Deep Learning Approach to Trespass Detection using Video Surveillance Data

by

Muzammil Bashir

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

April 2019

APPROVED:

Professor Elke A. Rundensteiner, Thesis Advisor

Professor Jacob Whitehill, Thesis Reader

Professor Craig E. Wills, Head of Department

Abstract

While railroad trespassing is a dangerous activity with significant security and safety risks, regular patrolling of potential trespassing sites is infeasible due to exceedingly high resource demands and personnel costs. There is thus a need to design an automated trespass detection and early warning prediction tool leveraging state-of-the-art machine learning techniques. Leveraging video surveillance through security cameras, this thesis designs a novel approach called ARTS (Automated Railway Trespassing detection System) that tackles the problem of detecting trespassing activity. In particular, we adopt a CNN-based deep learning architecture (Faster-RCNN) as the core component of our solution. However, these deep learning-based methods, while effective, are known to be computationally expensive and time consuming, especially when applied to a large amount of surveillance data. Given the sparsity of railroad trespassing activity, we design a dual-stage deep learning architecture composed of an inexpensive prefiltering stage for activity detection followed by a high fidelity trespass detection stage for robust classification. The former is responsible for filtering out frames that show little to no activity, this way reducing the amount of data to be processed by the later more compute-intensive stage which adopts state-of-the-art Faster-RCNN to ensure effective classification of trespassing activity. The resulting dual-stage architecture ARTS represents a flexible solution capable of trading-off performance and computational time. We demonstrate the efficacy of our approach on a public domain surveillance dataset.

Acknowledgements

I would like to thank my advisor, professor Elke Rundensteiner for the guidance and mentorship I received from her during the last two years. Her patience and attention to details has been instrumental in making me a better researcher. I also want to thank my thesis reader, professor Jacob Whitehill, for giving useful feedback to make this thesis a success. I would also like to thank Ramoza Ahsan for her continued support and useful feedback she provided during this time. With their help, I have been able to improve my knowledge in data science in general and machine learning in particular. I would also like to thank DSRG in general, particularly Luke Buquicchio and Walter Gerych. Finally, I would like to acknowledge the generous support by the U.S. State Department for funding my master's degree under the Fulbright Scholarship program.

Contents

1 Introduction **10**

1.1 Background and motivation 10

1.2 Problem definition 12

1.3 Goals 13

1.4 Technical challenges 13

1.5 Proposed approach 15

2 Background and related work **16**

2.1 Convolutional Neural Networks (CNN) 16

2.1.1 Convolution layer 16

2.1.2 Activation layer 18

2.1.3 Pooling layer 19

2.1.4 Fully connected layer 20

2.2 Transfer learning 22

2.3 Object detection 23

2.3.1 OverFeat 23

2.3.2 RCNN 24

2.3.3 Fast-RCNN 24

2.4 Background subtraction 26

3 Methodology **27**

3.1 Problem formulation 27

3.2 Proposed pipeline 27

3.3 Stage 1 28

3.3.1 Mixture of Gaussian (MoG) 29

3.4 Stage 2 32

3.4.1 Feature extraction 33

3.4.2 Region Proposal Network (RPN) 33
3.4.3 Fast-RCNN head 38
3.4.4 Training loss 40

4 Experimental evaluation **43**

4.1 Dataset 43
4.1.1 Synthetic dataset generator 44
4.1.2 Adding noise 46
4.2 Evaluation metrics 46
4.2.1 F1 score 47
4.2.2 Area Under the Curve (AUC) 47
4.3 Time-accuracy trade-off experiment 48
4.4 Stage based experiments 53
4.5 Noise analysis experiments 55

5 Conclusion **59**

5.1 Summary 59
5.2 Future work 59

List of Figures

1 Challenges in trespassing detection system 14
2 Our approach 15
3 LeNet-5 17
4 Convolution operation 18
5 ReLU function 19
6 Pooling operation 20
7 Fully connected layer 21
8 Trespassing detection pipeline 28
9 Background subtraction model 29
10 Gaussian mixture model 31
11 Posterior probability 32
12 Faster-RCNN pipeline 34
13 Anchor illustration 35
14 RPN architecture 36
15 Anchor target assignment 37
16 Intersection over Union 38
17 Non Maximum Suppression (NMS) 38
18 RoI pooling 39
19 Fast-RCNN classifier 40
20 Experimental methodology 43
21 Samples from VIRAT 2.0 dataset 44
22 Synthetic dataset generator 44
23 Synthetic video illustration 45
24 AUC illustraction 48
25 Time-accuracy trade off - MoG vs GSoC 50
26 Time-accuracy trade off for varying AR using MoG 51

27 Time-accuracy trade off for varying AR using GSoC 52
28 Stage 1 evaluation - MoG 53
29 Stage 1 evaluation - GSoC 54
30 Stage 2 evaluation 56
31 Noise analysis - varying p 57
32 Noise analysis - varying μ 58

List of Tables

1 Relationship of number of background blocks and activity ratio . 46

2 Noise parameters 46

3 Synthetic data parameters for time-accuracy trade off 49

4 Stage 1 AUC and time analysis 53

5 Noise analysis - AUC for varying p 57

6 Noise analysis - AUC for varying μ 57

Listings

1 OverFeat pseudo code 23

2 RCNN pseudo code 24

3 Fast-RCNN pseudo code 25

1 Introduction

1.1 Background and motivation

Automated trespassing detection is an important problem that has applications ranging from railroad security to safe neighborhood. In US, 1080 people were either killed or injured as a direct result of trespassing in 2016 alone[1]. The number increased to 1224 casualties (13.3 % increase) in 2017 [2]. Recently, Worcester Police Department (WPD) conducted a four month long study of trespassing activities and found at least 150 trespassing events involving more than 200 trespassers with the average trespassing event lasting over 15 minutes. Trespassers frequently encountered either a moving or stationary train and in most cases, received little warning about the approaching train. This situation clearly poses a risk for both the train as well as the trespasser. In most cases, contact with the train proves to be fatal. Aside from human costs, these casualties, whether fatal or not, are exceeding expensive. Property damage, emergency services, safety investigations, insurance, legal and delay costs may account for hundreds of thousands up to millions of dollars per accident.

A straightforward solution to this problem is to station police officers at the potential trespassing sites round the clock. However, this option has little practicability due to the sheer overwhelming requirement of large number of trained human personnel. Another solution is to set up a surveillance network of CCTV cameras and employ human analysts to review the video feed on 27×7 basis. Video surveillance data can be transformed to infer trespassing statistics. This can be useful to determine potential trespassing sites and time for more efficient resource utilization i.e. Police officers or relevant personnel (such as social workers) can be sent to potential sites only. Though attractive,this manual approach has numerous severe downsides:

- **Limitation of human resources:** In this era of big data, we simply don't have enough human personnel. Scaling up the surveillance network doesn't mean adding more cameras, but also addition of numerous trained human analysts.
- **Subjectivity in analysis:** Even well trained humans tend to be subjective in nature. What may be considered a threat by one analyst may not be considered so by the other.
- **Unreliability:** Manual surveillance is a dull and tedious task. Over a period of time, a human analyst may lose interest and neglect penitential activities.

Due to the above mentioned reasons, bringing automation to any trespassing prevention solution is of vital importance. Trespassing detection indeed serves as the first step towards any AI-based automated solution. A reliable automated trespassing detection solution not only provides detection in a timely fashion but also allows us to develop advanced analytics by studying trespassing patterns over time. For example, analysis over a period of six months may reveal that a group of children like to play football during the evening time. Certain locations might see increased trespassing during the morning and/or evening times because people returning home from jobs may want to take a short-cut. Other locations such as underpasses and bridges may provide a preferred meeting location for drug addicts. An advanced trespassing prevention and analytics tool may use more information than just trespassing detection and study correlation patterns between demographics, weather and traffic etc. This can help in making better predictions and subsequent prevention of trespassing.

As indicated above, an automated trespassing detection system serves as the backbone for an overall AI-propelled automated prevention system. Therefore in this thesis, we shall focus on that first critical component. We aim to develop

a computer vision based detection system that takes in a stationary surveillance video as input and produces trespassing detections as output. Below we sketch a list of key advantages of our proposed system:

- **Speed:** Since, the system is fully automated we can take advantage of high performance computing to speed up video processing. Multiple surveillance videos can also be processed in parallel by using multiple compute nodes.

- **Scalibility:** More and more data can be efficiently handled by simply adding more computational resources. In most cases, this can have the additional advantage of lower running costs of the complete surveillance system.

- **Relative objectivity:** All the data is analysed by the same system i.e. data gets processed using the same *set of equations*. This adds an inherent notion of objectivity to the results w-r-t underlying *set of equations*.

- **Reproducibility:** Computers are well known for carrying out tedious tasks with reproducible results (a quality that humans lack). A computer will reliably give the same output to a given input provided the internal functionality doesn't change. This property is useful in studying errors and working towards improving them.

1.2 Problem definition

Given an input surveillance video, the problem of trespassing detection is to classify whether each frame has human trespassing activity or not. In order to keep it simple, we define trespasser as a human spotted near a railway line. Notice that anyone within the camera field of view shall be considered a trespasser by our currently proposed solution. Detecting a trespasser in a given frame is

a special form of general object detection problem where only objects of type *person* are detected.

1.3 Goals

Although in Section 1.2 we formulate the problem we tackle as classifying each frame as trespassing or not, we have a more ambitious goal. We not only want to predict the label but also want to do so in a time-efficient manner. We notice that railroad surveillance video is sparse in terms of trespassing activity. We aim to leverage this property to reduce the processing time.

Further, we postulate that the detection performance and speed (of detection) are two opposite goals. Generally, if one wishes to improve the speed, they will have to sacrifice accuracy[1] and vice versa. Therefore, we are interested in developing a flexible solution that is capable of trading-off performance versus computational time.

1.4 Technical challenges

There are several key challenges in building a computer vision based trespassing detection system. Figure 1 depicts a few of them.

- **Occlusion:** Several time trespassers may be occluded by other objects or fellow trespassers. If the occluding object is stationary, trespasser may become un-occluded later. However, it becomes a more challenging if two trespassers move side by side.
- **Low resolution:** Most of the surveillance cameras capture low resolution videos to cut down video archiving costs. Further, it is supposed to capture a large field of view. Under these situations, a trespasser is only

[1] refers to how good a detector is performing, not necessarily the metric accuracy

Figure 1: Challenges in trespassing detection system

represented by a small number of pixels in the video footage. This poses a significant challenge in detecting low-resolution and blurry trespassers.

- **Background:** If the trespasser has other types of objects in the background, this may interfere with the detection. Depending upon the extracted features, it might be hard to discriminate between background object and trespasser.

- **Hard negatives:** Hard negatives are a source of false positives. They have visual features that look like humans but are actually not human. Typical examples include pictures and posters containing humans and electricity poles.

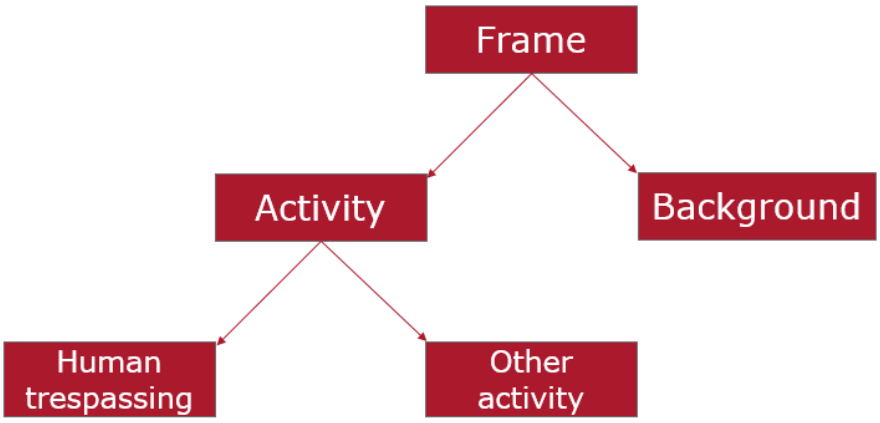

Figure 2: Our approach

1.5 Proposed approach

In order to fulfill our goals, we design a two step approach. Figure 2 depicts the overall idea of our approach. In the first step, we decide whether a particular frame has activity or not. If it turns out that the given frame has no activity, then it is classified as background frame. No further action needs to be taken for this frame. On the other hand, if it shows activity, then the next step will be to investigate whether it can be classified as human trespassing activity or not.

2 Background and related work

As the background knowledge to this work, we review Convolution Neural Networks (CNN) and transfer learning. Then we discuss the relevant literature for object detection followed by a discussion on background subtraction techniques.

2.1 Convolutional Neural Networks (CNN)

CNN based architectures[3] are the most widely used models for solving computer vision based tasks such as image classification, object detection and object segmentation. During the last few years, they have proved their effectiveness in solving many image based problems[3]. In a normal feed-forward neural network, input must be vectorized and each input feature is connected to each output feature in each layer. This results in huge number of parameters. Further, due to vectorization, the image loses its spatial structure and features computed by subsequent neural network layers cannot be mapped to image coordinates. CNN by design avoids both of these problems. It not only uses far less number of parameters than traditional feed-forward networks but it also preserves the 2D grid based structure of images[4].

Due to the fast pace research in this domain, many new architectures have been proposed. However, the convolution layer, pooling layer and fully connected layer still are the most widely used components of any CNN based architecture. Figure 3 shows the architecture of one of the earliest CNN network which employed all three of above mentioned basic components.

2.1.1 Convolution layer

As the name suggests, the convolution layer applies the operation of convolution. This operation should not be confused with convolution in other domains such as signal processing. Unlike the fully connected layer (discussed in sec. 2.1.4),

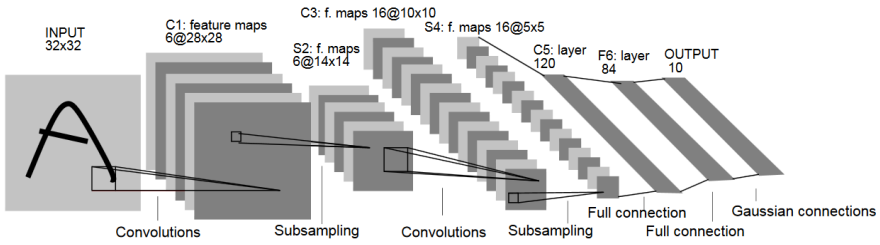

Figure 3: One of the earliest CNN "LeNet-5" used to recognize handwritten digits. Image taken from [5].

this operation can be applied to any arbitrary sized $m \times n$ matrix. Being a binary operator it accepts two parameters: input matrix I and kernel K. The operation can be defined mathematically as:

$$M(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

where I is a 2D matrix (a grayscale image) of size $m \times n$ and K is the kernel[6]. (i,j) represents the location in output M. In the computer vision literature, M is also known as feature map. Generally, the size of the kernel K is much smaller (usually 3×3) than I.

Although the definition looks complex, the convolution operation in practice is quite simple. $M(i,j)$ is simply the sum of the element-wise product of the sub-matrices of I and K. The sub-matrix of I has a center at (i,j) and is of size equal to K. Figure 4 explains the concept in a graphical manner for the location $(2,2)$.

Although the above definition defines the 2D convolution concept, it can simply be extended to 3D as well. In practical architectures, 3D convolution is used. Apart from preserving the grid-based image structure, another important aspect of CNN is parameter sharing. In the fully connected layer, each parameter of the weight matrix is used exactly once while computing the output

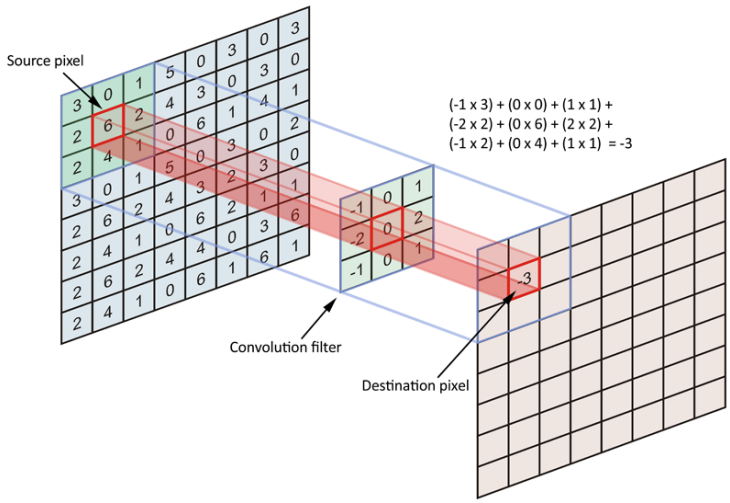

Figure 4: Convolution operation illustration: destination pixel location is sum of product of kernel weights and corresponding sub-matrix of source matrix. Image taken from [7].

feature map. On the other hand, when convolution is applied to input image, it generates a feature map which shows how strongly a particular feature occurs at a given location. This parameter sharing nature makes CNN based architecture not only practical but also robust.

2.1.2 Activation layer

A convolution layer is generally followed by a non-linear activation layer. Convolution being a linear operator can only capture linear transformations, therefore a non-linearity is fundamental to learn complicated relationships between input and output. ReLU (**Re**ctified **L**inear **U**nit) function is one of the most widely used non-linearity. It allows a positive input to pass as it is and blocks the negative input. Figure 5 shows the response of relu function. Other less commonly used non-linearities include TanH (**Tan**gent **H**yperbolic) and sigmoid functions.

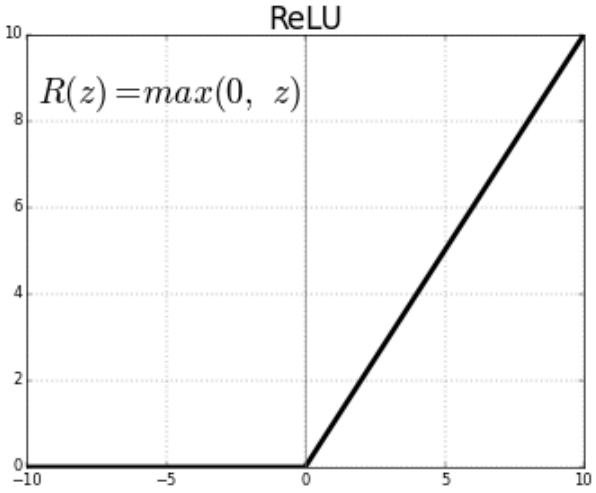

Figure 5: ReLU function

2.1.3 Pooling layer

The main goal of pooling layer is to reduce the data dimensionality. It often follows the activation layer. This layer slides a window on input feature map applying a particular pooling operation. This operation is applied on all the values inside the window and can be min, max or average. Based on the operation, it is either called min pooling, max pooling or average pooling. Apart from the window size, another important detail of pooling layer is stride. This parameter refers to number of pixels sliding window moves forward each time.

Generally stride of the pooling layer is set such that it form non-overlapping windows. For example, applying pooling with 2×2 size and a stride of 2 converts a 14×14 input feature map to 7×7. It results into non-overlapping windows as window of size 2×2 moves forward by 2 pixels every time. On the other hand, if stride is 1, then every possible window location is visited and windows shall be non-overlapping (if window size is greater than 1). Figure 6 shows an example of max and average pooling.

Pooling can also increase the robustness of feature map by making it *invari-*

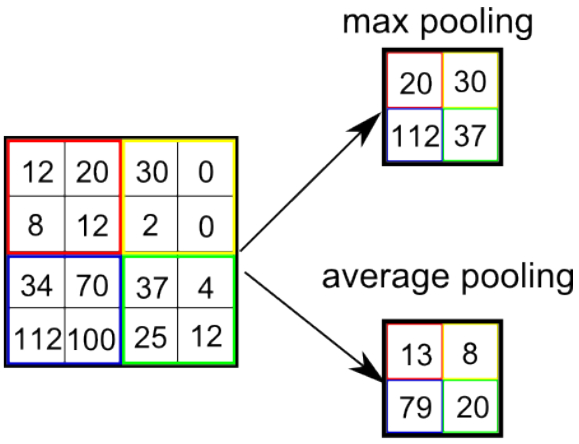

Figure 6: An example of max and average pooling applied on feature map of 4×4 size with window size of 2×2 and stride of 2. Image taken from [8].

ant to small translations[6]. Invariance to small translation means that output of pooling layer will not change much even if input experiences a slight translation. This property can be highly desirable as the same object can appear at multiple locations within different images.

2.1.4 Fully connected layer

A fully connected layer connects each element of the input feature map to each element of the output feature map. If input and output feature maps contain m and n elements respectively then a fully connected layer has $n \times m$ parameters. Figure 7 shows a simple fully connected layer.

A fully connected layer models the input-output relation as an affine linear model. If f and g represent the input and output feature map respectively, then they can be mathematically modeled as

$$g = Wf + b$$

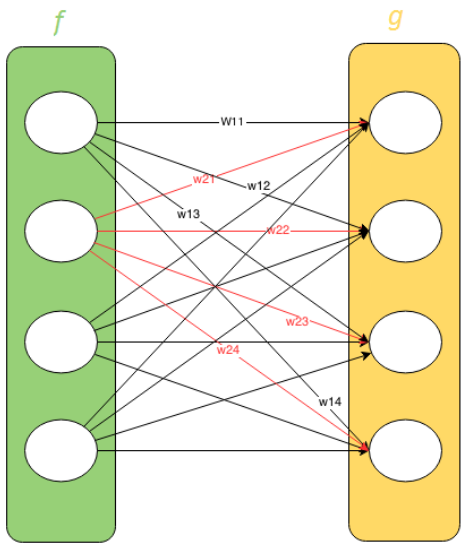

Figure 7: Fully connected layer: f represents input feature map and g represents output feature map. Each arrow Wij represents the element connecting f_i to g_j. Image taken from [9].

where W represents the weight parameter matrix and b represents the bias vector. Notice that by the definition of matrix multiplication, g_i is the inner product of i^{th} row of W and f. b_i simply adds the bias term.

This structure can lead to very powerful models, however it is not well suited for images. Even a small 256×256 image can generate a large input feature map of 65536. This means the size of W shall be $n \times 65536$ where n represents the size of output feature map. Finding the right parameters for such a large W is computationally quite expensive. Further, notice that this layer requires the input feature map f to be a flattened vector. This means that g does not have any spacial interpretation at all. These two reasons make this layer less attractive to generate features for the image data. Never the less, they are still used towards the final stage of model to classify feature maps generated by convolution and pooling layers.

2.2 Transfer learning

Transfer learning is a technique in which model weights learnt for one task using one dataset can be used to solve another task on some other dataset. Oquab et. al[10] showed that model weights (for convolutional layers) learnt for object classification using PASCAL VOC dataset[11] could be reused for action classification task. Initial layers of the model learn relatively simpler patters such as horizontal and vertical lines. Deeper layers tend to learn more complicated shapes such as blobs and corners. Learning these layers which can extract these simple features is crucial for any computer vision task. Thus instead of relearning for each task, it makes sense to simply use the pre-learnt model parameters that extract these features. Further, it should be noted that this is useful in both cases where task at hand and dataset are different. This means that weights of a model trained on ImageNet[12] for object classification can be used for object detection on COCO[13] dataset.

This technique is particularly useful if dataset is limited. In such a scenario, transfer learning is used in conjunction with fine-tuning. Fine-tuning is closely related with transfer learning. In fine tuning, instead of starting the training with random weights, we start with pre-learnt weights (transfer learning). We take a pre-trained network and chop off the fully connected layer. Now, we attach new fully connected layers and only train these layers while keeping the old convolution layer weights in frozen state. Once the network loss saturates or the model tries to over-fit, we stop the training. Now, we unfreeze the old network weights and restart training. This allows the network to adapt to new data. Since, we are not training the weights of convolution layers from scratch, this is known as fine-tuning.

2.3 Object detection

Problem of object detection refers to detecting objects of certain class (such as person, bird, vehicle etc) at a particular location in an image. Traditionally, sliding window based approach had been used where a window of particular size is slided over the image. Image patch (sub-image covered by the sliding window) is fed to a feature extractor to produce features such as Harr[14], SIFT[15] or HoG[16]. These features are then fed to a classifier such as fully connected neural network or SVM (Support Vector Machine). This classifier predicts the class labels (object category) whereas the location of sliding window is considered to be the location of predicted object.

However in recent years, significant improvement has been made by switching from hand-crafted features (such as Harr, SIFT and HoG) to CNN based features. One of the initial approaches to solve object detection problem using CNN was OverFeat[17]. Other approaches that successively built upon OverFeat include RCNN[18], Fast-RCNN[19] and Faster-RCNN[20]. We shall briefly discuss the approach followed by these methods.

2.3.1 OverFeat

OverFeat[17] was the first paper to propose the use of CNN for object detection on ImageNet[12] dataset. Their approach is simple and straight forward. They slide a window on image and for each window they compute convolutional features. These convolutional features are fed to two separate fully connected sub-networks which predict the label and bounding box for each window respectively. Though this approach significantly improves detection accuracy, it is inherently slow as sliding window generates a lot of patches. Following pseudo code explains their approach.

Listing 1: OverFeat pseudo code

```
for window in windows:
    patch = get_patch(image, window)
    features = compute_conv_features(patch)
    label = classify_label(features)
    bbox = regress_bbox(features)
```

2.3.2 RCNN

Instead of using sliding window approach Ross Girshick et al.[18] proposed to use a RoI (Region of Interest) based approach. They use selective search[21] to generate category independent RoIs. These RoIs (which are far less in number than sliding windows) are then passed through the same pipeline as Overfeat. The result is significant reduction in time as far less regions needs to be evaluated.

Listing 2: RCNN pseudo code

```
rois = apply_region_proposal(image)
for roi in rois:
    patch = get_patch(image, roi)
    features = compute_conv_features(patch)
    label = classify_label(features)
    bbox = regress_bbox(features)
```

2.3.3 Fast-RCNN

Though RCNN showed a significant speedup and improvement in accuracy as compared to OverFeat, it still applied expensive convolutional operation on each RoI independently. This makes it slow in inference as well as in training.

Most of the these RoIs are overlapping and thus computational resources are wasted during re-computation of overlapping RoIs. Fast-RCNN[19] circumvents this issues by sharing convolutional features. Convolutional feature extraction process is taken out of the loop and features are computed for the complete image in single step. Later on, features corresponding to each ROI are extracted from the pre-computed feature map. This technique makes Fast-RCNN 10x faster than RCNN in training and 150x faster in inference.

One important detail in Fast-RCNN is how we handle RoIs of different sizes. Each RoI has feature size corresponding to its own size. In order to feed the features to fully connected networks (for label and bounding box prediction), these features must be transformed to a particular size. Fast-RCNN proposes RoI pooling for this purpose. This is similar to Max pooling. However instead of sliding the window on feature map, the whole feature map is converted to a fixed size grid. Max pooling operation is applied on each grid cell and the result has the same size as the size of grid. Following pseudo code explains Fast R-CNN.

Listing 3: Fast-RCNN pseudo code

```
features = compute_conv_features(image)
rois = apply_region_proposal(image)
for roi in rois:
    patch_features = apply_roi_pooling(features, roi)
    label = classify_label(patch_features)
    bbox = regress_bbox(patch_features)
```

2.4 Background subtraction

Background subtraction is a technique that allows foreground in an image to be extracted. It is a fundamental component in most conventional (non-deep learning) computer vision pipelines. All the background subtraction techniques depend on some kind of background model. When a new images comes in, it is compared with the existing background model. Those pixels (or regions) which do not fit the background model well are considered to be foreground. Generally, foreground in image is closely related to motion or change.

We studied a few different background subtraction techniques. SVD[22] and RPCA[23] are two well known schemes for background modeling. Thus, they are well suited for background modeling but not for background subtraction/ foreground extraction. Furthermore, both of these techniques take an array of frames to develop the background model. They are not flexible enough to update their model to changing scenarios such as change in light. MoG (Mixture of Gaussian)[24] based background subtraction presents itself as a simple and effective method. Due to low computational cost and simplicity, it has very low frame processing time. However, it is highly susceptible to noise.

GSoC (Google Summer of Code)[25] and LSBP (Local SVD Binary Pattern) [26] are two recent background subtraction algorithms. They produce state-of-the art results on foreground segmentation datasets. However, they are extremely slow as compared to MoG. GSoC, however is relatively faster than LSBP.

3 Methodology

3.1 Problem formulation

Given an input video containing N frames, we want to produce a binary time series of same length N such that each index i predicts the labels y_i of corresponding frame f_i . Human trespassing label is assigned to positive (1) class and "other activity" label is assigned to negative class (0). Since, each prediction depends only on corresponding frame f_i, our problem boils down to determining a function D such that

$$D(f_i) = \hat{y}_i$$

This function D has parameters θ such that $D(f_i; \theta) = \hat{y}_i$. The aim is to find a θ^* such that $D(f_i; \theta) \rightarrow y_i$ where y_i is the ground truth label corresponding to f_i. The ground truth label has the following definition:

$$y = \begin{cases} 1, & \text{if } f_i \text{ has trespassing activity} \\ 0, & \text{otherwise} \end{cases}$$

We define the trespassing activity as the presence of at least one person in the frame.

3.2 Proposed pipeline

In order to tackle this problem, we propose a two-stage trespassing detection model. This model is in accordance with our approach in Section 1.5. Figure 8 shows the block diagram of our system implementing trespassing detection framework. Stage 1 of our pipeline corresponds to step 1 of approach. Likewise, stage 2 of pipeline corresponds to step 2 of approach. Input to our pipeline is a video and each frame is processed one by one. Each frame is first processed

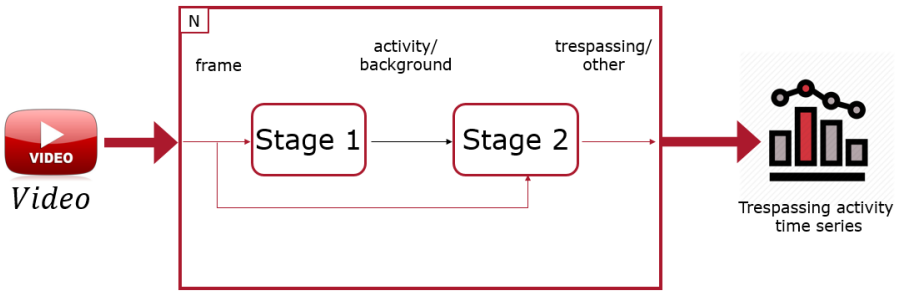

Figure 8: Trespassing detection pipeline

through stage 1 to determine if it shows activity. Only the frames classified as showing activity are further processed through stage 2. Output of the system is a time series as discussed in Section 3.1

3.3 Stage 1

As discussed above, goal of this stage is to filter non-activity frames from the activity frames. Thus, it is modeled as a background subtraction problem. Figure 9 shows a typical pipeline of background subtraction method.

Input to this stage is the given frame f_i in question. This frame f_i is first used to update the background model from b_{i-1} to b_i. Both f_i and b_i are then compared with each other by the foreground extraction sub-stage. Output of this sub-stage is a binary mask which indicates whether a pixel belongs to foreground or not. All the foreground pixels in the image can be summed up and their ratio to the the total number of pixels in frame can be compared to a threshold value. If the ratio is greater than threshold, then this frame is regarded as activity frame; otherwise it is classified as background frame. In this work, we use Mixture of Gaussian (MoG) and Google Summer of Code (GSoC) methods for background subtraction. We shall elaborate the working principal of MoG.

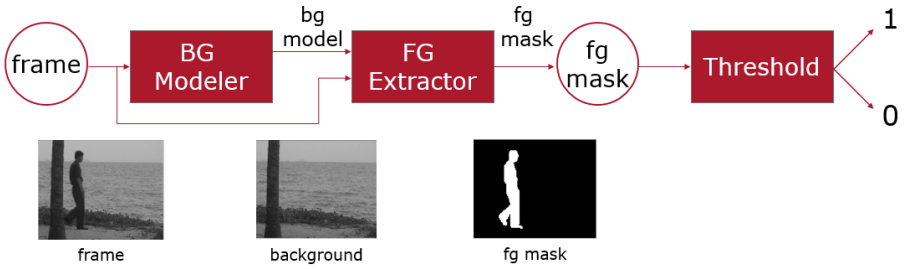

Figure 9: Background subtraction model

3.3.1 Mixture of Gaussian (MoG)

Before diving into the technical detail of MoG based background subtraction, let us explain it intuitively. This method attempts to model the pixel values as a gaussian process (normal distribution). Since, an image usually represents many different surfaces/objects, each surface/object is expected to give rise to a new gaussian. Thus all the pixel values are better represented by a mixture (sum) of gaussians. This is how gaussian mixture models an image. Notice that this model represents both foreground and background simultaneously. In order to apply this model to background subtraction problem, we associate each pixel with a particular surface and then associate that surface with either foreground or background. The label of each pixel (foreground/background) is determined by the label of corresponding surface. The methodology being described here is due to [27] and [28]

Each surface (or uniform object) that comes into the view is represented by a state $k \in 1, 2, 3, ..., K$. Some of these states correspond to background while remaining ones are considered to be foreground. The process **k** which generates the states is modeled by parameters set $w_1, w_2, ..., w_K$ where $w_k = P(k)$. Each of these parameters represents a priori probability of surface k appearing in the image. Further, $\sum_{k=1}^{K} w_k = 1$.

This surface process $\mathbf{k}$ is hidden and is only indirectly observable through pixel value process X. The pixel value process $\mathbf{X}$ is an observable random variable modeled by a gaussian process for given surface k. $\mathbf{X}$ is 1-D in case of gray scale images and 3-D for color images. If $\theta_k = \{\mu_k, \Sigma_k\}$ represent the associated gaussian process then pixel value process $\mathbf{X}$ given k is:

$$f_{\mathbf{X}|k}(X|k,\theta_k) = \frac{1}{\sqrt{(2\pi)^n|\Sigma_k|}} e^{-\frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k)}$$

where μ_k is the mean and Σ_k is the covariance matrix of associated k^{th} gaussian.

We assume these k events are disjoint so $\mathbf{X}$ can be modelled as sum of gaussians.

$$f_{\mathbf{X}}(X|\Phi) = \sum_{k=1}^{K} w_k f_{\mathbf{X}|k}(X|k,\theta_k)$$

where $\Phi = \{w_1, \mu_1, \Sigma_1, ..., w_K, \mu_K, \Sigma_K\}$. Figure 10 illustrates the pixel value probability $f_{\mathbf{X}}(X|\Phi)$ for 1-D pixel values $X \in \{0, 1, 2, ..., 255\}$, $K = 3$, $w_k \in \{0.2, 0.2, 0.6\}$, $\mu_k \in \{80, 100, 200\}$ and $\Sigma_k \in \{25, 5, 10\}$.

In order to apply the model to background subtraction problem, first step is to determine which of the K states is most likely to give rise to current pixel value $\mathbf{X} = X$. The posterior probability $P(k|X,\Phi)$ is the likelihood that pixel value X was generated by surface k. Using the Bayes's theorem:

$$P(k|X,\Phi) = \frac{P(k) f_{\mathbf{X}|k}(X|k,\Phi)}{f_{\mathbf{X}}(X,\Phi)}$$

The k which maximizes the $P(k|X,\Phi)$ is considered to be the surface associated with X. Figure 11 illustrates the posterior probability $P(k|X,\Phi)$ as a function of X for each $k \in \{1, 2, 3\}$ with the parameters in Figure 10.

$$\hat{k} = \underset{k}{\operatorname{argmax}}\, P(k|X,\Phi)$$

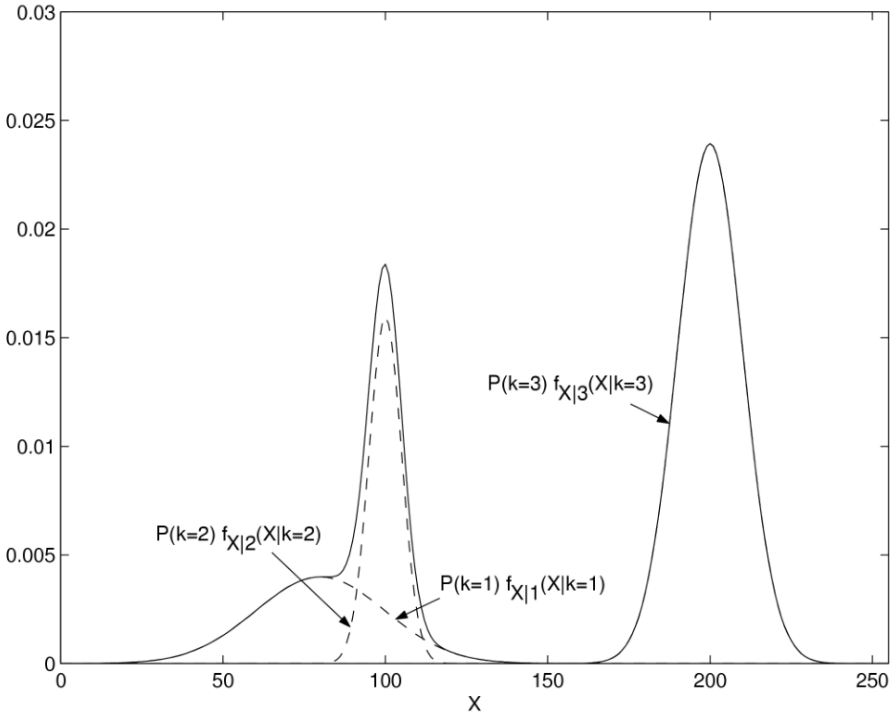

Figure 10: Gaussian mixture model[28]

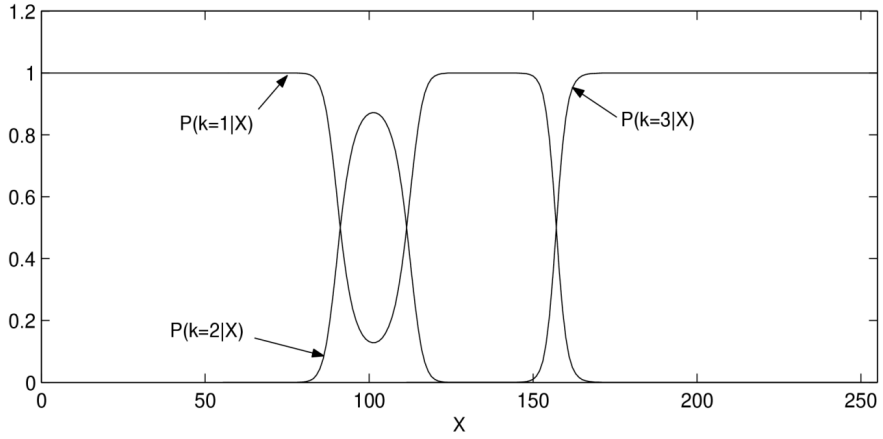

Figure 11: Posterior probability $P(k|X, \Phi)$[28]

Once X has been associated with a particular surface $\hat{k}$, it needs to be determined whether $\hat{k}$ is a foreground surface or background.

The procedure for demarcation starts with ranking K states by $w_k/|\Sigma_k|$ in decreasing order. This ratio is proportional to height of weighted distribution $w_k f_{\mathbf{X}|k}(X|k, \theta_k)$. A surface k is considered to be background if it occurs more frequently (higher w_k) and does not vary much (low $|\Sigma_k|$). To separate the foreground and background surfaces, an overall prior probability T of anything being in the background is used. The first B of the ranked states whose accumulated probability crosses the threshold T are considered to be background.

$$B = \underset{b}{\operatorname{argmin}}(\sum_{j=1}^{b} w_j > T)$$

3.4 Stage 2

As mentioned before, goal of this stage is to verify human trespassing in case of activity. We implement this stage using Faster-RCNN[29]. Faster-RCNN is an

object detection algorithm which takes in an image and predicts different objects in the image with their corresponding labels and bounding boxes. This is known as object detection and localization task which is different from the classification task (human trespassing/other activity) which we attemp to solve. We employ a simple an straight forward methodology to convert the Faster-RCNN output to our required output. If Faster-RCNN predicts at least one person with a probability greater than threshold τ, then we label the input frame f_i as showing trespassing activity. Otherwise we label it as a frame showing other activity.

Faster-RCNN has three main components/sub-networks (Figure 12)

1. Feature extraction (Conv Net)
2. Region Proposal Net (RPN)
3. Fast-RCNN head

3.4.1 Feature extraction

This component/sub-network takes in the input image and produces convolutional features. These convolutional features will be used by further sub-networks to predict the proposals and detections. This sub-network is also known as the backbone of network as it is responsible for producing high-quality, highly-discriminative features. This sub-stage is flexible in the sense that it can use any feature extraction network such as VGG or Resnet. Recent implementations also employ FPN[30] to improve the discriminative power of this sub-stage. In our experiments, we used Resnet-50 with FPN.

3.4.2 Region Proposal Network (RPN)

This sub-stage as the name suggests is responsible for proposing regions (rectangles) potentially containing objects. The idea of proposing regions using a neural network was proposed by Faster-RCNN for the first time. They also

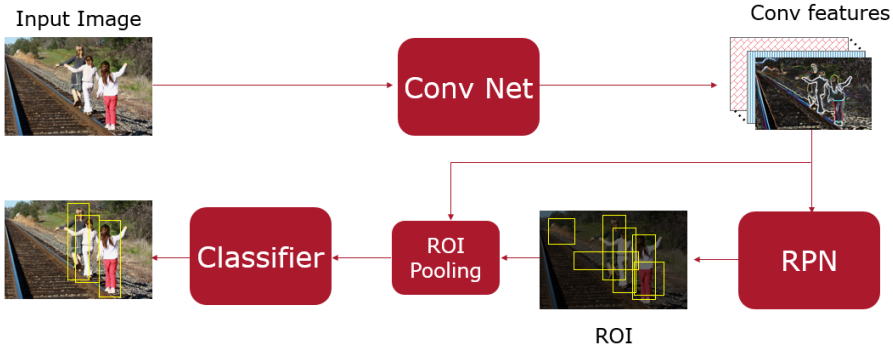

Figure 12: Faster-RCNN pipeline

proposed the novel idea of anchors. As seen in Figure 12, this sub-stage takes in the feature map and produces a list of proposals for the given image. Each proposal consists of binary label and proposed bounding box of the region of interest. The label indicates whether the proposal corresponds to an object or background. The proposals thus produced by the network are subjected to non-maximum suppression. This process removes the duplicate proposals and makes subsequent processing more efficient.

a) Anchor

The novel idea of anchor has been introduced by Faster-RCNN. Anchor act as default region proposals. Their idea has been motivated from multi-scale sliding windows. Suppose we use a feature extraction convolutional network such that it converts a 800×800 image to 50×50 feature map (Figure 13). This means every (x, y) location on feature map corresponds to 16×16 patch/window on original image. Similarly, 8×8 window on feature map corresponds to 128×128 window on original image. This 8×8 window on feature map is known as anchor. Faster-RCNN proposes multi-scale, multi-aspect ratio anchors. A total of 3 scales (8, 16, 32 on feature map) with 3 aspect ratios $(1:1,\ 1:2,\ 2:1)$ produces 9 anchors on each (x, y) location on feature map. Since, we have 50×50

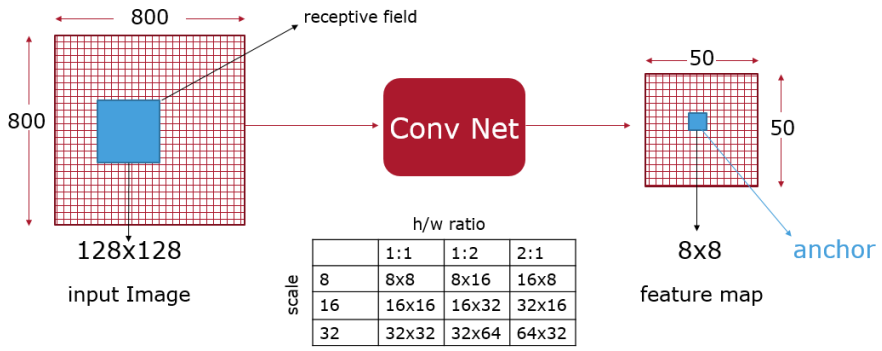

Figure 13: Anchor illustration

locations, therefore this setting produces 22,500 anchors in total. However, in practice we use less than that. All the anchors whose regions lie outside feature map (eg. anchors near edges); they don't participate in training the network.

b) Architecture

Figure 14 shows the architecture of RPN sub-network. Input to this network is the features generated by backbone network. These features are passed through a 3×3 "same[2]" convolution layer. Faster-RCNN uses 512 output feature depth for this layer. Output of this layer is fed to the bounding box regressor layer and objectness layer which predicts bounding box locations and objectness score simultaneously. Both of these layers are modeled with 1×1 convolution. Bounding box regressor layer has $4k$ output depth where k is the number of anchors and 4 follows from the fact that each proposal is defined by 4 scalar values. For similar reasons, objectness layer has $2k$ output features. Thus each anchor produces a proposal. All of these proposals are post-processed by Non Maximum Suppression (NMS) discussed later.

c) Anchor targets

While discussing the RPN architecture, we maintained that each anchor pro-

[2] (h, w) of input and output feature map remains same by automatic padding

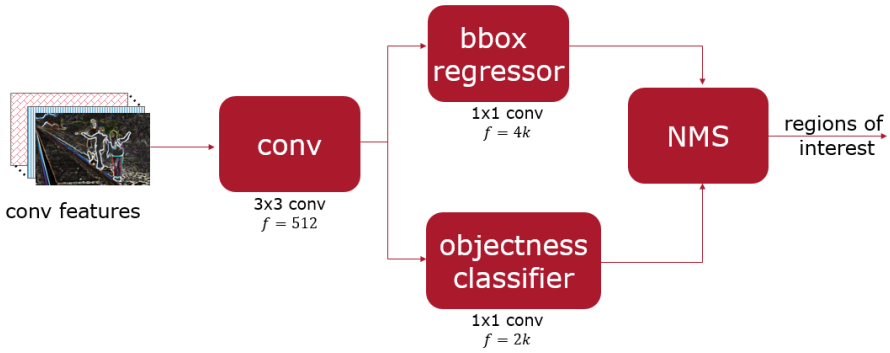

Figure 14: Region Proposal Network architecture

duces a proposal. Since, we shall train this network in a supervised manner therefore, we need targets corresponding to anchors. We shall use ground truth annotations to generate targets. Following steps need to be taken for anchor target generation.

1. compute IoU for each anchor-target pair
2. determine positive(negative) anchors and label them
3. confirm each ground truth is mapped to at least one anchor

Step 1 is simple. Given an anchor A_i and target bounding box T_j, we can compute Intersection over Union ($IoU(A_i, T_j)$) between them. Intersection over union is simply the ratio between area of overlap to area of union of two rectangles. Figure 16 illustrates the concept of IoU graphically. Once we have IoU for each anchor-target pair, we can proceed to step 2. For each anchor A_i, it is matched to ground truth bounding box $T_{\hat{j}}$ such that A_i has maximum IoU with $T_{\hat{j}}$ over all ground truth bounding boxes. In other words

$$\hat{j} = \underset{j}{\text{argmax}}\{IoU(A_i, T_j)\}$$

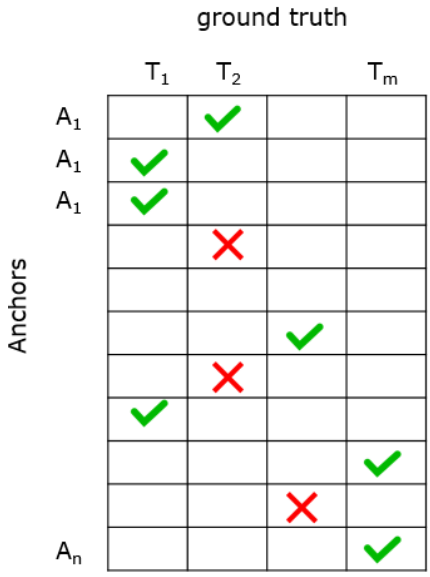

Figure 15: Anchor target assignment. Green tick indicates anchor assignment to ground truth object and red cross indicates background anchor assignment.

If $IoU(A_i, T_{\hat{j}}) > 0.7$, then A_i is assigned positive label i.e. this anchor corresponds to an object and A_i will regress to bounding box of T_j. If $IoU(A_i, T_{\hat{j}}) < 0.3$, then A_i is assigned negative label i.e. anchor corresponds to background. However, background anchors do not contribute towards bounding box regression learning process.

d) Non Maximum Suppression (NMS)

As indicated in Section 3.4.2, NMS is responsible for removing the duplicate predictions. Figure 17 illustrates the goal of this process graphically. In order to suppress duplicate proposal predictions with the less confidence, first step is to sort all the proposals in descending order. The first proposal is made the reference proposal and pushed to "keep" list. IoU of this reference proposal with all the remaining proposals is computed and the proposals which sufficiently overlap with the reference proposal ($IoU > 0.7$) are discarded. They are considered to be the duplicate of reference proposal. In the next iteration

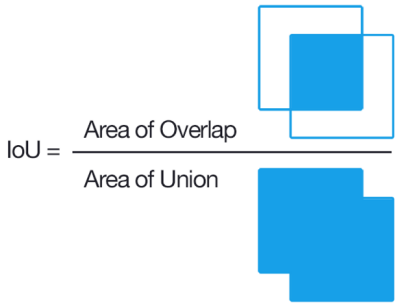

Figure 16: Intersection over Union

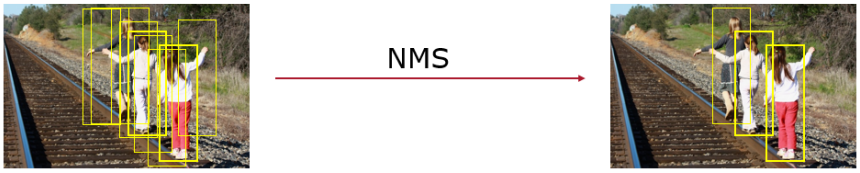

Figure 17: Non Maximum Suppression (NMS). Highly overlapping predictions with lesser confidence are suppressed.

the first proposal in the list of undecided proposals is made reference proposal and the process of first iteration is repeated. Again this leads to removal of all the proposals considered to be duplicate of reference proposal. The process continues on until all the proposals are decided i.e. either kept or discarded. Output of this process is the list of kept proposals.

3.4.3 Fast-RCNN head

Once we have the proposals from RPN, we need to predict the corresponding objects' labels and location. This is done by Fast-RCNN head. Fast-RCNN consists of two components: 1) convolutional feature extraction and 2) head. Since, we have already computed the features, we only need the Fast-RCNN head to do the remaining task. Fast-RCNN head itself has two further sub-

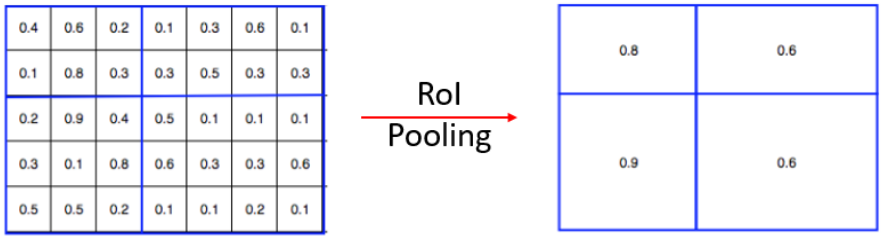

Figure 18: RoI pooling

components. First one is Region of Interest (RoI) pooling and second one is classifier layer. Input to the Fast-RCNN head will be proposal feature maps and output shall be improved bounding box locations of corresponding proposals along with class labels.

a) Region of Interest (RoI) Pooling

Different proposals have different feature map sizes. However, the classifier expects them to be of same size. This process (RoI pooling) is responsible for converting variable sized feature maps into fixed sized. The methodology used by Fast-RCNN in this case is quite simple. Suppose a feature map of size $n \times m$ has to be converted to $a \times b$ size. Then, a grid of size $a \times b$ is placed on top of feature map and maximum feature value from each grid cell is copied to corresponding cell in output buffer. this converts an $n \times m$ feature map to a size of $a \times b$. Figure 18 illustrates the concept by an example. In this case, feature map of size 5×7 is converted to 2×2.

b) Classifier

Once RoI pooling has adjusted the size of feature map to fixed dimensions, the feature maps are ready to be fed to classifier. The classifier takes in those features and pass them through two fully connected layers. The output of those two layers is fed to two separate fully connected layers responsible for predicting bounding boxes and object class labels. The bounding boxes and

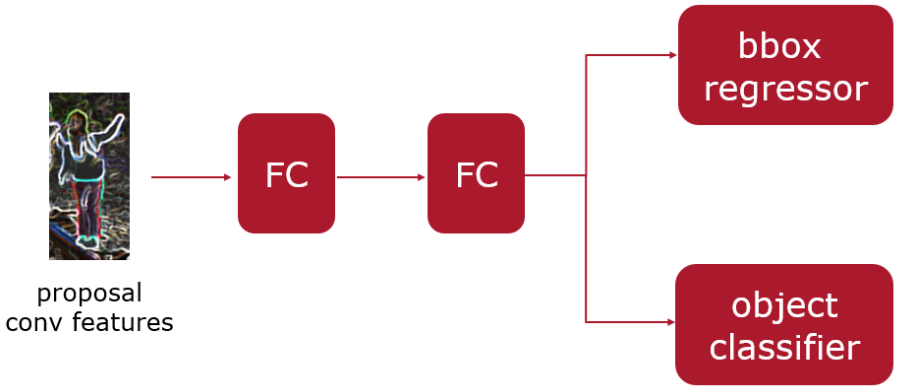

Figure 19: Fast-RCNN classifier

labels so predicted are the final output of Faster-RCNN. Figure 19 illustrates the architecture of classifier.

3.4.4 Training loss

While training the Faster-RCNN, we train two sub-networks: RPN and classifier. Both of these networks have two objectives: label classification and bounding box regression. Following two equations indicate the RPN and classifier loss. First term corresponds to label classification and second term corresponds to bounding box regression.

$$Loss_{RPN} = \frac{1}{N_{cls}} \sum_i L_{cls}(\hat{p}_i, p_i) + \frac{\lambda_r}{N_{reg}} \sum_i p_i L_{reg}(\hat{t}_i, t_i)$$

$$Loss_{classifier} = \frac{1}{N_{cls}} \sum_i L_{cls}(\hat{q}_i, q_i) + \frac{\lambda_c}{N_{reg}} \sum_i [q_i > 0] L_{reg}(\hat{u}_i, u_i)$$

where

$\hat{p}_i$ = anchor label prediction

$\hat{t}_i$ = anchor bounding box prediction

$\hat{q}_i$ = RoI label prediction

$\hat{u}_i$ = RoI bounding box prediction

p_i = anchor ground truth label

t_i = anchor ground truth bounding box

q_i = RoI ground truth label

u_i = RoI ground truth bounding box

λ_r = RPN loss balance coef.

λ_c = classifier loss balance coef.

N_{cls} = mini-batch size

N_{reg} = total anchors

Classification loss L_{cls} is the standard log loss and regression loss L_{reg} is the smooth-L_1 loss as defined below.

$$L_{cls}(\hat{y}, y) = -\sum_j y_j log(\hat{y}_j)$$

$$L_{reg}(\hat{b}, b) = \sum_{j=1}^{4} SL_1(\hat{b}_j - b_j)$$

$$SL_1(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ x - 0.5, & \text{otherwise} \end{cases}$$

where input parameters to each function carry standard meaning.

Total loss of the network is simply the sum of RPN loss and classifier loss.

$$Loss = Loss_{RPN} + Loss_{classifier}$$

Apart from that notice the term p_i in regression term of RPN loss. This makes sure that regression loss is activated only if proposal corresponds to an object, not the background. Thus bounding box predictions corresponding to background proposals do not contribute towards training. Furthermore, the

expression $[q_i > 0]$ does a similar job in classifier loss. This again acts as a flag to add regression loss only corresponding to actual objects and not the background.

λ_r and λ_c act as the balancing parameter between label classification and bounding box regression. The authors of Faster-RCNN claim that λ_r is redundant and Faster-RCNN remains insensitive to a large range of λ_r.

4 Experimental evaluation

In order to validate our approach, we carry out an extensive and in depth experimental evaluation. As discussed in Section 1.3, we study the time-accuracy trade-off. We also do stage-wise analysis where each stage is evaluated independently of other. This helps us understand which stage is acting as bottleneck in terms of performance. Additionally, we also do a noise based analysis to understand the robustness of system to noise. Figure 20 illustrates our experimental evaluation approach graphically.

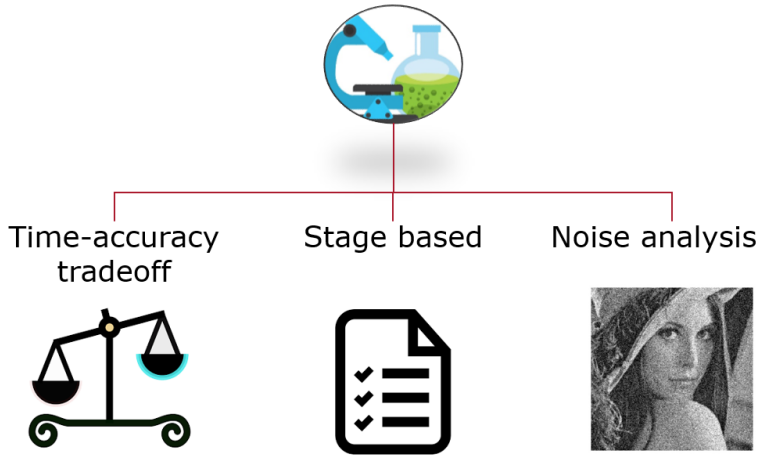

Figure 20: Experimental methodology

4.1 Dataset

The dataset we used for experimental evaluation is VIRAT 2.0 [31]. It is an outdoor video survillance dataset whose main aim is to facilitate activity classification. It has ground truth annotations of vehicle, person and other arbitrary objects (objects which come in contact with persons during activity). It has approximately 8.5 hours of video data in varying resolution and frames-per-seconds (fps). Figure 21 shows some samples from the dataset.

Figure 21: Samples from VIRAT 2.0[31] dataset

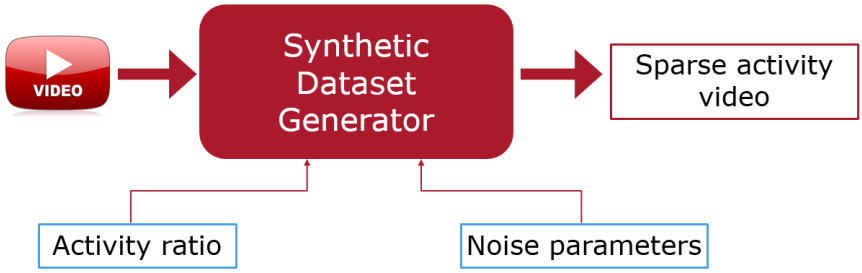

Figure 22: Synthetic dataset generator

4.1.1 Synthetic dataset generator

Instead of directly using the VIRAT 2.0 dataset, we synthesize more data from it. The reason why we need to do that is original dataset targets activity classification and is therefore enriched with human activity. We on the other hand need sparse activity data. Therefore, we use original VIRAT 2.0 dataset to generate new data that has controlled amount of activity to background ratio. Figure 22 helps understand our synthetic dataset generator.

In order to generate the synthetic data, we follow the following steps:

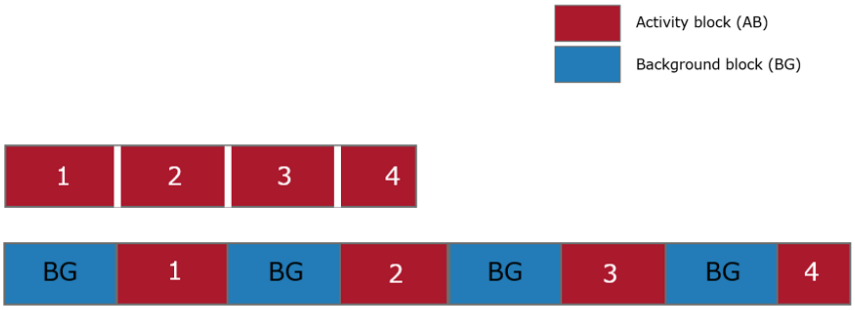

Figure 23: Synthetic video illustration

1. identify background frame
2. make background block
3. write background block(s)
4. write activity block
5. repeat (3) and (4)

Step 1 is simple. We manually scroll through a video and identify a frame with no activity (ideally no person). In step 2, we repeat the frame $\Delta \times fps$ times where fps is the frame-per-second of original video and Δ is the target length of background block in seconds. Then we alternatively write background block(s) and activity block in turn. An activity block is simply a section of original video. By default, we use $30s$ long activity blocks. Length of background block is also kept at $30s$ by default. Figure 23 helps understand the concept of activity block and synthetic video.

An important concept associated with this procedure is *Activity Ratio*. It is defined as

$$\text{Activity Ratio} = \text{AR} = \frac{1}{1 + nBG}$$

Table 1: Relationship of number of background blocks and activity ratio

nBG	nAB:nBG	AR
1	1:1	0.5
3	1:3	0.25
5	1:5	0.16

Table 2: Noise parameters

parameter	description
p	percentage of noisy frames in background block
μ	average number of noisy pixels in noisy frame
σ	standard deviation of number of noisy pixels

where nBG = number of background blocks per activity block. Figure 23 has $nBG = 1$. Table 1 explains the relationship between nBG and AR.

4.1.2 Adding noise

In order to test the robustness of our approach, we also add noise to our synthetic data. Table 2 discuses the parameters that control the level of noise. We use **salt and pepper** noise in our experiments.

In order to add noise, we first select $p\%$ of frames from each background block. Each of the frame is equally likely to be selected. Now we draw an integer r from normal distribution with parameters μ and σ. Now for each selected frame, we select r pixels and add noise to them. Again all pixels are equally likely.

4.2 Evaluation metrics

We use two performance metrics to measure the performance: f1 score and AUC. F1 score is selected as a metric to study performance w-r-t detector/classifier threshold. It is useful in the sense that it helps in picking the appropriate threshold for the classifier. AUC on the other hand is independent of a particular

threshold. Further, since we are dealing with unbalanced data (there may be much more background frames than activity frames); therefore AUC is a more reliable metric for classifier performance as it is insensitive to class imbalance.

4.2.1 F1 score

F1 score is the harmonic mean of precision and recall.

$$\frac{1}{f_1} = \frac{0.5}{precision} + \frac{0.5}{recall}$$

$$precision = \frac{\text{True positive}}{\text{positive predictions}} = \frac{TP}{TP + FP}$$

$$recall = \frac{\text{True positive}}{\text{actual positives}} = \frac{TP}{TP + FN}$$

where

TP = correctly predicted to be positive class

FP = incorrectly predicted to be positive class

FN = incorrectly predicted to be negative class

Precision indicates out of the samples predicted to be positive, how many of them are correct. Recall on the other hand indicates how many of the positive samples have been correctly identified.

4.2.2 Area Under the Curve (AUC)

AUC stands for Area Under the Curve. It is the area under the TPR-FPR curve (which is also known as Receiver Operating Characteristics (ROC) curve). TPR (True Positive Rate) and FPR (False Positive Rate) are defined as

$$TPR = \frac{TP}{TP + FN} = recall$$

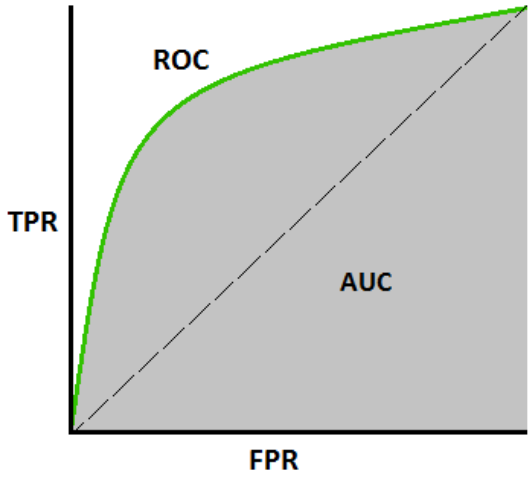

Figure 24: AUC illustraction. Image taken from [32]

$$FPR = \frac{FP}{FP + TN}$$

A particular value of threshold gives a particular (FPR, TPR) point. This point can be plotted on a 2D plot with TPR on the y-axis and FPR on the x-axis. Figure 24 shows a sample ROC curve. As the threshold goes from 0 to 1, the curve moves from top-right to bottom-left. A perfect classifier will have an AUC of 1.0 and a classifier that does no better than random guessing will have an AUC of 0.5.

4.3 Time-accuracy trade-off experiment

In time-accuracy trade off experiment, we study how the data can be processed in less time by compromising on accuracy. We vary certain control parameters (stage 1 threshold in this case) and observe the time it takes to process the data along with the performance of the complete pipeline. Figure 25 shows a comparison of MoG and GSoC for the trade off. The trade off curve depicts f1 score on y-axis and normalized processing time on the x-axis. Next we give the

Table 3: Synthetic data parameters for time-accuracy trade off

parameter	value
activity ratio	0.25
p	1%
μ	0.50%
σ	0.20%

definition of normalized processing time.

$$\text{normalized time} = \frac{\text{total time to process video data}}{\text{length of video data}}$$

It is clear that as the f1 score goes up, the normalized processing time also goes up, indicating the trade off. The figure also shows that MoG significantly outperforms GSoC. For a given f1 score, MoG takes less time than GSoC. Alternatively, for a given processing time, MoG shows better performance than GSoC.

Another interesting fact to note is that it would take around 2.6 hours of normalized processing time to process same data by stage 2 (Faster-RCNN) only. i.e. if stage 1 does not perform filtering. On the other hand, it takes around 1 hour to process the same data using our proposed pipeline. This shows a 2.6 times improvement in processing time for dataset having AR=0.25. The synthetic dataset parameters using in corresponding dataset are shown in table 3

Figure 26 and 27 show similar time-accuracy trade off for varying activity ratio (AR). For both algorithms, MoG and GSoC, the trade off curve shifts to the left as AR decreases. This is expected as stage 1 can filter more and more frames. Synthetic data parameters are the same as in table 3 with the exception of activity ratio.

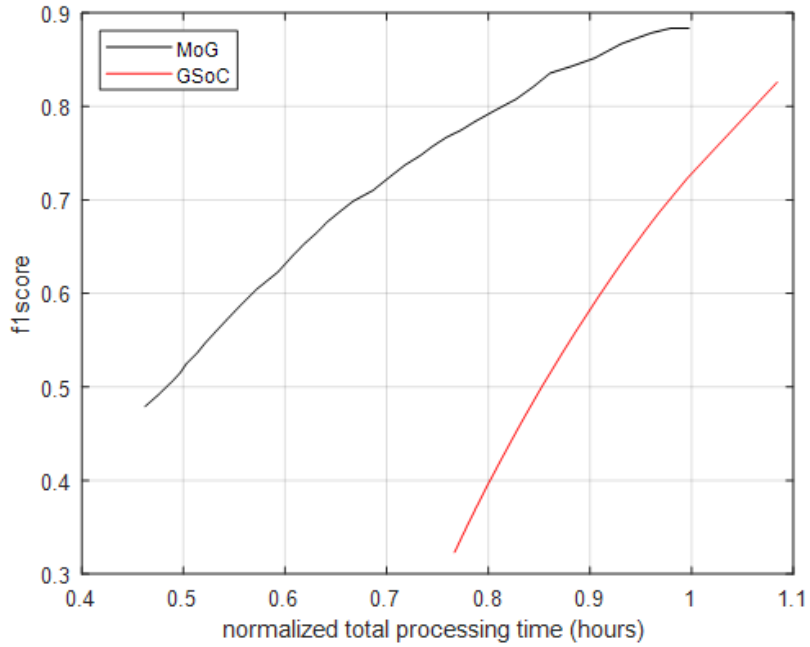

Figure 25: Time-accuracy trade off - MoG vs GSoC. Synthetic data parameters shown in table 3

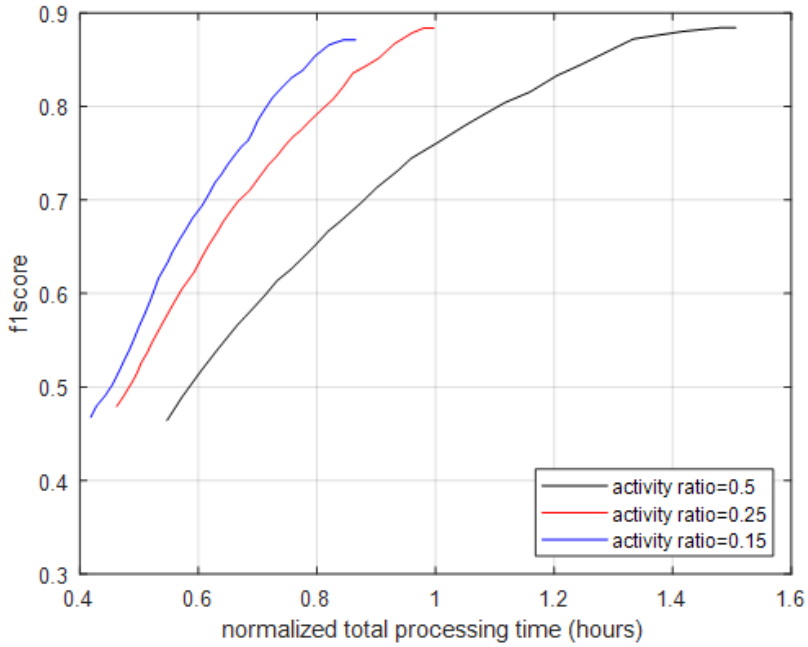

Figure 26: Time-accuracy trade off for varying AR using MoG

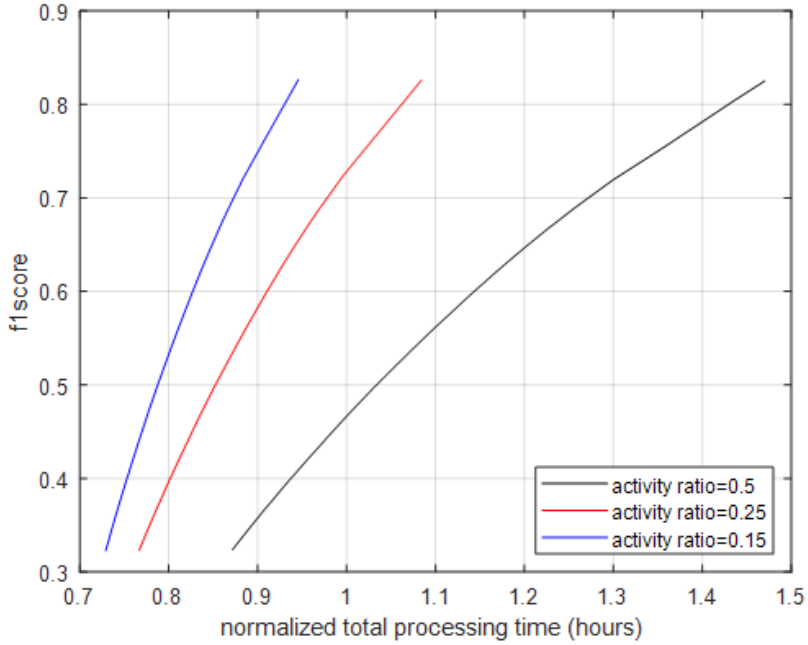

Figure 27: Time-accuracy trade off for varying AR using GSoC

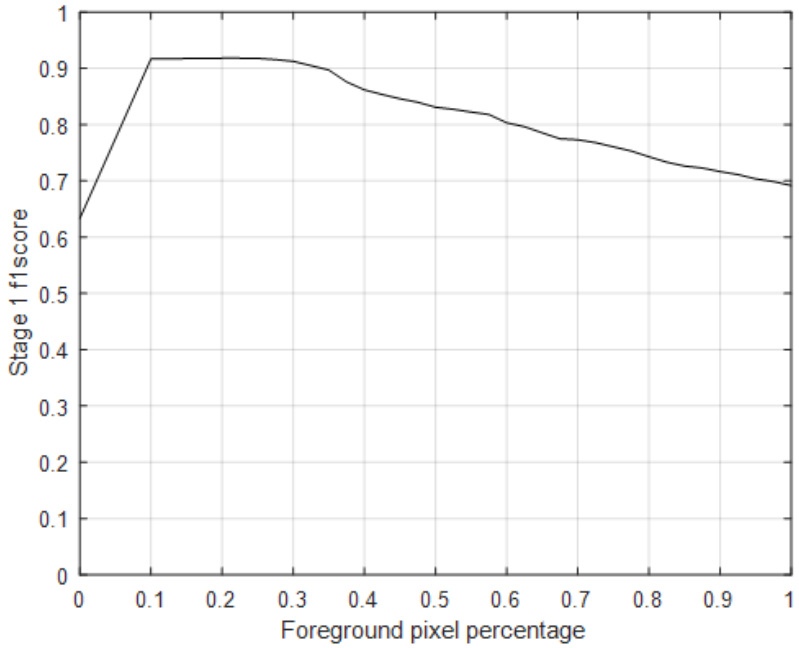

Figure 28: Stage 1 evaluation - MoG

Table 4: Stage 1 AUC and time analysis

algorithm	AUC	mean processing time (ms)
MoG	0.94	30
GSoC	0.6	150

4.4 Stage based experiments

In this part, we analyse both stage 1 and 2 independently. For stage 1, we are doing binary classification between activity and background frames. We shall vary our threshold (percentage of foreground pixels) and observe f1 score. AUC shall also be computed. We will evaluate both MoG and GSoC. Figure 28 and 29 show the results for MoG and GSoC respectively. Table 4 shows the AUC for both algorithms. It is clear that MoG outperforms GSoC in both f1 score and AUC. Further, MoG is significantly faster as compared to GSoC.

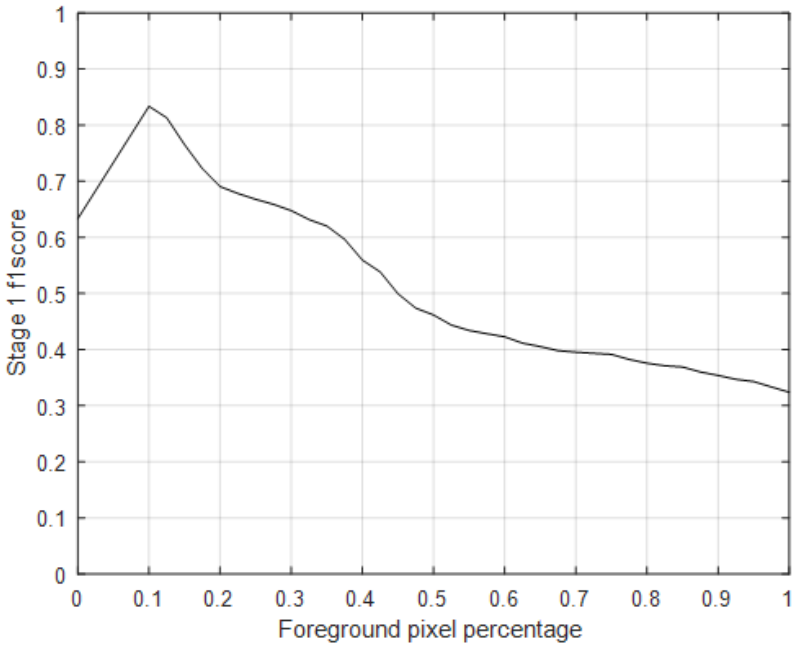

Figure 29: Stage 1 evaluation - GSoC

For stage 2 evaluation, we adopt a similar methodology. However, we need to clarify how we binarize our predictions. Remember for stage 2

$$\text{Ground Truth} = \begin{cases} 1, & \text{if there is a person} \\ 0, & \text{otherwise} \end{cases}$$

We define the prediction of stage 2 as

$$\text{prediction} = \begin{cases} 1, & \text{if there is at least one valid person detection} \\ 0, & \text{otherwise} \end{cases}$$

where

valid person detection $= \hat{p}_i > \tau \quad \& \quad IoU(\hat{b}_i, b_j) > 0.5$

$\hat{b}_i = i^{th}$ prediction bounding box

$b_j = j^{th}$ ground truth bounding box

$\hat{p}_i = i^{th}$ prediction probability

Based on that, Figure 30 shows the f1 score w-r-t prediction threshold. We achieve an AUC of 0.81 for this stage and we notice that on average it takes 0.5 seconds for the stage 2 (Faster-RCNN) to process one frame at 1080×960 resolution.

4.5 Noise analysis experiments

Third and final part of our experimental evaluation is noise analysis. In this part, we evaluate the robustness of stage 1 against noise. As discussed in Section 4.1.1, we only add noise to background frames. Noise analysis for stage 2 is not our goal and shall not be discussed here.

In terms of noise, we have 3 parameters: $p, \mu,$ and σ. Description of these parameters has been discussed in table 2. To study the influence of one param-

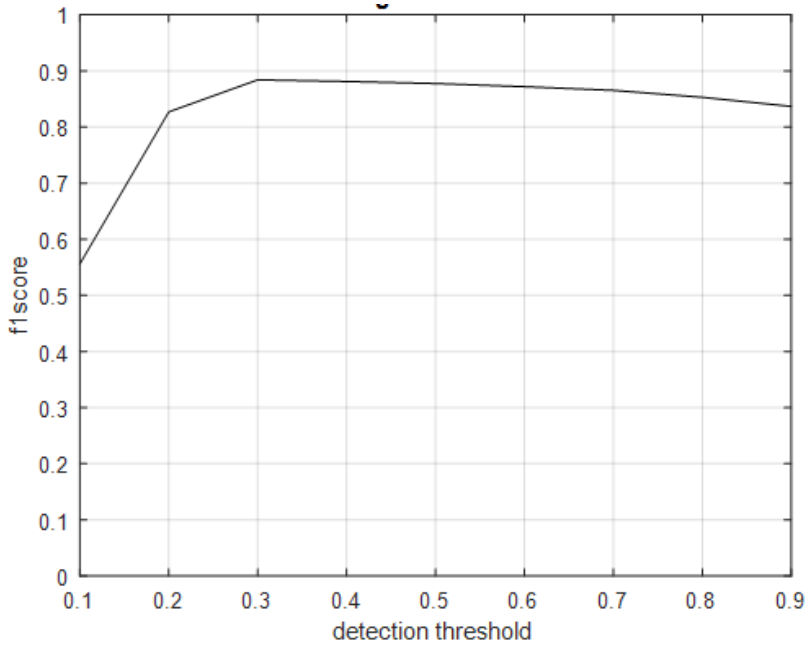

Figure 30: Stage 2 evaluation

eter, we vary it while keeping the others constant.

Figure 31 shows MoG's (stage 1) f1 score graph with variation in p. We use $AR = 0.5$, $\mu = 0.5\%$ and $\sigma = 0.2\%$ for this experiment. It is clear that increasing p forces the graph to go down. Thus more noise in terms of p directly affects the performance of stage 1. This result is also verified by reduced AUC as shown in Table 5.

Figure 32 shows MoG's (stage 1) f1 score graph with variation in μ. We use $AR = 0.5$, $p = 2\%$ and $\sigma = 0.2\%$ for this experiment. For variation in μ, the f1 score graph again seems to shift downwards when increasing μ. However, the difference is less visible in this case. This result is again verified by a reduction in AUC as shown in Table 6.

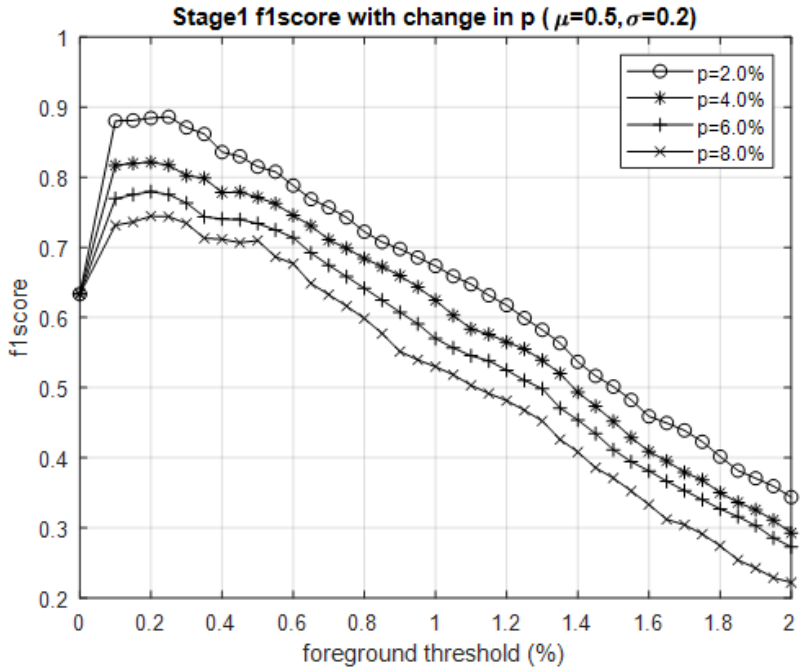

Figure 31: Noise analysis - varying p

Table 5: Noise analysis - AUC for varying p

p	AUC
2%	0.93
4%	0.87
6%	0.83
8%	0.78

Table 6: Noise analysis - AUC for varying μ

μ	AUC
0.5%	0.93
0.7%	0.92
0.9%	0.91
1.1%	0.88

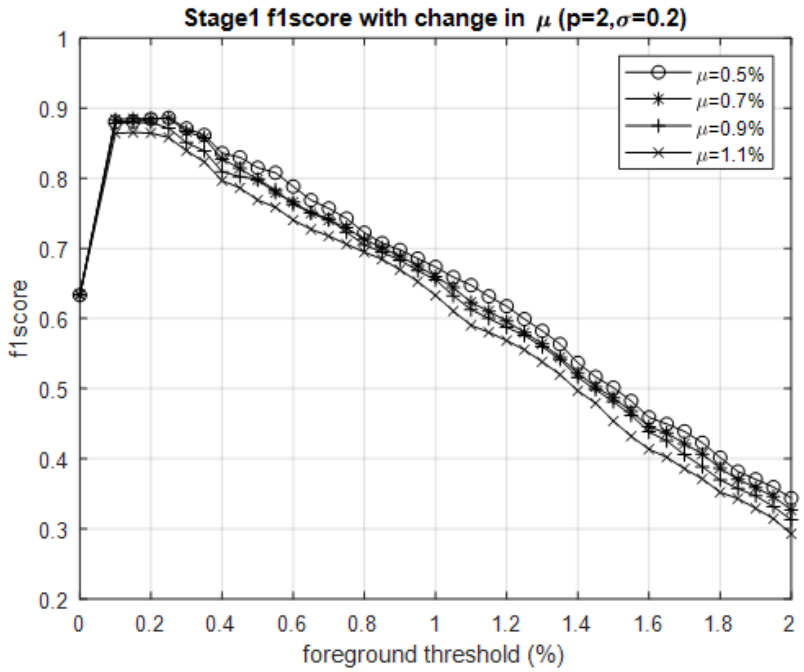

Figure 32: Noise analysis - varying μ

5 Conclusion

5.1 Summary

In this work, we design the critical component for a railroad trespassing detection system. Although initially envisioned for railroad security, the proposed approach has potential applications in video surveillance domains characterized by a sparsity in activity. The contributions of this thesis include a flexible pipeline that can trade off speed and accuracy. The system by design consists of two stages where the first stage is responsible for efficiently removing the background frames from the activity frames. The second stage is responsible for differentiating between human trespassing activity and any other unknown activity. Our proposed pipeline is composed of off-the-shelf components. Other algorithms relevant to stage 1 and stage 2 could equally be plugged in. We demonstrate the effectiveness of our approach on a public domain surveillance dataset.

5.2 Future work

There are many interesting directions in which this work can be further advanced. One key direction is to build a trespassing prediction system that uses the output of this detection system to predict trespassing events in near future. Another direction is towards improving the accuracy of detection system. We note that the current performance is limited by the performance of stage 2. Currently stage 2 doesn't use any temporal information i.e. each frame is treated independently and is not conditioned on the previous frames (history). We believe that utilizing the temporal information can significantly improve the performance specially for challenging cases of occlusion and background (discussed in section 1.4).

Furthermore, an approach towards reducing the processing time could be to reduce the number of proposals (Regions of Interest) generated by Faster-RCNN. Since, we are interested in detecting human trespassers only, we can use the foreground mask to produce RoI (Region of Interest) proposals. Suppressing the color information corresponding to the background area should significantly reduce the number of proposals. Also, the whole process can be simplified by considering the foreground areas as proposals and classifying them directly with a classification network as opposed to using Faster-RCNN.

References

[1] "Trespass casualties by state." https://oli.org/about-us/news/statistics/trespassing-fatalities-by-state. Accessed: 2018-12-10.

[2] "Railroad crossing collisions, deaths up last year." https://www.newsleader.com/story/news/local/2018/03/27/railroad-crossing-collisions-deaths-up-last-year/462122002. Accessed: 2018-12-10.

[3] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 253–256, IEEE, 2010.

[4] A. M. Aung, "Automatic eye-gaze following from 2-d static images: Application to classroom observation video analysis," Master's thesis, Worcester Polytechnic Institute, 2018. https://digitalcommons.wpi.edu/etd-theses/251.

[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[7] https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size/23186. Accessed: 2019-03-05.

[8] https://stackoverflow.com/questions/44287965/trying-to-confirm-average-pooling-is-equal-to-dropping-high-frequency-fourier-co?noredirect=1&lq=1. Accessed: 2019-03-06.

[9] https://pennlio.wordpress.com/2014/04/11/fully-connected-locally-connected-and-shared-weights-layer-in-neural-networks. Accessed: 2019-03-07.

[10] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724, 2014.

[11] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, pp. 98–136, Jan. 2015.

[12] 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.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[14] P. VIODA, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE CVPR 2001*, pp. 905–910, 2001.

[15] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features.," in *iccv*, vol. 99, pp. 1150–1157, 1999.

[16] Z. Lin and L. S. Davis, "A pose-invariant descriptor for human detection and segmentation," in *European conference on computer vision*, pp. 423–436, Springer, 2008.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[19] R. Girshick, "Fast r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[20] 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.

[21] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[22] D. Chetverikov and A. Axt, "Approximation-free running svd and its application to motion detection," *Pattern Recognition Letters*, vol. 31, no. 9, pp. 891–897, 2010.

[23] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[24] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[25] https://docs.opencv.org/4.0.0/d4/dd5/classcv_1_1bgsegm_1_1BackgroundSubtractorGSOC.html. Accessed: 2019-04-20.

[26] L. Guo, D. Xu, and Z. Qiang, "Background subtraction using local svd binary pattern," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 86–94, 2016.

[27] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2, pp. 246–252, IEEE, 1999.

[28] P. W. Power and J. A. Schoonees, "Understanding background mixture models for foreground segmentation," in *Proceedings image and vision computing New Zealand*, vol. 2002, 2002.

[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] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2117–2125, 2017.

[31] 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.

[32] "Receiver operating characteristics curve." https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. Accessed: 2019-04-24.