Multi-frame deep learning models for action detection in surveillance videos

T. A. Khan

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Multi-frame deep learning models for action detection in surveillance videos

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

T. A. Khan

November 17, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by TNO. Their cooperation is hereby gratefully acknowledged.

Abstract

Visual surveillance technologies are increasingly being used to monitor public spaces. These technologies process the recordings of surveillance cameras. Such recordings contain depictions of human actions such as "running", "waving", and "aggression". In the field of computer vision, automated detection of human actions in videos is known as action detection. Recently, deep learning models have been proposed for the task of action detection. Deep learning models for this task can be grouped into single-frame models and multi-frame models. Single-frame models detect actions using individual frames of videos whereas multi-frame models detect actions using sequences of frames.

This thesis proposes to use multi-frame models as compared to single-frame models for action detection in surveillance videos. To compare multi-frame and single-frame models, we implement the ACT-detector [1]. The ACT-detector is a deep learning model that takes as input a sequence of K frames and outputs tubelets (labeled sequences of bounding boxes). We train and evaluate ACT for various values of K on the VIRAT dataset [2]. In our comparison, $K = 1$ serves as the single-frame model and $K > 1$ as the multi-frame models. When compared qualitatively, we find that multi-frame models have less missed detections. When compared quantitatively, we find that multi-frame models outperform single-frame models in performance measures such as classification accuracy, MABO, frame-mAP, and video-mAP.

To assess whether the improvements of multi-frame models yield purely from the increased number of frames, or also from the temporal order encoded by those frames, we experiment with training multi-frame models on unordered sequences of frames, i.e., sequences for which the frames are shuffled in time. When compared qualitatively, we find that multi-frame models have less precise localization when trained on unordered sequences. When compared quantitatively, we find that multi-frame models perform worse when trained on unordered sequences, indicating that multi-frame models learn temporal dynamics of actions. Nevertheless, even when trained on unordered sequences, multi-frame models outperform single-frame models for action detection in surveillance videos.

Table of Contents

Preface

As a student of Systems and Control with an interest in deep learning for computer vision, I was excited to be given the opportunity to work on the topic of action detection. I worked on this topic during my one-year internship at TNO. During this internship, I researched deep learning models for action recognition and action detection. I was impressed by how these models are capable of recognizing what actions are taking place in videos. My curiosity led me to implement and experiment with these models. This graduation thesis documents the findings of these experiments.

I would like to express my sincere gratitude to dr.ir. Gertjan Burghouts and dr. Raimon Pruim. Their expert advice and continuous encouragement have been invaluable throughout all stages of this work. I would also like to thank the members of TNO's Intelligent Imaging group for making me feel welcome and for providing resources to help advance my research. I am also very thankful to my supervisor dr. ing. Raf Van de Plas. His guidance and critique helped me remain sharp throughout this work. Finally, I am thankful to my family for continuously supporting me throughout my studies.

Tiamur Khan
November 2019

Chapter 1

Introduction

1-1 Motivation

Artificial intelligence is increasingly being used to aid authorities in processing data from surveillance cameras. For a visual surveillance operator, whose job is to observe recordings of surveillance cameras for suspicious behavior, it can become tiring to monitor multiple persons on multiple cameras for long periods of time. Vandalism, theft, or aggression can occur in only seconds, and if this is missed, authorities and medical professionals may respond too late. In the field of computer vision, automated detection of human actions such as "running", "waving", and "aggression" in videos is known as action detection. An automated action detection system can aid the task of a visual surveillance operator by alerting when it detects suspicious actions. Such a system could help improve the safety and security of public spaces and may potentially save the lives of victims.

1-2 Video processing

Deep learning, a class of machine learning algorithms, has become the go-to approach for many computer vision applications. They have outperformed conventional computer vision approaches on tasks such as image recognition, object detection, and semantic segmentation. Recently, deep learning models have been proposed for video processing. Videos, unlike images, contain spatio-temporal data. The spatial data represents the appearance of the scene and the temporal data represents the motion of the scene. Motion data can be crucial for distinguishing human actions. For example, the actions "sitting down" and "standing up" are similar in appearance, but differ in motion. This idea is visualized in Figure 1-1. Tasks in video processing include action recognition and action detection. Modern action recognition and action detection frameworks use deep learning to process the spatio-temporal data in videos.

Figure 1-1: The actions "sitting down" and "standing up" are challenging to distinguish from a single frame. The action becomes more clear when looking at sequence of frames. Figure from [1].

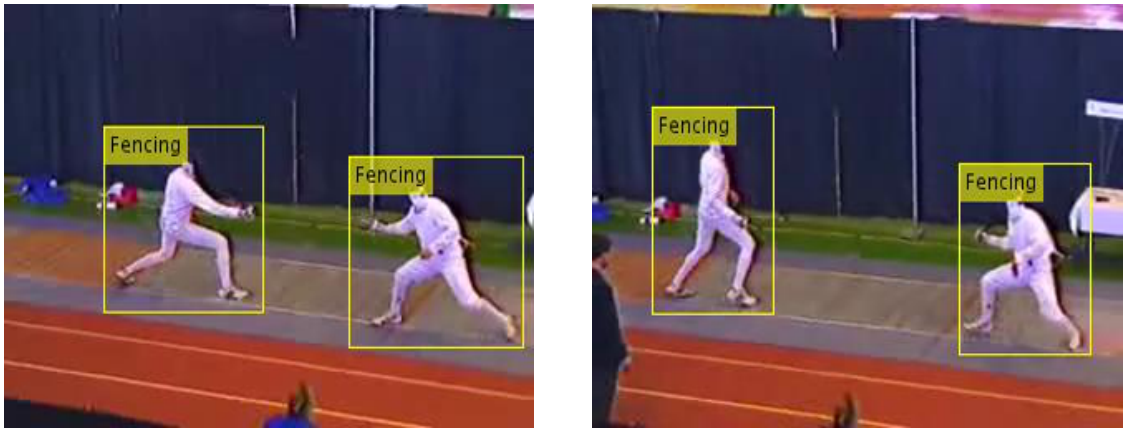
1-2-1 Action recognition

Action recognition is the activity of recognizing a human action from a video containing complete action execution [3]. Given such a video, the task of action recognition is to classify the video into a set of labels. For this reason, action recognition is also referred to as action classification. Action recognition answers the question: *what* is happening in the video? Videos used for this task typically contain a single actor performing a single action. Deep learning approaches for action recognition model the spatio-temporal data in videos by hybrid architectures [4], multi-stream architectures [5], or 3D convolution [6, 7]. Figure 1-2 shows sample frames from a dataset commonly used to benchmark approaches for action recognition.

Figure 1-2: A visualization of action recognition. Six sample frames from the UCF101 dataset [8] are shown. Each frame denotes a video containing the action described by the text.

1-2-2 Action detection

Action detection is to localize in space and time and classify all action instances in a video. Action instances are localized by sequences of bounding boxes through time. A labeled sequence of bounding boxes through time is commonly referred to as an *action tube* [1]. Action detection is also referred to as spatio-temporal action detection or spatio-temporal action localization. Action detection answers the question: *where*, *when*, and *what* is happening in the video? Since videos used for action detection are not necessarily trimmed to the action, action detection is considered more challenging than action recognition. Several deep learning approaches have been proposed for action detection. These can be divided into single-frame approaches [9, 10], and multi-frame approaches [1, 11]. Single-frame approaches detect actions using individual frames of videos whereas multi-frame approaches detect actions using sequences of frames. Figure 1-3 visualizes the task of action detection. Action detection is the primary focus of this thesis.



(a) Frame: 1 **(b)** Frame: 128

Figure 1-3: A visualization of action detection. Figure 1-3a: the first frame of a video containing action instances. Figure 1-3b: frame 128 of a video containing action instances. These are samples frames from the UCF101 dataset [8].

1-3 Research questions

This thesis proposes to use multi-frame models as compared to single-frame models for action detection in surveillance videos. Multi-frame models are typically better performing as they can infer motion patterns from sequences of frames. Recently, multi-frame approaches for action detection have been evaluated on short sports videos (2-20 seconds per video) [1]. We evaluate multi-frame models for the more challenging use case of visual surveillance. Surveillance videos are typically longer than sports videos and may contain multiple actions happening simultaneously. There may also be time spans in which no actions take place. The research question is formulated as:

- Do multi-frame models outperform single-frame models for action detection in surveillance videos?

The performance increase of multi-frame models may yield purely from the increased number of frames. To assess whether this is the case, we experiment with the temporal ordering of frames for training multi-frame models. We expect that the temporal order of frames contributes to the performance increase of multi-frame models as actions are characterized by their dynamics (e.g. "sitting down" and "standing up"). The corresponding research question is formulated as:

- Does the temporal ordering of frames matter for learning multi-frame models?

To answer these research questions, we implement the ACT-detector [1]. The ACT-detector is a deep learning model that takes as input a sequence of K frames and outputs tubelets (labeled sequences of bounding boxes). In our comparison, $K = 1$ serves as the single-frame model and $K > 1$ as the multi-frame models.

1-4 Thesis outline

This thesis is divided into 7 chapters. Chapter 2 gives an introduction to artificial and convolutional neural networks. Convolutional neural networks are used to model the spatio-temporal data in videos. The VGG [12] and SSD [13] architectures are also discussed. The VGG architecture is used by the SSD architecture and the SSD architecture forms the basis for the ACT-detector [1]. The ACT-detector and our implementation of the framework are discussed in Chapter 3. Chapter 4 discusses the dataset and the performance measures we use for evaluating the ACT-detector for action detection. Chapter 5 presents and discusses the experiments we conducted to answer the research questions. Chapter 6 analyzes the performance of the ACT-detector in more detail. Chapter 7 concludes the thesis and provides recommendations for future work.

Chapter 2

Background

Recent approaches for action recognition [5, 6, 7] and action detection [9, 10, 1] use convolutional neural networks to model the spatio-temporal data in videos. This chapter provides a background in artificial and convolutional neural networks, discusses the VGG16 architecture [12] for image recognition, and discusses the SSD architecture [13] for object detection. The SSD architecture is the basis for the ACT-detector [1] for action detection, which will be discussed in Chapter 3.

2-1 Artificial neural networks

An artificial neural network is a collection of computational nodes called neurons [14]. A neuron processes multiple inputs and produces a single output. The output is the weighted sum of its inputs, passed through an activation function. Mathematically, a neuron is defined as:

$$f(x, w) := \phi\left(w_0 + \sum_{i=1}^{N} w_i x_i\right), \tag{2-1}$$

where x are the inputs, w the weights, and ϕ the activation function. The variable w_0 denotes the bias term, which is added to the weighted sum of inputs.

Neurons can be arranged by means of layers (see Figure 2-1). There are three kinds of layers: the input layer, the output layer, and the hidden layer. The *input layer* takes the input data and passes them to the first hidden layer. The *output layer* contains neurons that process the outputs from the final hidden layer to produce an output. The *hidden layers* are layers that are neither the input layer nor the output layer [14]. A neural network has one input layer, one output layer, and zero or more hidden layers. Neural networks with two or more hidden layers are called *deep neural networks* [14]. The number of neurons in the input layer is equal to the number of data inputs, the number of neurons in the hidden layers is a design choice, and the number of neurons in the output layer is equal to the number of desired

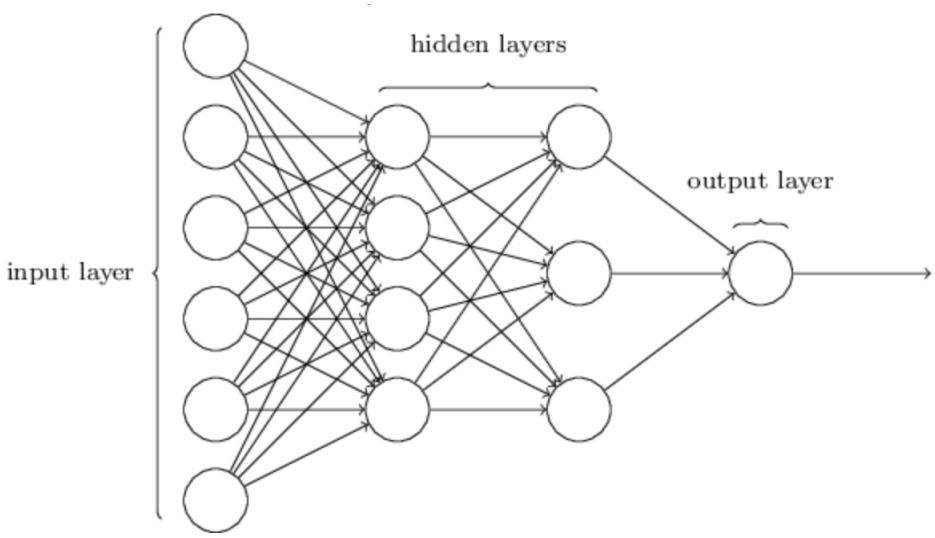


Figure 2-1: An example deep neural network. The neural network contains one input layer, one output layer, and two hidden layers. Figure adapted from [14].

outputs. To give an example, an application of neural networks is the recognition of digits in 2-dimensional images. For a 2-dimensional image of size 28×28, the input layer contains $28 \times 28 = 784$ pixel values. The number of hidden layers and the number of neurons in those layers is a design choice and is determined by cross-validation. The number of neurons in the output layer is 10, as we wish to recognize the digits 0 through 9.

Several types of neural networks have been proposed for various applications. These neural networks differ in the way their neurons are connected between layers. The neural network shown in Figure 2-1 is a *dense neural network*. The neurons in dense neural networks are connected to all neurons in the layer before it. Another type of neural network is the *convolutional neural network*. These are commonly used for image and video processing tasks. Section 2-2 discusses the convolutional neural network in more detail. Neural networks that contain feedback loops are called *recurrent neural networks*. Recurrent neural networks are commonly used for processing sequential data such as text or time-series. As recurrent neural networks contain feedback loops, they can be used for modeling dynamical systems. Some approaches for video processing also make use of recurrent neural networks [4, 15]. Neural networks that contain no recurrent connections are also referred to as *feed-forward neural networks*. Examples of feed-forward neural networks are dense neural networks and convolutional neural networks.

2-1-1 Activation functions

As mentioned in the previous section, the output of a neuron is the weighted sum of its inputs, passed through an activation function ϕ. It has been shown that if ϕ is a non-linear function, a neural network with 1 hidden layer can approximate any continuous function [16]. For this reason, non-linearity is a desirable property of activation functions. Examples of non-linear activation functions are: the sigmoid function [17], the hyperbolic tangent (tanh) [18], and the Rectified Linear Unit (ReLU) [19]. These are visualized in Figure 2-2.

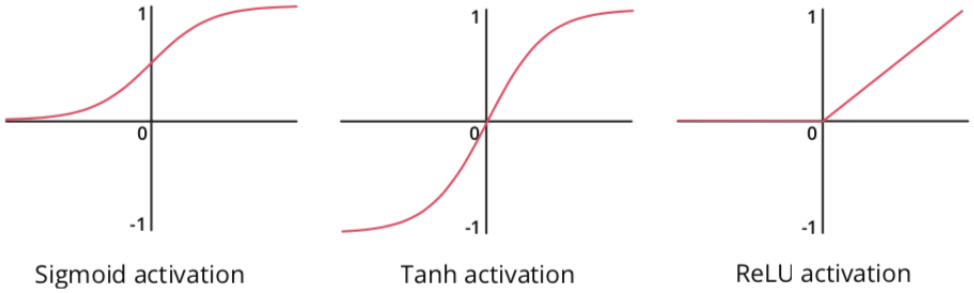


Figure 2-2: Commonly used activation functions for neural network design. Figure adapted from [20].

The sigmoid activation function is defined as:

$$\phi_{sigmoid}(x) = \frac{1}{1+\exp(-x)}. \tag{2-2}$$

The sigmoid activation function maps an input x to a value in the range $(0, 1)$. This is useful for neural networks that predict probabilities, as these are also in the range $(0, 1)$. The hyperbolic tangent is defined as:

$$\phi_{tanh}(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}. \tag{2-3}$$

The range of the hyperbolic tangent is $(-1, 1)$. This is useful for cases in which both a positive and negative output is desired. The hyperbolic tangent has a similar shape to the sigmoid function. The hyperbolic tangent and the sigmoid function are related as follows:

$$\phi_{tanh}(x) = 2\phi_{sigmoid}(2x) - 1. \tag{2-4}$$

The sigmoid and the hyperbolic tangent functions have been the historical tools of choice for adding non-linearity in neural networks [21]. In recent years, piece-wise linear activation functions have been proposed. One of such functions is the Rectified Linear Unit (ReLU):

$$\phi_{ReLU}(x) = \max\{0, x\}. \tag{2-5}$$

The ReLU activation function is 0 when the input x is negative and x when x is positive. Therefore, the gradient of the ReLU activation function is either 0 or 1. A gradient of 0 results in no change for the weights during training. This problem is known as the vanishing gradient problem. Some solutions have been proposed such as the Leaky ReLU activation function [22].

2-1-2 Training

The goal of a neural network is to approximate a function f by tuning its weights w. The process of tuning the weights to produce a desired output is called training. During training, the neural network is presented with an input X and the corresponding label y. In the field of deep learning, the label y is commonly referred to as the ground-truth [13, 1]. Based on the input X, the neural network predicts an output $\hat{y}$. The predicted output is compared to the ground-truth to measure how accurate the prediction of the neural network is. This comparison is done by means of a loss function (also referred to as error function). Several loss functions have been proposed for various tasks. An example loss function is the squared L2-norm:

$$L(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{2-6}$$

where L is the loss, y the ground-truth, $\hat{y}$ the predicted output by the neural network, and N the number of outputs. The squared L2-norm is small when the error $|y - \hat{y}|$ is small, and large when the error is large. The goal then becomes to minimize the loss function by tuning the weights w. Minimizing the loss function is typically done by computing the gradient of the loss function ∇L with respect to all weights.

The gradients in a neural network can be computed efficiently by the backpropagation algorithm [23]. The backpropagation algorithm computes the gradients by using the chain rule. The loss at the output layer is propagated backwards so that the gradients at the hidden layer can be updated. The gradients at each layer are then used to update the weights at each neuron:

$$w \rightarrow w' = w - \alpha \nabla L, \tag{2-7}$$

where α is the learning rate. The learning rate is determined empirically and is typically in the order of 10^{-3}. Equation (2-7) updates the weights in the steepest descent direction. This optimization strategy is known as Stochastic Gradient Descent (SGD) [24]. Other optimization strategies have been proposed to increase the convergence rates of neural networks. An example of such a strategy is Adam [25]. To summarize, the training procedure of a neural network is:

1. An example input is given to the neural network. The input is forwarded to the hidden layers, which forward their outputs to the output layer. The neural network produces an output.
2. The loss at the output is calculated based on the ground-truth and the predicted output.
3. The gradients with respect to all weights are computed using the backpropagation algorithm.
4. The weights are updated using the gradients.

By iterating this process for multiple training examples, a local minimum can be approximated for the loss function. As neural networks contain large numbers of weights, the training set should be sufficiently large for convergence.

2-2 Convolutional neural networks

Convolutional neural networks are artificial neural networks that use the convolution operator in at least one of its layers [21]. They contain three main building blocks: convolutional layers, max-pooling layers, and fully connected layers. Fully connected layers are layers in which each neuron is connected to all neurons in the layer before it. The following sections describe the convolutional and max-pooling layers.

2-2-1 Convolutional layers

Convolutional neural networks are designed to process grid-structured inputs. An image, which can be viewed as a 2D grid, contains spatial dependencies since local regions in images are similar in pixel values. Convolution can exploit this property by producing similar values for local regions with similar patterns. The *convolution* operator is a dot-product operation between a grid-structured set of weights and similar grid-structured inputs drawn from different spatial locations in the input volume [21] (see Figure 2-3). Convolution can also be defined over multiple axes. For the case of image processing, it is common to use 2D convolution. 2D convolution for an image I with kernel $K \in \mathbb{R}^{m \times n}$ can be expressed as:

$$F(x, y) = \sum_m \sum_n I(x - m, y - n) K(m, n), \tag{2-8}$$

where $F(x, y)$ denotes the output of sliding the kernel over all pixels of the image. $F(x, y)$ is often called the feature map. The convolution operation has three parameters: kernel matrix dimension, stride, and padding. The kernel matrix dimensions are the height and width of the kernel, stride is the number of pixels the kernel shifts in the input, and padding controls how the border of the input is padded. Since convolution decreases the dimension of the input (see Figure 2-3), padding can be used to preserve the input dimensions.

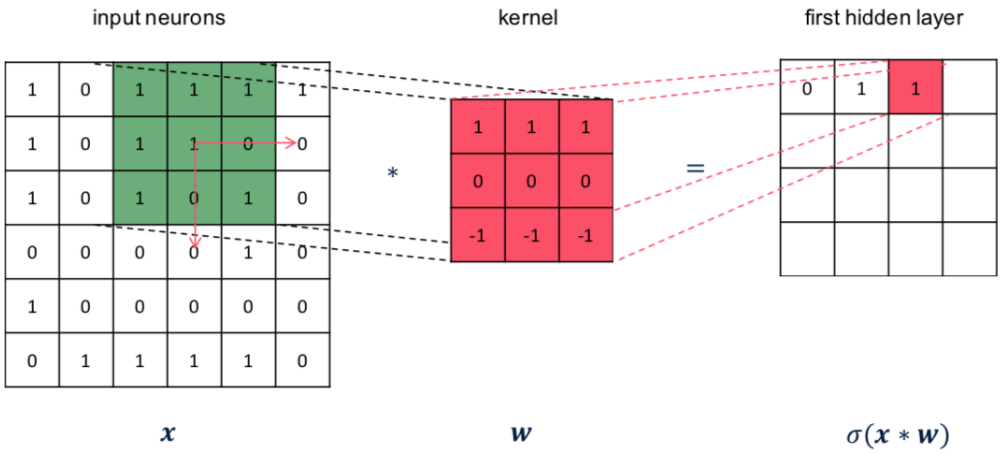


Figure 2-3: The kernel $w \in \mathbb{R}^{3 \times 3}$ is used for convolution of the input volume x to produce the output feature map in the first hidden layer. The activation function σ is applied after convolution. Figure adapted from [20].

Compared to fully connected layers, neurons in a convolutional layer are sparsely connected to the neurons in the layer before it (see Figure 2-4). This is because each neuron in a convolutional neural network receives input from a small region of the image. An additional observation is that parameters in convolutional layers are re-used as the same kernel slides over all pixel values of an image. This is called parameter sharing. Due to sparse connectivity and parameter sharing, convolutional layers are much more efficient in terms of computational complexity and memory requirements than densely connected layers.

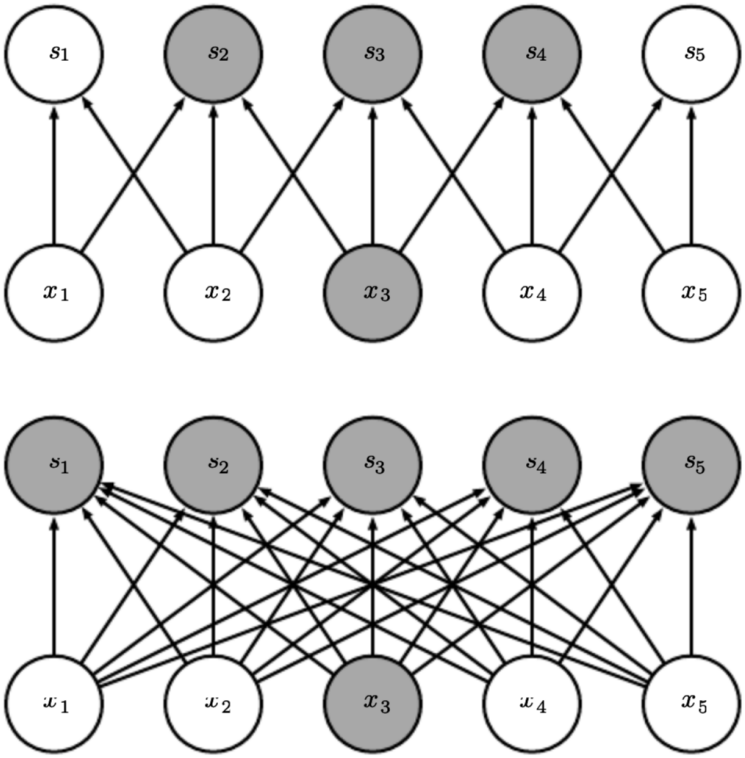


Figure 2-4: Top: sparse connectivity of a convolutional neural network. Each neuron in a convolutional layer is sparsely connected to the neurons in the layer before it. Bottom: fully connected layers. Each neuron is a densely connected layer is connected to all neurons in the layer before it. Figure adapted from [17].

2-2-2 Max-pooling layers

Convolutional neural networks also have max-pooling layers. These layers reduce the spatial dimensions of feature maps by an operation called *max-pooling*. Max-pooling divides the input volume into a set of equal regions, and for each region, it takes the maximum value (see Figure 2-5). Max-pooling reduces the computational cost of a convolutional neural network and can also reduce the number of parameters if placed before a fully connected layer. Max-pooling has two parameters: dimension and stride.

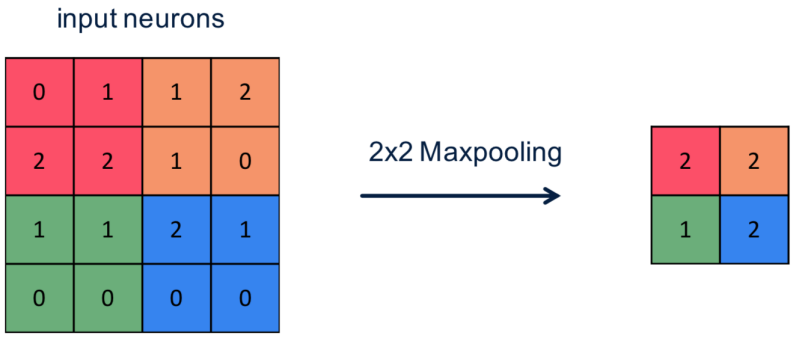


Figure 2-5: Max-pooling: taking the maximum value of regions in feature maps. Dimensions = 2×2 and stride = 2. Figure adapted from [20].

2-3 VGG

We now discuss the VGG [12] convolutional architecture for image recognition. Image recognition is the task of classifying images into a set of labels. Several convolutional architectures have been proposed for this task, but the design principles of VGG are worth mentioning as they played a major role in the design of more recent convolutional architectures [26]. The VGG architecture was developed for the ImageNet Large Scale Visual Recognition Challenge 2014 (ISLVRC 2014) [27]. For this challenge, VGG achieved a top-5 error rate of 6.7%.

2-3-1 Design principles

VGG uses small kernel dimensions of 3×3 to process the input image. As VGG uses small kernel dimensions, the depth (the number of layers) of the network is increased. This is because small kernels capture only a small region of the input. However, successive convolution with small kernel dimensions captures the same region as a single convolution with large kernel dimensions. For example, 3 successive convolutions of a kernel of size 3×3 capture a region in the input of size 7×7. The main advantage of using smaller kernels is that successive convolutions can capture more complex and detailed features than a single convolution. They also require less parameters ($3 \times 3 \times 3 = 27$ for small kernels versus $7 \times 7 = 49$ for large kernels). By increasing the depth, the model also becomes more non-linear. This is because more convolutions are computed, which are followed by non-linear activations. An increase in non-linear activations results in an increase in the discriminative power of the network [21]. For these reasons, modern convolutional architectures increase the depth, while decreasing the kernel dimensions.

2-3-2 Architecture

The input to VGG is an image tensor of size $(224, 224, 3)$. These values denote the height, width, and number of channels for the image, respectively. VGG uses kernel dimensions of 3×3 and max-pooling dimensions of 2×2 to process the input image. The convolutions are done with stride 1 and max-pooling is done with stride 2. A zero-padding of 1 is used for convolution to preserve the input dimensions. Max-pooling with a size of 2×2 and stride 2

decreases the spatial dimensions of the feature maps by a factor of 2 (see Figure 2-5). The number of kernels is increased by a factor of 2 whenever max-pooling takes place. This is to balance the computational effort across layers and to capture more complex features in the top layers [21]. VGG designed multiple architectures using these design principles. The best performing version for the ISLVRC 2014 was the VGG16 architecture. This version of VGG contains 16 weighted layers, hence the name VGG16. VGG16 contains 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. All convolutions are followed by ReLU activations. The full design of VGG16 is shown in Table 2-1. In total, VGG16 contains about 138 million parameters.

Table 2-1: The VGG16 architecture. The term C3D64 refers to 64 kernels of dimensions 3×3, the padding is chosen to preserve the input dimensions. Each convolution is followed by a ReLU activation. The max-pooling layer is denoted by M and FC4096 refers to a fully connected layer with 4096 neurons.

name	config
`conv1_1`	C3D64
`conv1_2`	C3D64
`pool1`	M
`conv2_1`	C3D128
`conv2_2`	C3D128
`pool2`	M
`conv3_1`	C3D256
`conv3_2`	C3D256
`conv3_3`	C3D256
`pool3`	M
`conv4_1`	C3D512
`conv4_2`	C3D512
`conv4_3`	C3D512
`pool4`	M
`conv5_1`	C3D512
`conv5_2`	C3D512
`conv5_3`	C3D512
`pool5`	M
`fc6`	FC4096
`fc7`	FC4096
`fc8`	FC1000

2-4 SSD

Having discussed the VGG16 architecture, we now discuss the SSD (Single Shot MultiBox Detector) [13] architecture. The SSD architecture forms the basis of the ACT-detector [1], which will be discussed in Chapter 3. The SSD architecture is a fully convolutional neural network designed for the task of object detection. In computer vision, object detection is to localize and classify multiple objects in images. The term fully convolutional neural network refers to a convolutional neural network that contains no fully connected (dense) layers.

The SSD architecture is a one-stage detector; the detector localizes and classifies the objects in an image in a single evaluation of the network. This is unlike two-stage detectors [28, 29], which propose regions and then classify these regions. One-stage detectors learn localization and classification of objects jointly and have less computational costs than two-stage detectors.

2-4-1 Architecture

The SSD architecture is visualized in Figure 2-6. The input to SSD is an image tensor of size $(H \times W \times 3)$, where H and W are the height and width of the image, respectively. The 3 refers to the number of channels, which is 3 for an RGB-image. The output is a tensor of size $(B \times (c + 1 + 4))$, where B is the number of *anchor boxes* and c the number of classes. The 1 refers to the background class and the 4 to the number of coordinates in a *bounding box*. The terms anchor box and bounding box will be discussed in the next section.

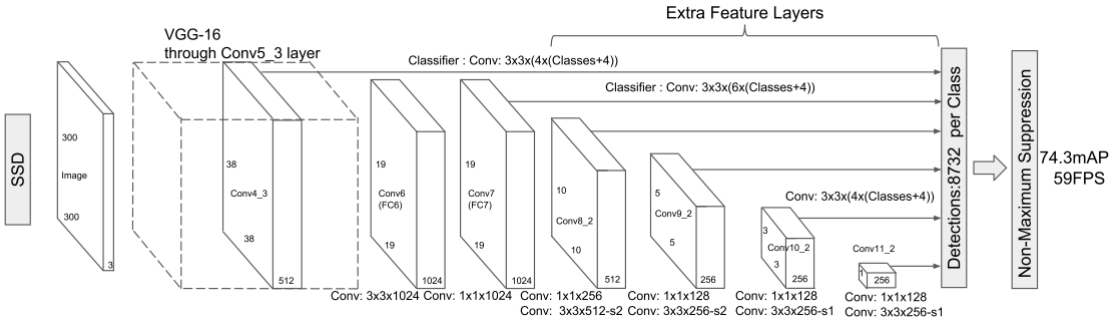


Figure 2-6: The SSD architecture. The SSD architecture contains three main components: a modified version of VGG16 [12], auxiliary convolutional layers to provide additional feature maps, and prediction layers to produce the output. We discuss these components in the following paragraphs. Figure adapted from [13].

VGG16 base

A modified version of the VGG16 architecture is used by SSD to encode the input image into lower level representations. The SSD architecture modifies the VGG16 architecture by removing `fc8`, and sub-sampling and converting `fc6` and `fc7` into convolutional layers. The resulting convolutional layers are denoted by `conv6` and `conv7` in Figure 2-6.

Auxiliary convolutional layers

On top of the modified VGG16 architecture, 4 blocks of convolutional layers are added. Each block contains two convolutional layers. These convolutions provide additional feature maps. The second convolutional layer of each block performs convolution with a stride of 2. This decreases the spatial dimensions of the feature maps. The auxiliary convolutional layers are denoted by `conv8_1`, `conv8_2`, `conv9_1`, `conv9_2`, `conv10_1`, `conv10_2`, `conv11_1`, and `conv11_2` in Figure 2-6.

2-4-2 Anchor boxes

Before we discuss how the auxiliary convolutional layers are used to compute the output, we explain the terms *bounding box* and *anchor box*.

For object detection, we use bounding boxes to describe the location of an object in an image. We can represent bounding boxes by their pixel coordinates. The pixel coordinate system the SSD architecture uses is the center-size coordinate system. The center-size coordinates of a bounding box are: (c_x, c_y, w, h), where c_x is the x-coordinate of the center of the box, c_y the y-coordinate of the center of the box, w the width of the box, and h the height of the box. An example of center-size coordinates is given in Figure 2-7.

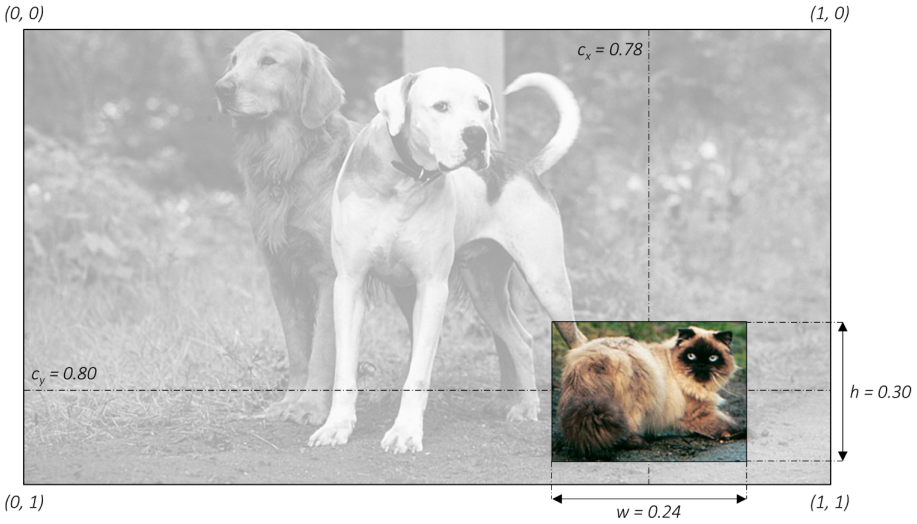


Figure 2-7: A visualization of the center-size coordinate system. Figure adapted from [30].

We are now ready to discuss anchor boxes. Anchor boxes are prior bounding boxes encoded in the SSD architecture. They serve as candidate regions for where an object may be. The original implementation of SSD contains 8732 anchor boxes. They are manually chosen and cover the full image. Anchor boxes are applied to low-level and high-level feature maps. The feature maps used for this are from the layers `conv4_3`, `conv7`, `conv8_2`, `conv9_2`, `conv10_2`, and `conv11_2`. This is also indicated in Figure 2-6. The anchor boxes vary in scale and aspect ratio.

The scale of the anchor boxes depends on the feature map. Larger feature maps (e.g. from `conv4_3`) have smaller scales than smaller feature maps (e.g. from `conv11_2`). The larger feature maps are used to detect smaller objects and the smaller feature maps are used to detect larger objects. If m feature maps are used for prediction, the scale of the anchor boxes for each feature map is computed as:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), \quad k \in [1, m], \tag{2-9}$$

where s_{min} is the minimum scale and s_{max} is the maximum scale. In the original implementation of the SSD $m = 6$, $s_{min} = 0.1$, and $s_{max} = 0.9$.

For each element (also referred to as pixel) in these feature maps, a set of anchor boxes is associated. The number of anchor boxes per element is 4 for feature maps from `conv4_3`, `conv10_2`, and `conv11_2`, and is 6 for feature maps from `conv7`, `conv8_2`, and `conv9_2`. Figure 2-8 shows the anchor boxes associated with the center pixel of a feature map from layer `conv9_2`. As can be seen from Figure 2-8, these anchor boxes vary in aspect ratio. The SSD architecture uses aspect ratios of $a_r \in \{1, 2, \frac{1}{2}\}$ for feature maps from `conv4_3`, `conv10_2`, and `conv11_2`, and aspect ratios of $a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$ for feature maps from `conv7`, `conv8_2`, and `conv9_2`. Each pixel has an additional enlarged anchor box of aspect ratio 1. Given the scales and aspect ratios, the width and height of the anchor boxes can be computed. The width is computed as $w_k^a = s_k\sqrt{a_r}$ and the height is computed as $h_k^a = \frac{s_k}{\sqrt{a_r}}$. Table 2-2 summarizes the design of the anchor boxes used by the SSD architecture.

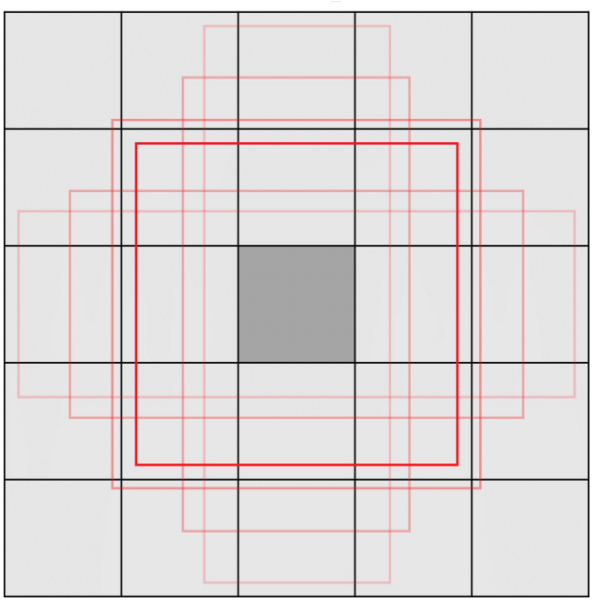


Figure 2-8: The anchor boxes associated with the center pixel of a feature map from layer `conv9_2`. There are 6 anchor boxes in total. Figure adapted from [30].

Table 2-2: Anchor boxes design used by the SSD architecture. The input is assumed to be an image tensor of size $(300, 300, 3)$.

Feature map from	Feature map dimensions	scale	number of anchor boxes per element	aspect ratios
`conv4_3`	(38, 38, 512)	0.1	4	$a_r \in \{1, 2, \frac{1}{2}\}$
`conv7`	(19, 19, 1024)	0.2	6	$a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$
`conv8_2`	(10, 10, 512)	0.375	6	$a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$
`conv9_2`	(5, 5, 256)	0.55	6	$a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$
`conv10_2`	(3, 3, 256)	0.725	4	$a_r \in \{1, 2, \frac{1}{2}\}$
`conv11_2`	(1, 1, 256)	0.9	4	$a_r \in \{1, 2, \frac{1}{2}\}$

2-4-3 Prediction layers

The output of the SSD architecture is the classes and bounding box coordinates of all anchor boxes encoded in the architecture. The prediction layers produce this output by convolving the feature maps shown in Table 2-2. The prediction layers compute $k(c+1+4)$ outputs for each element in a feature map. Here k is the number of anchor boxes per element shown in Table 2-2, $c+1$ is the number of classes (including background), and 4 is the number of bounding box coordinates. For this, $k(c+1+4)$ convolutional kernels of size 3×3 are used per prediction layer. Table 2-3 shows the dimensions of the feature maps involved in the prediction layers. The original implementation of SSD computes the class confidence scores and bounding box coordinates of $38 \cdot 38 \cdot 4 + 19 \cdot 19 \cdot 6 + 10 \cdot 10 \cdot 6 + 5 \cdot 5 \cdot 6 + 3 \cdot 3 \cdot 4 + 1 \cdot 1 \cdot 4 = 8732$ anchor boxes. The output is a tensor of size $(8732, c+1+4)$.

Table 2-3: The input dimensions and the output dimensions of the feature maps involved in the prediction layers. The prediction layers compute for all anchor boxes their class confidence scores and their bounding box coordinates.

layer name	Feature map input dimensions	Feature map output dimensions
`pred_conv4_3`	(38, 38, 512)	(38, 38, $4(c+1+4)$)
`pred_conv7`	(19, 19, 1024)	(19, 19, $6(c+1+4)$)
`pred_conv8_2`	(10, 10, 512)	(10, 10, $6(c+1+4)$)
`pred_conv9_2`	(5, 5, 256)	(5, 5, $6(c+1+4)$)
`pred_conv10_2`	(3, 3, 256)	(3, 3, $4(c+1+4)$)
`pred_conv11_2`	(1, 1, 256)	(1, 1, $4(c+1+4)$)

Chapter 3

Methodology

We implement the ACT-detector (ACtion Tubelet Detector) [1] to compare single-frame and multi-frame models for action detection in surveillance videos. The ACT-detector is a fully convolutional, one-stage detector that takes as input a sequence of frames and outputs tubelets (labeled sequences of bounding boxes). The following sections discuss the components of the ACT-detector, as well as the implementation details.

3-1 Architecture

The ACT architecture extends the SSD architecture along the temporal dimension. The input to the ACT-detector is a sequence of frames. The input tensor is of size $(K, H, W, 3)$, where K denotes the number of frames, H the height of the frames, W the width of the frames, and 3 refers to the number of channels, which is 3 for RGB-images. The height H and width W is equal for all K frames. The original implementation of the ACT-detector is designed for sequences with $H = 300$ and $W = 300$. For our implementation, we chose $H = 360$ and $W = 640$. As will be discussed in Section 4-1, the surveillance videos we use for performance evaluation are recorded at 1280×720 and 1920×1080 pixel resolutions. We downscale this to 640×360 to reduce the computational costs but preserve the aspect ratio of the recordings. An overview of the ACT-detector is shown in Figure 3-1.

3-1-1 VGG16 and auxiliary convolutional layers

The VGG16 convolutional base and auxiliary convolutional layers of the SSD architecture are used by the ACT-detector to process individual frames. For each frame in the sequence, the frame is reduced to feature maps of various dimensions by the VGG16 convolutional base and auxiliary convolutional layers. Since we have a larger input dimension than the original implementation of ACT (640×340 versus 300×300), we add an additional convolutional block with convolutional layers `conv12_1` and `conv12_2` to further reduce the feature map dimensions. The weights of the VGG16 convolutional base and auxiliary convolutional layers are shared among all frames in the sequence.

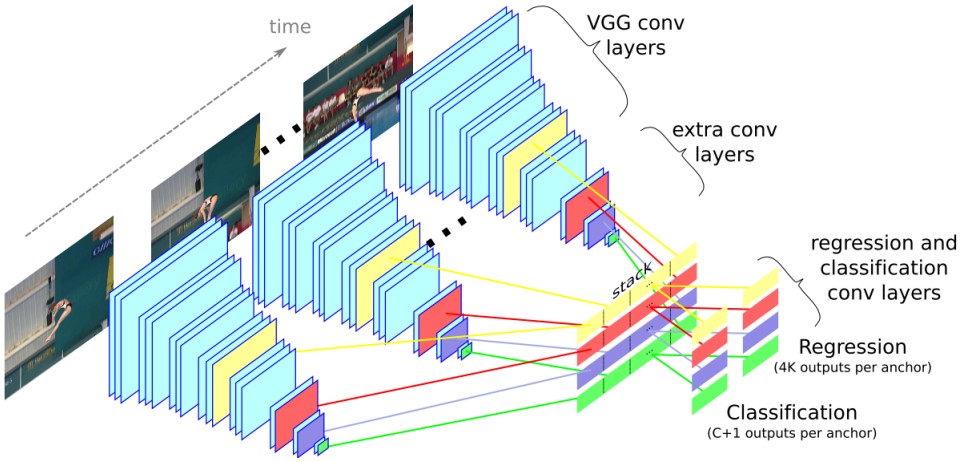


Figure 3-1: An overview of the ACT-detector [1]. The ACT-detector contains four main components: a modified version of VGG16 [12], auxiliary convolutional layers, temporal stacking, and prediction layers. We discuss these components in the following paragraphs.

3-1-2 Temporal stacking

Once we have computed the feature maps for all frames, we stack the feature maps from the following layers: `conv4_3`, `conv7`, `conv8_2`, `conv9_2`, `conv10_2`, `conv11_2`, and `conv12_2`. These are also the layers used by the SSD architecture to classify and regress anchor boxes. Table 3-1 shows the individual and temporally stacked dimensions of the feature maps from the previously mentioned layers for $K = 6$. As can be seen from Table 3-1, the feature maps are stacked in the channel dimension. These stacks of feature maps represent the spatio-temporal volume of the input sequence at reduced spatial dimensions.

Table 3-1: Individual and temporally stacked dimensions of feature maps for $K = 6$. The third dimension denotes the number of feature maps.

Feature map from	Feature map dimensions	Stacked feature map dimensions
`conv4_3`	(45, 80, 512)	(45, 80, 3072)
`conv7`	(23, 40, 1024)	(23, 40, 6144)
`conv8_2`	(12, 20, 512)	(12, 20, 3072)
`conv9_2`	(6, 10, 256)	(6, 10, 1536)
`conv10_2`	(3, 5, 256)	(3, 5, 1536)
`conv11_2`	(2, 3, 256)	(2, 3, 1536)
`conv12_2`	(1, 2, 256)	(1, 2, 1536)

3-2 Anchor cuboids

Before we discuss how the stacked feature maps are used to compute the output, we explain the term *anchor cuboid*.

The ACT-detector uses anchor cuboids as priors for action detection. Anchor cuboids are temporal extensions of anchor boxes. They are sequences of bounding boxes in time which serve as candidate volumes in which actions may occur. Their temporal length is equal to the number of frames in a sequence (K). Their spatial extent is fixed over time. The anchor cuboids are applied on the stacked feature maps shown in Table 3-1. Similar to SSD, they vary in size and aspect ratio. The original implementation of ACT contains 8732 anchor cuboids. Our implementation contains 21842 anchor cuboids, since we have larger input dimensions than the original implementation of ACT. Figure 3-2 visualizes anchor cuboids.

Similar to SSD, the larger stacked feature maps are used to detect smaller actions and the smaller stacked feature maps are used to detect larger actions. We compute the scales using Equation (2-9), with $m = 7$, $s_{min} = 0.05$, and $s_{max} = 0.5$. We chose the values of s_{min} and s_{max} small as the action in the dataset are small in pixel resolution. The aspect ratios of $a_r \in \{1, 2, \frac{1}{2}\}$ are used for feature maps from `conv4_3`, `conv11_2`, and `conv12_2` and aspect ratios of $a_r \in \{1, 1\frac{1}{2}, 2, \frac{2}{3}, \frac{1}{2}\}$ for feature maps from `conv7`, `conv8_2`, `conv9_2`, and `conv10_2`. We chose these aspect ratios more square than the original implementation of ACT as the actions in our dataset are more square than wide or tall. A summary of the design of our anchor cuboids is shown in Table 3-2.

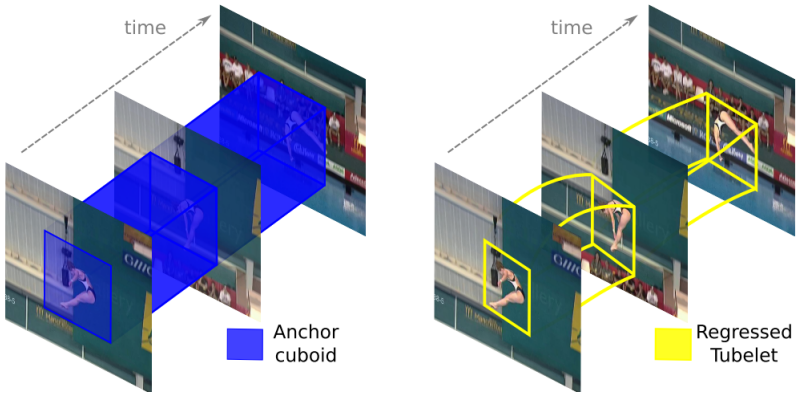


Figure 3-2: Anchor cuboids (left) regress to tubelets (right) to follow the action in time. Figure adapted from [1].

Table 3-2: Anchor cuboid design for our implementation of ACT $K = 6$. The input is assumed to be an image sequence tensor of size $(6, 360, 640, 3)$.

Feature map from	Stacked feature map dimensions	scale	number of anchor cuboids per element	aspect ratios
`conv4_3`	(45, 80, 3072)	0.05	4	$a_r \in \{1, 2, \frac{1}{2}\}$
`conv7`	(23, 40, 6144)	0.125	6	$a_r \in \{1, 1\frac{1}{2}, 2, \frac{2}{3}, \frac{1}{2}\}$
`conv8_2`	(12, 20, 3072)	0.2	6	$a_r \in \{1, 1\frac{1}{2}, 2, \frac{2}{3}, \frac{1}{2}\}$
`conv9_2`	(6, 10, 1536)	0.275	6	$a_r \in \{1, 1\frac{1}{2}, 2, \frac{2}{3}, \frac{1}{2}\}$
`conv10_2`	(3, 5, 1536)	0.35	6	$a_r \in \{1, 1\frac{1}{2}, 2, \frac{2}{3}, \frac{1}{2}\}$
`conv11_2`	(2, 3, 1536)	0.425	4	$a_r \in \{1, 2, \frac{1}{2}\}$
`conv12_2`	(1, 2, 1536)	0.5	4	$a_r \in \{1, 2, \frac{1}{2}\}$

3-3 Prediction layers

The output of the ACT detector is the classes and sequences of bounding box coordinates for all anchor cuboids encoded in the architecture. Similar to SSD, the prediction layers produce this output. The main differences are that the inputs to the prediction layers are stacked feature maps and that they predict the bounding box coordinates for each frame in the sequence. The prediction layers compute $k(c+1+4K)$ outputs for each element in a stacked feature map. Here k is the number of anchor cuboids per element shown in Table 3-2, $c+1$ is the number of classes (including background), and K is the number of frames in a sequence. For this, $k(c+1+4K)$ convolutional kernels of size 3×3 are used per prediction layer. Table 3-3 shows the dimensions of the feature maps involved in the prediction layers. We classify and regress $45 \cdot 80 \cdot 4 + 23 \cdot 40 \cdot 6 + 12 \cdot 20 \cdot 6 + 6 \cdot 10 \cdot 6 + 3 \cdot 5 \cdot 6 + 2 \cdot 3 \cdot 4 + 1 \cdot 2 \cdot 4 = 21842$ anchor cuboids into *tubelets*. Tubelets are labeled sequences of bounding boxes through time. The size and aspect ratio of the bounding boxes in a tubelet can change over time. The idea is that these boxes follow the action in time (see Figure 3-2). The output of our implementation of the ACT-detector is a tensor of size $(21842, c+1+4K)$.

Table 3-3: The input dimensions and the output dimensions of the feature maps computed by the prediction layers of ACT $K = 6$. The prediction layers compute for all anchor cuboids their class confidence scores and their sequence of bounding box coordinates. The third dimension is the number of feature maps.

layer name	Feature map input dimensions	Feature map output dimensions
`pred_conv4_3`	(45, 80, 3072)	(45, 80, $4(c+1+24)$)
`pred_conv7`	(23, 40, 6144)	(23, 40, $6(c+1+24)$)
`pred_conv8_2`	(12, 20, 3072)	(12, 20, $6(c+1+24)$)
`pred_conv9_2`	(6, 10, 1536)	(6, 10, $6(c+1+24)$)
`pred_conv10_2`	(3, 5, 1536)	(3, 5, $6(c+1+24)$)
`pred_conv11_2`	(2, 3, 1536)	(2, 3, $4(c+1+24)$)
`pred_conv12_2`	(1, 2, 1536)	(1, 2, $4(c+1+24)$)

3-4 Training

To train the ACT-detector, we give an example input image sequence X and corresponding ground-truth tubelets y. The ground-truth tubelets y contain the class labels and sequences of bounding box coordinates of all actions happening in image sequence X. We represent the ground-truth tubelets as a tensor of size $(N, 1+4K)$, where N is the number of actions, 1 is the class label, and $4K$ are the bounding box coordinates for all boxes in the sequence. The ACT-detector predicts tubelets of size $(21842, c+1+4K)$, where $c+1$ is the number of classes (including background). To compare the ground-truth tubelets to the predicted tubelets, we encode the ground-truth tubelets into a shape similar to the predicted tubelets. We do this by a process called "anchor cuboid matching". Before we discuss anchor cuboid matching, we briefly mention the spatio-temporal IoU metric.

3-4-1 Spatio-temporal IoU

We can measure the amount of overlap of two bounding boxes by using the Intersection over Union (IoU) metric. This metric is also referred to as the Jaccard index. As the name implies, for two bounding boxes A and B, the IoU is defined as the intersection of A and B divided by the union of A and B:

$$\text{IoU} = \frac{A \cap B}{A \cup B}. \tag{3-1}$$

Equation (3-1) is visualized in Figure 3-3.

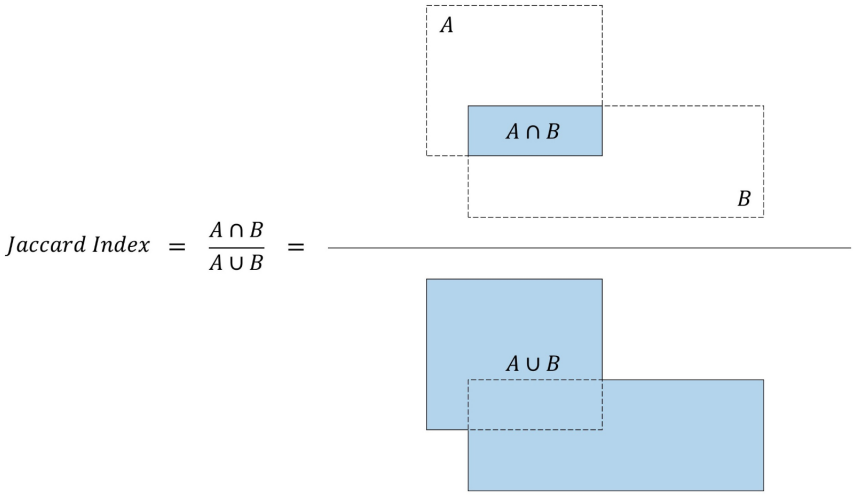


Figure 3-3: A visualization of the IoU. Figure adapted from [30].

We can extend the IoU along the temporal dimension by taking the average of the per-frame IoU. This is known as the spatio-temporal IoU [1]. We use the spatio-temporal IoU to compare two tubelets of equal temporal length.

3-4-2 Anchor cuboid matching

Anchor cuboid matching matches the anchor cuboids of the ACT detector to the ground-truth tubelets. To do so, we compute the spatio-temporal IoUs between all anchor cuboids B and all N ground-truth tubelets. This is a tensor of size (B, N). The value of B is 21842 in our implementation of the ACT-detector. For each ground-truth tubelet, we match the ground-truth tubelet to the anchor cuboid with the highest spatio-temporal IoU. Then, we match anchor cuboids with a spatio-temporal IoU larger than a threshold (0.5) to the ground-truth tubelet with the highest spatio-temporal IoU. The anchor cuboids that are matched to any ground-truth tubelets are considered positives. The remaining anchor cuboids are considered negatives. The positive anchor cuboids are assigned the class labels and sequence of bounding box coordinates of the ground-truth tubelets they are matched with. The negative anchor boxes are assigned the background class. Anchor cuboid matching is visualized in Figure 3-4.

Figure 3-4: Anchor cuboid matching. The anchor cuboids (red) are matched to the ground-truth tubelet (green). For this, the spatio-temporal IoU is used as a criterion.

3-4-3 Loss function

Only sequences that contain the same actions for all frames in the sequence are considered for training. During training, a sequence of frames is given as input to the network. The network produces an output, which is compared to the ground-truth tubelets. Let $x_{ij} \in \{0, 1\}$ be the binary variable whose value is 1 if and only if anchor cuboid a_i is matched to the ground-truth tubelet g_j of label y. The loss $\mathcal{L}$ between the predicted tubelets and the ground-truth tubelets is computed as:

$$\mathcal{L} = \frac{1}{N}(\mathcal{L}_{conf} + \mathcal{L}_{reg}), \tag{3-2}$$

where N is the number of positive assigned anchor boxes, $\mathcal{L}_{conf}$ the confidence loss, and $\mathcal{L}_{reg}$ the regression loss. The loss $\mathcal{L}$ is the sum of the confidence loss $\mathcal{L}_{conf}$ and the regression loss $\mathcal{L}_{reg}$. This enables the ACT-detector to learn classification and regression of anchor cuboids jointly. The confidence loss $\mathcal{L}_{conf}$ is defined as:

$$\mathcal{L}_{conf} = -\sum_{i \in \mathcal{P}} x_{ij}^{y} \log(\hat{c}_i^{y}) - \sum_{i \in \mathcal{N}} \log(\hat{c}_i^{0}), \tag{3-3}$$

where $\mathcal{P}$ denotes the set of positive anchor cuboids, $\mathcal{N}$ denotes the set of negative anchor cuboids, $\hat{c}_i^{y}$ is the predicted confidence score for label y, and $\hat{c}_i^{0}$ is the predicted confidence score for the background class. Equation (3-3) is known as the cross-entropy loss. The regression loss $\mathcal{L}_{reg}$ is defined as:

$$\mathcal{L}_{reg} = \frac{1}{K} \sum_{i \in \mathcal{P}} \sum_{c \in \{x,y,w,h\}} x_{ij}^{y} \sum_{k=1}^{K} \text{smooth}_{\text{L1}}(\hat{r}_i^{c_k} - \mathfrak{g}_{ij}^{c_k}),$$

$$\begin{aligned} \text{with} \quad \mathfrak{g}_{ij}^{x_k} &= \frac{g_j^{x_k} - a_i^{x_k}}{a_i^{w_k}}, \\ \mathfrak{g}_{ij}^{y_k} &= \frac{g_j^{y_k} - a_i^{y_k}}{a_i^{h_k}}, \\ \mathfrak{g}_{ij}^{w_k} &= \log\left(\frac{g_j^{w_k}}{a_i^{w_k}}\right), \\ \mathfrak{g}_{ij}^{h_k} &= \log\left(\frac{g_j^{h_k}}{a_i^{h_k}}\right), \end{aligned} \tag{3-4}$$

where $\hat{r}_i^{c_k}$ is the predicted regression for the $c \in \{x, y, w, h\}$ coordinate of anchor a_i at frame f_k, and g_j is the ground-truth tubelet. The regression loss is defined using a $\text{smooth}_{\text{L1}}$ loss [28] and is averaged over K frames.

3-5 Inference

Given a sequence of frames as input to the ACT-detector, the output is B tubelets. The value of B in our implementation is 21842. This number is much higher than the number of actions in the sequence. Thus, we filter the tubelets when we infer the model on sequences not seen in the training data. Filtering is done in two steps: 1. threshold filtering and 2. non-maximum suppression (NMS). Threshold filtering is simply removing the tubelets with a low confidence score. We set the threshold for this at 0.05. Non-maximum suppression is to remove the tubelets that have a spatio-temporal IoU larger than a threshold θ with the tubelet with the highest confidence. Non-maximum suppression is performed on a per-class bases. We chose $\theta = 0.3$, as was done by [1]. This is repeated until there are no tubelets that match this criterion.

3-6 Tubelet linking

So far we have only discussed processing sequences of K frames by the ACT-detector. However, an untrimmed video usually has many more frames than K. We process such videos by splitting the video into sequences of K frames. For a video of N frames, the frames are processed in windows of $[f_1, f_2, \ldots, f_K]$, $[f_2, f_3, \ldots, f_{K+1}]$, $\ldots$, $[f_{N-K+1}, f_{N-K+2}, \ldots, f_N]$, where f_n denotes frame n. Each sequence is given as input to the ACT-detector. The ACT-detector computes tubelets for each sequence. These tubelets are, by design, fixed in their temporal length, i.e., they are sequences of K bounding boxes. To follow the action in time, beyond the sequences, we process these tubelets into temporally coherent *action tubes*, or *tubes* for short. Similar to tubelets, tubes are sequences of bounding boxes. However, their temporal extent is not fixed. They can, for example, be 10 frames long or even 1000 frames long. Processing

tubelets into tubes is known as "tubelet linking" [1]. Tubelet linking is done by an external processing algorithm and was originally proposed by [10] for linking frame-level detections. The algorithm was extended by [1] for linking sequence-level detections (tubelets). We briefly discuss the tubelet linking algorithm in the following paragraphs.

Input tubelets For each sequence in a video, we compute B tubelets. We perform per-class NMS and keep the top $N = 10$ tubelets for each class. These tubelets are used as candidate tubelets for tubelet linking.

Initialization In the first sequence, a new link is started for each of the N tubelets. A link L is a sequence of tubelets. The confidence score of a link is the average of the confidence scores of the tubelets in the link. At a given sequence, new links start for tubelets that are not associated to any existing links.

Overlap The tubelet linking algorithm depends on an overlap measure between a link L and tubelet t. The overlap of L and t is defined as the spatio-temporal IoU between the last tubelet of the link L and t.

Linking Given a new sequence, we extend the existing links with one of the N tubelets starting from the sequence. We extend a link L, with tubelet t if it meets the following criteria: (i) t is not already selected by an other link, (ii) t has the highest confidence score, and (iii) the overlap of L and t is larger or equal than a threshold τ. For our experiments, we chose $\tau = 0.1$.

Termination A link is terminated if no tubelets meet the criteria discussed in the previous paragraph for $K - 1$ consecutive frames.

Temporal smoothing We build an action tube for each link. The confidence score of a tube is equal to the confidence score of the link. For each sequence in the link, we average the bounding box coordinates of tubelets that pass through the same frames. We now have a sequence of averaged bounding box coordinates through time, i.e, an action tube. We chose to average the bounding box coordinates to obtain smoother tubes.

Chapter 4

Dataset and performance evaluation

The previous chapter discussed our implementation of the ACT-detector, the framework we use to evaluate single-frame and multi-frame models for action detection. This chapter discusses the dataset and the performance measures we chose for evaluating these models.

4-1 Dataset

This thesis considers the surveillance application of action detection. The chosen dataset for this application is the VIRAT dataset [2] from the Activity in Extended Videos Prize Challenge 2018 (ActEV-PC 2018) [31]. VIRAT is a video dataset that contains recordings of surveillance cameras of multiple scenes. An example scene of VIRAT is shown in Figure 4-1. Depending on the scene, these recordings have pixel resolutions of 1280×720 or 1920×1080. All videos are recorded at 30 fps. The reason why we chose VIRAT is that it contains high-quality recordings and annotations.

VIRAT has labels for 18 action classes. Some example classes are: "activity_carrying", "pulling", "entering", "exiting", "vehicle_turning_right", and "vehicle_turning_left". The training set has 1138 labeled action tubes, which in total are 247971 labeled boxes. Multiple actions may happen simultaneously. There are also time spans in which no actions take place. VIRAT is considered challenging in terms of pixel resolutions of humans, the wide spatial coverage of scenes, and background clutter [2].

The dataset comes with a training, validation, and test split. As the test split does not contain annotations, the validation split is regarded as the test split for this thesis. The training and validation sets contain 64 and 54 videos each, where each video is between 25 seconds and 14 minutes long. The videos in the validation set contain recordings from the same scenes as videos in the training set.

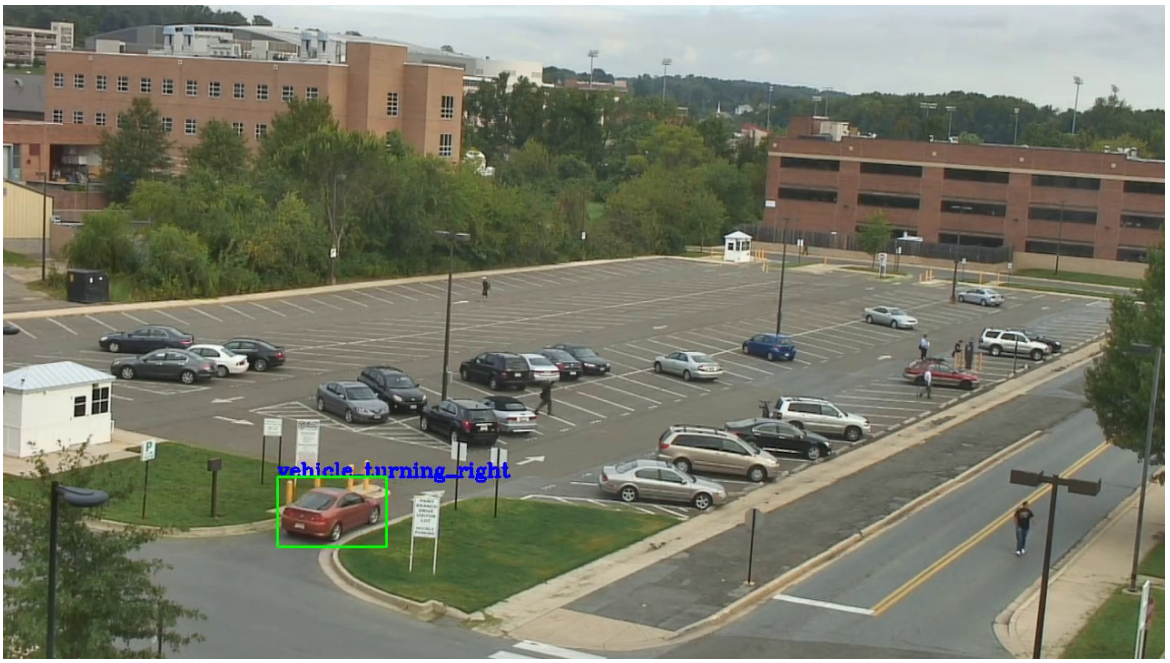


Figure 4-1: An example scene and action from the VIRAT dataset. The blue text is the action label (in this case "vehicle_turning_right"), and the green box is the action location.

4-2 Performance evaluation

The performance of action detection frameworks can be quantified by means of frame-level metrics and video-level metrics. Frame-level metrics compare the ground-truth boxes at frame f, to the boxes originating from all predicted tubelets that pass through frame f. Frame-level metrics are independent of the linking strategy. Video-level metrics compare the predicted tubes of video v to the ground-truth tubes of video v. The following performance metrics are used:

- Classification accuracy;
- Mean Average Best Overlap (MABO);
- Frame-mAP: Frame mean Average Precision; and
- Video-mAP: Video mean Average Precision.

Before we discuss the above performance metrics, we define the following terms:

- Precision and recall; and
- Average Precision (AP).

4-2-1 Precision and recall

Precision and recall are commonly used metrics for classification and detection tasks. They are also used to compute the Average Precision, a metric that will be discussed in the next paragraph. Precision (P) is defined as the number of true positives (T_p) divided by the number of true positives plus the number of false positives (F_p):

$$P = \frac{T_p}{T_p + F_p}, \tag{4-1}$$

The sum of the number of true positives and the number of false positives is the total number of positive predictions. Precision measures how accurate the positive predictions of a system are. High precision relates to a low false positive rate. A precision of 1.0 means that every positive prediction of the system is correct.

The recall (R) is defined as the number of true positives (T_p) divided by the number of true positives plus the number of false negatives (F_n):

$$R = \frac{T_p}{T_p + F_n}, \tag{4-2}$$

The sum of the number of true positives and the number of false negatives is the number of ground-truths. Recall measures how many ground-truths are detected. High recall relates to a low false negative rate. A recall of 1.0 means that all ground-truths were detected.

Both precision and recall are necessary to measure the performance of a detection system. A detection system with high precision but low recall misses many detections and a system with low precision but high recall has a high false positive rate. A good performing system has both high precision and high recall.

4-2-2 Average Precision

As predictions have a class confidence score, the precision and recall scores can be evaluated at various confidence thresholds. Typically, for high confidence thresholds, the precision is high and the recall is low, and for low confidence thresholds, the precision is low and the recall is high. These precision and recall scores can be plotted to create a precision-recall curve. Example precision-recall curves are shown in Figure 4-2.

The area under a precision-recall curve is commonly used to summarize the plot and is referred to as Average Precision. The Average Precision (AP) is computed as:

$$AP = \sum_n (R_n - R_{n-1}) P_n, \tag{4-3}$$

where P_n and R_n are the precision and recall scores at the nth threshold. The Average Precision is computed per class. The Average Precision averaged over all classes is known as the mean Average Precision (mAP). The mean Average Precision is used to compute the frame-mAP and video-mAP scores for action detection systems. These metrics will be discussed in the upcoming paragraphs.

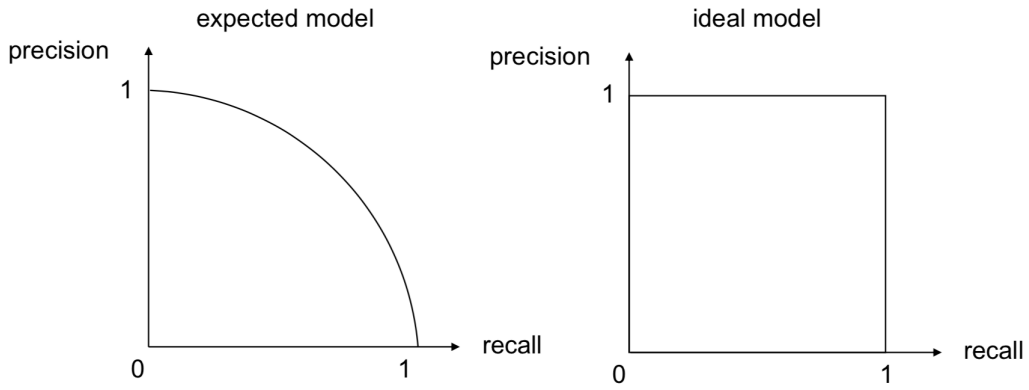


Figure 4-2: Example precision-recall curves. Typically, the precision is high for low recall values, and is low for high recall values. The ideal model has high precision and high recall. Figure adapted from [20].

4-2-3 Classification accuracy

The classification accuracy [1] measures the ratio of correctly classified boxes. This is a frame-level metric and is evaluated per class. For each ground-truth box of class c, the IoU is computed between the ground-truth box and all predicted boxes. The predicted boxes that have an IoU larger than 0.5 are averaged in their class confidence scores. If the maximum of these scores corresponds to class c, the ground-truth box is correctly classified. As the classification accuracy is computed per class, the final score is defined as the mean over all classes.

4-2-4 MABO

Mean Average Best Overlap (MABO) [1] measures localization performance of action detection systems. Similar to classification accuracy, for each ground-truth box, the IoU is computed between the ground-truth box and all predicted boxes. The IoU between the ground-truth box and the best overlapping predicted box is kept (BO). Then, for each class, the average over all ground-truth boxes is taken (ABO). Finally, the mean is taken over all classes (MABO). MABO is a frame-level metric.

4-2-5 Frame-mAP

Frame-mAP [9] is a commonly used metric to measure the detection performance of action detection systems. The frame-mAP is a frame-level metric, which means that the predicted boxes at frame f are compared to the ground-truth boxes at frame f. A predicted box is considered correct if its IoU with the ground-truth box is larger than a threshold θ and its class label is correctly predicted. For each class, the frame-AP is computed, which is then averaged over all classes to form the frame-mAP. As frame-mAP is independent of the linking strategy, frame-mAP is a good measure to compare detection performances of models.

4-2-6 Video-mAP

Video-mAP [9] is used to measure the detection performance of action detection systems on the video-level. For video-mAP, the predicted tubes of video v are compared to the ground-truth tubes of video v. A predicted tube is considered correct if its spatio-temporal IoU with the ground-truth tube is larger than a threshold θ and its class label is predicted correctly. For each class, the video-AP is computed, which is then averaged over all classes to form the video-mAP. Video-mAP is a good measure to compare detection performances of models and linking strategies combined.

4-2-7 Weighted evaluation

We also evaluate the weighted classification accuracy, weighted MABO, weighted frame-mAP, and weighted video-mAP. We define these by weighting the per-class performances by their appearances. For frame-level metrics, we weight by the frame-level appearances, and for video-level metrics, we weight by the video-level appearances. Frame-level appearances refer to the number of bounding boxes for a specific class and video-level appearances refer to the number of tubes for a specific class. The number of bounding boxes divided by the number of tubes is a measure for the average duration of a tube. The frame-level and video-level appearances of classes of VIRAT are shown in Table 4-1. The weighted evaluation metrics are then computed as:

$$M_w = \sum_c \frac{A_c}{\sum_c A_c} M_c, \tag{4-4}$$

where M_w denotes the weighted performance score, A_c the appearance of class c, and M_c the class performance score. M could be one of classification accuracy, MABO, frame-mAP, or video-mAP. Values for A_c can be found in Table 4-1. The frame-level appearances are used to compute the weighted classification accuracy, weighted MABO, and weighted frame-mAP. The video-level appearances are used to compute the weighted video-mAP.

The reason why we consider weighted evaluation is that there is a large variation in class appearances (see Table 4-1). This is known as class imbalance. Weighting these classes equally could skew the performance numbers if the classes with little appearances perform much better or worse than classes with high appearances. The implication of class imbalance is discussed in Section 6-1.

Table 4-1: Frame-level and video-level appearances of classes in the training set of VIRAT.

Class	Frame-level appearances	Video-level appearances
Closing	5326	132
Closing_trunk	1138	21
Entering	7701	71
Exiting	5371	65
Loading	4715	37
Open_trunk	1378	22
Opening	7404	127
Pull	9252	23
Riding	9420	22
Talking	14698	41
Transport_HeavyCarry	15661	31
Unloading	4465	32
activity_carrying	128422	205
specialized_talking_phone	5098	17
specialized_texting_phone	683	4
vehicle_turning_left	12783	133
vehicle_turning_right	12843	137
vehicle_u_turn	1613	8

Chapter 5

Experiments

Two distinct experiments have been conducted to evaluate single-frame and multi-frame models for action detection in surveillance videos. The framework used for this evaluation is the ACT-detector [1] and the surveillance videos are from the VIRAT dataset [2]. ACT $K = 1$ serves in this evaluation as the single-frame model and ACT $K > 1$ as the multi-frame models. The following sections present and discuss the results of these experiments.

5-1 Experiment A: Single-frame models versus multi-frame models

In this first experiment, we compare single-frame and multi-frame models on action detection performance in surveillance videos. The goal is to assess whether multi-frame models outperform single-frame models for this use case. For this, we train and evaluate ACT for $K = 1, 2, 4, 6$, and 12. The sequences used for training originate from all 64 training videos and do not contain overlapping frames. We chose not to overlap frames during training as we found little difference in performance by doing so and to reduce the training time of our models. For testing, we use all 54 test videos. The test videos are processed in overlapping sequences, as described in Section 3-6. Each sequence is given as input to ACT. The computed tubelets are used for tubelet linking and performance evaluation.

5-1-1 Qualitative comparison

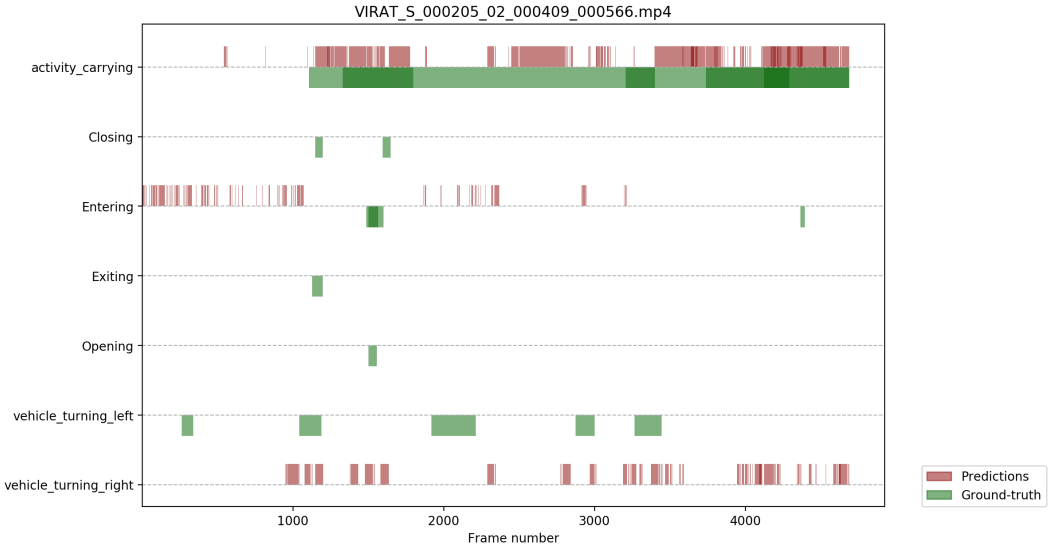
Once the predicted tubes are computed, we can draw them on the videos. We can then visually observe how accurate our detections are. We visually compare the predicted tubes for ACT $K = 1$ and ACT $K > 1$. We see that $K = 1$ misses detections more frequently, i.e., actions found in one frame may have been missed in the next. This occurs less frequently for $K > 1$. The idea of missing detections has been visualized in Figure 5-1. We also notice that actions that may have been ambiguous on individual frames are classified more accurately for $K > 1$. Finally, we see more precise localization for $K > 1$; the actions are followed more smoothly through time.

(a) Predicted tube for ACT $K = 1$.

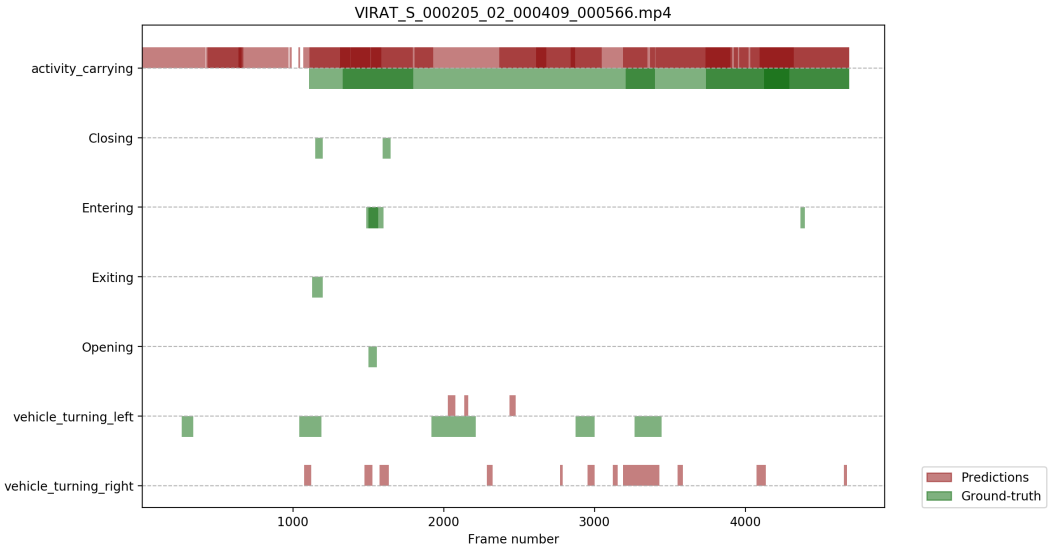
(b) Predicted tube for ACT $K = 6$.

Figure 5-1: A visualization of the predicted tubes of ACT $K = 1$ and ACT $K = 6$ for the video `VIRAT_S_000205_02_000409_000566`. The action "activity_carrying" is missed by ACT $K = 1$ in the 4th frame (4th row).

The visual comparison can be summarized by the figure shown in Figure 5-2. The figure shows the time ranges of the predicted and ground-truth tubes for a video in the test set for ACT $K = 1$ and ACT $K = 6$. A higher color intensity in the time ranges signifies multiple actions of the same class are taking place simultaneously. This figure shows that the predicted tubes of $K = 1$ contain more gaps as compared to the predicted tubes of $K = 6$. Gaps in the time ranges mean that there are no detections in those frames. This confirms our findings from the visual comparison of ACT $K = 1$ missing more detections than ACT $K > 1$. We observe similar behavior for other videos in the test set.



(a) Time ranges of tubes for ACT $K = 1$.



(b) Time ranges of tubes for ACT $K = 6$.

Figure 5-2: A visualization of the temporal ranges of the predicted and ground-truth tubes of ACT $K = 1$ and ACT $K = 6$ for the video `VIRAT_S_000205_02_000409_000566`. A higher color intensity in the time ranges signifies multiple actions of the same class are taking place simultaneously.

5-1-2 Quantitative comparison

Classification accuracy

The classification and weighted classification accuracies (in percentages) are shown in Table 5-1 and Figure 5-3.

Table 5-1: Classification and weighted classification accuracies of ACT on VIRAT @IoU=0.5.

	Classif accuracy	Weighted classif accuracy
ACT $K = 1$	7.71	17.53
ACT $K = 2$	18.47	44.19
ACT $K = 4$	18.27	45.54
ACT $K = 6$	17.50	45.82
ACT $K = 12$	**19.53**	**50.84**

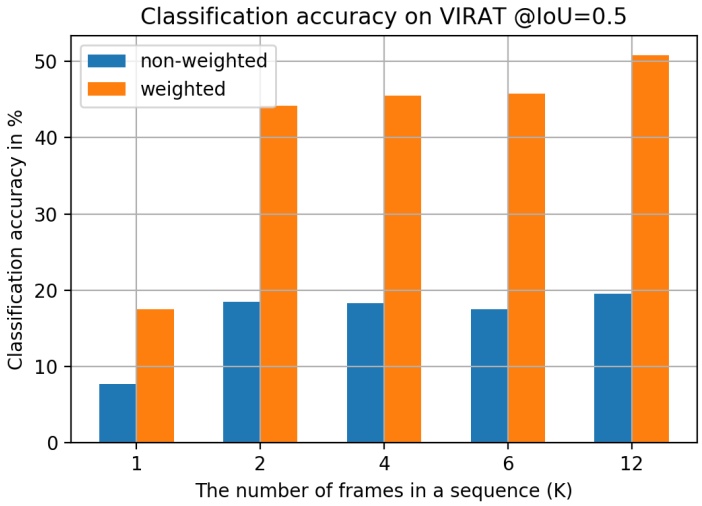


Figure 5-3: Classification accuracy of ACT on VIRAT @IoU=0.5.

When looking at the classification and weighted classification accuracies in Table 5-1, we see a much larger accuracy for $K > 1$ compared to $K = 1$. This shows that multi-frame models perform better in classification than single-frame models for this dataset. As multi-frame models process sequences of frames, they can find motion patterns. Motion patterns can help distinguish actions that have little spatial variance. Two frames are enough to infer motion data, this may explain the large performance difference between $K = 2$ and $K = 1$. As found from the qualitative comparison, single-frame models miss detections more frequently. If a detection is missed, the accuracy for that ground-truth box is considered 0. This contributes to the lower score for $K = 1$. From Table 5-1 we also see that the weighted classification accuracy increases when K increases. This is expected as more frames allow for more motion patterns, making actions more distinguishable.

MABO

The MABO and weighted MABO scores (in percentages) are shown in Table 5-2 and Figure 5-4.

Table 5-2: MABO scores of ACT on VIRAT.

	MABO	Weighted MABO
ACT $K = 1$	21.27	21.55
ACT $K = 2$	54.31	54.86
ACT $K = 4$	56.25	57.14
ACT $K = 6$	57.52	57.95
ACT $K = 12$	**59.59**	**62.01**

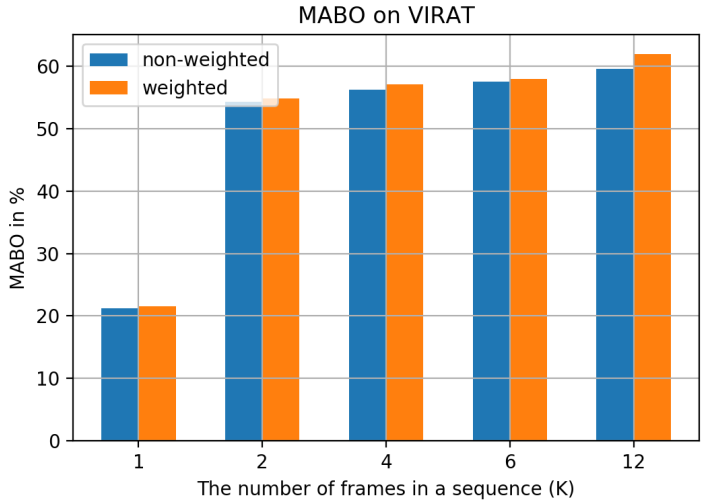


Figure 5-4: MABO scores of ACT on VIRAT.

As MABO is a measure for localization performance, we can see from Table 5-2 that $K > 1$ is more precise in localizing actions than $K = 1$. This confirms our findings from the qualitative comparison. Since multi-frame models output tubelets, bounding boxes are produced for each frame in the sequence. This reduces the chance an action is missed, which improves the MABO scores. An additional reason why the MABO scores are higher for multi-frame models is that a sequence of frames is used to regress the anchor cuboids. The anchor cuboids can use data from neighboring frames to regress the bounding boxes in an anchor cuboid. We also notice that MABO increases for larger values of K. This is expected as with larger K, more frames can be used to regress the anchor cuboids.

We notice little difference when comparing the MABO scores to the weighted MABO scores. When visualizing the tubes of ACT, we saw that ACT is well capable of localizing moving humans and vehicles. Predicting the correct action label is however more challenging. As all actions involve humans and vehicles, and the IoUs are computed independent of the class labels, there is little difference in scores by weighting the scores by their class appearances.

Frame-mAP

The frame-mAP and weighted frame-mAP scores (in percentages) are shown in Table 5-3 and Figure 5-5.

Table 5-3: Frame-mAP scores of ACT on VIRAT @IoU=0.5.

	Frame-mAP	Weighted frame-mAP
ACT $K = 1$	2.59	6.12
ACT $K = 2$	6.06	17.92
ACT $K = 4$	6.69	19.20
ACT $K = 6$	6.91	19.85
ACT $K = 12$	**7.71**	**21.89**

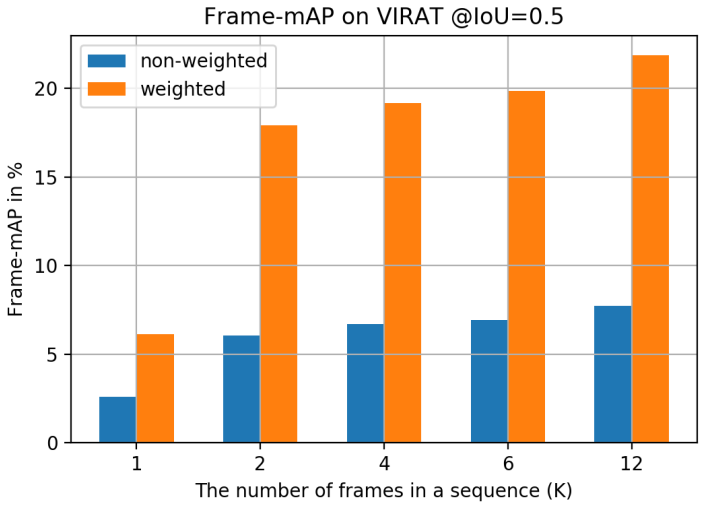


Figure 5-5: Frame-mAP scores of ACT on VIRAT @IoU=0.5.

From Table 5-3 we see that ACT $K > 1$ has higher frame-mAP and weighted frame-mAP scores than ACT $K = 1$. As frame-mAP measures detection performance, this shows that multi-frame models have higher detection performance compared to single-frame models. As shown in the previous paragraphs, multi-frame models have higher classification and localization performances than single-frame models. The task of detection, which for the case of frame-mAP is to localize and classify actions for each frame, becomes easier for models with higher classification and localization performance. Hence, multi-frame models have higher frame-mAP scores than single-frame models.

We also notice an improvement in frame-mAP scores by increasing K. There is a large performance difference between ACT $K = 1$ and ACT $K = 2$. For larger values of K there is a performance increase, but not as significant. This is expected as a similar pattern was observed for the classification and localization performances. The combined task of detection thus follows a similar trend.

Video-mAP

The video-mAP and weighted video-mAP scores (in percentages) are shown in Table 5-4 and Figure 5-6.

Table 5-4: Video-mAP scores of ACT on VIRAT @IoU=0.2.

	Video-mAP	Weighted video-mAP
ACT $K = 1$	0.93	1.65
ACT $K = 2$	5.47	7.4
ACT $K = 4$	4.45	6.95
ACT $K = 6$	4.09	7.37
ACT $K = 12$	**5.01**	**8.91**

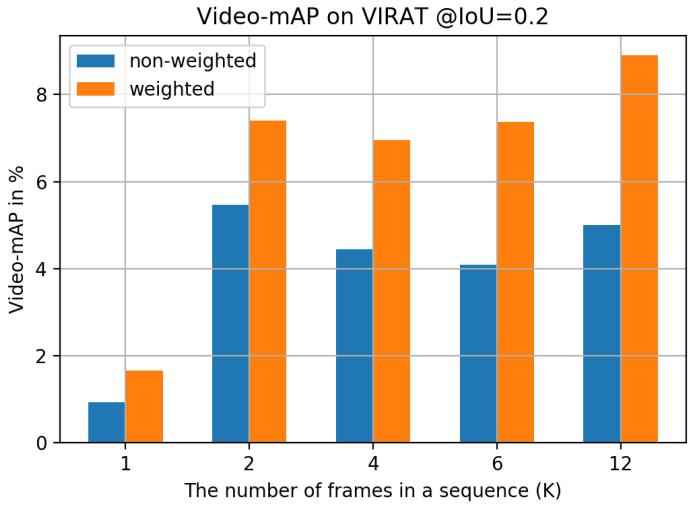


Figure 5-6: Video-mAP scores of ACT on VIRAT @IoU=0.2.

Video-mAP measures the combined performance of the model and the linking strategy. As the linking strategy is the same for all models, video-mAP indirectly measures the action detection performance of a model on the video-level. In Table 5-4 we see a similar pattern for the classification, MABO, and frame-mAP scores. We see a large increase for ACT $K > 1$ compared to ACT $K = 1$. Multi-frame models outperform single-frame models on the video-level as well. The higher detection performances of multi-frame models on the frame-level translate to higher detection performances on the video-level. This is because the tubelet linking algorithm processes the detections obtained at the frame-level. If these detections are more accurate, the produced tubes are more accurate as well. We notice that ACT $K = 12$ is the best performing model, followed by ACT $K = 2$.

Model sizes

We report the number of parameters for each model we evaluated in Table 5-5. We do this to give an impression of the processing speed of the models. The number of parameters depends on K and on the number of classes in the dataset.

Table 5-5: The number of parameters of ACT as a function of K. Here M denotes 1 million.

Model	Parameters
ACT $K = 1$	27M
ACT $K = 2$	31M
ACT $K = 4$	44M
ACT $K = 6$	62M
ACT $K = 12$	142M

ACT $K = 2$ has around 15% more parameters than ACT $K = 1$. However, the performance increase of ACT $K = 2$ is much larger than 15%. For a small increase in parameters, the performance increases significantly. When increasing K to higher values such as 4, 6, and 12, the number of parameters increases substantially. For real-time applications, ACT $K = 4$, $K = 6$, and $K = 12$ are less suitable due to the larger number of parameters.

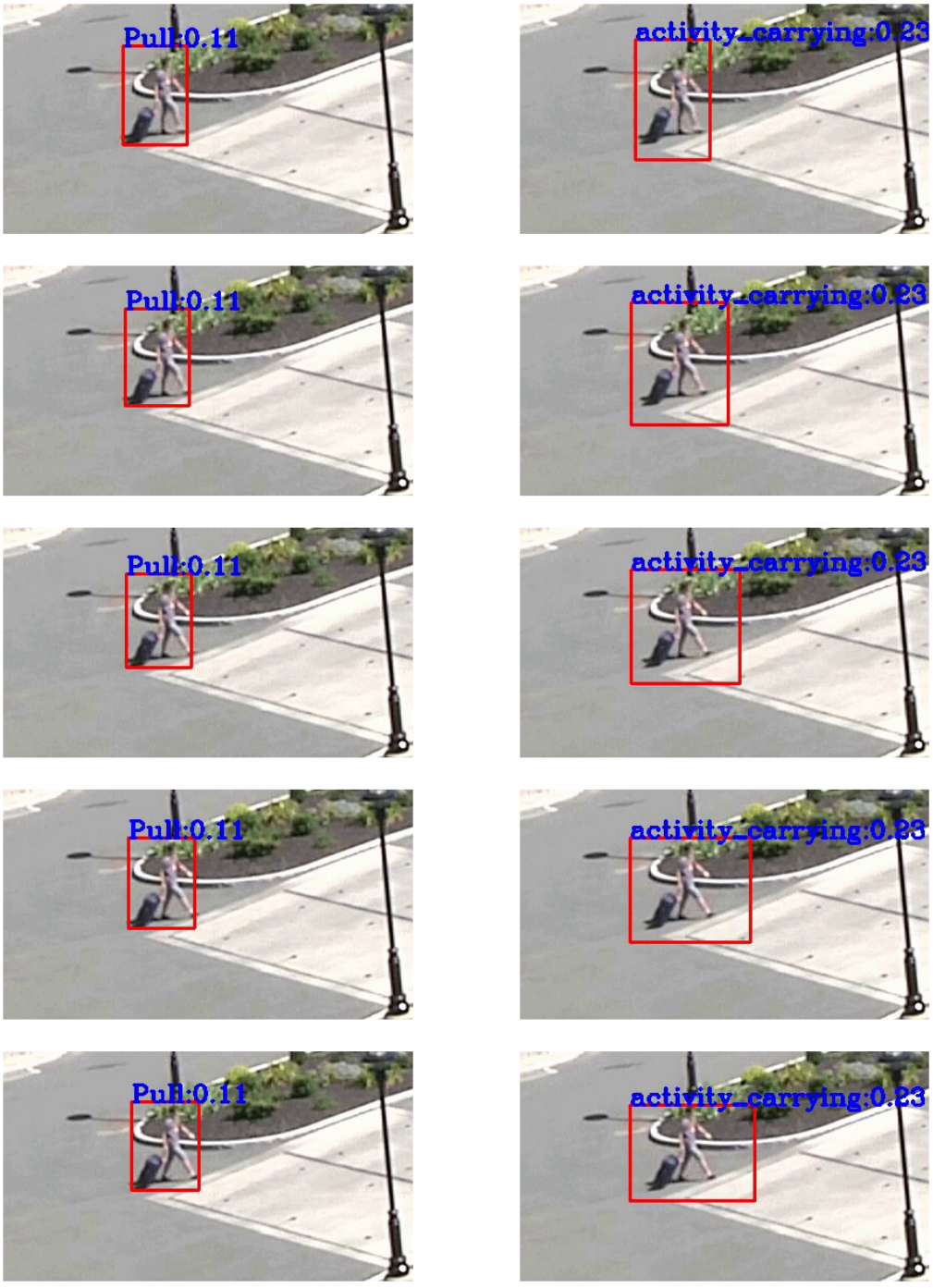
5-2 Experiment B: Temporal order of frames in sequences

For this experiment, we compare multi-frame models trained on ordered sequences of frames and multi-frame models trained on unordered sequences of frames. The goal is to assess whether the performance increase of multi-frame models is purely from the increased number of frames, or also from the temporal order encoded in those frames. We train ACT $K = 6$ on unordered sequences of frames by shuffling the frames in a sequence before we give it as input to the model. We expect that the temporal order in sequences contributes to the performance increase of multi-frame models as actions can be characterized by their dynamics. We use the same 54 test videos and evaluate these videos in a similar fashion as experiment A.

5-2-1 Qualitative comparison

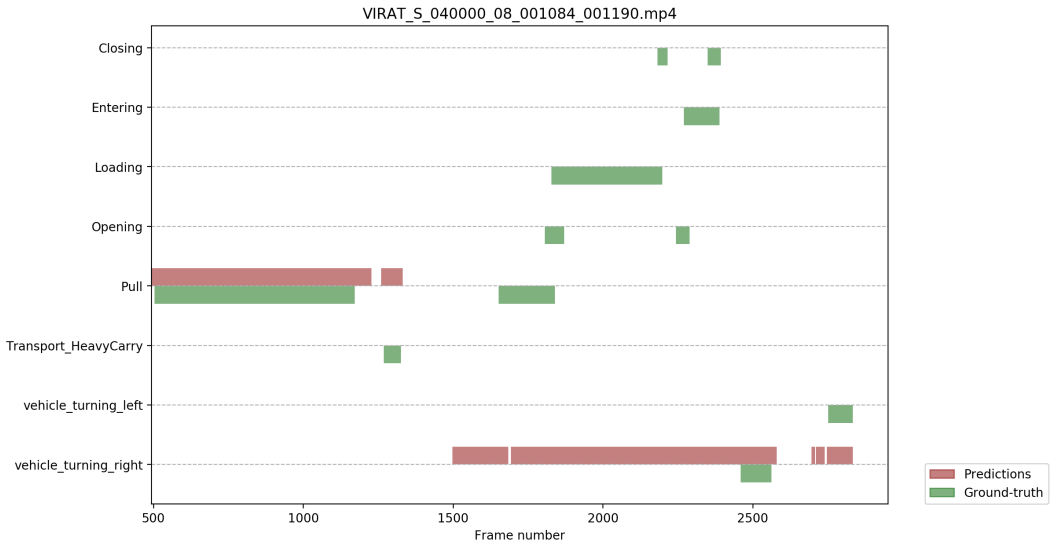
When comparing the predicted tubes for ACT $K = 6$ trained on ordered sequences and for $K = 6$ trained on unordered sequences, we see a difference in localization performance. ACT $K = 6$ trained on unordered sequences is able to follow the action in space and time, however, the localization is less precise compared to $K = 6$ trained on ordered sequences. The behavior we see for $K = 6$ trained on unordered sequences is that the boxes in an action tube expand and shrink sporadically. This is likely because the unordered sequences used for training behave similarly. This behavior is visualized in Figure 5-7.

Figure 5-8 shows the time ranges of the predicted and ground-truth tubes for a video in the test set for ACT $K = 6$ trained on ordered sequences versus ACT $K = 6$ trained on unordered sequences. Similar to ACT $K = 6$ trained on ordered sequences, ACT $K = 6$ trained on unordered sequences has little to no gaps in its predicted tubes. So multi-frame models, even if they learn from unordered sequences, contain less missed detections than single-frame models. Figure 5-8 also shows the confusion of ACT $K = 6$ trained on unordered sequences for the classes "vehicle_turning_right" and "vehicle_turning_left". This could be because the model has not learned the dynamics of these actions during training and therefore, struggles to differentiate between the two.

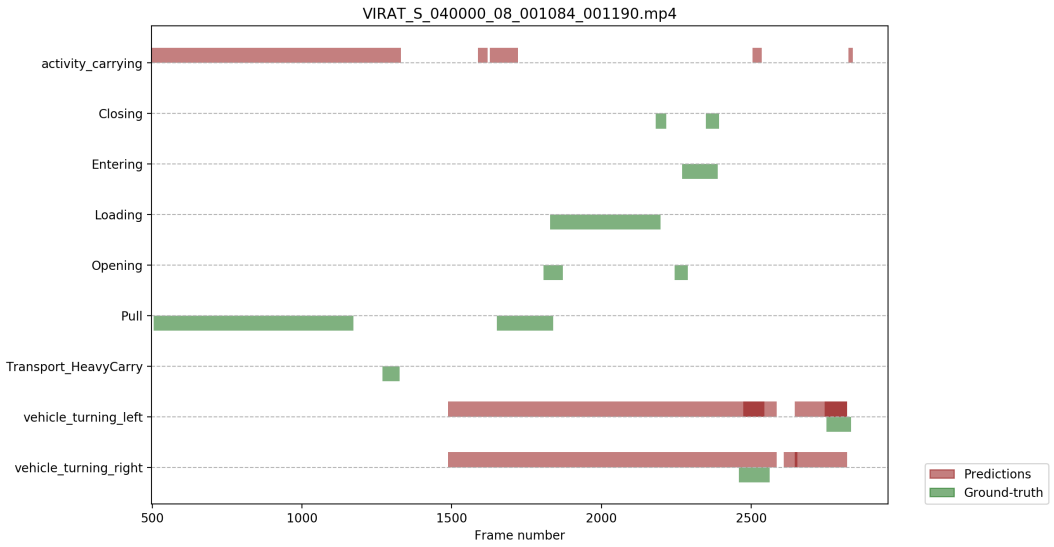


(a) Predicted tube for ACT $K = 6$. **(b)** Predicted tube for ACT $K = 6$ unordered.

Figure 5-7: A visualization of the predicted tubes of ACT $K = 6$ trained on ordered sequences and ACT $K = 6$ trained on unordered sequences for the video `VIRAT_S_040000_08_001084_001190`. The bounding boxes expand for ACT $K = 6$ trained on unordered sequences. The correct class label is "Pull".



(a) Time ranges of tubes for ACT $K = 6$.



(b) Time ranges of tubes for ACT $K = 6$ unordered.

Figure 5-8: A visualization of the temporal ranges of the predicted and ground-truth tubes of ACT $K = 6$ trained on ordered sequences and ACT $K = 6$ trained on unordered sequences for the video `VIRAT_S_040000_08_001084_001190`. A higher color intensity in the time ranges signifies multiple actions of the same class taking place simultaneously.

5-2-2 Quantitative comparison

Classification accuracy

The classification and weighted classification accuracies (in percentages) of ACT for $K = 1$, $K = 6$ trained on ordered sequences, and $K = 6$ trained on unordered sequences are shown in Table 5-6 and Figure 5-9.

Table 5-6: Classification and weighted classification accuracies of ACT for $K = 1$, $K = 6$, and $K = 6$ trained on unordered sequences on VIRAT @IoU=0.5.

	Classif accuracy	Weighted classif accuracy
ACT $K = 1$	7.71	17.53
ACT $K = 6$	**17.50**	**45.82**
ACT $K = 6$ unordered	15.74	44.19

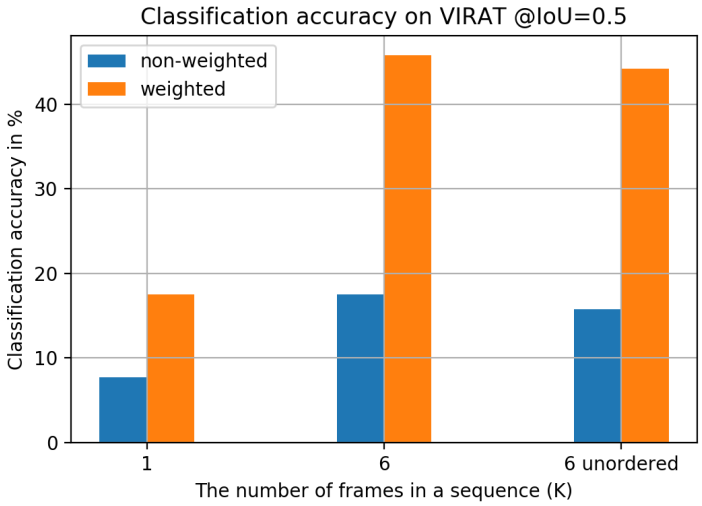


Figure 5-9: Classification accuracy of ACT on VIRAT @IoU=0.5.

From Table 5-6 we see that classification performance decreases when training on unordered sequences of frames in comparison to training on ordered sequences of frames. This is expected as the model has not learned the temporal dynamics of actions, and thus cannot use this characteristic for classification. However, multi-frame models that have not learned action dynamics still outperform single-frame models. Having more data available at the input leads to more accurate predictions. And outputting tubelets means that each frame in the sequence contains predictions. Thus, reducing the chance that a detection is missed. Thereby increasing the classification, localization, and detection performances.

MABO

The MABO and weighted MABO scores (in percentages) of ACT for $K = 1$, $K = 6$ trained on ordered sequences, and $K = 6$ trained on unordered sequences are shown in Table 5-7 and Figure 5-10.

We notice little difference in MABO scores for ACT $K = 6$ trained on ordered sequences and $K = 6$ trained on unordered sequences. From the qualitative comparison we found that the bounding boxes expand or shrink sporadically, indicating a lower MABO score. However, the MABO scores are around equal, with $K = 6$ trained on ordered sequences being slightly higher. We expect this is the case since MABO is evaluated on the frame-level and the visual comparison is done on the video-level. Also, MABO looks at the best overlap in each frame. However, the bounding box with the highest overlap with the ground-truth is not always

Table 5-7: MABO scores of ACT for $K = 1$, $K = 6$, and $K = 6$ trained on unordered sequences on VIRAT.

	MABO	Weighted MABO
ACT $K = 1$	21.27	21.55
ACT $K = 6$	**57.52**	**57.95**
ACT $K = 6$ unordered	57.44	55.86

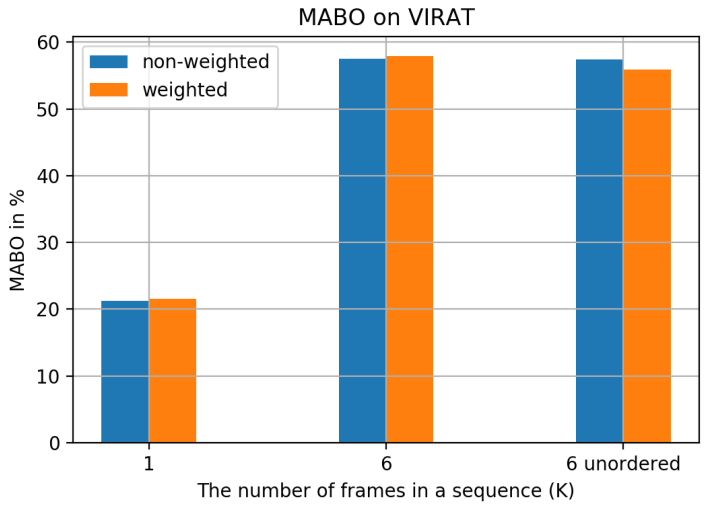


Figure 5-10: MABO scores of ACT on VIRAT.

chosen for tubelet linking since tubelet linking has no access to the ground-truth. Perhaps taking the average overlap would better reflect the behavior we observed in the qualitative comparison. We still notice a large performance increase for ACT $K = 6$ trained on unordered sequences compared to ACT $K = 1$.

Frame-mAP

The frame-mAP and weighted frame-mAP scores (in percentages) of ACT for $K = 1$, $K = 6$ trained on ordered sequences and $K = 6$ trained on unordered sequences are shown in Table 5-8 and Figure 5-11.

Table 5-8: Frame-mAP scores of ACT on VIRAT @IoU=0.5.

	Frame-mAP	Weighted frame-mAP
ACT $K = 1$	2.59	6.12
ACT $K = 6$	**6.91**	**19.85**
ACT $K = 6$ unordered	5.25	16.33

The frame-mAP scores decrease when training on unordered sequences. Due to the lower classification accuracies and slightly lower MABO scores, the detection scores are lower as well. The temporal order of frames matters for learning multi-frame models for action detection.

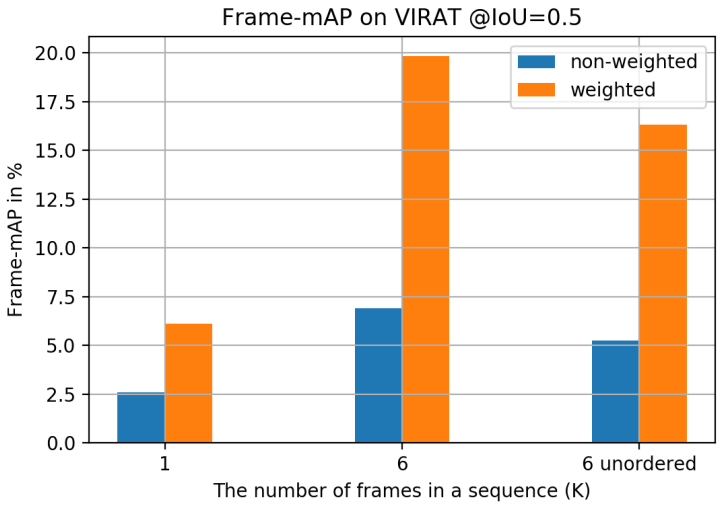


Figure 5-11: Frame-mAP scores of ACT on VIRAT @IoU=0.5.

Multi-frame models trained on ordered sequences outperform multi-frame models trained on unordered sequences. Again we note the frame-mAP scores of multi-frame models trained on unordered sequences are higher than the frame-mAP scores of single-frame models.

Video-mAP

The video-mAP and weighted video-mAP scores (in percentages) of ACT $K = 1$, $K = 6$ trained on ordered sequences and $K = 6$ trained on unordered sequences are shown in Table 5-9 and Figure 5-12.

Table 5-9: Video-mAP scores of ACT on VIRAT @IoU=0.2.

	Video-mAP	Weighted video-mAP
ACT $K = 1$	0.93	1.65
ACT $K = 6$	**4.09**	**7.37**
ACT $K = 6$ unordered	3.05	5.13

From Table 5-9 we see that the video-mAP score decreases when trained on unordered sequences. Since the detections on the frame-level are worse for ACT $K = 6$ trained on unordered sequences, the produced tubes are also worse compared to $K = 6$ trained on ordered sequences. However, the video-mAP score of ACT $K = 6$ trained on unordered sequences is higher than ACT $K = 1$. Using multi-frame models, even when trained on unordered sequences, leads to better results than training on individual frames.

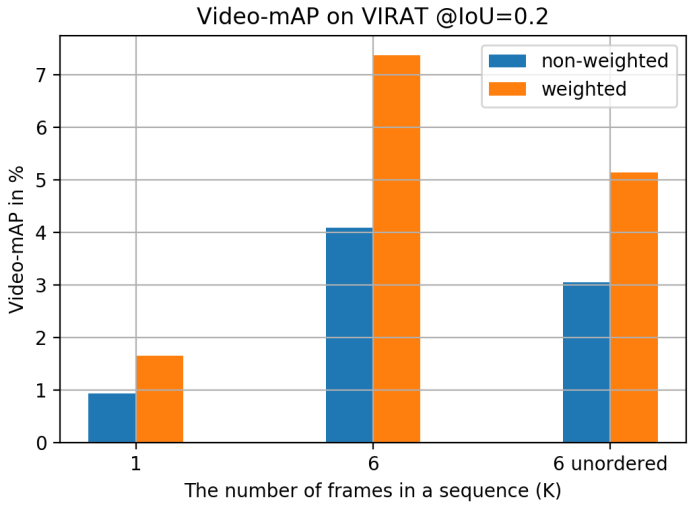


Figure 5-12: Video-mAP scores of ACT on VIRAT @IoU=0.2.

Chapter 6

Discussion

This chapter discusses the performance of ACT on VIRAT in more detail. The per-class classification accuracies and the localization performance for small pixel resolution actions will be analyzed. The focus is on multi-frame models ($K > 1$).

6-1 Per-class classification accuracy

Table 5-1 and Figure 5-3 in Section 5-1-2 show the classification and weighted classification accuracies (in percentages) of ACT on VIRAT. As can be seen from Table 5-1, there is a large difference in the classification and the weighted classification accuracies. For example, the classification accuracy of ACT $K = 12$ is 19.53, whereas the weighted classification accuracy is 50.84. As mentioned in Section 4-2-7, the classification accuracy takes the average of the per-class classification accuracies and the weighted classification accuracy weights the per-class classification accuracies by their appearances. The reason why the classification accuracy is lower than the weighted classification accuracy is that some classes are more challenging to classify than others (will be discussed in the following paragraphs). For such classes, the classification accuracy is low. This can be seen in Table 6-1, which shows the per-class accuracies (in percentages) for ACT $K = 12$ and the per-class appearances of VIRAT. The classification accuracy (the average of the per-class accuracies) is low because the per-class accuracies of some classes are low.

We also see a large variance in class appearances in Table 6-1. As mentioned in Section 4-2-7, this is known as class imbalance. Due to this class imbalance, the models become biased during training. When making predictions, the models tend to predict classes with more appearances since they saw more examples of these during training. For example, when the action "Exiting" takes place, in which a person exits a vehicle, it is often the case that the model predicts "activity_carrying" for the person, but misses the interaction of the person with the vehicle. This is visualized in Figure 6-1. This is a trend we have noticed for some other classes as well. Classes with low appearances are rarely predicted, resulting in low classification accuracies for these classes.

Table 6-1: Per-class accuracies for ACT $K = 12$ on VIRAT @IoU=0.5 and the frame-level appearances of VIRAT.

Class	Classif accuracy	Frame level appearance
Closing	1.16	5326
Closing_trunk	5.62	1138
Entering	0.74	7701
Exiting	0.61	5371
Loading	21.53	4715
Open_trunk	3.85	1378
Opening	0.65	7404
Pull	64.79	9252
Riding	7.98	9420
Talking	43.47	14698
Transport_HeavyCarry	22.10	15661
Unloading	3.18	4465
activity_carrying	75.11	128422
specialized_talking_phone	0.59	5098
specialized_texting_phone	0.0	683
vehicle_turning_left	39.35	12783
vehicle_turning_right	48.99	12843
vehicle_u_turn	11.90	1613

We investigate the per-class accuracies of ACT $K = 12$ for all test videos and plot the results in Figure 6-2. In Figure 6-2, we indeed see a bias for the model. Actions such as "Closing", "Closing_Trunk", "Entering", "Exiting", "Loading", and "Open_Trunk" have low accuracies. This could be because they have fewer appearances during training, or it might be because these actions are inherently challenging to classify. We note that these classes are interactions of humans and vehicles and have little spatial variance. We provide some recommendations in Section 7-1 to improve the classification accuracy for such classes.

6-2 Localizing small pixel resolution actions

The surveillance videos from VIRAT are recorded at 1280×720 and 1920×1080 pixel resolutions. Even though these recordings are of high quality, the (human) actions are small in pixel values. The human heights in VIRAT are in the range of 20 to 180 pixels, which means the human to video height ratio is in the range of 2 to 20%. An example small action is shown in Figure 6-3. Sequences of frames like Figure 6-3 are down-scaled to 640×360 before given as input to ACT. So some humans are only up to 10 pixels long. Despite these challenges, ACT shows impressive localization performance. It is able to locate actions that are small in pixel resolutions. One reason for this could be that it detects movements of pixels in sequences of frames to locate actions.

Despite the impressive localization performance, some actions remain challenging to detect due to the low per-class classification accuracies. We conclude that the classification accuracy is the bottleneck for action detection performance in surveillance videos for the ACT-detector.

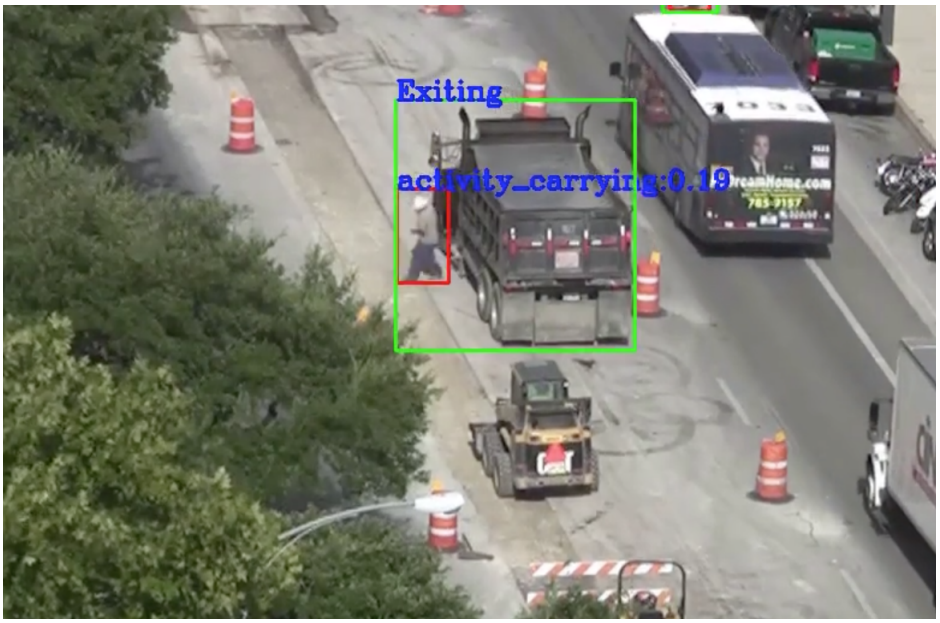


Figure 6-1: An example case of the action "Exiting" being missed. The ground-truth is denoted by the green box and the prediction is denoted by the red box. The action label of the ground-truth is "Exiting" and the action label of the prediction is "activity_carrying".

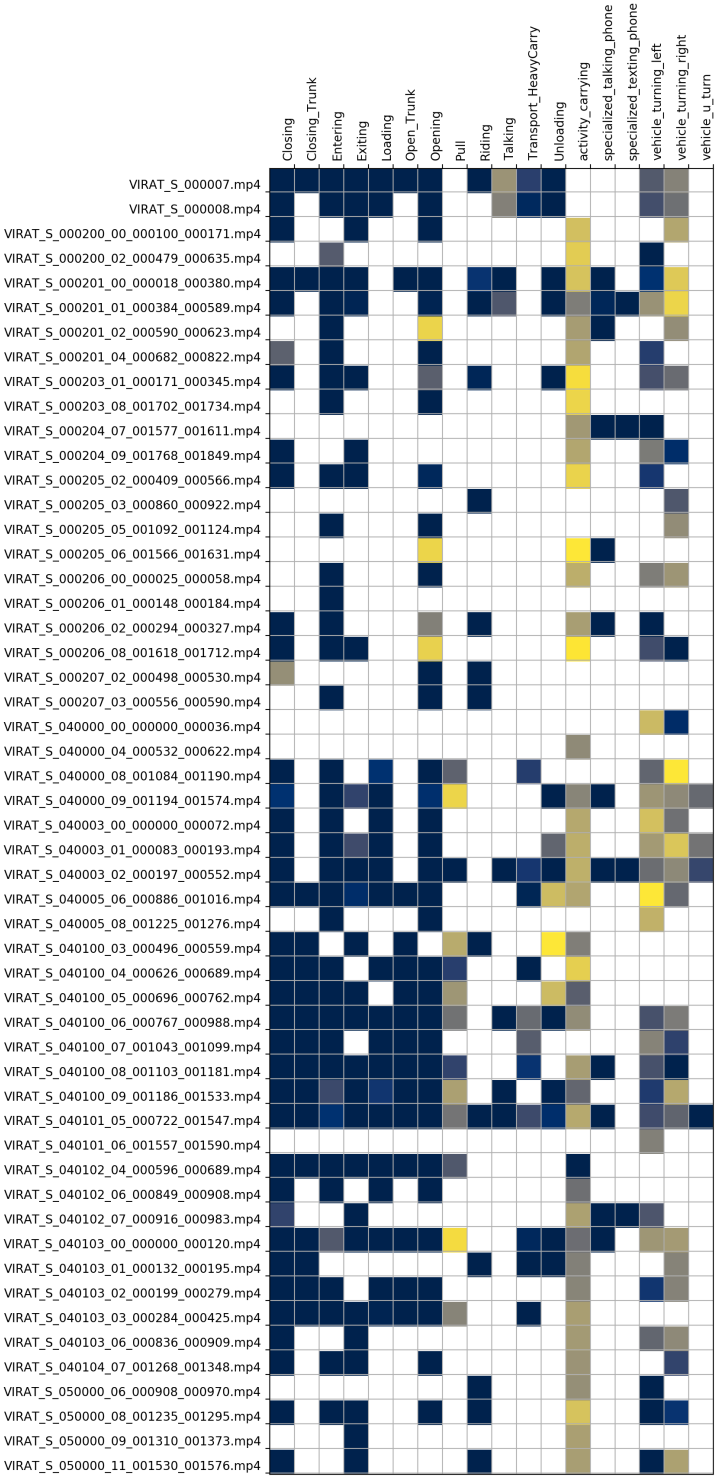


Figure 6-2: Per-class classification accuracy of ACT $K = 12$ for all test videos of VIRAT. The vertical axis denotes the video names of all test videos and the horizontal axis denotes all action classes. A white box indicates that the class does not appear in the video. Dark colors mean low classification accuracy and light colors mean high classification accuracy. The colormap is cividis.

Figure 6-3: A visualization of the localization performance of ACT for small pixel resolution actions. The ground-truth box is denoted by the green box and the prediction is denoted by the red box. The action label of the ground-truth is "activity_carrying" and the action label of the prediction is "activity_carrying".

Chapter 7

Conclusion

This thesis evaluated single-frame and multi-frame deep learning models for action detection in surveillance videos. We implemented the ACT-detector, which takes as input a sequence of K frames and output tubelets (labeled sequences of bounding boxes). We set $K = 1$ to evaluate single-frame models and $K = 2, 4, 6$, and 12 to evaluate multi-frame models. We find that multi-frame models outperform single-frame models for action detection on the VIRAT dataset. Multi-frame models have less missed detections since they output labeled bounding boxes for each frame in a sequence. They also have higher classification accuracies, more precise localization, and higher detection scores. This is because they find motion patterns, which allows them to distinguish actions that are ambiguous in individual frames. From our tests, we find that $K = 12$ is the best performing multi-frame model. However, $K = 2$ performs nearly as well but has fewer parameters. We conclude that ACT $K = 2$ is a good trade-off between detection performance and model size for action detection in surveillance videos.

We also experimented with the temporal ordering of frames for training multi-frame models. We trained ACT $K = 6$ on unordered sequences and compared the performance of this model to ACT $K = 6$ trained on ordered sequences. We find that multi-frame models trained on ordered sequences outperform multi-frame models trained on unordered sequences for action detection on the VIRAT dataset. This shows that the temporal ordering of frames matters for learning multi-frame models, indicating that multi-frame models recognize actions by their dynamics. Nevertheless, multi-frame models outperform single-frame models, even when trained on unordered sequences. Having more data available at the input leads to an increase in performance, regardless of the temporal continuity of the data.

When analyzing the performance of the ACT-detector on the VIRAT dataset, we find that some classes are more challenging to classify than others. However, the localization performance of ACT is impressive. Even for small pixel resolutions, ACT is able to localize actions. For applications for which the class label is less important, the ACT-detector is a good candidate framework. For the surveillance use case, the ACT-detector can be a good framework to aid visual surveillance operators to detect actions with more spatial variance. For actions with less spatial variance, the class label should be re-evaluated manually.

7-1 Future work

We trained and evaluated the ACT-detector on the VIRAT dataset for $K = 1, 2, 4, 6$, and 12. We found an increase in performance when increasing K. This trend, however, may not hold for larger values of K. Actions can become ambiguous if too many frames are used in a sequence, as the same actor has enough time to perform multiple actions. This may ultimately lead to difficulties during training since the model learns to output one label for each anchor cuboid. For this reason, we expect that there is an optimum value for K. We also believe this value is dataset dependent as this optimum depends on the duration of action tubes in the dataset. We chose $K = 12$ as the largest value for our experiments since $K > 12$ resulted in memory exhausted errors during training. This is because of the limited memory of the GPU we used. We leave training and evaluating multi-frame models for larger values of K for future work.

Data augmentation is a commonly used technique to improve the performance of deep learning models for computer vision. Action detection frameworks can also benefit from applying data augmentation during training. For this work, it was decided not to implement a data augmentation pipeline as the focus was on the comparison of deep learning models, and not on maximizing performance. We leave data augmentation to improve performance for future work.

As mentioned in the discussion, some actions of the VIRAT dataset are challenging to classify for ACT. ACT uses 2D convolution to classify anchor cuboids from temporally stacked feature maps. However, recent work in action recognition [6, 7] has shown that 3D convolution is a more promising way of encoding video data. The ACT-detector could be extended to apply 3D convolution. Instead of processing each frame separately and then stacking the resulting feature maps temporally, the sequence of frames would be processed entirely and would be reduced to 3D feature maps. These feature maps can then be used to classify and regress anchor cuboids to form tubelets. 3D convolution is a promising research direction for action detection frameworks.

Another way to increase the classification performance of action recognition and action detection frameworks is to explicitly model human-object interactions. For example, the action "picking up" requires the interaction of human and object. If there are no objects detected nearby the human, then the action "picking up" is likely a false detection. These interactions can be modeled by graph convolutional networks [32]. This is a promising way of adding real-world knowledge to improve the performance of deep learning models for action detection.

Bibliography

[1] V. Kalogeiton, P. Weinzaepfel, V. Ferrari, and C. Schmid, "Action tubelet detector for spatio-temporal action localization," *ICCV, Oct*, vol. 2, 2017.

[2] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, *et al.*, "A large-scale benchmark dataset for event recognition in surveillance video," in *CVPR 2011*, pp. 3153–3160, IEEE, 2011.

[3] Y. Kong and Y. Fu, "Human action recognition and prediction: A survey," *arXiv preprint arXiv:1806.11230*, 2018.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.

[5] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in neural information processing systems*, pp. 568–576, 2014.

[6] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

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

[8] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[9] G. Gkioxari and J. Malik, "Finding action tubes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 759–768, 2015.

[10] G. Singh, S. Saha, M. Sapienza, P. H. Torr, and F. Cuzzolin, "Online real-time multiple spatiotemporal action localisation and prediction.," in *ICCV*, pp. 3657–3666, 2017.

[11] C. Gu, C. Sun, D. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, *et al.*, "Ava: A video dataset of spatio-temporally localized atomic visual actions," in *CVPR 2018*, 2018.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[14] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA:, 2015.

[15] D. Zhang, X. Dai, X. Wang, and Y.-F. Wang, "S3d: Single shot multi-span detector via fully 3d convolutional networks," *arXiv preprint arXiv:1807.08069*, 2018.

[16] H. Huang and C. Wu, "Approximation capabilities of multilayer fuzzy neural networks on the set of fuzzy-valued functions," *Information Sciences*, vol. 179, no. 16, pp. 2762–2773, 2009.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] S. Pattanayak, Pattanayak, and S. John, *Pro Deep Learning with TensorFlow*. Springer, 2017.

[19] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th international conference on computer vision*, pp. 2146–2153, IEEE, 2009.

[20] E. al Hakim, "3D YOLO: End-to-End 3D Object Detection Using Point Clouds," Master's thesis, KTH Royal Institute of Technology, Sweden, 2018.

[21] C. C. Aggarwal, "Neural networks and deep learning," *Cham: Springer International Publishing*, 2018.

[22] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[23] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[24] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[28] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[30] S. Vinodababu, "a pytorch tutorial to object detection." https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection, 2018.

[31] G. Awad, A. Butt, K. Curtis, Y. Lee, J. Fiscus, A. Godil, D. Joy, A. Delgado, A. Smeaton, Y. Graham, *et al.*, "Trecvid 2018: Benchmarking video activity detection, video captioning and matching, video storytelling linking and video search," 2018.

[32] Y. Zhang, P. Tokmakov, M. Hebert, and C. Schmid, "A structured model for action detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9975–9984, 2019.