INCORPORATED TEMPORAL ACTION PROPOSAL GENERATION

JOSEPH SANU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
FOR THE DEGREE OF
MASTERS OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

APRIL 2020

Abstract

We propose a new unified approach to generate high quality temporal action proposals from untrimmed videos called Incorporated temporal action proposal generation. The concept behind this model is to consolidate the processes that classify the small temporal segments and evaluate the larger proposal features. In doing so, we seek to reduce the total number of computational units necessary for end-to-end proposal generation. For our research, we have conducted our experiments on the action proposal task in the ActivityNet challenge where the goal is to produce a set of candidate temporal segments that are likely to contain a human action.

In addendum to the aforementioned research, we also propose an extension to this work by applying video level classification on our proposals. For this work, we emphasize the importance of accurate sequential modelling of temporal segments to properly distinguish between macro and micro-level actions within untrimmed videos and we compare our results with other state-of-the-art spatio-temporal models.

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	ARTIFICIAL INTELLIGENCE
BPTT	BACK-PROPAGATION THROUGH TIME
BSN	BOUNDARY-SENSITIVE NETWORK
CNN	CONVOLUTIONAL NEURAL NETWORK
FIR	FINITE IMPULSE RESPONSE
FNN	FEEDFORWARD NEURAL NETWORK
FSMN	FEEDFORWARD SEQUENTIAL MEMORY NETWORK
HOF	HISTOGRAM OF OPTICAL FLOW
IOU	INTERSECTION OVER UNION
LSTM	LONG SHORT-TERM MEMORY
MBH	MOTION BOUNDARY HISTOGRAM
NTM	NEURAL TURING MACHINES
RNN	RECURRENT NEURAL NETWORK
TCN	TEMPORAL CONVOLUTIONAL NETWORK

Chapter 1

Introduction

1.1 Motivation

Computer vision is a research field that has seen remarkable progress over recent years. It deals with the processing and analysis of digital images and videos to extract information and generate high-level representations. Traditionally, this pursuit was dominated using feature detection and feature extraction through the use of mathematical methods such as edge detection and feature descriptors such as SIFT. Recently, advancements in artificial intelligence have contributed to the better understanding and processing of digital images and videos. Fueling this success is the design and utilization of complex Deep Learning neural architectures that eclipse the tradition approaches in terms of performance. Furthermore, they have shown their versatility through translation-invariance in image processing as in the case of Convolutional Neural Networks (CNN).

While these advancements have made various applications for image related tasks increasingly viable, the employment of such progressively complex architectures for video related tasks is a challenge. The temporal dimension in untrimmed videos results in an exceedingly large computational cost that makes practical use of the models with large datasets unfeasible. We specifically target the temporal action proposals task and undertake this effort to prove we can obtain comparable results using an improved end-to-end design. We achieve this using a simpler neural network and a more efficient sequential modeling approach using Feedforward Sequential Memory Networks (FSMNs) which was first proposed by Zhang [12] for speech recognition and language modeling tasks. In this thesis, we present a novel approach in using feedforward sequential memory networks (FSMN) for the temporal action proposal task and the action classification task.

1.2 Contribution and Thesis Organization

Over the past few years, the ActivityNet challenge has gained increasing popularity with the temporal action proposals being the hardest task to tackle.

Contrary to many of the top design choices for this task, we propose an architecture with a reduction in computational complexity by using Feedforward Sequential Memory Networks (FSMN).

In recent years, Feedforward Sequential Memory Networks (FSMNs) have been successfully implemented in various speech recognition and language modelling tasks.[11] For our research, we apply to FSMNs to model spatio-temporal data. We also contend we can make significant

design modifications in the visual features to proposals pipeline to reduce the number of computational units while still being able to successfully capture sequential data in untrimmed videos.

Subsequently, we expand the use of FSMNs for other tasks such as action classification to culminate our research.[13]

The rest of the thesis is organized as follows:

• Chapter 2 reviews Deep Learning for Image Processing and related works.

• Chapter 3 reviews feedforward sequential memory networks.

• Chapter 4 presents the Incorporated temporal action proposal generation design and describes our

unified end-to-end approach to generate temporal action proposals.

•Chapter 5 demonstrates our success in applying our Incorporated model to temporal action proposal tasks.

• Chapter 6 presents our concluding remarks.

Chapter 2

Related Work

2.0.1 Deep Learning for Image Processing

Recently, large amounts of annotated image data collected from search engines have facilitated the rise of powerful models that shine at image recognition tasks. The largest and extensive of these being the ILSVRC image classification dataset also known as the ImageNet dataset with over a million images and 1000 object categories.[1] All of the images in the ImageNet dataset contain manually labeled ground truth labels. They can be further subdivided into 1.2 million training images, 50 thousand validation images and 100 thousand test images. One of the tasks in the ImageNet Large Scale Visual Recognition Challenge is image classification which tests the ability of an algorithm to recognize the objects in the image without localizing them. The ImageNet dataset builders were able to generate a diverse set of images by using multiple search engines and an expanded set of queries in multiple languages. To annotate the image data, they employed Amazon Mechanical Turk (AMT) which is an online platform

where a monetary reward is available for users that perform tasks. One of the more powerful architectures that arose from the ImageNet image classification task are deep convolutional networks such as VGG and ResNet.[14] [15] These models have shown success through the leveraging of large amounts of image data, utilization of devices like skip connections as shown in Fig. 2.4 and through the use of multiple convolutional and pooling layers.

This work has been further expanded by [3] through modifications to the standard VGG architecture for video classification. This is a logical extension from the image classification task as they both utilize RGB images as input. The main difference being that image classification requires one input image for accurate classification while the other typically requires the sampling of several RGB frames in order to accurately classify an entire video.

Optical flow can be defined as the pattern of motion of objects and edges based on relative motion as perceived through the lens of an onlooker. Many different methods of estimating optical flow using images exist and they are usually based on taking partial derivatives of image signals. A commonly used approach is the Lucas-Kanada method which computes optical flow for a local neighborhood of pixels in an image frame using the least squares method. To compute the optical flow for all the points in the frame, the authors in [4] use a dense optical flow algorithm using opencv-python to generate the optical flow frame. They then store the precomputed flow fields in JPEG format while clipping displacement vectors greater than 20 pixels and commence training. The optical flow frame consists of a 2-channel array with optical flow vectors (u, v). They contain information with respect to the direction and magnitude of the optical flow. Fig 2.2 shows the optical flow computed using two image frames where the

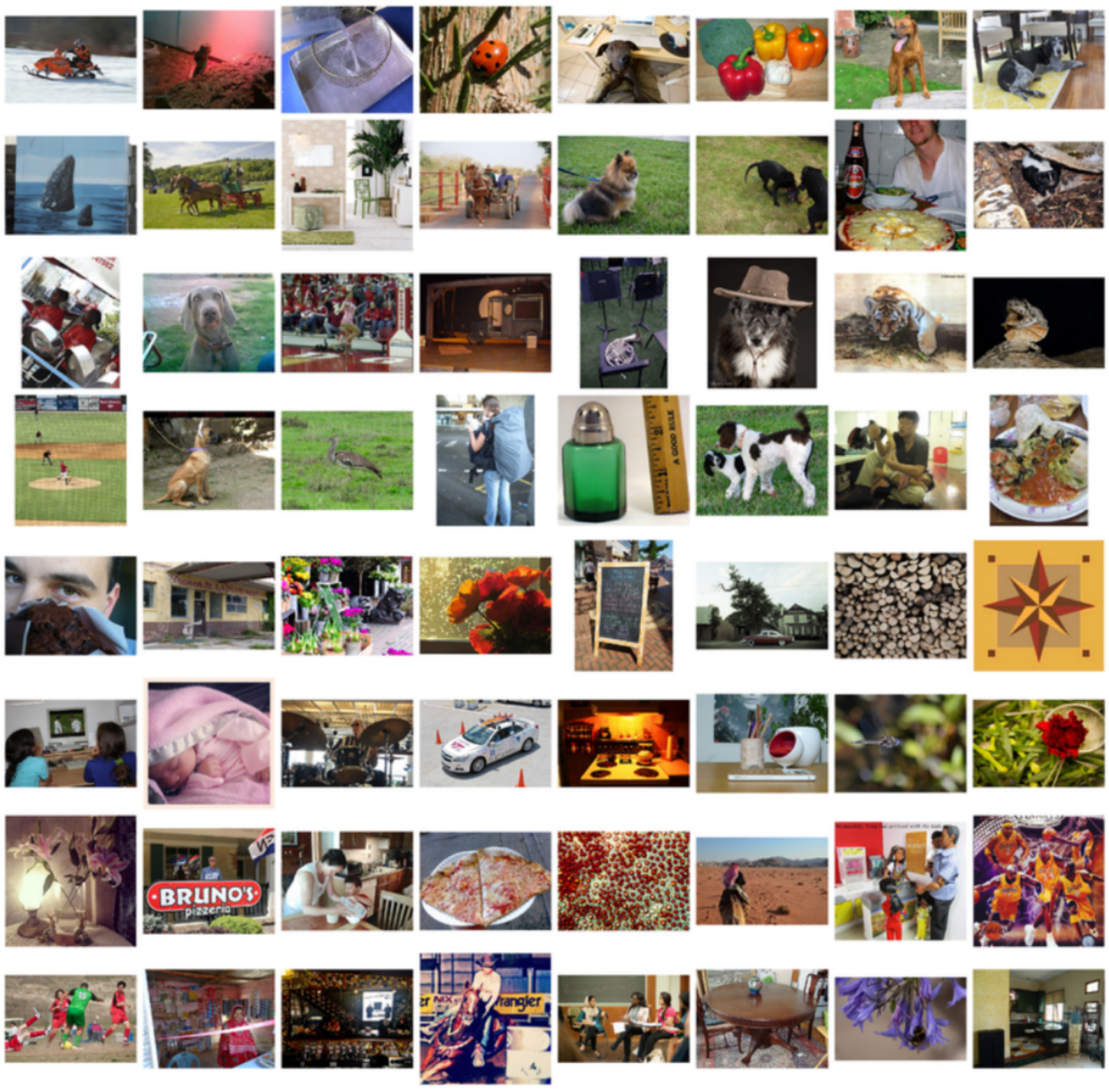

Figure 2.1: Diverse sample of images from the ImageNet Dataset [1]

Figure 2.2: Graphic of optical flow vectors overlaid on a greyscale image.[2]

direction is depicted by the hue and the magnitude denoted the brightness.

2.0.2 Two-Stream Residual networks

The two-stream Residual network [4] has its basis in the two-stream hypothesis from neuroscience. This theory postulates that there are two pathways in the human visual cortex. The first being the ventral pathway which relate to visual identification of objects. The second being the dorsal pathway which is receptive to object transformations.[15] Research further suggests that the visual cortex is equipped with connections between the dorsal and the ventral stream to share motion information to different visual regions. Skip-connections were introduced to address the problem of accuracy saturation and degradation with increasing network depth. An obvious benefit in adding skip-connections is enabling the direct signal propagation from the first layer to the last layer of the network. In addition to this, gradients can also be propagated to other layers of the network directly and skip middle weight layers which could lead gradient vanishing.

The authors in [4] experiment with hierarchical learning of spacetime features by connecting the appearance and motion channels through the use skip connections that are already present in standard ResNet architectures. The skip connection is implanted from the second residual at each spatial resolution of the motion stream to the spatial stream. This is done to ensure that the network can learn spatio-temporal features at different scales. However they do not implant skip connections from the spatial streams residual unit to the motion stream to ensure that the network would not bias the loss towards appearance clues.

Figure 2.3: Figure 1. Visualization of dense optical flow using OpenCV.[3]

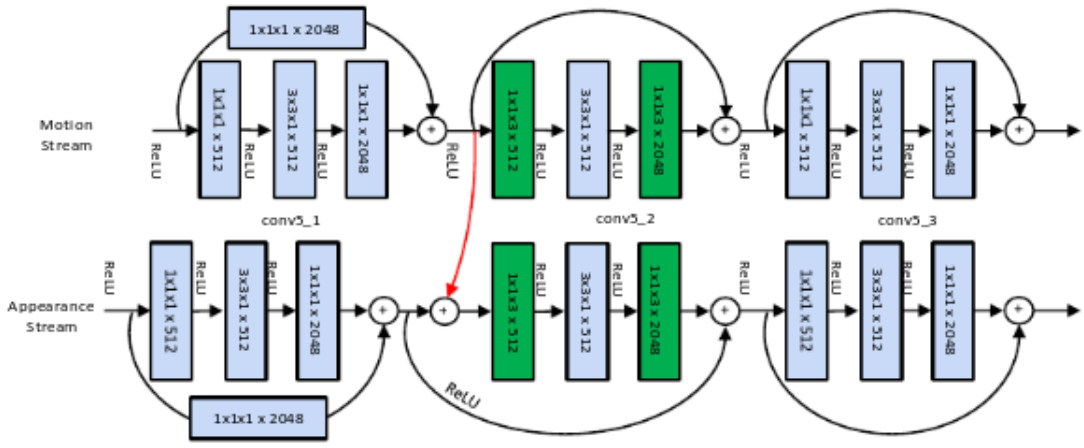

Figure 2.4: Illustration of skip connection from the motion stream to the spatial stream.[4]

2.0.3 Deep Learning Models

Machine Learning is the study of techniques based on mathematical models then can be used for analyzing data or making decisions. It falls in the greater umbrella of artificial intelligence design. These machine learning techniques can be used to construct a model that is capable of generalizing patterns from a given data set. A popular subset of machine learning tools is deep learning. Deep learning techniques in recent years have been steadily rising in popularity and is the prominent approach in the development of intelligent systems. They signify the development of complex neural network structures that mimic the connections in human brains. This artificial neural network consists of many neurons which are represented by mathematical functions that are interconnected with each other.

The deep network architecture has the capability to build hierarchical representations with non-linear transformations to form complex structures. This mechanism has parallels to the manner in which humans process information. The following is an overview of some of the

most commonly used artificial neural network models.

2.0.4 FeedForward Neural Network

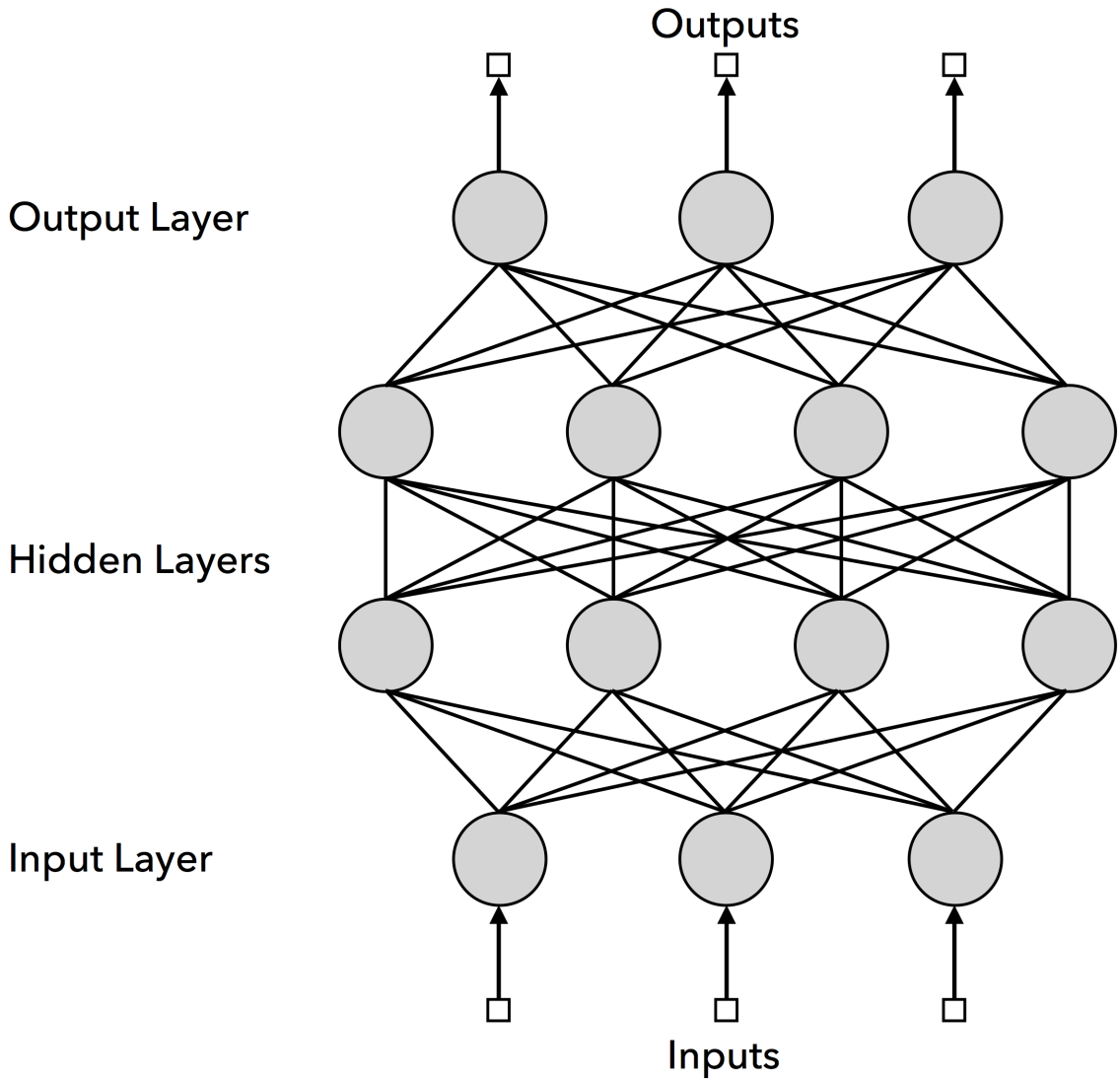

Figure 2.5: A multilayered Feedforward Neural Network.

Feedforward neural network (FNN) is an artificial neural network that is composed of sequential layers containing multiple nodes. Each layer is fashioned in such a way that they are

fully connected to subsequent layers. In FNNs, there is no cyclic connections between nodes and the information flows in the forward direction from the input layer to the output later and does not flow backwards. The following describe computations that are done by FNNs:

•Let x_n denote the value of n-th user input.

•Let y_n denote the value of n-th output.

•Let l_n^i denote the value of n-th node in i-th layer.

•Let $w_{n,m}^i$ denote the weight of the connection from m-th node in (i-1)-th layer to n-th node in i-th layer.

•Let σ denote the activation function.

•Let b^i denote the bias.

The value of each node at the input layer of an FNN can be represented using the following equation.

$$l_n = x_n$$

The output at each node in the i-th hidden layer can be determined using the equations shown below.

$$a_n^i = \sum_m w_{n,m}^i \cdot l_m^{i-1} + b^{i-1}$$

$$l_n^i = \sigma(a_n^i)$$

At the output layer, the value of each output node can be computed using the following

equation.

$$y_n = l_n^o = \sum_m w_{n,m}^{o-1} \cdot l_m^{o-1}$$

If the last layer is the softmax layer which is usually true for classification tasks, the output values are computed as shown below.

$$Softmax(y_n) = \frac{exp(y_n)}{\sum_m exp(y_m)}$$

In the process of training a feedforward neural network, the weights in its hidden layer are adjusted using back-propagation. The results of fine tuning the weights can be expressed using the following notation.

•Let y be the output of the network at the current state.

•Let γ be the target output of the network.

•Let $E(y, \gamma)$ denote the loss function, which measures the distance between the expected output and the predicted output.

•Let α denote the learning rate parameter.

The weights are updated after each iteration. The results of updating the hidden layer weights can be calculated by the following equations.

$$\frac{\partial E}{\partial w_{n,m}^i} = \frac{\partial E}{\partial \sigma} \frac{\partial \sigma}{\partial a_n^i} \frac{\partial a_n^i}{\partial w_{n,m}^i} = \frac{\partial E}{\partial \sigma} \frac{\partial \sigma}{\partial a_n^i} l_m^{i-1}$$

$$w_{n,m}^i := w_{n,m}^i - \alpha \frac{\partial E}{\partial w_{n,m}^i}$$

2.0.5 Recurrent Neural Networks

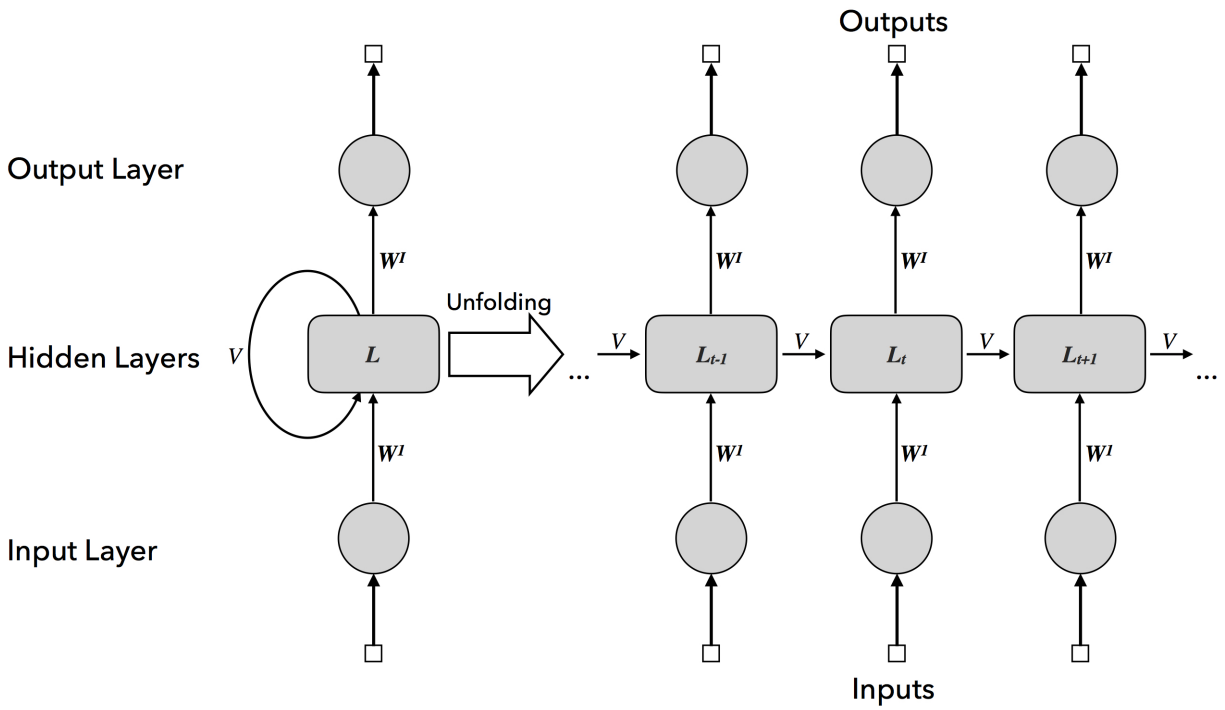

Figure 2.6: Recurrent Neural Network.

Recurrent Neural Networks (RNNs) are a type of neural network that is designed to capture long-term dependency given some sequential data. Rather than computing complicated transformations on the sequential data, RNNs store long term information in the network weights. This gives RNNs the capability to map a variable length sequence to another as opposed to only allowing fixed sized input to outputs as in the case of FNNs.

It does this by introducing a recurrent connection to the architecture. The recurrent connection allows the network to access previous hidden layer information. This enables past information computed by the neural network to persist and can be used with current inputs. We can view recurrent neural networks as a chain-like structure by unfolding the RNN. An example

is shown in Fig. 2.6 where the output of the hidden layer in the previous time step affects the hidden layer output at the current time step.

Unlike feedforward networks, RNNs learn through a process known as back-propagation through time (BPTT). So all of the weights are updated by first unfolding the network through time and then computing back-propagation similar to a feedforward neural network.

An RNN's hidden layer can be computed using the following equation:

$$L_i^t = \sigma(V^i \dot{L}_{t-1}^i + W^i \dot{L}_t^{i-1} + b^{i-1})$$

where

•L_t^i represents the values of all nodes in the i-th layer at the t time step

•W^i represents the weights of the forward connection from the (i-1)-th layer to the i-th layer

•V^i represents the weights of the recurrent connection from the i-th layer at the previous time step to the i-th layer at the current time step

•σ denotes the activation function

•b^i denotes the bias.

An unfolded RNN [11], is an RNN which is unfolded in time for a fixed number of time steps. The unfolded RNN has similar training time as the FNNs while achieving better performance than FNNs. However, the contextual information learned by the unfolded RNNs is still restricted due to the limited number of unfolding steps in time.

2.0.6 LSTM

As mentioned earlier, RNNs learn through BPTT which increases the computational complexity and adds a host of new problems. The common problems associated with BPTT learning include gradient vanishing and exploding. In an attempt to address these problems, new architectures have been introduced with the most popular being the Long term short memory (LSTM) model. While RNNs are capable capturing dependencies, this ability is more or less limited to the short term as the current sample would have to rely on context from samples in the not-so-distant past. It becomes difficult for RNNs to capture long term dependency as the further away the current input becomes from a past sample since it is less likely that the information will be preserved in the network.

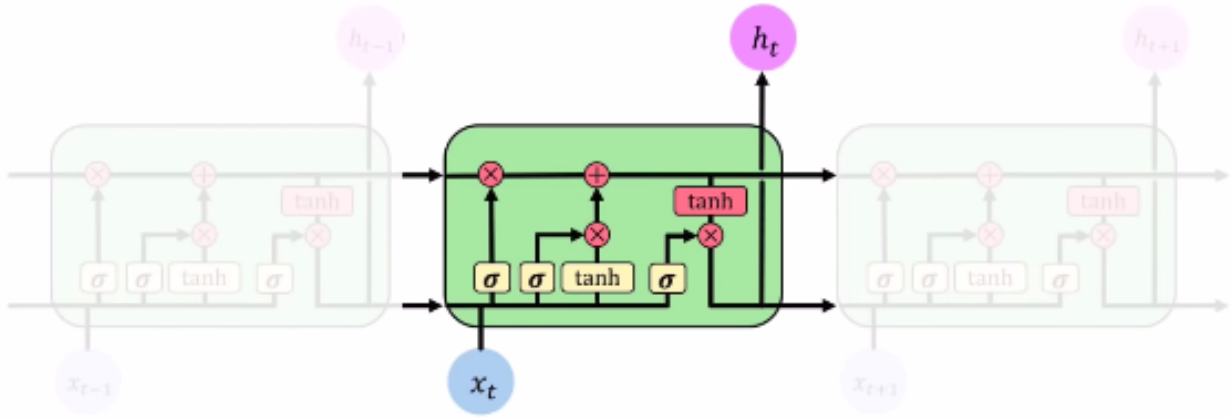

Figure 2.7: Long-Short Term Memory Neural Network. [5]

LSTMs improve on the RNN architecture through the introduction of learnable gates to implement recurrent feedbacks. This enhancement effectively allows gradient flow back to the past. These learnable gates control the flow of information and allow for the cell state

to be updated. This gives LSTM cells increased flexibility in deciding how prior and current information changes the cell state. Therefore information can persist for a long period as the network make selective choices on what to remember and what to forget.

In LSTM, the information flows are controlled by the gate mechanism and the cell state. This enables LSTM to freely add or remove information from the cell state; simply put, the LSTM has the ability to selectively remember or forget any information [12, 13]. This allows the important information to be preserved within the network for a long period; conversely, the information that deemed unnecessary could also be quickly forgotten.

In an LSTM cell, the state (C_t) is dependent on:

•The previous cell state (C_{t-1}), which represents the information carried on from the previous time step.

•The previous hidden state (h_{t-1}), which represents the output of the previous LSTM cell.

•The current input (x_t), which represents information given at the t time step.

LSTMs make use of gate mechanisms to allow for better control of information within the LSTM's cell. There are three main gates in a LSTM cell and they are composed of the forget gate, the input gate, and the output gate.

The Forget Gate, which is responsible for dropping information that has lost its relevancy from the cell state [13]. The output of this gate is computed as follows [12]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where

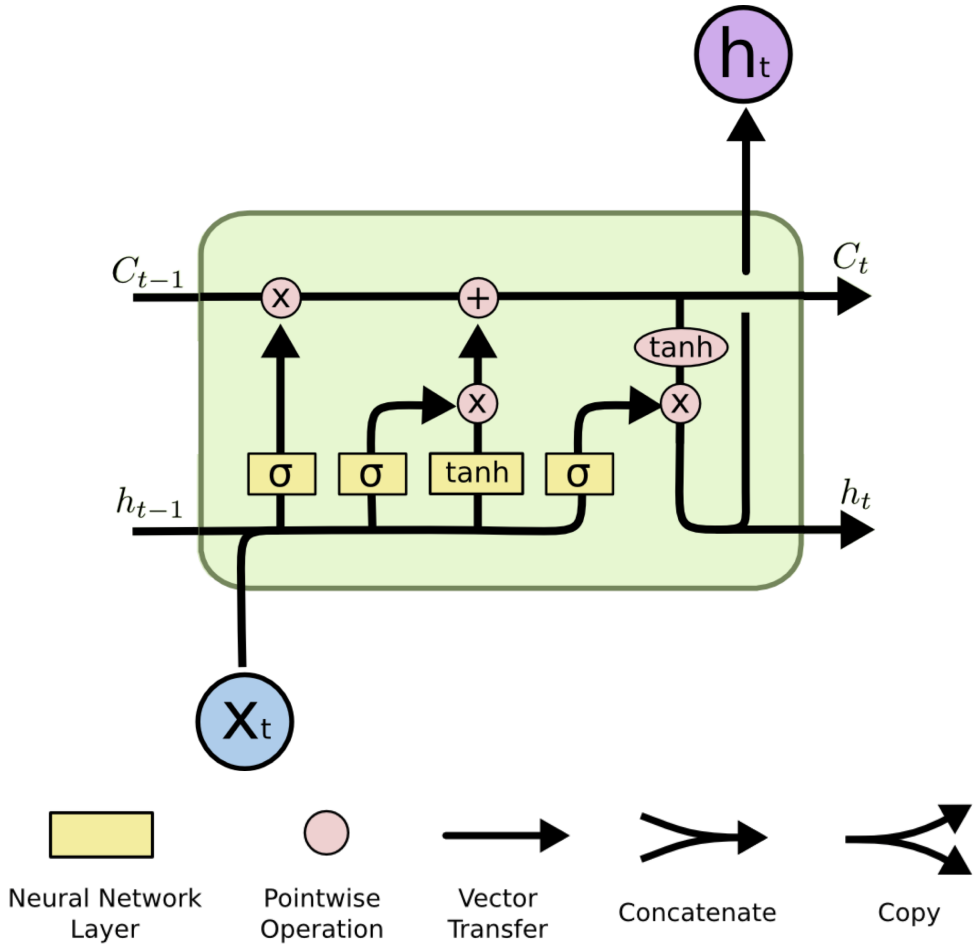

Figure 2.8: LSTM Cell.[5]

•h_{t-1} denotes an input; it is provided by the output of the previous cell at time step $t-1$.

• x_t denotes the input at current time step t.

•W_f represents the weight matrix.

•b_f represents the bias.

•σ denotes the activation function which in a standard LSTM would be the Sigmoid function.

•f_t is the output of the gate which ranges from 0 to 1. $f_t = 0$ results in the complete disregard of current input information and $f_t = 1$ results in the complete conservation of current input information.

The Input Gate, which is used to allow new information from the input to change the cell state [13]. The output of the gate can be computed as follows [12]:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\hat{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where

•h_{t-1} denotes an input; it is provided by the output of the previous cell at time step $t-1$.

•x_t denotes the input at current time step t.

•W_i and W_C represents the weight matrices.

•b_i and b_C represents the biases.

•σ denotes the activation function which in a standard LSTM would be the Sigmoid function.

•$tanh$ denotes the tanh function.

•i_t * $\hat{C}_t$ is the output of the input gate.

The Output Gate ultimately decides what is the assigned output of the LSTM cell at time

step t. The output of this gate can be computed as follows [12]:

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) * tanh(C_t)$$

where

•h_{t-1} denotes an input; it is provided by the output of the previous cell at time step $t-1$.

•x_t denotes the input at current time step t.

•C_t is an input to the gate and represents the current cell state.

•W_o represents the weight matrix.

•b_o represents the biases.

•$tanh$ denotes the tanh function.

•h_t is the gate's output and represents the current cell's hidden state.

•σ denotes the activation function which in a standard LSTM would be the Sigmoid function.

LSTMs have be shown to be an effective instrument in many application such as sequence modeling, speech recognition and machine translation.[12]

2.0.7 Convolutional Neural Network

Convolutional Neural Network (CNN) is a neural learning architecture that is most notated for its successful application in visual tasks such as image recognition, image classification and object detection. A CNN architecture is composed of an input layer, convolutional layers,

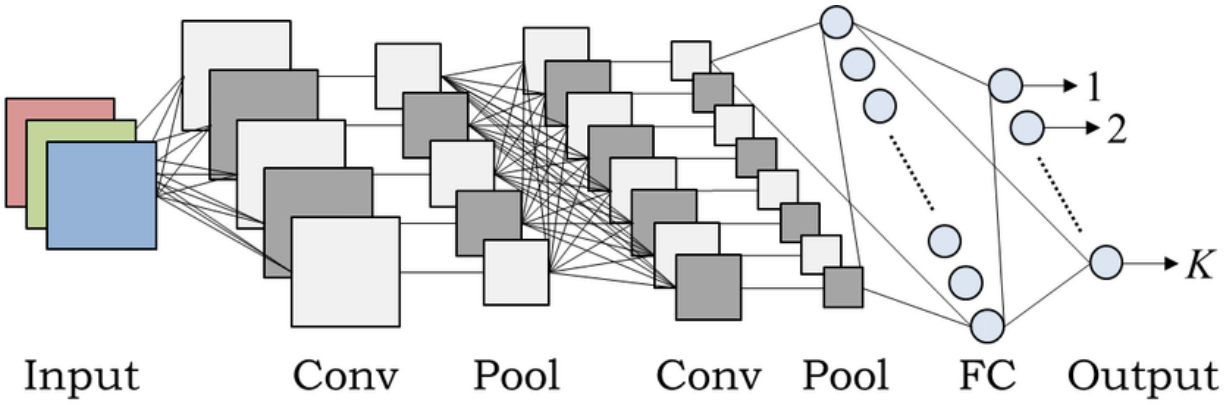

Figure 2.9: A Convolutional Neural Network. [6]

pooling layers, fully connected layers, and a final output layer. In addition to the mentioned layers, CNNs can also include normalization layers for use in classification tasks. They main characteristic in CNNs are the convolutional layers which are crucial for learning features from images and pooling layers which enable the reduction of dimensions in these feature maps. CNNs are insensitive to shifts in the raw data as well as distortions through the clever use of local connection, weight sharing and pooling.

Convolutional layer: The main mechanism used in this layer is the convolution operation. This involves multiplying a cross section of an image with a kernel. This kernel is essentially a filter matrix and that would continually stride and multiply with other sections of the image right until the end. While this convolution operation occurs, features are collected to form a feature map which then goes through a non-linear activation function.

Pooling layer: As mentioned earlier, when we have a large number of parameters or the input image is very large, we can reduce the number of parameters through a method called pooling. This down-sampling process in CNNs can be achieved through different approaches.

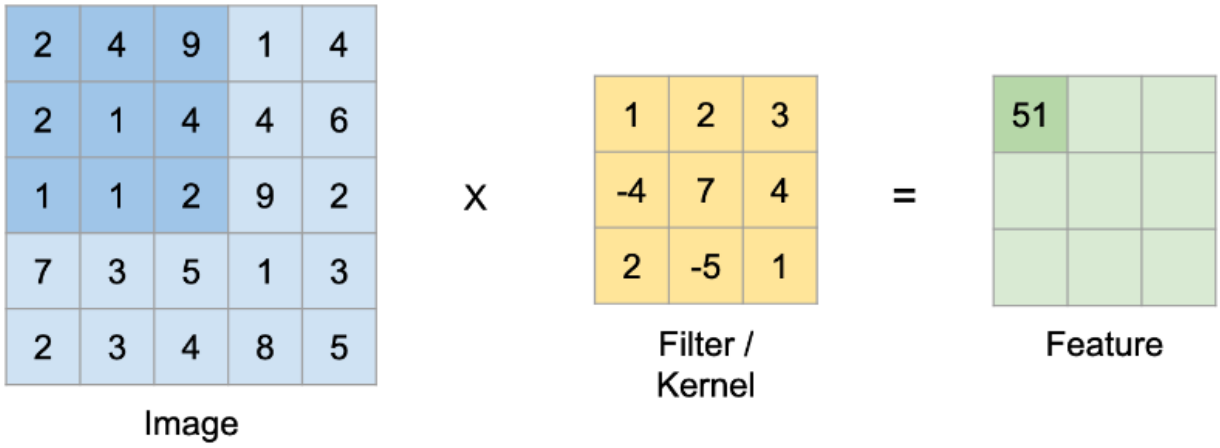

Figure 2.10: Convolution operation. [6]

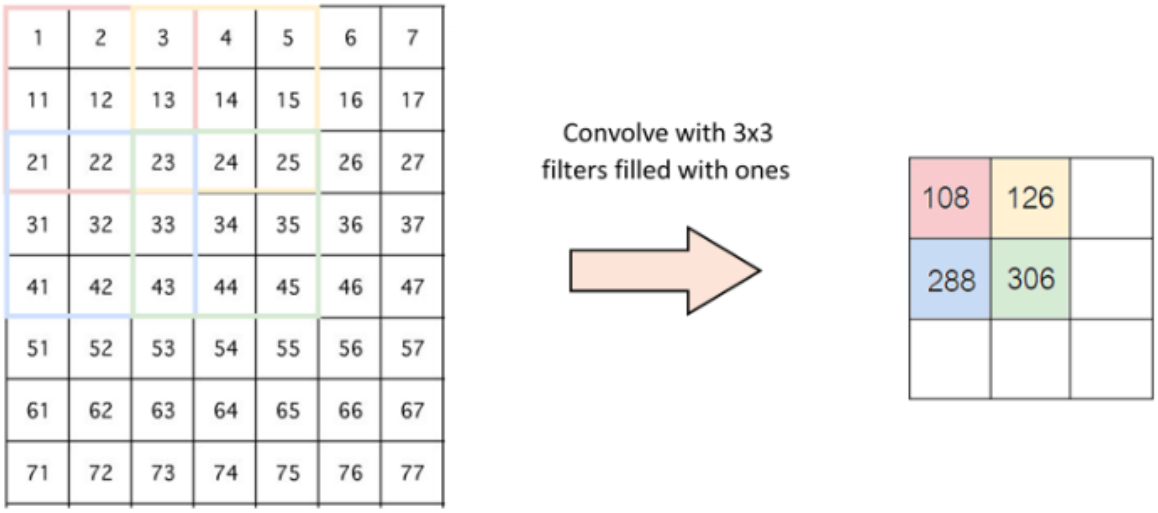

Figure 2.11: 3x3 convolution with stride of 2.[7]

Some of these include max pooling, average pooling and sum pooling. Max pooling is the most common as it has the added benefit of suppressing the noise. This can be achieved by simply taking the maximum value from each partition as shown in Fig 2.12 of the input matrix.

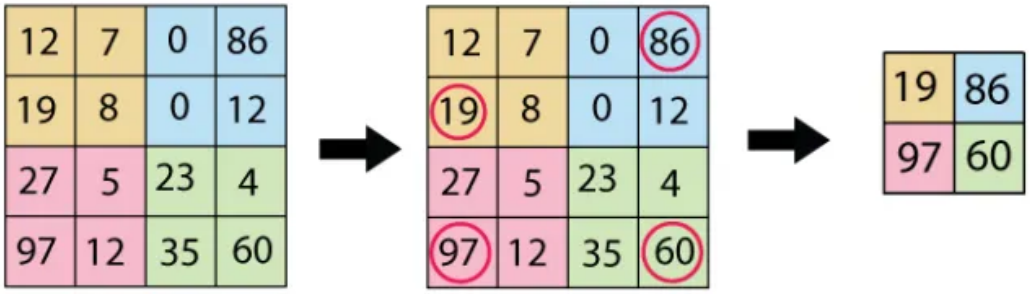

Figure 2.12: Max pooling. [8]

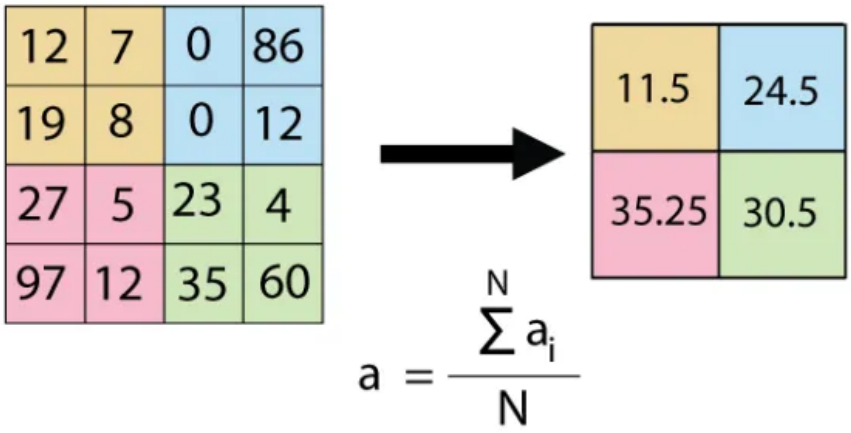

Figure 2.13: Average pooling.[8]

2.0.8 Temporal Action Proposals

Recently, deep learning models have performed exceptionally well in visual recognition tasks. One of these tasks called temporal action proposals is crucial in advancing the area of video comprehension. Temporal action proposal generation is a challenging task which aims to locate temporal regions in real-world videos where an action or event may occur. Recently, there

have been significant gains in performance through the adoption of deep learning models. The primary approach consists of a two-stream network that exploits visual cues from RGB images and horizontal and vertical motion cues that is obtained from stacked optical flow frames.[15]

In circumstances where large scale analysis of videos is required, the most interesting aspects are found through localizing and identifying human actions that occur in short intervals within long untrimmed videos.

Most models adopt a top-down approach where proposals of predefined duration are generated. This approach suffers from a lack of boundary flexibility as the proposal boundaries are not exceedingly accurate. Some methods alleviate these concerns by first determining temporal boundaries with a high degree of precision and then generating proposals using this information.[10]

In many large-scale video analysis scenarios, one is interested in localizing and recognizing human activities occurring in short temporal intervals within long untrimmed videos. Current approaches for temporal action proposals still struggle to handle large-scale video collections and efficiently addressing this task remains elusive to our visual systems.[16] This is in part due to the computational complexity of current action recognition approaches and the lack of methods that propose fewer targeted intervals in the video, where activity processing can be focused.

2.1 Temporal Convolution Based Action Proposal

Until recently, Recurrent Neural Networks (RNN) have been the straightforward option for tackling sequence modeling tasks given their capability to model temporal dependencies. However, architectures that are based on Convolutional Neural Networks (CNN) have been steadily rising in popularity for sequence modeling tasks. One model that is commonly used for the temporal action proposal task is Temporal Convolutional Networks (TCN). [17] The main characteristic of TCNs that make them well suited for sequence modeling is the fact that they can map a sequence of arbitrary length to an output sequence of the same length.[17] They also make use of casual convolutions which ensures that information from future to past does not leak into the network. [9] In order to map sequences of varying lengths, TCNs use a 1D convolutional network architecture where each layer starting from the input and hidden layers have the same length which is achieved through zero padding. The use of casual convolutions also ensures that an output at time step t is convolved with items from time step t and prior.

In additional to casual convolutions, TCNs also make use of dilated convolutions. Dilated convolutions can be considered as convolutions using uniformly sparse filters. So a large dilated convolution filter can have the same number of weights but have a larger receptive field than a standard convolution filter. A dilated convolution with a factor of 1 is equivalent to the standard convolution operation. Fig 2.14 displays how dilated casual convolutions are employed in a network with two hidden layers.

The advantages of using TCNs include lower memory usage since they share filters across a layer and the main factor in the back-propagation path is the number of layers in the network.

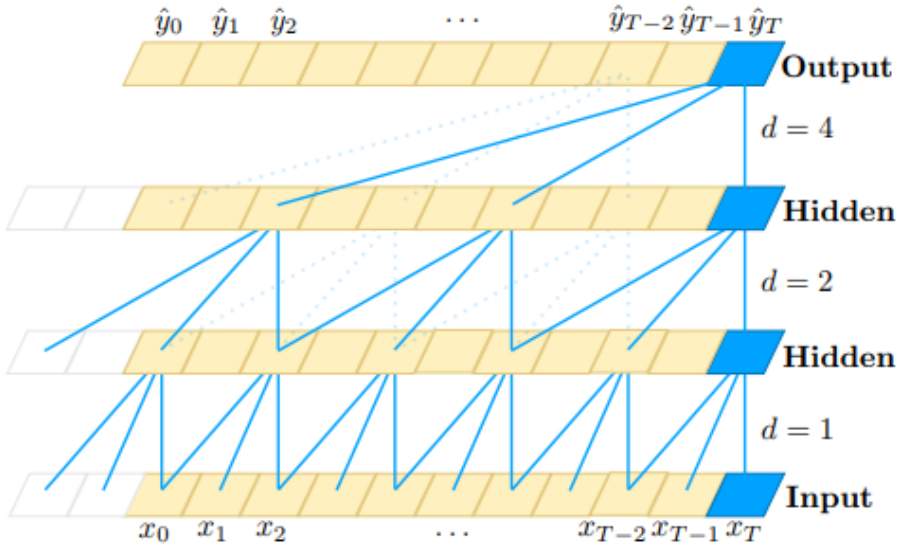

Figure 2.14: Dilated causal convolution with dilation factors d = 1, 2, 4 and filter size of 3. [9]

The convolution operation using filters can also help capture contextual information within a temporal window.

Temporal Convolution Based Action Proposal [18] is an approach that achieves state of the art performance in the ActivityNet temporal action proposal task. The primary mechanism they employ to obtain proposals are so called temporal anchors with 7 temporal feature maps with length: 1, 2, 4, 8, 16, 32, 64. They use these multi-scale temporal feature maps in conjunction with temporal convolutional layers to predict the overlap between video segments of varying lengths and the ground-truth proposals.

2.2 Boundary Sensitive Network (BSN)

Boundary Sensitive Network (BSN) is a network designed for temporal action proposal generation with high recall. Their method focuses on generating proposals with precise temporal

boundaries and high overlap with the ground-truth instances. They achieve this by adopting a "local to global" approach.[10] They define the local task as determining temporal segments that are highly likely to be an action boundary. These temporal boundaries are then used to construct a narrow set of proposals. To construct a proposal, first spatio-temporal features are obtained from a pre-trained two-stream network. These two-stream feature sequences are used as the input of BSN. These features are evaluated for the starting, ending and actionness probabilities of each temporal location using a multi-layer CNN. A proposal with a high starting and ending boundary probability is constructed by sampling the actionness sequence by linear interpolation with 16 points.[10] They also sample starting and ending regions of the actionness sequence using linear interpolation with 8 points. An example of how a proposal is generated is shown in Fig. 2.15.

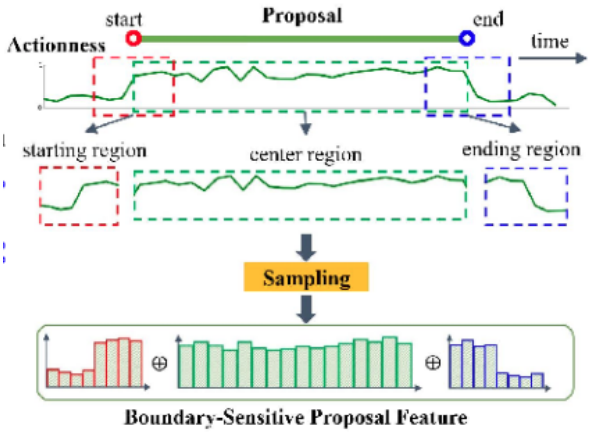

Figure 2.15: Construction of a Boundary Sensitive Network proposal feature. [10]

The global task can now commence and is defined as the evaluation of the proposal features. This is accomplished using a multi-layer perceptron model with one hidden layer. The hidden layer contains 512 units with RELU activation. The output layer generates the final score which

is used to determine which proposals are the most likely to contain an action and overlap with the ground-truth.

Chapter 3

Feedforward Sequential Memory Networks (FSMN)

In a variety of tasks including language modeling and speech recognition, it is crucial to model long term dependency. Recurrent neural networks (RNN) are popular given their ability to capture long term dependency within the sequential data using recurrent feedback. RNNs also have to capacity to learn mapping from one variable-length sequence to another. This is an advantage they have over FNNs which can only learn to map a fixed-size input to a fixed-size output. However, to capture the long term dependency in sequences, RNNs rely on back-propagation through time. This considerably increases the computational complexity of the learning due to the internal recurrent cycles. [11] Therefore it is desirable in some way or another to use FNNs to learn the long-term dependency in sequences. One proposal that attempts to just this is the unfolded RNN. [21] While this approach needs similar training time

as standard FNNs with better performance, it still leaves much to be desired. The unfolded RNN falls short due to its use of a limited number of unfolding steps in time. Furthermore, in more complicated recurrent neural architectures such as LSTMs, an unfolded architecture is much harder to formulate.

Recently, there have been several attempts to construct a neural model with varying types of memory units. One of these neural models features a type of memory unit that can be read or written. Another proposal called neural turing machines (NTM) connects the memory unit with resources that enable it to solve tasks such as sorting sets of numbers and other symbolic manipulation tasks. [22]

Feedforward sequential memory networks (FSMN) are a type memory neural network capable of learning long term dependency without using recurrent feedback connections.[11] While there are many well established approaches that can model the long-term structure within the sequential data, they rely on recurrent feedbacks such as in regular recurrent neural networks and LSTM networks. FSMNs capture long term dependency by extending the standard feedforward neural networks by means of introducing memory blocks in the hidden layers. FSMN implements recurrent feedbacks using learnable gates to ensure that the gradients can adequately flow back to the past. The main benefit is managing to avoid training using BPTT (Back-propagation through time) which increases the computational complexity and causes problems such as gradient vanishing and exploding.[12]

Feedforward sequential memory network (FSMN) is a standard feedforward neural network usually with a single memory block in the hidden layer. Given a sequence, $X = \{x_1, x_2, ..., x_T\}$,

each $x_t \in \mathbb{R}^{D\times 1}$ represents an input data at time instance t. The corresponding hidden layer outputs are denoted as $H = \{h_1, h_2, ..., h_T\}$. Through the use of memory blocks with a tapped-delay line structure, we can encode the long context information into a fixed-size representation. We can h_t and its previous N histories in the memory block with the equation (1). A single memory block can represent h_t and its N previous histories into a fixed-size representation as follows:

$$\tilde{\mathbf{h}}_t = f(\sum_{i=0}^{N} a_i \cdot \mathbf{h}_{t-i}) \tag{1}$$

where all coefficients form an N-dimension learnable vector $a = \{a_0, a_1, a_2, ..., a_N\}$ and $f(\cdot)$ represent the activation function which is usually sigmoid or RELU.

The output from the memory block can be fed into the next hidden layer and we compute the activation of the units in the next hidden layer as follows:

$$\mathbf{h}_t^{l+1} = f(W^l \mathbf{h}_t^l + \tilde{W}^l \tilde{\mathbf{h}}_t^l + b^l) \tag{2}$$

At the beginning of every training instance, the memory is flushed. In our model, we include two FSMN hidden layers with memory blocks as shown in Fig 2.1.

The aforementioned equations only store past information in the memory block and is thus referred to as an unidirectional FSMN. This model can be extended to a bidirectional version that has lookahead window into inputs in the future using equation (3).

$$\tilde{\mathbf{h}}_t = f(\sum_{i=0}^{N_1} a_i \cdot \mathbf{h}_{t-i}) + f(\sum_{j=0}^{N_2} c_j \cdot \mathbf{h}_{t+j}) \tag{3}$$

where

• N_1 is the length of the lookback window which represents the number of previous items from the past to factor in.

• N_2 is the length of the lookahead window which represents the number of upcoming items from future time steps to factor in.

• c denotes the lookahead coefficient from a set of N_2 different vector coefficients.

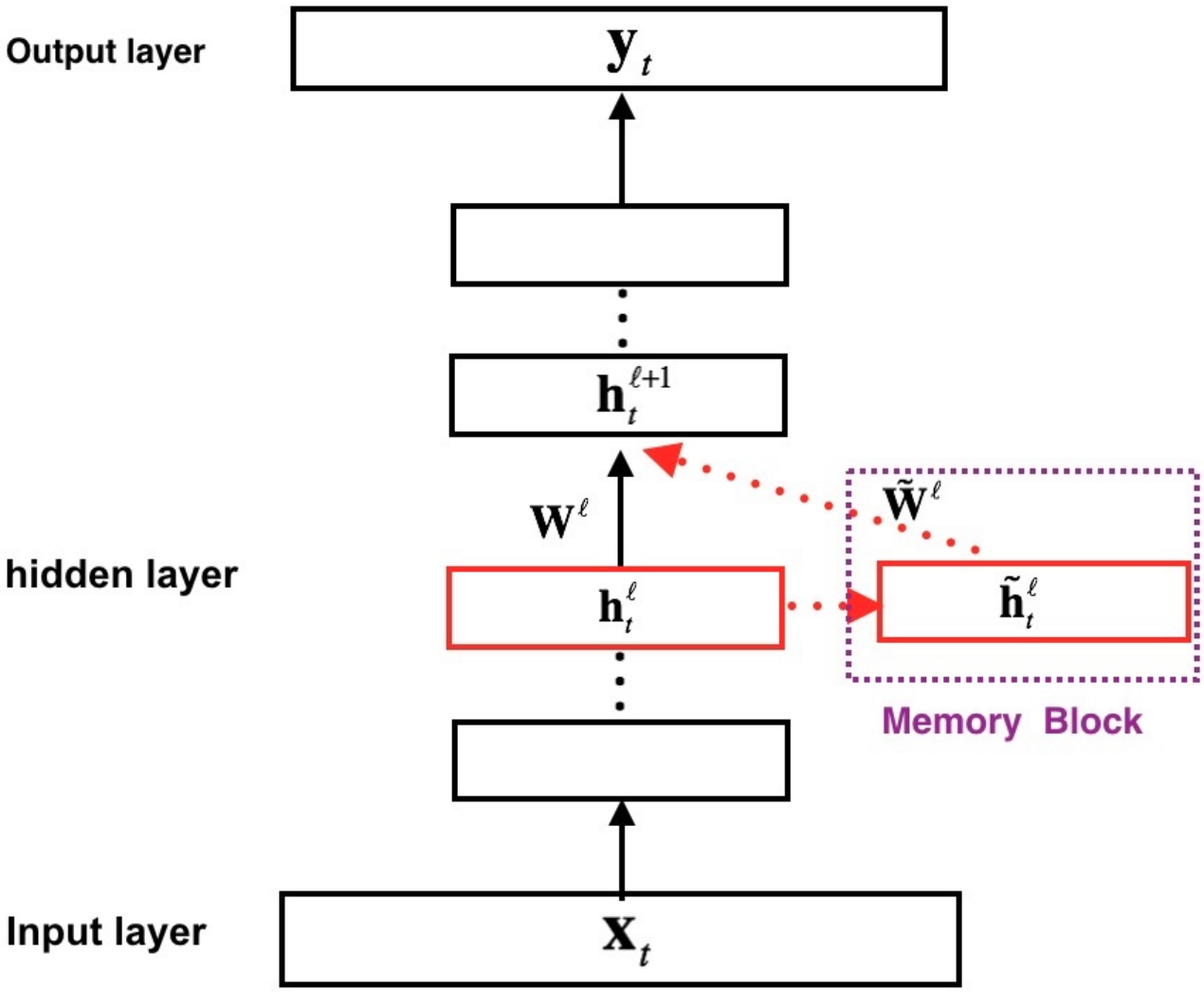

Figure 3.1: Feedforward Sequential Memory neural network [11]

3.0.1 Compact Feedforward Sequential Memory Network (cFSMN) Structure

Recently, a modification to the standard FSMN architecture has been introduced called compact feedforward sequential memory networks (cFMNs). A cFSMN is similar to FSMN in that it is essentially an FNN equipped with memory blocks. The main difference is the introduction of special linear projection layers called cFSMN-layers. In addition to the linear projection layer and memory block, the cFSMN-layer also comprises of a weight connection between the memory block and the next hidden layer in the network. There is no weight connection from the linear projection layer to the next hidden layer as a measure to simplify the FSMN architecture. These modifications can be represented using the following memory block encoding equations for unidirectional and bidirectional cFSMN.

$$\mathbf{p}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b} \tag{4}$$

$$\tilde{\mathbf{p}}_t = \mathbf{p}_t + \sum_{i=0}^{N_1} a_i \cdot p_{t-i} \tag{5}$$

$$\tilde{\mathbf{p}}_t = \mathbf{p}_t + \sum_{i=0}^{N_1} a_i \cdot \mathbf{p}_{t-i} + \sum_{j=0}^{N_2} c_j \cdot \mathbf{p}_{t+j} \tag{6}$$

where $\mathbf{p}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}$ is the output of the linear projection layer. In the above equations, we use $\mathbf{p}_t$ to represent all the memory block computations at time instance t. We use the output of the memory block to calculate the activation of the units in the next hidden layer.

Similar to FSMNs, compact feedforward sequential memory network are able to capture

long-term information and can also be trained using standard back-propagation which is faster and less susceptible to problem such as gradient vanishing.

3.0.2 Feedforward Sequential Memory Networks (FSMN) Implementation

The recurrent neural network layer can be viewed as a first-order infinite impulse response (IIR) filter. While the memory blocks in FSMNs can be viewed as an N-th order finite impulse response (FIR) filter. It should be noted that IIR filter like recurrent neural networks are more difficult to train since they learn through back-propagation through time and that first-order IIR filter can be approximated by a high-order FIR filter. The FIR-like memory blocks in FSMNs is used to encode long context information into a fixed-size representation, which helps the model to capture long-term dependency as an alternative to RNNs.

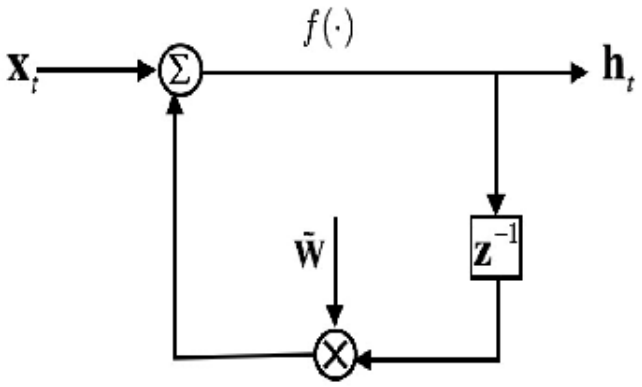

Figure 3.2: Recurrent layer as IIR filter. [12]

FSMNs can be constructed by taking a standard feedforward network and augmenting memory blocks that follow a tapped-delay line structure as in FIR filters. These memory blocks are connected to hidden layers in the FNN.

FSMNs use learnable coefficients to encode the past context within a pre-defined window into a fixed-size representation. The final representation is a weighted sum of the hidden

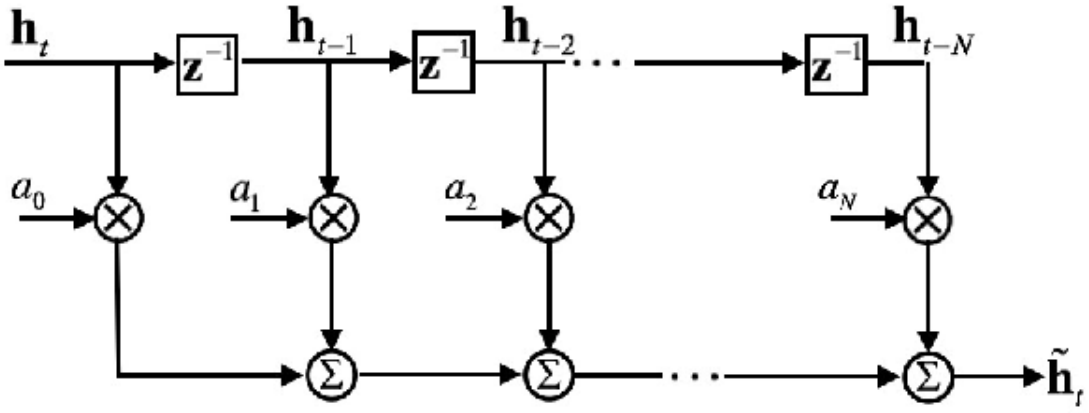

Figure 3.3: Illustration of a feedforward sequential memory network (FSMN) and its tapped-delay memory block. (Each z^{-1} block stands for a delay or memory unit) [12]

activations of all previous N time instances shown in Fig 3.3 as a tapped-delay structure.

A forward pass of an FSMN can be computed using sequence by sequence matrix multiplications. Using only one matrix multiplication, we can compute the sequential memory operations in for the whole sequence as follows.

$$\tilde{\mathbf{H}} = \mathbf{H} \cdot \mathbf{M} \tag{8}$$

where $\mathbf{H}$ is a matrix composed of all hidden activations for the whole sequence and $\tilde{\mathbf{H}}$ is the memory block outputs for the whole sequence. Given a batch consisting of K sequences, i.e., $\mathscr{L} = \{X_1 \cdots X_K\}$. We compute the memory outputs for all K sequences in the mini-batch as follows:

$$\tilde{\mathbf{H}} = [H_1, H_2 \cdots H_K] \begin{bmatrix} \mathbf{M}_1 & & & \\ & \mathbf{M}_2 & & \\ & & \ddots & \\ & & & \mathbf{M}_K \end{bmatrix} = \bar{\mathbf{H}}\,\bar{\mathbf{M}} \tag{9}$$

where each $\mathbf{M}_k$ can be constructed for a N-th order scalar FSMN using N + 1 coefficients as such:

$$\mathbf{M} = \begin{bmatrix} a_0 & \cdots & a_N & 0 & \cdots\cdots & 0 \\ 0 & a_0 & \cdots & a_N & \cdots\cdots & 0 \\ \vdots & \cdots & \ddots & & \ddots & \vdots \\ 0 & \vdots & & a_0 & \cdots & a_N \\ \vdots & \cdots & & & \ddots & \vdots \\ 0 & \cdots & & 0 & & a_0 \end{bmatrix} \tag{10}$$

Chapter 4

Incorporated Temporal Action Proposal Generation

While top-down approaches produce satisfactory results, the lack of parallelism and inefficient evaluation modules result in models that are cumbersome. In practice, there are large amounts of prerecorded untrimmed video that needs to be analyzed efficiently to localize key actions. Thus, we have come up with a better approach that reduces latency by adding parallelism in the proposal generation process and adopting units that more efficient and are better suited to model sequential data. We call our approach the "Incorporated temporal action proposal generator network" and provide the details of the model below.

The main idea of Incorporated Proposal Generation is to concurrently compute temporal boundary probabilities and generate confidence scores of varying starting and ending windows. In doing so, we develop a method to efficiently capture sequential data and generate confidence

scores for a large number of candidate proposals. Our design consists of end-to-end temporal proposal generation where temporal boundary probability and proposal confidence scores are generated in two separate branches of our Incorporated model. We then train the modules jointly in the form of a multitask loss function.

Our training objective consists of temporal boundary classification loss and proposal evaluation classification loss. The advantage of this method is that we can generate proposal scores of multiple candidate proposals as opposed to evaluating them individually as executed by other state of the art methods.

A detailed view of our improved design is shown in Fig. 4.1. We show in our experiments that jointly training both modules leads to improved proposal generation speed and performance gain. Alternate approaches are based on cost volume by combining two feature maps, using feature concatenation. Thus, generated proposals are made up of concatenated temporal starting and ending boundary features.[10]

4.1 Model Overview

4.1.1 Boundary Predictions

In Fig. 4.1, this module is highlighted in red and represents the classification of individual visual segments. These segments are evaluated using a Convolutional Neural Network and two boundary classification scores are computed. The first being the starting boundary probability score which indicates how likely this segment is the beginning of an action sequence. The

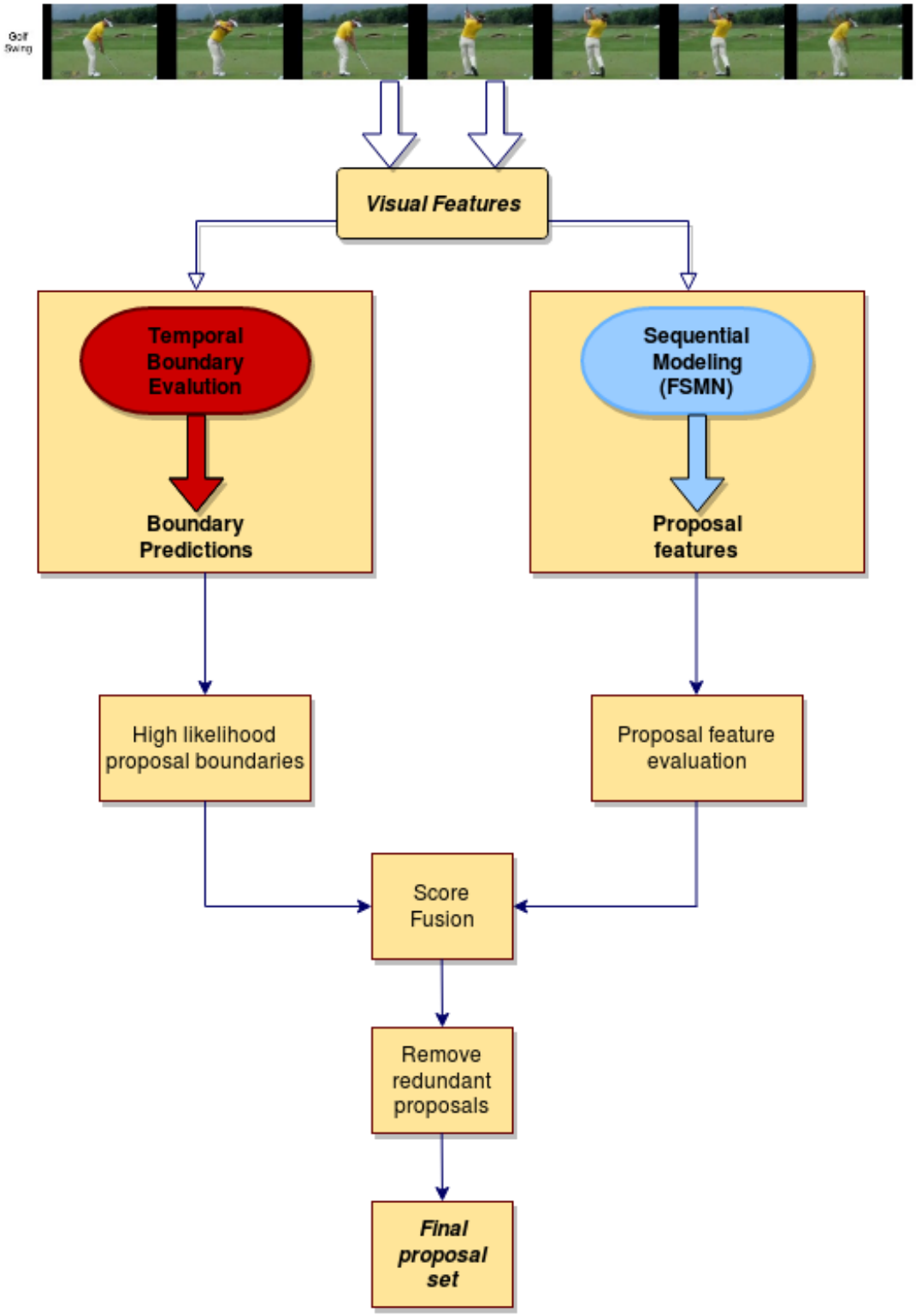

Figure 4.1: Incorporated Temporal Action Proposal Generation model overview

second score computed is the ending boundary probability score. This evaluation is crucial as it allows for increased precision when constructing proposals through better boundary selection.

4.1.2 Proposal Level Features

In Fig. 4.1, this module is highlighted in blue and represents the encoding of a visual sequence into a fixed-sized representation using Feedforward sequential memory networks (FSMN). We feed our FSMN a sequence of visual features representing a window within the video clip. The first input to the FSMN denotes the start of the window and proposal features are generated at each iteration subsequently. The output of the FSMN yields proposal level features for temporal windows based on the input visual feature sequence. The process is run recursively for different starting boundaries in order to generate proposal features for wide range of candidate proposals. The proposal features are further evaluated using a multilayered CNN to produce a classification score indicating the likelihood of an action within the visual sequence they represent. Since there could be many proposals that are likely to contain an action, we refine our candidate proposal set using our boundary predictions which we elaborate in later sections.

4.1.3 Merging and trimming

This part of our Incorporated model consists of score fusion and removal of redundant proposals. We utilize this section to generate our final set of proposals. As displayed in Fig. 4.1, we merge the scores from our boundary predictions module with the results of the proposal feature evaluation. This process is necessary to reduce the number of proposals and to preserve the

candidate proposals that are more likely to be similar to ground-truth instances. Furthermore, this step is important since we have evaluated a lot of proposal features generated using FSMN. We discard proposals that represent huge chunks of visual sequences that are devoid of any interesting actions. Lastly we recognize that we could have multiple proposals that have overlapping windows that represent similar sequence of frames. This is dealt by removing highly redundant proposals. The first step is to determine how significant is the overlap. If the length of overlap of two proposals is substantial, we decay the scores of the proposal with the lower score. After discarding proposals with scores below the threshold value, we retain our final set of proposals.

4.1.4 Intuition behind Incorporated Temporal Action Proposal Generation

The concept behind this model is to connect the rigorous examination of small temporal segments with longer representations contained in proposal features. This grants us the ability to cut down on indispensable memory and time needed to finish training our model.

A key design aspect is to speed up the training process by jointly training boundary probabilities and proposal level features. One of the goals of our design is to capitalize on the abundant ground truth labels that not only provide information on the locations of actions themselves but also the setup to actions. We make the most of this data by utilizing two modules to generate temporal action proposals.

Proposal features offer us insight as to whether an action occurs within a temporal window

but will not reliably give us enough precision that is necessary to obtain a competitive score. We require instruments to further refine and assist in improving the quality of our proposals. This is possible thorough an evaluation module that encapsulates the analysis of potential action boundaries. Thus, we form a secondary module called temporal boundary evaluation. This module will compute the probability of a single image segment being the start of an action sequence or the end of an action sequence.

The added benefit of jointly training the modules is the increased parallelism that can be utilized. This architecture leads to further training to the minimum. Previous approaches that rely on a bottom-down approach have to exhaustively classify individual image segments first before they can begin to construct proposals. Our Incorporated temporal action proposal generator finely combs through individual segments from a video while simultaneously generating high level features for the candidate proposals. We validate this approach is more optimal than using back-propagation through time for training. In the final step of the process to generate proposals, we begin unification by prudently merging the scores generated by the modules.

4.1.5 Visual Feature Encoding

To generate quality temporal action proposals, we require highly relevant input visual sequence features for our model to process. We opt to use features generated from a pre-trained two-stream ResNet model. We gather the output scores from the last fully-connected layer of the two-stream network and concatenate them to obtain visual features for each frame. The output features contain visual clues from RGB frames that provide various spatial information. They

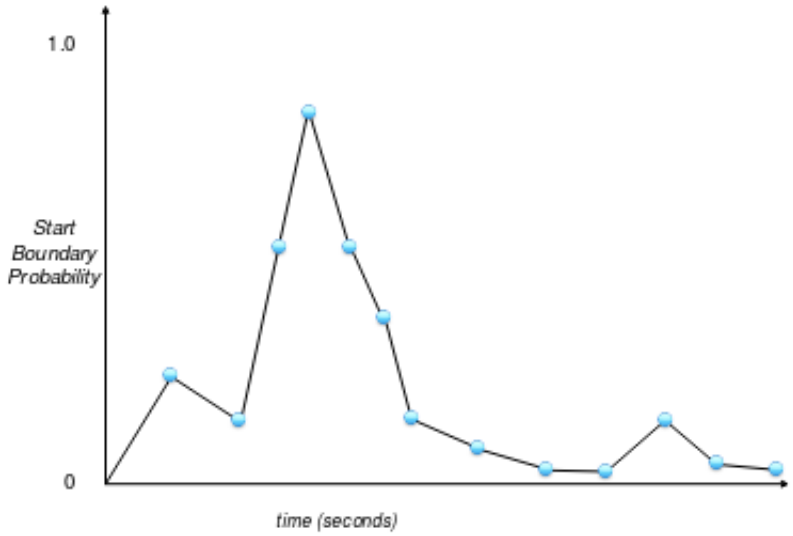

Figure 4.2: Example showing starting boundary probabilities computed for each temporal segment.

also contain temporal clues from optical flow frames that allow insight about motion between pairs of frames. To generate an adequate number the input features for our Incorporated model, we extract these features in a regular time interval σ, such that $feature\ length = video\ length\ /\ \sigma$. We set the interval to 16 for our experiments.

Separating the spatial net and the temporal net also allows us to take advantage of the large ImageNet challenge dataset[15]. The spatial network is pre-trained on available annotated image data and exploits the deep ConvNet image classification architecture . The pre-trained spatial stream can help provide information from individual frames about objects and scenes which is beneficial in determining temporal action boundaries. Since certain actions are highly correlated with specific objects, the spatial recognition of these objects will aid in procuring accurate temporal action proposals.

The temporal ConvNet on the other hand takes in stacked optical flow displacement fields

computed from pairs of video frames. The displacement fields can be further broken down into horizontal and vertical vector fields, d_t^x and d_t^y. They are stacked together to form separate input channels.

The motion stream allow us to differentiate groups of frames that are visually similar through the exploitation of optical flow. These sets of displacement fields can allow us to determine various actions that occur in the video that is not easily established by solely looking at static RGB frames. Furthermore, our experiments reveal that motion stream ConvNet performs even better in action recognition tasks compared to spatial ConvNets, thus highlighting their importance in video classification related tasks.

When comparing the features from temporal ConvNet, it is important to note that certain hand-crafted features such as motion boundary histograms (MBH) and histogram of optical flow (HOF) can themselves be generalized from optical flow and its gradients using convolutional filters and pooling layers. [15]. There is also a proven history of success demonstrated with the use of two-stream architecture in video action classification tasks. In the UCF101 task, the ResNet based deep two-stream model achieves a performance of 94%.

Let $feature(t_i)$ represent the spatio-temporal input at time instance t_i,

$$feature(t_i) = Concat(SpatialNet(t_i), TemporalNet(t_i))$$

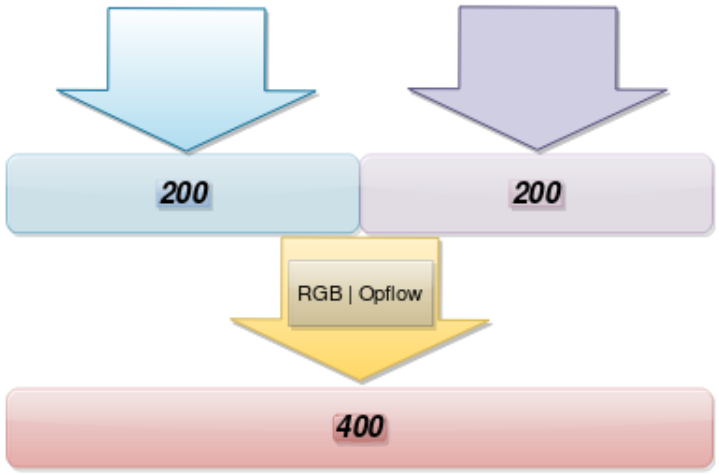

Figure 4.3: Concatenated RGB and Optical Flow features.

4.1.6 Linear projection

State of the art proposal generation models like BSN incorporate multiple wide convolutional layers in their design to capture global video level features with a large degree of temporal context.

When training a large neural network with many hidden units, it can be very slow to learn the large number of parameters in the weight matrix between the input and hidden layers. Assuming hidden layers of H dimensions and input of D dimensions, the ensuing weight matrix between the input and hidden layer will be of size $O(DH)$. Due to the number of parameters and costly matrix multiplications, the rate of convergence of gradient descent during training is time-consuming.

To alleviate this problem, we introduce a linear layer with m linear units between the input

and hidden layer to boost training speed.[19] We thus represent the weight matrix W as the product of two matrices, $U \in \mathbb{R}^{D \times m}$ and $V \in \mathbb{R}^{m \times H}$, where $m < H$. Using back-propagation, we can learn these weights using much less memory. Since we reduce the total number of learnable parameters, the weight matrix between the hidden layers is now of size $O(m(D+H))$ from $O(DH)$.

4.1.7 Depth channel reduction

When evaluating proposal features and boundary probabilities, experiments show that 5×5 and 3×3 convolutions on visual features are immensely valuable. However, the computational demand required to perform them makes it very difficult when memory is limited. We deal with this obstacle by addressing the sizable filter space dimensionality. The authors in [14] remarked on the success of low dimensional embeddings and their ability to contain information about a relatively large image patch. We acknowledge these sentiments and implement bottleneck units in-between our network structure. These units are composed of two 1×1 convolution layers with 5×5 or 3×3 convolution layers in between. The first 1×1 convolution layer is used to reduce the number of input feature maps while the latter is used to revamp the number of output feature maps.

4.1.8 Sequential Modelling

For any given pair of starting and ending indexes up to the max duration index, we obtain features that represent the sequence of visual data contained within that temporal window.

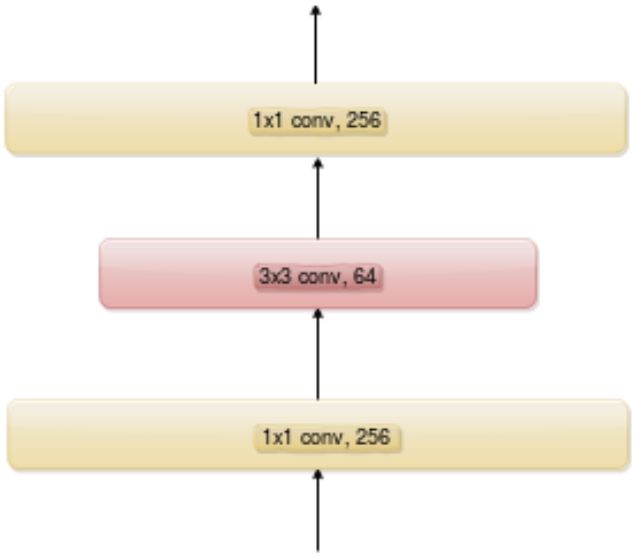

Figure 4.4: Feature map reduction using 1×1 convolution

These temporal windows can be thought of as proposal candidates. To obtain features for candidate proposals at varying starting indexes, we apply a mask to input features in the temporal dimension. This would then allow us to model the sequential data for various starting and ending indexes using our Feedforward Sequential Memory network.

Given a collection of visual features, every possible candidate proposal in a video up to a maximum duration length is evaluated and given a confidence score.

Visual features are fed into an FSMN sequentially from the initial starting boundary and the output of the network at each iteration is used as proposal level features for a specific temporal window. The process is then repeated recursively for all the possible starting boundary locations. Each time a new initial staring boundary is fed in the FSMN, all alpha parameters are zeroed out and all memory units are flushed out.

Harking back to the convolution optimizations described earlier, we employ multiple bottlenecked convolutions. Beginning with a 5×5 convolutional layer, followed up with a 3×3 convolutional layer. In our models, we experimented with multiple convolutional layers with performance gains as the depth increases. However, while increasing the network depth, the training duration increases as well. As does the total number of trainable parameters but the boosts in performance drastically diminish. In our model, we opt to only have three bottlenecked convolutional layers to achieve top performance while keeping memory usage low. The final convolutional layer outputs the candidate proposal confidence scores.

These confidence scores are computed and stored in a matrix. Each point in this matrix represents all the different starting and ending points that are possible in a video. The columns of the matrix represent various proposal starting boundaries and the rows represent the duration length of the proposal. These confidence scores are trained using classification loss with a proposal being positive when they overlap the ground truth with a threshold of 0.7.

4.1.9 Computing scores

We jointly train the boundary prediction and proposals scores using the following multitask loss function $L_{TOTAL} = L_{start} + L_{end} + L_{fsmn}$ where L_{start} is the starting boundary loss, L_{end} is the starting boundary loss and L_{fsmn} is the proposal feature loss. L_{TOTAL} represents the total loss as the sum of the classification losses from the separate modules.

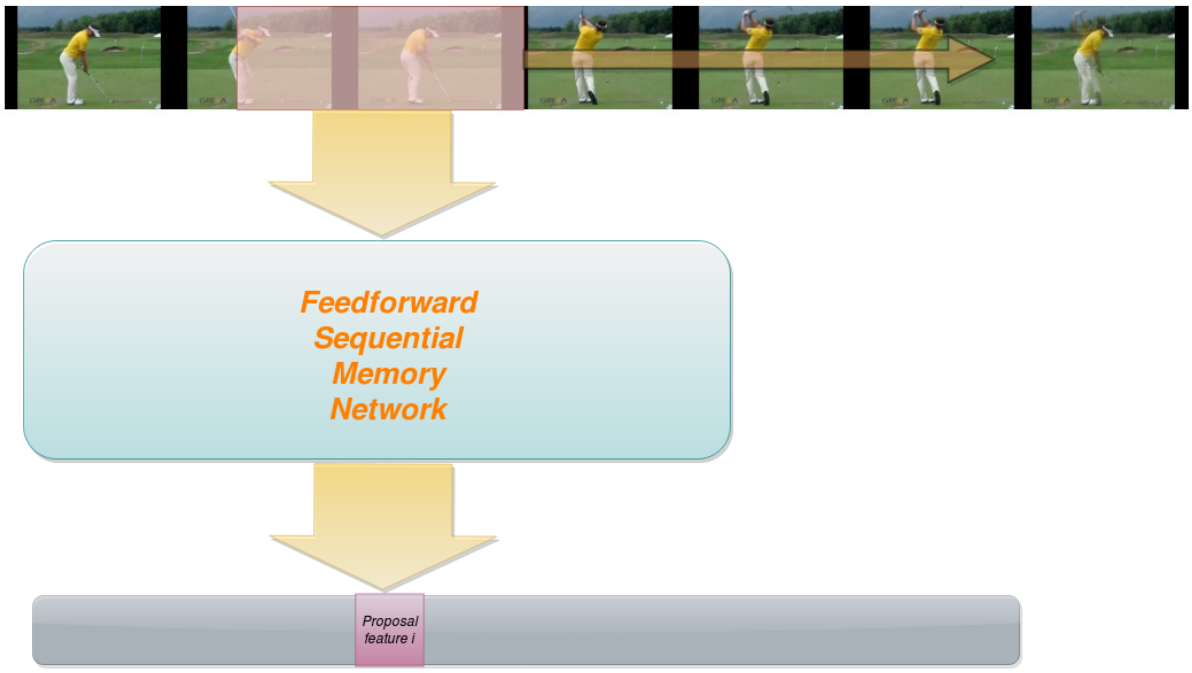

Figure 4.5: Example of how FSMN is used for proposal feature generation.

4.1.10 Collecting high likelihood boundaries

During the inference stage, we obtain results regarding the starting and ending boundary likelihood at each time instance t from the first segment of a video up to its maximum duration. To truncate the total set of candidate proposals, we assess the boundary probabilities at each location and locate positions where the probability is high, starting with $StartBoundaries$. Likewise, we do the same for all ending boundary probabilities in $EndBoundaries$ that are high. In doing so, we collect a more concrete set of candidate proposals. We start by first pairing starting locations from the set $StartBoundaries$ with ending locations from the set $EndBoundaries$ while ensuring that that the latter represents an ensuing time instance after

t_{start}. We define a proposal with a starting and ending boundary at as follows.

$$CandidateProposal_i = \{t_{start}, t_{end}, P_{start}, P_{end}\},$$

where P_{start} represents the starting boundary probability at time instance t_{start} and P_{end} represents the ending probability of an action at time instance t_{end} respectively. Now that we have manageable set of candidate proposals, we initiate a comprehensive review using our proposal feature evaluation.

4.1.11 Merging scores

To better evaluate the quality of our proposals, we need to fuse our temporal boundary probabilities with the scores from our proposal feature evaluation model. We can merge the scores of any given candidate proposal by multiplying their starting and ending probabilities with the feature evaluation score. Thus, we define this merged score as the likelihood that an action is contained within a temporal window defined by a clear starting and ending boundary.

$$FusedScore = ProbabilityStarting * ProbabilityEnding * ConfidenceScore$$

4.1.12 Removing redundant proposals

To further narrow our set of candidate proposals, we rid of redundant proposals. Doing so will enable us to achieve better recall scores with fewer and more accurate proposal candidates.

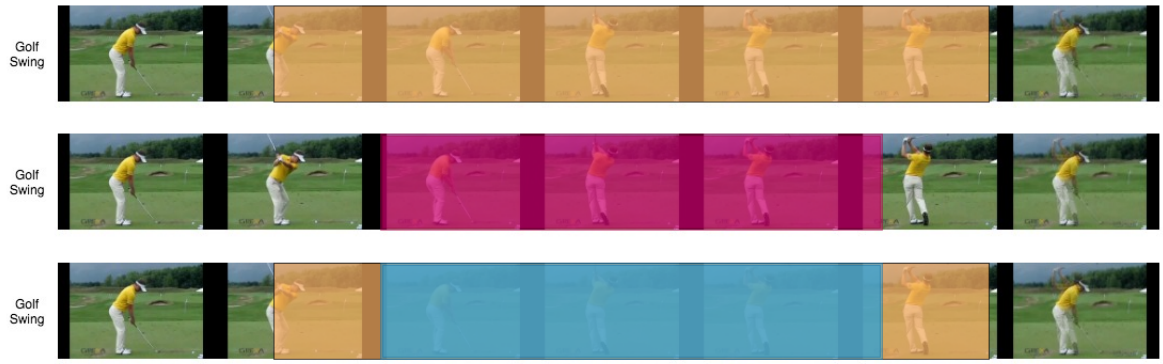

Figure 4.6: Intersection highlighted in blue for two overlapping proposals

One way to achieve this is to look at proposals with a high degree of temporal overlap. First we sort all the proposals by their evaluated scores. Then we apply an exponential decay algorithm to suppresses redundant proposals.[10] The scores of highly overlapped proposals are then decayed. This step is recursively applied to the remaining proposals to generate a truncated set wherein no Intersection over Union (IoU) between two proposals is greater than a fixed parameter value of 0.5. We obtain the final confidence scores by multiplying the original evaluated scores with the suppressed score. Any candidate proposals with a final score below the threshold value of 0.7 are removed from the final set of proposals.

Given two overlapping proposals, where $ConfidenceSore(p1) > ConfidenceScore(p2)$ and

IoU is defined as:

$$IoU = \frac{Lengthof Overlap}{Lengthof Union}$$

We can generate our final confidence scores using the following algorithm.

$$ProposalSupressed = e^{-2.5 * IoU(p1,p2)}$$

$$Proposalfinal = ConfidenceScore * ProposalSupressed$$

Chapter 5

Incorporated Proposal Generation for Temporal Action Proposals

5.1 Improving Efficiency and Dimensionality Reduction

In this section, we evaluate the effectiveness of our proposed methods on several existing state of the art methods. When comparing our method with existing approaches, we not only measure performance but also analyze the total number of learnable parameters. We use the same hardware for all our experiments including the utilization of two Nvidia GPUs. To analyze the efficacy of our sequence modelling methods, we compare it with a two-layer LTSM architecture. We also analyze the effects of bottlenecks in our architecture and report the difference in performance.

5.2 Experiments

5.2.1 ActivityNet-1.3 Dataset

ActivityNet-1.3 dataset contains temporally annotated untrimmed videos with 200 action categories.[16] The categories of the actions are diverse and many include different activities. They include sporting activities and everyday tasks such as drinking coffee, shoveling snow etc. There are relatively few categories that contain videos where numerous individuals occupy the scene. The ActivityNet dataset hardly contains any obscure action categories, with most of them easily relatable.

A majority of videos have a duration between 5 and 10 minutes and about 50% of the videos are in HD resolution.[16] It differs from other video related datasets primarily with its large number of action categories while also being completely untrimmed as most of the videos are scrubbed from an online source. Factors such as camera motion and video quality play a role in how difficult it is to discern the actions within videos. Since the dataset contains human annotations and does not utilize fully automatic annotation algorithms, the labels are likely to be sound.

5.2.2 Training

The ActivityNet 1.3 dataset contains 9532 training samples and 4728 validation samples. [16] We use the subset of training videos and use the ResNet101 features and optical flow features as input to our model. We train our model for a total of 9 epochs and validate our model after

each epoch. Since our model has fewer learnable parameters and uses less memory, we take take advantage of this by increasing our batch size to 64. We run our model using two Nvidia GPUs and divide the batch between the two devices. The initial learning rate is set to 5.0e-3 with the weight decay is set to $1e{-}4$. We optimize our objective function using gradient descent to train our model. We set our step decay scheduler for learning rate annealing every 6 epochs. Once the training is finished, we evaluate the proposals that our model generates by computing average recall using multiple IoU thresholds. As shown in Fig. 5.1, the IoU thresholds used for evaluating our proposals are $[0.5 : 0.05 : 0.95]$. Following the aforementioned computations, we generate our final score using the evaluation metric, Area under the (AR vs. AN) Curve (AUC). We do this by first calculating average recall (AR) multiple times using different average number of proposals (AN) and then calculate the area under the (AR vs. AN) curve. A proposal is a true positive if it has a temporal intersection over union (tIoU) with a ground-truth segment that is greater than or equal to a given threshold of 0.7. Fig. 5.1 shows the average recall scores computed using varying average number of proposals.

5.2.3 Label

For training temporal boundary probabilities, we need two temporal boundary label sequences $label_{start}$ and $label_{end}$. These are generated from ground truth proposals where each instance is represented as (t_{start}, t_{end}). We have separate label sequences to classify individual segments as a starting boundary for an action or an ending boundary for an action. The starting boundary

and ending boundary label label sequence can be denoted as:

$$label_{start,end} = \{t_{start,i}, t_{end,i}\}_{i=0}^{N}$$

For evaluation of candidate proposal features, we use a different set of labels from which we generate a label map. The ActivityNet dataset contains ground-truth information pertaining to different proposal instances. They are each denoted by a starting and ending location. Since we could have multiple ground-truth instances for a single video that represent actions at different locations. We compute a label map that allows us to access similarity scores for any given candidate proposal.

This label map contains the maximum Intersection over Union (IoU) score computed using ground truth instances and all possible candidate proposals. Given a proposal $P_{m,n}$ where m and n represent the start and end time respectively. We calculate the Intersection over Union (IoU) between the proposal and all ground-truth instances for that video:

$$IoU(G_i, P[m, n])$$

where

- IoU measures the similarity between the ground-truth instance and the proposal and is defined as the ratio of the overlapping area of ground truth and predicted area to the total area.
- G_i represents the i-th ground truth instance for the video.

We do this all possible proposals boundaries and fill our label map with maximum computed IoU to be used for training our proposal features.

5.2.4 Results

We train on the ActivityNet 1.3 dataset which is divided into three parts: the training set, the test set and the validation set. The dataset consists of more than 648 hours of untrimmed videos from a total of 20K videos and contains 200 different activity classes. [16] Using pre-computed visual input data, we have RBG and optical flow spatio-temporal features have 400 dimensions combined. These input features are trained with several architectures which differ in the sequential modelling approach and the hidden layer depth of the Incorporated model. We experimented with including more hidden convolutional layers at the expense of lowering the dimension of the layer. We present our best results using our model and with traditional architectures and compare our scores with the top models in the Activity 1.3 challenge.

Model	AUC (val)
CTAP[20]	64.40
Temporal convolution based action proposal[21]	65.72
BSN[10]	67.10
Incorporated Model with LSTM [1*400]-1024 (no bottlenecks)	67.13
Incorporated Model with FSMN [1*400]-1024-1024 (no bottlenecks, 7 Conv layers)	**67.29**
Incorporated Model with FSMN [1*400]-1024-1024	**66.90**

Table 5.1: Comparison of AUC scores between our methods and other state of the art models

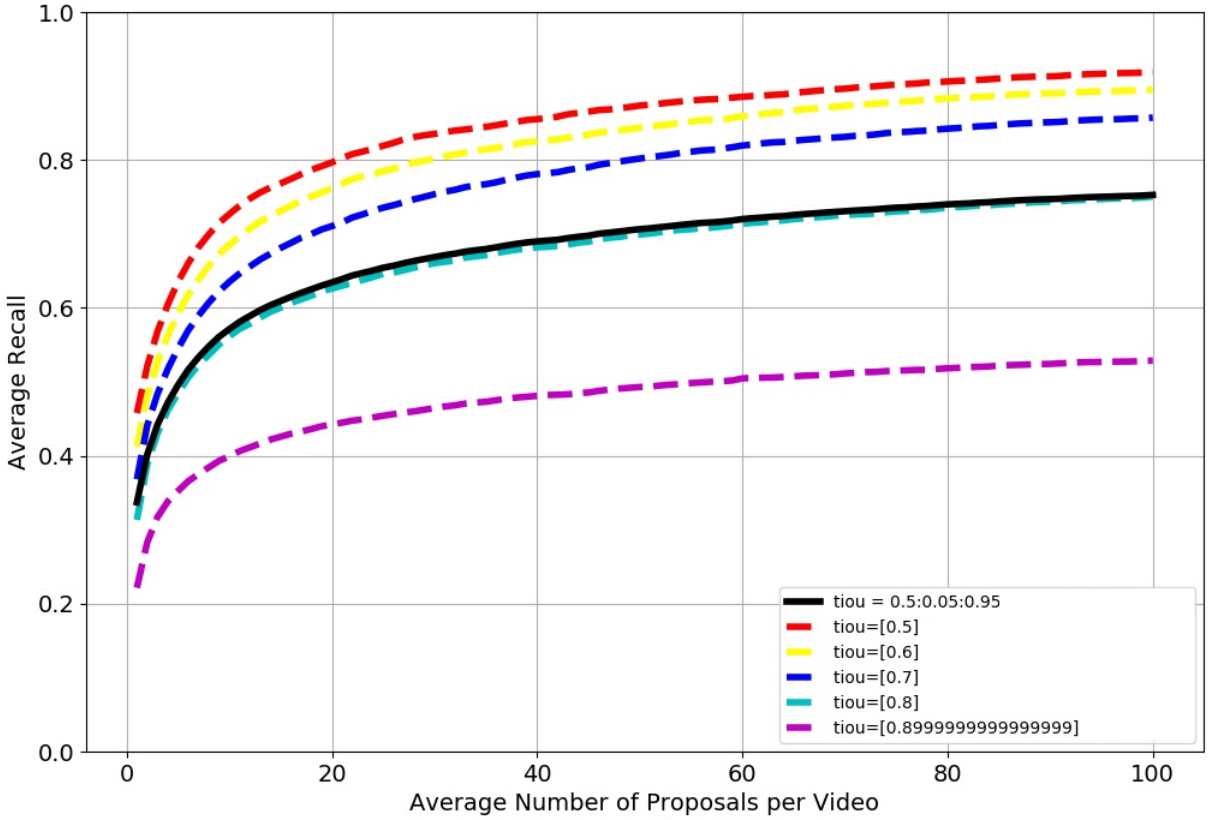

Figure 5.1: Area under the average recall vs average number of proposals per video curves.

5.2.5 Ablation Comparison

To further evaluate the effectiveness of our design, we conduct an ablation study comparing our approach with a more traditional method for generating temporal action proposals. We carry out this test with two models with similar architecture for obtaining temporal boundary probabilities and constructing proposals. We can then compare the consequence of training temporal boundary probabilities and proposal feature evaluation modules jointly and training them separately using this baseline architecture.

Model	AUC (val)
Jointly trained temporal boundary and proposal feature evaluation	66.9%
Separately trained temporal boundary and proposal feature evaluation	64.7%

Table 5.2: Comparison in terms of Area under the average recall vs average number of proposals per video.

We find that the jointly trained configuration results in significantly better evaluation of proposal features. This is manifested in the improvement in AUC of the validation set performance in our experiments. The results are shown in Table 5.2. This is evidence that jointly training the modules generates confidence scores that result in higher quality proposals.

5.2.6 Efficiency Analysis

We demonstrate in our ablation comparison that our jointly trained model achieves better performance. We also highlight that our model is the more efficient approach in terms of training and inference speed. The separately trained model requires temporal boundary probabilities to

be computed first. Then proposals can be constructed and evaluated. This approach requires no score fusion as proposals are constructed slowly using a bottom-up fashion.

Using the jointly trained approach, computing temporal boundary probabilities along with proposal construction are done simultaneously. Score fusion is used to obtain more reliable confidence scores. It is accomplished by a simple multiplication operation. Therefore it is much quicker than generating proposals one-by-one by searching for temporal locations with high boundary probabilities.

Model	Number of trainable parameters
Incorporated Model with LSTM [1*400]-1024	11.7M
Incorporated Model with FSMN [1*400]-1024-1024	4.6M

Table 5.3: Difference in the number of trainable parameters using LSTM and FSMN.

We also compare the number of trainable parameters used for generating proposal level features. As shown in Table 5.3, the FSMN based model uses fewer number of trainable parameters when compared with a LSTM model that is similar in performance. We find that the use of FSMN for modeling the visual features leads to reduced network end-to-end inference time. The end result of this approach is a model that can generate confidence scores for a large number of proposals in an efficient manner.

5.2.7 Proposal Evaluation Inference Speed

We exhaustively compare the efficiency of Incorporated Temporal Action Proposal Generator with other models by including measurements capturing the inference speed in evaluating temporal action proposals. We detail the time required for each model to construct and evaluate proposals for a fixed number of input videos. Since we have a large selection of videos with varying durations, we measure the inference speed for 100 batches. We compare our model to the state-of-the-art BSN model and report the proposal evaluation inference time in milliseconds. We run our experiments using a single Nvidia TITAN X and the results are presented in Table 5.4.

Model	Inference time (ms)
Boundary-Sensitive Network (BSN)	87.78
Incorporated Model	45.40

Table 5.4: Average Proposal Evaluation Inference time for 100 batches measured in milliseconds.

Based on these measurements, it is clear that the Incorporated Proposal Generator is able to evaluate confidence scores for a large selection of proposals rapidly. On the other hand, the measured inference time for the BSN model is noticeably longer to evaluate and generate scores for multiple proposals. This is further evidence that the evaluation process in BSN is tedious as confidence scores are generated by accessing each carefully crafted proposal individually. This confirms that the Incorporated Proposal Generator is much more adept at evaluating numerous proposals at once.

5.3 Action Classification

We also test the capability of Feedforward Sequential Memory neural Networks (FSMN) to capture sequential visual information by experimenting on the action classification task. UCF101 is a video dataset that contains finely trimmed human actions. The average length of a clip in the dataset is only 7 seconds. Since there are only 101 action categories which can be easily discriminated, action classification is more feasible here than in larger datasets such as ActivityNet. There are 101 action classes which are divided into five types: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, and Sports.[13] We train a two layer Feedforward Sequential Memory neural network (FSMN) with cross-entropy to learn to classify the actions. We sample 5 evenly distributed frames and perform random flipping and cropping. Then we use a ResNet101 model pretrained on ImageNet dataset to obtain visual features. We also obtain 5 optical flow frames sampled evenly using two concurrent frames to compute the horizontal and vertical flows. We compare FSMN based sequential modelling method with a similar model that uses LSTMs for sequential modelling instead. We set our initial learning parameter to 0.01 and commence training. We drop the learning by 0.1 every 10 epochs and run for total of 270K iterations. Since the ResNet101 model was pre-trainned for ImageNet classification task, we remove the last fully-connected layer and attach our FSMN layers at the end of the ResNet model. To achieve a competitive score to SOTA methods in UCF101, it requires training visual features from scratch using a deep convolutional network. We do not train the convolutional layers of the ResNet101 model. We report our results for the action classification task in Table 5.4.

Model	Top-1 Accuracy
ResNet10 with LSTM [1*400]-1024	82.1%
ResNet101 with FSMN [1*400]-1024-1024	84.3%

Table 5.5: Top1 accuracy for UCF101 Action Classification task.

Chapter 6

Conclusion

6.1 Conclusion

FSMNs have demonstrated success in a variety of different tasks such as speech recognition and language modeling. However, it has not been considered for sequential modeling of spatio-temporal features. In this thesis, we propose a novel unified approach by using Feed-forward Sequential Memory Networks (FSMNs) for the temporal action proposal task. This is accomplished through the Incorporated temporal action proposal generation model that jointly computes boundary probabilities for temporal segments while evaluating proposal confidence scores. The utilization of FSMNs for sequential modeling enables the quick and efficient processing of spatio-temporal features. To demonstrate the effectiveness of FSMNs, we also apply it to the UCF101 video action classification task. The results for this classification task demonstrate a 2% increase in performance over standard sequential modeling approaches such as Long-term

Short Memory networks (LSTMs). Likewise, our experiments on the temporal action proposal task verify that the FSMN based Incorporated model performs better than LSTM based methods.

In summary, we have proposed a unified approach to generate high quality temporal action proposals for untrimmed video data that is computationally efficient while at the same time capable of generating good performance. We have also shown that the learning of FSMNs is easier and faster than that of RNNs or LSTMs and is suitable for spatio-temporal related tasks. Our Incorporated model is designed to process large amounts of untrimmed video data and localize key human actions within high precision. The experiments on the ActivityNet 1.3 temporal action proposal task demonstrate the effectiveness of our approach in lowering computational complexity without sacrificing performance.

6.2 Future Works

As demonstrated by the video classification models for the UCF101 dataset, pre-training on the ActivityNet dataset is an option to obtain better spatio-temporal features. This however would be resource intensive as these video datasets are large and the process would require multiple GPUs training for an extended period of time. We also have to considered the parameter tuning process which would be another roadblock in this task.

Bibliography

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015.

[2] Alexander Mordvintsev. Optical flow — opencv-python tutorials 1 documentation. `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html`.

[3] Medhekar Medhekar. Accelerate opencv: Optical flow algorithms with nvidia turing gpus, Dec 2019. `https://devblogs.nvidia.com/opencv-optical-flow-algorithms-with-nvidia-turing-gpus/`.

[4] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 7445–7454. IEEE, Jul 2017. `http://ieeexplore.ieee.org/document/8100270/`.

[5] Christopher Olah. Understanding lstm networks – colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, Dec 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5.

[7] Prabhu. Understanding of convolutional neural network (cnn) — deep learning, Nov 2019. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning.

[8] Akinori Hidaka and Takio Kurita. Consecutive dimensionality reduction by canonical correlation analysis for visualization of convolutional neural networks. volume 2017, pages 160–167, 12 2017.

[9] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271 [cs]*, Apr 2018. arXiv: 1803.01271.

[10] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. BSN: boundary sensitive network for temporal action proposal generation. *CoRR*, abs/1806.02964, 2018.

[11] Shiliang Zhang, Hui Jiang, Si Wei, and LiRong Dai. Feedforward sequential memory neural networks without recurrent feedback. *CoRR*, abs/1510.02693, 2015.

[12] Shiliang Zhang, Cong Liu, Hui Jiang, Si Wei, Li-Rong Dai, and Yu Hu. Feedforward sequential memory networks: A new structure to learn long-term dependency. *CoRR*, abs/1512.08301, 2015.

[13] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[16] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[17] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. *arXiv:1608.08242 [cs]*, Aug 2016. `http://arxiv.org/abs/1608.08242`.

[18] Tianwei Lin, Xu Zhao, and Zheng Shou. Temporal convolution based action proposal: Submission to activitynet 2017. *arXiv:1707.06750 [cs]*, Sep 2018. `http://arxiv.org/abs/1707.06750`.

[19] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *CoRR*, abs/1312.6184, 2013.

[20] Jiyang Gao, Kan Chen, and Ram Nevatia. CTAP: complementary temporal action proposal generation. *CoRR*, abs/1807.04821, 2018.

[21] Tianwei Lin, Xu Zhao, and Zheng Shou. Temporal convolution based action proposal: Submission to activitynet 2017. *CoRR*, abs/1707.06750, 2017.

[22] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. BMN: boundary-matching network for temporal action proposal generation. *CoRR*, abs/1907.09702, 2019.

[23] Bernard Ghanem, Juan Carlos Niebles, Cees Snoek, Fabian Caba Heilbron, Humam Alwassel, Ranjay Krishna, Victor Escorcia, Kenji Hata, and Shyamal Buch. Activitynet challenge 2017 summary. *CoRR*, abs/1710.08011, 2017.

[24] Tianwei Lin, Xu Zhao, and Zheng Shou. Temporal convolution based action proposal: Submission to activitynet 2017. *CoRR*, abs/1707.06750, 2017.

[25] Yuanjun Xiong, Limin Wang, Zhe Wang, Bowen Zhang, Hang Song, Wei Li, Dahua Lin, Yu Qiao, Luc Van Gool, and Xiaoou Tang. CUHK & ETHZ & SIAT submission to activitynet challenge 2016. *CoRR*, abs/1608.00797, 2016.

[26] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *CoRR*, abs/1507.02159, 2015.

[27] H Wang, A Klaser, C Schmid, and C Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR)*, page 3169–3176, 2011.

[28] Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Fast temporal activity proposals for efficient detection of human actions in untrimmed videos. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1914–1923, 2016.

[29] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Daps: Deep action proposals for action understanding. In *ECCV*, page 1914–1923, 2016.

[30] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[31] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Dahua Lin, and Xiaoou Tang. Temporal action detection with structured segment networks. *CoRR*, abs/1704.06228, 2017.

[32] Zheng Shou, Dongang Wang, and Shih-Fu Chang. Action temporal localization in untrimmed videos via multi-stage cnns. *CoRR*, abs/1601.02129, 2016.

[33] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[34] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[35] Ze-Huan Yuan, Jonathan C. Stroud, Tong Lu, and Jia Deng. Temporal action localization by structured maximal sums. *CoRR*, abs/1704.04671, 2017.

Appendix

AUC - The evaluation metric used for the temporal action proposal task is the area under the Average Recall vs. Average Number of Proposals per Video (AR-AN). A proposal is a true positive if it has a temporal intersection over union with a ground-truth segment that is greater than that or equal to given threshold of 0.5. Average Recall is the mean of all recall values using temporal intersection over union value between 0.5 and 0.9 with a step size of 0.05. Average number of proposals is defined as the total number of proposals divided by the number of videos in the testing subset.