Traffic Sign Recognition

Jayapratha T , Computer science and engineering , Sri Eshwar college of Engineering

Dinesh Kumar S , Computer science and engineering , Sri Eshwar college of Engineering

Guna seelan D , Computer science and engineering , Sri Eshwar college of Engineering

Avinash Madhav K , Computer science and engineering , Sri Eshwar college of Engineering

Jagadheeshwaran A , Computer science and engineering , Sri Eshwar college of Engineering

---

ABSTRACT

Traffic Sign Recognition (TSR) from video images is an integral part of the driver support functions needed to make intelligent vehicles a reality. TSR is composed of two components: detection and classification. The focus of the proposed research is detection of U.S traffic signs based on the LISA dataset, the largest publicly available U.S traffic sign dataset in the world, comprising over 9,000 images.

Detection methods involving Integral Channel Features and Aggregate Channel features have achieved state-of-the art performance. Our proposed research consists of a comparative study of the performance of

Integral Channel Features and Aggregate Channel features versus using Convolutional Neural Networks (CNN). Our aim is to explore the detection performance of the CNN by varying the convolutional layers, max-pooling layers, and the fully-connected layers. We will evaluate our detection performance by using the PASCAL measure, which is a standard metric for this application.

INTRODUCTION

For the past several years there has been significant research interest in self-driving vehicles and Advanced Driver Assistance Systems (ADAS), most notably Google's Self-Driving Car Project. In order for these systems to become more and more autonomous, it is essential for the integration of Traffic Sign Recognition (TSR) technology. TSR systems are composed of two main components, classification and detection. The classification component focuses strictly on classifying the type of traffic sign, after the sign has be detected. Whereas, the detection component focuses on locating the traffic sign in a sequence of video images. This work focuses exclusively on the detection task of TSR systems.

Traffic sign detection has been heavily researched on European traffic signs, or more specifically traffic signs that follow the Vienna Convention of Road Signs and Signals (United Nations et al., 1978). This is largely due to the introduction of the German Traffic Sign Detection Benchmark (GTSDB) (Houben 2013) competition. This competition has shown promising results, however the performances of the top performing models of the GTSDB have not been able to translate as well when used for U.S traffic signs (Mogelmose 2015). One of the main reasons that European models have not be able to translate as well is that, U.S signs and European signs, which have very similar meanings can look significantly different as shown in Figure 1.1 (Mogelmose 2015). Figure 1.1 (a)-(c) are example of Vienna Convention signs and (d)-(f) are examples of U.S traffic signs. Figure 1.1(a)(d) are both Keep Right signs, figure 1.1(b)(e) are both Left Turn signs, and figure 1.1 (c)(f) are both Speed Limit signs. From figure 1.1 we can clearly see there are significant difference, in geometry and color, for the Vienna

(a) (b) (c) (d) (e) (f)

Figure 1.1: Examples of Vienna Convention signs and U.S signs

2.BODY OF PAPER

Convention and U.S traffic signs that have the same meaning. In order to close this gap in performance, the largest U.S traffic sign dataset in the world, the Laboratory for Intelligent and Safe Automobiles (LISA) dataset (Mogelmose 2012), was created. The LISA dataset contains approximately 10,000 images containing U.S traffic signs and approximately 11,000 images that have similar scenes but do not encompass any traffic signs in the images.

Two of the most promising methods used for traffic sign recognition include, Integral channel Features and Aggregate Channel Features. Integral Channel and Aggregate Channel features were originally applied to Pedestrian Detection (Dollar 2009, 2014) and later adopted for TSR. Where they have shown to achieve state-of-the-art performance for both European and U.S traffic signs (Mogelmose 2015). Although these methods have demonstrated state-of-the-art performance there is still room for improvement. To our knowledge, deep learning architectures, specifically Convolutional Neural Networks (CNN), have not been explored for TSR. CNNs have shown to be very powerful models for image recognition tasks, as was shown in the ImageNet Challenge (Krizhevsky 2012). Motived by these results, our proposed research consists of a comparative study of the performance of Integral Channel Features and Aggregate Channel features versus using Convolutional Neural Networks (CNN). Our aim is to explore the detection performance of the CNN by varying the convolutional layers, max-pooling layers, and the fully-connected layers. We will evaluate our detection performance by using

the PASCAL measure (Everingham 2010), which is a standard metric for this application.

This chapter will focus first on a simple overview of traffic sign recognition systems, followed by a discussion on the related work on both the German Traffic Sign Detection Benchmark (GTSDB) and on the LISA dataset. We will conclude this chapter with an overview of the entire thesis proposal.

1.1 Traffic Sign Recognition Overview

authors are referring to TSR systems they are referring to the detection of the traffic signs, the classification of that traffic sign, once it has been detected, and the tracking of the traffic sign. Figure 2 (Mogelmose 2012) shows a basic block diagram of the TSR system.

When given an image or video sequence as an input, the detection task is only concerned with discovering where at in the image the traffic sign is located. Whereas, classification is only concerned with deciding what type of traffic sign

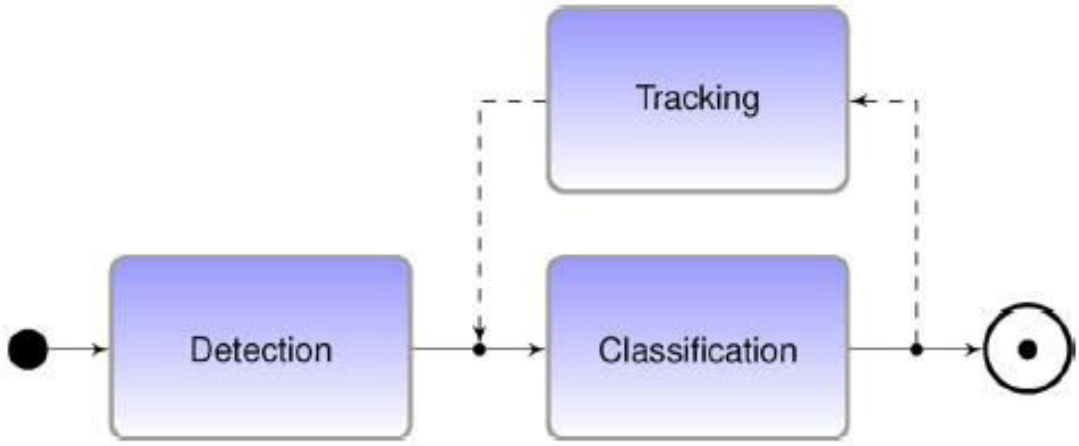

Figure 1.2: Block Diagram of a TSR System

Although in this work we will be examining the detection aspect of Traffic Sign Recognition (TSR) systems, it will be useful to describe the whole system. Typically when we

the detected traffic sign is. The final component is tracking, which tracks the detected sign from frame to frame. Each component can be approached separately in different ways. However

for a full TSR system to work each component will depend on each other.

Since our work is focused on the detection task we will only look at this task at a closer level. A simple model of the Detection task can be seen below in figure 3. Figure 3 is a very simplistic block diagram of the detection process, however most method will look like this in some form or another. They will typically have an image preprocessing stage, then a feature extraction stage, and then a detection stage. The image preprocessing stage is used to help clean the data or transform the image in order to prepare it for the feature extraction stage. In the feature extraction stage, features such as edges, Harr-like features, and color features are taken from the image and sent to the detector. The detector depends on the type of method that we use. Typically there are two methods used, one is a model-based approach and the other is a learning-based approach, both will be discussed in more detail in the next chapter. The purpose of the detector is to take the features as input and make a decision on where the traffic sign is located in the image. Although in this work the focus is on the detection step it is important to note that without the classification and tracking components, a TSR system is useless.

1.2 Related Work

Traffic sign detection has been researched

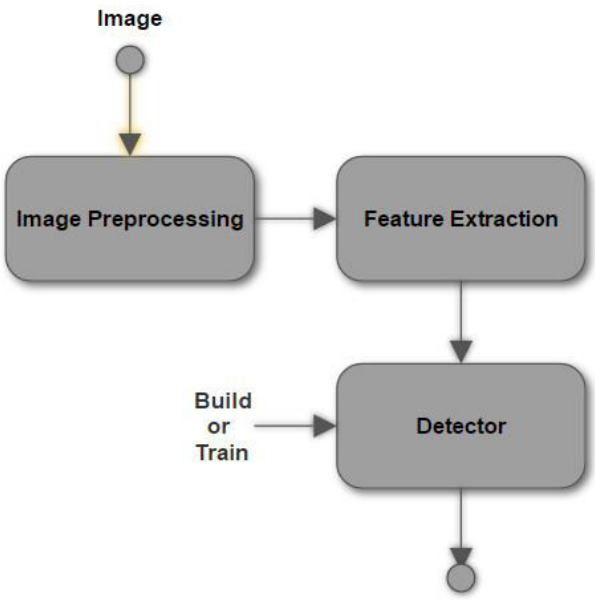

Figure 1.3: Basic Block Diagram of the Detection Task

for a little over a decade. Some of the earliest methods tried to take advantage of the fact that traffic signs were designed to stand out in the environment, often through shape and color. The early shape-based methods searched for specific geometrical shapes such as rectangular or circular objects. For example Loy (2004) used edge features with a radial symmetry model for a dataset of approximately 50 images. Early color-based methods extracted colors from different color spaces of the input image, and used different thresholding techniques to extract the traffic sign. For example Vazquez-Reina (2005) converted the image into HIS color space and used thresholding in order to extract the traffic signs. While in theory, the color and shape of traffic signs can be very well defined, they can have some very practical issues, such as, illumination changes, damaged or worn signs, occlusions, and traffic signs blending in with background color. This lead to the implementation of learning-based methods that used more descriptive features such as Harr-like wavelet features, introduced by Viola and Jones (2001) for facial recognition, and HOG features, introduced by Dalal and Triggs (2005). Mogelmose (2012) surveyed the traffic sign detection field which illustrates both the early model-based methods and learning-based methods. A summary of the different techniques surveyed can be seen below in table 1 (Mogelmose 2012). For more specific details of each method please refer to Mogelmose (2012). However trying to compare the performances of each surveyed methods becomes very problematic due to the inconsistencies in types of traffic signs and the number of traffic signs used in each method.

Paper	Year	Author group	Segmentation method	Features	Detection method
[16]	2005	1	HSI thresholding with addition for white signs ([49])	Boundary distance transform	Correlation with model distance transforms
[47]	2007	1	HSI thresholding with addition for white signs ([49])	DtB (distance to bounding box)	Linear SVM
[33]	2008	1	HSI thresholding with addition for white signs ([49])	FFT of shape signatures	Euclidian nearest neighbor
[48]	2010	1	HSI thresholding with addition for white signs ([49])	DtB (distance to bounding box)	Linear SVM
[66]	2005	2	None	Haar wavelet features computed on specific color channels	Cascaded classifier
[40]	2008	2	Extended radial symmetry voting	Haar wavelet features	Cascaded classifier
[17]	2006	3	LCH thresholding (obtained with CIECAM97)	HOG	Comparison with template vectors
[50]	2008	3	LCH thresholding (obtained with CIECAM97)	None	None
[62]	2007	4	None	Color filtered edges	Extended radial symmetry voting
[34]	2010	4	HSV discretization	Edges	Extended radial symmetry voting
[20]	2011	4	Quad-tree color selection	Edges	Extended radial symmetry voting
[35]	2004	5	None	Edges	Extended radial symmetry voting
[54]	2006	5	None	Edges	Extended radial symmetry voting
[55]	2008	5	None	Edges	Radial symmetry voting
[56]	2008	6	None	Edges	Votes for symmetric areas to be used as ROI with another shape-detector
[57]	2011	6	None	Edges	Two-tier radial symmetry voting
[58]	2011	7	None	Edges of closed contours with certain aspect ratios	Hough shape detection
[59]	2011	7	None	Edges of closed contours with certain aspect ratios	Hough shape detection
[18]	2009	8	Adaptive RGB threshold	Edges	Fuzzy templates (a Hough derivative)
[51]	2010	8	Adaptive RGB threshold	Edges and Haar-like features	Fuzzy templates, cascaded classifier, and SVM
[64]	2008	9	None	HistFeat (HOG derived)	Cascaded classifier
[39]	2011	9	None	Various HOG-features	5 stage cascaded classifier trained with LogitBoost
[49]	2002	None	HSI thresholding with edge detection and removal of achromatic colors	Edges	Genetic algorithm looking for circles
[63]	2007	None	None	Edge orientation histograms	Comparison with template vectors
[41]	2007	None	HSI thresholding	Edges	Hough shape detection
[60]	2007	None	None	Edges	Hough transform for circular signs, proprietary (not described) for rectangular
[42]	2008	None	HSI thresholding	30x30 px YcbCr patches	Neural network
[37]	2009	None	None	Dissociated dipoles	Cascaded classifier
[61]	2009	None	None	Edges	Vertex and Bisector transform (VBT)
[36]	2009	None	Radial symmetry voting combined with SIFT features	Edge colors	Contracting Curve Density
[43]	2009	None	HSV thresholding	Edges	Hough shape detection
[19]	2009	None	Saliency detection with color and edges	HOG	SVM
[44]	2009	None	Hue thresholding on chromatic colors only	Tangent function of simplified contours	Distance from model tangent function
[45]	2010	None	HSI thresholding	Edges	Circle center voting
[65]	2010	None	None	HOG augmented with color information	SVM
[21]	2010	None	Biologically inspired attention model	Color, corner positions, height, excentricity	Color, corner positions, height, excentricity
[46]	2010	None	HSI thresholding	Edges	Radial symmetry voting
[52]	2011	None	Nested cascade classifier with Local Rank Pattern features (based on 7 RGB based colors)	Edges	RANSAC circle fit
[38]	2011	None	Radial symmetry voting	Colors in modified RGB-space	Distance to learned colors
[8]	2011	None	Probabilistic color preprocessing	Edges	Hough derivative shape detector
[25]	2011	None	None	Fourier descriptors	Correlation based matching

Table 1.1: Summary of Surveyed TSD Techniques Until 2012

The creation of the German Traffic Sign Detection Benchmark (GTSDB) competition (Houben 2013) has made the comparison of different TSD methods much easier and has significantly pushed state-of-the-art performances on European signs. Typically TSR research was split into two different methodologies, model-based and learning-based approaches. However more recently learning-based methods have continued to outperform model-based methods and are the preferred choice over model-based approaches. This was also observed in the GTSDB competition, where all of the frontrunners used learning-based approaches. The top three teams of the GTSDB competition where: Team VISICS (Houben 2013), Team Litsi (Liang 2013), and Team wgy@HIT501 (Wang 2013). Team VISICS proposed a method that uses Integral

Channel Features (ICF), which was originally proposed by Dollar et al. (2009) for pedestrian detection. Team VISICS trained these features using a boosted decision forest. Team Litsi proposed a method that first segmented the image into regions of interest, through shape matching and color classification. They then used HOG features and color histograms to train a SVM. Team wgy@HIT501 also used HOG features, however found the best candidates with LDA. The best HOG feature candidates were trained using an IK-SVM. The GTSDB was one of the major catalysts that pushed the traffic sign detection performance to near perfection for European traffic signs.

However, the research activity in U.S traffic signs was significantly less than that of the European traffic signs. Staudenmaier (2012) proposed a method for U.S speed limit signs, which used a Bayesian Classifier Cascade with intensity features and tensor features. The worked showed to have a good detection rate above 90% but at the cost of multiple false positives per image, which in comparison to the European state-of-the-art methods was much worse. Liu (2012) also ran into a similar issue of multiple false positives with a good detection rate, but only for speed limit signs. One of the main problems for U.S traffic signs, was the fact that there wasn't a large dataset that can be used as a benchmark. It wasn't until Mogelmose (2012) that the LISA dataset became available. This dataset became the largest collection of U.S traffic signs in the world. With it, Mogelmose (2015) was able to achieve state-of-the-art performance by using Integral Channel features and Aggregate Channel features with a boosted decision tree forest. There was still a performance gap between their performance and the state-of-the-art performance of the European dataset, however this gap was significantly less than that of the methods used for U.S traffic signs. Lim (2014) also presented work on the LISA dataset with somewhat worse detection performance that Mogelmose (2015), but tries to address the issue of adverse weather conditions.

1.3 Thesis Overview

The remainder of this thesis will be divided into the following sections. In Chapter 2 we will discuss all of the necessary background

information. Including model-based approaches vs. learning-based approaches, the current state-of-the-art methods for the GTSDB and LISA datasets, and deep learning architectures. Chapter 3 will explain the proposed method and the data that we will be using in this work. Chapter 4 will include the preliminary experiments and the results of the experiments. Chapter 5 will discuss the expected outcomes and a timeline for the remaining tasks.

CHAPTER 2

BACKGROUND

In this chapter we will describe all of the necessary background information for this work. We will first describe the sliding window detection method, which is implemented in most object detection/recognition systems. We will then transition into a detailed description of the current state-of-the-art method for traffic sign detection. Then the final section will discuss the basics of deep learning and conclude with a detailed description of Convolutional Neural Network (CNN) architectures, which will be our proposed method for traffic sign detection.

2.1 **Sliding Window Detection**

Most object detection/recognition systems will typically implement a sliding window method. The first step of any learning-based object detection system, which implements a sliding window, is to train a classifier from a dataset of labeled images of size $n \times m$. The dataset should contain positive and negative image samples. Where the positive samples are large centered instances of the object we are looking for, of size $n \times m$. The negatives images in the dataset should contain images with scenery similar to that of the positive images however, they cannot contain the object in them. To obtain the negative samples, image patches of size $n \times m$ are randomly extracted at different scales of the negative images. Once the dataset is organized a binary classifier needs to be trained on the positive and negative samples. Instead of training on the image patches themselves we train on features that are extracted from the image patches.

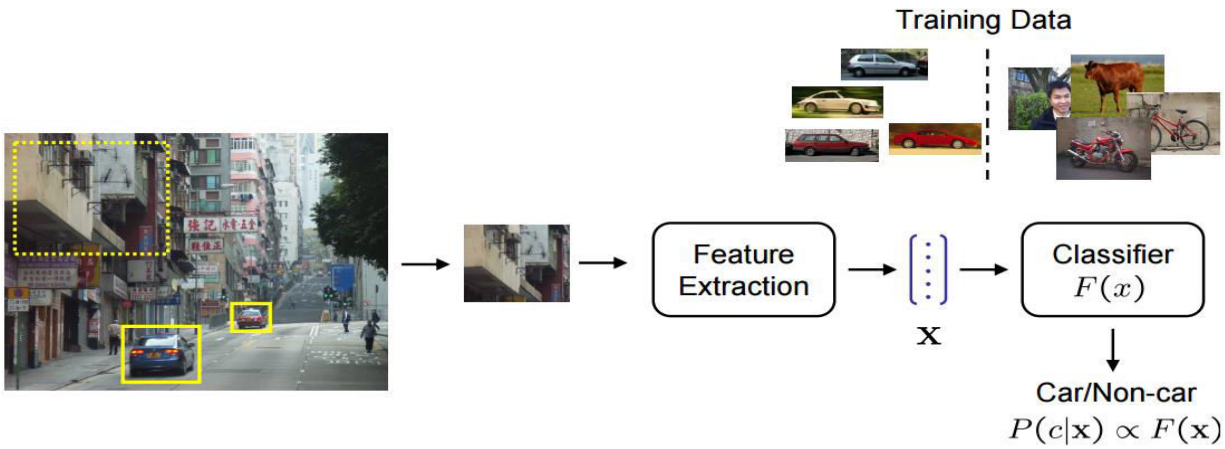

Figure 2.1: Sliding Window Process

Once the classifier is trained, we will test on test images by extracting an image patch, extract the features from the image patch, then pass the features into the binary classifier, which will determine whether it is or is not the object we are looking for. Figure 2.1 illustrates this process for a car detection system (Zimmerman 2012).

There are two subtleties that need to be examined closer when applying the sliding window detector. The first being that not all of the objects in the image will be the same size as the $n \times m$ image patches that the classifier was trained on. To remedy this issue, we can slide the windows across different scales of the same image (Forsyth 2012). The second subtlety deals with the response of the classifier, if the classifier response is above a specified threshold it will classify that window as a detected object. However depending on the distance between each window, we can have multiple windows that detect the same object, this can be visualized in figure 2.2. In order to handle this issue we would apply a method called Non-Maximum Suppression (Forsyth 2012). Non-Maximum Suppression is a method that looks at local responses of windows classified as a detected object. If we have multiple windows that are overlapping, where the overlapping area is above a certain threshold, we will only retain the window with the highest response. To summarize, figure 2.3 shows the sliding window detection algorithm (Forsyth 2012).

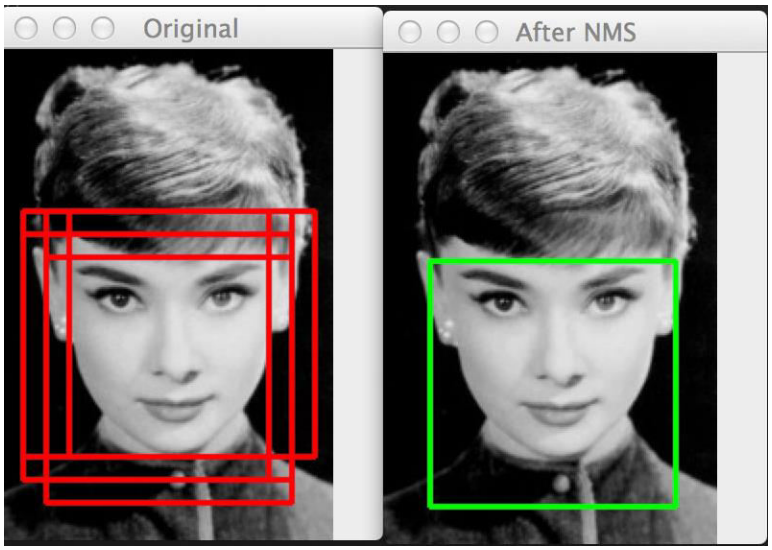

Figure 2.2: Example of Non-Maximum Suppression

Train a classifier on $n \times m$ image windows. Positive examples contain the object and negative examples do not.
Choose a threshold t and steps Δx and Δy in the x and y directions

Construct an image pyramid.

For each level of the pyramid
  Apply the classifier to each $n \times m$ window, stepping by Δx and Δy, in this level to get a response strength c.
  If $c > t$
    Insert a pointer to the window into a ranked list $\mathcal{L}$, ranked by c.

For each window $\mathcal{W}$ in $\mathcal{L}$, starting with the strongest response
  Remove all windows $\mathcal{U} \neq \mathcal{W}$ that overlap $\mathcal{W}$ significantly,
    where the overlap is computed in the original image by expanding windows in coarser scales.

$\mathcal{L}$ is now the list of detected objects.

Figure 2.3: Sliding Window Detection Algorithm

2.1 CurrentState-of-the-Art Performance

Mogelmose (2015) has shown that Integral Channel Features (ICF) and Aggregate Channel Features (ACF) have achieved state-of-the-art performance for U.S traffic signs on the LISA dataset. In this section we will provide a detailed description of Mogelmose's (2015) methodology, which we will use as the baseline measurements of this work. The main focus of Mogelmose's (2015) methodology is in the feature extraction stage. The features used were Integral Channel features, which were first introduced by dollar (2009) for pedestrian detection. Dollar later introduced an extension of the Integral Channel features, which were the Aggregate Channel Features (Dollar 2014). The features were trained using an AdaBoost Classifier. This section will provide all of the necessary background information for these topics.

2.1.1 Integral Images

In order to understand the specific features used by Mogelmose (2015), we must give some background information on integral images, which were first introduced by Viola and Jones (2001). An integral image is a quick way of calculating rectangular features by summing the pixel values in a given rectangular region of an image. An integral image is defined at each pixel by the following equation (Viola and Jones 2001):

$$\boldsymbol{ii(x,y)} = \sum_{\boldsymbol{x'\leq x, y'\leq y}} \boldsymbol{i(x',y')} \qquad \textbf{(1)}$$

Where *ii(x,y)* is the integral image and *i(x,y)* is the original image. Equation 1 tells us that the integral image at pixel location *x* and *y* is the sum of the all of the pixel values above and to the left of the original image. We can calculate the entire integral image in one pass over of the original image by using the following two equations:

$$\boldsymbol{s(x,y)} = \boldsymbol{s(x,y-1)} + \boldsymbol{i(x,y)} \qquad \textbf{(2)}$$

$$\boldsymbol{ii(x,y)} = \boldsymbol{ii(x-1,y)} + \boldsymbol{s(x,y)} \qquad \textbf{(3)}$$

Where *s(x,y)* is the cumulative row sum of the original image. Using an integral image we can compute and sum or difference of any rectangular area, this is illustrated in figure 2.4 (Viola and Jones 2001).

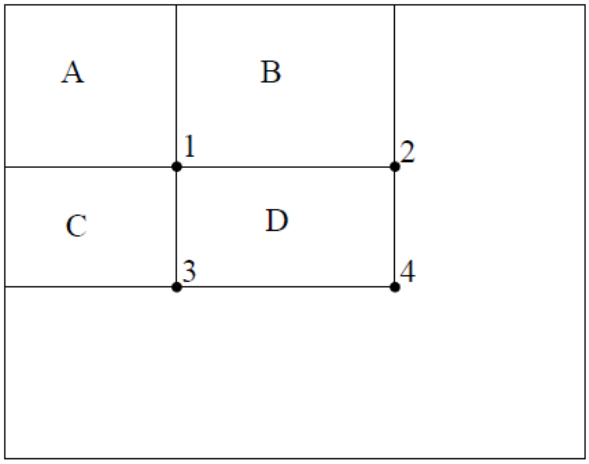

Figure 2.4: Integral Image Example

If we want to find the sum of the pixels in the rectangle D, in figure 2.4, we only need to use the four reference points: 1, 2, 3, and 4. Where the value at reference point 1 is the sum of the pixels in rectangle A. The value at reference point 2 is the sum of rectangles A and B. At reference point 3 the value is A+C. We can then compute the sum within D as $4 + 1 - (2 + 3)$. This gives us the ability to calculate the sum in constant time, O(1). With the knowledge of integral images we can now examine, in more detail, the Integral Channel Features used in Mogelmose (2015).

2.1.2 Integral Channel Features

The idea behind ICF is to create multiple image channels by making linear and non-linear transformations of the original input image. The channels are then transformed into integral image. This allows for more efficient feature extraction through the integral images. For a given input image *I*, a channel can be viewed as a new image where the original input pixels are mapped to the new image channel, *C*. These mappings can be either linear or non-linear transformations and can be represented by the channel generating function **Ω**. The transformation can be written:

$$C = \Omega(I) \quad (4)$$

There are multiple channel features that can be extracted from a transformed image channel, some that are more complex and longer to compute than others. One of the fastest to compute is a first-order channel feature, which is the sum of pixels in a fixed rectangular region. We have shown in the previous section that if the integral image is generated we can calculate the sum of a rectangular region in constant time, O(1), as it only requires operations on 4 reference values. A higher-order channel feature, is defined as a feature that is computed using multiple first-order features. An example of a higher-order feature is a Haar feature (Viola Jones 2001), which involves a sum of 2-4 rectangles arranged in patters that compute first and second order image derivatives at multiple scales.

Before we extract the first-order or higher-order channel features, we must first transform out image into channels. Figure 2.5 shows an example of different types of channels that were transformed from the original input image. Each channel contains different pieces of information from the input image. For example the Gabor channels, in figure 2.5, are a convolution of 4 oriented Gabor filters with the input image. There are many different types of transformations that can performed on the input image. One such transform is a linear filter where we convolve a linear filter, such as the Gabor filter, with the input image. We also have non-linear transformations, such as gradient magnitudes, which looks at specific edge strengths

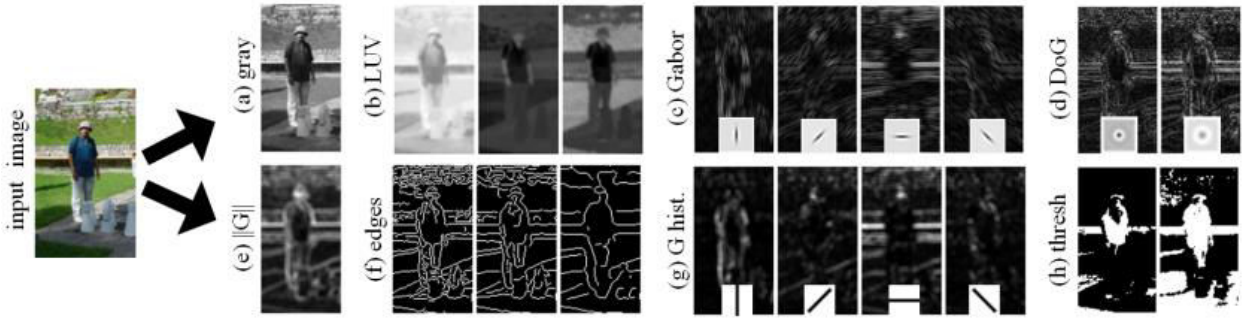

Figure 2.5: Example of Image Channels

in the image, see figure 2.5(e). There are also gradient histogram channels, which is a weighted histogram where the gradient angle is the index of the bin and the gradient magnitude at that angle is the magnitude of the bin. We can express the channels mathematically as:

$$\boldsymbol{Q_\theta(x,y) = \mathbf{G(x,y)} * \mathbf{1}[\Theta(x,y) = \theta]} \quad \textbf{(5)}$$

$\boldsymbol{Q_\theta(x,y)}$ is the gradient histogram channel for angle $\boldsymbol{\theta}$, where $\mathbf{G(x,y)}$ and $\mathbf{\Theta(x,y)}$ are the gradient magnitude and quantized gradient angle of $\mathbf{I(x,y)}$, respectively.In chapter 3 we will discuss the specific channels and features used for our baseline results.

They were designed to provide a faster alternative to ICF, but they have also shown to have slightly better detection performance when compared to ICF, in some cases. ACF uses the same principle idea of ICF, which is computing features across different image channels and in this work the channels are actually the same channels used in ICF. The ACF method proposed by dollar (2014) for pedestrian detection can be visualized in figure 2.6.

The first step to this process is the same as the Integral Channel Features, where we map the image into channels with some mapping function, $C = \Omega(I)$. Then next step is to sum

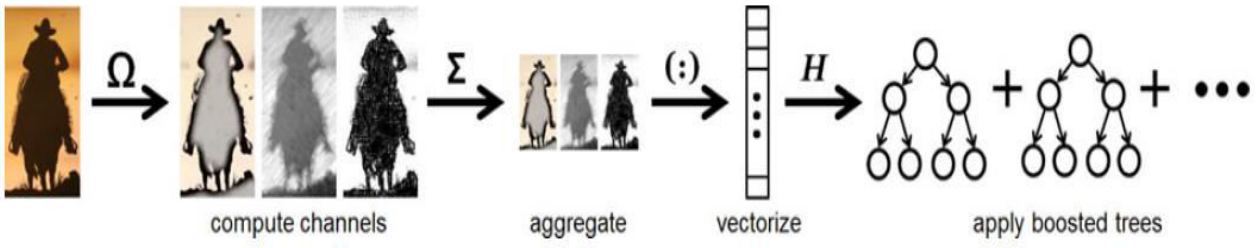

Figure 2.6: Aggregate Channel Framework

2.1.3 Aggregate Channel Features

Aggregate Channel Features (ACF) were proposed as an extension of ICF by dollar (2014). each block of pixels into an integral image. The features are even simpler than ICF, they are just summed blocks of pixels at various scales and

stored in a feature vector. The classifier is learned from the feature vectors using AdaBoost.

2.1.4 AdaBoost Classifier

The final part of Mogelmose's (2015) method is the classifier. Their method uses machine learning technique called, AdaBoost. This section will give a description of the AdaBoost technique. Freund and Schapire (1996) discovered that it was possible to build a strong classifier from a weighted sum of weak classifiers, which they called boosting. They defined a weak classifier as a classifier that is just better than random guessing, meaning that the accuracy of the classifier is greater than 50%. The weights of each classifier are learned by adjusting the weights of the wrongly classified samples. Each sample that is incorrectly classified, by the current weak classifier, will have a higher weight on the next iteration of the algorithm. The pseudocode for the AdaBoost algorithm can be viewed in figure 2.7.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 2.7: AdaBoost Algorithm

We can see from figure 2.7 that we have m labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$. Where the label values $y_i \in \{-1, +1\}$, where -1 will represent negative samples and +1 will represent positive samples. The first step is to initialize our distribution D_t of our samples, which is first initialized as each sample equally weight and the sum of the weights of all the samples are equal to 1. In the algorithm we iterate from t=1,…,T, where T is the number of weak classifiers. On each iteration we are trying to find the weak classifier, $h_t(x)$, that is going to give us the smallest error rate. This is typically done by using a linear decision stump. Also in the same iteration we want to find the new distribution and the weight of the weak classifier. The following equation is used to calculate the weak classifier weight, α_t:

$$\boldsymbol{\alpha_t = \frac{1}{2} * ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)} \quad \textbf{(6)}$$

Where,

$$\boldsymbol{\varepsilon_t = \sum_{i \epsilon\, h_t(x_i) \neq y_i} D_t(i)} \quad \textbf{(7)}$$

Equation 7 is just stating that the error rate of the weak classifier is equal to the sum of the misclassified sample weights. The next step of the algorithm is to update all of the sample weights by using the following equations:

$$\boldsymbol{D_{t+1}(i) = \frac{D_t * exp(-\alpha_t y_i h_t(x_i))}{Z_t}} \quad \textbf{(8)}$$

Where Z_t is the normalization factor, this will result in the final strong classifier H(x) seen below:

$$\boldsymbol{H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)} \quad \textbf{(8)}$$

2.2 Deep Learning

In this section we will discuss, in detail, the main components of deep learning. We will start with the basic neuron and feed forward neural networks. We will then proceed to discuss the different components of Convolutional Neural Networks (CNNs), which will be the architecture proposed in this work.

2.2.1 Feed Forward Neural Network

The original inspiration for Neural Networks came from the attempt to model the biological neural systems. The neuron is the basic computational unit of the nervous system. The nervous system is composed of billions of interconnected neurons. A single neuron receives

$$y(x, b) = f\left(\sum_i w_i x_i + b\right) \quad (9)$$

many input signals from the dendrites and sends a single output signal along the axon. Where the axon branches to multiple synapses that connect to other dendrites, of other neurons. Based off of this biological knowledge, the computational model was created. Where the signal (x_0) coming from the axiom, of a previous neuron, is multiplied to the weights (w_0) of the synapses. The signals of all the synapses are summed and sent to an activation function (f), which then sends a single signal back out the axiom. What the model tries to learn are the weights of the synapses. We can visualize the biological neuron and the computational model of the neuron below in figure 2.8 (Li 2015).

The computational model output of a single neuron can be expressed mathematically as:

There are many different types of activations functions that can be used, as the output axiom, but the one that has become the most popular is the Rectified Linear Unit (ReLU). The ReLU is simply a ramp function and expressed as:

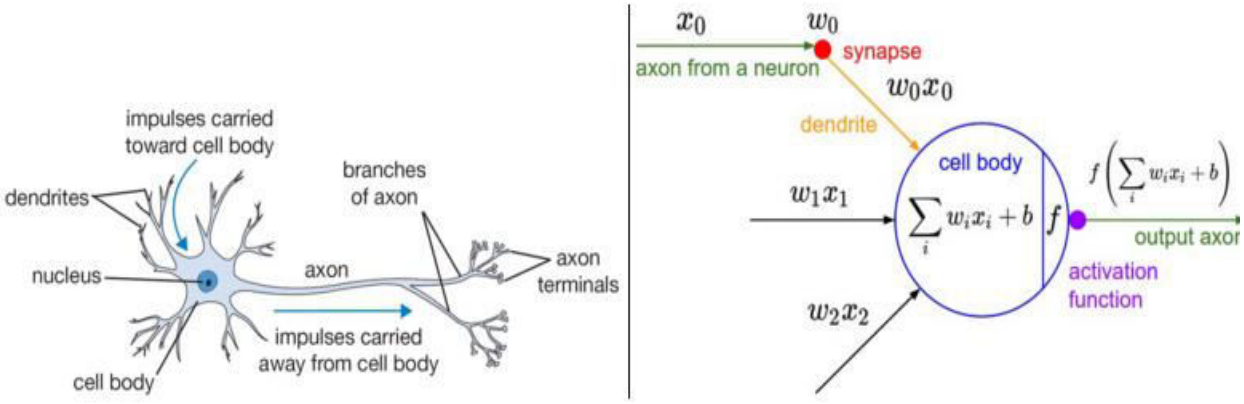

Figure 2.8: Biological Neuron (left) and Computational Model (right)

$$f(x) = max(0,x) \quad (10)$$

A complete Neural Network is an interconnection of the modeled neurons. Where the output of one neuron will be the input of another. If we have multiple neurons then all of these outputs can be inputs of a single neuron. However, this should not be interpreted as a big blob of neurons, but as neurons connected in layers. We can visualize this in figure 2.9 (Li 2015).

$$MSE(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - nn(x_i,w))^2 \quad (11)$$

discussing an N-layer Neural Network the input layer is excluded. The architecture or size of a Neural Network is defined by two measures, how many layers we have and how many neurons are in that layer.

Now that we have a general idea of the architecture we need to examine how to optimize the parameters for our data. Once the architecture is decided upon we need to decide which parameters will minimize the loss (or error) function. Typically the loss function is defined by the mean square error (Bishop 2007):

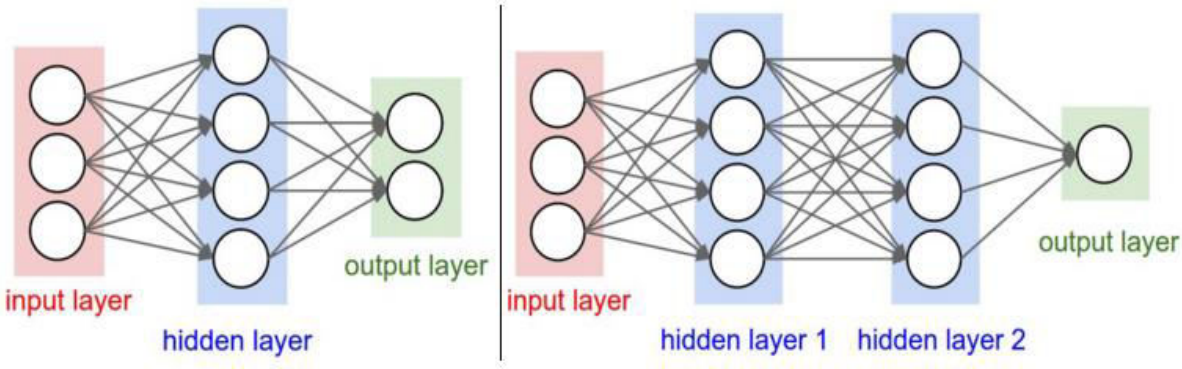

Figure 2.9: Fully Connected Feedforward Neural Network

On the left of figure 2.9 we have a 2-layer Neural Network and on the right we have a 3-layered Neural Network, typically when

Where w, is a vector of the parameters of the Neural Network architecture ($nn(x_i,w)$); x_i and y_i are the feature vector input and the feature vector labels respectively; and n is the number of

training examples used. As stated before the goal is to find the parameters, w, that will minimize the MSE:

$$\boldsymbol{arg\ \min_{w}\quad MSE(w)} \tag{12}$$

In order to minimize the mean square error there are many different numerical methods we can choose however the most popular is the gradient decent algorithm. The idea of gradient decent is to find the direction of steepest decent in a local area. The first step is to initialize all of our parameters. After they are initialized, we will update the each parameter by taking the derivate of the mean squared error with respect to each parameter and subtracting it from the original parameter value. The parameter update equation can be seen below:

$$\boldsymbol{w^{t+1} = w^{t} - \alpha * \nabla MSE(w^{t})} \tag{13}$$

Where α is called the learning rate and can be thought of as the size of the step we want to take in our local area. Also to note, one update of the parameters is called an epoch, and when implementing it practically we will limit the number of epochs or stop updating the parameters if it converges, meaning:

$$\boldsymbol{\left|w^{t+1} - w^{t}\right| < \varepsilon} \tag{14}$$

Where ε is a very small value. By using gradient decent we will be able to learn all of the parameters, which will optimize our model to minimize the model's MSE.

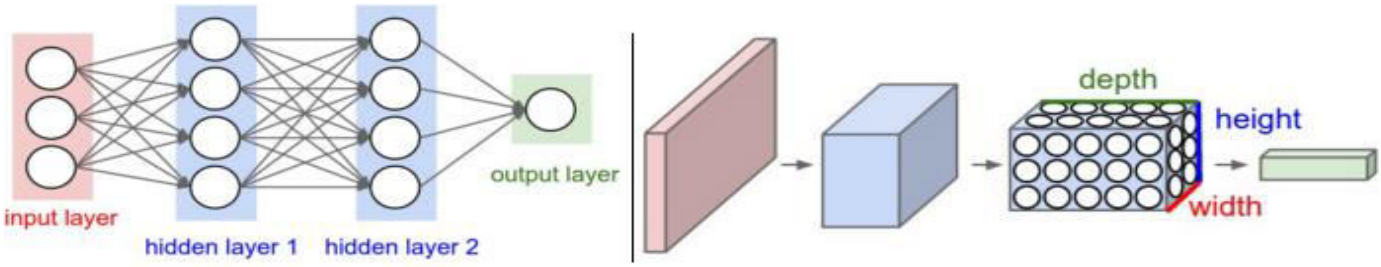

Figure 2.10: Neural Network (left) and Basic Neuron Arrangement of CNN (right)

2.2.2 Convolutional Neural Network

Similar to Neural Networks, Convolutional Neural Networks (CNNs) are made up of neurons that have learnable weights. They

have a loss function and even the last layer of the CNN can be thought of as a fully connected neural network. However Convolutional Neural Networks are specialized for processing data with a grid-like topology, such as images. The difference between the CNN and a Neural Network is that the layers of a CNN have neurons arranged in 3 dimensions, width, height, and depth. The high dimensionality of the image data makes it impossible to use a single fully connected neural network. We will see that through the architecture of the CNN we will be able to reduce the full image into a single neuron. A visualization of the neuron arrangement of a CNN can be seen below in figure 2.10 (Li 2015).

The CNN's architecture has three main layers: the Convolutional Layer, the Pooling Layer, and the Fully-Connected Layer. A basic CNN architecture may be structured with the input image connected to the Convolutional layer which is connected to a Pooling layer and finally connected to a Fully-Connected layer. In the remainder of this section we will give a description of each layer.

The first layer we are going to discuss is the convolutional layer, which is the main building block of the CNN. The convolutional layer as the name suggests, applies multiple convolutions to the input image with filters that are learned. The filters will slide across the image's width, height and then depth, which generates a new image. The filters are stacked on top of each other giving the depth dimension that we see in figure 2.10. Intuitively, we want to learn these filters so that when they see a specific feature they will activate a response. The reason for these filters is so the network can create a local connectivity of neurons instead of having learnable weights at each pixel. For example if we have a 32x32x3 image, where the depth is 3 for the RGB channel, if we assigned learnable weights to each pixel we would have a 3072 parameters that need to be learned. However if we have a convolutional filter that is 5x5 now we only have to learn 5x5x3=75 parameters. Note that the depth is three because we must apply the filter to each channel. This also gives us the ability to use stack multiple filters at each convolutional layer and still have less parameters

to learn. The number of filters, or the depth, that we use can be analogous to the number of neurons in a hidden layer of a Feedforward Neural Network.

The next layer we will discuss is the Pooling/Sub-sampling layer. The Pooling layer will be found between successive Convolutional layers. The Pooling layer is needed to reduce the spatial size of the output of the convolutional layer. The Pooling layer is applied to every depth image generated from the Convolutional layer. The most common form of pooling is the max 2x2 filter, which downsamples the input image by taking the max value of every 2x2 block as it slides along the height and width of the image. There are two specific hyperparameters associated with the Pooling layer. The first is the spatial extent, F, which is the size of the downsampling filter. The second is the stride, which is the distance between the filters as it moves from left to right and top to bottom of the image. The pooling process can be seen below in figure 2.11 (Li 2015).

On the left of figure 2.11 we can see how the Convolutional Layer is downsampled to half the size of the original input image, through the pooling process. On the right of the figure we see how the down sampling is performed using max pooling with 2x2 filters with a stride of 2. A complete architecture can be visualized in figure 2.12.

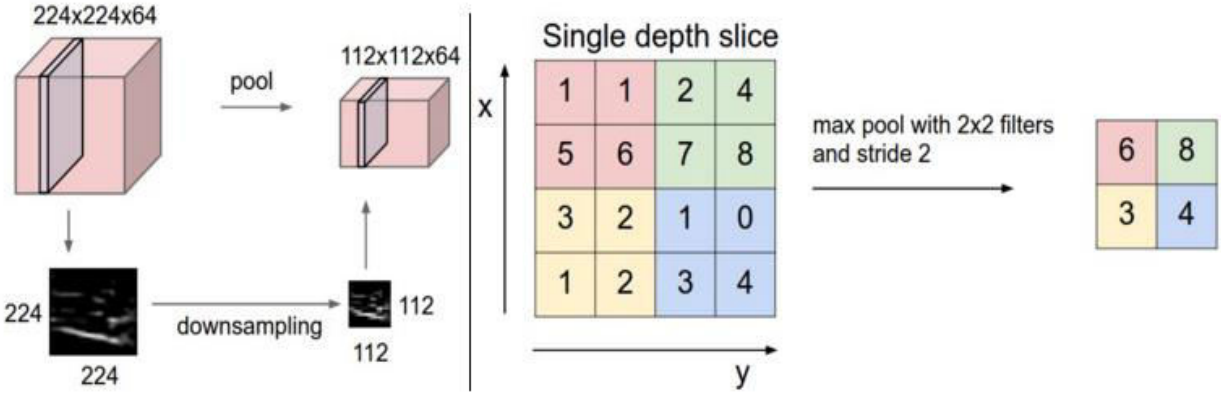

Figure 2.11: Pooling Process

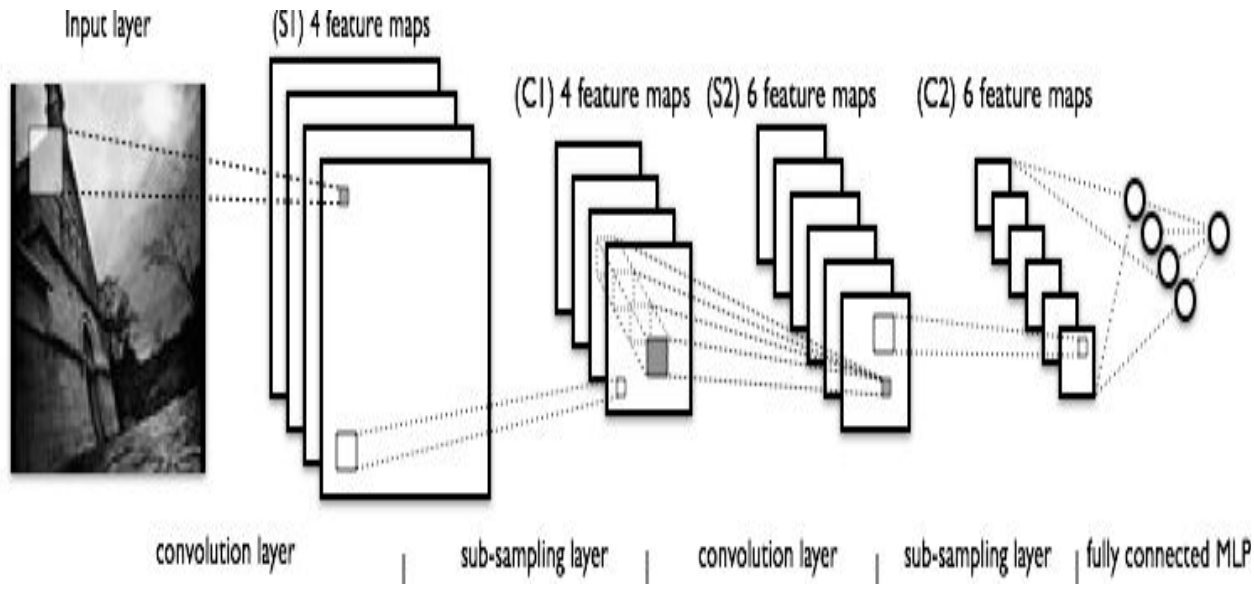

Figure 2.12: Full Convolutional Neural Network Architecture

CHAPTER 3

DATA & EXPERIMENTS

3.1 Data

In this section we describe the details of the dataset used for this experiment. The dataset used for the detection of U.S. traffic signs is the LISA dataset, the largest collection of publicly available U.S traffic sign images.

3.1.1 LISA Dataset

For computer vision and machine learning applications the dataset is one of the most important components of the system. When a dataset is made publicly available, it not only helps to save time and energy but it also helps to compare state-of-the-art results, since the performance is evaluated on the same data. For this reason the LISA dataset was created (Mogelmose 2012). Although there are datasets publicly available for TSR, most of these datasets are for European, or Vienna Convention, traffic signs. The LISA dataset was created exclusively for U.S traffic signs. As there were no large existing U.S traffic sign datasets publicly available at the time of its creation.

The LISA dataset was developed by the Laboratory for Intelligent and Safe Automobiles at the University of California, San Diego

(UCSD) in 2012. When it was first introduced the dataset contained over 6000, fully annotated, images contain U.S traffic signs from in-car video cameras around the San Diego area. The dataset also has approximately 10,000 negative images, that don't contain traffic signs, but are still captured from an in-car video camera. The image sizes vary from 640x480 to 1024x522 pixels. The type of traffic signs vary over 49 different classes of traffic signs and three so-called Superclasses which are: warning signs, speed limit signs and no-turn signs.

The same authors of the original LISA dataset made an extension to this dataset in 2015 (Mogelmose 2015). This extension almost doubled the amount of positive annotated high-resolution images. The extended images were also collected in and around the San Diego, California area. It is important to note that the weather conditions, during image acquisition, were generally dry and sunny. No adverse weather conditions were captured in this dataset. The hardware used to capture the images, for the extended dataset, was a Point Grey FL3 color camera with a resolution of 1280x960 pixels. It is also important to note that the extended dataset split the data into a training and testing set. Table 3.1 (Mogelmose 2015) shows the statistics of the original LISA dataset the extended dataset with a

Original LISA Traffic Sign Dataset		**LISA-TS Extension**			**GTSD**
		Training Set	**Testing Set**	**Combined**	
Number of classes:	47	31	17	35	4
Number of annotations:	7855	3672	2422	6094	1206
Number of images:	6610	3307	2331	5638	900
Sign Sizes, longest edge:	6 – 168 pixels	23–222 pixels	25–222 pixels	23–222 pixels	16-128 pixels
Image Sizes:	640x480 to 1024x522	1280x960	1280x960	1280x960	1360x800
Videos Included:	Yes	Yes	Yes	Yes	No

Table 3.1: Traffic Sign Dataset Statistics

comparison to

Superclasses			
461	warning	679	speed limit
82	noTurn		
Detailed classes			
13	curveRight	40	signalAhead
17	dip	406	speedLimit25
21	doNotEnter	264	speedLimit30
11	keepRight	9	speedLimit45
37	merge	1151	stop
29	noLeftTurn	86	stopAhead
53	noUTurn	43	turnRight
80	pedestrianCrossing	145	warningUrdbl
17	school		

In total: 2422 sign annotations

Table 3.4: LISA Dataset Extension (Testing) Breakdown by Class

11	rampSpeedAdvisory20	5	thruMergeLeft
5	rampSpeedAdvisory35	7	thruMergeRight
3	rampSpeedAdvisory40	19	thruTrafficMergeLeft
29	rampSpeedAdvisory45	60	truckSpeedLimit55
16	rampSpeedAdvisory50	32	turnLeft
3	rampSpeedAdvisoryUrdbl	92	turnRight
77	rightLaneMustTurn	236	yield
53	roundabout	57	yieldAhead
133	school	21	zoneAhead25
105	schoolSpeedLimit25	20	zoneAhead45
925	signalAhead		

In total: 7855 sign annotations

Table 3.2: Original LISA Dataset Breakdown by Class

the German Traffic Sign Detection (GTSD) dataset.

One important component of object detection is the aspect ratio of the objects in the data. Many of the traffic signs, in the LISA dataset, have an aspect ratio of approximately 1.0. The remaining traffic signs are between 1.2 and 0.8, which gives us a lot of insight into the design of our object descriptor. With this knowledge we will be more inclined to have a square detection box when we implement the sliding window detector, which will be discussed later. For more specific details of the type of signs and classes in the LISA dataset refer to Tables 3.2-3.4.

3.2 Experimental Setup

The proposed work aims to do a comparative study of the state-of-the-art methods verses that of deep learning architecture, for traffic sign detection. This section describes the specifics of the detailed setup and methodology used for the baseline results.

3.2.1 Data Preparation

The first step of any machine learning or computer vision task is to first prepare the data. The data preparation step taken in this work are similar to that of Mogelmose (2015). When preparing the data, we split the dataset into a training, validation, and test set. This was done for both the original LISA dataset and the extensions. Table 3.5 shows how the positive and negative samples were separated.

It is also important to note that the original LISA-TS dataset contains 11634 negative color images of size 704x480 pixels. The negative images are image taken from an in-car video camera but do not have any traffic signs in the scene. To get the negative samples random image patches are extracted from the negative images at different scales of the images. It is required for the positive and negative sample image patches to be the same size. The sample image patch sizes that are used in this work are 48x48 pixels.

After the separation of the data we want to preprocess each of the samples. The samples are preprocessed using a contrast-limited

Superclasses			
1232	warning	752	speed limit
184	noTurn		
Detailed classes			
2	addedLane	91	school
20	bicyclesMayUseFullLane	428	signalAhead
50	curveLeft	4	speedBumpsAhead
59	curveRight	17	speedLimit15
39	doNotEnter	259	speedLimit25
11	intersection	90	speedLimit30
24	intersectionLaneControl	158	speedLimit35
127	keepRight	92	speedLimit40
47	laneEnds	80	speedLimit45
6	leftAndUTurnControl	53	speedLimit50
18	merge	3	speedLimit60
73	noLeftAndUTurn	1181	stop

	LISA-TS		LISA-TS Extension		Total	
	Positive	**Negative**	**Positive**	**Negative**	**Positive**	**Negative**
Training	6284	12000	2939	6000	9223	18000
Validation	1571	3000	735	1500	2306	4500

Table 3.5: Number of Samples used for Training and Validation

histogram equalization (CLAHE) (dollar 2014). CLAHE is an adaptive histogram equalization method that reduces the excessive contrast and noise that may arise when performing an ordinary histogram equalization. Ordinary histogram equalization looks at the CDF of each pixel value and attempts map new pixel value so that the PDF, of the pixel values, becomes uniform. CLAHE does something similar however instead of looking at all of the pixel values in the image it only considers the CDF of the pixels in a nearby tile. Resulting in a local contrast enhancement, which reduces the noise that would normally be enhanced across the whole image. Once CLAHE is finished, the data will be properly separated and preprocessed, which will then allow for feature extraction.

3.2.2 Feature Extraction and Training

Once the data has been separated and preprocessed, the next stage is feature extraction. The features that are extracted are Integral Channel Features and Aggregate Channel features, as was proposed by Mogelmose (2015). As described above the first step of ICF is to compute the different channels from the input image. The features are computed over 10 different channels, these channels are the following: three LUV color space channels, one unoriented gradient magnitude channel, and six gradient channels in varying directions. Each channel is transformed into an integral image channel, in order to speed up the computations of the features. First-order Haar-like features are computed for each of the 10 different integral channels. The Haar-like features are simply the difference of the sum of two rectangular image filters. After the features are computed, they are trained using an AdaBoost classifier with 200 depth-2 decision trees as weak learners.

Aggregate Channel Features are similar to ICF, however they use even simpler features in order to speed up the computations. The initial process of transforming the input image into 10 different integral channels is the same. In fact the channels used in ICF are the same ones used for ACF. However, the difference between the two methods is that the features ACF uses, are a summing up of blocks of pixels at various scales. Similarly after the features are computed, they are

trained using an AdaBoost classifier with 200 depth-2 decision trees as weak learners. The ICF model and the ACF model are the two models used as our baseline comparisons.

3.2.3 Convolutional Neural Network Setup

The purpose of this work is to do a comparative study of U.S traffic sign detection. We will compare the current state-of-the-art method with a deep learning architecture. The deep learning architecture is going to be a Convolutional Neural Network (CNN). The big difference between the deep learning architectures and knowledge based architectures, like the ICF model and the ACF model, is that we eliminate the feature extraction stage. We instead learn the features through the different filters in the convolutional layers of the CNN. Unlike the data preparation discussed in the previous section, we will not have to preprocess the samples or extract the features. Instead, we will only pass the positive and negative image patches to the CNN for training.

There are many design aspects when creating a CNN architecture, such as the type of activation function to use, the number of filters or neurons in the convolutional layer, the type of pooling in the pooling layer, the number of neurons in the fully connected output layer, and number of convolutional and pooling layers used. In our study we will examine multiple architectures but we will keep a couple of components constant. The activation functions used will consistently be a rectified linear unit (ReLU). Also the pooling layer will be a 2x2 max pooling filter with a stride of 2, so that the pooling filter is not overlapping. Instead we will experiment with the depth of the architecture, in other words we will vary the number of convolutional layers and pooling layers. The number of filters or neurons in the convolutional layer will be experimented with. As well as, the number of neurons in the fully connected layer. The performance of each architecture will be examined and compared with the ICF and ACF models. The baseline detection performance of the ICF and ACF models, proposed by Mogelmose (2015), will be discussed in the next chapter.

$$a_0 \equiv \frac{area(BB_{dt} \cap BB_{gt})}{area(BB_{dt} \cup BB_{gt})} > 0.5 \quad (15)$$

Preliminary Experiments & Results

4.1 Evaluation

This section discusses how the detection performance will be evaluated. The standard measure for object detection tasks is the PASCAL measure, which was introduced in (Everingham, 2010). All of our models will be evaluated with the PASCAL measure. The PASCAL measure is used to determine whether one of the positively detected bounding boxes, in the test image, is a true positive. This is determined by the following equation:

Where BB_{dt} is the detected bounding box and BB_{gt} is the ground truth bounding box. Equation 15 tells us that if the overlap between the ground truth and the detected bounding boxes is greater than 50% then the detected bounding box will be considered a true positive.

The PASCAL measure is a measure of true positives however this is not a measure of the overall performance of the system. The detection performance of the system is evaluated by the area under the curve of a precision recall graph. Precision is defined as the percentage of retrieved items, or in our case detection boxes, that are actually relevant. Recall is defined as the percentage of relevant items that are detected or recovered. We can formally define them with the following equations:

Where *TP* is the total number of true positives, *FP* is the total number of false positives, and *FN* is the total number of false negatives.

4.2 Baseline Results

In this section we will discuss the detection results presented by Mogelmose (2015), which will be replicated in our work. The previous section explained how detection performance is evaluated by the area under the curve of the precision recall graph. The precision recall curves evaluated for ICF and ACF models presented by Mogelmose (2015), for both the LISA and GTSD datasets, can be seen below in figure 4.1 and

figure 4.2 respectively.

$$Precision = \frac{TP}{TP + FP} \quad (16)$$

$$Recall = \frac{TP}{TP + FN} \quad (17)$$

3.CONCLUSION

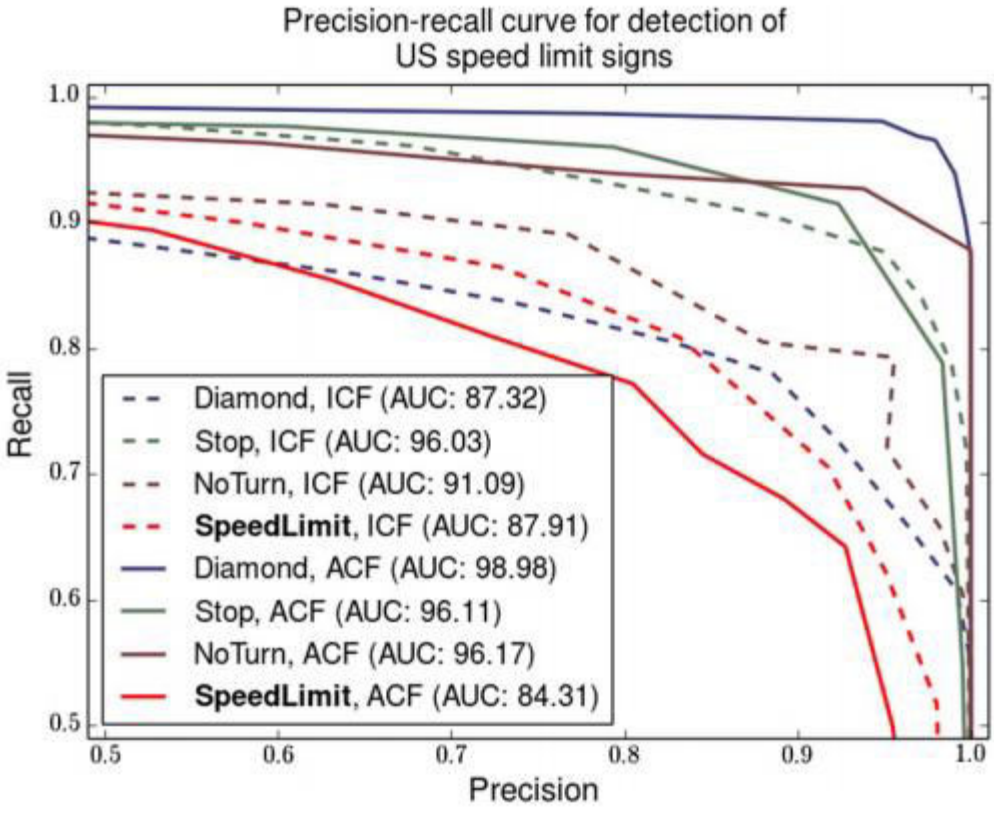

Figure 4.1: Precision-Recall Curves for Detection of U.S Traffic Signs

	LISA Dataset					GTSD Dataset			
	Speed Limit	**Warning**	**No Turn**	**Stop**	**Mean AUC**	**Prohibitory**	**Danger**	**Mandatory**	**Mean AUC**
ACF	84.31	98.98	96.17	96.11	93.89	99.86	100.00	98.38	99.41
ICF	87.91	87.32	91.09	96.03	90.59	99.58	99.15	98.52	99.08
8bit-MCT AdaBoost (Lim 2014)	81.43	21.22	57.63	65.81	56.52	N/A	N/A	N/A	N/A
Boosted_ICF (Houben 2013)	N/A	N/A	N/A	N/A	N/A	100.00	100.00	96.98	98.99
HOG + SVM (Liang 2013)	N/A	N/A	N/A	N/A	N/A	100.00	98.85	92.00	96.95
HOG_LDA_SVM (Wang 2013)	N/A	N/A	N/A	N/A	N/A	100.00	99.95	93.99	97.98

Table 4.1: State-of-the-Art Detection Performances on the LISA and GTSD Datasets

Precision

Figure 4.2: Precision-Recall Curves for Detection of European Traffic Signs

Mogelmose demonstrated the detection performance for the U.S traffic signs was slightly worse than the performance on European Traffic signs. In this work we want to do a comparative study of a deep learning network versus other knowledge based representations. To do this we need performance evaluations of different methods. Table 4.1 provides a performance summary of the top performing algorithms for both the LISA and GTSD datasets.

References

Bishop, C. (2007). "*Pattern Recognition and Machine Learning*" (2nd ed). New York, New York, USA: Springer.

Dalal, N. and Triggs, B. (2005) "Histograms of oriented gradients for human detection," *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, San Diego, CA, USA, 2005, pp. 886-893 vol. 1. doi: 10.1109/CVPR.2005.177

Dollár, P., "Piotr's Computer Vision Matlab Toolbox (PMT)." [Online]. Available: http://vision.ucsd.edu/\ignorespacespdollar/toolbox/doc/index. html

Dollar, P., Appel, R., Belongie, S., and Perona, P. (2014), "Fast Feature Pyramids for Object Detection," in *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, vol. 36, no. 8, pp. 1532-1545, Aug. 2014. doi: 10.1109/TPAMI.2014.2300479

Dollár, P., Tu, Z., Perona, P., and Belongie S. (2009), "Integral channel features," in Proc. BMVC, vol. 2, no. 3, p. 5, 2009

Everingham, M., Gool, L., Williams, C., Winn, J., and Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision* 88, 2 (June 2010), 303-338. DOI=http://dx.doi.org/10.1007/s11263-009-0275-4

Freund, Y. and Schapire, R.E. (1996), ``Experiments with an new boosting algorithm, Machine Learning: Proceedings of the Thirteenth International Conference, Morgan Kauffman, SanFrancisco, pp.148-156 (1996). Introduced AdaBoost

Friedman, J., Hastie, T., and Tibshirani, R. (2000), "Additive LogisticRegression: A Statistical View of Boosting," Annals of Statistics,vol. 38, no. 2, pp. 337-374, 2000.

Forsyth, D. and Ponce, J. (2012). *Computer Vision a Modern Approach* (2nd ed., p. 549). New York, New York, USA: Pearson

Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., and Igel, C. (2013), "Detection of traffic signs in real-world images: The German traffic sign detection benchmark," *Neural Networks (IJCNN), The 2013 International Joint Conference on*, Dallas, TX, 2013, pp. 1-8. doi: 10.1109/IJCNN.2013.6706807

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012), Imagenet classification with deep convolutional neural networks. In NIPS, 2012. 1, 2, 3, 4

Li, F., Karpathy, A., and Johnson, J. (2015), "CS231n: Convolutional Neural Networks for Visual Recognition," www.http://cs231n.stanford.edu/

Liang, M., Yuan, M., Hu, X., Li, J., and Liu, H. (2013) "Traffic sign detection by ROI extraction and histogram features-based recognition," in Proc. IEEE IJCNN, Aug. 2013, pp. 1–8.

Lim, K., Lee, T., Shin, C., Chung, S., Choi, Y., and Byun, H., (2014). Real-time illumination-invariant speed-limit sign recognition based on a modified census transform and support vector machines. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication* (ICUIMC '14). ACM, New York, NY, USA, , Article 92 , 5 pages. DOI=http://dx.doi.org/10.1145/2557977.2558090

Liu, W., Wu, Y., Lv, J., Yuan, H., and Zhao, H. (2012) "U.S. speed limit sign detection and recognition from image sequences," *Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on*, Guangzhou, 2012, pp. 1437-1442. doi: 10.1109/ICARCV.2012.6485388

Loy, G. and Barnes, N. (2004), "Fast shape-based road sign detection for a driver assistance system," in *Proc. IEEE/RSJ Int. Conf. IROS*, 2004, vol. 1, pp. 70–75

Mogelmose, A., Liu, D., and Trivedi, M. M. (2015), "Detection of U.S. Traffic Signs," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3116-3125, Dec. 2015. doi: 10.1109/TITS.2015.2433019

Mogelmose, A., Liu, D., and Trivedi, M. M. (2014), "Traffic sign detection for U.S. roads: Remaining challenges and a case for tracking," *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, Qingdao, 2014, pp. 1394-1399. doi: 10.1109/ITSC.2014.6957882

Mogelmose, A., Trivedi, M. M., and Moeslund, T. B. (2012), "Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver

Assistance Systems: Perspectives and Survey," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484-1497, Dec. 2012. doi: 10.1109/TITS.2012.2209421

Staudenmaier, A., Klauck, U., Kreßel, U., Lindner, F., and Wöhler, C. (2012) "Confidence measurements for adaptive Bayes decision classifier cascades and their application to US speed limit detection," in Proc. Pattern Recognit., ser. Lecture Notes in Computer Science, vol. 7476, pp. 478–487, 2012.

Vázquez-Reina, A., Lafuente-Arroyo, S., Siegmann, P., Maldonado-Bascón, S., and Acevedo-Rodríguez, F. (2005), "Traffic sign shape classification based on correlation techniques," in *Proc. 5th WSEAS Int. Conf. Signal Process., Comput. Geometry Artif. Vis.*, 2005, pp. 149–154.

Viola, P. and Jones, M. (2001), "Rapid object detection using a boosted cascade of simple features," *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001, pp. I-511-I-518 vol.1. doi: 10.1109/CVPR.2001.990517

Wang, G., Ren, G., Wu, Z., Zhao, Y., and Jiang, L. (2013), "A robust, coarse-to- fine traffic sign detection method," in Proc. IEEE IJCNN, Aug. 2013, pp. 1–5.

United Nations (1978), "Vienna Convention on road signs and signals,"

Zimmerman, A (2012), "Category-level Localization", Visual Geometry Group University of Oxford, http://www.robots.ox.ac.uk/~vgg