Change-point Detection Methods for Body-Worn Video

Stephanie Allen[†], David Madras[‡], Ye Ye[§], Greg Zanotti[¶] [*]

July 14, 2017

Abstract

Body-worn video (BWV) cameras are increasingly utilized by police departments to provide a record of police-public interactions. However, large-scale BWV deployment produces terabytes of data per week, necessitating the development of effective computational methods to identify salient changes in video. In work carried out at the 2016 RIPS program at IPAM, UCLA, we present a novel two-stage framework for video change-point detection. First, we employ state-of-the-art machine learning methods including convolutional neural networks and support vector machines for scene classification. We then develop and compare change-point detection algorithms utilizing mean squared-error minimization, forecasting methods, hidden Markov models, and maximum likelihood estimation to identify noteworthy changes. We test our framework on detection of vehicle exits and entrances in a BWV data set provided by the Los Angeles Police Department and achieve over 90% recall and nearly 70% precision — demonstrating robustness to rapid scene changes, extreme luminance differences, and frequent camera occlusions.

1 Introduction

Body-worn video (BWV) cameras are becoming increasingly popular tools for police departments [22]. They are used to provide a record of police-public interactions, and have been shown to increase accountability among officers [15]. Furthermore, BWV has recently become a topic of widespread interest among the general public, especially given the recent controversies regarding police-public relations and policy. To produce this video, police officers wear specially designed cameras on their chests to record their interactions with the public. However, large-scale BWV deployment produces terabytes of data per week, far too much for complete review by humans. This necessitates the development of effective computational methods to identify salient changes in video between various states — such as in or out of a building, interacting or not interacting with the public, and in and out of a car.

In early architectures in the literature, changes in videos are detected using a variety of statistical and image processing techniques based on computing differences in image feature representations [5]. Other methods extend these basic spatiotemporal models in interesting ways to produce video-specific change-point detection algorithms. For example, in [27], the authors introduce a Bayesian method to segment videos containing specific scenes into clusters in an online, unsupervised way accompanied by confidence probabilities. The method is applied to robotics. In [37], the authors extend a statistical change-point detection algorithm to video in order to track 3D objects. In recent deep learning literature, the authors in [14] propose a convolutional network with a sliding frame window input capable of creating spatiotemporal features to classify videos.

The change-point detection literature informs part of our approach as well. Classic statistical methods range from simple sum- and mean-based thresholding algorithms for single change-point detection in offline

[†]Department of Mathematics, State University of New York at Geneseo, stephanie.a.allen95@gmail.com
[‡]Department of Computer Science, University of Toronto, madras@cs.toronto.edu
[§]Department of Mathematics, University of California, Los Angeles, yeye@ucla.edu
[¶]Department of Mathematics/Computer Science, DePaul University, greg.zanotti@gmail.com

[*]Names ordered alphabetically to signify equal contribution to the work by all authors. We would also like to thank our mentor, Dr. Giang Tran, for all of her help. Her email address is: gtran@math.utexas.edu

data [6], to nonparametric tests for changes in distributions [44]. Other statistical methods use Bayesian priors to incorporate time-dependent information into the probability of a change-point occurring [2].

In this paper, we present a novel two-stage framework (summarized in Figure 1) for video change-point detection which draws on methods from machine learning, computer vision, and change-point detection. We begin with a video data set with ground truths — the time at which changes between the two predefined states occur. These states are mutually exclusive and collectively exhaustive, and we refer to them as positive and negative states. Then, we selectively extract frames from each video to create a time series of frames. In the first stage, we utilize feature extraction and image representation methods to generate a compact representation of each frame and then label each representation via classifiers — support vector machine (SVM) and convolutional neural network (CNN) — and we ultimately construct a time series of scores. These scores measure the confidence of a classifier about whether a video frame corresponds to the positive state. In addition, by setting a threshold, we are able to convert these scores into binary labels (0,1) corresponding to positive and negative states. Finally, change-point detection algorithms analyze the scores or labels to identify salient changes between the two states of interest, thereby locating the times at which change-points occur. This modular format enables generalization to a variety of change-point classes.

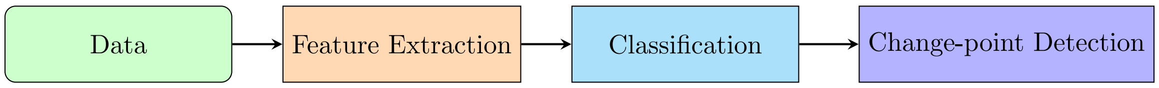


Figure 1: Our framework's workflow for video change-point detection.

The paper is organized as follows. Section 2 presents construction of video representation and classification approaches, turning to the computer vision literature using feature detection methods, SVMs, and CNNs Change-point detection methods are presented in Section 3, utilizing mean squared-error minimization, forecasting methods, hidden Markov models, and maximum likelihood estimation. Finally, we perform an experiment on a body-worn video data set provided by the Los Angeles Police Department (LAPD). We parameterized our framework to detect changes from in-car scenes to out-of-car scenes, and we achieve promising results, which are presented in Sections 4.3 and 4.4.

2 Video Preprocessing and Frame Classification

A video can be regarded as a sequence of frames. In a preprocessing step, we sample frames from videos and save them as JPEG images. The goal is to classify these frames as either one of the two states — the states between which we wish to identify change-points. We frame this problem as one of scene classification. Scene classification has been extensively studied by the computer vision community; consequently we use methods from computer vision to classify scenes. Current state of the art approaches use either keypoint detection and the Bag-of-Visual-Words (BoVW) technique with a classifier (such as an SVM) capable of comparing the histograms it produces [43], or a convolutional neural network (CNN) on raw pixel values [17]. To extend the SVM, we propose a novel technique for constructing a histogram, which improves classification accuracy. We also modify the architecture of a pre-trained CNN to create a CNN capable of two-state video frame classification. The details are described in the following sections.

2.1 Keypoint Detection and Support Vector Machine

Intuitively, keypoints are distinctive image features. After a keypoint is located by a keypoint detector, image features in the keypoint's neighborhood can be described by a keypoint descriptor. We use scale-invariant feature transform (SIFT) [20] for keypoint detection and description, because SIFT features are shown to be invariant to image scale and rotation, and it is partially invariant to changes in illumination. The major steps of constructing SIFT can be summarized as:

- Keypoint detection and localization
    - Apply Gaussian filters with different standard deviations to the input frame

  - In the differences of Gaussians, search for local extrema in scale and space. These extrema are potential keypoints.

- Orientation assignment
  - Assign one or more orientations to each keypoint based on the directions of pixel gradients in the keypoint's neighborhood

- Keypoint description
  - Compute gradients of pixels relative to the keypoint orientation in the 16-by-16 neighborhood around each keypoint
  - Divide this 16-by-16 patch into blocks of 4-by-4 in size and create an 8-bin orientation histogram for each block
  - Concatenate histograms to get a 128 dimensional descriptor for each keypoint

Using this approach, each frame can be represented as a SIFT matrix, with each row being a 128-dimenional SIFT descriptor. However, since the number of SIFT descriptors extracted varies among frames and an SVM requires inputs to have the same dimension, we use BoVW as an additional step to construct image representations.

Bag-of-Visual-Words and Vector Quantization

After extracting SIFT features from all video frames of interest, we take 20% of frames of the two states in the training set and apply k-means clustering *separately* on their feature vectors, with each state having K clusters. After centroids of clusters are computed, we assign each feature vector from frames in testing set and the remaining part of the training set to its closet centroids based on Euclidean distance. This general technique is called BoVW, and the number of clusters K is often referred to as the size of vocabulary. BoVW is an example of vector quantization (VQ) in computer vision. After VQ, a feature vector is represented by the indexes of clusters to which it is assigned [16]. In general, there are two distinct forms of VQ: *hard* VQ and *soft* VQ. In hard VQ, a feature vector is assigned to exactly one cluster, which corresponds to the closest centroid; whereas in soft VQ, a feature vector can be assigned to more than one clusters [40]. In our work, a feature vector's membership at each cluster depends on the feature vector's distance to the corresponding centroid. We propose the following technique to perform soft VQ.

Let $\{c_j\}_{j=1}^{C}$ be a set of centroids computed in the clustering stage, where $c_j \in \mathbb{R}^{128}$ and $C = 2K$ is the total number of centroids. Let $\{f_i\}_{i=1}^{F}$ denote the set of all SIFT feature vectors extracted from a frame, where $f_i \in \mathbb{R}^{128}$. The goal is to construct $H \in \mathbb{R}^{C}$, where H_j measures the effective number of feature vectors assigned to cluster j for $1 \leq j \leq C$. For each feature vector f_i, we compute its Euclidean distance D_{ij} to each of the centroids c_j. Then, the *relative* distance R_{ij} between centroid c_j and feature vector f_i can be defined as

$$R_{ij} = \frac{D_{ij} - \min\limits_{1 \leq p \leq C}(D_{ip})}{\max\limits_{1 \leq p \leq C}(D_{ip}) - \min\limits_{1 \leq p \leq C}(D_{ip})},$$

where the centroid closest to the f_i has relative distance 0 whereas the farthest centroid gets relative distance 1. To control the contribution of f_i to clusters whose corresponding centroids are not the closest to f_i, a parameter $E \in \mathbb{R}$ is introduced. We then define the exponentially decayed relative distance R'_{ij} as $R'_{ij} = \exp(-ER_{ij})$ so that we essentially recover hard VQ as E approaches positive infinity. The contribution of f_i to H is then normalized to 1, and so every feature vector has the same weight. This procedure is summarized in Algorithm 1 below.

Note that the idea of VQ is closely related to construction of a histogram: Assigning a vector to clusters essentially achieves the same effect as incrementing the counts at the corresponding histogram bins, except that in the later case bin counts are discrete. For notational convenience, we refer to soft VQ and soft histogram interchangeably, and H is called "BoVW histogram" in subsequent sections.

Note that conventional BoVW and VQ do not consider spatial information of keypoints. In other words, the locations of objects within images are not taken into consideration. Previous literature [43] suggests

Algorithm 1: Soft VQ

```
Inputs : set of centroids $\{c_j\}_{j=1}^{C}$,
         set of feature vectors $\{f_i\}_{i=1}^{F}$ from an frame,
         a positive constant $E$
Output: $H \in \mathbb{R}^C$
1 For $i = 1 : F$
2     For $j = 1 : C$
3         $D_{ij} \leftarrow ||f_i - c_j||_2$
4     For $j = 1 : C$
5         $R_{ij} \leftarrow \frac{D_{ij} - \min\limits_{1 \le p \le C}(D_{ip})}{\max\limits_{1 \le p \le C}(D_{ip}) - \min\limits_{1 \le p \le C}(D_{ip})}$
6         $R'_{ij} \leftarrow \exp(-ER_{ij})$
7     For $j = 1 : C$
8         $H_j \leftarrow H_j + \frac{R'_{ij}}{\sum\limits_{p=1}^{C} R'_{ip}}$
```

that for a small size of visual vocabulary, including spatial information can improve classifiers' performances significantly, while for a large size of visual vocabulary, the improvements are not substantial.

Support Vector Machine with Pyramid Match Kernel

To evaluate the effect of including spatial information of keypoints, we experiment with pyramid match kernel [18], which partitions an input image into increasingly fine spatial bins. At level l, 2^l cells are placed on each side of an image, so there are 4^l spatial bins with equal size. No partition occurs at level 0. By setting parameter L, which is the maximum number of levels, we are able to control how much detailed spatial information are included. For example, if L is set to 0, no spatial information of keypoints will be considered.

After VQ, we can represent a feature vector by a histogram of cluster membership, where each histogram bin measures the similarity between the feature vector and the corresponding centroid. For each spatial bin, we create an aggregated histogram by summing up histograms corresponding to feature vectors falling in this bin. These aggregated histograms are then weighted according to the level at which the spatial bin is located. Because matches of features at finer spatial resolutions are expected to yield more information about the similarity between two images, histograms at finer grids are weighted more heavily. We follow the practice in [18] and give weights 1/4, 1/4, and 1/2 to levels 0, 1, and 2 respectively. In the final step, we concatenate these weighted aggregated histograms from all spatial bins, and an input image is represented by a vector of a fixed length. This vector is later input into SVM.

A two-class SVM works by finding the optimal hyperplane that gives the maximum separation between training examples from the two classes. This goal can be achieved by solving an optimization problem, which maximizes the two-class separation while penalizing training examples lying on wrong sides of margins. We apply a kernel SVM, which first maps training examples to a higher dimensional feature space before optimizing for the maximum separation. A kernel function takes two training examples and measures their similarity. In our project, the kernel function corresponds to the pyramid match kernel [18], which sums up the intersections between two histogram created using the method described above.

2.2 Convolutional Neural Network

In the second classification approach, we use deep neural networks. Deep neural networks are machine learning algorithms that jointly learn a feature representation and discriminative classifier over a data set [10]. Nonlinear computational nodes called *neurons* are stacked on top of one another in layers to form complex, richly informative sets of features that have highly discriminative characteristics. Neural networks are trained by changing weights, thresholds, and other parameters, generally through the use of an iterative optimization algorithm like stochastic gradient descent [10, 29]. An overview of the historical development of neural networks and deep learning can be found in [31].

Convolutional neural networks (sometimes referred to as "ConvNets", "convolutional networks", or "CNNs") have their origins in the study of the visual cortex in primates [11]. ConvNets were popularized in [19], although earlier forerunners such as [9] contributed to their development. ConvNets extract information from input data using overlapping convolutions. Each convolution operation consists of "sliding" a feature detector over input data, which generates an output of similar dimensionality. Each feature detector looks for one specific feature, and is made up of a number of trainable weights. Features are dependent on data; for example, image features may include edges, color blobs, or simple shapes. Multiple convolutions are performed in a single *convolutional layer*, and the output of a convolutional layer is transformed by nonlinear activation functions. Convolutional layers are stacked and interspersed with *pooling layers*, which subsample their input (e.g. by taking averages over various input sections) and produce an output with lower dimension.

A convolutional layer is made up of several different feature maps, which take the form of tensors, in the sense that they are (small) multi-dimensional arrays of numbers. The feature maps require this definition because the two-dimensional input image requires the first convolutional layer to contain two-dimensional feature maps. Because there are multiple feature maps in the first layer, the output matrices are concatenated to form output tensors—three-dimensional arrays that are convolved with the feature maps in the next convolutional layers. This structure means that convolutional networks pass tensors in between their intermediate layers.

At the last pooling or convolutional layer, the produced activation tensor is generally flattened into a single vector, and connected to a fully connected layer. This fully connected layer is followed by one or two more fully connected layers, and an output layer. In the case of binary classification, we measure the error of the output layer activation (or "score") using the *hinge loss* function. Hinge loss is defined as $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$, where y is the true label and $\hat{y}$ is the predicted score (which are both scalars). The combination of convolutional layers, which create high-quality, deep feature representations of images, and fully connected neural networks, which are excellent classifiers, make convolutional neural networks very effective at most computer vision tasks.

2.3 Pre-trained Networks

Unfortunately, deep neural networks take large amounts of time and computational power to train. One way to bypass this problem is to re-use a popular, well-known network configuration for which trained weights already exist. These pre-trained neural networks are created by neural network researchers, and consist of a known architecture (e.g. a number of convolutional layers, specific sizes of feature maps, etc.), and a file containing the numbers for each weight in the network.

Pre-trained networks have been released for ILSVRC, a competition in which participants aim to classify images into one of 1,000 classes. It has been found that the weights and structure of these networks provide excellent starting points for classifying images into a different set of classes. For example, in [25], it is reported that simply removing the output layer of a popular pre-trained ConvNet and replacing it with a new output layer trained to detect a different set of classes provides state of the art accuracy on several computer vision problems.

VGG-16 Architecture

The pre-trained VGG-16 network of [32] was initially conceived and publicly released in 2014. It was a top-performing model in the 2014 ILSVRC, and has been used widely in the literature to achieve excellent results on image classification problems. The original training process of the VGG-16 network can be found in [32]. The main reason we use the VGG-16 convolutional network is for its very deep architecture, which is what allowed it to perform so well in the 2014 ILSVRC competition.

Adapting VGG-16

Although VGG-16 ends with a 1,000-dimensional output layer, this layer can be removed and replaced with a layer made for a binary classification task such as detecting whether a scene is classified into one state or another. To facilitate this, the weights were downloaded from the authors' website, and the network was implemented using the machine learning software libraries mentioned in Section 4. Because the weights of the last fully connected layer are often tuned to the task of the next (output) layer [25], we removed this layer as well as the output layer. We replaced the two layers with a single output layer that uses the hinge loss function. This new layer, once the weights are trained, produces a univariate scalar score for each frame that conveys the positive/negative state label for the frame. The use of a hinge loss function is reported in [35] to produce excellent results on different classification problems. In addition to using hinge loss, we regularize the weights of the output layer using elastic net regularization, which is defined as adding a penalty term to the loss function $\ell(y, \hat{y})$, such that the function minimized becomes $\ell(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y}) + \lambda\big[\alpha\|w\|_1 + (1-\alpha)\|w\|_2\big]$, where w is the set of weights, $\|\cdot\|_p$ is the L_p norm, λ is a penalty importance parameter, and $0 \leq \alpha \leq 1$.

To train this network, we first froze the weights of the non-modified layers so that they would not be changed. Training then proceeded using mini-batch stochastic gradient descent. Detailed information on training and results will be discussed in Section 4.

3 Change-Point Detection

To recap our framework, we sample every n-th frame of a video and apply our classifier to each frame to distinguish between our states. This gives us a series of confidence scores. We then seek to find change-points in this series — points at which the frames switch between our two states of interest. Due to the modular nature of our framework, we were able to explore several approaches to the problem of change-point detection. All of these methods were coded in MATLAB, and we list any specific packages used in the relevant sections below.

3.1 Change-Point Methods Overview

Given a time series $X_i, i = 1...n$, we define a change-point c as a place in the series where the underlying distribution of the X_i changes. That is, in the case of one change-point:

$$X_i \sim F_1 \;\forall\; i \leq c,\; X_i \sim F_2 \;\forall\; i \; > c$$

for some distributions $F_1 \neq F_2, c \in \{1...n\}$. In the context of this problem, the distributions F_1, F_2 are over all frames representing our two states of interest. There may be zero, one, or multiple change-points in a given series of scores.

In this section, we discuss a variety of approaches to change-point detection. They are:

- Mean squared-error minimization, for which we derive a distribution for its central statistic
- Forecasting methods, which can be easily adapted to online change-point detection
- Hidden Markov models and maximum likelihood estimation, which provide state labels for each frame

3.2 Mean Squared Error minimization (MSE)

We will first outline how this method (inspired by [36]) works for sequences with one change-point, then show how we extend it to address the possibility of multiple change-points.

In a single change-point sequence, we attempt to optimally describe the sequence by using two constant functions. That is, we find the optimal point to split the sequence such that the the two halves of the sequence cluster closely around their sample means. Formally:

$$MSE(c) = \sum_{i=1}^{c} \big(X_i - \bar{X}_1\big)^2 + \sum_{i=c+1}^{n} \big(X_i - \bar{X}_2\big)^2,$$

where

$$\bar{X}_1 = \frac{1}{c}\sum_{i=1}^{c} X_i,\ \bar{X}_2 = \frac{1}{n-c}\sum_{i=c+1}^{n} X_i,$$

We are finding the total squared error from the two sample means. In the univariate case, we can reduce our expression to the following:

$$MSE(c) = \sum_{i=1}^{n} X_i^2 + c\bar{X}_1^2 + (n-c)\bar{X}_2^2.$$

We now wish to create a hypothesis test to determine if a measurement of MSE at a given c is significantly small enough to represent a change-point. Let H_0 be that X_i does not have a change-point. We can reject this null hypothesis, and declare H_1 to be true (i.e., a change-point exists), if the p-value for $MSE(c)$ is below some significance threshold α. The p-value calculations are given below.

Under H_0, we assume all X_i are independently and identically distributed according to some distribution X. By the central limit theorem, we can take sample means $\bar{X}_1$ and $\bar{X}_2$ to be normally distributed for large enough sample size, i.e. $\bar{X}_1 \sim \mathcal{N}(\mu_X, \frac{\sigma_X^2}{c})$, $\bar{X}_2 \sim \mathcal{N}(\mu_X, \frac{\sigma_X^2}{n-c})$, where μ_X and σ_X^2 are the mean and variance of the X, respectively. Without loss of generality, let us assume $\mu_X = 0$. Then, the squared normal variable $\bar{X}_1^2$ is from the gamma distribution $\Gamma(\frac{1}{2}, \frac{2\sigma_X^2}{c})$, and also $\bar{X}_2^2$ is from $\Gamma(\frac{1}{2}, \frac{2\sigma_X^2}{n-c})$. So by the properties of the gamma distribution, we have $c\bar{X}_1^2 \sim \Gamma(\frac{1}{2}, \frac{2c\sigma_X^2}{c}) = \Gamma(\frac{1}{2}, 2\sigma_X^2)$ and $(n-c)\bar{X}_2^2 \sim \Gamma(\frac{1}{2}, \frac{2(n-c)\sigma_X^2}{n-c}) = \Gamma(\frac{1}{2}, 2\sigma_X^2)$. Therefore, $c\bar{X}_1^2 + (n-c)\bar{X}_2^2 \sim \Gamma(1, 2\sigma_X^2)$. Let us call this variable G_c. We then have

$$MSE(c) = \sum_{i=1}^{n} X_i^2 + G_c$$

$$MSE(c) - \sum_{i=1}^{n} X_i^2 = G_c \sim \Gamma(1, 2\sigma_X^2).$$

Since $\sum\limits_{i=1}^{n} X_i^2$ is a constant for each sequence, we can now calculate a p-value for $MSE(c)$, using the cumulative distribution function (CDF) for $\Gamma(1, 2\sigma_X^2)$. The CDF is as follows:

$$CDF(X) = \frac{1}{\Gamma(k)}\gamma(k, \frac{X}{\theta}) \quad = \frac{1}{\Gamma(1)}\gamma(1, \frac{X}{2\sigma_X^2}) = \frac{1}{1}\int_0^{\frac{X}{2\sigma_X^2}} t^0 e^{-t} dt \quad = -e^{-\frac{X}{2\sigma_X^2}} + 1$$

where $\Gamma(k)$ is the gamma function, and $\gamma(s,x)$ is the lower incomplete gamma function. Therefore, the p-value of X is $1 - CDF(X) = e^{-\frac{X}{2\sigma_X^2}}$. Now, we can reject this null hypothesis if $p = e^{-\frac{X}{2\sigma_X^2}} < \alpha$, for significance level α. When testing every point in a sequence, we can use a Bonferroni correction, with a new significance level of $\frac{\alpha}{n}$.

To find a single change-point, we find $MSE(c)$ for all c, and pick the one with the lowest p-value. If that p-value is below our threshold, that is the change-point; otherwise, we declare the sequence to be change-point free.

We can then recursively extend this method to find multiple change-points in a sequence, if they exist. We first find a single change-point — as described above. If that change-point is deemed significant, we recursively test the intervals on each side of the change-point for another change-point. When an interval is deemed to not have a significant change-point, the algorithm stops.

3.3 Forecasting Methods

Forecasting methods allow us to fit a model to a set of data and then predict future observations using this model. To take advantage of the power of forecasting in the change-point detection setting, we develop what we will call the "future window technique" (inspired by [34]) and combine it with univariate and multivariate modeling methods to detect change-points in the time series of frames.

To employ the future window technique, we establish an initial model (or "baseline model") based on a set number of observations in the beginning of a time series — assuming that a change will not occur within the first few observations of the time series. To find potential changes, we use the baseline model to predict the next observation in the series and compare this prediction against a set number of future observations — call this the "future window." Comparing this prediction to multiple observations in the future allows us to see if the series deviates from the established model for a significant amount of time, which reduces instances of false positives created by outliers. The number of observations in the window can be changed depending upon the desire of the user to either minimize false positives or false negatives. If the differences between the prediction and each observed value in the window are all greater than some threshold — whose determination differs from method to method — then we call the value at the beginning of this window of observations a change-point. If a change-point is established, we reestimate the baseline model using the observations in the future window [34].

The process outlined above repeats for every point in the time series, except for the last few where it would have been impossible to take a full future window into account. This methodology of estimating future observations based on a current model lines up with the framing of the change-point problem, which assumes that there is a shift in the model after a change-point, and the methodology enables the handling of cases where there are multiple change-points and cases where there are no change-points. Furthermore, with minor modifications (which are not discussed in this paper), the future window technique could handle situations where-in-which the user does not have the entire data set all at once but, instead, receives pieces of data over-time.

Univariate Forecasting Methods

For the following univariate methods, we assume that — between change-points — frames close to each other are temporally related and, thus, the SVM or CNN output for the frames is stationary [33]. Furthermore, we utilize the future window technique in conjunction with each of these univariate models; all of the values in the future window are used to re-estimate the model when a change-point is found.

We first utilize a one-lag autoregressive model, which accounts for the correlation between values in a time series by predicting the next value in the series on the basis of the previous observation. Our threshold for the future window technique is the standard deviation of the entire time series [38, 8]. Next, we combine the future window technique with a mean model, which computes the mean for a set number of observations and compares the values in the future window against this mean. We again use the standard deviation of the time series as the threshold for the future window technique [24].

Finally, we develop what we will call the "sign-change filter." Between each potential change-point identified by a univariate algorithm, this filter computes the average of the CNN or SVM scores and then finds the sign of each average. If the sign of the average does not change at a potential change-point, we eliminate the change-point from the final output. This filter significantly increases the above methods' precision.

Multivariate Forecasting Methods

The BoVW histograms provide us with a succinct representation of a frame by counting the number of key-points (in a frame) associated with each visual word. We can apply methods directly to this time series, which is an unsupervised way of approaching the change-point detection problem. We utilize histogram comparison methods in conjunction with the future window technique to accomplish this goal. We apply this methodology to the raw BoVW histograms and to condensed representations.

We produce condensed representations by employing the agglomerate clustering algorithm to group the BoVW centroids that are within close proximity to each other [12]. For our specific application, we apply the algorithm to the first one-hundred centroids, which represent features corresponding to the negative state, and then separately to the second set of one-hundred, which represent features corresponding to the positive state. After constructing the cluster tree, we choose the clusters of visual words such that we simplify the histograms without significantly reducing their informational content, and we achieve this by choosing an inconsistency coefficient cutoff. For each histogram, we use the our chosen clusters to aggregate the key points into new "bins" [21].

To handle these multivariate representations, we utilize the chi-squared goodness-of-fit test and the match distance in conjunction with the future window technique. For each of these histogram comparison methods, we set the first observation in the series as the "baseline model" and, when we find a change-point, we set the new "baseline model" as the histogram in the beginning of the future window.

The chi-squared goodness-of-fit test computes the squared differences between the bins of two histograms (call one the "observed" and one the "expected") and, for each bin, divides the squared difference by the number of elements in the "expected" histogram's bin, as demonstrated below

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i},$$

where o_i is the number of elements in the i-th bin of the "observed" histogram, e_i is the number of elements in the i-th bin of the "expected" histogram, k is the number of bins, and $k-1$ is the degrees of freedom. A p-value is computed for the resulting chi-squared value and compared to an alpha value, with the null hypothesis stating that the two histograms are similar and rejection of the null hypothesis indicating they are not. The expected histogram is the baseline histogram, and the observed histograms are the histograms in the future window. The threshold for the future window technique is the alpha level for the test so, if the p-values associated with the chi-squared values of the histograms in the future window are all less than alpha, we declare a change-point at the beginning of the window [42, 28].

The match distance finds the cumulative sum for each of two histograms, finds the absolute difference between the two sequences of partial sums, and then sums this resulting sequence. This can be summarized by the equation:

$$d_M(H, K) = \sum_{i=1}^{n} |h_i - k_i|,$$

where n denotes the number of bins, h_i is the cumulative sum of the elements of h up until and including bin i, and k_i is the cumulative sum of the elements of k up until and including bin i [28]. We find the match distance between the baseline histogram and each of the histograms in the future window. To set our threshold for the future window technique, for each feature/bin, we find the mean of the differences between successive observations in the series; we then sum these means and multiply this value by a constant. Both of these histogram methods were applied to the raw histograms and the condensed histograms.

3.4 Hidden Markov Model

In a hidden Markov model (HMM), the system being modeled is a sequence of discrete latent states, which, in our project, are the ground truths of whether the frames corresponds to the positive state. Such sequence is modeled using a Markov chain, in which the conditional probability of the future state only depends on the present state. Each type of state has an associated emission probability, according to which an output is assumed to be generated. While each latent state is not directly observable, the associated output is observable. Our goal is to construct the most probable sequence of latent states, given the sequence of associated classifier scores.

Let $Z = \{z_n\}_{n=1}^{N}$ be a sequence of latent states, where $z_n = [1 \quad 0]^T$ if the frame corresponds to the negative state and $z_n = [0 \quad 1]^T$ otherwise. Let $O = \{o_n\}_{n=1}^{N}$ be the associated sequence of classifier scores. The initial distribution $p(z_1)$ is given by $\pi = [\pi_1 \quad \pi_2]$, so that

$$p(z_1) = \begin{cases} \pi_1 & \text{if} \quad z_1 = [1 \quad 0]^T, \\ \pi_2 & \text{otherwise.} \end{cases}$$

The transition matrix of latent states is denoted as A, where $A_{ij} = p(z_{n,j} = 1 | z_{n-1,i} = 1)$ and $i, j \in \{1, 2\}$. We model the conditional distributions of observed variables using Gaussian distribution:

$$p(o_n | z_n, \Phi) = \left(\frac{1}{\sqrt{2\pi\sigma_1^2}} \exp(-\frac{1}{2\sigma_1^2}(o_n - \mu_1)^2) \right)^{z_{n,1}} \left(\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp(-\frac{1}{2\sigma_2^2}(o_n - \mu_2)^2) \right)^{z_{n,2}},$$

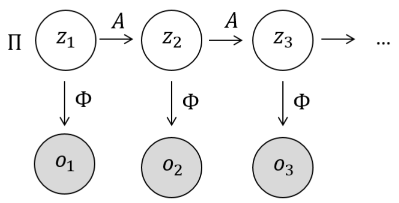


Figure 2: Structure of HMM

where $\Phi = \{\sigma_1, \sigma_2, \mu_1, \mu_2\}$ is the set of emission parameters. The structure of HMM is illustrated in Figure 2.

Following [4], the likelihood function of the HMM can be written as

$$p(O|Z) = \sum_{Z} p(O, Z|\Theta),$$

where $\Theta = \{\sigma_1, \sigma_2, \mu_1, \mu_2, A, \pi\}$ is the set of parameters to be estimated. Since the number of terms in the likelihood function grows exponentially with N, which is the number of the observations in the sequence of classifier scores, parameters in a HMM are often estimated using Expectation-Maximization (EM) algorithm and the Baum-Welch algorithm [3]. In the E step of EM algorithm, the posterior probability $p(Z|O, \Theta')$ is evaluated, where Θ' is the set of current parameter estimates. In the M step, we maximize $Q(\Theta, \Theta')$ with respect to Θ, where $Q(\Theta, \Theta')$ is the expectation of the logarithm of the complete data likelihood function and is given by

$$Q(\Theta, \Theta') = \sum_{Z} p(Z|O, \Theta') \ln p(O, Z|\Theta).$$

The algorithm iterates between the E step and M step until convergence. After parameters are estimated, the most likely sequence of latent states is inferred using the Viterbi algorithm [41].

HMM is an unsupervised method, in which labels are not required for training. In principle, one can estimate parameters and then use these estimates to infer the most probable sequence of states using the same sequence of scores. In this project, however, we are more interested in evaluating how well a trained HMM can generalize to a new video. Our training and testing data sets are designed in the following way.

Videos with at least one exit or entrance into five folds are split into five folds. A HMM model is trained on four folds, and we apply a Savitzky-Golay filter [30] on the sequences of scores corresponding to videos in the remaining folds. The filtered sequences of scores are input into the trained HMM model. We declare a change-point occurs if two adjacent latent variables are inferred to have different states. This process is repeated five times, with each fold being the testing set exactly once.

In another experiment, the goal is to estimate the precision of HMM on all videos, including those without exit or entrance. To test this method on videos without actual change-points, we apply the HMM trained on all videos that contain at least one actual change point. The results are presented in Table 3. We use the implementation of HMM included in package pmtk3 [23].

3.5 Maximum Likelihood Estimation

Solving problems through maximum likelihood estimation is a common technique in the machine learning literature. The goal in maximum likelihood estimation is to find the values of some parameters which are most likely given the data available. Here, we develop a maximum likelihood formulation of the change-point detection problem, where the parameters we are trying to find are the true state labels, L_i. Let $L_i \in \{0, 1\}$ be the ground truth labels of a series and $x_i \in \{0, 1\}, i \in 1...n$ be the labels from a classifier with accuracy p. Then, we can find the log-likelihood of a series of labels given the data as follows.

$$\begin{aligned}\log \mathcal{L}(L|X) &= \log P(X|L) \\ &= \log \prod_{i=1}^{n} P(X_i|L_i) \\ &= \log \prod_{i=1}^{n} p^{I[x_i=L_i]}(1-p)^{I[x_i \neq L_i]} \\ &= \sum_{i=1}^{n} I[x_i = L_i]\log(p) + I[x_i \neq L_i]\log(1-p) \\ &= \log(p)\sum_{i=1}^{n} I[x_i = L_i] + \log(1-p)\sum_{i=1}^{n} I[x_i \neq L_i]\end{aligned}$$

Using integer programming (IP), we maximize this quantity. We used CVX for MATLAB to solve the IP. Note that the log-likelihood here is very similar to the one in the HMM section in 3.4, except the HMM model uses Gaussian distributions to model continuous scores, and this model uses Bernoulli distributions to model discrete labels. The essential addition in the IP formulation is a constraint M on the number of change-points allowable; otherwise, the algorithm will not be robust to any sort of noise. The IP optimizes over values of L as follows:

$$\begin{aligned} \text{maximize} \quad & \log(p)\sum_{i=1}^{n} I[x_i = L_i] + \log(1-p)\sum_{i=1}^{n} I[x_i \neq L_i] \\ \text{subject to} \quad & \sum_{i=1}^{n-1} |L_i - L_{i+1}| < \ M \\ & L_i \in \{0,1\}\end{aligned}$$

The expression to maximize is the likelihood, the first constraint limits the number of change points, and the second constraint ensures that there are only two labels, one for each state. The results can be found in Table 2 and Table 3.

4 Experimental Results

We now present the results of applying our framework to a data set provided by the LAPD. We define change-points in this data set as the places where an officer exited or entered a vehicle. Our two states of interest are inside and outside a vehicle, and being outside of a car corresponds to the positive state in our framework. These change-points are important because police-public interactions often occur when officers are outside of their vehicles.

4.1 Data Set Description

Our data is provided by LAPD, from their BWV pilot program in Los Angeles' Central Division in 2014-2015. The body-worn videos were recorded using cameras that have roughly a 130° field-of-view, a resolution of 640x480, and a fisheye lens. All videos are from the officer's point-of-view, as body cameras are mounted on officers' chests.

There are 691 videos in our data, with an average length of 9 minutes. 420 of these videos contain at least one change-point of interest (either a vehicle entrance or exit), up to a maximum of 11. Of these videos, 270 of them begin from the driver's side, 176 are during the nighttime, and in 274 of them, the vehicle is moving at some point during the video. In addition, some videos contain occasional camera field-of-view occlusions from the officers' hands, arms, or clothing. The overall effect is that this data set is highly varied, and it presents many of the challenges that one might expect from real-world video data — unclear images, rapid camera movement, extreme luminance and contrast differences, etc.

4.2 Training and Testing Sets Description

Since SVM is more sensitive to redundancy in training set, we prepare different training and testing sets for SVM and CNN. SVM is sensitive to redundancy in training set because the learned decision boundary may be shifted in response to aggregated penalties imposed by repeated examples that lie on wrong side of margin. This poses a challenge to our project: as content of consecutive video frames are often highly correlated, these frames' representations are expected to be quite similar. We therefore manually select video frames that go into a data set for training and testing SVM to reduce the impact of redundancy in video data. Out of the 420 videos that contain at least one entrance or exit of a car, we take 200 of them and then randomly assign these 200 videos into 10 folds. For each of these selected videos, we select 'in-car" and "out-car" frames, and the resulting data set has 515 "in-car" frames and 529 "out-car" frames. In a trial, a SVM is trained using nine folds and tested on the remaining fold. This process is repeated ten times, and each fold is used as the testing fold exactly once. The testing accuracy on the ten trials are then averaged to give one estimate. We use the MATLAB implementation of SVM, and the VLFeat [39] implementation of SIFT feature descriptors.

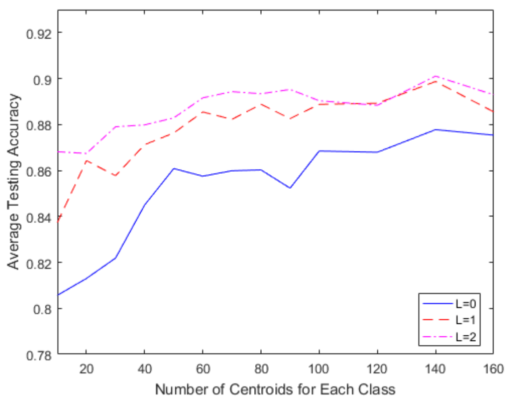
Training for the CNN proceeds by 10-fold cross-validation on the entire data set of 691 videos. First, videos are split into ten different folds. Then, from the videos, frames are extracted every one second, to form a data set of approximately 466,000 frames. No deletions or selections were made, and all frames are retained. During the training process, one fold is held out, and the CNN is trained on nine folds. Performance statistics are computed for each of the ten folds, and then averaged. Averaging is a valid way to combine performance statistics in our case, because the number of frames and in-car/out-of-car percentages are roughly equal across folds.

4.3 Classification Results

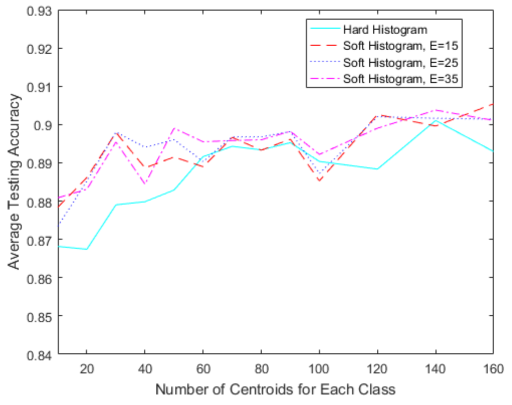
This section presents performance evaluations of our classifiers, SVM and CNN. Figure 3a plots classification accuracy of SVM with spatial pyramid match kernel and hard histogram configuration versus number of clusters. The choice of L, which determines the total number of levels, has significant impact on classifier's performance. As L increases from 0 to 1, which implies the spatial information of each keypoint is now taken into account, classifier's performance improves greatly. As L increases from 1 to 2, each frame is partitioned into finer cells, and the extra spatial information also contributes to improvements in classification accuracy. The results also show that as the size of vocabulary increases, spatial information becomes less important. This observation is consistent with results in [43]. Figure 3b compares performance of SVM with hard and soft histogram configurations. To obtain the results shown in Figure 3b, we set parameter L to 2, as we are interested in checking whether soft histogram can further improve the classifier's performance when its accuracy rate is high. K, the number of centroids for each class, is shown in the horizontal axis. As is evident from the figure, the soft VQ technique generally improves classification accuracy. For $E = 35$, SVM with soft histogram outperform that with hard histogram at every size of visual vocabulary.

For CNN, the VGG-16 convolutional network architecture is modified for generalization, and to use the hinge loss function, as described in Section 2.2. After this modification, all weights in all layers except for the last layer (the output layer) are frozen so that weight updates are not computed for them. For preprocessing, frames are then resized to 240x320, and the mean pixel value reported by the VGG-16 authors is subtracted from each color channel. The network is then trained via stochastic gradient descent with a mini-batch size equal to the size of the training set. Elastic net weight regularization, described in Section 2.2, is used with $\alpha = 0.15$ and a penalty coefficient of 0.0003. The learning rate is initialized and scheduled according to an adaptive scheme, and decreases at every epoch. The network is trained for six epochs. Results are shown in Table 1. We trained the networks on Linux Mint 18.0 workstations equipped with second-generation Intel Core i7 CPUs. Our implementation was written in Python 3.6, and we used the TensorFlow v0.9.0 [1], Keras v1.0.5 [7], scikit-learn v0.17.1 [26], SciPy v0.17.1 [13] Python libraries, as well as their dependencies.

The convolutional network results show the large improvement in performance statistics gained by using deep feature represenations of an frame (e.g. those computed by a ConvNet) as opposed to shallow feature representations (e.g. those computed by the BoVW process). We believe that more sophisticated training methods (such as jointly training a change-point detection method and CNN), or unfreezing the weights of the adapted VGG-16 network may be able to produce more accurate scene classifications.



(a)



(b)

Figure 3: Classification Accuracy of Support Vector Machine with Spatial Pyramid Match Kernel using hard histogram with different number of levels (a), and using soft and hard histogram with different values of parameter E (b).

Table 1: Classification Results

Classifier	Accuracy	Precision	Recall
Best Convolutional Neural Network	**94%**	**96%**	**95%**
Best Support Vector Machine	90%	92%	89%

4.4 Change-point Detection Results

In this section, we discuss the results of our change-point detection methods. As stated in the beginning of Section 4, we aim to identify points of vehicle entry and exit. For each video in our data set, we apply a classifier — either CNN or BoVW-SVM — to every nth frame (n = 30, 10 respectively). The classifiers output scores or class labels in {0,1}. We then run each sequence through each of five univariate change-point detection methods; for each sequence, we identify some number (possibly zero) of change-points. We also run sequences of multivariate, unsupervised frame representations through multivariate change-point detection algorithms and, for each sequence, we identify some number (possibly zero) of change-points.

To evaluate the performance of all of our change-point detection algorithms on our data set, we compare our algorithms' predicted change-points to the true change-points in the videos (where officers actually exit/enter their vehicles). We use a ten-second window of error, so a predicted change-point and a true change-point are considered equivalent if they are within ten seconds of each other. This accounts for the fact that it may take several seconds to exit or enter a vehicle. We then calculate precision and recall for each method to evaluate our performance — where recall is the percentage of actual change-points which are within ten seconds of a predicted change-point, and precision is the percentage of predicted change-points which were within ten seconds of an actual change-point. These are aggregate measurements for all of the videos, meaning we count the total number of actual change-points and predicted change-points across all videos.

This section is organized as follows. First, we apply our methods on videos that contain at least one actual change-point using outputs from the CNN. Our algorithms are then tested on the full data set which contains videos without actual change points, and the results are jointly presented. We then discuss the results of running change-point detection algorithms on SVM outputs. Finally, we present the results of our change-point detection methods on multivariate, unsupervised representations.

As mentioned in Section 4.2, we undertake cross-validation to produce scores for all 691 videos using the CNN along with a traditional neural net classifier. Table 2 shows the results of applying change-point detection algorithms on the CNN output for the 420 videos that contain at least one exit or entrance. While the five methods discussed in Section 3 give comparable recall and precision, HMM produces the highest recall of 93%, and MSE gives the highest precision of 75%.

We further test our change-point detection methods on CNN scores for the full data set containing 691 videos in total, 271 of which do not contain an entry into or an exit from a car. As shown in Table 3, recall calculations remain the same as those presented in Table 2, as these 271 videos do not contribute to the total number of actual change-points. Precision calculations, however, decrease because of false alarms.

For each of these methods, we can adjust some parameters. MSE uses a median filter window size of 30, a p-value cutoff of 0.1 with Bonferroni correction, and a maximum recursive depth of 3; it acts on the CNN binary labels. The autoregressive and mean model forecasting methods use the sample standard deviation of the series as the future window threshold, a future window of five, the first five observations to establish the baseline model, and the sign-change filter; they act on the CNN scores. MLE uses a parameter for classifier accuracy of 0.9, and a constraint on the number of allowable change-points of 10; it acts on the CNN binary labels. For HMM, the size and the polynomial order of the Savitzky-Golay filter are set to 15 and 1 respectively; this method acts on the CNN scores.

Table 2: Results of CNN Univariate Change-Point Detection on Videos with Exit or Entrance

Method	Recall	Precision
Hidden Markov Model	**93%**	72%
Mean-Squared Error Minimization	88%	**75%**
Forecasting Method – Mean Model	88%	70%
Maximum Likelihood Estimation Method	88%	67%
Forecasting Method – Autoregressive One Lag	85%	70%

Table 3: Results of CNN Univariate Change-Point Detection on all Videos

Method	Recall	Precision
Hidden Markov Model	**93%**	65%
Mean-Squared Error Minimization	88%	**68%**
Forecasting Method – Mean Model	88%	61%
Maximum Likelihood Estimation Method	88%	58%
Forecasting Method – Autoregressive One Lag	85%	60%

Table 4 presents the results of applying the change-point detection methods on the SVM scores, where SVM parameters are set to $K = 70$ and $L = 1$ for computational convenience. By comparing Table 4 with Table 2, we conclude that the precision measurements we calculate after running the change-point algorithms on SVM scores are significantly lower than the precision measurements we calculate after running the algorithms on CNN scores. Recall measurements, however, are generally comparable, except for MLE, whose recall decreases from 88% to 66%. From this piece of empirical evidence, we conclude that the performance of classifiers have a significant impact on the precision of change-point detection results.

Again, there are parameter values which we can adjust. For the SVM output, the autoregressive forecasting method uses the sample standard deviation of the series as the future window threshold, a future window of five, and the first five observations to establish the baseline model; it acts on the SVM scores. The mean model forecasting method uses the sample standard deviation of the series as the future window threshold, a future window of seven, the first ten observations to establish the baseline model, and the simple rounding filter (which rounds change-point values to the nearest thirty because of the way frames were sampled for the SVM); it acts on the SVM scores. MSE, MLE, and HMM use the same parameters as described above, but they act on SVM scores.

Table 4: Univariate Change-point Detection Results on BoVW-SVM

Method	Recall	Precision
Mean-Squared Error Minimization	**91%**	**30%**
Forecasting Method – Mean Model	96%	18%
Hidden Markov Model	90%	17%
Forecasting Method – Autoregressive One Lag	90%	17%
Maximum Likelihood Estimation Method	66%	34%

Finally, Table 5 presents the results of change-point detection using multivariate data which primarily comes from the BoVW histograms. These multivariate histograms represent the frames in their entirety, so they do not classify the frames into states — unlike the scores and labels from the SVM and CNN. Therefore, our methods may be detecting change-points in the video apart from exits from and entrances into vehicles. Consequently, these methods may have fairly low precision values because they are detecting other changes besides car exits and entrances, and our precision measurements are just concerned with the car exit and entrance change-points. It is also worth noting that, for our condensed histogram representations, we yield similar results as the full histogram results recorded in Table 4. Table 4 results are slightly better than results obtained using the condensed representations but, nevertheless, we realize that the histograms can be simplified without large losses in recall and precision.

As with our univariate methods, our multivariate methods have some parameter values which we can adjust. MSE uses the same parameters as outlined in the univariate results section. The chi-squared test uses an alpha level of 0.001, a future window of seven, and the baseline for comparison as the first histogram. The match distance uses a constant of 20 times the threshold discussed in Section 3.3, a future window of 10, the first histogram as the baseline, and the simple rounding filter (which rounds change-point estimations to the nearest thirty because of the way frames were sampled).

Table 5: Results of Multivariate Change-Point Detection on Videos with Exit or Entrance

Method	Recall	Precision
Chi-Squared Test	**100%**	**20%**
Match Distance	98%	13%
Mean-Squared Error Minimization	86%	17%

Above, we show both precision and recall for all our methods. In tuning our parameters, we prioritize recall over precision because, in a law enforcement application, we want to ensure to the best of our ability that we do not miss any important events. Our methods mostly achieve 85-90% recall (MLE's recall is lower on SVM scores because it has fewer parameters to optimize, so we have less flexibility in choosing how we would like to manage the precision-recall trade-off.). The methods run on SVM scores yield a 15-35% precision, and the methods run on CNN scores yield a 58-68% precision. It is interesting to note the large discrepancy in change-point detection results for our different classification methods, despite a relatively small discrepancy (roughly 5%) in their classification accuracy. It seems that a small improvement in classification performance can cause a large increase in precision of change-point detection. Finally, for the CNN results in particular, we see that many of the methods yield quite similar recall and precision values. This suggests there are multiple ways of approaching the change-point detection problem — enabling a user to choose a method based on additional considerations such as algorithmic speed.

5 Conclusion

In this paper, we present a novel framework for change-point detection in video, using concepts from machine learning, image recognition, and change-point detection. We outline our methods for classification at the frame level, including CNNs and feature extraction techniques. We then describe methods from four approaches to change-point detection: mean square error minimization, forecasting, hidden Markov models, and maximum likelihood estimation. We present the performance of these methods on classifier output and on BoVW histogram representations. With our multivariate methods, we discuss the challenges of applying change-point detection methods to flexible, unsupervised, and multivariate representations.

Testing specifically for identification of vehicle entrances and exits, our methods succeed with 90% recall and nearly 70% precision on a highly complex, realistic data set provided by the LAPD. However, we believe our framework is highly adaptable to different change-point classes, both within the domain of law enforcement BWV and outside of it. For instance, with an appropriately re-labeled data set, we believe our framework would succeed comparably at identification of video segments where an officer is speaking to a member of the public, handcuffing a suspect, or engaging in a foot chase.

With this in mind, the framework presented in this paper represents a promising step toward law enforcement's long-term goal of automatic video tagging of important segments. This would make the large-scale deployment of BWV (one camera to each officer in a police force) much more feasible.

6 Acknowledgments

First, we would like to thank our academic mentor, Dr. Giang Tran, for her guidance and continuous support. Her suggestions have helped us tremendously throughout this research. We would also like to thank Sgt. Javier Macias and Dr. Jeff Brantingham as our industry mentors; they have provided us with important context for this project. This work was completed as part of the 2016 Research in Industrial Projects for Students (RIPS) program at the Institute for Pure and Applied Mathematics (IPAM) and was supported by grants from the LAPD and NSF.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] R. P. Adams and D. J. MacKay, *Bayesian online changepoint detection*, arXiv preprint arXiv:0710.3742, (2007).

[3] L. E. Baum, *An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes*, Inequalities, 3 (1972), pp. 1–8.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] P. Bouthemy, M. Gelgon, and F. Ganansia, *A unified approach to shot change detection and camera motion characterization*, IEEE Transactions on Circuits and Systems for Video Technology, 9 (1999), pp. 1030–1044.

[6] J. Chen and A. K. Gupta, *On change point detection and estimation*, Communications in statistics-simulation and computation, 30 (2001), pp. 665–697.

[7] F. Chollet, *keras*. https://github.com/fchollet/keras, 2016.

[8] J. R. Evans, *Statistics, Data Analysis, and Decision Modeling*, Prentice Hall: Pearson Education Inc., 5 ed., 2013.

[9] K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological cybernetics, 36 (1980), pp. 193–202.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Book in preparation for MIT Press, 2016.

[11] D. H. Hubel and T. N. Wiesel, *Receptive fields and functional architecture of monkey striate cortex*, The Journal of physiology, 195 (1968), pp. 215–243.

[12] Y.-G. Jiang and C.-W. Ngo, *Visual word proximity and linguistics for semantic video indexing and near-duplicate retrieval*, Computer Vision and Image Understanding, 113 (2009), pp. 405–414.

[13] E. Jones, T. Oliphant, P. Peterson, et al., *SciPy: Open source scientific tools for Python*, 2001–. [Online; accessed ¡today¿].

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, *Large-scale video classification with convolutional neural networks*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[15] C. M. Katz, D. E. Choate, J. R. Ready, and L. Nuño, *Evaluating the impact of officer worn body cameras in the phoenix police department*, Phoenix, AZ: Center for Violence Prevention and Community Safety, Arizona State University, (2014).

[16] K. Kitani, *Bag-of-visual-words, 16-385 computer vision.* http://www.cs.cmu.edu/ 16385/lectures/Lecture12.pdf.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[18] S. Lazebnik, C. Schmid, and J. Ponce, *Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories*, in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, IEEE, 2006, pp. 2169–2178.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[20] D. G. Lowe, *Distinctive image features from scale-invariant keypoints*, International journal of computer vision, 60 (2004), pp. 91–110.

[21] MathWorks, *Hierarchical clustering.* http://www.mathworks.com/help/stats/ hierarchical-clustering.html, 2016.

[22] L. Miller and J. Toliver, *Implementing a body-worn camera program: Recommendations and lessons learned*, 2014.

[23] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, The MIT Press, 2012.

[24] R. Nau, *Mean (constant) model.* http://people.duke.edu/ rnau/411mean.htm, n.d.

[25] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, *Learning and transferring mid-level image representations using convolutional neural networks*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[27] A. Ranganathan, *Pliss: Detecting and labeling places using online change-point detection*, in Proceedings of Robotics: Science and Systems, Zaragoza, Spain, June 2010.

[28] Y. Rubner, C. Tomasi, and L. J. Guibas, *The earth mover's distance as a metric for image retrieval*, International journal of computer vision, 40 (2000), pp. 99–121.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, tech. rep., DTIC Document, 1985.

[30] A. Savitzky and M. J. Golay, *Smoothing and differentiation of data by simplified least squares procedures.*, Analytical chemistry, 36 (1964), pp. 1627–1639.

[31] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117.

[32] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, CoRR, abs/1409.1556 (2014).

[33] A. H. Studenmund, *Using Econometrics: A Practical Guide*, Addison-Wesley: Pearson Education Inc., 6 ed., 2011.

[34] J.-i. Takeuchi and K. Yamanishi, *A unifying framework for detecting outliers and change points from time series*, IEEE transactions on Knowledge and Data Engineering, 18 (2006), pp. 482–492.

[35] Y. Tang, *Deep learning using linear support vector machines*, arXiv preprint arXiv:1306.0239, (2013).

[36] W. Taylor, *Change-point analysis: A powerful new tool for detecting changes.* http://www.variation.com/cpa/tech/changepoint.html, 2000.

[37] G. Tsechpenakis, D. N. Metaxas, C. Neidle, and O. Hadjiliadis, *Robust online change-point detection in video sequences.*

[38] P. S. University, *Stat 501 regression methods: Autoregressive models.* https://onlinecourses.science.psu.edu/stat501/node/358, 2016.

[39] A. Vedaldi and B. Fulkerson, *VLFeat - an open and portable library of computer vision algorithms*, in ACM International Conference on Multimedia, 2010.

[40] V. Viitaniemi and J. Laaksonen, *Spatial extensions to bag of visual words*, in Proceedings of the ACM International Conference on Image and Video Retrieval, ACM, 2009, p. 37.

[41] A. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE transactions on Information Theory, 13 (1967), pp. 260–269.

[42] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability and Statistics for Engineers and Scientists*, Pearson Education Inc., 9 ed., 2012.

[43] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, *Evaluating bag-of-visual-words representations in scene classification*, in Proceedings of the international workshop on Workshop on multimedia information retrieval, ACM, 2007, pp. 197–206.

[44] I. T. Young, *Proof without prejudice: use of the kolmogorov-smirnov test for the analysis of histograms from flow systems and other sources.*, Journal of Histochemistry & Cytochemistry, 25 (1977), pp. 935–941.