise ratio (SNR), structural similarity index measure and image sharpness. Nonetheless, the number of frames predicted were not very high, and after first couple of frames the predictions got blurry.

Mishra et al. [2] trained a Siamese binary classifier network to predict if a given sequence of video frames is in the correct order or not. The model takes in a triplet of frames which are sampled from high motion windows. The issue with this method is that a triplet of frames is very small, and does not extrapolate well to longer sequences. Also, the model uses only a simple two-way classifier, which does not require the learning of complex features useful for downstream tasks.

Gould et al. [3] introduced a more difficult self-supervision task by predicting which video sequence out of a series of inputs is temporally incorrect.

Wang et al. [17] focused on making this problem harder by making the network predict statistics of both appearance and motion. The appearance statistics would include predicting change of dominant color of the region (over different frames) and spatio-temporal color diversity. For motion statistics this would include predicting direction of largest motion change. They implement this specifically by dividing a video frame into a $N \times N$ grid, and assigning these tasks to be predicted by their CNN. Their backbone architecture is the 3D convolutional network (C3D) [26]. These tasks they've incorporated into their model are inspired from our understanding of on how the human visual system works.

Chapter 4

Methodology

Our model aims to expand on the current research by introducing a novel self-supervised task. Current research in this domain used only a small number of frames from the video. Misra et al. [2] sampled a triplet of frames (*N*=3), and while Gould et al. [3] has a number of frames of *N* = 6,8. We postulate that by using only a few frames, the network is losing information it can learn from. Using fewer frames also leads to ambiguity between classes. The reason previous works had lower number of frames (*N)* is because of using Siamese CNNs without recurrent networks. We propose that by using LSTM networks, our network would be able to handle a larger number of frames. We also propose to train our network in a self-supervised fashion in which our network is fed a video sequence of *N* frames. The self-supervision task for the network to figure out is if the sequence is temporally consistent or not. Thus, by doing this we modify the self-supervised task from frame prediction to a simpler and less ambiguous one. For the baseline model we have added another pretext task - an ordering task where the position of the frame has to be predicted. We also expand on previous research and introduce a LSTM RNN block to take advantage of the temporal information in videos. We evaluate different encoding techniques and number of frames, and use a 3D convolutional (C3D) encoder. We describe our sampling technique below, followed by three architectures – Baseline model, model with higher number of frames (*N* >3) and model with C3D encoder.

In our dataset, for every video sample we would get a list of frames/images generated from it. These frames are numbered sequentially and written to our local drive. Out of all these frames, we are going to select *N* frames, *N* depending on the encoder/model we are using. These frames are then fed into the network. For self-supervised task, we are going to have two different classes – correct order and wrong order frames. We generate this dataset by randomly selecting 70% of the samples, and then jumbling the numbering in it.

For extracting video frames, we can have 3 different techniques. We can have uniform sampling where we pick frames from the list of frames uniformly. Instead of this we could have random sampling where we pick frames randomly. Another technique we could use is called key frame selection. Key frames are frames in a video which are rich in information, or are important/key. Out of these techniques, we choose the random sampling. We do not choose uniform sampling as adding randomness to the network helps improve performance, and it has been shown in previous works as well [3]. The reason we do not choose key frame selection is because we are working with higher number of frames, and we might not have as many key frames from a video. Also, these key number of frames might not be the same for different videos, so we cannot train the network with unequal sequence lengths. Hence, we choose the random sampling.

4.1 Baseline Model (*N* = 3)

In our proposed model we aim to leverage the temporal relation between the frames as the self-supervised learning signal. Taking cues from previous works, we aim to improve and address the deficiencies mentioned above. The architecture for our network is shown in Fig 4.1.

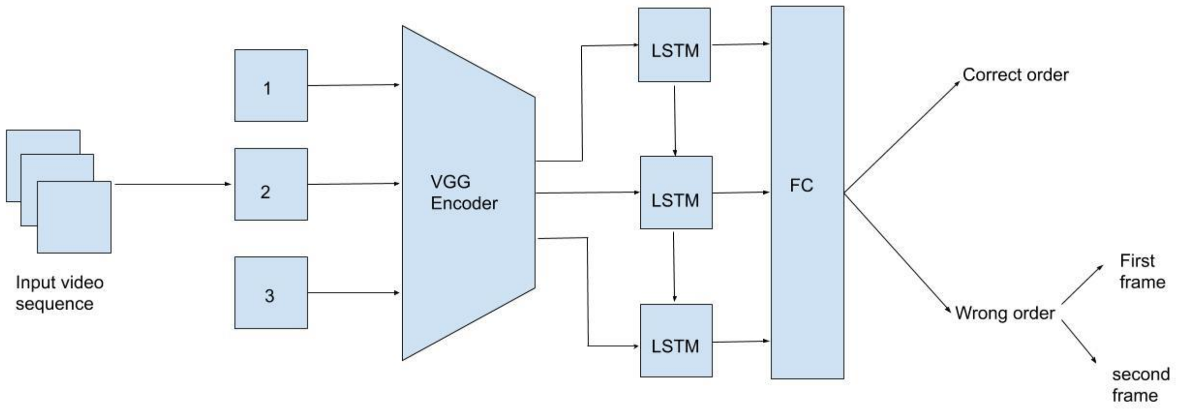


Fig 4.1 Our baseline model.

In our baseline model, we sample *N* frames from the video. These *N* frames are sampled uniformly, and we feed these sampled frames to a VGG-16 network. In our baseline model we start with the VGG-16 CNN as encoder. We remove the dense layers of this VGG-16 network and take output of final pooling layer. This layer acts as an encoding step, taking inspiration from [2].

The encoded vectors are sent as input to the LSTM network. The reason for incorporating LSTM, instead of directly sending these embeddings into a dense layer like in [3], is because of two reasons:

1. Be able to use frame sequences of larger lengths;
2. RNNs are inherently capable of modelling sequential data, unlike independently feeding them into a dense layer.

One of the limits from previous works of video representation learning using self-supervision was that the number of frames they used were small [2,3]. Therefore, our thesis focuses on ways to increase that sequence length by adding the LSTM RNN block, and also evaluate different encoding networks.

Research in self-supervised models usually test on simpler networks, without any feature engineering for the data. This is done to test the actual effect of our learning signal and not get results due to model performance. Therefore, we did not consider adding hand crafted features or additional inputs like optical flow for our network.

The output from the LSTM RNN is then fed into the fully-connected layer. Our self-supervised task is as follows – given a sequence N of video frames, the network has to predict if this sequence is in the right/wrong order. In the baseline model, we work with $N = 3$, so we add an auxiliary task to this network to stress test, and make the task harder. The network has to predict not only if the sequence is in the correct order or not, but it also has to reason how the positioning of the frames with respect to each other.

Practically, when we implement this there would be three different classes for the network with respect to the middle frame. We have to predict where the middle frame should be positioned – in place (correct order), first place or last place (wrong order). These would be three different classes with labels 0, -1 and +1 for these positions respectively.

Consider three frames sampled from the video $f1, f2, f3$. The sequence in correct order would be $f1 \rightarrow f2 \rightarrow f3$. Now we would be putting the middle frame $f2$ in either the first position or the last position, so the two different sequences could be $f2 \rightarrow f1 \rightarrow f3$ or $f1 \rightarrow f3 \rightarrow f2$. Similarly, we consider another type of frame sequence $f3 \rightarrow f2 \rightarrow f1$ which is in reverse order, but it still would be considered as correct order. The ordering of these frames is correct, but reversed in temporality. For this sequence as well, we can place the middle frame either in the first position or the last. For all of these frame sequences, the network has to predict which one it thinks the middle frame is. (Using the three different labels). Thus, by having a complex task, where the network has to reason about the scene semantics and learn relation between frames, we predict it would improve performance.

By using these classes, we can train for both different types of data using this self-supervised task, we train on split-01 of the UCF-101 dataset.

4.2 Model with higher number of frames ($N > 3$)

In this model, we increase the number of frames to be fed into the network. The drawback with this model is that the ordering task in the baseline model, where the network has to predict the positioning of the middle frame can't be used.

For getting the video frames, as mentioned in the beginning of Chapter 4. we would have all the frames for a video sample written to our disk. Out of these total frames T, we would be randomly picking N number of frames which are then fed into the network. For this around 70% of the samples are wrongly ordered, and for these we would jumble the ordering before sending them to the network.

But with higher number of frames, we postulate that the network would be able to perform better. This is because the higher number of frames we have, the better the network is able to learn without any ambiguity. With just three frames, the images can be too general, and the classes can be ambiguous. With higher number of frames, there is a lot more information the model would be able to use. Using number of frames N from 10 to 40, we conduct several experiments and see how the model performance changes with respect to frame size.

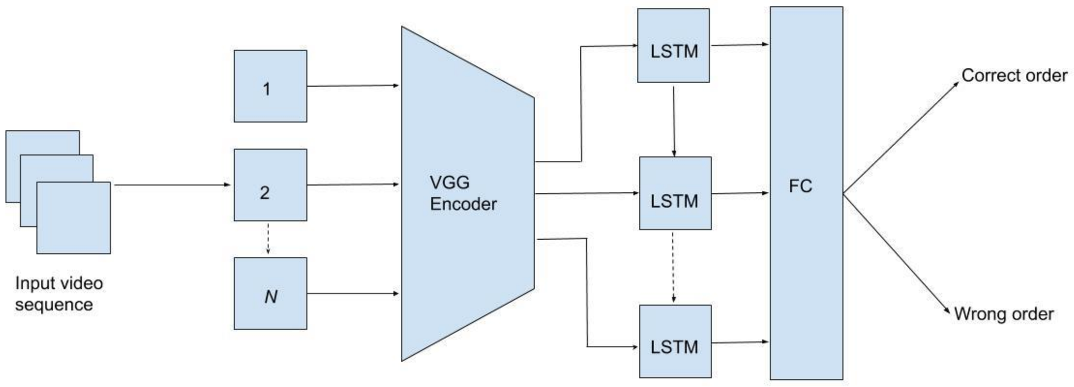


Fig 4.2 Model with higher number of frames.

In Fig 4.2 we see the updated model architecture. There are a variable number of frames *N* sampled uniformly from the video sequence. The number of classes for self-supervised training has also been reduced to two classes – is the frame in the right order or wrong order. In the results and training procedure section we outline the evaluation of this model.

4.3 Model with 3d Convolutional (C3D) Encoder (N > 40)

We also evaluate the performance of the model with a different encoder network. 3D convolutional neural network (C3D) [26] is one of the salient works in action recognition on videos. The overview for the network architecture is described in Chapter 2. We use this encoder for sequences with $N > 40$ frames.

The C3D encoder has several layers of 3D convolution and pooling, followed by fully connected layer. The C3D model originally can take in an entire sequence of video (default is 16 frames) in a single time step and process it. Since we want to leverage the LSTM network we have, and also be able to use longer frame sequences $N > 40$ we propose a different method.

In our model, we would be using the C3D network to encode n frames, where $n < N$. This step effectively reduces the length of frame sequence we give to the LSTM. This fewer number of features n would be sent to our LSTM network for training. Since this model is $N >3$, the self-supervised task would remain the same as above model. The learning task is to predict if the sequence of frames is in the right order or wrong order. We won’t be predicting the positioning of frame in the middle.

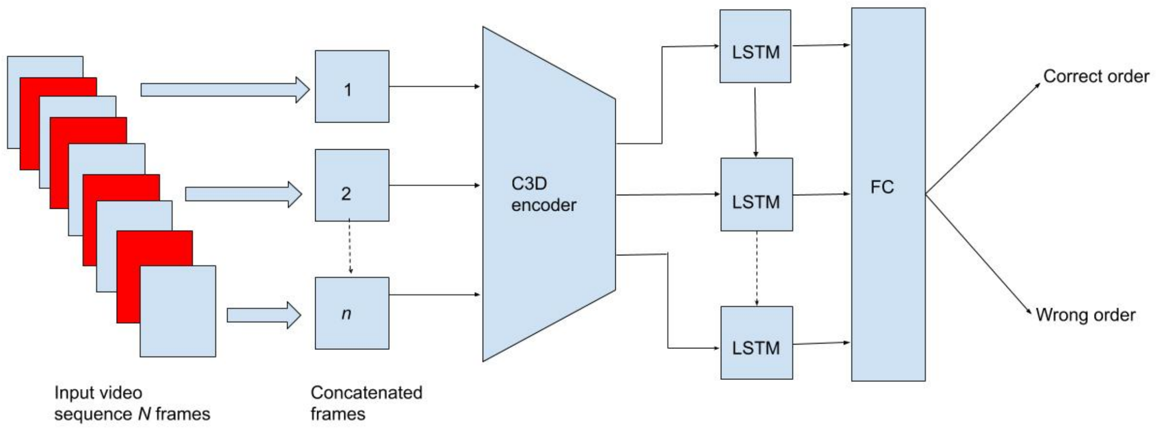


Fig 4.3 Model architecture for C3D encoder.

As seen in Fig 4.3. we have an input sequence of *N* frames, with are concatenated to form *n* inputs. These *n* concatenated frames are sent as separate 3D inputs to our C3D encoder. The value of $N > 40$, usually taking in all the frames of a video. The value of *n* is a hyperparameter in our network, usually much smaller $n <= 10$. The reason for this is because of the smaller value of *n*, the LSTM network would be able to perform better. During the supervised training, we train in a similar manner.

Chapter 5

Implementation and Results

5.1 Datasets

Some of the popular datasets for action recognition in videos are as follows:

- UCF 50 dataset [21]
- Kinetics dataset [16]
- Sports 1M dataset[34]
- Youtube 8M dataset[23]
- UCF-101 Dataset[24]
- HMDB 51 dataset[25]

The UCF- 50 dataset is one of the eariler datsets in action recognition. It was released by the University of Central Florida in 2012. It has 50 different action categories, and all the videos are divided into 25 groups with four or more video clips. The dataset comes with a train-test split, and three different splits. Since there are repeated identities in samples, it is recomendded to use their splits.

Kinetics datset is large dataset of nearly 65,000 video samples and has about 700 different types of action classes. It has a minimum class size of 600 samples per class, and has diverse samples.

The Sports 1M dataset contains nearly 1,133,158 video samples (~ 1M). These videos are shared in the form of Youtube URLs in a JSON script which can be downloaded. The labelling is not human-annotated and rather is from the YouTube tags which are generated using the YouTube topics API. Using this technique there are about 487 different labels.

YouTube 8M dataset is another huge dataset in this domain, with millions of videos from YouTube, nearing about 350,000 hours of total run-time. These also use the YouTube tags and have about 3862 different classes. These include a wide variety of not only action poses but also videos of cartoons, movies and animals.

Out of the datasets mentioned, we focus our training and results primarily on UCF-101 and HMDB 51 datasets, which will next be described below. We chose them because the size of these datsets is not too huge, and will not overwhelm the GPU resources available. The size of the dataset is also important in relation to the architecture we choose, as there exists a tradeoff between model complexity and dataset size.

UCF-101 and HMDB 51 datasets both encompass a wide varitey of actions, including human motions. We describe these datasets in depth below.

UCF -101 Dataset

The UCF-101 dataset [24] is an extension of the original UCF-50 dataset. It has 101 different action categories as shown in Fig 5.1, collected from YouTube. They are all realistic action scenes. It has a total of 13,320 videos. UCF-101 is one of the standard action recognition datasets, and has wide variation in pose, illumination, and clutter. The five different action categories are Human-Object Interaction, Human-Human Interaction, just Body-Motion, and Playing Musical Instruments and Sports.

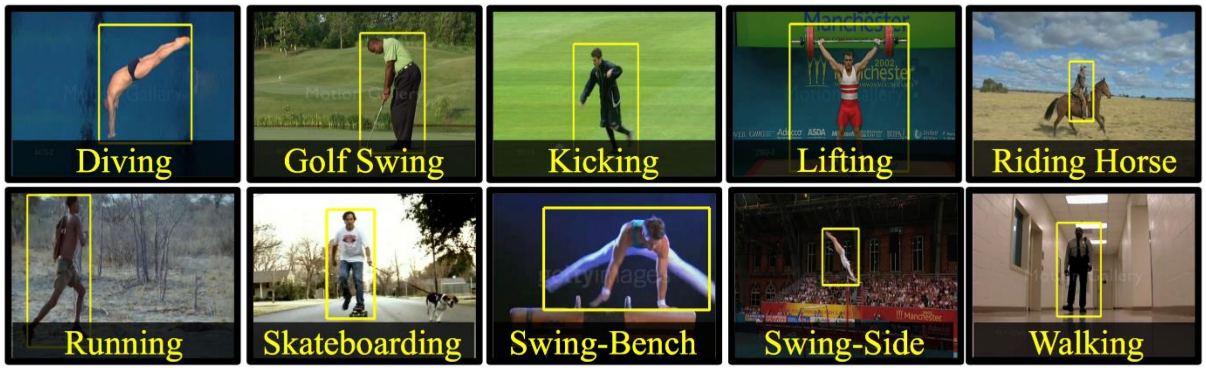


Fig 5.1.1 Example of UCF-101 dataset [24].

UCF-101 dataset has three different train-test splits. The train-test splits are divided such that there are equal division of videos between classes and no overlapping between identities in train and test sets. For our training we use split 1 of UCF-101 dataset as done by several other papers in this domain [1,2,3].

The statistics of this dataset are shared below in Table 5.1.1.

	Split 1	Split 2	Split 3
Train	9537	9586	9624
Test	3783	3734	3696

Table 5.1.1Statistics of UCF-101 dataset [24].

HMDB-51 Dataset

The HMDB-51 [25] dataset is an action recognition dataset which has 51 different action classes, and about 6,766 videos as shown in Fig 5.2. This is also a challenging dataset owing to the wide variation in illumination, camera viewpoints, video quality and occlusions. The dataset consists of general face actions and body movements like cart wheel, push-ups, and climbing.

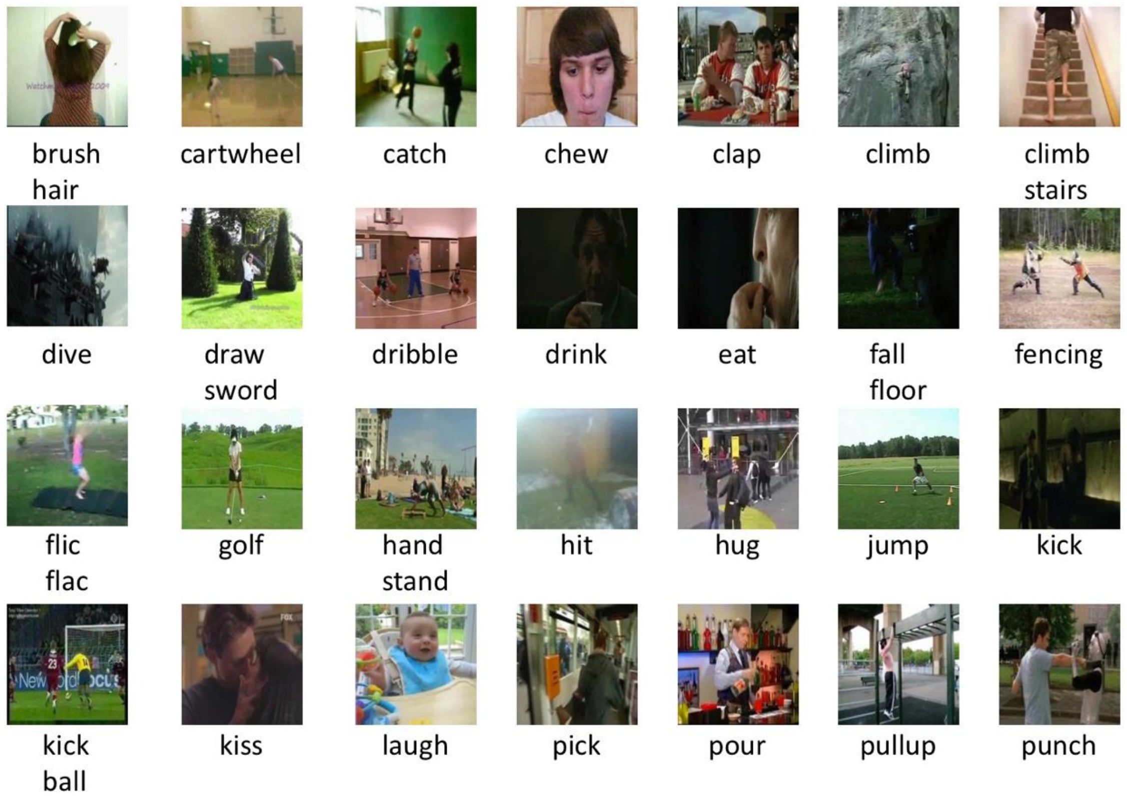


Fig 5.1.2 Example of HMDB 51 dataset [25].

The stastistics of the HMDB 51 dataset are shared below in Table 5.1.2. The three splits of the HMDB-51 dataset have the same number of samples provided by the authors. However they did not mention train/test splits, and we've used a 70:30 train:test ratio for all our training purposes.

	Split 1	Split 2	Split 3
Train	4735	4735	4735
Test	2031	2031	2031

Table 5.1.1Statistics of HMDB-51 dataset [24].

5.2 Training details

All of the code for our network is implemented in TensorFlow and Python. For the baseline model and the model with $N > 3$, we use the time distributed convolutional blocks in Keras. In this we apply a time distributed wrapper around normal convolution to add a fourth temporal dimension into our inputs. For the C3D model, we implement 3D convolutional blocks. This is used for spatial convolution over volumes, and can be used in our C3D architecture.

The training procedure is as follows:

- Train the model with our pre-text task on the dataset, and save the model. This pretrained model when used will be denoted as SS (self-supervised)
- For supervised training, we can have 3 different initializations for our model.

    - Random initialization – here we don’t use any pretrained weights, we use Xavier initialization
    - Transfer learning from pretrained model - ILSVRC -2012 trained model for VGG encoder / Sports 1M trained model for C3D encoder.
    - Using the SS initialization
- Similarly, when training for the SS pre-text task, we can have 2 initializations
    - Random initialization similar to supervised training.
    - Transfer learning from model as done for supervised model training.

To compare the performance of SS + supervised model with a supervised model (without any self-supervision), we keep the hyperparameters between the two as similar as possible when training. We describe below the specifications for three types of training: 1) self-supervised training; 2) supervised training; and 3) SS + supervised training.

Self-supervised (SS) training

For SS training, we have two classes –correct and incorrect order frame sequences. Our dataset split is 70:30, or 70% of video samples are purposely made of out of order, which the other 30% of samples remain in correct order. In order classes are those with frames not jumbled, and out of order class has frames in wrong order. We sample the frames randomly from the video sequence. There can be two initializations for SS training

SS training with Random initialization

We use an Adam optimizer with learning rate = 1e-3 and decay = 1e-6. We train for 100 epochs. We use a Xavier initialization as the random initialization.

SS training with pretrained model initialization

For the VGG encoder, we can start with the ImageNet pretrained model as the initialization. We use transfer learning to then train the model on our pre-text task. Similarly, if we use the C3D encoder we use Sports 1M dataset trained model as the initialization, and start from there.

We use similar hyperparameters here – a learning rate of 1e-3 and decay = 1e-6. We train this model for 100 epochs.

Supervised training

In supervised training, we wish to test the efficacy of our network on downstream task which is action recognition. As mentioned above we can have three different starting points for this training, described below. We use the default splits of our datasets of training. We use Adam optimizer and started with learning rate = 1e-3 and decay = 1e-6.

Supervised training with random initialization

In supervised training with random initialization, we use Xavier initialization. Here we use standard learning rate of 1e-3 for our network. We train the model for 100 epochs. These hyperparameters are not the same as those for SS + supervised training because these give better performance and accuracy compared to those.

Supervised training with SS initialization

The hyperparameters for this network are almost similar to supervised training, the only difference being that we used a much lower learning rate and higher regularization. We used a learning rate =2e-4, as we are using the pretrained SS model to help us converge. Using these settings would help in training with SS model. Using a larger learning rate makes both the networks perform similarly without any improvement with SS pre-trained model. This might be because the huge learning rate would modify the weights of the network too much and lose the effectiveness of self-supervised training.

Supervised training with pretrained model initialization

For supervised model we use transfer learning from the ImageNet pretrained model. For training this network we start with small learning rate and train the final classification layer, and then train the entire network with a similar learning rate of 1e-3.

5.3 Results

In this section we are broadly going to be looking at results for two different models- the self-supervised model, and the supervised model. We focus our results primarily on the standard dataset UCF 101. We also test our models on the HMDB 51 dataset as a secondary dataset.

Self-Supervised model results

The self-supervised model is trained on the dataset we created using pseudo computer generated labels. These pseudo labels are correct order and incorrect order, with respect to frame sequence. We train using our pre-text task, which is frame order prediction. There are two different initializations we can use for the network. Table 5.3.1 and Table 5.3.2 look at the results with randomly initialized model and compares with transfer learning from pretrained model as starting point. The tables correspond to UCF-101 and HMDB 51 datasets respectively. In these results, we also compare the different encoding architectures we've used. For number of frames, *N* = 3,8,10,48 we represent the VGG Encoder, and for C3D encoder we use a value *N* of 100. The reason with using a value of 100 is because we wanted to use the C3D architecture to encode all the frames of a video sequence. Since the video samples are variable in time length, there are different number of total frames we can sample from a video. For a consistent pipeline and training method, we chose 100 as it was the minimum frame count for any sample. As for the lower values of *N*, we experiment with different number of frames and compare the performance.

	$N = 3$	$N = 8$	$N = 10$	$N = 48$	$N = 100$ (C3D Encoder)
Random initialization	0.7092	0.7213	0.7347	**0.9214**	0.8172
Pretrained model	0. 7254	0.7620	0.7876	**0. 9321**	0.8001

Table 5.3.1 Accuracy results of UCF-101 dataset on self-supervised task.

In Table 5.3.1 we look at the results of UCF-101 on self-supervised task. The VGG Encoder model with number of frames of *N* = 48 has the best accuracy of all the models. This agrees with our hypothesis that with higher number of frames, the network performance would improve. With lower frames rates, there is an ambiguity between classes. Also, it has lower information (lower number of frames) and we also found that the model easily overfit during training.

The C3D model performs well compared to the model with lower number of frames, but it doesn't have the best performance. We hypothesize that the reason for this is because of two reasons – a complex network architecture and addition of non-trivial information from the videos. As we have higher number of frames, there are consecutive and identical frames which are fed into our network. Due to such identical frames used for training, there is a lot of repeated or sort of 'duplicated' data being sent to the network.

In table 5.3.2 we look at the results of HMDB 51 dataset. The results of HMDB 51 dataset are a bit lower compared to that of UCF101. We hypothesis that this is due to the nature of HMDB 51 dataset. This dataset is sourced from the internet and has clippings from the media and movies. Looking at the results of HMDB 51, we notice a similar trend in the number of frames and performance.

	$N = 3$	$N = 8$	$N = 10$	$N = 48$	$N = 100$ (C3D Encoder)
Random initialization	0.6714	0.6913	0.6641	**0.7792**	0.6172
Pretrained model	0.7121	0.7348	0.6976	**0.7854**	0.6301

Table 5.3.2 Accuracy results of HMDB-51 dataset on self-supervised task.

Supervised model results

For our supervised training, we have three different starting points. The random initialization, the transfer learned model and using the SS initialization. The SS model used is the one with transfer learning from pretrained model, as it gives better or comparable performance compared to random initialization for all architectures. These results are for the action recognition task, and are tabulated below in Table 5.3.3.

	$N = 3$	$N = 8$	$N = 10$	$N = 48$	$N = 100$ (C3D Encoder)
Random initialization	0.4472	0.4395	0.4737	**0.5491**	0.4210
Pretrained model	0.4971	0.4857	0.5160	**0.6473**	0.4821
Self-supervised initialization	0.4721	0.4468	0.4901	**0.6019**	0.4431

Table 5.3.3 Results of UCF-101 dataset on supervised task.

Looking at the results in Table 5.3.3, we notice that the performance of models is dependent on both the pretrained model used and also the number of frames. The best performance with respect to the number of frames used is seen with the N = 48 model. This is consistent across the three different initialization techniques. This sample rate is also better compared with a much higher number of frames of N = 100.

We postulate that this is because of the following reasons

- Using a much higher number of frames means that we have more data to process. There is also a possibility of frames being consecutive, and consecutive frames of a video sequence have very similar content, as videos don't change content or scene suddenly.
- The higher number of frames means that there is more data for the network to process, and the model complexity might not be sufficient enough to capture the information necessary.

Comparing the effect of the initialization used, we look at the random, pre-trained model and SS initializations. The random initialization seemed to perform the least out of all these models consistently. Both pre-trained initialization as well as using the SS model helped training our networks. This could be because videos are a huge dataset and take a lot of epochs to learn. In such a high complex space there is a higher possibility for our network to get stuck in saddle points or local minima. Using either pretrained model or SS initialization is a good starting point, and helps in converging faster.

Comparing the performance between the pretrained model and SS initialization, we see the best performance from pretrained model in most cases. But the SS initialization also performed considerably well, and close to the pretrained model. We hypothesize that our encoder which is trained on ImageNet is a huge dataset (> 1M images) compared to our pretrained dataset (13K videos). Nonetheless as we achieve almost comparable performance with our smaller dataset, using a larger dataset for SS pre-training might get better performance.

Finally, we look at the influence of our model architecture and encoder we have chosen on the performance. The C3D encoder has good performance compared to VGG encoder for the lower number of frames of *N* = 3, 8 and 10 but doesn't beat the performance of VGG encoder for *N* = 48. We see that there is a trade-off between the encoder architecture complexity and frame count. Using the right balance between the two gives best performance.

We also take a look at the HMDB-51 dataset in Table 5.3.4. Our hypothesis fits for HMDB 51 dataset as well. The results are comparatively smaller as HMDB 51 is not a standard dataset, and has samples sourced from media and movie clippings, but are similar to the recent self-supervised works in this domain [9,3].

	N = 3	*N = 8*	*N = 10*	*N = 48*	*N = 100* (C3D Encoder)
Random initialization	0.2711	0.2874	0.2795	**0.3011**	0.2903
Pretrained model	0.3161	0.3086	0.2903	**0.3429**	0.3020
Self-supervised initialization	0.2845	0.3128	0.3053	**0.3285**	0.2982

Table 5.3.4 Results of HMDB-51 dataset on supervised task.

We also compare our results against other self-supervised techniques in this domain. These results are tabulated below in Table 5.3.4 and 5.3.5.

Method	Random initialization	Self-supervised initialization	Degree of improvement
Shuffle and Learn [2]	38.6	50.2	11.6
Ours	**54.9**	60.1	5.2
O3N [3]	43.4	**60.3**	**16.9**
Wang et al. [17]	45.4	58.8	13.4

Table 5.3.4 Comparison of results on UCF-101 dataset with other self-supervised works.

We compare our results primarily against other recent works which have a self-supervised pre-text task, and also test their results on action recognition task for two datasets – UCF-101 and HMDB 51. We compare the performances between random and self-supervised initialization among these networks. We also propose to use a metric 'Degree of improvement' in which we analyse how much the performance has improved by introducing the self-supervised training. This is an important metric which helps us analyse how beneficial our self-supervised task is above random initialization.

We notice that with random initialization our method performs the best compared to other recent works. With self-supervised initialization we notice that the best model is O3N [3] network, followed closely by ours.

With respect to degree of improvement, we see that the O3N network [3] has the best performance. Our model doesn't have the best improvement in performance, and falls close to

Shuffle and Learn [2]. The reason for this could be the level of toughness of self-supervision task. Wang et al. [17] also performed well in this metric, and has a complex pre-text task in which the network has to predict motion and colour statistics for future frames. Similarly, O3N also has different sampling and encoding techniques which improved its performance. In Table 5.3.6. we look at the comparison of results for HMDB 51 dataset.

Method	Random initialization	Self-supervised initialization	Degree of improvement
Shuffle and Learn [2]	13.3	18.1	4.8
Ours	**30.1**	**32.8**	2.7
O3N [3]	21.8	32.4	10.6
Wang et al. [17]	19.7	32.6	**12.9**

Table 5.3.5 Comparison of results on HMDB-51 dataset with other self-supervised works.

In the results for HMDB-51 we notice a similar trend. Our network performed similarly to other techniques under random and self-supervised initialization. Under degree of improvement, we see Wang et al. [17] performs well compared to O3N, and our results are similar to Shuffle and Learn [2].

Visualizations

1) Good examples

For easy visualization purposes of the samples, the number of frames shown for the examples below are not indicative of actual *N* we used for different networks above. Looking at the good examples below, we notice that our best model (*N* = 48) was able to perform well on complex action recognitions. These actions had a lot of movement in them, and sometimes had different objects interacting with each other (For example in the Salsa dancing class there were two people dancing with each other). Some of these movements are also cyclical in nature as the push ups or pull-ups.

a. Salsa dancing

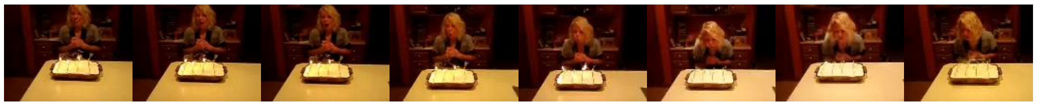
b. Pizza throwing

c. Pole Vault

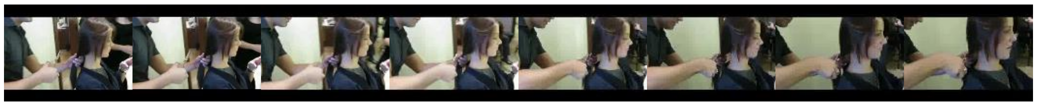
d. Pull-ups

2) Bad examples:

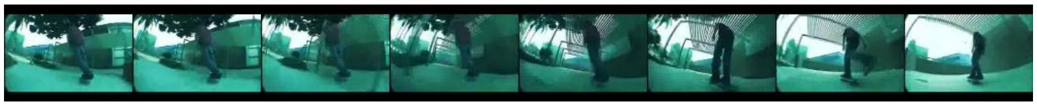
Below we show some visualizations for samples the network failed on. We notice that in these samples there could be different reasons for it. The skate-boarding example had different view point and occlusion, and few other samples had movements which were simple and not a lot of motion like haircut, walking dog, head massage, blowing candles.



a. Blowing candles



b. Haircut



c. Skate boarding

d. Walking Dog

e. Head massage

Chapter 6

Conclusions and Future work

6.1 Conclusions

In this thesis, we presented our work on self-supervised video representation learning and our pre-text task, which is frame order prediction. Our proposed architecture with recurrent networks was able to capture this temporal information, and work on higher number of frames. Using our self-supervision technique, we've demonstrated that it improved the performance compared to random initialization. We also explored different encoding mechanisms for our network, and looked at their effects on performance.

6.2 Future Work

In this thesis work we've primarily explored the effects of number of frame on network performance. The results improved with increasing number of frame *N*, but with higher rate of *N* = 100 the network performance dropped. Future research in this domain could expand on few ideas such as:

- With increasing number of frames *N*, there should also be a more complex network architecture. This is to avoid having more data than the parameters to be learned. Research in this domain can expand on replacing the C3D encoder with another video encoder such as Temporal 3D convnets (T3D) or I3D.

- Another area of interest is to explore different auxiliary tasks along with the main pre-text task - frame order prediction. Using a complex task forces the network to learn the semantic relations better, and also helps the supervised training for better performance.
- Look into adding more information such as optical flow, add attention modules into our network, and also try different sampling techniques.
- In our thesis, we primarily focus our results on the action recognition task on videos. We postulate that learned video representations from a modality could be used as a starting point for various other downstream tasks in that modality. In case of videos these domains could be video retrieval, object tracking, pose estimation and video highlight.

Bibliography

[1]. Srivastava, Nitish and Mansimov, Elman and Salakhudinov, Ruslan. *ICML 2015*. Unsupervised Learning of Video Representations using LSTMs.

[2]. Ishan Misra, C. Lawrence Zitnick and Martial Hebert. *ECCV 2016*. Shuffle and Learn: Unsupervised Learning using Temporal Order Verification

[3]. Basura Fernando and Hakan Bilen and Efstratios Gavves and Stephen Gould. *CVPR 2017*. Supervised Video Representation Learning with Odd-One-Out Networks using Temporal Order Verification

[4]. Luo, Zelun and Peng, Boya and Huang, De-An and Alahi, Alexandre and Fei-Fei, Li. *CVPR 2017*.Unsupervised Learning of Long-Term Motion Dynamics for Videos.

[5]. Chuang Gan and Boqing Gong and Kun Liu and Hao Su and Leonidas J. Guibas. *CVPR 2018*. Geometry Guided Convolutional Neural Networks for Self-Supervised Video Representation Learning .

[6]. Biagio Brattoli*, Uta Büchler*, and Björn Ommer. *ECCV 2018.* Improving Spatiotemporal Self-Supervision by Deep Reinforcement Learning

[7]. Understanding LSTMs, Blog link: http://colah.github.io/posts/2015-08-Understanding-

LSTMs/

[8]. Kim, Dahun and Cho, Donghyeon and Yoo, Donggeun and Kweon, In So. *AAAI 2019*. Self-Supervised Video Representation Learning with Space-Time Cubic Puzzles.

[9]. Jiangliu Wang; Jianbo Jiao; Linchao Bao; Shengfeng He; Yunhui Liu; Wei Liu. *CVPR 2019.* Self-Supervised Spatio-Temporal Representation Learning for Videos by Predicting Motion and Appearance Statistics.

[10]. LinkAI, blog link- https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/

[11] Ian J. Goodfellow , Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley. Generative Adversarial Nets.

[12]. Michael Mathieu, Camille Couprie, Yann LeCun. Deep multi-scale video prediction beyond mean square error.

[13]. A Krizhevsky, I Sutskever, GE Hinton. Imagenet classification with deep convolutional neural networks.

[14]. Xiaodan Liang, Lisa Lee, Wei Dai, Eric P. Xing. Dual Motion GAN for Future-Flow Embedded Video Prediction.

[15]. Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network

[16]. T Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, Andrew Zisserman. The Kinetics Human Action Video Dataset

[17]. Jiangliu Wang, Jianbo Jiao, Linchao Bao, Shengfeng He, Yunhui Liu, Wei Liu. Self-supervised Spatio-temporal Representation Learning for Videos by Predicting Motion and Appearance Statistics.

[18]. Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, Ming-Hsuan Yang. Unsupervised Representation Learning by Sorting Sequences

[19]. Xiaolong Wang, Abhinav Gupta. Unsupervised Learning of Visual Representations Using Videos

[20]. Carl Vondrick, Hamed Pirsiavash, Antonio Torralba. Generating Videos with Scene Dynamics

[21]. Kishore K. Reddy, and Mubarak Shah, Recognizing 50 Human Action Categories of Web Videos, Machine Vision and Applications Journal (MVAP), September, 2012.

[22]. Longlong Jing and Yingli Tian. Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey

[23]. Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, Sudheendra Vijayanarasimhan. YouTube-8M: A Large-Scale Video Classification Benchmark

[24]. Khurram Soomro, Amir Roshan Zamir, Mubarak Shah. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild

[25]. H. Kuehne, H. Jhuang, E. Garrote,T. Poggio, T. Serre HMDB: A large video database for human motion recognition

[26]. Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, Manohar Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks

[27]. Diederik P. Kingma, Jimmy Lei Ba. Adam: A method for stochastic optimization

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep

Convolutional Neural Networks," in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778

[30] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten, "Densely connected convolutional networks," arXiv preprint arXiv:1608.06993, 2016.

[31] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for largescale image recognition," arXiv preprint arXiv:1409.1556, 2014

[32] Zador, A.M. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat Commun* **10,** 3770 (2019). https://doi.org/10.1038/s41467-019-11786-6

[34] Andrej Karpathy and George Toderici and Sanketh Shetty and Thomas Leung and Rahul Sukthankar and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks

[35] Doersch, Carl and Gupta, Abhinav and Efros, Alexei A. *ICCV 2015*. Unsupervised Visual Representation Learning by Context Prediction.

[39] Zhang, Richard and Isola, Phillip and Efros, Alexei A. *ECCV 2016*. Colorful Image Colorization.

[40] Pathak, Deepak and Krahenbuhl, Philipp and Donahue, Jeff and Darrell, Trevor and Efros, Alexei A. CVPR 2016. Context Encoders: Feature Learning by Inpainting.

[41] Understanding LSTMs, Blog link: http://colah.github.io/posts/2015-08-UnderstandingLSTMs/

[42] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[43] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[44] Zhang, Z., Tao, D.: Slow feature analysis for human action recognition. TPAMI 34(3) (2012) 3

[45] Farneback, G.: Two-frame motion estimation based on polynomial expansion. In: Image analysis. *Springer 2003*

[46] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR. (2014)*

[47] Misra, I., Shrivastava, A., Hebert, M.: Watch and learn: Semi-supervised learning of object detectors from videos. *CVPR. (2015)*

[48] Pfister, T., Charles, J., Zisserman, A.: Flowing convnets for human pose estimation in videos. In*: ICCV. (2015)*

[49] Liang, X., Liu, S., Wei, Y., Liu, L., Lin, L., Yan, S.: Towards computational baby learning: A weakly-supervised approach for object detection*. In: ICCV.*

[50] Agrawal, P., Carreira, J., Malik, J.: Learning to see by moving. In*: ICCV. (2015)*

[51] Poppe, R.: A survey on vision-based human action recognition. Image and vision computing

[52] Vondrick, C., Pirsiavash, H., Torralba, A.: Anticipating the future by watching

unlabeled video. arXiv preprint arXiv:1504.08023 (2015)

[53] Singh, S., Gupta, A., Efros, A.: Unsupervised discovery of mid-level discriminative patches. *ECCV (2012)*