2017

Symbolic and Deep Learning Based Data Representation Methods for Activity Recognition and Image Understanding at Pixel Level

Manohar Karki
Louisiana State University and Agricultural and Mechanical College

SYMBOLIC AND DEEP LEARNING BASED DATA REPRESENTATION METHODS FOR ACTIVITY RECOGNITION AND IMAGE UNDERSTANDING AT PIXEL LEVEL

A Thesis/Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in

The School of Electrical and Computer Science

by
Manohar Karki
B.S., Univeristy of Texas at Arlington, 2010
August 2017

Acknowledgments

The project was partially supported by Army Research Office (ARO) under Grant #W911-NF1010495.

I would like to thank all the members of my committee, Supratik Mukhopadhyay, Jianhua Chen, Konstantin Busch, Yejun Wu, Hsiao-Chun Wu, and members of my research group: Robert DiBiano, Saikat Basu, and Malcolm Stagg for their time, encouragement, advice, and support.

I am grateful for my parents who have supported me in every stage of my life,and my wife who has been helping me in the difficult stages of completing my PhD. I also appreciate my brother, and all other members of my family who have always provided guidance and support.

Table of Contents

List of Tables

List of Figures

Abstract

Efficient representation of large amount of data particularly images and video helps in the analysis, processing and overall understanding of the data. In this work, we present two frameworks that encapsulate the information present in such data. At first, we present an automated symbolic framework that is used to detect activities from videos in real time. This framework makes use of regular expressions for symbolic representation of motion and subsequent activity resulting from the motion, that are present in videos. The framework is designed to uniformly handle trajectory based videos that follow certain paths and articulated activities with periodic motion. The motion present in the videos are represented by regular expressions that are provided manually or learnt automatically using positive and negative examples of strings using offline automata learning frameworks. A modified Levenshtein distance algorithm is used to determine the confidence measure between any such string that represents motion and activities described by regular expressions. Recognitions on trajectory based activities like vehicle turns (U-turns, left and right turns, and K-turns), vehicle start and stop, person running and walking, and articulated activities such as digging, waving, boxing, and clapping in videos from the VIRAT public dataset, the KTH dataset, and videos obtained from YouTube are evaluated using this framework.

Next, we describe a core sampling framework that makes use of activation maps from several layers of a pre-trained Convolutional Neural Network (CNN) as features to another neural network using transfer learning to provide an understanding of an input image. The intermediate map responses of a Convolutional Neural Network (CNN) contain information about an image that can be used to extract contextual knowledge about it. Our framework creates a representation that combines features from the test data and the contextual knowledge gained from the responses of a pretrained network, processes it and feeds it to a separate Deep Belief Network. We use this representation to extract more information from an image at the pixel level, hence gaining understanding of the whole image. We experimentally demonstrate the usefulness of our framework using a pretrained VGG-16

model to perform segmentation on the BAERI dataset of Synthetic Aperture Radar (SAR) imagery and the CAMVID dataset.

Using this framework, we also reconstruct images by removing noise from noisy character images. The reconstructed images are encoded using Quadtrees. Quadtrees can be an efficient representation in learning from sparse features. When we are dealing with handwritten character images, they are quite susceptible to noise. Hence, preprocessing stages to make the raw data cleaner can improve the efficacy of their use. We improve upon the efficiency of probabilistic quadtrees by using a pixel level classifier to extract the character pixels and remove noise from the images. The pixel level denoiser uses a pretrained CNN trained on a large image dataset and uses transfer learning to aid the reconstruction of characters. In this work, we primarily deal with classification of noisy characters and create the noisy versions of handwritten Bangla Numeral and Basic Character datasets and use them and the Noisy MNIST dataset to demonstrate the usefulness of our approach.

Chapter 1
Introduction

The process of extracting information from images involves various steps of Image Analysis. The smallest components of images consists of pixels. A collection of pixels together form different objects in the images. These individual component of images are correlated with respect to their positions, which make them distinct from other data types. Data Representation has various uses in Computer Science, Mathematics, Engineering and a host of other fields. Effective representation of raw data is helpful for data analysis, manipulation, storage, machine learning etc. For example a good visualization of raw data can help add insights about the characteristics of that data and binary representation of characters, strings and instructions are the basic building blocks of a computer itself. In Machine Learning, data representation helps in solving the problem of learning from the data. Principal Component Analysis (PCA) is an example of an algorithm to reduce the dimensionality of data so that specific algorithms designed to understand the problems are able to solve the problem effectively. Once we learn about the data we gain knowledge from it. The information learned from complex systems also needs to be represented so that a computer can use it. In this work, we have defined various frameworks and ways to represent data and knowledge learned from a variety of complex real world problems. We primarily deal with image and video data and the ways we represent them, the knowledge gained from them and the knowledge gained from the algorithms used to learn them.

1.1 Symbolic Framework

Our symbolic framework is an approach to encapsulate information seen on videos, specifically different activities performed by humans or objects in the videos using symbols or strings. We take advantage of regular expressions and advances in algorithms that can learn regular expressions efficiently to learn those activities that have been encoded using symbols.

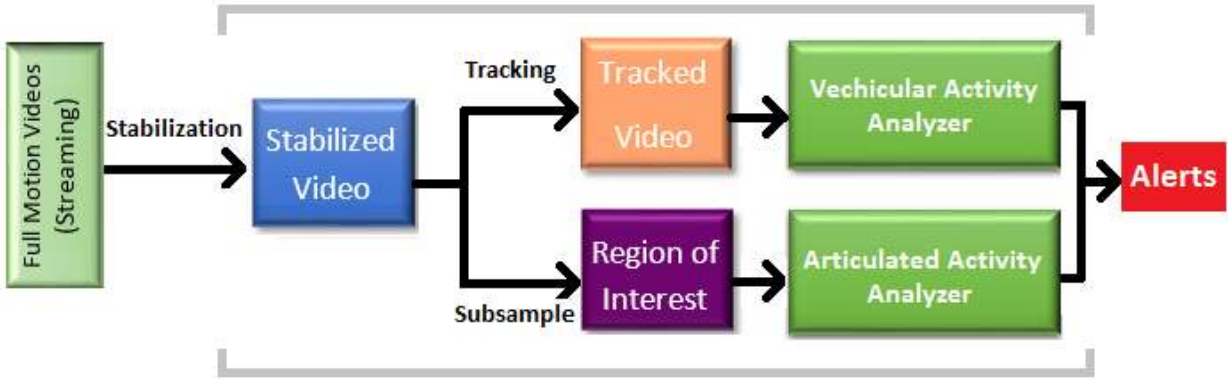


Figure 1.1: The architecture of the Symbolic framework.

Intelligence can be obtained from recognizing activities underlying the dynamics of moving objects [95, 5] in surveillance videos. This knowledge obtained is a key enabler for many video analytics applications [87]. Moving objects in a surveillance video can comprise humans, vehicles, animals, etc. While many deployed surveillance systems provide automatic tracking, describing the activities of tracked objects still generally requires human intervention. Manually watching video streams continuously is time consuming, and the human mind is not well suited to watching endless hours of repetitive video. After a while, analysts are prone to miss events and even if no events were missed; manually keeping a log of everything that happens in a video would not be viable.

It is therefore essential to develop techniques to automatically analyze the motions and behaviors of objects in video streams. Potentially important events could then be flagged in real time, giving analysts a more manageable amount of data to handle. And detailed logs of all events could be automatically kept. While in the past few years there has been a slew of research progress in real time activity recognition from videos, the general problem is inherently hard. Each activity can be assigned to distinct motion characters and with little variations. We define *trajectory-based activities* as activities involving turns of a vehicle, stopping a vehicle, moving from a stopped position, people moving around a certain path and the physical motion between frames. We define *periodic articulated activities* like a human gestures. In this type of motion, we assume the human or object is stationary

and only parts of the object or human are moving. Existing algorithms that recognize individual activities include using specialized complex descriptor matching (such as bag-of-words or histogram-of-gradients), probabilistic logics, and other classification methods. Complex descriptor matching tends to be computationally expensive. Even with recent improvements in activity recognition, a uniform approach that is able to efficiently recognize and be applicable to a wide variety of problems is missing. Our approach can be seamlessly integrated with reasoning platforms so that inferences at a higher level of abstraction such as anomalies can be recognized. In this work, we describe a symbolic framework that automates the recognition of activities from videos with real time performance.

1.1.1 Regular Expressions for Encoding Activities

In our framework, regular expressions are used to represent (possibly infinite) sets of motion characteristics obtained from a video. It can uniformly handle both trajectory based and articulated activities that are periodic in nature while providing polynomial time graph algorithms for fast recognition. The regular expressions representing motion characteristics for activities are in many cases simple enough to be provided manually (e.g., by an expert analyst or by crowd sourcing, etc.) providing a *generative* framework; or they can be learnt automatically from positive and negative examples of strings describing dynamic behavior using offline automata learning frameworks like libalf [17]. Confidence measures are associated with recognitions using Levenshtein's distance between a string representing a motion signature and a regular expression describing an activity [71]. Since regular languages described by regular expressions are as expressive as monadic second-order logic over strings (MSO-S) [91], we get for free a rich logical framework that can be integrated with reasoning platforms for performing high level inference by linking together activities. We have used our framework to recognize trajectory-based activities like vehicle turns (U-turns, left and right turns, and K-turns), start and stop, person running and walking, and periodic articulated activities like human digging, waving, boxing, and clapping in videos from the VIRAT public dataset [73], the KTH dataset [86], and a set of videos obtained

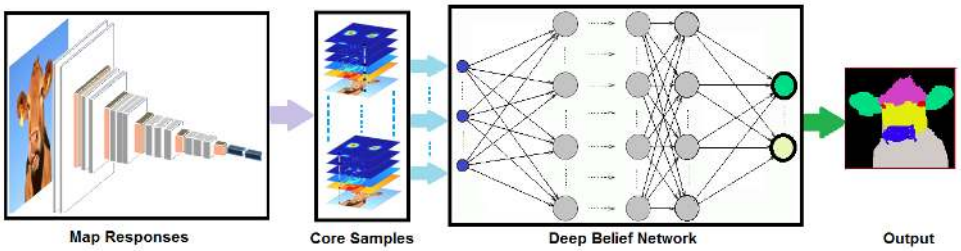


Figure 1.2: The architecture of the core sampling framework.

from YouTube. The details of these examples and the experimental results are provided in Chapter 3.

1.1.2 Overview of the Architecture

Fig. 1.1 describes the overview of the architecture of our framework. The full motion surveillance videos first go through a stabilization stage where the random movements in the video are minimized. We feed the video through a tracker which tracks objects in the video and passes it to the activity analyzer which then produces alerts when some kind of activity is triggered. For the articulated activity videos it might not be necessary to track objects if they have stationery objects. In this scenario, the region of interest is passed through the activity analyzer. The activity analyzer is responsible for representing various activities using regular expressions. Activities are represented using a series of symbols and differentiated from other activities based on these symbols.

1.2 Core Sampling Framework

We designed our core sampling framework to do pixel level classification and regression in images. The core sampling framework also makes use of pretrained networks to bypass training large networks with a huge dataset. We make use of features that have been learnt in the previous network to help classification and regression in different datasets. We use such features and random samples of the data and trains a second network to solve the problem, we are interested in more effectively. Our core sampling framework is able to use the activation maps from several layers as features to another neural network using transfer learning to provide an understanding of an input image. Our framework creates

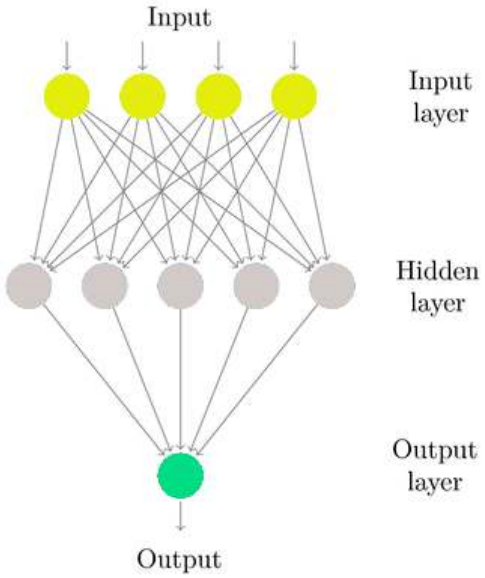


Figure 1.3: Artificial Neural Networks

a representation that combines features from the test data and the contextual knowledge gained from the responses of a pretrained network, processes it and feeds it to a separate Deep Belief Network. We use this representational model to extract more information from an image at the pixel level, thereby gaining understanding of the whole image.

1.2.1 Prediction at Pixel-Level

Pixel-wise prediction from images has a lot of applications [42] in scene understanding. It helps to generate a fine grain segmentation compared to existing techniques [41],[9],[89] that segment at a coarser level. The rise of machine learning has led to novel and automatic segmentation techniques that require very little user input [8],[67],[39]. Deep learning[14],[45], [60] in particular, has enabled this as it makes it possible to learn data representations without supervision.

1.2.2 Artificial Neural Networks

Neural Networks are built upon the mechanism of biological neurons and their networks. Artificial Neural Networks consist of a large amount of nodes inter-connected by weighted lines across layers that act together as non-linear "summing devices" [32]. These weights are updated as the network receives more training data. Artificial networks as an associate

memory that can be used as associate memory and this memory can work with the presence of of certain level noise [98]. Fig 1.3 shows a sample architecture of an artificial neural network. It is made up of inputs and output values with several types of layers each with neurons in the middle.

1.2.3 Convolutional Neural Networks (CNNs)

CNNs are particularly useful for image understanding, work on parts of the input locally usually at different pyramidal levels. In doing so, the network gathers both low level and high level information about an input image. So, the lower layers encode the pixels while the deeper layers provide representation of objects comprising of those pixels that eventually help in understanding the entire image. Figure 1.4 shows an example a popular CNN architecture: LeNet [62]. Convolutional Neural Networks are a derivative from artificial neural networks where the feed forward network consists of special layers that take advantage of spatial correlation between the data points. They have translation invariant characteristics. The spatial arrangement of images allow for CNNs to perform better. Pixel wise classification and image understanding can be improved with the local and global information that are encoded in the different layers of a CNN; this information, stacked at different pyramidal levels, can be viewed as a *core sample* that can enable better understanding of an image. A CNN consists of different layers that convolve the input image and/or parts of the image with filters. Training such a network involves determining these filters so that they produce the expected output from a given input. We can obtain map responses by feeding in test images through these (pre)trained layers. Fig. 4.3 shows some of the map responses that we get when we feed an input image of a cow through the various layers of the pretrained network.

- **Hypercolumn**

A hypercolumn derived using an image and each hypercolumn is defined as a vector with k columns, where k is the amount of intermediate response maps from the VGG-16 model, with each component of the vector being a corresponding pixel in each of the maps. A

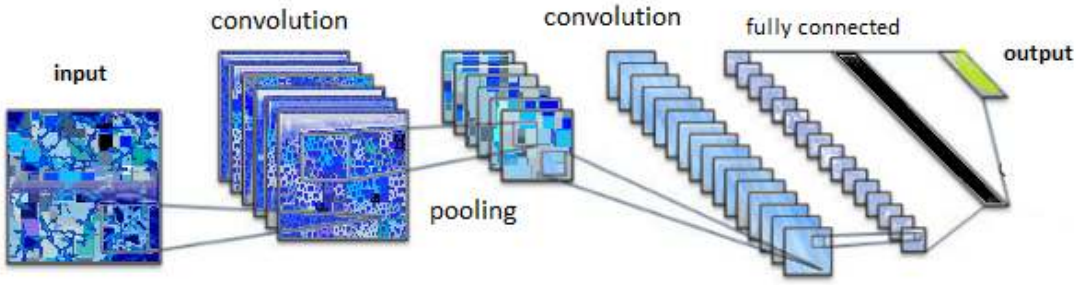


Figure 1.4: Example of Convolutional Neural Network.

hypercolumn does not preserve any spatial correlation between the constituent maps. The notion of *Hypercolumn* introduced in [42] builds upon this. The term Hypercolumn is used to describe a column of a group of map responses from the convolution layers of a network, aligned together such that each value in that column refers to the output of different layers from each of these maps, for an individual pixel as shown in Fig 4.4. A random sample drawn from a core is called *a core sample*. A *core* is a collection of hypercolumns, one per pixel in an input image. Core samples generated from input images a fed to the second stage of our framework. The two-stage architecture of our core sampling framework is given in Figure 1.2. We use the VGG-16 [90] model to bootstrap our framework. This model was trained using the ImageNet dataset [56] which comprises of more than a million training images and 1000 classes. The intermediate maps, that are generated for each pixel during the testing phase when images are input to the above model, when stacked together and resized to a uniform size, form hypercolumns. Image pyramids have already been used to extract or learn more information from images [4], [25] as well as for image segmentation [9]. In a sense, the response from the layers of a CNN can also be viewed as the different pyramidal levels of the image viewed at different locations of that image. In our framework, we accumulate such hypercolumns from each pixel of every training image and use that as training data to a deep belief network.

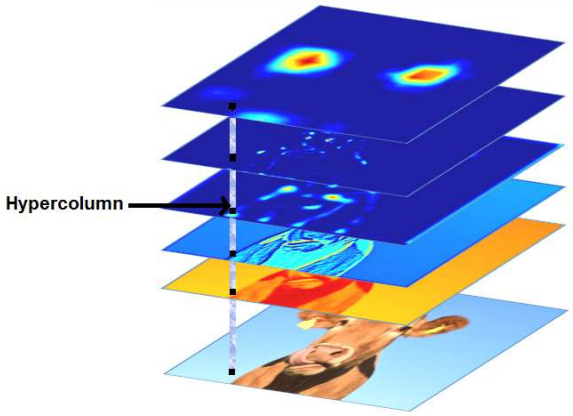


Figure 1.5: A hypercolumn is [42]represented using response maps when an image of cow is passed through a Convolutional Layer.

1.2.4 Deep Belief Networks

The second stage of our framework consists of a Deep Belief Network (DBN). Deep Belief Networks are unsupervised deep learning models [45]. A deep belief network is trained by first training stacked Restricted Boltzmann Machines (RBMs) in a greedy manner [46]. A weight matrix for an RBM is obtained by training it in an unsupervised manner. This weight matrix act as the initial weights on the connections between the first two layers of the DBN. The RBM then transforms the matrix of input feature vectors by calculating the mean activation of the hidden unit. The transformed data is then used to initialize the weights of the connections between the second and the third layer in a similar way. This process is repeated until the weights for all connections are initialized. These weights are later fine-tuned using supervised training to change the learned representation to a classifier. Because of the unsupervised training in the beginning, DBNs are useful even with limited labeled data. An example of a Deep Belief Network (DBN) is shown in Fig. 1.6.

Since no spatial correlation among the maps is preserved in the hypercolumns comprising the input core samples, a CNN cannot be used in the second stage as the filters in a CNN presume spatial correlation between adjacent maps. The DBN interprets the input

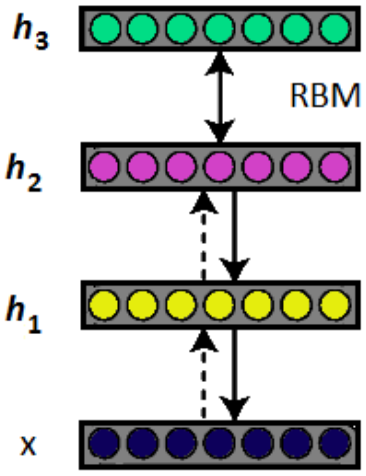


Figure 1.6: Sample architecture of a Deep Belief Network.

core samples to provide an understanding of the original input image.

1.2.5 Contributions

These are the contributions made with the help of our Core Sampling Framework:

1. We present a novel core sampling framework that is able to use activation maps from several layers of a CNN as features to another neural network using transfer learning to provide an understanding of an input image. It creates a representation that combines features from the test data and the contextual knowledge gained from the responses of a pretrained network. This model is used to extract more knowledge about an image at the pixel level, thereby gaining understanding of the whole image.

2. We experimentally demonstrate the utility of the core sampling framework by showing its ability to automatically segment the BAERI dataset [38] of Synthetic Aperture Radar (SAR) imagery and the CAMVID dataset [23].

1.3 Pixel Level Reconstruction and Feature Representations using Quadtrees

We use the data representation methods used in the Core Sampling Framework to do a pixel level reconstruction of noisy images by removing the noise. We leverage the recent advancements in Deep Learning to develop techniques for efficient representation of sparse features. Sparse representations are usually compact representations of signals or features that have default values that are unnecessary [49], [19]. Quadtrees can be an efficient representation for learning from sparse features. Real world images, especially those of handwritten characters, are usually noisy. Efficient algorithms to remove noise can help in classification. We improve the effectiveness of probabilistic quadtrees by using a pixel level classifier to extract the character pixels and remove noise from handwritten character images.

1.3.1 Overview of our approach

Using Figure 1.7, we describe the steps in which we classify the handwritten character images. This algorithm makes use of a Convolutional Neural Network (CNN), which has been previously trained, to gather features from images of handwritten characters. These extracted features help learn the shape of the characters and segment out those pixels that do not belong to the characters. The information acquired from the pre-trained CNN helps train a Deep Belief Network (DBN) (called the reconstruction network) that reconstructs the handwritten characters through transfer learning [99] segmenting out noisy pixels.

1.3.2 Character Reconstruction Network

Handwritten character images are the input to the Character Reconstruction Network (CRN) and it produces as output a denoised binary version of it. This network is the backbone of our approach. The goal of this network is to be able to remove noise from noisy data and output images that are similar to standardized datasets. This enables us to be able to compete with noise-free character image datasets. The reconstruction network

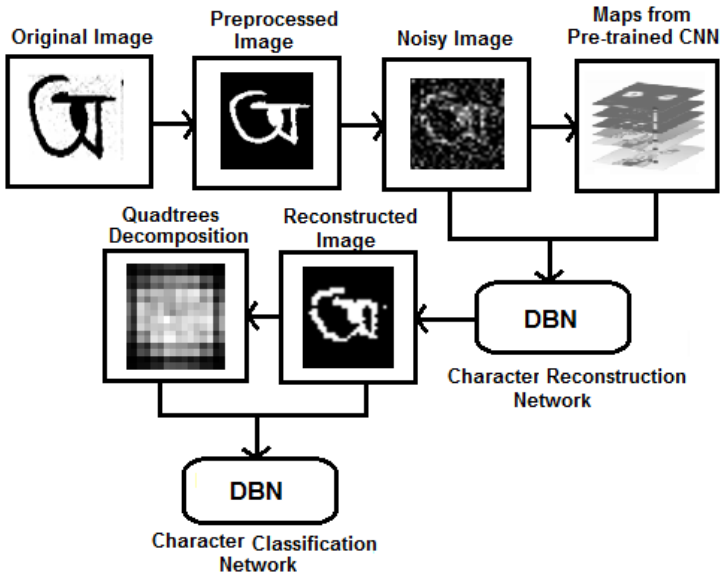


Figure 1.7: Steps involved in the handwritten character classification

segments out those pixels that do not belong to the character. A probabilistic quadtree is then used to learn the sparse features from the resulting denoised binary image obtained from the reconstruction network and used to train a Character Classification Network (another DBN) to classify the character images.

We experimentally demonstrate the effectiveness of our algorithm at multiple resolutions using a recognition scheme similar to the one used in [15] while obtaining better performance compared to [15]. We also improve upon the quadtree decomposition technique used in [11] by using a saliency map to eliminate blocks that do not provide discriminating power. The saliency map helps reduce the dimension of the training data enabling efficient learning.

1.3.3 Contributions

With the help the pixel level reconstruction algorithm, we make the following contributions with this approach:

- It develops an efficient framework that removes noise from noisy images of handwritten characters

- It improves the effectiveness of probabilistic quadtrees by using a pixel level classifier to extract the character pixels removing noise from handwritten character images.
- It introduces a dataset [1] comprising noisy Bangla basic characters and numerals with three different noise types. This dataset can serve as the basis and benchmark for future research in noisy Bangla character recognition.

[1] https://drive.google.com/open?id=0B0gFcrqVCm9pT1VjbmUwYzdOTjA

Chapter 2
Related Work

2.1 Symbolic Framework

Activity Recognition involves recognizing a diverse set of activities of human, animals, vehicular and other objects. While activities of vehicles and other non-living objects are simpler the activities of humans and the objects operated by humans usually are complex [54]. There have been various approaches to recognize activities in the past which include: logic based [85], probabilistic reasoning based [47] and pattern-based activity recognition [54]. Vision based activity recognition involves using videos (and images) of humans or objects to recognize activities being performed by them. Usually, vision based approach also involves tracking the objects in the videos because often times they are in motion.

2.1.1 Logic based Activity Recognition

Logic based detection of activities makes use of "formal" and "declarative" semiotics [7] and their efficiency and scalability coupled with the tools to allow machine learning to described event structures and recognition makes their use valuable. Our approach provides a logical framework with fast algorithms for recognition. A Dynamic Bayesian Networks method is used for detecting activities in the presence of tracking failures (e.g., track switching) [47]. Markov Logic Network has been used to recognize interesting activities in video [94]. Text-based classification was used to recognize actions in movies [59]. Movie scripts are used to annotate actions for providing training data. Other frameworks include representing contexts to facilitate activity recognition [51] and logic programming [88].

2.1.2 Probability based Activity Recognition

Activities have been often been modeled using markov chains which can represent simpler activities [54]. For complex activities, Hidden Markov Models (HMM) and Conditional Random Fields have been popular for recognition. A stochastic process is used by the authors in [78], [36], popularly called Hidden Markov Models, that has hidden processes and

sequence symbols of these are observable symbols. A framework based on 3D Markov Random Field has also been proposed to recognize trajectory-based and articulated activities like dancing, talking, etc. [30]. Even a combination of Gaussian Mixture Models has been used to representing activities for activity recognition [65]. In contrast, we use a symbolic regular expression-based framework for representing possibly infinite sets of motion signatures. Events at a traffic intersection were detected using a combination of foreground/background segmentation and color model with probabilistic data association [96]. Dynamic Bayesian Network learning have been used to represent context knowledge as first order probabilistic logic formulas [101].

2.1.3 Pattern based Activity Recognition

Modern approaches have been geared towards pattern based activity recognition. Using a normalized edit distance measure, representation and retrieval of object trajectories has also been proposed [27]. The symbolic framework builds on top of our robust tracking framework that uses a combination of foreground background segmentation and a color histogram model to compensate for track failures. An activity recognition framework has been developed that considers activities as resulting from the execution of a dynamical system [64]. A generative mixture model was designed for activity recognition using the velocity history collected from tracks [70]. Another uniform framework for activity recognition was used to model motion using a frequency domain transform [80]. The authors in [84] provide a review of such frameworks to recognize activities and mention the use of context-free grammars that are in use to detect complex actions. Time complexity for recognition of context-free languages is $O(n^3)$ whereas that for regular ones is linear. Besides, context-free languages cannot be directly tied to a logical framework unlike the regular expression framework which provides expressiveness equivalent to MSO-S. In our framework, each moving object generates a string. Our regular expression-based framework is equivalent to MSO-S and thus is more expressive than first order logic (over strings) [35]. Speech signals has been used to encode activities and employ speech recognition techniques for

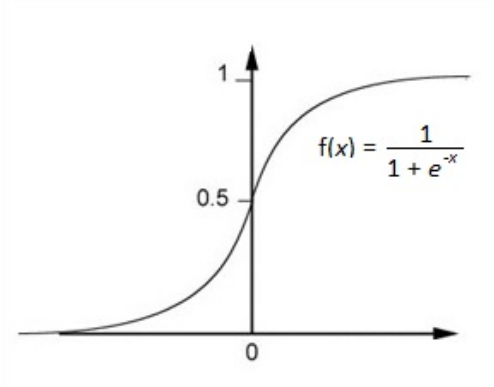


Figure 2.1: Sigmoid Function

activity recognition [26]. We use a measure based on the Levenshtein distance to quantify the accuracy (or uncertainty) of recognition of an activity.

2.2 Core Sampling Framework

The Core Sampling Framework follows the recent advancement in Machine Learning particularly Neural Networks where deep architectures are supported. The frame work uses deep architectures and the information learned during by one network as features to a second network. This eases the problem of manually selecting appropriate features. *Core sampling* has been used in engineering and science to understand the properties of natural materials[68], climatic record from ice cores [55] etc.

2.2.1 Logistic Regression

Logistic Regression is the general model utilized for regression where dependent variables are based on categories and for binomial classification [33]. Linear regression is generally modeled to evaluate continuous output. The probabilities of the classes are determined by logistic regression For multiple classes, one vs rest classification is used, for Logistic regression. A sigmoid function, seen on Fig. 2.1 is a special type of logistic function used as squashing function to clip large values show that the values can remain between 0

and 1. A general sigmoid function is of the form [66]:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

2.2.2 Stochastic Gradient Descent

Gradient descent is an iterative process of optimizing a function. Stochastic gradient descent is an incremental process that helps optimize the objective function by estimating the gradient each iteration with respect to a randomly selected example [18]. Stochastic gradient descent moves more quickly compared to a regular gradient descent because the estimation is taking place with the help of a smaller number of examples instead of the entire set [1]

2.2.3 Multi-layer Perceptron (MLP)

An MLP is a finite directed acyclic graph [81]. Single neurons are not enough to learn complex functions. In an MLP, the number of dimensions for the input must match the amount of neurons in the input layer. A perceptron with multiple layers can also be considered a simple Artificial Neural Network. To learn the parameters a stochastic gradient descent (SGD) approach that relies on the backpropagation algorithm to reach the optimum setting is used. We also use logistic regression on our MLP to calculate the output. Our Deep Belief Network is built up using a Multi-layer Perceptron.

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks are a type of artificial neural network that take advantage of the connections between the related data points. The connections between neurons is reduced, which helps to combat the curse of dimensionality in deep networks where neurons are fully connected to other neurons in the adjacent layers. CNNs have been used to extract information from images; deep CNNs have enabled recognition of objects in images with high accuracy without any human intervention [63], [56], [13], [83]. There has been some research in using the information acquired from the intermediate layers of a CNN to

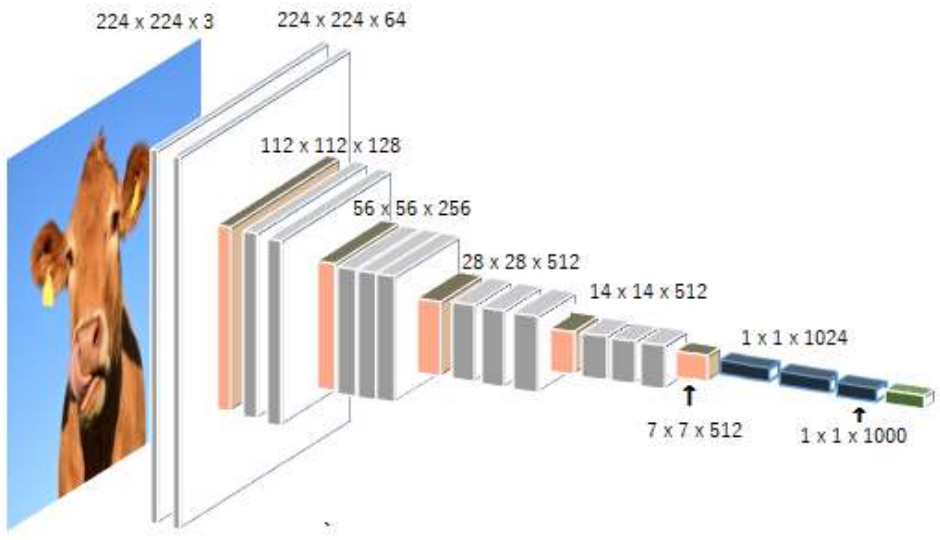


Figure 2.2: VGG-16 Architecture. The architecture shows multiple convolution layers, with the max-pooling layers in between followed by 3 fully connected with a softmax layer for the final output.[90]

solve tasks such as classification, recognition, segmentation or a combination of these [42], [8], [44], [79], [67].

2.2.5 Image Segmentation

We make use of the classical Image Segmentation problem to show the usefulness of our Core Sampling Framework. Various techniques have been used over the years where manual techniques were used for the segmentation. Haralick and Shapiro [41] describe classical image segmentation techniques such as thresholding, multidimensional space clustering, region growing, etc. Comanciu and Meer [31] use the mean shift algorithm to provide automatic segmentation, where human intervention is needed to choose the class of a segment. There are well known graph-based algorithms for image segmentation; e.g., Normalized Cuts [89], where segmentation is achieved by measuring the similarity of graph partitions; Graph Cuts [37], where a segmentation is defined as set of regions; this set of regions is repeatedly combined based on the similarity between neighboring regions. Rother et. al. [82] implement iterative estimation on top of graph cuts algorithm [20] to define a boundary for the segmentation of objects where a user selects the broader region.

2.2.6 Deep Learning Based Segmentation

Recent approaches to segmentation also include various ways to use a CNN to segment images. Girshick et. al. [39] use Region-based CNNs (R-CNNs) where category-independent region proposals are defined during the pre-processing stage, that are input to a CNN to generate feature vectors. A linear Support Vector Machine (SVM) divides the image into regions. Unsupervised Sparse auto-encoders have been used in [29] on Synthetic Aperture Radar (SAR) data to classify different types of vehicles. They only deal with classification of images already segmented into smaller regions containing the objects. We deal with segmentation by classification at pixel level. Ladicky et. al. [57] use a Conditional Random Fields (CRF) [58] based approach to aggregate results from different recognizers. Zhang et. al. [102] recover dense depth map images and other information about the frames in video sequences such as height above ground, global and local parity, surface normal, etc. They use graph cut based optimization and decision forests to evaluate their features. The need of video sequences limits their application. Both the previous approaches need manual feature extraction.

2.2.7 Pixel Level Segmentation

Another approach is to use deconvolution layers after the convolution layers as a way to reconstruct segmented images as done in [72] and [8]. SegNet [8] uses an encoder architecture similar to VGG-16. The decoder is constructed by removing the fully connected layers and adding deconvolution layers. It is used to transform low resolution maps to high resolution ones. The original paper that proposed the notion of hypercolumns [42] also uses the maps from the intermediate layers of a CNN to segment and localize objects in images. They use a location specific approach; in particular, they use K x K classifiers across different positions of the images where K is a constant. A linear combination of these classifiers is used to classify each pixel. They also use bounding boxes of size 50 x 50 and try to predict the heat map to localize objects in an image.

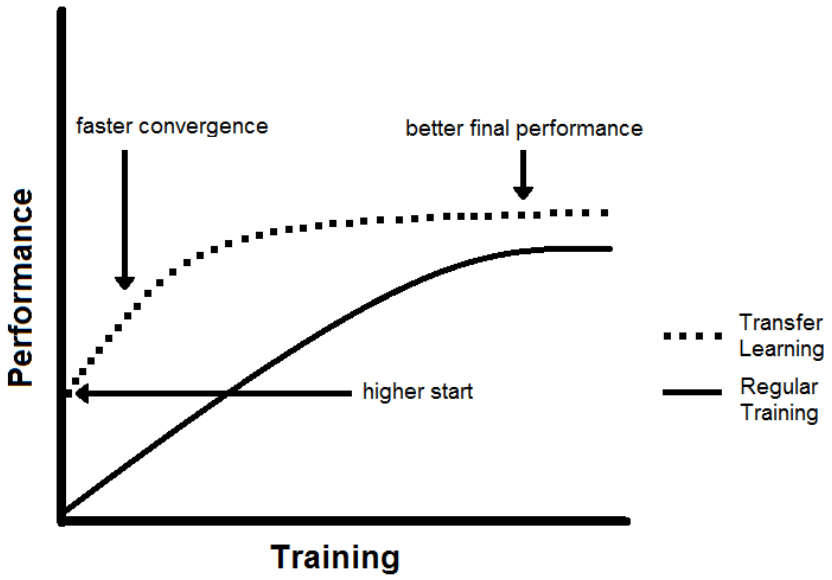


Figure 2.3: Characteristics of Transfer Learning.

2.2.8 Transfer Learning

Transfer Learning allows the use of the knowledge acquired from solving one problem to improve the solution to another [93]. Our core sampling framework makes use of transfer learning, where the core sample acquired from a previously trained network is used for training a second network. Fig. 2.3 shows the benefits of using transfer learning during training. This strategy helps increase the overall performance and speeds up training. The other benefit of this approach is that it avoids training a very large network on a huge dataset. Often there is not enough training data for a particular task; this can be overcome by using the knowledge learned from a similar task [76] or a different task [99]. In [34], the authors use transfer learning for segmentation of hyper-spectral images by training on an unlabeled dataset and then using transfer learning to improve separability based on the learned knowledge.

2.2.9 Deep Belief Networks

Deep Belief Networks (DBNs) use hierarchical representations to generalize and learn data in a domain by first pre-training a Restricted Boltzmann Machine (RBM) [14]. DBNs

are first trained in an unsupervised manner. After that the stochastic and latent variables are fine tuned in a supervised phase. After the layers are trained, the weights are transferred to the main Neural Network [13]. This initialization makes the convergence of the neural network much faster than a neural network that has not been initialized. The supervised phase comprises of a feed forward backpropagation based Neural Network having multiple layers.

2.3 Pixel Level Reconstruction and Feature Representations using Quadtrees

Reducing the dimensionality of data helps improve the efficiency of learning algorithms while making the models simpler and more robust [43], [46]. Similar to the Core Sampling Framework, we again use Transfer Learning to use the knowledge acquired from one domain to be used in another The pixel level reconstruction o removal of noise at pixel level. The reconstructed images consist of either the character pixels or the background.

2.3.1 Quadtrees

Quadtrees have been used previously to compress images [69] and represent spatial data [6]. In [11], the authors use probabilistic quadtrees to represent character images and classify them using a deep belief network (DBN). Deep Belief Networks use hierarchical representations to generalize and learn from data in a domain by first pre-training a Restricted Boltzmann Machine (RBM) [14].

2.3.2 Bangla Characters

For Bangla Characters recognition, Chain code histogram features are used to discriminate classes in a multi stage approach in [15] where rejected images are classified again at higher resolutions. We follow a similar approach of using higher resolutions when classifying the Bangla Numeral Dataset [15]. In [16] the authors classify handwritten Bangla Basic Characters that has 50 classes. They use a two stage approach where they employ chain-code and gradient based features. The first stage is a modified quadratic discriminant

function (MQDF) based classifier while the second stage is a Multi-layer Perception based classifier that helps to improve recognition on confused classes.

2.3.3 Probabilistic Quadtrees Based Approaches

On the Noisy Bangla Datasets that we focus on, [11] have used probabilistic quadtrees to learn sparse representations of handwritten character images and have used a two layer DBN for the classification. Quadtrees are decomposed into blocks and a homogeneity criterion is used to decide whether or not to decompose them further into smaller blocks. For handwritten character images containing significant amounts of noise, the sparse representations learnt using quadtrees in [11] are not efficient. The authors also introduce the handwritten Bangla Numeral Dataset which includes the 10 Bangla Numeric Characters and three types of noise: gaussian random noise, motion blurred, and diminished contrast [11] . We inject these three types of noises into the Bangla Basic Character dataset consisting of 50 classes resulting in the Noisy Bangla Basic Character Dataset. In [11], once a block has been chosen for decomposition in any one image based on the homogeneity criterion, that block is identically decomposed for every other image. We build upon this approach and use a saliency map to improve the representation and also use another DBN for character reconstruction removing noise.

Chapter 3
Symbolic Framework

Activities are associated with possibly infinite sets of motion signatures (think of the numerous ways in which a U-turn can be made). It is necessary to identify the common components of activities to find a proper representation of processes involved in an activity. In our framework activities are represented by regular expressions describing their motion signature set. Various components of an activity are encoded as symbols and patterns are observed based on those activities. We recognize activities from videos by keep track of objects across the frames. Incoming full motion video is first input to a stabilization engine for jitter correction and background noise subtraction. With the presence of noise and jitter, there is a possibility that the collection of symbols would misrepresent the activities. The stabilized video is then input to a tracker. The tracker follows objects present in the videos and outputs an annotation file which keeps track of the perceived location of the object. The output of the tracker is input to an activity analyzer. The analyzer smooths the data and extracts from the tracked video strings describing motion signatures of moving objects in it. The smoothing of the data further removes left over noise. The strings are then matched against regular expressions representing activities using approximate pattern matching algorithms for "soft" matching. A strict matching would not allow small variations in complex activities. We describe the different components of the framework in details below. An overview of the Framework is presented in Fig. 3.1.

3.1 Regular Expressions

A regular expression [48] describes a pattern representing a (possibly infinite) set of strings (i.e., a language) over an alphabet. Its a great utility for representations where patterns exist. Each regular expression contains set of characters and symbols that match to different words in a language. Here, the language refers to set of all possible sequence of characters that can be formed using the rules set by the regular expression. The lan-

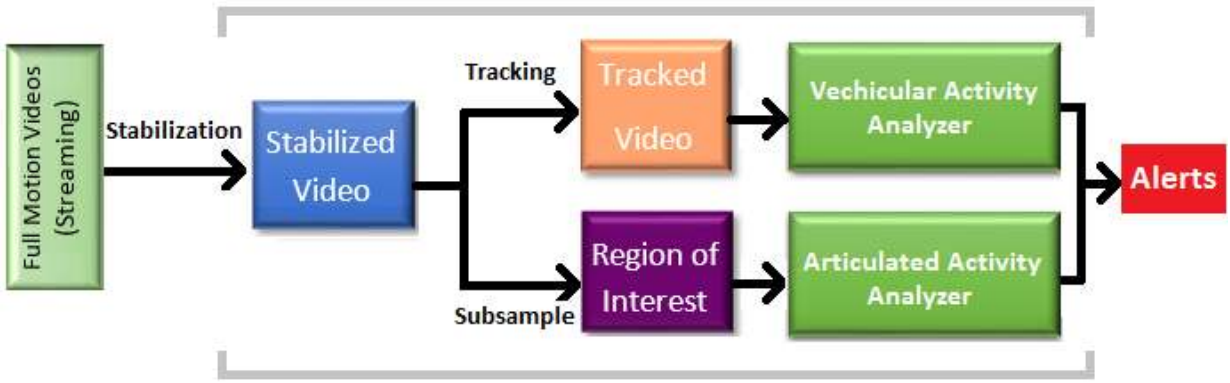


Figure 3.1: The architecture of the proposed framework.

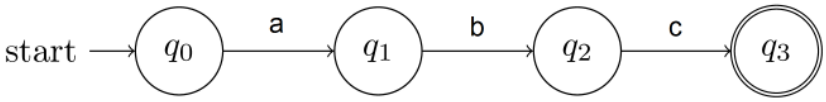


Figure 3.2: An automata that takes in the string 'abc'.

guage contains the set of all strings accepted by the automata corresponding to that regular expression. For example a regular expression /mat/ would match every occurrence of the string 'mad'. Regular expressions also have wild cards such as '?' and '*' where 'represents' a single wild character and '*' can represent multiple wild characters. For example /m?d/ can accept 'mad', 'mid', ' mud' or 'mid' etc. and /t*o/ could match 'two','tomato','tempo' or even just 'to'. Fig. 3.2 shows an examples of a graphical representation of a simple automata that only accepts a string 'abc'. Regular expressions can be conveniently represented by finite state automata that allows efficient manipulation using graph algorithms. They support efficient string search operations, generally polynomial in the number of bytes of data to be searched and provide great flexibility in symbolically describing sets of strings.

In terms of expressiveness, regular languages (i.e., those described by regular expressions) are as expressive as monadic second-order logic over strings (MSO-S) [91]. MSO-S subsumes temporal logics like LTL [77] that are popularly used for describing dynamic behavior sequences. We will use the flexibility of the framework of regular expressions to describe and classify object motions and recognize underlying activities.

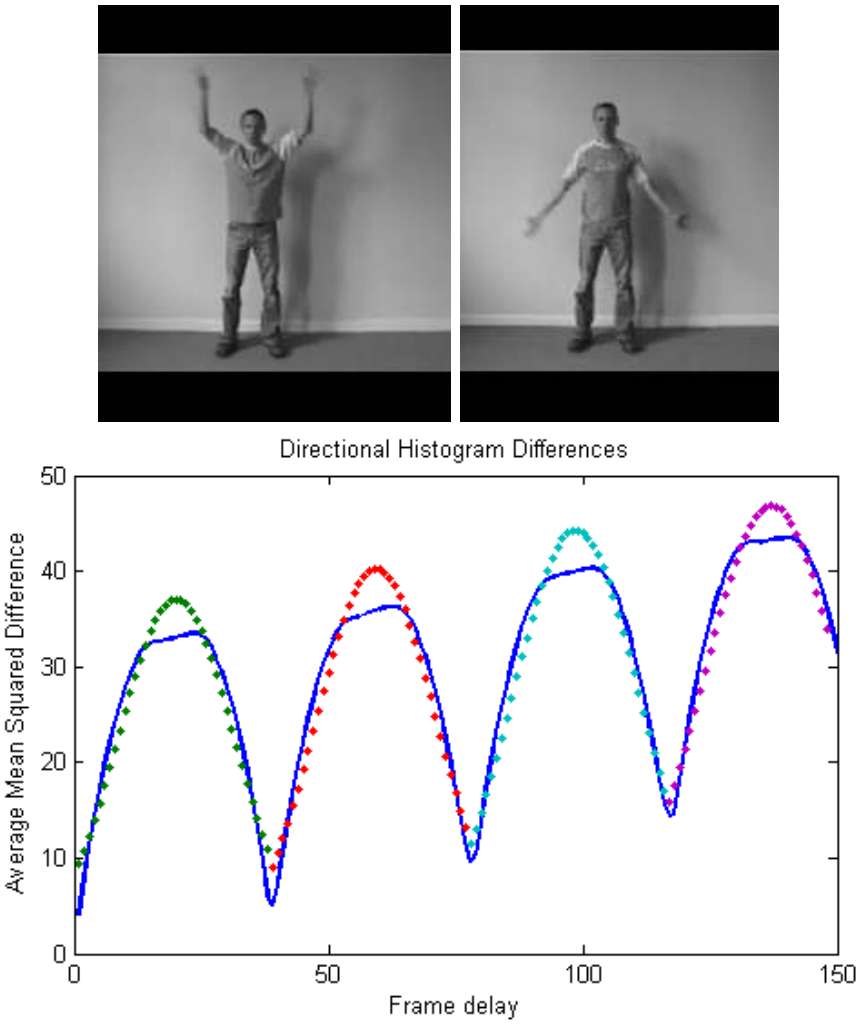


Figure 3.3: Running average of Mean Squared Difference of Directional Histograms, fitted with Gaussian curves for Hand waving action.

3.1.1 Representing Directions and Periodicity

As Fig. 3.4 shows, we use the characters a through h to represent both the cardinal (N, S, E, W) and ordinal (NW, NE, SW, SE) pixels representing "unit vectors" in the respective directions. To make our algorithm more robust and detailed we can increase the number of such directions. In fact, for our final experiments, we used 24 directions instead of just 8. This allowed us to represent activities at a finer level. Using this language of motion description, the string abc represents a movement to the east, then from there to the northeast, then from there to the north. The string *aa* represents a motion two units eastward, the string *aeaeae*. . . represents a left-and-right oscillation, and so on. Fig. 3.7 demonstrates how this mapping applies to vehicle motion. Each instance of the depicted vehicle represents its position at each frame during a left turn. The vehicle begins traveling eastward, then veers northeast as it begins to turn; at the end of the turn, it is situated facing north. The first and second frames map to the character *a*; the third and fourth to *b*, and the final to *c*. Thus, the full string which describes this motion is *aabbc*. We also represent activities with periodic motion with our framework. We divide the region of interest in the video i.e. a large block within which the object exists, into several small blocks. At each of these blocks we try to look for periodic motions.Periodic motions as in articulated activities like digging, gesturing, etc., can also be mapped to regular expressions as shown in Fig. 3.3 and illustrated in Section 3.5. We use directional histograms and a difference between such histograms at these blocks and evaluate the periodicity of the activity that object is involved in. Here, we fit Gaussian equations with the running average of the histogram differences as data. The widths of Gaussian curves are subsequently mapped to character strings to extract the periodicity information in them.

To classify motion signatures obtained from a video, one first needs to track moving objects. From the tracks, symbols are extracted corresponding to motion characteristics in individual frames. After accumulating string representations of activities from a tracked

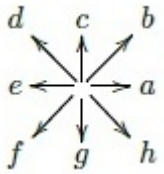


Figure 3.4: Cardinal and ordinal directions are mapped to characters.

video, we do approximate pattern matching against regular expressions representing activities.

3.2 Tracking

A robust probabilistic tracking framework is used for tracking objects in the videos [12]. This scheme uses a model that combines motion model with appearance of an object and its position in the frame where it is estimated to be. To determine the motion component using the tracking algorithm, we use a Gaussian Mixture Background Subtraction algorithm [12]. The object's appearance is determined by the color histogram that defines the distribution of the colors of the object. The position is an estimation based on optical flow between frames and last location in the previous frame. The tracking algorithm is robust to abrupt movements because of the image stabilization. A spatio-temporal filtering algorithm is used to automatically start tracking an object [10].

Object tracking is a process of measuring the optical flow which measures the perceived movement of an object, while maintaining the coordinates of its pixels [10]. During this process, it is assumed that the intensity of that object doesn't vary significantly. This is essential to calculate the optical flow between frames. The algorithm uses Gaussian Mixture background subtraction for objects who have higher degrees of motion and the Lucas-Kanade method for objects with slower motion. This approach takes cares of both types of movements. object tracking, Gaussian Mixture background subtraction, subtracts the image intensities between two sequential frames. If the image intensity at a pixel does not vary between two frames, the pixel is likely part of the background; if not, it is likely part of a moving object. We assume that most features are in fact background

features; in real- world scenarios, this is most often the case. As we will discuss later, the main disadvantage of using Gaussian Mixture background subtraction is its susceptibility to camera motion. To compensate for this, we fall back on another method for computing optical flow.

The optical flow algorithm often is associated with a displacement to apply constraint to it. The Lucas-Kanade method imposes it as well. The displacement $(\delta I_x; \delta I_y)$ represents the change in intensities of the a pixel at (x, y) with intensity value of (I_x, I_y) from the pixel $(I_x + \delta I_x; I_y + \delta I_y)$ in the next frame. Over time, the displacement is usually minimal and constant. For any pixel p:

$$I_x(p)V_x + I_x(p)V_x = -I_t(p) \tag{3.1}$$

where I_x, I_y and I_t are the partial derivatives of the image intensity with respect to x, y and t, and and V_x and V_y are the velocity vectors [10]. The constraint imposed by the Lucas-Kanade algorithm, makes it suitable for objects of varying levels of motion including uniform velocity. The type of optical flow used in this method is pyramidal Lucas-Kanade. This means that the algorithm computes the flow at a higher res image I_0 and then, low-res result is obtained :I_1 from I_0. This is followed by obtaining I_2 with the help I_1, and the process continues. Together with the Lucas- Kanade method, using the Gaussian Mixture background makes the tracking perform better than using either approach separately. Object tracking can have a lot of potential challenges: movement of cameras, pan and tilts etc. can cause the quality of the videos to be poor. These unintended rotations and movements can challenge the tracking algorithms. Besides, the videos themselves may not be in higher resolutions. Chances of occlusions and missing the objects are also present in real motion videos. Effective real time algorithm to track videos therefore has a lot challenges.

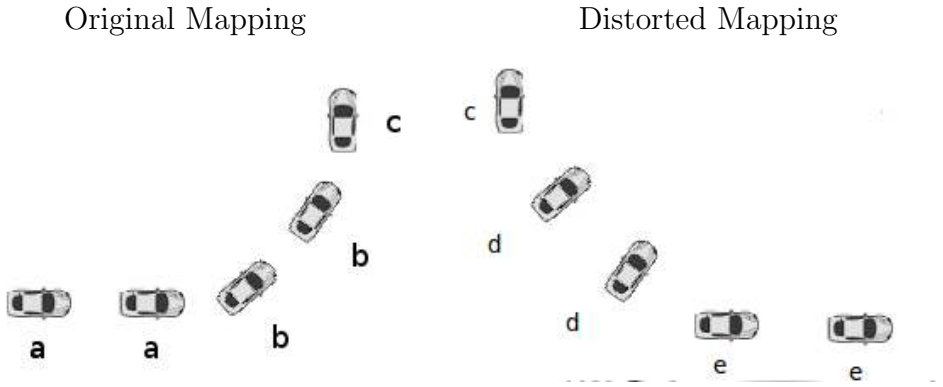


Figure 3.5: Rightward camera motion distorts the tracking and motion classification- of a left-turning vehicle.

3.2.1 Image Stabilization

Imagine holding a pen still on an unmoving sheet of paper; then, imagine doing the same on a rightward-moving sheet. The shape produced in the latter case is a line drawn from the point of initial contact to the left. As with the sheet of paper, stationary background objects appear to move directly opposite to actual camera motion. This also has ramifications for moving objects. Imagine drawing a circle on an unmoving sheet, then a rightward-moving one; the latter circle appears as a swirl. Consider the impact on the mapping of the vehicle motion in Figure 3.5 of a camera which moves rightward two pixels per frame. Were we to map the vehicle motion without first stabilizing the image, our mapping would fail to account for the motion across pixels due to the camera. The car, though oriented in the same fashion as before, would appear to be moving westward, then northwest, and finally north; this motion would be falsely encoded as *eeddc*. Clearly, this is an unacceptable result; the image must be stabilized to ensure correct tracking.

Should the camera move even slightly, the Gaussian Mixture background subtraction used to track the target object falsely detects many other moving objects which, from the frame of reference of the moving camera lens, appear stationary. In order to stabilize the image, we must ensure that each pixel remains in the same position despite the camera motion. We use the following algorithm [75] for image stabilization:

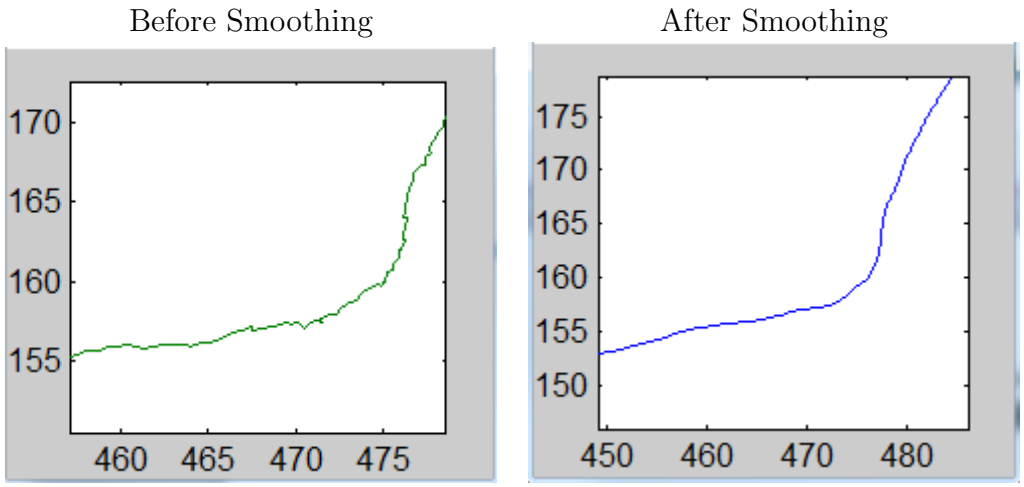


Figure 3.6: The position (x, y) of a left-turning vehicle before and after smoothing.

1. First Frame: Extract motion features using Shi and Tomasi's algorithm .
2. Subsequent Frames: Calculate Optical Flow using Lucas-Kanade's algorithm
3. Keep track of the object and find the average of all motion features.
4. Distort frame to accommodate these motion.

3.3 Symbol Extraction

Symbol extraction is the process of translating tracked object positions into strings of characters. As we will discuss later, these strings are needed to support efficient search operations through the use of regular expression patterns. Symbol extraction involves two steps: smoothing of the data and mapping of the data into character symbols. The classification of the patterns is tedious when there are noisy symbols that do not add meaning to the activity but are present. We discuss the motivation for data smoothing and the mapping process in the following subsections.

3.3.1 Data Smoothing:

Object tracking is imperfect; the tracks of real-world motion usually do not follow any smooth pattern. For example, even if a car makes a smooth turn, on close inspection the track appears jagged due to small additive noise. Raw tracks, if left unsmoothed, are often

incorrectly mapped. As we shall see in the following section, such an intervening character has no place in the regular expression pattern used to define a northbound left turn; were the character included in the corresponding character string, the regular expression search would fail to find a left turn. The data must be smoothed to improve accuracy. To smooth the data, we use a simple Gaussian filter with a width of α (set to 20) . This means the data can be processed with an 20-frame delay, which is about two-thirds of a second of full-motion video. This straightforward approach to smoothing has a profound effect, not only serving to eliminate faulty tracks that would lead to erroneous classification, but also supports greater accuracy in the data mapping process, as the derivative is less noisy (Figure 3.6).

3.4 Mapping Vehicle Motions to Strings

Data mapping is the process of converting motion vectors into strings of characters which may be efficiently matched by regular expression patterns. To map the data, we consider the points output by the object tracker. Each movement and the direction is encompassed by a character. If it continues to move in the same direction, a series of same characters will be seen and the character changes once the direction varies. In this way, a string that represents the movement and the direction in which the object travelled along the path is obtained. In the ideal case, the vehicle moves exactly one pixel per frame, the mapping conforms precisely to the actual motion. For this scenario, we can simply generate as output one character per frame. However, this is usually not the case. Sometimes, there is not enough movement across frames.

3.4.1 Subpixel Movement

Subpixel movement is the movement of an object across less than one pixel per frame (Figure 3.7) and it occurs quite often in videos of slow moving objects. To account for subpixel movement, we accumulate movement until it can be mapped into a single larger vector. Figure 3.8 (left) shows a smoothed track as a series of dots. Superimposed on this is the track as it would be approximated by a sequence of symbols. In this case, the first

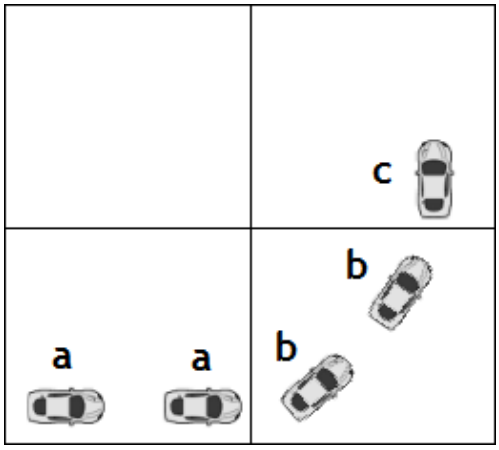


Figure 3.7: Characters describe vehicle motion.

approximated movement is horizontal (symbol a), followed by diagonal direction (symbol b). Figure 3.8 (middle) shows the next sub-pixel movement. At this point, there are two choices.

1. Keep accumulating movements.

2. Map the movement to a symbol.

The decision depends on the amount of error that would result from such a mapping. We take the derivative at the current point and compare this slope against the slopes of possible eight symbols that could be generated. If the slope is between the slopes of two symbols, they become the candidates for the next mapping. In the figure, the candidates for the next mapping are b and c, shown in gray. We seek to minimize the error (i.e., Euclidean distance) between the approximated track and the actual movement. In this case, the distance to the endpoints of the two candidates is greater than the distance to the endpoint of the last mapped symbol. Therefore, error is minimized by not generating a new symbol, and we continue to accumulate the movement. This strategy can also be helpful to differentiate a u-turn from a left turn. During a u-turn, naturally a left turn exists especially if we do not consider the final direction an object is headed. Figure 3.8 right shows what happens after two more positions have been accumulated. Now symbol c

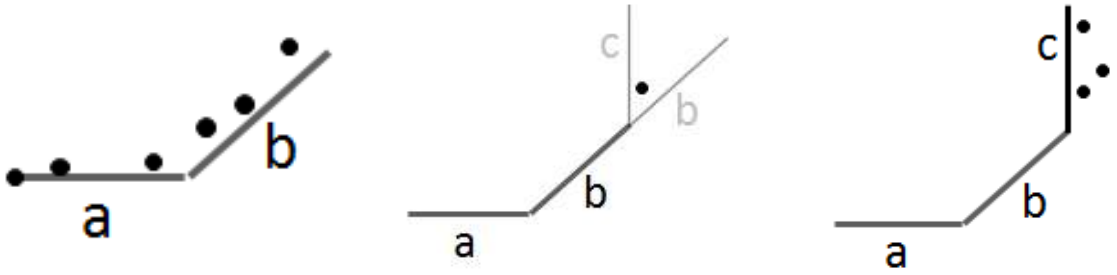


Figure 3.8: Symbolic mapping as trajectory updates.

can be mapped because it is the candidate that minimizes error. The strings representing other trajectory-based motion signatures for vehicles such as start and stop and those for humans (i.e., those associated with walking, running, etc.) can be determined similarly.

3.5 Mapping Articulated Motions to Strings

In surveillance videos, recognizing the various active entities as part of the foreground is the first part in modeling the generic behaviors. Once this has been achieved, the next phase is to classify the motions of these entities into various categories. Earlier, these classifications were done by human operators. When the amount of hours in videos is too large, there appears the need for automating such classification modules. Articulated motions are associated with human activities like digging, waving, boxing, clapping, etc. We use the directional histograms of pixel intensities in the frames of a target video to map them into a regular expression. The directional histograms help in capturing the periodic motion in the articulated activities like clapping, gesturing, digging etc. As an example, lets consider the digging shown in Fig. 3.13 (c). Here, the directional histogram for the x-axis in the lower part of the video is periodic. Similarly, for the hand waving video in Fig. 3.13 (d), the horizontal directional histogram for the upper part of the video is periodic in nature.

3.5.1 Representing Articulated Activities with Directional Histograms

In order to recognize the various activities, we use the concept of directional histograms. Directional histograms are computed separately for both the x and y axes for a two dimensional image. Each histogram represents the mean of the intensity of pixels in either the horizontal (x) or vertical direction (y). Our tracking framework identifies the region of interest from the trajectories of humans or vehicles. The directional histogram algorithm processes the RGB image and converts it to grayscale. Histogram differences are then calculated frame-by-frame from the video sequence. A running average of the histogram differences that decays with time (e.g.150 frames) is accumulated in bins. The first bin represents the difference between consecutive frames and the second bin represents the difference of histograms that are 2 frames apart and so on. A low value in the n^{th} bin would mean the frames that are n bins apart are similar and suggest that the action reoccurs every n frames.

A directional histogram is different from a normal histogram in that it is computed separately for both the x and y axes for a two dimensional image. The mean intensity values along the horizontal direction or the vertical direction are stored in each histogram. Because we work with the intensity of the pixels, we first convert the RGB image into grayscale. As can be followed from Algorithm Listing 1, following this step, the Region of Interest (ROI) [21] of the image containing the activity is computed. The ROI images are read in successively from the video sequence and the corresponding histogram differences are calculated. The running average of the histogram differences are piled up in a queue (currently its of length 150). So, after the third frame, the first value in the Pile will have an average of the histogram differences between consecutive frames (first and second; second and third) and the second value will have the histogram difference between two frames apart (first and third in this case). This is continued as the Pile gets filled. The 150^{th} value in the Pile represents the running average of histogram difference between the

image at current frame and the image at 150 frames ago.

After 150 frames, the queue starts popping the first value and pushing the last value (basically it starts acting like a queue). The values in this pile (histogram), thus represent the difference (or similarity) with the other frames. The first value represents the average of difference of consecutive frames. A low value at n^{th} position would mean that the frames n places apart are similar. Thus, there is some type of action occurring every n frames.

3.5.2 Mapping the histogram data into strings

The first step that is performed to extract the periodicity information available in the histograms, is to map the histogram data into strings that can be represented in a formal language specification at a later stage. To map the histogram data into strings, we fit a linear composition of a mixture of Gaussians with the horizontal and vertical histograms as data as shown in Fig. 3.3 . A Gaussian function is given by :

$$f(x) = ae^{-(x-b)^2/2c^2} \tag{3.2}$$

for some constants a, b, and c. Here, the periodicity information can be approximately mapped to the value $2c^2$. So, we get the corresponding period of motion as $P_i = 2{c_i}^2$. Thus we can derive strings of the form $P_1P_2P_3...P_n$. in vehicles as well as various human activities like walking, running, digging, gesturing etc.

3.5.3 Defining Patterns of Motion

As stated before, search operations which use regular expression matching are efficient because their running time is linear in the number of bytes of data searched. If we use regular expression patterns to describe general motions, we may harness the efficiency of such search operations. Consider the case of the northbound left turn; the vehicle moves from east to north, traveling the intervening northeast direction. The string matching such a motion must begin with a, end with c, and may contain only a, b, and c as intervening

Algorithm 1 Action Detection

function MAIN( )
  $data.Qctr \leftarrow 1$;▷ Increment for the Queue and data is a data structure storing Pile etc.
  $MaxSize \leftarrow 150$; ▷ maximum size of queue and pile
  $data.MultiplyCtr \leftarrow zeros(1, MaxSize)$; ▷ stores the multiplying indices for running averages
  $data.WindowSize \leftarrow 100$; ▷ decaying rate
  for all frame **do**
    $roi \leftarrow$ GetFrame($video$);
    $img \leftarrow mean(roi, 3)$; ▷ converts to grayscale
    $img1, img2 \leftarrow$ DivideImage(img); ▷ divides image horizontally in two halves ▷ Following are the image histograms along vertical/ horizontal direction
    $hist[1] \leftarrow mean(img1, 1)$; ▷ top half horizontal
    $hist[2] \leftarrow mean(img2, 1)$; ▷ bottom half horizontal
    $hist[3] \leftarrow mean(img1, 2)$; ▷ top half vertical
    $hist[4] \leftarrow mean(img2, 2)$; ▷ bottom vertical
    $hist[5] \leftarrow mean(img, 1)$; ▷ complete horizontal
    $hist[6] \leftarrow mean(img, 2)$; ▷ complete vertical
    for all histogram **do** $data \leftarrow$CreateStruct($hist, data$); ▷ creates queue and calls accumulator
    end for
  end for
end function
function CREATESTRUCT($hist, data$)
  if (**then**$Qctr \leq MaxSize$) ▷ Filling up the queue
    $Queue[Qctr] \leftarrow hist$;
    if $Qctr > 1$ **then**
      $Pile \leftarrow$ Accumulate ($Pile, Queue, MulitplyCtr$);▷ Creates an accumulator referred to as Pile
    else if $Qctr > MaxSize$ **then**
      $Queue[MaxSize + 1] \leftarrow hist$; ▷ Adds new hist to first element (Enqueue)
      $Queue \leftarrow$ Dequeue($Queue$); ▷ Removes the first element
    end if
  end if
  $MulitplyCtr \leftarrow$ Increase($MulitplyCtr$); ▷ Keeping increasing the value of each element in vector until a WindowSize is reached
end function
function ACCUMULATE($Pile, Queue, MultiplyCtr$)
  $TempPile \leftarrow$ HistDiff($Queue$); ▷ Stores difference of the current hist with all other hist in the Queue. Current and Last Frame, Current and two frames ago and so on in order.
  $Pile \leftarrow$ RunningAvg($Pile, Temp_Pile, MulitplyCtr$); ▷ Stores the running average of the difference
end function

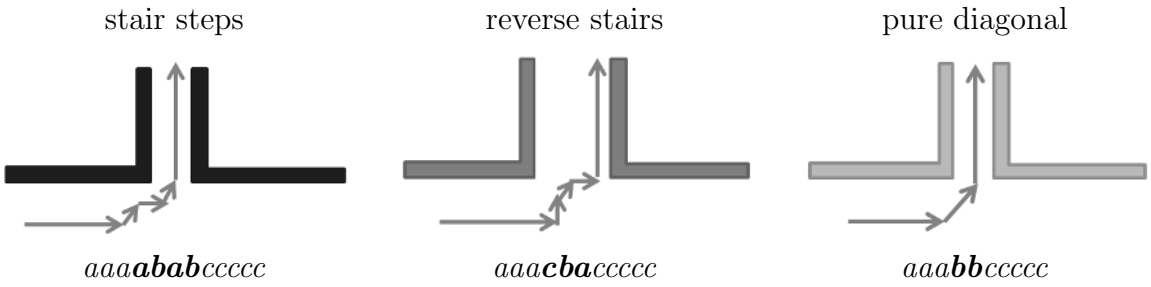


Figure 3.9: Turns encoded with strings of characters

characters; thus the corresponding pattern is $/a[abc]^{+}c/$. The general form of the left-turn expression is as follows:

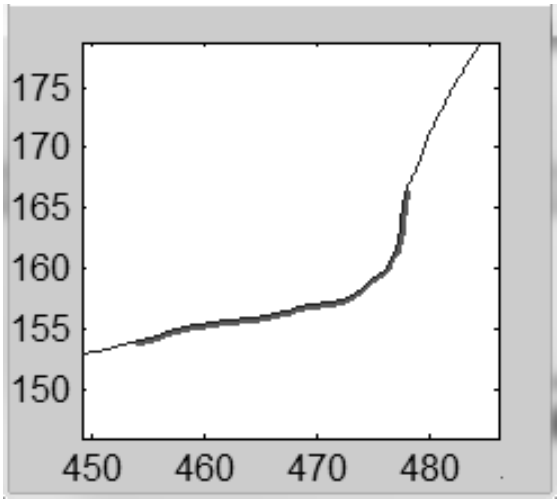
$$a^{s}\{\{a,b,c\}^{l} \cup \{a,b,c\}^{l+1} \cup \{a,b,c\}^{l+2} \cup \ldots \cup \{a,b,c\}^{u}\}c^{f} \tag{3.3}$$

This expression classifies left turns that begin facing due east. $\{a,b,c\}^{x}$ where $x \in [l,u]$ represent the possibility of some variations during the turn which is often the case. A similar expression is used for each of the other 7 directions. Right-turn patterns are simply the reverse of left-turn patterns. U-turn expression are similar except the starting and ending directions are 180 degrees apart, and there are a combination of five symbols in the middle.

The general expression has four parameters s, l u , and f that can be tuned based on data by the analyst.

- s, f: minimum length of the start and finish of the turn
- l, u: lower and upper limit for the middle of the turn

Figure 3.10 shows how different instances of northbound left -turns are encoded. Figure 3.9 shows how an actual turn is encoded. In this analysis, we assume that s=1, which means the start of the turn can be just one symbol; f=3, which means that the finish of the turn must be a string of at least three symbols that are 90 degrees from the start. The middle of the turn is any combination of the three symbols in between and including the start and



baaaaaaa**abaaaabaabaabaabaaaaabaabaaabababbbbcbccc** cbbcbcbcbcbbcbbbbcabc

Figure 3.10: This graph shows the tracking data from an actual turn and the string to which it is mapped.

finish symbol, with a minimum lower length of l= 10 and maximum upper length of u=60. The figure shows a track of a right turn from due east to north, so the start symbol is a, the middle portion is a combination of 35 a's, b's, and c's, and the finish is a string of c's.

The general form of a K-turn expression is as follows:

$$\{\{a\Sigma^* b\}p\Sigma^* q\{a\Sigma^* b\}\}^+ \tag{3.4}$$

where, p and q are strings 180 degrees apart and a and b are mutually opposite strings skewed by 2 to 3 characters and Σ is the alphabet. The regular expression patterns for both vehicle and human trajectory-based activities like start, stop (vehicle), walking, running (human) can be similarly determined.

For articulated activities we can derive regular expressions as well, e.g., the regular expression derived for waving is of the form

$$\{\{c\} \cup \{b\}^*\}^+ \tag{3.5}$$

A detailed discussion on the way these regular expressions are derived is presented in

section 3.5 and further illustrated in section 3.6.

3.5.4 Confidence Measure and Approximate String Matching

Standard pattern matching algorithms for regular expressions provide hard matching. To obtain soft matching, we use an approximate matching algorithm. This algorithm provides a confidence measure for a string approximately matching a regular expression based on how closely it matches the expression. The confidence measure acts as a threshold parameter which we can manipulate to allow certain patterns to be accepted or rejected as a particular turn or activity. It computes the confidence based on the Levenshtein distance between a string and a regular expression. The Levenshtein distance LD between a string s and a regular expression R is given by $min_{s' \in \mathcal{L}(R)} \bar{LD}(s', s)$ where $\mathcal{L}(R)$ is the regular language associated with R and $\bar{LD}$ is the standard Levenshtein distance between two strings, and s and s' have the same length. We have designed an algorithm for computing the Levenshtein distance LD between a regular expression R and a string s; the algorithm is based on repeated depth-first graph search and runs in time $O(k^2)$ where k is the length of the string s. Given the Levenshtein distance LD, the equation for computing the confidence measure of a string s matching a regular expression R is given as:

$$CM = \frac{length(s) - LD}{length(s)} \tag{3.6}$$

where s represents the string whose confidence measure we are calculating and LD is the Levenshtein Distance between R and s. The above method is illustrated in Algorithm Listing 2.

3.6 From Strings to Regular Expressions

While in many cases, the regular expressions representing activities are simple enough to be provided manually, we can also use offline automata learning algorithms for learning regular patterns from positive and negative examples of strings encoding motion charac-

Algorithm 2 Levenshtein Distance Computation

```
    i ← 0;
 2: LD ← String.Length();                          ▷ Livenshtein Distance
    MAPTOSTATES();      ▷ Map characters of Input String to the states of Automaton
 4: if FinalStateReached = AcceptingState then
        StringAccepted = TRUE;
 6: else
        while CurrentState ≠ AcceptingState do
 8:          i ← i + 1;
            BACKTRACK(i);                  ▷ Backtracks i steps through the automaton
10:         LD ← LD − i;
            DFS(i);                        ▷ Performs Depth First Search up to a depth of i
12:     end while
    end if
```

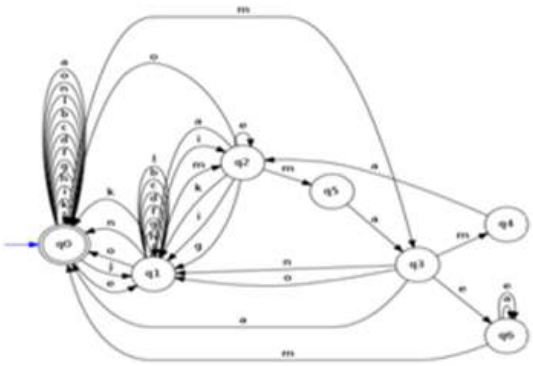


Figure 3.11: Sample Automata created using the RPNI

teristics.

3.6.1 Learning the regular expressions by classification of strings

Once we have formed the strings representing positive and negative examples, the next step is to use a learning algorithm to infer a finite state automaton representing the regular language that accepts strings belonging to the language. For this, we use the RPNI (Regular Positive Negative Inference) offline learning algorithm [17] to learn the finite state automaton. This is implemented using the Libalf (Automata Learning Framework) library available online. The method is illustrated in Algorithm listing 3. Once, we input the strings belonging to the positive and negative classes, the library function calls upon the

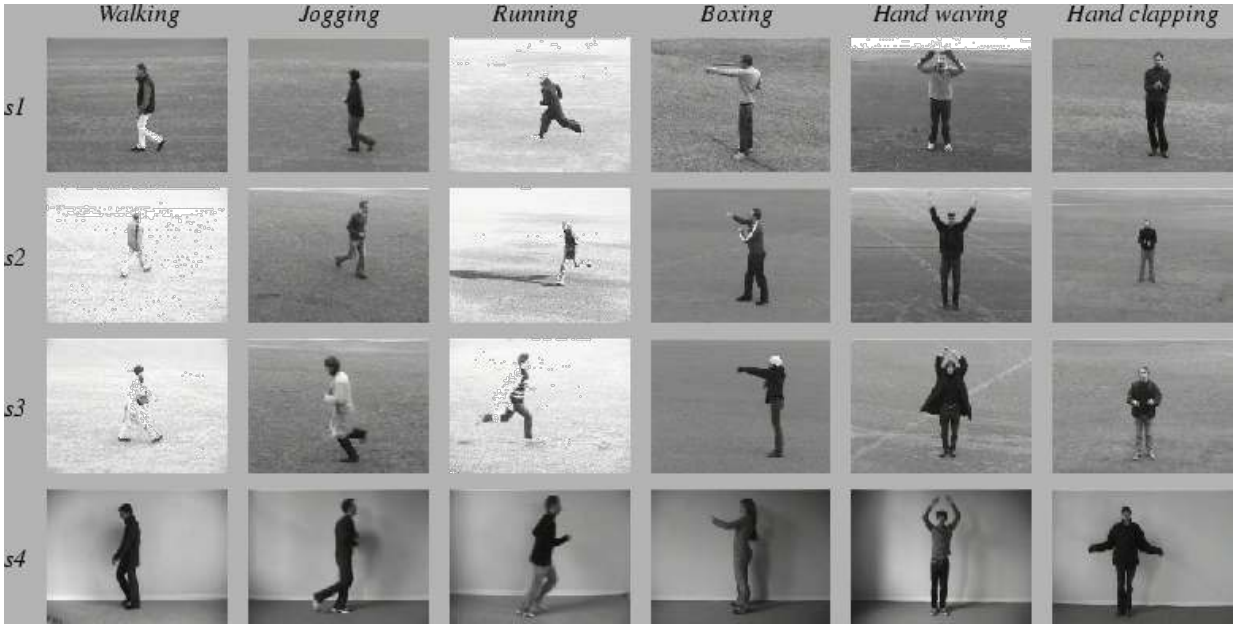
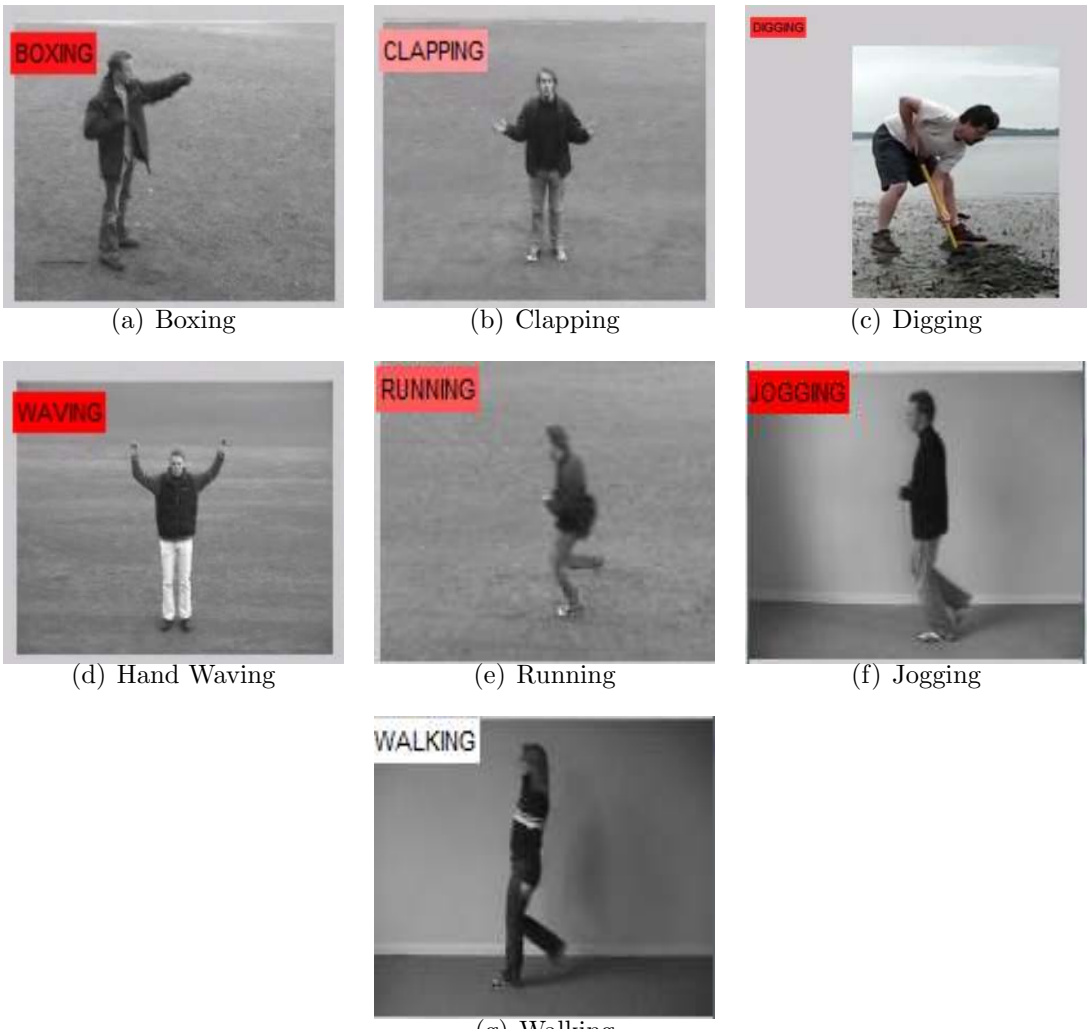


Figure 3.12: Samples from the KTH dataset [86]

learning algorithm to infer an automaton that accepts all the positive examples and none of the negative ones. Next, the automaton is converted to a regular expression using the JFLAP library available online. The input strings representing motion signatures can be matched against the regular expressions to detect trajectory-based or articulated activities. An example of an automata created using the offline learning algorithm can be seen in Figure 3.11.

Algorithm 3 Learner

$P \leftarrow GetPositiveExamples()$;
$P_{class} \leftarrow P.addclassification(1)$; ▷ classification 1 is for positive examples
$N \leftarrow GetNegativeExamples()$;
$N_{class} \leftarrow N.addclassification(0)$; ▷ classification 0 is for negative examples
$AddToKnowledgeable(P, P_{class}, N, N_{class})$;
$CallLearner(RPNI)$;



(a) Boxing (b) Clapping (c) Digging

(d) Hand Waving (e) Running (f) Jogging

(g) Walking

Figure 3.13: Screenshots of different activities detected by our algorithm.

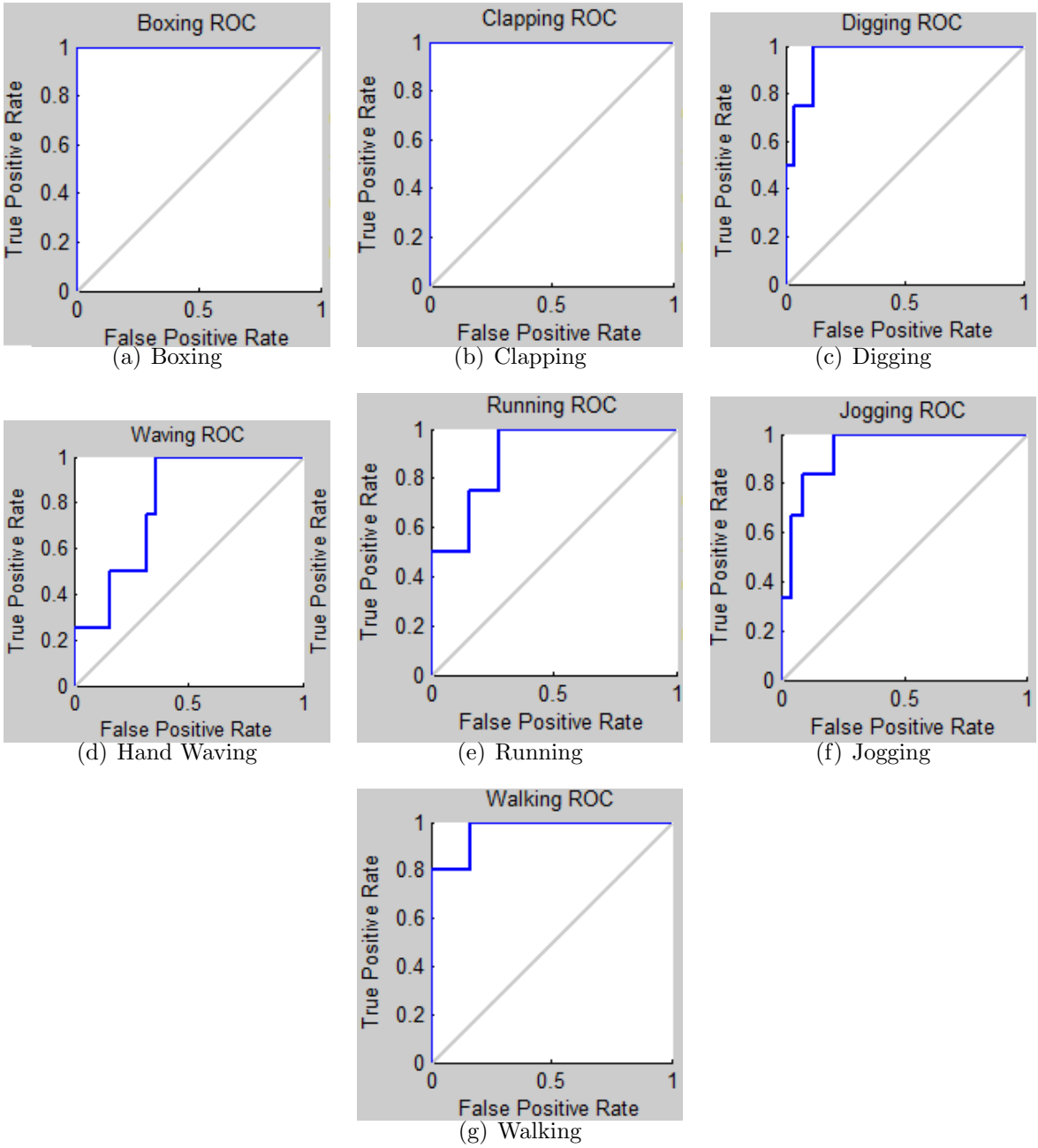
3.7 Experimental Results

3.7.1 Articulated Activities

We performed the experiments on different datasets for the vehicular motion and trajectory based activities. We use multiple binary classifiers concurrently, each producing a probability per activity. By comparing these probabilities, we find the most likely activity. The digging examples are obtained from YouTube while the other articulated activity examples are from the KTH dataset [86], as seen in Figure 3.12. Examples in these datasets are segmented to contain one activity each. Figure 3.14 shows the ROC curves for four different articulated activities recognized by our activity recognizing module. For all these activities, the best algorithms such as [28] produce a correct detection rate of 95.83% while the approach in [50] produces a correct detection rate of 94%. In [97], the authors report their best result to be 92.1% at 4.6 fps. Our framework outperforms these approaches by producing correct detection rates of 96 %. On a 29.97 fps video, our algorithm is able to process frames at 86 fps on an Intel Centrino machine, that is nearly three times real time. The best results were seen on Boxing and Clapping as seen in Fig. 3.14. Waving was slightly difficult to recognize as it was often misidentified. Running and jogging were often confused.

3.7.2 Trajectory Based Activities

Table 3.1 shows the results of using our framework for trajectory based activity examples obtained from the VIRAT Public Dataset [73] where the "Total Expected" column is based on ground truth. On the other hand, the Automata Learning Framework was able to correctly detect 53 out of 78 examples at 16 false positives. Because of the lack of learning examples for u-turns and k-turns, we perform the learning only on left and right turns. The learning framework shows promising results for turns and is expected to perform better in cases where we have a larger training set. Fig. 3.15 shows screenshots of some trajectory-based turn detection examples, whereas Fig. 3.13 shows the results from the activity classification module.



(a) Boxing (b) Clapping (c) Digging

(d) Hand Waving (e) Running (f) Jogging

(g) Walking

Figure 3.14: ROC curves for different activities.

	False Positives	Correct Detections	Total Expected
Left turn	8	4	5
Right Turn	2	7	8
U-Turn	2	2	2
K-turn	1	1	1
Walking	4	16	21
Starting	0	5	6
Stopping	1	3	4
Running	0	1	1

Table 3.1: Detection rates for left turns, right turns, U-turns, K-turns and other events using the string matching approach(without learning).

(a) Right Turn

(b) U-turn

(c) K-turn

Figure 3.15: Detection of Right turn, U-turn and K-turn by our Algorithm.

Chapter 4
Core Sampling Framework

The core sampling framework was designed to do pixel level classification and regression in images. It also is able to make use of pretrained networks instead of training a large network. Large networks with large datasets obviously need a large amount of processing power and time. These large networks learn a lot of knowledge about the datasets. We want to be able to use a subset of features that are only applicable for our experiments. For example a large network maybe able to distinguish between thousands of varieties of animals. While learning the knowledge needed to distinguish such animals, it possibly learns details about the shape, size, color, texture, positions of various shapes at various places etc. Both high level features and low levels features will be learned. We may not need the information needed to distinguish between animals but all we may require is the pre trained network's ability to extract the general shape of any object (or animal) in an image. Pixel-wise prediction/classification can help in scene understanding. Distinguishing pixels belonging to foreground and background have been quite common in the past. We take it a step further by classifying each pixel into even more classes. Hence, it gives us a finer segmentation. Because recognizing an object as a tree is not enough, the goal is to be able to color parts of the tree different, and color leaves separately from the branches and the background. Also, even in the same leaf not all pixels need to be the same color. *Core sampling* is a term used in engineering and science where a long cylindrical sample is

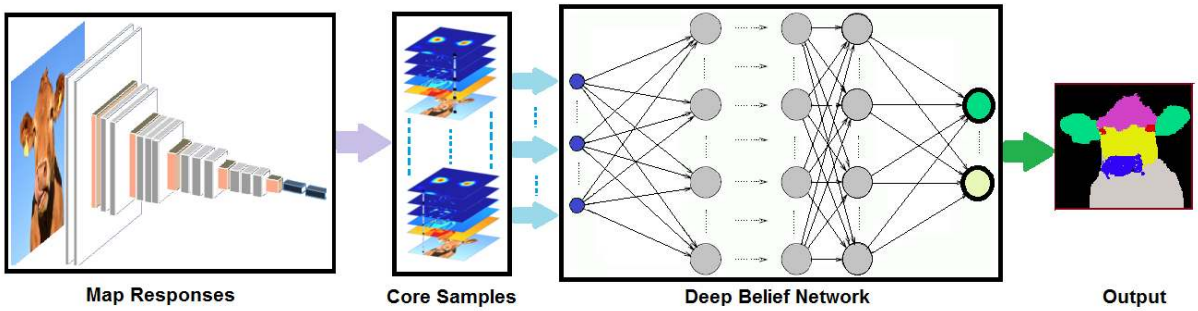


Figure 4.1: The architecture of the Core Sampling framework.

extracted for analysis natural materials, climatic record from ice cores etc. Convolutional Neural Networks (CNNs) work on parts of the input locally usually at different pyramidal levels. The lower layers encode the pixels while the higher layers provide representation of objects comprising of those pixels that eventually help in understanding the entire image. Pixel wise classification and image understanding can be improved with the local and global information encoded in different layers of a CNN. The information from these layers can be stacked at different pyramidal levels, and used as a *core sample* that can enables better understanding of an image. A CNN convolves the input image and/or parts of the image with filters. Training a CNN involves determining what these filters need to be to get the desired output for a given input. We use the VGG-16 [90] model to bootstrap our framework. The ImageNet dataset was used to trained this model [56]. It comprises of more than a million training images and 1000 classes [56]. The intermediate maps, that are generated for each pixel during the testing phase when images are input to the above model, when stacked together and resized to a uniform size, form hypercolumns. The second stage of our framework consists of a Deep Belief Network (DBN). Deep Belief Networks are unsupervised deep learning models [45]. Figure 4.1 describes the architecture of the core sampling framework. The left-most box represents an image being passed through the VGG-16 architecture. There are numerous map responses from each layer of the network. These map responses from multiple images are then up-sampled and aligned together and samples from them are used to train a Deep Belief Network (DBN). The DBN treats each pixel and corresponding features (map response values for that pixel) as a data point and uses the labels supplied to it to train a network that can perform the desired task.

4.1 Pixel Level Classification

We use hypercolumns introduced in [42] as a data structure for representing the layer outputs from a CNN. A hypercolumn for a pixel in an input image is a vector with k columns, where k is the number of intermediate maps in the VGG-16 model, with each component of the vector being a map. A hypercolumn does not preserve any spatial

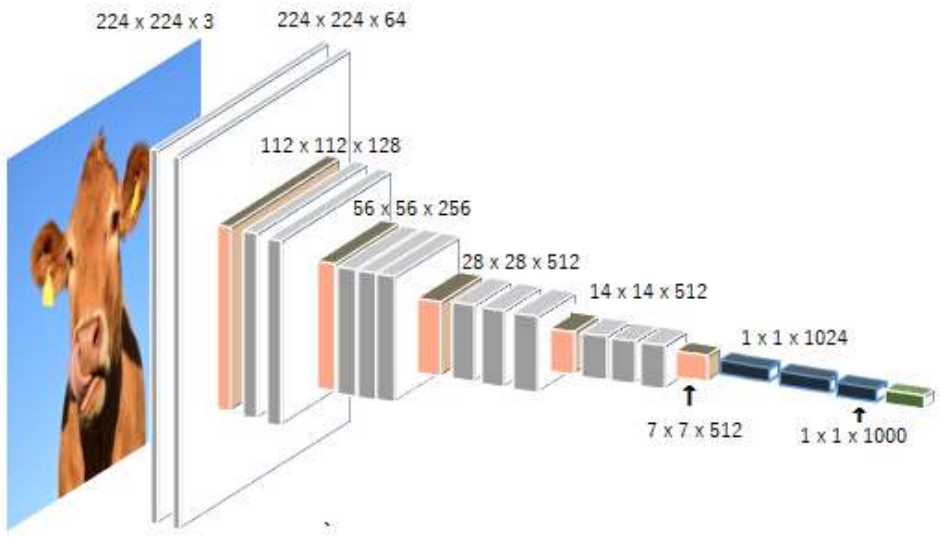


Figure 4.2: VGG-16 Architecture. There are multiple convolution layers, max pooling layers, fully connected layers and a softmax layer.[90]

correlation between the constituent maps. A *core* is a collection of hypercolumns, one per pixel in an input image. A random sample drawn from a core is called *a core sample*. Core samples generated from input images a fed to the second stage of our framework. The first few layers are used for accurate localization of an object and the layers close to the output layer help to distinguish between different objects. Pre-trained VGG-16 [90] model for bootstrapping the core sampling framework. Fig. 4.2 shows the architecture of the VGG-16 network. There are also pooling layers between the convolution layers and fully connected layers at the end [90]. This network is trained on the ImageNet dataset which has a large variety of objects. This makes it a perfect model to construct a framework that works for a variety of datasets [99]. Our framework uses the intermediate maps that are acquired during the testing phase when images are input to this model. The individual pixels of the intermediate maps, when stacked together and resized to a uniform size, form hypercolumns. Each pixel's value, combined with the map values produced using the pretrained model, is used as a data point. The map values are thus the features for their respective pixels. While using the maps from just 5 convolutional layers of the pretrained model, the number of maps per image is already around 1500. As the size of the core

sampling data (processed output data from the pretrained network, described in 4.1.3) gets large, we came up with with the use of response maps from multiple layers of the CNN. For this reason we use a randomly sampled subset of pixels to train the DBN in the second stage.

4.1.1 Preprocessing and Data Augmentation

The input to the pre-trained VGG-16 model needs to be of the size 224 by 224. The BAERI dataset (see Section 4.3.1) consists of raw images at inconsistent intensity levels and variable image sizes. Resizing the images would create images that are at different scales. So we added padding around smaller images to create 224 by 224 images. Because we did not resize them, the scale information remained intact. For the same reason, we created sub-images (tiles) from larger images before extracting the map responses. These image tiles are created by using a sliding window of 224 by 224 and a smaller stride size. As before, there is no resizing of images; hence there is no need for scale normalization. We also generated more data by varying contrast to improve robustness to images that the framework might not have seen and to create more training data for the next stage. The map responses, which are now used as features, are individually normalized and the same normalization parameters are used for the corresponding features during testing. CAMVID consists of images of the same size (480 x 360) and are at the same scale, being a standard dataset; hence not much preprocessing or data augmentation needs to be done.

4.1.2 Response Maps

The layers of a deep neural network learn different features at different layers or combinations of layers. The first layer of the network learns features that are similar to Gabor Filters or Color blobs [99]. The deeper layers help to discriminate objects and parts of objects while losing spatial and local information [100]. Hence, the combination of maps at different layers helps to capture the spatial as well as the discriminative features. In [100], it is pointed out that removing the fully connected layers as features had resulted in very little increase in the error rate. Since the response from the fully connected layers is a

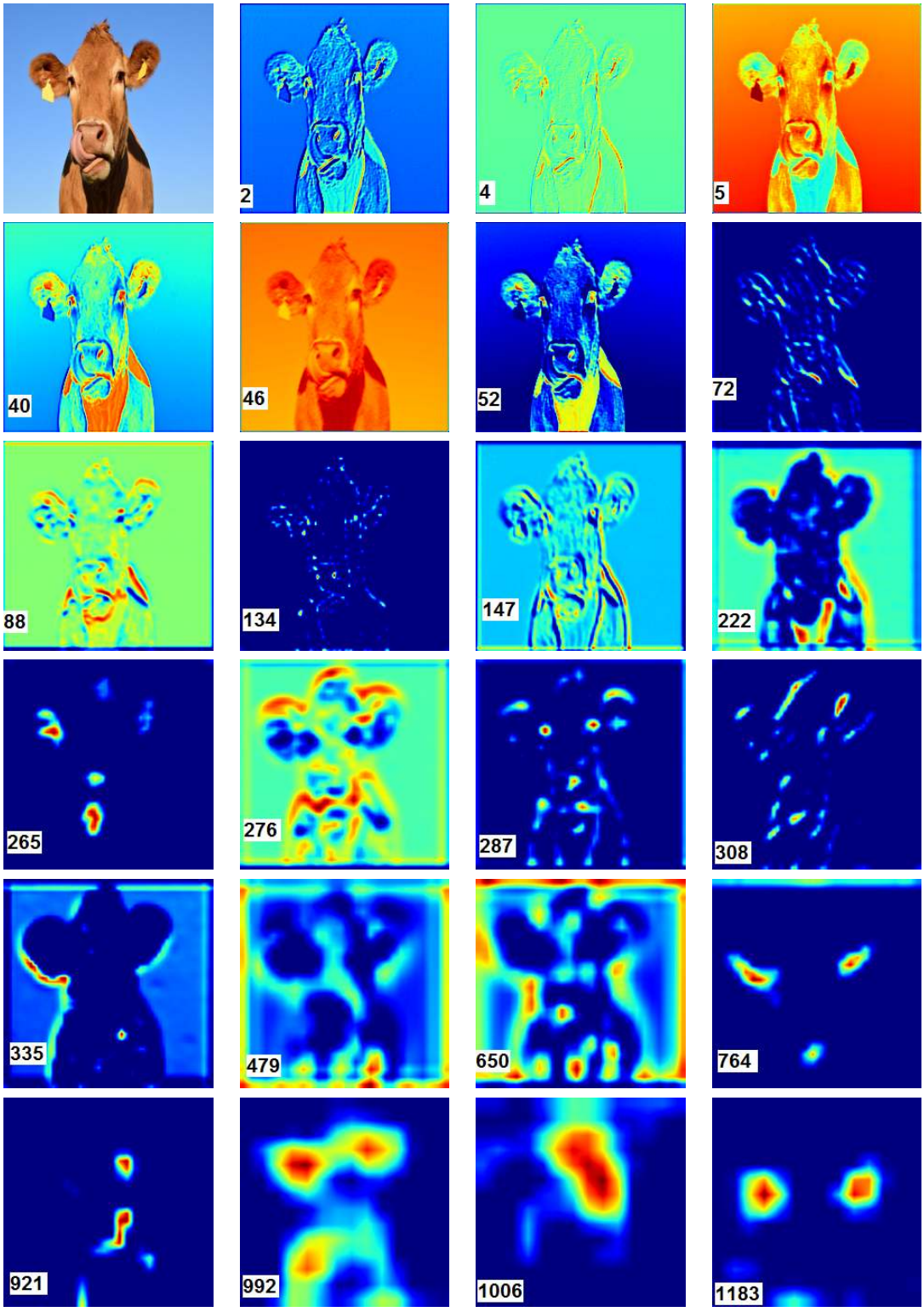


Figure 4.3: Response Maps resized to original image size. Higher number indicates maps from deeper layers.

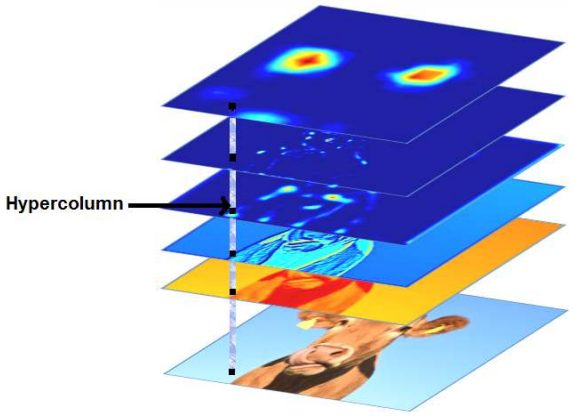


Figure 4.4: Response maps when stacked together. Each pixel and the stacked response pixel together forms a hypercolumn. These response maps values are used as features for our deep belief network

vector (either of size 1024 or 1000) resizing it would drastically increase the size of the data without any significant increase in performance. We can see from Fig. 4.3 that the deeper maps extract more and more abstract features. For example, the 1183rd map identifies the eyes/ears of the cow whereas the first few maps detect the edges of the image. The deeper maps, however, lose the detailed spatial information about the objects.

- **ImageNet**

The ImageNet [56] dataset is database of images. It consists of hierarchically organized images. The quality of the images and the annotations are maintained by the ImageNet Team. The broader types of images in the dataset are [56]:

1. plant, flora, plant life
2. geological formation, formation
3. natural object
4. sport, athletics
5. artifact, artifact

6. fungus

7. person, individual, someone, somebody, mortal, soul

8. animal, animate being, beast, brute, creature, fauna

9. Misc

- **VGG-16 (Visual Geometry Group's 16 layer architecture) [90]**

The pretrained model that we use is a 16 layer architecture that is trained with the ImageNet dataset [56]. The VGG-16 [90] architecture performs up to 7.4% classification error on the ImageNet test case. It consists of large number of convolutional layers, with a large amount of 3 x 3 convolutional filters. The layers of the VGG-16 architecture is described in Figure 2.2. The first two layers are of the same size as input (224 x 224) [90]. There are 64 such convolutional filters in each. They are succeeded by a max-pooling layer of half the size (112x112)[90]. The next two convolutional layers consists of 128 filters [90]. It is then trailed by another max pooling layer of half the size (56 x 56)[90]. There are 3 more convolutional layers consisting of 256 filters and then, two more sets of max-pooling layer succeeded by 3 convolutional layers of sizes 28x28 and 14x14 follow [90] each with 512 filters. There is then a 7x7 maxpooling layer for each of the 512 filters [90]. Towards the end, there are dense layers of 1 x 1 x 4096 that are fully connected, a drop out layer and final soft max layer for the final output layer [90].

4.1.3 Core Sample: Intermediate Data Representation

Since, we extract maps from a pre-trained model, we normalize the images by subtracting constants from the Red, Green and Blue values (the same procedure that was used while training the original model). From each image we then acquire the map responses from each layer. Most of the map responses are n x n shapes ($n \in 2^i$, i = positive integer). Each of these map responses of various sizes are then resized to the input image size using bilinear interpolation. These map responses are then stacked along with original input image. From this point on wards, each pixel is a distinct data point with the map response

values as its features. We define a *core* as a collection of hypercolumns, one per pixel for an input image. *Core samples* are random samples drawn from a core. We feed the core samples to the second stage of our framework. When we are dealing with classification, we are expecting a data point to have a single label value, when we train on this data in the second stage of our framework. In case of colorization, we use a regression layer where we have normalized the possible output between 0 and 1. Also, in case of Colorization, we are predicting two different channels (Cb and Cr), and both the output can have values between 0 an 1.

4.1.4 Prediction using Deep Belief Network

We use unsupervised pretraining using RBMs followed by supervised learning using DBNs for the final pixel-wise prediction. Our framework is flexible; depending upon the task at hand, different types of output layers can be used. We have implemented two different types of layers: a regression layer that implements linear regression a logistic regression layer . For the loss function, the mean-squared error (MSE) is an appropriate function for regression and for the logistic regression layer that is used to classify pixels a negative log likelihood loss function is preferred. For example, if two pixels have red (R) values 0.5 and 0.6 they are more similar than if they were 0.1 and 0.6. Hence, the use mean squared error function is more appropriate on regression problems. Mean Squared Error function is given by,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2, \tag{4.1}$$

where $\hat{y}$ is vector of predicted values and y is vector of actual values for n observations. On most neural networks that classify rather than perform regression, the distance between any two labels would be the same. In such cases, the likelihood ($\mathcal{L}$) and loss (l) functions are given by:

$$\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} log(P(Y = y^{(i)}|x^{(i)}, W, b)) \\ l(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D}), \tag{4.2}$$

where W, b are weights and biases respectively and $\mathcal{D}$ is the dataset [1] . Given an input, the weights matrix, and a bias vector, it outputs the likelihood for a given input x^i pertains to a class represented by y^i. Since this equation is based on probability values rather than distance measure, it is more suitable for classification.

However, the two tasks that we evaluate our framework on, only deal with the classification task. We however have demonstrated examples where the regression layer can be useful. The input to the data is the intermediate data representation that we described above. The map responses from each layer is normalized using standard feature scaling. The unsupervised training helps us to cluster the features together further, and help to converge the network faster. Since, we only deal with classification for the we use negative log likelihood as our cost function.

- **Why DBN**

If we cluster images by average map response, we can see a Convolutional Neural Network gets more organized every layer as larger chunks are recognized. In simple cases where many high level features / object parts are recognized, a standard two layer network should be enough. A standard CNN is the extreme example of this: the convolutional layers handle the bulk of the data abstraction, and the fully connected layers are only receiving the very abstracted data from the bottom layer. If the maps from the upper layers are a major part of solving the problem, a DBN should be used to ensure the data is properly abstracted. In [99], the authors showed how CNNs lose generality as data travels through them; so this would certainly be the case when trying to make predictions about images that differ considerably from the original CNN training images (as is the case for our data).

Similarly, the more the classification problem diverges from simple object recognition, the less our desired output distributions will resemble the internal distributions in the CNN, and the more additional levels of abstraction will be desirable.

We train our data by first doing a pre-training step with persistent chain contrastive divergence (P-CD) and then fine tuning the network using a deep neural network. Keeping the pretrained CNN and the sample interpreting DBN distinct makes it easier to pre-process the data and also to normalize the testing set with respect to the training data. We used L1 and L2 norms for regularization and implemented dropout on the hidden layers.

4.1.5 Core Sampling Algorithm

Algorithm 4 implements the core sampling framework. There are two modes of running the framework: training (Line 2) and testing (Line 6). In both cases, we generate core samples from images in a folder and normalize them before training or testing begins. The function CoreSample() loads the pretrained model and the images and iterates through the images. The aforementioned preprocessing/normalizing of the images is done and then each image is fed into the model. GetFeature() extracts the feature maps and then each map is upscaled to a uniform size of $W \times H$ using bilinear interpolation where W and H are the width and height of the input image. Then we concatenate all the cores into a single array and normalize them. The normalization parameters for each feature map are separately maintained. During training, an array of target vectors is created by using the labeled images. Normalized samples from the cores and the labels are used as training data for the Deep Belief Network and the trained model is saved to hard disk. During testing, we load the model that we trained using our interpretive network (DBN) feed the core data in and finally make our prediction.

4.2 Image Colorization

We originally intended to use our core sampling framework on image colorization. We tried to train the framework on the entire samples from the ImageNet dataset. The amount of training samples was huge (more than a million images). We were not able to

Algorithm 4 Core Sampling FrameWork

```
1: function MAIN(mode, params, path)
2:     if (mode = training) then
3:         X, Y ← CORESAMPLE(True, path)
4:         model ← trainDBN(params, X, Y)   ▷ Trains the DBN and returns a trained interpretive model
5:         SAVEMODEL(decision_model, params.name)
6:     else
7:         X ← CORESAMPLE(False, path)
8:         model ← LOADMODEL(params.name)
9:         output ← PREDICT(model, X)
10:    end if
11: end function
12: function CORESAMPLE(isTrain, path)
13:    model ← LOADPRETRAINEDMODEL(name)
14:    images, labels ← LOADIMAGES(path)
15:    for each instance in images do
16:        instance ← PREPROCESS(instance)
17:        feature_maps ← GETFEATURE(instance)
18:        for each map in feature_maps do
19:            upscaled ← IMRESIZE(map, (W, H), 'bilinear')
20:            core ← RESHAPE(upscaled, WXH)   ▷ converts upscaled to a vector of length W x H
21:            X ← CONCATENATE(X, core)
22:        end for
23:    end for
24:    X ← NORMALIZE(X)
25:    if (isTrain) then
26:        Y = CREATETARGETS(labels)   ▷ Uses label images to create targets
27:        return X, Y
28:    else
29:        return X
30:    end if
31: end function
```

train the network efficiently. We are only able to load about 20 images worth of pixels in the Graphical Processing Unit (GPU) at a time.

4.2.1 Preprocessing Data

Color images consists multiple channels. Some of the popular ways to represent color images are using RGB, YUV, CMYK, HSI, HSV, YCbCr etc [52]. These are different methods to internally store images. RGB is the most common method to color modes to represent images. These images have 3 channels where R channel represent the Red value, G channel represent the Green value and B channel represents the Blue value of each pixel in that image. The composite of all three images gives the final color image. Using the RGB model is not the best choice for colorization problem. These three channels contain more information than just colors. During colorization, the grayscale images already have the outline, texture and other information about the images. They are just missing the color components. Hence, we use the YCbCr color model where we the grayscale image is used as the Y component and we predict the other two channels (Cb and Cr) which consist of the color information. For training purposes, we convert the color images in to YCbCr where Y is treated as the data and CbCr are the labels.

4.2.2 Regression

We use regression instead of classification for predicting the color values. The Cb and Cr values are predicted in the range from 0 to 1. We add the regression layer in the DBN of our core sampling framework as the output layer for the prediction. For the cost function, we use the mean squared error function.

4.2.3 Complications

We focused mainly on landscape images with 3 different types of objects in our testing of the image colorization problem: blue skies, forest and desert from public online images. However, on a larger dataset there were some improvements required to outperform state of the art. There were issues with storing very large size data. The core data increased

(a) Sample Test Image (b) Sample Colorized Image

Figure 4.5: Colorization Example [40]

the size of the dataset by more than 1500 times. With a bigger pretrained network and more maps from more layers, the size of the data would increase further. One way we tried to combat this issue was use Core Samples. These are randomly chosen samples from the Core Data. However, even with this and the significantly large amount of possible types of images, further improvements were needed to use the framework to colorize any input image.

4.3 Core Sampling Framework for Segmentation

Segmentation involves dividing an image into parts or segments based on some meaningful criteria to make it easier to analyze the image. Pixel wise segmentation allows us to get a finer boundary between segments even on highly irregular shapes. We use two different datasets for the image segmentation problem. On both the datasets, the prediction is only on one channel. Each pixel can be classified as belonging to one of multiple classes

and the entire image is segmented using pixel level prediction.

4.3.1 Datasets

The CAMVID dataset [23], [22] consists of 32 semantic classes of objects out of which, like most of the other approaches [57], [102], we evaluate our algorithm on the 11 major classes and 1 class that includes the rest. These classes are Building, Tree, Sky, Car, Sign-Symbol, Road, Pedestrian, Fence, Column-Pole, Side-walk and Bicyclist [23], [22]. The training set includes input images, that are regular three channel color images and the targets are segmented single channel images. The images are from a few videos taken from inside a car driven on streets. These consist of labeled images with 367 training, 101 validation and 233 test images of consistent sizes at the same scale [23], [22].

The BAERI dataset [38] that we introduce here consists of imagery collected from a Synthetic Aperture Radar (SAR). Both the input and output are treated as single channel images. The input single channel image consists of SAR values and the labels are the ground truth values at each of the pixels. Labels were obtained by morphological image processing techniques for noise removal, i.e., opening and closing. The erosion and dilation on the images fail to remove all the noise in the ground truth images. Therefore, there are certain areas in the ground truth images that still contain some noise with incorrect labels. As, we are classifying each pixel separately, the noise must be taken into account during training. The values in the training images were in the range between -40 and 25. In the BAERI dataset, the pixel classes are those belonging to the ship class and the rest. There are only 55 images of variable sizes available in this dataset with the total size of 68 megabytes.

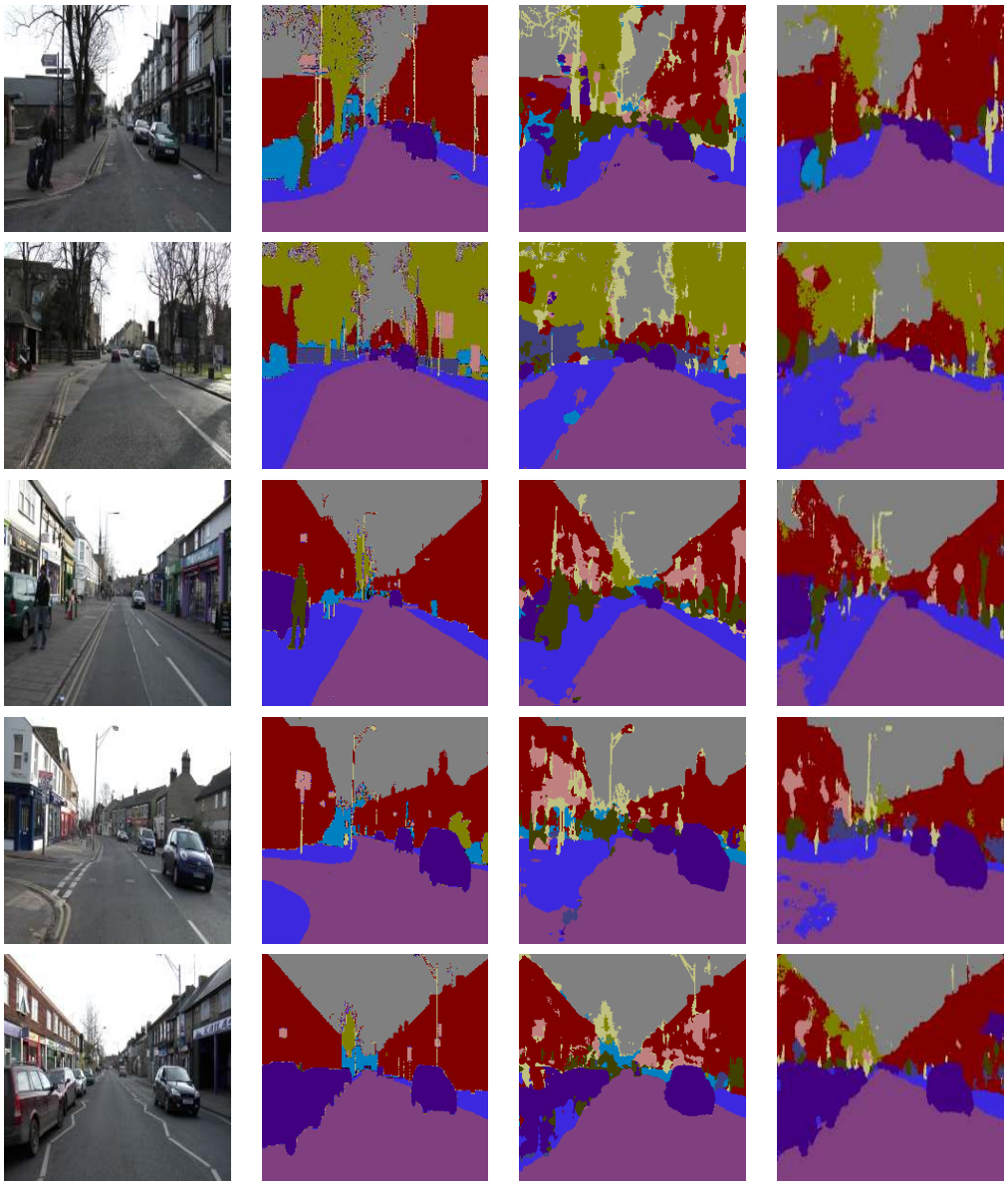


Figure 4.6: Results on the CAMVID dataset. [23], [22] The images from left to right: a) Original Image b) Ground Truth c) Our Algorithm d) SegNet [8]

4.4 Experimental Results and Discussion for Classification

4.4.1 Experimental Setup

We use negative log likelihood as our cost function for Classification. We trained the DBN in the second stage by first performing a pre-training step with persistent chain contrastive divergence (P-CD) and then fine tuning the network using a deep feed forward neural network trained by backpropagation. Keeping the pretrained CNN and the DBN interpreting the core samples distinct makes it easier to preprocess the data and also to normalize the testing set with respect to the training data. We used L1 and L2 norms for regularization and implemented dropout on the hidden layers. The input to the DBN are the core samples that we described above. The map responses from each layer of the CNN are normalized using standard feature scaling. The unsupervised training helps us to cluster the features together further, and helps to converge the training faster. We used the theano [92] deep learning library and an Intel i7 six core server with TITAN X GPU for our experiments.

4.4.2 Analysis on the BAERI Dataset

Between the CAMVID dataset and the BAERI one, there are more objects in the CAMVID dataset that are also present in the ImageNet dataset. On the other hand, the BAERI dataset is quite different from ImageNet because it contains SAR data not present in ImageNet. As a result of transfer learning, the knowledge acquired from ImageNet based the wide variety of features abstracted at various levels by the pretrained VGG-16 network prevented the sparsity of the BAERI dataset from creating any problem in training the DBN in the second stage. On the BAERI dataset, our framework was able to slightly outperform SegNet (see Table 4.1). The output images of SegNet had less blobs that could be classified as noise but it also missed a few of the smaller ships and had a larger area of pixels incorrectly classified as ships around the main clusters compared to our algorithm.

Table 4.1: Results on BAERI Dataset

Metric	Our Method	SegNet [8]
Accuracy (%)	**99.24**	98.08
Mean Squared Error (MSE)	**.0115**	.0142

Table 4.2: Results on CAMVID Dataset

Classes	Our Method	Boosting (CRF + Detectors)[57]	Dense Depth Maps [102]	SegNet [8]
	367 Training Images			3.5 K Images
Building	71.0	81.5	85.3	**89.6**
Tree	62.6	76.6	57.3	**83.4**
Sky	**96.8**	96.2	95.4	96.1
Car	72.2	78.7	69.2	**87.7**
Sign-Symbol	52.3	40.2	46.5	**52.7**
Road	80.4	93.9	**98.5**	96.4
Pedestrian	56.4	43.0	23.8	**62.2**
Fence	48.1	47.6	44.3	**53.5**
Column-Pole	**39.5**	14.3	22.0	32.1
Sidewalk	77.3	81.5	38.1	**93.3**
Bicyclist	**38.5**	33.9	28.7	36.5
Class Avg.	63.2	62.5	55.4	**71.2**

All result images can be seen on [3]. Qualitatively, our algorithm performed better on the BAERI dataset than SegNet.

4.4.3 Analysis on the CAMVID Dataset

On the CAMVID dataset[23], [22], the core sampling framework outperformed both [57], [102] on 10 of the 11 classes in terms of accuracy and had a better per class accuracy (see Table **??**). Both [57], [102] and our framework were trained on 367 labeled training images. As can be seen from Table 4.2, our framework could not match the performance of SegNet on the CAMVID dataset in terms of accuracy except for the Sky, Column-Pole, and Bicyclist classes where it outperformed SegNet. However, while our framework was trained on 367 labeled images, SegNet was trained on 3500 labeled images.

Chapter 5
Pixel-level Reconstruction and Classification for Noisy Handwritten Bangla Characters

Dimensionality reduction is a key concept in Machine Learning. The reduction in the number of random variables to a least set of principal components / variables makes training and learning more efficient. These learning algorithms that reduce dimensions make the model simpler and easy to train. Quadtrees are decomposed data structures usually constructed out of images by dividing the image into quadrants and subdividing each quadrant recursively. We have used probabilistic quadtrees have been used to represent character images and classify them using a deep belief network (DBN). Deep Belief Networks use hierarchical representations to generalize and learn from data in a domain by first pre-training a Restricted Boltzmann Machine (RBM) [14]. We also make use of transfer learning to make use of pretrained networks to facilitate the fast training of our algorithms. In the next sections, we will discuss how we use these techniques to focus on character images on the Bangla and n-MNIST dataset [61],[11] . We use the same dataset provided for n-MNIST data and create a new dataset by adding noise to the Bangla Basic Character Dataset. On the Noisy Bangla Datasets that we focus on, [11] have used probabilistic quadtrees to learn sparse representations of handwritten character images and have used a two layer DBN for the classification. Quadtrees are decomposed into blocks and a homogeneity criterion is used to decide whether or not to decompose them further into smaller blocks. We inject three types of noise: random gaussian noise (awgn), noise due to camera or object movements (motion blur), and noise due to poor illumination (reduced contrast) into the Bangla Basic Character dataset consisting of 50 classes resulting in the Noisy Bangla Basic Character Dataset. In [11], once a block has been chosen for decomposition in any one image based on the homogeneity criterion, that block is identically decomposed for every other

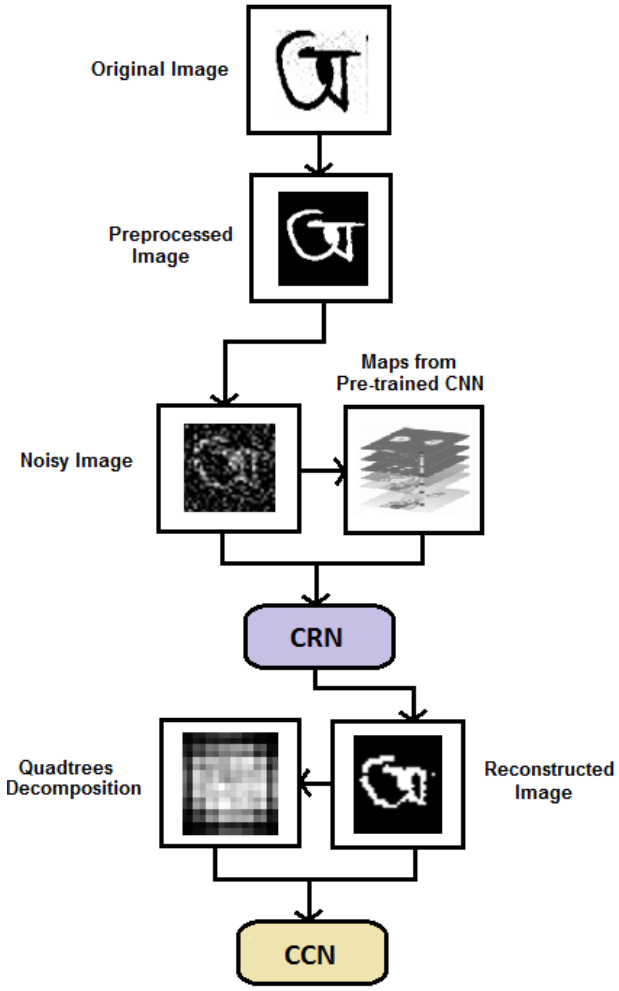


Figure 5.1: The architecture of our character recognition scheme.

image. We build upon this approach and use a saliency map to improve the representation and also use another DBN for character reconstruction removing noise.

5.1 Preprocessing Data

The datasets that we evaluate our algorithm on consist of three types of handwritten characters: a) Bangla Basic Characters b) Bangla Numeric Characters c) Noisy MNIST dataset. There are 50 classes in the first type and 10 in the other two. Preprocessing the data is required for the Bangla Datasets as the images in the dataset provided to us were all of different sizes. So, our preprocessing stage involves creating a dataset that is similar

Figure 5.2: Original Images Bangla Numeral Characters. One image for each Bangla Numeral. [15]

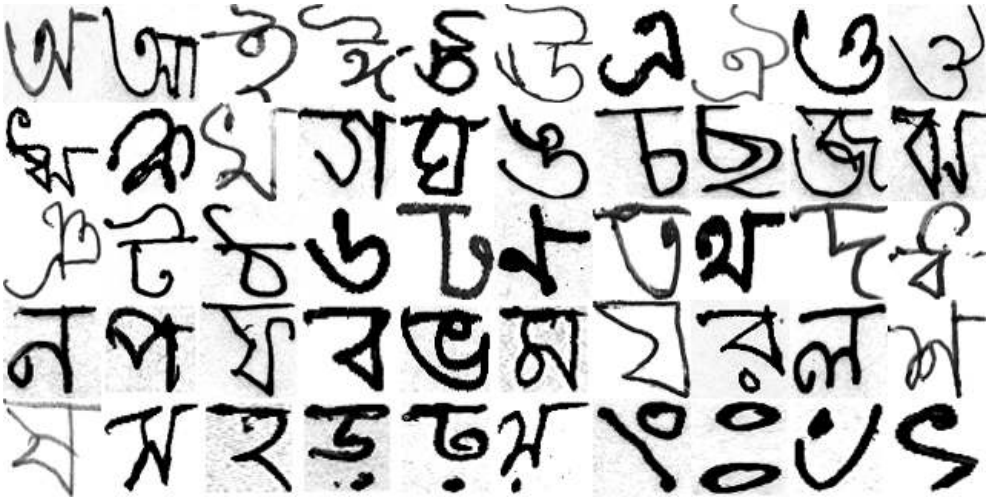


Figure 5.3: Original Images Bangla Basic Characters. One image each of the 50 Bangla Basic Characters.[16]

to the standard datasets for character recognition such as MNIST [61].

5.1.1 Standardize Raw Data

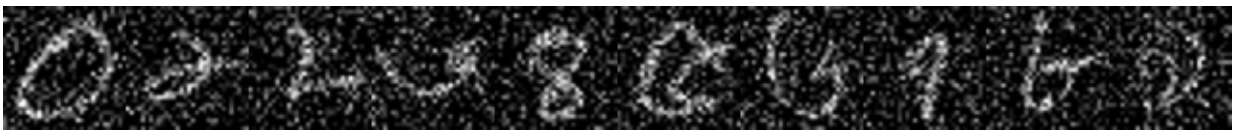
The n-MNIST images are already standardized. The Bangla Basic Characters and Numerals images are all of different sizes, have different intensity levels for the foreground and background pixels, and contain light blobs not part of the characters. These raw images are first processed using the non-local means denoising algorithm [24]. Non local means is unique in that it adds error even in the denoising process. Instead of looking at just the pixels surrounding the image, nonlocal means also looks at every pixel. After performing the non-local means denoising algorithm, the resulting images are bimodal in nature, with the pixels belonging to the background having value around one and the rest, belonging to the character or noise, having lower values. The next step is to use Otsu's binarization scheme [74] to threshold the images to binary. Otsu's binarization is useful for bimodal images because we can take the approximate mid value of the peaks as a threshold value [2]. It calculates the threshold value automatically using the histogram of the bimodal image. In the subsequent binarized versions, we set the values of the pixels in the background to 0 and those in the foreground to 255. Following the procedure described in [11], we then find the largest connected component for each image and center the image around that component. We pad the images, to create square images, resulting in at least 10 pixel long borders on all sides. To avoid large boundaries, we crop images that have too many background pixels surrounding the characters. Fig. 5.3 shows one sample image each from the 50 different classes of the Bangla Basic Characters dataset and Fig. 5.2 shows the 10 different Bangla Numerals.

5.1.2 Bangla Noisy Data Creation

Three kinds of noisy handwritten character image datasets are created from the pre-processed dataset by adding three distinct types of noise. Similar to [11], we create: a) the *awgn* noisy dataset by adding white gaussian noise(awgn) to the preprocessed dataset. The signal to noise ratio (snr) for these images was: 9.5, b) the *contrast* noisy dataset by

(a) added white gaussian noise (awgn)

(b) motion blurred



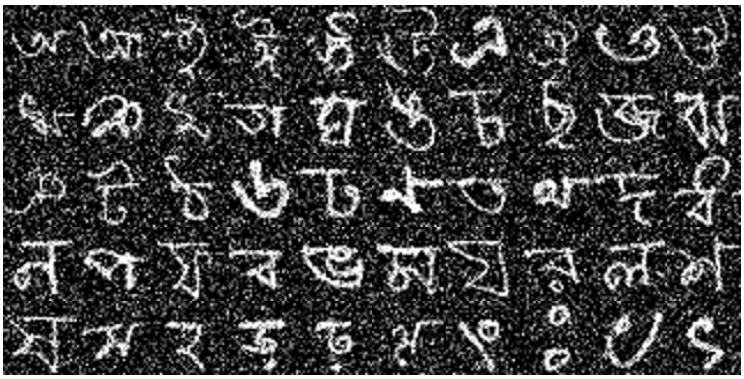
(c) decreased contrast and awgn

Figure 5.4: Noisy Version of the Bangla Numerals.

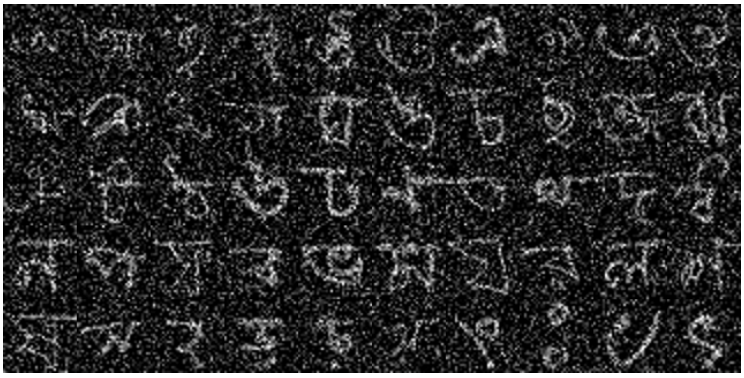
dividing the intensity of the preprocessed images by 2 and awgn with a snr of 12, and (c) the *motion* blurred noisy dataset by blurring the images with a linear motion of 5 and an angle of 15 degrees in the counterclockwise direction. Fig. 5.4 shows the samples from the noisy Bangla Numeral dataset for each of the three noise types and Fig. 5.5 shows the samples from the noisy Bangla Basic Characters dataset for each of the three noise types. The three noise types we added are commonly found in images due to camera movements, poor illumination, high temperature, and movement of objects [103]. Natural images taken from cameras, scanner etc. often contain such noises that need to be taken into consideration during classification tasks.

5.1.3 Ground Truth for the Character Reconstruction Network

We do a pixel-level character reconstruction to clean the noisy images using the character reconstruction network. We use binarized/cleaned version of the original images as labels for training this network. To keep the images binary after resampling, we use the nearest neighbor method during the resizing process. Each pixel in the ground truth can be grouped into one of the following two classes: a) belonging to the background or b) belonging to the character. Fig. 5.6 shows binary images used as ground truth for the



(a) added white gaussian noise (awgn)



(b) decreased contrast and awgn

(c) motion blurred

Figure 5.5: Training Data for the Character Reconstruction Network. The noisy images of the the basic bangla characters

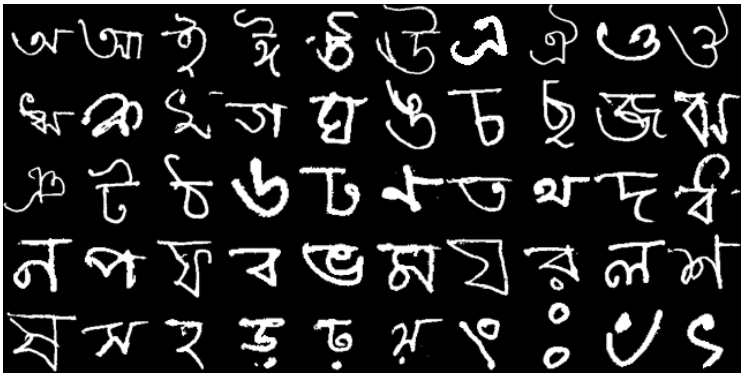


Figure 5.6: Ground truth sample images of the the basic bangla characters for the Character Reconstruction Network

Character Reconstruction Network.

5.2 Deep Belief Networks

We use Deep Belief Networks at two stages of our approach. First, we use a DBN as a Character Reconstruction Network (CRN) to segment the pixels belonging to characters from the background and then we also a DBN as a Character Classification Network (CCN) to generate the final classification using the feature vector representation provided by the probabilistic quadtrees.

5.3 Character Reconstruction Network (CRN)

The Character Reconstruction Network uses the map responses from the intermediate layers of a previously trained CNN as features for pixel-wise classification. The CRN segments the pixels representing the characters from the rest of the pixels. It uses the map responses to the noisy images obtained when they are fed to the pretrained CNN as features. Each pixel, combined with the extracted features, is a single data point and the whole character image is reconstructed based on the classification of each pixel in that image. We do not know beforehand the types of noise that could be in the input images. A single simple filter may not be enough to denoise images with unknown types of noise.

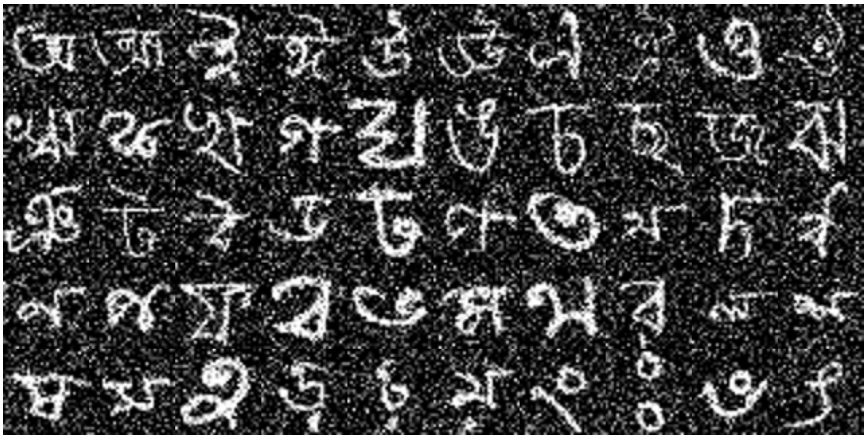
5.3.1 Transfer Learning

We use transfer learning to extract information from the character images using the map responses from the pre-trained CNN as features to the CRN. The data that we feed in to either the CRN or the CCN is not suitable for a convolutional network.

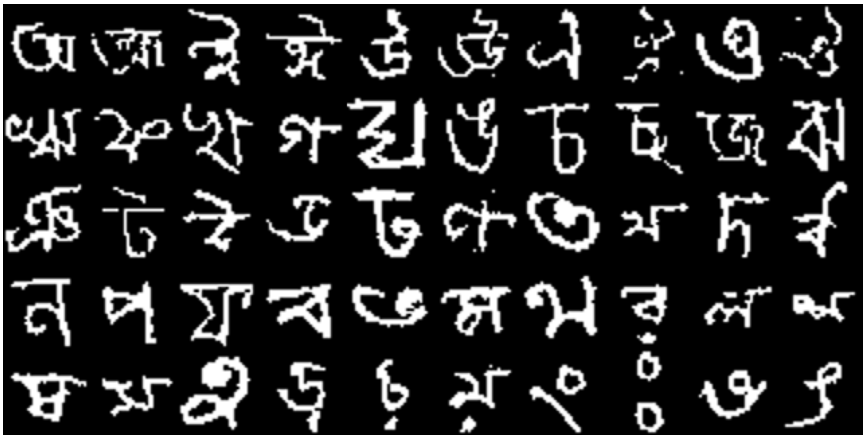
The pretrained CNN has been trained on the ImageNet dataset [56] which has 1000 object classes and more than a million images. It is trained mostly on objects, animals, scenes, and some geometric shapes. Though we are dealing with a different type of data, it has been found that some of the higher level features learned from the convolutional layers are applicable in other types of images as well [99]. Since the objective of the CRN is to use pixel-wise classification to segment the pixels belonging to the character from the rest and the pre-trained CNN already takes into account the contextual information in the image, we use a DBN as the CRN.

5.3.2 Training the CRN

Our input to the Character Reconstruction Network is the preprocessed character images along with the maps extracted from the pre-trained CNN when these images are passed through it. Our ground truth, as explained in Section 5.1.3, is the binarized version of the original images. For the training of this network, all three types of noisy images, that includes both Numeral and Basic Character images, are used. We also train the CRN with images without noise to make it more robust to images containing very little noise as well. We only use images generated from the training set of the original dataset (and not the testing or validation set) for the training, validation, and testing of this network. We sample 30 images each from all the three noisy versions of each of the two types of data: Bangla Numeral and Bangla Basic Character and 14 images each from the images without noise. We use the framework described in [53] to train the CRN where map responses from the pre-trained CNN are all rescaled to our input image size. Now, every pixel in the input image has a corresponding pixel in each of the map responses. The values of the pixels in the corresponding map responses are used as features that when aligned together form



(a) Noisy Samples of images (awgn)



(b) Corresponding output

Figure 5.7: Results from the Character Reconstruction Network

hypercolumns [42]. We train the Character Reconstruction Network by taking random samples from the pool of all available pixels along with their respective features. We do no use the n-MNIST dataset in any training of the CRN. However, we use the same model to reconstruct the n-MNIST images.

5.3.3 Reconstruction on the n-MNIST dataset

As discussed before, we train our CRN using the training images of Noisy Bangla Characters that we created and create a model that reconstructs clean images. We use the same model to reconstruct our n-MNIST images [11]. Standardizing our Noisy Bangla Dataset allowed us to use the same model without having to use the n-MNIST images to

(a) added white gaussian noise (awgn) (b) motion blurred (c) decreased contrast and awgn

Figure 5.8: Sample Images from the n-MNIST Training dataset [11]

train a second model of the CRN. Transfer Learning is also applied here even though the images have a lot of similarity between them. Figure 5.8 shows the noisy images from the n-MNIST dataset. Similar to the Noisy Bangla dataset there are same 3 types of noise that have been inserted. They are of size 28 x 28 compared to 32 x 32 on the Noisy Bangla Dataset.

5.4 Feature Representation using Quadtrees

Decomposing an image window into maximal quadtree blocks has been used as an efficient way to represent sparse features [6]. We use this technique to represent our images. Considering that the use of quadtree decomposition is more effective in representing images with less noise, the previous step is done to denoise noisy images. However, it is also beneficial to reduce dimensionality. Hence, we improve the sparse representation offered by probabilistic quadtrees [11] further by eliminating a) blocks that have been decomposed in only a small fraction of samples and b) blocks that are present in almost all samples. This step would make the dataset itself less redundant.

5.4.1 Saliency Measure

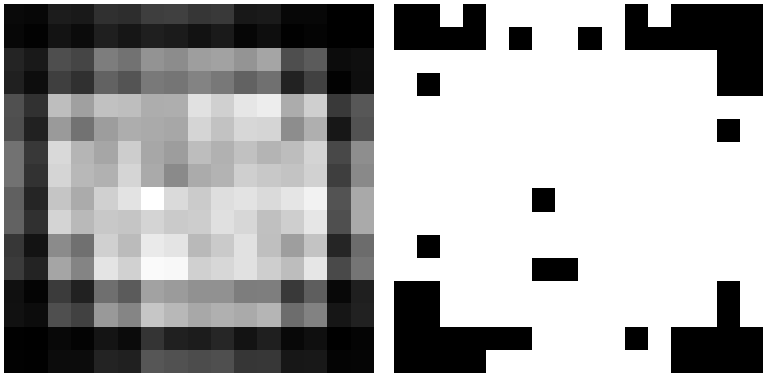
The homogeneity criterion described in [11] guides the process of reducing the character images into a vector of intensity values. Blocks containing textural details would be decomposed into smaller blocks. The quadtree representation is then converted to a linear

vector by performing a depth first search (DFS). The average pixel values of the decomposed blocks are used as features. While this approach reduces the number of features, when the data is noisy the quadtrees tend to be broken down into smaller blocks. Also, using this approach, whenever an image is broken into smaller blocks, all other images use this block in the final feature vector.

To combat this problem, we only use salient blocks that help in discrimination of characters. We define a saliency measure and prune smaller blocks that are not decomposed in μ percentage of the total number of training images or if they are contained in more than ν percentage of the samples. This technique decreases the number of features while not removing key blocks helpful to distinguish characters. The decomposition map on Fig. 5.9 (a) shows the normalized recurrence of the decomposed blocks for the entire training dataset for motion blurred (noisy) Bangla Numeral images. The map shows that a lot of the blocks on the edges are hardly ever used and some blocks in the middle are present in almost all of the images. These blocks are not very likely to be useful in discriminating the characters. The saliency mask obtained in Fig. 5.9 (b) shows the blocks actually used. Table. 5.1 and 5.2 show the reduction of feature vectors using our technique compared to the previous approach.

5.4.2 Character Classification Network

The Character Classification Network is a DBN that uses the average pixel values of the different blocks that have been decomposed in the quadtree. We train the CCN to recognize the handwritten characters as belonging to individual character classes. In this case, the output is a probability value that represents what character is present in the entire image instead of each pixel. The training data consists of the pixel values at all quadtree blocks. The labels are provided by the labels of the images. In contrast to the Character Reconstruction Network, we use separate models for each type of noise for the final classification of the characters into the classes that they represent.



(a) Decomposition Map (b) Saliency Mask

Figure 5.9: Using saliency measure to reduce dimensionality.

Table 5.1: Comparison of the number of features used in the Bangla Numeral Dataset

Noise Type	**Original**	**Ours**	**Basu et. al.[11]**
awgn	1024	**208**	244
contrast	1024	**203**	244
motion	1024	**196**	202

Table 5.2: Comparison of the number of features used in the n-MNIST dataset

Noise Type	**Original**	**Ours**	**Basu et. al.[11]**
awgn	784	**153**	244
contrast	784	**146**	244
motion	784	**145**	211

5.5 Experimental Results and Discussion

5.5.1 Experimental Setup

We trained both the CRN and the CCN by first performing a pre-training step with persistent chain contrastive divergence (P-CD) and then fine tuned the network using backpropagation. We again use L1 and L2 norms for regularization and implemented dropout on the hidden layers. We ran our algorithms on an Intel i7 six core server with TITAN X GPU and used the Theano deep learning library for the Deep Belief Networks [92]. We used $\mu = 5\%$ and $\nu = 95\%$ for our experiments.

5.5.2 Original Numeral Dataset

Table 5.6 shows that our approach performs better on the Original Numeral dataset compared to [15]. For a better comparison, we used a similar multistage approach as in [15] for our classification. We trained a CCN at multiple resolutions of the images: 16 x 16, 32 x 32, 64 x 64 and a combination of these. A classification with rejection scheme was chosen. We had a better recognition accuracy and encountered less rejection (i.e., no class assigned) and substitution (i.e., wrong classification) using our method. We used a uniform threshold for the CCN at each resolution. Fig. 5.12 shows the performance (recognition accuracy) at various thresholds and a probability threshold of .993 yielded best results. Table **??** shows the exact number of correct predictions, rejections, and substitutions on Training and Testing data.

5.5.3 Results on the Bangla Noisy Datasets

On the Bangla Numeral Dataset, Table 5.8 shows that we also significantly improve on the results for each of the noise types in the Noisy Numeral dataset compared to [11]. We further improve the results using the saliency mask. We can observe that the classification accuracy increased even with a decrease in the number of features. In Table 5.10, we compare our results on the Noisy Bangla Characters dataset with that obtained by just using a traditional DBN using raw pixels as features (similar to [11]). We obtained significant

(a) added white gaussian noise (awgn) (b) motion blurred (c) decreased contrast and awgn

Figure 5.10: Reconstructed n-MNIST Images

improvements in overall accuracy for each of the noise types using our approach. We also studied the impact of various CCN architectures on the classification accuracy on the noisy Bangla Basic Characters dataset and the n-MNIST dataset. For the *awgn* noise type of Bangla Basic Characters, an architecture with two hidden layers each with 500 neurons in each layer performed best as seen in Table 5.3. From Table 5.4, we can see that for the *motion* noise type an architecture with 2 hidden layers each with 300 neurons performed best. In the Bangla basic characters, reduced contrast noisy images had the worst accuracy. For the *contrast* noise type, the architectures with 3 hidden layers each with 500 neurons performed best indicated in bold in the three columns of the Table 5.5. Fig. 5.14 shows the percentage of recognition of each class represented using a confusion matrix. Among the digits, 1 and 9 seem to be the most frequently confused characters, as their shapes are similar.

5.5.4 Results on the n-MNIST Dataset

Fig. 5.10 shows samples of the reconstructed images for the n-MNIST dataset. The quadtree representation was extracted from the reconstructed images and trained each using CCN. Each type of noise has its own trained model. Table 5.9 shows the results on the n-MNIST dataset provided in [11]. We get better significant improvements with the datasets with added white gaussian noise (awgn) and with reduced contrast and comparable

Table 5.3: Results on Bangla Characters with AWGN with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	25.29
200 - 200	23.57
300 - 300	23.47
400 - 400	23.35
500 - 500	**23.26**
1000 - 1000	23.28
100 - 100 - 100	26.81
300 - 300 - 300	24.08
500 - 500 - 500	23.28
1000 - 1000 - 1000	23.49

Table 5.4: Results on Motion Blurred Bangla Characters with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	24.43
200 - 200	23.26
300 - 300	**22.78**
400 - 400	23.09
500 - 500	23.35
1000 - 1000	23.03
100 - 100 - 100	26.03
300 - 300 - 300	23.49
500 - 500 - 500	23.24
1000 - 1000 - 1000	23.27

Table 5.5: Results on Contrast Reduced and AWGN MNIST with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	33.63
200 - 200	31.90
300 - 300	31.16
400 - 400	30.71
500 - 500	30.82
1000 - 1000	30.88
100 - 100 - 100	39.90
300 - 300 - 300	32.20
500 - 500 - 500	**30.34**
1000 - 1000 - 1000	31.31

Table 5.6: Comparison on the Bangla Numeral Dataset

	Ours	**Multistage Representation** [15]
Total Correct	**3942**	3928
Accuracy	**98.55%**	98.20%
Rejection	**0.13%**	0.18%
Substitution	**1.32%**	1.62%

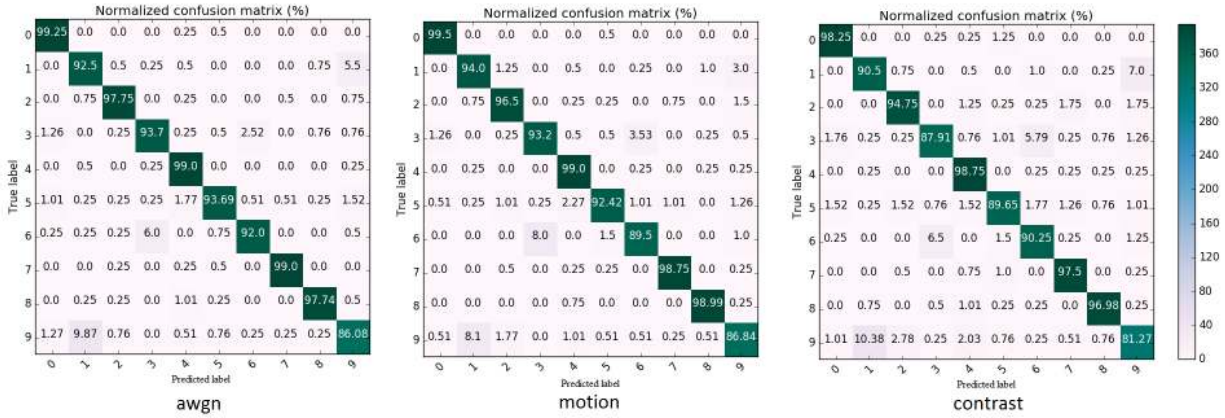


Figure 5.11: Confusion Matrix on the Noisy Bangla Numeral dataset

Table 5.7: Performance Statistics on the Bangla Numeral Dataset

Stage	**Total Samples**	**Recognition Accuracy**		
		Correct	**Rejection**	**Substitution**
Train				
16 X 16	17392	11647	5706	39
32 X 32	5706	4984	717	5
64 X 64	717	661	56	0
Combination	56	55	1	0
Overall		17347	1	44
Overall (%)		99.74	0.01	0.25
Test				
16 X 16	4000	2440	1532	28
32 X 32	1532	1093	417	22
64 X 64	417	349	65	3
Combination	65	60	5	0
Overall		3942	5	53
Overall (%)		98.55	.13	1.32

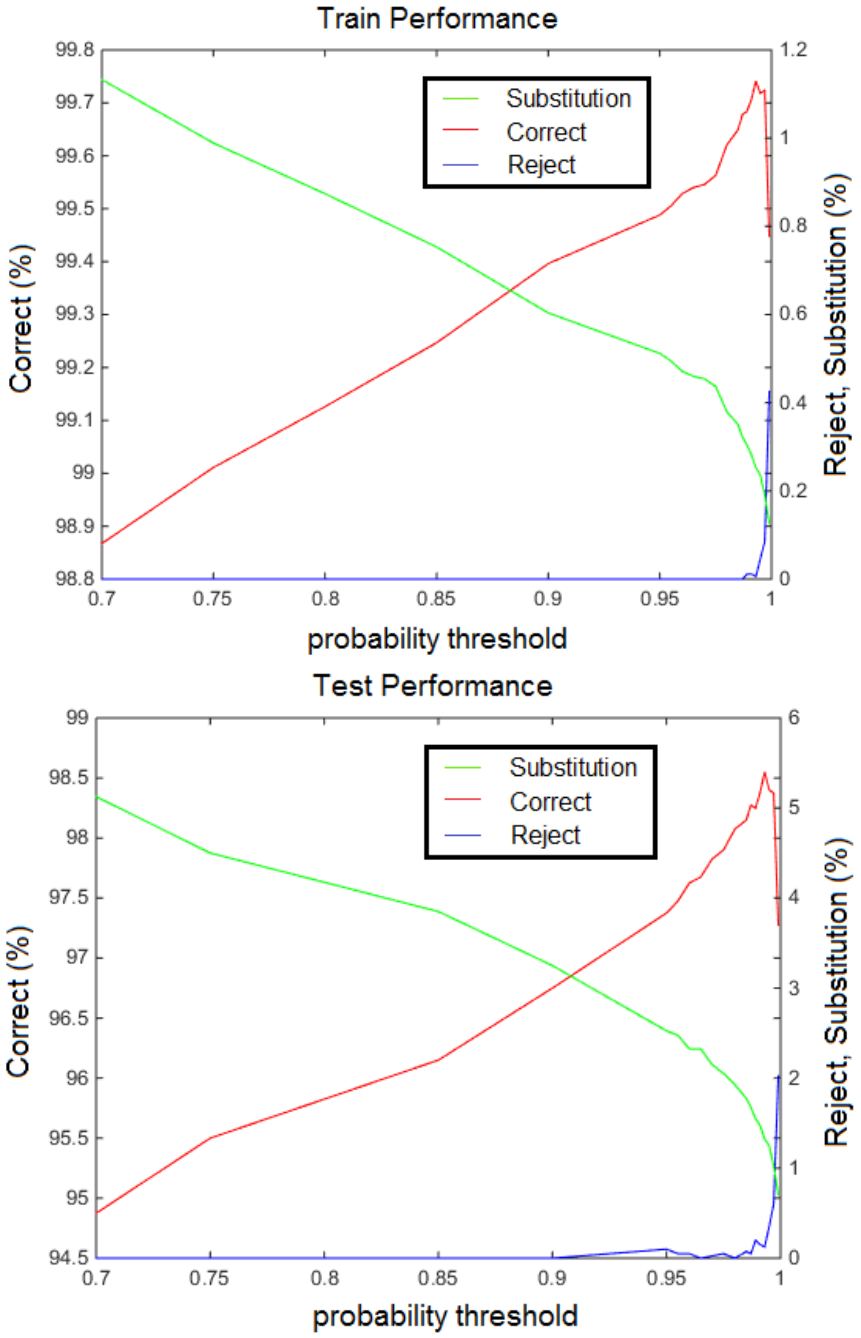


Figure 5.12: Performance on the Bangla Numeral Dataset over various rejection probability threshold. We used the same threshold for Training and Testing

Table 5.8: Comparison of Error(%) on the Noisy Bangla Numeral

Noise	Ours	Basu et. al. [11]	Ours (Saliency Measure)
awgn	4.92	8.66	**4.54**
motion	5.12	7.34	**4.96**
contrast	7.4	12.69	**7.15**

Table 5.9: Comparison of Error(%) on the Noisy MNIST Dataset

Noise	Ours	Basu et. al. [11])
awgn	**2.38**	9.93
motion	2.8	**2.6**
contrast	**4.96**	7.84

results with the motion blurred image. The observation made from this result is that the accuracy is improved on images with added white noise. We also use smaller number of features for the n-MNIST compared to using raw pixels and [11]. For the *awgn* noise type MNIST characters, an architecture with two hidden layers each with 500 neurons performed best as seen in Table 5.11. From Table 5.12, we can see that for the *motion* noise type an architecture with 2 hidden layers with 400 neurons in each layer performed best. The reduced contrast noisy MNIST images had the worst accuracy. For the *contrast* noise type, the architectures with 2 hidden layers with 300 neurons in each layer performed best indicated in bold in the three columns of the Table 5.5.

In both n-MNIST and noisy Bangla Basic Characters, reduced contrast images had the worst recognition rates. The reconstruction network eroded a lot of pixels from the characters of the reduced contrast images. The random noise pixels had similar intensity values to the character pixels. Because of that edges were not as sharp as the other two types

Table 5.10: Performance on Noisy Bangla Characters

Noise Type	Error (%)		
	Ours	Ours (Without Reconstruction)	Raw Pixels
awgn	**23.26**	29.36	42.69
motion	**22.78**	25.64	41.20
contrast	**30.34**	41.11	53.37

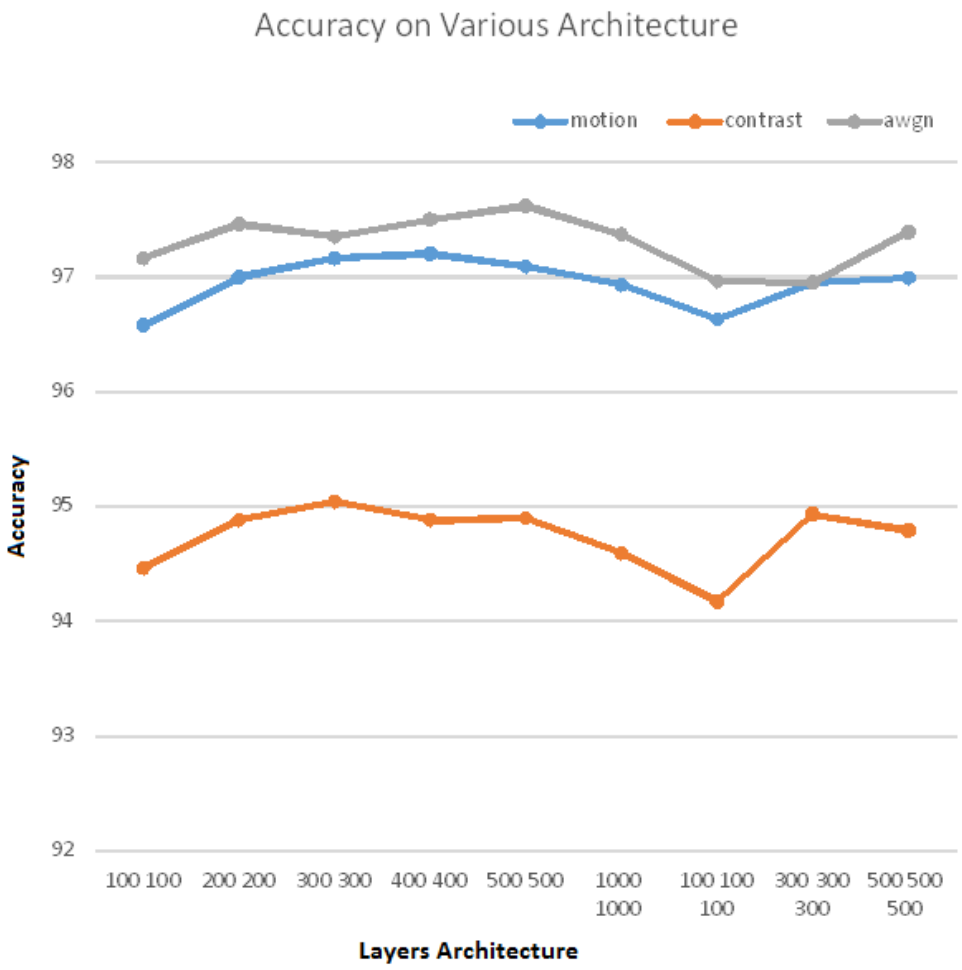


Figure 5.13: Accuracy on the n-MNIST with different architecture

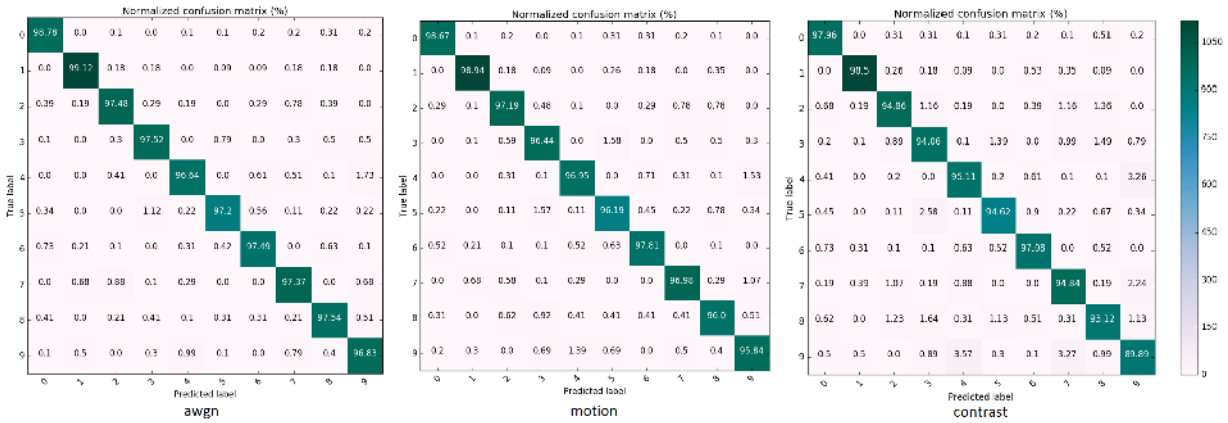


Figure 5.14: Confusion Matrix on the n-MNIST dataset

Table 5.11: Results on Added White Gaussian Noise MNIST with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	2.84
200 - 200	2.54
300 - 300	2.65
400 - 400	2.5
500 - 500	**2.38**
1000 - 1000	2.63
100 - 100 - 100	3.04
300 - 300 - 300	3.05
500 - 500 - 500	2.61

of noisy images. Fig. 5.11 shows the percentage of recognition of each class represented using a confusion matrix. It is notable that the erosion of pixels from the contrast dataset leads to a recognition of 89

Table 5.12: Results on Motion Blurred MNIST with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	3.42
200 - 200	3.00
300 - 300	2.84
400 - 400	**2.80**
500 - 500	2.91
1000 - 1000	3.07
100 - 100 - 100	3.37
300 - 300 - 300	3.05
500 - 500 - 500	3.01

Table 5.13: Results on Contrast Reduced and AWGN MNIST with Different Architecture

Architecture (Neurons)	Error (%)
100 - 100	5.54
200 - 200	5.12
300 - 300	**4.96**
400 - 400	5.12
500 - 500	5.10
1000 - 1000	5.41
100 - 100 - 100	5.83
300 - 300 - 300	5.07
500 - 500 - 500	5.21

Chapter 6
Conclusions

Our rich symbolic framework based on regular expressions is used for recognizing diverse types of activities in surveillance videos. Though simple, the framework not only provides fast algorithms for activity recognition but is also flexible enough to admit both generative and learning based approaches. We plan to integrate our framework with reasoning engines to provide inference capabilities at a higher level of abstraction. We also plan to compare to incorporate the Probabilistic Finite State Automata (PFA) and make comparisons with the current approach as PFAs can be learnt from a set of strings using Expectation-Maximization.

We also presented a core sampling framework that is able to use the activation maps from several layers of a pretrained CNN as features to a DBN using transfer learning to aid pixel level classification. We experimentally demonstrate the usefulness of our framework using a pretrained VGG-16 [90] model to perform segmentation on the BAERI dataset [38] of Synthetic Aperture Radar(SAR) imagery and the CAMVID dataset [23]. In the future, we intend to use the core sampling framework to facilitate compression of images and texture synthesis.

We improved the efficiency of probabilistic quadtrees by using a pixel level classifier to reconstruct character images by removing the noisy pixels from them. We used a pixel level denoiser based on a pretrained CNN on a distinct but huge dataset and used transfer learning to learn features for that network.

References

[1] Classifying mnist digits using logistic regression. *deeplearning.net*. URL: `http://deeplearning.net/tutorial/logreg.html`.

[2] Otsus binarization. URL: `http://docs.opencv.org/3.1.0/d7/d4d/tutorial_py_thresholding.html`.

[3] Results on baeri images. URL: `https://drive.google.com/open?id=0B0gFcrqVCm9pUy01bWpheGs3RlU`.

[4] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[5] J.K. Aggarwal and M.S. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3):16:1–16:43, apr 2011. `doi:10.1145/1922649.1922653`.

[6] Walid G Aref and Hanan Samet. Decomposing a window into maximal quadtree blocks. *Acta Informatica*, 30(5):425–439, 1993.

[7] Alexander Artikis, Marek Sergot, and Georgios Paliouras. A logic programming approach to activity recognition. In *Proceedings of the 2nd ACM international workshop on Events in multimedia*, pages 3–8. ACM, 2010.

[8] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[9] Jonathan T Barron, Mark D Biggin, Pablo Arbelaez, David W Knowles, Soile VE Keranen, and Jitendra Malik. Volumetric semantic segmentation using pyramid context features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3448–3455, 2013.

[10] Saikat Basu, Robert Dibiano, Manohar Karki, Malcolm Stagg, Jerry Weltman, Supratik Mukhopadhyay, and Sangram Ganguly. An agile framework for real-time motion tracking. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 3, pages 205–210. IEEE, 2015.

[11] Saikat Basu, Manohar Karki, Sangram Ganguly, Robert DiBiano, Supratik Mukhopadhyay, Shreekant Gayaka, Rajgopal Kannan, and Ramakrishna Nemani. Learning sparse feature representations using probabilistic quadtrees and deep belief nets. *Neural Processing Letters*, pages 1–13.

[12] Saikat Basu, Manohar Karki, Malcolm Stagg, Robert DiBiano, Sangram Ganguly, and Supratik Mukhopadhyay. Maptrack-a probabilistic real time tracking framework by integrating motion, appearance and position models. In *VISAPP (3)*, pages 567–574, 2015.

[13] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[14] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[15] Ujjwal Bhattacharya and Bidyut Baran Chaudhuri. Handwritten numeral databases of indian scripts and multistage recognition of mixed numerals. *IEEE transactions on pattern analysis and machine intelligence*, 31(3):444–457, 2009.

[16] Ujjwal Bhattacharya, Malayappan Shridhar, Swapan K Parui, PK Sen, and BB Chaudhuri. Offline recognition of handwritten bangla characters: an efficient two-stage approach. *Pattern Analysis and Applications*, 15(4):445–458, 2012.

[17] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. libalf: The automata learning framework. In *In Proceedings of CAV*, 2010.

[18] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[19] Y-lan Boureau, Yann L Cun, et al. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*, pages 1185–1192, 2008.

[20] Yuri Y Boykov and M-P Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.

[21] Ron Brinkmann. *The art and science of digital compositing: techniques for visual effects, animation and motion graphics*. Morgan Kaufmann, 2008.

[22] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, xx(x):xx–xx, 2008.

[23] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV (1)*, pages 44–57, 2008.

[24] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60–65. IEEE, 2005.

[25] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[26] Chia-Chih Chen and JK Aggarwal. Modeling human activities as speech. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3425–3432. IEEE, 2011.

[27] Lei Chen, M Tamer Özsu, and Vincent Oria. Symbolic representation and retrieval of moving object trajectories. In *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 227–234. ACM, 2004.

[28] Ming-yu Chen and Alexander Hauptmann. Mosift: Recognizing human actions in surveillance videos. 2009.

[29] Sizhe Chen and Haipeng Wang. Sar target recognition based on deep learning. In *Data Science and Advanced Analytics (DSAA), 2014 International Conference on*, pages 541–547. IEEE, 2014.

[30] Wongun Choi, K. Shahid, and S. Savarese. Learning context for collective activity recognition. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3273 –3280, june 2011. doi:10.1109/CVPR.2011.5995707.

[31] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: color image segmentation. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 750–755. IEEE, 1997.

[32] Judith E Dayhoff and James M DeLeo. Artificial neural networks. *Cancer*, 91(S8):1615–1635, 2001.

[33] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5):352–359, 2002.

[34] Bo Du, Liangpei Zhang, Dacheng Tao, and Dengyi Zhang. Unsupervised transfer learning for target detection from hyperspectral images. *Neurocomputing*, 120:72–82, 2013.

[35] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. springer, 2005.

[36] Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[37] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[38] Sangram Ganguly. Baeri dataset. *Personal Communication*. URL: https://drive.google.com/open?id=0B0gFcrqVCm9peTdMdndTV0pQMFE.

[39] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[40] Debanjan Das Gupta. Mesmerizing kaziranga. *Thirsty Eyes*. URL: http://www.debanjanphotography.com/into-the-wild.html.

[41] Robert M Haralick and Linda G Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.

[42] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

[43] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[45] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[46] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[47] A. Hoogs and A. G. A. Perera. Video activity recognition in the real world. In *Proceedings of AAAI*, 2008.

[48] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[49] Ke Huang and Selin Aviyente. Sparse representation for signal classification. In *NIPS*, volume 19, pages 609–616, 2006.

[50] Nazli Ikizler, Ramazan Gokberk Cinbis, and Pinar Duygulu. Human action recognition with line and flow histograms. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[51] Yu-Gang Jiang, Zhenguo Li, and Shih-Fu Chang. Modeling scene and object contexts for human action retrieval with few examples. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(5):674 –681, may 2011. `doi:10.1109/TCSVT.2011.2129870`.

[52] George H Joblove and Donald Greenberg. Color spaces for computer graphics. In *ACM siggraph computer graphics*, volume 12, pages 20–25. ACM, 1978.

[53] M. Karki, R. DiBiano, S. Basu, and S. Mukhopadhyay. Core Sampling Framework for Pixel Classification. *arXiv pre-prints*, December 2016. `arXiv:1612.01981`.

[54] Eunju Kim, Sumi Helal, and Diane Cook. Human activity recognition and pattern discovery. *IEEE Pervasive Computing*, 9(1), 2010.

[55] VM Kotlyakov. A 150,000-year climatic record from antarctic ice. *Nature*, 316:591596, 1985.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[57] L'ubor Ladickỳ, Paul Sturgess, Karteek Alahari, Chris Russell, and Philip HS Torr. What, where and how many? combining object detectors and crfs. In *European conference on computer vision*, pages 424–437. Springer, 2010.

[58] John Lafferty, Andrew McCallum, Fernando Pereira, et al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

[59] Ivan Laptev, Marcin Marszalek, Cordelia Schmid, and Benjamin Rozenfeld. Learning realistic human actions from movies. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/CVPR.2008.4587756.

[60] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[61] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[62] Yann LeCun et al. Lenet-5, convolutional neural networks.

[63] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, E Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

[64] Binlong Li, M. Ayazoglu, T. Mao, O.I. Camps, and M. Sznaier. Activity recognition using dynamic subspace angles. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3193 –3200, june 2011. doi:10.1109/CVPR.2011.5995672.

[65] Weiyao Lin, Ming-Ting Sun, R. Poovandran, and Zhengyou Zhang. Human activity recognition for video surveillance. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 2737 –2740, may 2008. doi:10.1109/ISCAS.2008.4542023.

[66] Richard Lippmann. An introduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

[67] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[68] NO Lotter, DL Kowal, MA Tuzun, PJ Whittaker, and L Kormos. Sampling and flotation testing of sudbury basin drill core for process mineralogy modelling. *Minerals Engineering*, 16(9):857–864, 2003.

[69] Tassos Markas and John Reif. Quad tree structures for image compression applications. *Information Processing & Management*, 28(6):707–721, 1992.

[70] R. Messing, C. Pal, and H. Kautz. Activity recognition using the velocity histories of tracked keypoints. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 104 –111, 29 2009-oct. 2 2009. doi:10.1109/ICCV.2009.5459154.

[71] P. Muzatko. Approximate regular expression matching. Technical report, Czech Technical University, August 1996.

[72] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[73] Sangmin Oh, Anthony Hoogs, A. G. Amitha Perera, Naresh P. Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xioyang Wang, Qiang Ji, Kishore K. Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit K. Roy Chowdhury, and Mita Desai. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR*, pages 3153–3160, 2011.

[74] Nobuyuki Otsu. A threshold selection method from gray-level histograms. volume 11, pages 23–27, 1975.

[75] Our. An agile framework for real-time motion tracking. Provided as supplementary material.

[76] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[78] Lawrence Rabiner and B Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[79] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[80] E. Resendiz and N. Ahuja. A unified model for activity recognition from video sequences. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1 –4, dec. 2008. doi:10.1109/ICPR.2008.4761701.

[81] Martin Riedmiller. Machine learning: Multi layer perceptronsmachine learning: Multi layer perceptrons. URL: http://ml.informatik.uni-freiburg.de/_media/teaching/ss10/05_mlps.printer.pdf.

[82] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (TOG)*, volume 23, pages 309–314. ACM, 2004.

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

[84] M. S. Ryoo and J. K. Aggarwal. Recognition of composite human activities through context-free grammar based representation. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:1709–1718, 2006. doi:http://doi.ieeecomputersociety.org/10.1109/CVPR.2006.242.

[85] Fariba Sadri. Logic-based approaches to intention recognition. In *Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives*, pages 346–375. IGI Global, 2011.

[86] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *ICPR (3)*, pages 32–36, 2004.

[87] Mubarak Shah, Omar Javed, and Khurram Shafique. Automated visual surveillance in realistic scenarios. *IEEE MultiMedia*, 14:30–39, January 2007. URL: http://dl.acm.org/citation.cfm?id=1262177.1262230, doi:10.1109/MMUL.2007.3.

[88] V.D. Shet, D. Harwood, and L.S. Davis. Vidmap: video monitoring of activity with prolog. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 224 – 229, sept. 2005. doi:10.1109/AVSS.2005.1577271.

[89] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[90] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[91] P. Tesson and D. Therien. Logic meets algebra: The case of regular languages. In *In Proceedings of LICS*, 2005.

[92] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL: http://arxiv.org/abs/1605.02688.

[93] Lisa Torrey and Jude Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242, 2009.

[94] Son Dinh Tran and Larry S. Davis. Event modeling and recognition using markov logic networks. In *ECCV (2)*, pages 610–623, 2008.

[95] P. Turaga, R. Chellappa, V.S. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1473 –1488, nov. 2008. doi:10.1109/TCSVT.2008.2005594.

[96] Harini Veeraraghavan, Paul Schrater, and Nikolaos Papanikolopoulos. Switching kalman filter-based approach for tracking and event detection at traffic intersections. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 1167–1172. IEEE, 2005.

[97] Heng Wang, Muhammad Muneeb Ullah, Alexander Klaser, Ivan Laptev, and Cordelia Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC 2009-British Machine Vision Conference*, pages 124–1. BMVA Press, 2009.

[98] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[99] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[100] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[101] Zhi Zeng and Qiang Ji. Knowledge based activity recognition with dynamic bayesian network. In *ECCV (6)*, pages 532–546, 2010.

[102] Chenxi Zhang, Liang Wang, and Ruigang Yang. Semantic segmentation of urban scenes using dense depth maps. In *European Conference on Computer Vision*, pages 708–721. Springer, 2010.

[103] Yiren Zhou, Sibo Song, and Ngai-Man Cheung. On classification of distorted images with deep convolutional neural networks. *arXiv preprint arXiv:1701.01924*, 2017.

Appendix A
Title of Appendix A

Chapter 3 of this work is based on the paper titled "A Symbolic Framework for Recognizing Activities in Full Motion Surveillance Videos", published in the IEEE Symposium Series on Computational Intelligence, IEEE SSCI 2016.

Chapter 4 of this work is based on the paper titled "Core Sampling Framework for Pixel Classification", to be published in International Conference on Artificial Neural Networks, ICANN 2017.

Vita

Manohar Karki received Bachelor of Science degree in Software Engineering from University of Texas at Arlington in May 2010. In the Fall of 2011, being highly keen to work in the areas of Machine Learning and Artificial Intelligence, he decided to join Computer Science division for the doctorate program at Louisiana State University. He had an opportunity to be involved in many projects under the guidance of his advisor Dr. Supratik Mukhopadhyay as a Research Assistant in the Robitics Lab, advancing his broader knowledge of the fields. These included analysis from satellite imagery, detection and tracking of objects from videos, chained events in videos etc. which are aside from the work discussed in this dissertation.