THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER ENGINEERING

Extending Action Recognition in the Compressed Domain

SAMUEL ABRAMS
SPRING 2022

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Engineering

Reviewed and approved* by the following:

Vijaykrishnan Narayanan
Robert Noll Chair Professor
Thesis Advisor / Honors Advisor

John Morgan Sampson
Associate Professor
Faculty Reader

* Electronic approvals are on file.

ABSTRACT

As the internet continues to take root in every facet of society, video is becoming one of the most common mediums for the communication of ideas and information. As of 2021, video traffic makes up 81% of all internet traffic worldwide. It is, therefore, clear that being able to understand video autonomously and on a mass scale provides huge potential for technologies that will shape the future. This could immediately impact the entertainment industry, surveillance, data collection, human-computer interaction, and more. Much work has been done in an attempt to take the successful strategies that convolutional neural networks have had on image classification and apply them to video understanding. One instrumental approach leverages the inherent redundancy in video by utilizing the existing format in which video is commonly stored: its compressed state. By learning directly on compressed video, models are exposed only to the most pertinent spatial information as well as motion information that is present without any computation. While the present methods generate near-SOTA results, there are many areas for improvement. All notable works in this topic have been performed on video compressed with the MPEG-4 part-2 codec which is dated in part due to its non-optimized compression ratio. An important contribution made to this discipline was showing that using this non-optimized codec the abundance of uncompressed frames allowed for complete disregard for the motion information in the stream using a process of temporal shifting between the spatial frames. This generated improved accuracy with system energy savings of over 11x over the traditional method. Another contribution has been modifying the methods of compressed recognition on the archaic aforementioned

codec and applying them to the modern, highly-optimized H.264. With this codec the standard for platforms such as YouTube and Twitch, developing a method for performing recognition on H.264 directly is of immense importance for applying video understanding in the real world. Through my exploration of strategies for processing video in this format, I have developed a method that provides comparable accuracy to the methods on MPEG-4 part-2 even given the constraints of utilizing H.264 directly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor Dr. Vijaykrishnan Narayanan for supporting me throughout my time working in the department. I learned so much about AI systems gaining exposure in the design, development, and deployment stages. I would also like to thank my fellow lab mates for always being there to help support and guide me. In particular, I would especially like to thank Sahithi Rampalli, Nagadastagiri Challapalle, and Abhijeet Kumar for all of the advice and support they have given. I would also like to thank my friends and family for their constant support and guidance. I truly would not be here without them. This work was supported in part by the DARPA/SRC JUMP program. The findings and conclusions are only of the researchers and are not those of the sponsors.

Chapter 1

Video Compression

Compression is a key component to the prominence of video in today's online ecosystem. From storing hours of content on DVDs in the mid 1990s to the real-time 4k streaming that is becoming the standard today, there has long been a desire to shrink a video file to the absolute limit before quality is compromised. Modern codecs are able to reduce the bitrate of a stream by a scale of hundreds or even thousands under ideal conditions. Having a solid understanding of the methods utilized for compression along with their strengths and flaws is essential for exploring action recognition in the compressed domain. Though H.264 is the standard today, more experimental codecs will also be evaluated. There are immense benefits in having the ability to work with the most efficient codecs as supporting industry standard formats allows the method to be applied to video directly obtained from popular sources and the high compression ratios make method deployment more efficient.

Background

There are several key methods and technologies introduced in the mid 20th century that led to the ability to compress data to an impressively efficient degree. I will outline some of the key contributions to the discipline before going more in depth into how their application advanced the field of compression.

Intercoded and Intracoded Frames

In general, video compression relies on the principle of storing a key image along with offsets from that image to represent the frames that follow. From these ideas came the terms intracoded and intercoded frames. Intracoded frames are self-contained and intercoded ones rely on a reference to reconstruct the original image. This allows redundant information to be omitted from intercoded frames and significantly reduces the stream size. The idea was introduced in 1929 and was used for analog video. In 1959, temporal compression was coined by producing a set of evenly spaced key frames throughout a video. The Moving Picture Experts Group (MPEG) brought this method into modern codecs and coined the terms I-frame (Intracoded), P-frame (Predictive), and B-frame (Bidirectional). Each frame in a compressed stream is one of these three types. I-frames are self-contained and can be reconstructed without any references. P-frames reference a previous frame in the encoded stream for motion compensation. B-frames reference one previous and one future frame for their references. The goal of motion compensation is to represent the interframes with as little storage as possible by finding similar regions in the reference frame and storing the offset to locations of the interframe. To achieve this, each image is first broken up into macroblocks, generally, of size 16x16 pixels. For a P-frame, each macroblock scans its reference frame for a similar macroblock within a certain range. When the most similar block is determined, the distance is calculated and the motion vector for the P-frame's macroblock is determined. For B-frames, the source of the motion vector is recorded (either past or future) as both references are scanned to find the closest-matching block. The pixel-wise difference

between the macroblock and its corresponding block in the reference frame is called the residual error. The goal when selecting a macroblock from the reference frame(s) is to minimize the residual error which increases encoding efficiency. A residual that is mostly zero will be able to be compressed further than a residual with many non-zero values. This is why B-frames increase compression rate; they are likely to find a closely matching block. Frames are segmented into a group of pictures (GOP) which begins with an I-frame and contains all of the P and B-frames until the next I-frame. Originally, codecs often used fixed-length GOPs within a video while newer ones, H.264 for example, have varying I-frame spacing within a compressed video stream. The relationship between intraframes and the various types of interframes are some of the keys that make compression possible. How effectively they are used defines how optimal a codec can be.

Discrete Cosine Transform

The introduction of the discrete cosine transform (DCT) in 1972 was a major step toward modern compression techniques. The DCT converts a set of values to the frequency domain and is similar to the discrete Fourier transform. The DCT, however, is real-valued and is much more efficient to compute digitally. This operation is able to greatly reduce the size of an image due to the property that much of it usually consists of low-frequency information. The transform will generally produce a few low-frequency coefficients with non-zero values and large stretches of zero-valued high-frequency coefficients which are efficient to store with run-length encoding. In 1980, the motion-

compensated DCT was proposed for video compression. The standard is an 8x8 DCT which performs the transform on each 8x8 block of pixels. Because the low-frequency information is much more important to the representation of the image, a quantization matrix is used to grant more weight to the upper-left DCT values while setting as many high-frequency signals to zero as possible. After this, the values are extracted from the matrix using zig-zag ordering to select the low-frequency values first and group together the high-frequency information, which contains many zeros. The DCT is a staple in digital compression and is also used in JPEG compression.

Color Space and Chroma Subsampling

Another underlying principle of compression is the color space and sampling scheme used. It is standard practice for digital images to be converted to the YCrCb space, as the RGB space is redundant, making it expensive to transfer and error prone. YCrCb is a good approximation and allows the luminescence (Y), or luma/grayscale component, to be stored with high precision and chroma components (Cr and Cb) can be subsampled without affecting image quality. Subsampling selects chroma components at varying rates. 4:4:4 would be considered no subsampling and 4:2:2 would retain all of the luma information and sample every other row of chroma values. 4:2:0 is the standard subsampling rate and means that all luma information is retained while one out of four chroma components is selected for red and blue. The three sampled channels can be handled separately with each undergoing quantization and a DCT application. The size can be further reduced by applying Huffman encoding to the transformed matrices.

Early Video Codecs

With the initial tools and technologies developed, attempts began to create a commercially viable codec. The first successful standard was H.261 which was released in 1990. It used an early version of motion prediction and residual calculation along with a DCT and quantization. However, it did not encode audio and was more focused on efficiency than quality. After the release of JPEG for image compression in 1992, Motion JPEG was devised as a method of video compression. It simply compressed each video frame as a JPEG, so there was no motion compensation. This caused the compression ratio to be lower, but made editing faster as a frame could be decoded and modified without decompressing the entire stream [10]. MPEG-1 extended on H.261 with audio support but did not have the capability to compress interlaced images which are widely used. Interlaced video displays every other horizontal line of pixels from the top to the bottom of the frame, so it takes two passes to show the whole image. This doubles the effective frame rate. MPEG-2 (H.262) was designed in large part to add this capability and was released a year later in 1993. MPEG-2 supports standard definition (720x480) and high definition (1920x1080) resolutions. It utilized higher bitrates than MPEG-1 and became the dominant codec for DVDs and set-top boxes and is used for digital television broadcasting. H.263, released in 1995, was designed for very low bitrate as would be needed by mobile devices and for video conferencing. It had improved transform coefficient coding and quantization, increased motion estimation precision, and an improved rate-control loop. These codecs laid the groundwork for today's most common encoding schemes.

MPEG-4 part-2 and H.264

The early codecs discussed applied many foundational ideas of compression to the digital space and became widely used for television and DVD. As the turn of the century approached, though, the focus of compression shifted to a new medium: the internet. With its release in 1999, MPEG-4 part-2 played a major role in advancing the feasibility of video playback that is both high-quality and lightweight. The codec has 21 profiles that range from quality suitable for security cameras up to HD video. The "simple" profile is generally used for applications where hardware or bandwidth is limited. The following profiles, such as "advanced simple", adds support for B-frames and interlaced video. The standard introduced support for quarter-pixel motion compensation which allowed for unprecedented quality by removing artifacts from motion vector determinations, but it did increase the bitrate putting strain on the decoder. In general, MPEG-4 part-2 had a large amount of customization, so it was able to be used for many purposes. This helped the codec become widely used for internet applications. Motion vectors are set to 16x16 pixels, so there is a chance that motion can be blocky with artifacts when fine-grain motion needs to be represented. Also, regardless of the compression profile or the number of B-frames used, streams encoded with MPEG-4 part-2 have a fixed-length GOP. This is also true for MPEG2 and prior codecs, and it means I-frames will occur at a constant frequency regardless of the content being encoded. That means a video of a completely unchanging image will still require a memory-expensive I-frame every dozen-or-so frames (GOP length will vary by video depending on the number of B-frames used but is usually 10-12 frames). Fixed-length

GOPs are memory inefficient and can potentially miss frames if there is a scene change within a GOP, though this would only lead to minor visual disturbances. The GOP structure of MPEG-4 part-2 can be seen in Figure 1-1. H.264 attempted to improve upon several of the shortcomings of its preceding codec.

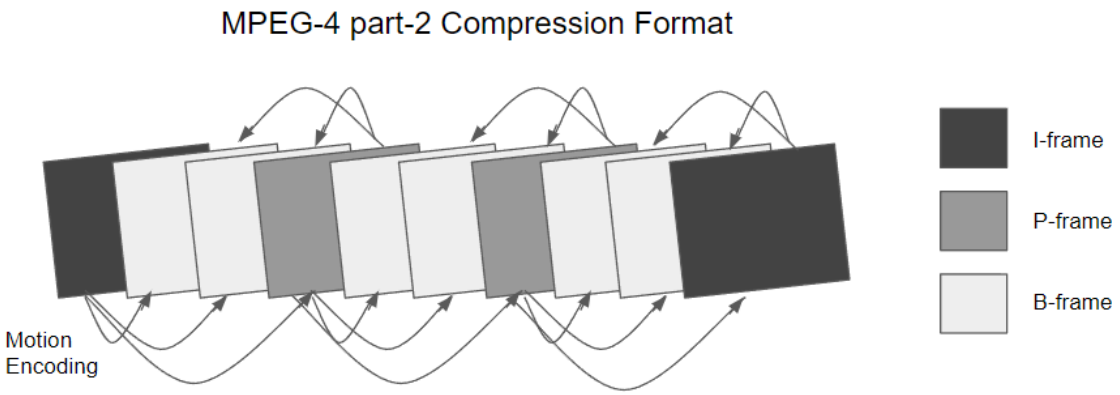

Figure **1-1**: MPEG-4 Part-2 Format

H.264 was released in 2003 and provides many additional features that increase the compression rate over its predecessor. First of all, H.264 supports variable-length GOPs where the size is determined dynamically based on the video content. If there is a similar background persisting for two-hundred frames, for example, then a single I-frame could be followed by P and B frames for the stretch. Later in the video, if the scene changes quickly several times a second, there could be an I-frame every few frames. This flexibility when encoding allows much of the redundant information to be excluded while retaining all scene changes. Motion compensation is enhanced by supporting motion vectors of varying

sizes. Hierarchical prediction begins with a 16x16 macroblock and can further subdivide it into 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 pixel sized partitions. This allows the capture of fine-grain information when it is necessary. The use of reference frames has also been updated. In MPEG-4 part-2, P-frames reference the previously decoded I or P-frame and B-frames reference the previous I or P-frame and future I or P-frame. H.264 opens up much more flexibility for assigning reference frames. A P-frame is said to be composed of P-macroblocks. Each P-macroblock can reference a different previous frame from the GOP. Similarly, B-macroblocks in a B-frame can reference any previous and any future frame. Combined with macoblock partitioning, dynamic referencing opens up possibilities for finding more similar blocks which creates very small residual errors.

Future/Experimental Codecs

The current state-of-the-art produced by MPEG is H.265, or High Efficiency Video Codec (HEVC). It offers up to 50% improved compression over H.264 and supports 8k resolution [15]. HEVC extends on macroblocks with more flexible and efficient Coding Tree Units (CTU) which are further broken into Coding Units (CU). A CTU can be as large as 64x64 pixels. Each CU can be intra-predicted from other units within the frame or inter-predicted from other frames. CUs are partitioned into Prediction Units (PU) which are inferred using intra or inter prediction. So, the larger maximum CTU size reduces bitrate while enhanced motion compensation allows for high decoding quality.

VP9 is a newer codec developed by Google with many similarities to H.265. The goal is likewise to reduce bitrate by 50% while keeping quality high. It uses blocks similar to CTUs which also are 64x64 pixels by default and are divided into sub-macroblocks. Intra and inter-prediction is used for frame reconstruction. In general, H.265 is about 20% more efficient than VP9, but the latter is open-source and does not require royalties [20] making it a widely-used alternative.

Chapter 2

Action Recognition Techniques

With the rise in prominence of convolutional neural networks over the past decade, the field of computer vision has been rapidly advancing. This technology has brought immense success to image classification with models achieving accuracies never before thought possible. There have recently been attempts to transfer the methods used on images to perform video classification. However, the jump from two dimensions to three has proven to be a challenging obstacle in the goal to understand video. This is due in part to its inherent large size and redundancy. In this chapter, I will discuss some of the prominent methods used to learn features in and ultimately classify video inputs.

Recognition in Uncompressed Video

Uncompressed video is simply treated as a series of RGB images as would be displayed in a standard video player. It is therefore evident that this input would be fairly large in size and will likely contain repeated information from frame to frame. Even with these drawbacks, several successful methods have been proposed which obtain acceptable results on challenging datasets. There is often a distinction in approaches between ones that learn spatial and temporal features separately and combine them via fusion and models that learn these features together. I will give an overview of several of the most influential methods.

Two-Stream Methods

One popular approach to address the extra dimension present in video inputs is to use two networks that have their results fused either during or after training. Often one network is used to learn spatial features from RGB images while the other learns temporal patterns through the use of generated optical flow. Optical flow is an approximation of the real motion within a video that accounts for the camera and viewing perspective. It uses the pixel intensity, brightness, to track the movement of pixels from frame to frame. It can provide a great approximation for the actual motion a video represents but is expensive to compute. One of the original works using this method [18] used late fusion and performed the best when using a linear support vector machine (SVM) to cluster the softmax outputs. The optical flow values, generally float displacements, were scaled to 0-255 valued ints and the horizontal and vertical components were stacked to create two-channel flow images. Based on this work, exploration was done into fusion techniques as it was argued that more communication between the streams could prove beneficial. Feichtenhofer et al.[19] introduced residual paths between the streams that allow spatiotemporal features to be learned. These paths between streams act like shortcut/residual connections in a ResNet. A skip connection is added from the temporal to the spatial network so that a combinatory set of features is learned during backpropagation. The methods used gained impressive accuracy and have helped to motivate continued work with the two-stream method.

3D CNNs

One of the most intuitive ways to represent learning in video is by inputting the entire stream of frames to a feedforward network. 3D CNNs. One of the earliest of such models, developed by Shuiwang et al. [5], was a relatively shallow network with seven layers (3 convolutional) accepting 7-frame clips. This work developed 3D convolutions which used a cube-shaped kernel to move both spatially across a frame and temporal along the temporal dimension. This work was extended by Tran et al. [6] in the development of C3D which is a deeper network with 16 layers, 8 of which are convolutional. They found that relatively small 3x3x3 kernels performed best, and the model obtained good results for the time (85.2% on UCF-101). 3D networks, however, are time consuming to train and require a very large dataset to generalize effectively. Methods for pseudo-3D networks began to arise. In 2018, Carreria et al. [2] proposed I3D. It used inflation to convert a 2D network to 3D by expanding each filter and pooling kernel by adding a temporal dimension. This was done by just copying the kernel and stacking it by the depth of the temporal dimension. Its major contribution was using a method to inflate/bootstrap 2D weights to 3D ones by exploiting the linearity of network weights and repeating the weights of the filters from models trained on an 2D image network. This allows pre-training for a 3D net to be done on 2D models which greatly speeds up the training process. The method used two inflated CNNs, one for RGB frames and one for optical flow, so it doubles as a two-stream method. I3D has an advantage over conventional 3D CNNs due to the inexpensive transfer learning it employs, but it requires costly optical flow calculation in pre-processing and full 3D kernels when

training. To combat the inefficiency of training a 3D CNN, several methods were presented which broke down 3D kernels to increase efficiency. R3D [7] and R2+1D [8] are examples which use the principles of separating a 3D kernel into a 2D convolution over the spatial domain followed by a 1D convolution over the temporal one to learn more nonlinearities in a shorter training window.

Temporal Segment Network

The redundancy in video slows down training as much of the information in temporally adjacent frames is repeated. The Temporal Segment Network (TSN) [16] is an early attempt to model long-range features in video with a lightweight architecture. Each video is split into N segments and from each a short clip is randomly selected. Each segment is operated on by both a spatial and temporal convolutional network. The spatial network is fed a random RGB frame from the clip and the temporal network is given a set of optical flow frames from within the clip. The spatial scores and temporal scores are fused respectively before determining a consensus from the two outputs. The results from TSN were strong with 69.4% and 94.2% on HMDB-51[21] and UCF-101[22] respectively. The method can be considered a two-stream approach, but the method of applying separate networks to each portion of the clip places it in a unique category. There is a large amount of randomness in this method present in the selection of frames from each segment. This works against overfitting as a video can be represented in many different ways based on the random selections. The number of segments also has the potential to affect the performance. More segments means a more fine-grained sampling

of the video meaning more detail can be extracted, but it could also lead to unnecessary computation and slowed learning. Because each clip is split into the same number of segments across the dataset, longer videos will have segments containing more frames than for shorter ones. This is an area that can be improved to perform better on videos of non-uniform length. The authors of TSN used a constant N=3 segments and a variation of this count is not discussed.

Temporal Shift Modules

A Temporal Shift Module (TSM) [12] is a neural network component that attempts to allow a 2D network to learn spatio-temporal features in video like a 3D CNN would. It processes clips using the segmentation method from TSN selecting 8 frames per video. Throughout the network, shift units move some channels from each frame to the next and previous frame in time. This effectively creates free motion information that can be used to predict action recognition from a 2D network. The authors presented an offline method which shifts channels forward and backward in time and an online method that only shifts frames backward.

Compressed Video Recognition Motivation and Methods

Working with raw video frames will often be expensive in terms of storage and computation. Redundancy between temporally local frames can slow learning and cause subtle patterns to be missed. It also generally takes a long time to optimize a network on

uncompressed frames. Learning directly on compressed video alleviates these issues because data in this format is designed to be concise and highlight the important changes occurring in a scene. Working in the compressed domain has gained popularity in recent years and several works have achieved competitive accuracies using models exponentially smaller and more efficient than those used for uncompressed video. All methods currently published to my knowledge utilize video encoded with MPEG-4 part-2. I will give an overview of some of the landmark works of the discipline.

CoViAR

The first work to perform action recognition in the compressed domain was developed in 2018 as a model called CoViAR [1] (Compressed Video Action Recognition). The approach used was to extract the 3 compressed features from a compressed stream (I-frames, motion vectors, and residuals) and apply a separate 2D CNN to each. Because each component is treated as a set of images, the models can use pretrained weights from training on ImageNet to achieve faster convergence. The three models are trained and inferred from independently and each captures a specific aspect of the video. Also, in order to simplify the dependency of encoded motion information, B-frames are not used for encoding which is the default for MPEG-4 part-2. So, each GOP contains 1 I-frame and 11 P-frames. Each P-frame contains motion information corresponding to the preceding I or P-frame. In order to remove this dependency chain, the accumulated motion vectors and residuals are determined. These are calculated by summing up all of the motion vectors/residuals that came before up to the GOP's I-frame

and adding it to the current P-frames motion vector/residual. This makes the motion information more robust and allows P-frames to be processed by the model independently regardless of their position. The I-frames, motion vectors, and residuals are extracted from an MPEG-4 part-2 compressed video through a C PyObject called load. C ffmpeg libraries, such as libavcodec and libavutil, are used to extract certain elements from the stream. The load function takes five parameters: a video path, GOP index, frame index, representation (I-frame, mv, res), and an accumulation boolean. A GOP index of 3 and frame index of 7, for example, would return the 7th frame of the 3rd GOP. If accumulation is true, the motion vector or residual will be summed up to the previous I-frame, and if it is false, the offset to the previous frame will be returned. For I-frames, this has no effect. To extract an RGB array from an I-frame, avpicture_fill is used to decode the AVFrame from the stream. For obtaining a motion vector array, frame side data has to be read. Initializing an AVMotionVector object allows the x and y coordinates to be read and a temporary motion vector array can be filled. If accumulation is set, each motion vector in the GOP, starting with the first P-frame, needs to be added to determine the cumulative motion at the desired frame. To achieve this, a pair of arrays is used to store where the positions of the referenced I-frame would be located in the target P-frame. The array will be copied and when the next motion vector is read, it will use the copied array as the base to which the offsets are applied. When the target P-frame is reached, the final offset of each pixel from the original I-frame is calculated and an array of shape w x h x 2, meaning that each pixel in the image has an x and y offset corresponding to the location in the reference frame from where it was obtained. Residuals are calculated by first generating that same motion vector array. Then, the

motion-compensated reference frame is compared with the actual P-frame to determine the error at each pixel. This produces an array of shape w x h x 3. It is important to note that this method of obtaining the residual values is indirect as they are not extracted directly from the decoder. FFMPEG only provides support for extracting motion vectors from the stream, so these must be utilized to find the approximate residual values for each macroblock.

The other major functionality present in the CoViAR module is the ability to get the number of GOPs and frames in a video. This is done by decoding each packet and checking the type of frame present with an I-frame updating the GOP count and the frame count always updating. Because the entire stream must be decoded, optimizations can be made by using preprocessing to extract the frame count of each video only once retrieving it from a text file.

The I-frame model is a deep convolutional network, ResNet-152, and is meant to learn the spatial features of the clip via the RGB images in the compressed stream. The network used for motion vectors is ResNet-18 which attempts to learn the temporal information. Residuals are also processed on a ResNet-18 model and provide a useful mixture of spatial and temporal information for the network to learn. To sample the videos for frames to utilize, a modified TSN was used. For each compression modality, the video was broken into N segments and from each a component of the expected type (I-frame, mv, residual) was selected. If the data loader was selecting I-frames, a random frame was chosen from the segment and it was rounded to the closest I-frame, which was a frame number with a multiple of 12. For mvs and residuals, if a multiple of 12 was selected, the frame number would be incremented by 1 to select a non-I-frame. Like the

TSN paper, CoViAR used N=3 segments for training. Once each model has been trained and tested, late-fusion is done to take consensus between the predictions for each video to determine a classification result. The datasets used were HMDB-51 and UCF-101 which generated accuracies of 59.1% and 90.4% respectively, which is impressive given that is reduced the number of GFLOPS by nearly 10x that of C3D, for example, which obtained only 51.6% and 82.3% accuracy respectively. This method created interest in compressed domain action recognition.

MFCD-net

Recognition in the compressed domain was revisited with MFCD-net [3] in 2020. It was based on the premise of the MultiFiber Network [12] which is a method that decouples kernels and connects them in a series called a fiber. Multiplexer modules are added which can selectively create a residual branch between parallel fibers so that information is shared during backprop. The method can be used for both 2D and 3D networks. MFCD-net applies the optimized 3D architecture to the compressed domain. In it, a pseudo-decompressed video clip is created for each GOP of the MPEG-4 part-2 stream. The GOP's I-frame is made the base of the clip and each accumulated residual taken and has it's RGB values averaged with the I-frame to create an approximation of the uncompressed frame. This method creates many inputs from a single video which is useful for training but may hurt accuracy due to the small temporal window (12 frames = 0.5 seconds for HMDB-51) which would not allow the motion to fully develop. Still,

though, MFCD-net achieved strong accuracies with 96% on UCF-101 and 74.6% on HMDB-51.

Analysis in the Frequency Domain

Generally, recognition done in the compressed domain is performed on partially decoded data that is beyond the frequency domain and is in the RGB space. However, a recent work [4] learning is done on the DCT coefficients which are in the frequency domain. The motivation was that images in the RGB space must be downsampled to be fed to a neural network which causes sharpness to be lost and specific features to be missed. Frames are converted to YCrCb space DCT components and only important channels are retained. To determine the importance of the channels, a dynamic gate module is used to perform binary classification on the channels. The module can be learned based on the outcomes of its previous selections. Static channel selection was also tested which had a higher probability of selecting luma channels with low frequencies. Reducing the number of channels made the model lighter and more efficient while still obtaining competitive accuracies.

Chapter 3

Optimizing Action Recognition on MPEG-4 part-2

One of the limitations of MPEG-4 part-2 is that its fixed-length GOP causes redundancy because many full RGB images are stored. A recent work done in uncompressed domain video analytics called Temporal Segment Modules (TSM) [11] adds blocks to a feedforward network which shifts sections of the feature maps between frames being processed to create pseudo-motion information. I argue that given the prevalence of RGB frames in MPEG-4 part-2, these TSMs can be applied to an I-frame network and completely remove the need for processing motion vectors and residuals.

Motivation

With the redundancy of MPEG-4 part-2, it can seem unnecessary to utilize three separate networks each acting on a different modality of the video. I argue that there is sufficient information contained solely in I-frames in the compressed domain and motion vectors and residuals can be ignored if the network architecture is modified. Being able to extract temporal information from the abundant spatial frames would allow for classification using a single 2D network which would be a major improvement over the multi-stream or 3D alternatives in terms of training and classification time. It would also prove useful for computation on edge devices with limited energy capacity. The utilization of TSM units allowed for zero-cost motion information to be extracted from image frames. While the method was originally applied to uncompressed video, there is reason to believe that it would translate well to the compressed domain. I-frames can be

treated just like uncompressed frames and using MPEG-4 part-2, they occur every 12th frame. With the utilized datasets having videos between 25 and 30 frames per second (fps), there is sufficient information present directly in the compressed stream to infer on all parts of a clip. The claim is that inserting channels from an adjacent I-frame to the current one can provide information about the motion present in the video without requiring multiple channels or pre-computed optical flow.

Approach

The idea behind the improvement to CoViAR was to modify the spatial ConvNet working on I-frames by adding temporal shift modules to allow for the sharing of features between frames creating temporal information. The method was called SpatioTemporal Action Recognition (STAR) and the architecture is shown as Figure 2-1. Each video is broken into segments and from each a single I-frame is selected. These frames are stacked together to run through the network, as is done with CoViAR. For the model, a ResNet-50 base is employed and Temporal Shift layers are added. Basically, these layers act as wrappers for some of the convolutional layers. They take a 4D input (segments/frames x channels x width x height) and shift some of the features between the frames. The shifts are done in a residual branch which allows all of the spatial information to be retained while learning motion features in parallel. When applying this approach to inputs in the compressed domain, the translation was clean with the inputs from I-frames corresponding well to the network layer inputs. Bidirectional shifting was used with 1/8 of the features being shifted left and exchanged with the previous frame

and 1/8 right with the next frame. It was also experimented shifting 1/4 of the channels forward and backward, effectively having each frame share half of its channels. The number of segments was adjusted as well which was predicted to affect performance based on the length of the video. Number of segments tested was 2, 4, 8, and 12. Since videos on the two tested datasets range in length from under 2 to over 10 seconds, it seemed like the selection choice would dictate performance on videos based on their length. This also caused interest in the idea of dynamic segmentation to be introduced where the number of segments would vary based on video length. This would be implemented in future work.

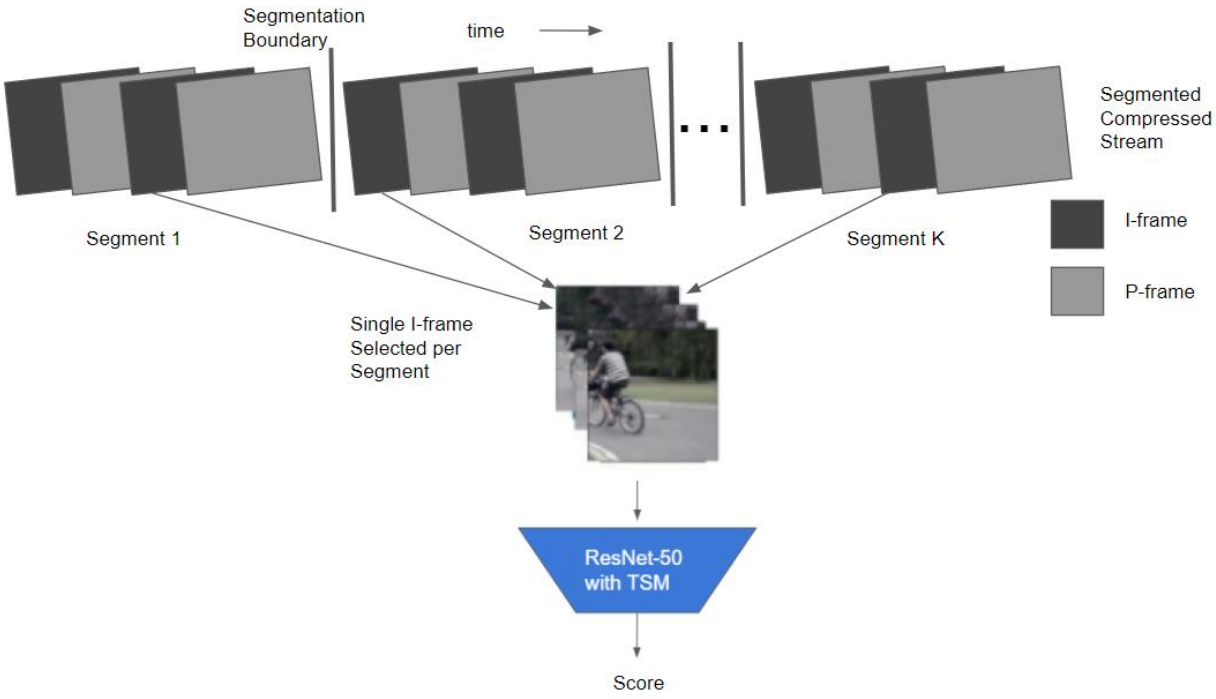

Figure **2-1**: STAR Architecture

Results

The approach of applying TSM units to spatial frames in the compressed domain generated improvements to performance while introducing slight improvements to accuracy on HMDB-51 and UCF-101 over CoViAR and performing similarly to SOTA models. The effect of the number of temporal segments on the accuracy of the classification and energy consumption were also observed and discussed.

Accuracy

The model was tested on the HMDB-51 and UCF-101 datasets. The temporal shifting approach was used with different segment counts to give insight into the relationship between the number of I-frames sampled and shifted and the accuracy of the model. The outcomes can be seen in Table 1-1. The best results for HMDB-51 were achieved using 8 segments which produced 62.51% top1 accuracy and with 12 segments for UCF-101 with 92.47% accuracy with 1/4 channels shifted. The difference in optimal segment count for the different datasets makes sense when comparing their properties. HMDB-51 has an average 93.5 frames per video which, with an I-frame every 12 frames, has about 7.8 unique I-frames per video, and UCF-101 has 186.7 frames making 15.5 I-frames per video. However, UCF-101 has a higher frame rate than HMDB-51, 30 fps to 25 fps, meaning that each I-frame in UCF contains less information change than for HMDB. So, it is not surprising to see a segment count that is closest to the dataset's average I-frame makeup gives the best performance. In general, using a lower selection rate would cause too few frames to be selected and selecting more frames than are

present in the stream would cause duplicates to be selected which clouds the data and hampers learning. The results did not vary much for any model using 4 segments or more, showing that segment count only needs to generally estimate the structure of the dataset and does not need to exactly match it to achieve SOTA results. The ability to surpass CoViAR and other action recognition methods (Table 2-1) shows the efficacy of the approach on accuracy.

Table **1-1**: STAR Architecture Testing Results by Segment Count

Dataset	Segment Count	Accuracy (%)	Processing Time (s)	Energy Consumption (J)
HMDB-51	2	59.74	35.96	830.67
	4	64.44	42.79	988.45
	8	64.51	72.62	1677.52
	12	64.11	104.1	2406.56
UCF-101	2	85.7	93.07	2149.96
	4	91.25	116.33	2687.14
	8	92.02	224.65	5189.52
	12	92.47	305.77	7063.31

Table **2-1**: CoViAR Accuracy and Testing Efficiency (*reported results [1])

Dataset	Compressed Element	Element-wise Accuracy (%) *	Overall Accuracy (%) *	Processing Time (s)	Energy Consumption (J)
HMDB-51	I-frame	52		213.89	
	MV	40	59.2	232.04	21,911.45
	Residual	43		256.36	
UCF-101	I-frame	87		610.15	

	MV	70	90.5	590.26	58,562.09
	Residual	80		676.58	

Efficiency

One major improvement of STAR over other compressed domain approaches is that it utilizes only a single network without the need for separately processing motion information. Whether this is utilizing motion vectors or residuals in separate models or pre-computing optical flow, processing motion information adds significant computational overhead to any approach. To compare the energy consumption of the STAR method to CoViAR, which is likely similar to other multi-stream methods, the Linux “perf” command was used to measure the power draw on the single Nvidia Titan V gpu used for validation. The time taken for each model to complete inference on the same dataset was also recorded. The energy consumption was then calculated. For STAR, the average power draw during inference was 23.1 W while for the three CoViAR models, it was 31.2 W. Because CoViAR processed three components separately, it would sustain that power draw for a longer period leading to greater energy consumption. The inference was performed on HMDB-51 split 1. For inference, CoViAR uses 25 segments for each video in all 3 modalities, so each stream takes longer to compute as well. Using my testing methodology, STAR utilized 1,678 J and 5,190 J respectively while CoViAR used over ten times as much with 21,911 J and 58,562 J. On systems with limited power supply, it is essential to limit energy usage making this method a strong candidate for use on edge devices. Its improved inference speed also makes it useful for surveillance and other tasks that need real-time results.

The number of trainable parameters and float operations (GFLOPs) are also reduced in the STAR method which has some of the most efficient computation amount action recognition models (Table 3-1). It can be seen that STAR has some of the lowest numbers of these metrics of all standard action recognition techniques. The number of parameters is able to be optimized because only a single relatively shallow network is utilized while many other methods require multiple streams each with their own sets of parameters. MFCD-net is the one exception which unwraps 3D kernels into smaller 2D ones.

In terms of GFLOPs, STAR again performs very well due to its shallow structure. The strong performance of the model makes it a viable choice for low-cost computing without a drop in accuracy. If efficiency is the top concern, it can make sense to use a lower segment count, as using 4 segments produces comparable results with a lower processing time. This is a value that can be considered when determining the constraints and requirements of the model application.

Table **3-1**: Comparison of Action Recognition Methods

Model	Settings		Efficiency		Accuracy (%)	
	Compressed Video	Optical Flow	Parameters (M)	GFLOPs	HMDB-51	UCF-101
Two-Stream [26]	No	Yes	46.6	3.3	59.4	88
Resnet-152 [23]	No	No	60.2	11.3	48.9	84.3
I3D [2]	No	No	33	108	74.8	95.6
C3D [6]	No	No	78.4	38.5	51.6	82.4
TSM (Kinetics) [11]	No	No	24.3	33	73.5	95.9
MV2Flow-Pro [24]	Yes	No	95.2	18.7	64.5	92.1
DMC-Net [25]	Yes	Yes	83.6	15.1	62.8	90.9
CoViAR [1]	Yes	No	83.6	14.9	59.1	90.4
MFCD-Net [3]	Yes	No	8	8.53	66.9	93.2
STAR	Yes	No	24.3	4	64.5	92.4

Chapter 4

Implementing H.264 Compressed Action Recognition

With the space of action recognition in MPEG-4 part-2 having been widely explored, one could argue that the natural next step would be to apply the practice of compressed domain action recognition to more optimized codecs. H.264 has soundly become the most widely used codec for physical and digital video due to the improvements made over its predecessors. These optimizations also pose several challenges to compressed learning that have not yet before been approached. In this chapter, I discuss my analysis of the topic and several methods used to attempt to transfer the success of MPEG-4 part-2 recognition to H.264.

Motivation

Action recognition in the compressed domain has a two-fold benefit in that video in this format has reduced redundancy with built-in motion information and it does not require the overhead of a video decoder. With the use of MPEG-4 part-2, these benefits are not fully leveraged. This codec encodes many full RGB frames into the stream which retains a degree of unnecessary redundancy from the original video. Also, rarely do online video playback/streaming sites or local video editing tools use MPEG-4 part-2. That means, to learn and infer on the codec, video must be decompressed from a more modern codec and then compressed again. It is clear that classification on MPEG-4 part-2 is useful even given the constraints, but the approach can be further improved. H.264 is the most commonly used codec on the internet with popular sites like YouTube and

Twitch using it for its efficiency and ability to preserve high quality. It is also used for the highest quality physical storage medium: Blu-ray. Being able to apply a classification model directly to video obtained from nearly any source without need for conversion provides a large amount of savings to preprocessing. It allows custom datasets to be easily created by scraping websites and simplifies the pipeline for inferring on streaming content. H.264 removes the redundancies of MPEG-4 part-2 by using variable-length GOPs with I-frames encoded only when necessary. However, this lack of redundancy makes classification more difficult as there is less spatial information to work with. Because of this, temporal frames are forced to model motion data that is far from the reference making the frames noisy and often difficult to interpret. These are some of the challenges that must be overcome when introducing an effective model for processing H.264 encoded video.

Approach

The method used in CoViAR was used as the baseline for H.264 processing in the 2D space. The approach used in MFCDnet was made the baseline for 3D processing. For both of these spaces, the first step was modifying the data loader. This was followed by modifying the compressed format to account for the sparsity of the stream in an attempt to achieve comparable results to recognition on MPEG-4 part-2

Loading Stream Data

The loader uses ffmpeg C libraries including libavcodec and libavformat. When attempting to modify it to add H.264 support, several deprecated functions had to be removed with functionality replaced. I will summarize the deprecated approach. An AVCodec object is explicitly set to MPEG-4 part-2 and allocated to an AVCodecContext is allocated with the default parameters for the AVCodec. Finally, an AVCodecParserContext is also explicitly initialized to MPEG-4 part-2. A file pointer is created and used to open the video and each packet is read from using av_parser_parse2, passing the codec context and parser context, from which a frame is decoded using avcodec_decode_video2, requiring the codec context and packet. This frame can be used to obtain the RGB data or motion vectors. I needed to modify this approach to work with H.264. First, I tried setting the decoder and parser context values to the corresponding H.264 options, but this led to segfaults down the line. I traced the issue back to avcodec_decode_video2 for reading packets from the stream which is now deprecated and did not work with the current configuration. Utilizing the ffmpeg documentation and some examples, I was able to modify the script to work with H.264 streams. First, instead of working with a parser context, a format context was used which allows more information to be inferred from the stream when decoding. The format context was initialized using avformat_open_input which was passed the file path, and avformat_find_stream_info obtains the stream information from the file. Using the format context, av_find_best_stream determines the highest quality stream index which is saved for later processing. The function also returns the decoder (codec context) for the

stream, basically applying avcodec_find_decoder internally to the selected stream. The codec context for the stream is allocated and initialized using avcodec_parameters_to_context using the stream configuration. Now, with the stream selected, and the format context and codec context initialized, the stream can be read from. The function av_read_frame takes the format context, then reads and returns the current packet. The packet is decoded using avcodec_send_packet to send it to the codec context and retrieving the result with avcodec_receive_frame. From there, each frame can be processed normally. The approach CoViAR used to calculate the I-frames, mvs, and residuals was utilized and H.264 streams could now be decoded.

In addition to the load and get_frames_by_gop function, I needed to add another to obtain the required functionality. Due to the varying GOP size in H.264 streams, I wrote a function that returned an integer array with the value at each index corresponding to the number of frames in each GOP. A helper function was created to obtain the number of GOPs which would define the length of the array. Then, the video would be decoded and each non-I-frame decoded would increment the index of the current GOP and a decoded I-frame would increment the GOP index. This function would be necessary for use with the PyTorch dataloader, as the existing methods utilizing MPEG-4 part-2 calculate the GOP_index and frame_index needed for the load function using the constant GOP size of 12. From the Python implementation side, it can only be seen what the current frame number is that is being processed, while the load function needs the current GOP and frame number. These can easily be calculated for the older codec, but the extra information is needed for H.264.

Three-Stream 2D Architecture

The original approach I applied for processing H.264 was similar to the CoViAR implementation. A three-stream architecture was utilized with I-frames and accumulated motion vectors and residuals used. With the modified functions in the C data loader, the videos could be loaded the same way as before. For the I-frame model, the sampling method needed to be adjusted since there were far fewer I-frames to select from. Two methods were devised to attempt to mediate this lack of redundancy. The first method was using all I-frames in the stream for each stream even though this number varied by video. To utilize this method, the batch size needed to be set to 1 since using a higher batch size would require that tensors be stacked and that would require that they are the same shape. This was not guaranteed to be true if every I-frame was utilized since some clips would contain more than others. To accommodate for the small batch size, gradient accumulation was used which only updated the trainable weights after a certain number of elements were processed, which effectively allowed for any batch size to be used. The average over the batch dimension was used to fuze the outputs of the I-frames used with the average being the classification prediction. The second approach was to use a constant number of I-frames much like using a pre-defined segment count. If fewer than 3 I-frames are present in the stream, the present ones will be duplicated to fit the segment count. The number of segments was set to 3, as this would allow for some different spatial views to be learned without producing too many duplicated frames. The average video in HMDB-51 contains ~1.4 I-frames with the maximum being 12. The vast majority of these relatively short clips with often unchanging backgrounds contain only 1

I-frame. This makes it difficult to model I-frames with a particular number of segments. The results can be seen in Table 3-1.

Object-detection on Residuals

The problems with modeling motion vectors and residuals are different from those of I-frames. Motion vectors and residuals were abundant in this codec, but they were not always as useful as the ones in MPEG-4 part-2, which became especially apparent when using accumulation. Because motion vectors are traced back all the way to the previous reference frame, the reduced number of I-frames often makes the backtracking lengthy which introduces noise into the mvs and residuals. The mvs and residuals were sampled using the standard approach of a constant pre-defined segment count with the value being tested at 3 and 5.

To try to account for the noise accumulated over the long path back to the reference I-frame, a technique of cleaning the residual frames was introduced. An off-the-shelf object detection network, Single Shot MultiBox Detector (SSD)[17], was used to attempt to detect the interest points in the residual (Figure 3-1). It was attempted to run this object detector in tandem with the action recognition network, but CUDA was not allowing the resources to be shared. To overcome this, the detected residuals were pre-computed and loaded in when a specific residual frame was selected. This did require a large amount of preprocessing, but the approach likely could be configured to run both networks in parallel with improved multiprocessing. The preprocessing worked well for this application, though.

Figure **3-1**: Object Detection on Residual Frames (HMDB-51[21])

Pseudo-I-frame Generation

An approach utilized in MFCDnet attempted to replicate uncompressed frames by fusing residual frames with their reference frame in the GOP. An averaging of the residual RGB channels was done with those of the I-frame to attempt to retain more of the scene's context in the residual. Because this method creates a pseudo-uncompressed frame, I wanted to try to run a 2D ConvNet on frames generated this way to attempt to improve on the I-frame accuracy seen in H.264. The initial approach used frames from without the video across GOPs. This method, which I called "Cross-GOP", first selects from I-frames to obtain the desired ones. If there are more I-frames than segments to fill, then I-frames are randomly selected. If there are fewer I-frames, then random residuals are uniformly sampled from each GOP. This approach attempted to learn all major aspects of the video by using many I-frames along with the temporal information found

in the pseudo-I-frames. The method was tested on HMDB-51 with both 5 and 8 segments. The 5 and 8 segment approaches achieved 38.42% and 38.43% accuracy on HMDB respectively.

A modification was made to the method which is called “Divided-GOP”. For this, each GOP of a video is stored as a separate clip by the dataloader and is trained on separately. Each clip has a single I-frame and the remaining frames are residuals fused with the I-frame. This method aims to process the different portions of the video separately. Since I-frames are only produced when a major scene change is made, processing each GOP as a self-contained entity may make the residuals more relevant to the scene and help improve the classification accuracy.

Pseudo-Decompression

While the previous approaches focused on creating representations of video using vanilla compressed components, an attempt was made to partially decompress the stream to create more RGB frames than would be originally present in a stream. To do this, the accumulated motion vectors and residuals for a target frame are used in tandem with the reference I-frame to create an approximate-decompressed frame. This was implemented with the Cross-GOP approach which first uses I-frames encoded in the stream and then performs pseudo-decompression on intermediate P-frames to fill in the gaps and reach the target segment count. This method is still more computationally efficient than fully decompressing the video because only select frames are decompressed while only the motion vectors and residuals are extracted from others. It should be noted that some

artifacts from the decompression may be generated from this method of decompressing because of the method of generating residuals in the RGB space described in Chapter 2. The actual H.264 decoder will extract the motion vectors and residuals directly from the DCT values which correspond to the YUV space. The estimated residual values are good approximations, but they do cause some inconsistencies to be produced when they are used to decode (Figure 4-1). This partial decoding approach shows promise, as it could mediate the main downfall of working solely in compressed H.264 which is the lack of spatial information

Figure **4-1**: Decoder Extracted Frame (left) vs Frame Decompressed using raw MVs and Residuals (HMDB-51[21])

.Applying 3D Architectures to H.264

The limited interaction between compressed components in the multistream becomes more apparent in the sparse H.264. This inspired the use of 3D CNNs that learn on pseudo-decompressed video. One of the benchmark deep learning techniques for learning trends in data with increased dimensionality is the use of 3D kernels which operate over the x, y, and z axes. For video inputs, this is width, height, and time. 3D

CNNs are expensive to train but can provide good results when the resources are made available [6]. MFnet[12] introduces the idea of breaking 3D kernels into a series of 2D ones which can learn the same information using significantly fewer parameters. The work was extended to the compressed domain with MFCDnet[3] where clips were created by stacking each I-frame with the following 11 residual frames. The residual values were fused with the clip's I-frame to attempt to adjust for the motion and create a full clip. The approach performed well on MPEG-4, but it still required a large overhead due to having to extract the motion vectors and residuals from every P-frame. This made training very time-consuming because of the large number of clips created from a dataset. With the approach for MPEG-4 part-2, each GOP is made into a clip which makes an average of 10 clips per video. On my machine, I was unable to complete a training epoch on an HMDB-51 data split with about 3,000 videos that was turned into over 30,000 clips. Either due to data transmission rates or CUDA memory, my machine froze after processing about half of an epoch on several occasions, so the approach would likely have to be adjusted to be viably used. The sparse structure of H.264 seems to make this kind of spatio-temporal fusion a good approach to capture the interesting elements of the video. The MFCDnet paper utilized the same data loader as CoViAR making use of the load and num_frames functions. The modified functions I implemented to incorporate H.264 support were able to be used in place of these, as they work with both codecs. The model uses pre-trained weights from ActivityNet and it is first finetuned on uncompressed HMDB-51 clips. It is then trained on clips of the same length in the compressed MPEG-4 format. Using this kind of decoupled 3D architecture seems like a solid starting point for modeling H.264 video.

Residual Intervals vs. Random Selection

The initial attempt to use the 3D net on H.264 inputs was done using uniform sized sets of randomly selected residual frames from each GOP to form a clip. The length of H.264 GOPs made it so that often one video would contain only 1 I-frame and would therefore produce one clip, but a video would produce as many clips as it has I-frames. The architecture from the application on MPEG4 videos was retained, so a clip length of 12 frames (1 I-frame + 11 residuals) was used. That means 11 residuals would be selected from each GOP. This was tested in two ways. The first way was selecting a random interval of residual frames, where all would be adjacent to each other. So, a clip could be made from the I-frame and residuals 8-18, for example. The second way was to select 11 random residuals within the GOP, not a certain interval. This would potentially cover a larger time period but with less temporal continuity. If fewer than 11 P-frames made up the GOP, the selected ones would be duplicated to reach the 12 frame length. This process was also tested on constant clip sizes of 24 frames.

Variable-Length Clips

Another method was implemented which, rather than utilize fixed-length clips, used clips of several varying sizes that would be selected based on the length of the GOP. For example, a GOP less than 20 frames long would produce a clip of 12 frames, and less than 30 produce a clip of 18. This was used to create 5 distinct clip sizes: 12, 18, 24, 30, and 36 frames. The residual frames corresponding to the clip would still be randomly selected. This was thought of as a good compromise between having a set size for all

clips regardless of length, which may miss important information in a long segment, and retaining all frames for each clip, which would be computationally expensive and the extreme variability in length would likely slow learning. Because this approach feeds different sized tensors to the network, the previously discussed practice of setting the batch size to 1 and using gradient accumulation was employed here as well.

Cross-GOP for 3D CNNs

A final method I designed utilized the Cross-GOP frame approach used with the 2D ConvNet. This would keep a video together by forming 12 frame clips using all of the I-frames in a video (unless there were more than 12) and modified residual frames for the rest. This would produce temporally sparser clips, but I believe that this reduction in redundancy could uncover stronger patterns in the videos.

Results

2D Methods

For the three-stream approach, the accuracy was fairly consistent with the CoViAR approach on MPEG-4. For segment counts of 3 and 5 (Table 3-1), the accuracy was about equivalent to the former method for residuals, but I-frame accuracy was noticeably lower with a score of 40.20% for H.264 against 51.38% for MPEG-4 as well as motion vectors, with 36% to 40% (Tables 2-1 and 4-1). This disparity in the I-frame accuracy makes sense as in H.264 there are inherently fewer of these frames to utilize so

many samples will be using only a single I-frame. The attempt to generate pseudo-I-frames by fusing them with residuals did not improve the accuracy. The Cross-GOP approach achieved a score of 38.4%, while the Divided-GOP method did not perform well-enough to consider. The approach may have not proved effective because summing the residuals with the I-frames can lead to noisy images without a clear area of focus (Figure 5-1). It may prove useful to utilize different fusion techniques to attempt to more accurately reconstruct the frame, which is an area of future work that I would strongly consider exploring.

Table **4-1**: H.264 3-Stream Architecture Accuracy by Component and Segment Count (HMDB-51)

Segment Count	Compressed Element	Accuracy (%)
3	I-frame	37.32
	MV	36.60
	Residual	39.74
	Combined	**56.34**
5	I-frame	40.20
	MV	36.86
	Residual	40.13
	Combined	**56.41**

Figure **5-1**: I-frame and Pseudo-decompressed Residuals (HMDB-51[21])

The best-performing method was the pseudo-I-frame-decompression technique that shifted the GOP's I-frame by the motion vectors and residuals of selected P-frames. This method obtained an accuracy of 38% for the I-frame model, though it does induce a greater cost than the other methods as pixel-wise operations must be done to "decompress" the frame (Figure 5-1). An extension of this approach could be implementing a partial decompression on some computed area of interest. This could reduce the added cost of shifting each pixel. This could likely be done by utilizing the motion vector array. Since the array is usually very sparse, only non-zero entries could be utilized, reducing the amount of computation needing to be performed.

While I-frame scores can continue to be improved on, the difference in the motion vector scores was relatively small but still noticeable. I speculate that the longer GOP length allowed excess noise to accumulate in motion vectors later in the GOP. This was the motivation behind using object-detection on the H.264 accumulated residuals. If this was successful, the bounding boxes for the residuals could have been transferred to the

motion vectors to capture only the areas of interest. However, the object-detected residual frames also did not generate any benefit, as an accuracy of only 20.7% was obtained on the HMDB-51 dataset. Because of this result, the method was not extended to motion vectors, but the approach could be retried using different configurations on the object detector. The residual scores were about equal on MPEG-4 part-2 and H.264 (about ~40% each). While motion vectors may become clouded over this longer exposure period due to their low dimensionality (only x and y), residual frames in H.264 can still effectively capture the contents of a frame and the longer GOP may actually help obtain more information about the scene. The three dimensions of a residual (RGB) may allow for the longer GOPs to not make the component too noisy and be able to present the scene more clearly. When performing the fusion on the three models, the testing accuracies were combined using the I-frame predictions weighted at 2x that of the mv and residual ones. This led to the highest accuracy at ~56% for 3 and 5 segments. Though the individual components were lower for H.264, the final prediction accuracy comes within 3% of the CoViAR score for MPEG-4 part-2 and about 8% of the STAR method for the older codec. Overall, this shows promise for the ability to perform action recognition on H.264. A future work could be to utilize an optimized pseudo-decompression of H.264 to extract frames and use them as inputs for the earlier mentioned STAR model. This could perform a single-stream operation that would only need a few input images to obtain improved results. Due to the sparsity of H.264, I did not expect such competitive results using late fusion on a multi-stream architecture. However, there are many promising directions to extend this work given the methods and results presented.

3D Methods

The various approaches using a 3D architecture on H.264 had major variations in performance with none of them matching the accuracy achieved on MPEG-4. The reported accuracy of MFCDnet on MPEG-4 was 66.9% using the default implementation. Upon performing finetune training on uncompressed HMDB-51 videos, as was done in the original approach, I was achieving a maximum accuracy of 50%. I used these fine-tuned weights for transfer learning for H.264 recognition (Table 5-1). It became clear that using longer clip sizes and variable-length clips caused sharp reductions in accuracy when the parameter count remains unchanged. It makes sense that varying the clip size would reduce the effectiveness as the kernels are fixed-size making certain patterns more discernible in shorter clips than longer ones. Using the longer 24-frame clips with the same parameter count also has the potential to have important features be missed, as there is more information to be processed by the same kernels as with 12 frame videos. For the 12 frame methods, using random residual selection achieved slightly better results than using a contiguous interval (40.08% vs 37.71%), which makes sense since using a wider range of residuals in the GOP allows motion from a wider temporal region of the video to be learned on. These methods were still splitting each video into multiple samples; one for each GOP. The Cross-GOP approach was faster to train, as each video was used as a single sample, and it was the most accurate at 43.11%. The 3D architecture did not perform very well overall with much lower accuracies than the 2D approach on H.264 and the 3D on MPEG-4. However, the Cross-GOP approach proved promising with the best scores out of all of the approaches on H.264, I see it as a solid starting point in this

domain. A way to potentially extend the approach could be to perform the pseudo-decompression used for the 2D ResNet to obtain the additional frames for the clips rather than using the simple I-frame-residual fusion technique. The feasibility of this approach would depend on the ability to efficiently utilize motion vectors to decompress areas of interest in a frame. Using these methods in combination could potentially make the use of 3D CNNs for H.264 recognition more viable. Another approach that I believe would be useful to attempt would be utilizing a vanilla 3D ResNet with some of the approaches discussed to produce partially-decompressed clips. Though these models were avoided due to the high parameter counts and training times, using them as a benchmark would help to better determine whether the lower accuracies were due to the architecture or the methods used.

Table **5-1**: MFCDnet using H.264 using varying configurations (HMDB-51)

Method	Accuracy (%)
Randomly selected 12 frame continuous interval	37.71
Random sparsely selected 12 frames	40.08
Randomly selected 24 frame continuous interval	21.06
Variable length samples based on GOP size	22.97
12 frame Cross-GOP samples	43.11

Chapter 5

Discussion

Upon exploring the field of action recognition in compressed video, I am confident that this technology is highly relevant to many areas of industry and study. Given its potential in both classification quality and computational efficiency, compressed recognition has the ability to disrupt industries ranging from security and surveillance to robotics and IoT. Upon analyzing works such as CoViAR and MFCDnet that laid the groundwork for inferring from compressed video, it is clear that there is much room for exploration and improvement in the field. The areas that I chose to focus on were reducing the redundancy related to the recognition of MPEG-4 video while maintaining, and even improving, accuracy, and being the first to implement compressed action recognition on H.264 video. Both of these contributions have the ability to introduce the application to new areas or increase its effectiveness in a field. The efficiency in power or the improved MPEG-4 method makes it a prime candidate for use in edge devices with limited energy capacity. The introduction of recognition on H.264 has the potential to exploit the large prevalence of this codec in internet video.

MPEG-4 Part-2 and STAR Method

The initial method of improving recognition scores on MPEG-4 encoded videos was able to provide increases in performance both in accuracy and efficiency over the original methods. The accuracy improved because of the ability to utilize more I-frames from the stream and the network was able to effectively determine the connection

between them through the feature shifting. The efficiency was improved because this shifting provided computationally free motion information that was processed along with spatial features. This allowed for the use of a single-stream network which greatly improved training and inference times. On HMDB-51, the STAR method obtained 64.51% accuracy and had 92.47% on UCF-101. Both of these metrics beat CoViAR's three-stream fusion technique, and with 8 segments, it improves on the energy performance by over 10x. If the segment count is reduced, there was not a significant performance drop while training and inference times improved noticeably. This creates the potential for the segment count to be reduced when performance needs to be optimized. The model performed best when the segment count was closest to the number of I-frames in the dataset's videos. So, this is another parameter that can be adjusted based on the makeup of the inputs. Overall, this method shows the connection between action recognition methods in the compressed and raw domains. The efficiency gained by using temporal shifting in addition to not having to decompress the streams is an important step forward in validating the use of compressed video for recognition. The further control granted by being able to modify the segment count makes the approach a suitable candidate for use on edge devices, whenever low power is a requirement, or simply when speed is a driving factor in selecting an algorithm.

H.264 Support

The rise in use of H.264 strongly coincides with the age of internet video with sites like YouTube and Twitch using this as their main video codec. Therefore, a design

for performing learning on compressed video should include support for this compression format. To my knowledge, this is the first set of approaches to support it, so the goal of them is to show that achieving comparable performance on the less redundant codec is feasible. Simply using the components separately without modification understandably led to lower scores for I-frames but produced nearly equivalent scores for mvs and residuals. Through the fusion of predictions, the consensus accuracy was only a few points lower (56% vs 59.2%) than on MPEG-4. Several attempts were made to improve individual stream performance, such as object detected residuals and pseudo-I-frames. These particular methods did not cause any gains, but the pseudo-decompression technique did boost the individual I-frame score. The increase in computation added, though, makes the technique less viable for real-world deployment. However, using a form of selected decompression by only modifying areas of interest in a frame, the overhead could be minimized while still retaining the performance benefit. Also, creating these decompressed predictions would potentially allow temporal shifting to be done between them and the STAR model could be applied to H.264.

The use of 3D CNNs on the codec did not produce competitive results to that of the 2D ones. Most promising of the approaches was the Cross-GOP approach that sandwiched fused residuals between all of the I-frames in a video to form a clip. Potentially, modifying the way that residuals are modified could lead to similar scores as the 2D ResNets. To better determine if it is the architecture causing the lowered accuracy or the methods themselves, vanilla 3D ResNet models should be tested with modified, compressed H.264 clips and the performance analyzed.

Closing Remarks

The methods introduced in this work have the ability to extend AI-based action recognition to many areas of industry, especially with the prevalence of web-based applications and IoT devices. With the improved efficiency of MPEG-4 recognition, there is potential to apply it to low-power devices, like a Ring doorbell camera, or security software. It is also suitable for high-speed scanning for both local/cloud storage and website playback/streaming content. For example, videos depicting illegal acts or containing copyrighted material can be flagged and handled. The prevalence of H.264 for internet video makes supporting it able to extend these applications to many mainstream sites. If a codec is not supported, then decompression and recompression to a different codec would have to be performed which hampers performance and removes a major benefit of working in the domain. That is why adding support for H.264 is such an important element of extending the application. This work implements new techniques and support for compressed domain video action recognition which can set the groundwork for its further adoption and use across disciplines.

BIBLIOGRAPHY

[1] Wu, C.-Y., Zaheer, M., Hu, H., Manmatha, R., Smola, A.J., Kr¨ahenb¨uhl, P.: Compressed Video Action Recognition (2018)

[2] Carreira, J., Zisserman, A.: Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset (2018)

[3] Battash, B., Barad, H., Tang, H., Bleiweiss, A.: Mimic the raw domain: Accelerating action recognition in the compressed domain. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2926–2934 (2020)

[4] Kai Xu1, F.S.Y.W.Y.-k.C.F.R. Minghai Qin: Learning in the frequency domain. CVPR 2020; abs/2002.12416 (2020) https://arxiv.org/abs/2002.12416

[5] Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(1), 221–231 (2013). https://doi.org/10.1109/TPAMI.2012.59

[6] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning Spatiotemporal Features with 3D Convolutional Networks (2015)

[7] Kensho Hara, Y.S. Hirokatsu Kataoka: Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? CVPR 2018 abs/1711.09577 (2018) https://arxiv.org/abs/1711.09577

[8] Du Tran, L.T.J.R.Y.L.M.P. Heng Wang: A closer look at spatiotempo-

ral convolutions for action recognition abs/1711.11248 (2018) https:
//arxiv.org/abs/1711.11248

[9] Wu, C.-Y., Zaheer, M., Hu, H., Manmatha, R., Smola, A.J., Kr¨ahenb¨uhl, P.: Compressed Video Action Recognition (2018)

[10] Jacobs, P.J. Marco: A Brief History of Video Coding. (2009)

[11] Lin, J., Gan, C., Han, S.: Temporal shift module for efficient video understanding. CoRR abs/1811.08383 (2018) https://arxiv.org/abs/
1811.08383

[12] Yunpeng Chen, J.L.S.Y.J.F. Yannis Kalantidis: Two-stream convolutional networks for action recognition in videos abs/1807.11195 (2018) https:
//arxiv.org/abs/1807.11195

Springer Nature 2021 LATEX template

2 STAR: Efficient SpatioTemporal modeling for Action Recognition

[13] Christoph Feichtenhofer, R.P.W. Axel Pinz: Spatiotemporal residual networks for video action recognition abs/1611.02155 (2016) https://arxiv.
org/abs/1611.02155

[14] Karen Simonyan, A.Z.: Two-stream convolutional networks for action recognition in videos abs/1406.2199

15] HEVC: An introduction to high efficiency coding. https://www.vcodex.
com/hevc-an-introduction-to-high-efficiency-coding/. Accessed: 2022-01-
06

[16] Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Gool, L.V.: Temporal Segment Networks: Towards Good Practices for Deep Action

Recognition (2016)

[17] Wei Liu, D.E.C.S.S.R.C.-Y.F.A.C.B. Dragomir Anguelov: Ssd: Single shot multibox detector abs/1512.02325 (2016) https://arxiv.org/abs/1512.02325

[18] Simonyan, K., Zisserman, A.: Two-Stream Convolutional Networks for Action Recognition in Videos (2014)

[19] Christoph Feichtenhofer, R.P.W. Axel Pinz: Spatiotemporal multiplier networks for video action recognition (2017)

[20] VP9 vs HEVC H.265 - Which is the Future of 4K World? https://www.macxdvd.com/mac-dvd-video-converter-how-to/vp9-vs-h265.htm. Accessed: 2022-01-06

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

[22] Soomro, K., Zamir, A.R., Shah, M.: UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild (2012)

[23] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition (2015)

[24] Hu, H., Zhou, W., Li, X., Yan, N., Li, H.: Mv2flow: Learning motion representation for fast compressed video action recognition. ACM Trans. Multimedia Comput. Commun. Appl. 16(3s) (2021). https://doi.org/10.1145/3422360

[25] Shou, Z., Lin, X., Kalantidis, Y., Sevilla-Lara, L., Rohrbach, M., Chang,

S.-F., Yan, Z.: DMC-Net: Generating Discriminative Motion Cues for Fast Compressed Video Action Recognition (2019)

[26] Simonyan, K., Zisserman, A.: Two-Stream Convolutional Networks for Action Recognition in Videos (2014)

ACADEMIC VITA

Education

Pennsylvania State University (2017 - 2022) (Anticipated)

- Bachelor's in Computer Engineering in College of Engineering and Schreyer Honors College
- Master's in Computer Science and Engineering

Relevant Coursework:

CS: Java, Python, Discrete Math, Algorithms, OS, Databases, Security, AI, Assembly
Additional: Mechanics, Electrical Engineering, Statistics, Linear Algebra, Diff. Eq.

Experience

Teaching Assistant: CMPEN 472 Microprocessors & Embedded Systems (August 2021-December 2021)

- TA for Dr. Kyusun Choi
- Assisted students with programming assignments in HCS12 assembly

Software Engineer Intern: Capital One, McLean VA (May 2021 - August 2021)

- IOS development using Swift
- Utilized Combine framework and UIKit to update Payments Plugin for Enterprise App

Research Assistant: Microsystem Design Lab, Penn State, State College PA (Dec 2020-Present)

- Designing models for performing compressed domain action recognition
- Surveying current trends in computer vision and action recognition

Software Engineer Intern: Capital One, McLean VA (May 2020 - August 2020)

- Full Stack development using React.js, Nest.js, ElasticSearch, and AWS
- Effectively integrated frontend components with ElasticSearch endpoints
- Worked with Jenkins pipeline to automate build deployment

Software Engineer Intern: CommScope Inc, Horsham PA (May 2019 - August 2019)

- Used C/C++, JavaScript, Git in CPE division of CommScope
- Gained experience in structured and object oriented design methodologies, multimedia streaming, and embedded development

Awards and Accomplishments

- Penn State President's Freshman Award
- Dean's List Fall '17, Spring '18, Fall '18, Spring '19, Spring '20, Fall '20, Spring '21, Fall '21
- John J. and Jean M. Brennan Trustee Scholarship Recipient