NEURAL NETWORK FEATURE EXTRACTION FOR ACTIVITY RECOGNITION IN VIDEO DATA

KYLE ALLAN CASTERLINE
SPRING 2014

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Shashi Phoha
Professor of Electrical Engineering
Thesis Supervisor

John Mitchell
Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.
Abstract

The goal of this research is to determine how effectively various objects within a video stream can be classified using autoencoding neural network methods which extract spatio-temporal features from data. More specifically, we wish to validate a method of feature extraction using a Sparse Autoencoding Neural Network (SAENN). This feature extraction method, which has shown success in static machine vision problems, has to the best of our knowledge never before been applied to video. The performance of autoencoder–derived features is compared with Principal Component Analysis (PCA) features as a relevant baseline. The performance evaluation of the SAENN model is based on two separate factors: data reconstruction, and object classification performance. Reconstruction performance is calculated from mean squared error values, and classification accuracy values are obtained from experiments using supervised learning algorithms (support vector machines and k-nearest neighbors). The results of this research indicate that features extracted through the SAENN model outperform those extracted through PCA in both areas of comparison.
# Table of Contents

Abstract .......................................................................................................................... i

Table of Contents ........................................................................................................ ii

List of Figures ................................................................................................................ iii

List of Tables ................................................................................................................ iv

Acknowledgments ......................................................................................................... v

Chapter 1 Introduction ................................................................................................ 1

Chapter 2 Activity Analysis & Machine Learning .................................................... 6
  Related Work ............................................................................................................... 6
  Supervised Machine Learning ............................................................................... 10
  VIRAT Dataset ......................................................................................................... 13

Chapter 3 Neural Networks and Autoencoders ....................................................... 15
  Artificial Neurons and the Sigmoid Function ....................................................... 15
  Neural Networks .................................................................................................... 17
  Sparse Autoencoders .......................................................................................... 18

Chapter 4 Extracting Feature Vectors ..................................................................... 22
  Video Processing .................................................................................................... 22
    Gaussian Mixture Model Background Subtraction ............................................. 23
    Establishing Regions of Interest ................................................................... 26
    Creating Training Sets .................................................................................. 27
  Tuning Sparse Autoencoder Parameters ............................................................. 27
  Reconstructing/Visualizing Data ....................................................................... 29
  Pooling .................................................................................................................. 30
  Comparison to PCA method ........................................................................... 33
    Feature Extraction with PCA ....................................................................... 35

Chapter 5 Results ........................................................................................................ 36
  Cross Validation .................................................................................................... 36
  Reconstruction MSE ......................................................................................... 38
  Classification Accuracy .................................................................................... 39

Chapter 6 Conclusions ............................................................................................... 42
  Future Work .......................................................................................................... 43

Bibliography ............................................................................................................... 45
List of Figures

Figure 1.1 Data Flow Diagram ................................................................. 5
Figure 2.1 Sample Frames from the VIRAT Dataset .................................. 14
Figure 3.1 Model of an artificial neuron .................................................. 15
Figure 3.2 Plot of Sigmoid Curve ............................................................. 16
Figure 3.3 Autoencoder Neural Network Structure .................................... 19
Figure 4.1 Grayscale Image with Applied Mask ....................................... 25
Figure 4.2 Left: Original Mask, Right: Modified Mask ............................... 25
Figure 4.3 Left: Grayscale Image, Right: Grayscale with Equalized Histogram .... 25
Figure 4.4 Plot of MSE of data reconstructions vs. sparsity ....................... 28
Figure 4.5 Visual Example of Data Reconstructions .................................. 29
Figure 4.6 Visual representation of feature pooling process ........................ 32
Figure 4.7 Representation of Feature Pooling Process ............................... 33
List of Tables

Table 1 Training Parameters for Determining Optimal Sparsity ............................................ 28
Table 2 Mean Squared Error of Reconstructions ....................................................................... 39
Table 3 Description of the Training and Testing Dataset ........................................................... 40
Table 4 SVM Confusion Matrix .................................................................................................. 40
Table 5 K-NN Confusion Matrix ............................................................................................... 41
Table 6 K-Fold Cross Validation Variance .................................................................................. 41
Acknowledgments

This thesis was funded by Dr. Shashi Phoha through the Information Science and Technology Division Distinguished Undergraduate Research Program of the The Applied Research Laboratory at The Pennsylvania State University. I would like to express my utmost gratitude to Dr. Phoha for her support and constructive suggestions crucial for the completion of this thesis. I am particularly grateful for the assistance given by Dr. Brian Smith in my research and throughout the writing process. Thank you both very much for your encouragement and understanding over this past year. Additionally, I would like to acknowledge my honors advisor, Dr. John Mitchell for the valuable suggestions and guidance he has provided. I would also like to thank everyone at the Applied Research Laboratory who have contributed important input for my research and allowed me to broaden my knowledge and technical skills. Finally I would like to thank all of the support I have received from my family and friends during this process.
Chapter 1

Introduction

Information overload is a modern issue in which many corporations and government entities are faced with having to deal with rapidly growing quantities of data. It has been calculated that by the year 2020, the size of the digital universe will have increased over 50 times in volume since 2010 [1]. This will result in an overall digital universe size of roughly 40 zettabytes (40 \times 10^{21} \text{ bytes}). In their Digital Universe study, the International Data Corporation (IDC) approximated that 23\% of all properly tagged and analyzed data will prove to be useful in big data analytics [1]. Furthermore, the largest portion of this percentage is video surveillance data. As a result, the improvement of video data analysis methods is becoming increasingly important. To prevent entities from becoming overloaded with data, a number of methods must be incorporated in order to interpret this data autonomously by machines without human supervision. In this work, we use a Sparse Autoencoding Neural Network (SAENN) as a means of extracting features in raw video data. We hypothesize that these networks will extract desirable features, which contain more relevant information for object classification than other available methods in the literature.

Therefore, in this work, we address the problem of extracting spatio-temporal features from objects appearing in video stream data for classification purposes. The anticipated product of this research project is to provide a scientific validation of the effectiveness of using Neural Network models as a feature extraction method to autonomously classify objects that appear within video stream data. Additional
background on the topic of activity analysis, and machine learning in general, is presented in Chapter 2.

The Sparse Autoencoding Neural Network (SAENN) is one of a class of more general models known as artificial neural networks. These neural networks, when used in a manner known as autoencoding, are known for their ability to find structure within an unlabeled dataset [2]. The term “autoencoder” is derived from two parts: “auto” and “encode”; giving the term an overall meaning of creating a compressed self-description for any given data point. The model that we apply to video data, and evaluate the performance of, incorporates the concept of Sparsity into a neural network [3]. That is, we desire to represent any particular input data point using only a small number of the possible basis vectors in a basis set, which in prior work has led to an increase in recognition capability. Additional detail on the SAENN model is provided in Chapter 3.

The main thrust of this research is to test the neural network based method of feature extraction and compare it to Principal Component Analysis (PCA). We desire to test the sparse autoencoding neural network in its ability to extract spatio-temporal features, an application never before investigated to our knowledge. The SAENN-derived features are then compared to those of PCA, a widely used method of data dimensionality reduction, as a means of validating performance. In short, PCA treats points in a high-dimensional dataset as being instances of a random vector, where the different dimensional components of this vector may be highly correlated. It will then determine an orthogonal transformation of the data, removing the correlation between dimensional components. Finally, the data is projected onto a lower dimensional subspace, removing
only the dimensions in which the variance of the data is low [4]. More information on the theory and implementation of PCA is provided in Chapter 4.

The scope of research encompasses two primary objectives:

- First, it will validate that spatio-temporal data reconstructions generated by SAENN are of better quality than those generated using PCA. Mean Squared Error (MSE) between the data reconstructions and input data are used to numerically evaluate the overall quality of the reconstructions.

- Secondly, it will validate that spatio-temporal features extracted by a SAENN provide better classification accuracy than those extracted with PCA. The spatio-temporal features extracted through each method are used to attempt to classify objects in video stream data using two different common supervised learning algorithms. The algorithms are trained using feature vector sets from each method of extraction. The algorithm’s classification accuracy, when trained on each particular set of features, is used to rate each set. K-Nearest Neighbors and Support Vector Machines are used as learning algorithms in this experiment. More details on each are provided in Chapter 2.

The primary steps performed in this work for accomplishing the aforementioned objectives are *data pre-processing*, *feature extraction* and *classification*. The details of implementing each are given in Chapter 4, illustrated in Figure 1.1, and summarized here:

- The raw video footage is first processed to remove any spatial information not associated with the objects of interest. For the dataset being used in this project, the objects of interest will typically be people and vehicles. The video’s background is
removed to highlight anything that moves throughout the course of the video. The second operation involves splitting each video into smaller clips in order to isolate the objects of interest. The dataset being used for this project includes pre-written annotation files that are designed to assist in performing this step.

- Small samples of data, patches that are $x$ by $y$ by $t$ pixels in height, width, and frame depth, are selected from the video clips and used to create training sets for a sparse autoencoding neural network. Using gradient descent to minimize a cost function consisting of terms which both minimize error and extract structure, we train the neural network on large sets of data patches.

- The trained values of the network’s weighted connections are used to generate a set of feature vectors that correspond with each object in the dataset. This involves incorporating a pooling algorithm to generate a compressed representation of the data that best corresponds to all of the features within a single object clip.

- Lastly, the feature vectors are passed to a classification algorithm. In this project, both a Support Vector Machine and K-Nearest Neighbors model will be used for classification. The performance results of classification will be compared to those using feature vectors extracted with PCA. These results are presented in Chapter 5.
Identification of Objects
Video
Pre-Processing
Create Training Sets
Train: NN-SAE Reconstructions
Trained Network Weight and Bias Terms
Pooling
Feature Vectors
Train: K-Nearest Neighbors
Classify: K-Nearest Neighbors
Cross Validation
Classification Accuracy
Train: Support Vector Machine
Classify: Support Vector Machine
Processed Video Data
Annotation Files
Video Data
Training Parameters

Figure 1.1 Data Flow Diagram
Chapter 2

Activity Analysis & Machine Learning

Related Work

Much work in machine vision dedicated to feature extraction has been based around extracting features from still images rather than from video data. Described below are several methods for extracting features that are similar to the approach described in this paper. The methods briefly described in this section are SIFT, and Zernike moments.

SIFT

The Scale Invariant Feature Transform (SIFT) is a method of extracting features from image data developed by David Lowe in 1999. The property of SIFT that makes it different from other methods of feature extraction is that it generates large numbers of features that span over the full range of scales and locations [5]. The SIFT algorithm is based on generating a set of scale-invariant keypoints relative to local features within an image. The SIFT algorithm involves 4 separate stages briefly described below [5]:

- Scale-space extrema detection: This is part of the initial preparation of the image. A scale space is created by progressively applying a Gaussian blur to the original image, rescaling the image to half its original size and progressively applying a Gaussian blur to that image. Each set of progressively blurred images is called an
octave and the degree to which each image is blurred is called scale. These values vary depending on the overall size and resolution of the original image.

- **Keypoint Localization**: Sets of Difference of Gaussian (DoG) images are created for each octave by taking the difference between each consecutive image within the octave. The location of local extrema within the sets of DoG images are used as keypoints. This is done by comparing the magnitude of a pixel with that of each of its neighboring pixels in the x, y, and scale dimensions. Certain keypoints are then filtered out based on measurements of their stability.

- **Orientation Assignment**: This process introduces a factor of rotation invariance to the selected keypoints. An orientation collection region is established around each keypoint with a size that is dependent upon what scale the keypoint is located on. The magnitude of the gradient is calculated from the orientation collection region over a range of angular orientations about the keypoint. The orientations with magnitudes that fall above a certain threshold are assigned to that keypoint. A keypoint is split into multiple points at the same location if more than one orientation falls above the threshold on any specific point.

- **Feature Generation**: Feature vectors are constructed using weighted gradients from a grid centered about a given keypoint. The sums of these gradients are categorized based on their magnitude and orientation.

The SIFT algorithm has been used in multiple applications including object recognition, image stitching, robotic mapping, 3D modeling, gesture recognition, and video tracking [5].
Zernike Moments

Zernike moments (ZMTS) are a sequence of orthogonal polynomials that are features of a 2-D dataset. A method of using of using ZMTS for extracting spatio-temporal features from time series video data was proposed by Shane Marcks in 2012 [6]. In his paper Marcks also VIRAT, the same dataset that we use for validation, for validating the Zernike Moment features (see VIRAT Dataset). The primary advantages this method are that ZMTS are scale invariant, translation invariant, and in-plane rotation invariant [6]. These are clearly properties that can account for variety often found in video data (i.e. scale invariance to account for an object’s distance from the camera).

Since Zernike moments are essentially a description of a 2-D dataset, some additional work must be done in order to use them in extracting features from a spatio-temporal dataset. First, a time-series of Zernike moments is computed for a series of frames in the video. After that process is complete, a probabilistic description of that time-series, specifically the stationary probability distribution (SPD), must be calculated. This SPD, calculated for a region of interest, is used as the feature vector for the machine learning and classification algorithm. A brief description of each of these two steps follows.

The first step is to compute a ZMTS time series for a video stream. This is done by breaking the video stream into frames and computing $l$ discrete Zernike moments for each frame. The value of $l$ is a parameter used to control the number of Zernike moments to calculate thus controlling the level of accuracy to which each image can be reconstructed. To calculate these ZMTS each frame must first be converted to grayscale.
A region of interest (ROI) is then established around the object/event from which the features will be extracted from. The next step is to create a binary mask using only the pixels of interest (see Video Processing). This mask is then placed at the center of a blank template large enough to encompass the pixels of interest within the binary mask. Once each image has been converted to this form it can be used to calculate the l ZMTS. These steps are then repeated for each frame in the time series.

The second step is creating a stationary probability distribution (SPD) vector. This is done through the process of Symbolic Dynamic Filtering (SDF) [7]. This process involves three separate steps. The ZMTS time series must first be converted into a sequence of symbols. Max entropy partitioning (MEP) is used to divide the ZMTS time series into a set of $k$ symbols where the size of each cell is determined by calculating the Shannon entropy of the symbol sequence. An optimal alphabet size ($k_{opt}$) is determined by minimizing a linear cost function to balance information content and computational complexity. A Probabilistic Finite State Automata (PFSA) is then generated from the symbol sequence. This is done by representing different sequences of symbols as the automaton’s states. The edges of the PFSA represent the state transition probabilities between the different states. Lastly, the PFSA state transition probabilities are used to create a state-transition probability matrix. The SPD feature vector can then be extracted from this matrix by selecting the eigenvector that corresponds to the unity eigenvalue.

Marcks used both a K-nearest neighbors algorithm and a support vector machine (see Supervised Machine Learning) to classify the extracted feature vectors. Data classes used include human, vehicle, human standing, human walking, vehicle still, and vehicle
moving. The biggest disadvantage of this method is that it only uses data from the object’s outline to generate the SPD feature vector. In other words all visual features contained within the objects outline are eliminated when the ZMTS time series is created.

**Supervised Machine Learning**

Supervised learning is a process in which a learning algorithm derives a function based off a set of labeled training samples. Each training sample is presented to the algorithm as a pair containing input data and a desired output. The derived function can then be applied to additional input data to generate a predicted output. In an ideal case the algorithm will produce correct outputs for each data sample that is presented. Through this process of deriving a function the learning algorithm is forced to find structure within the data and use it to make generalizations. Two instances of supervised machine learning used in this paper for the purposes of classification are Support Vector Machines (SVM) and the K-Nearest Neighbors (KNN) algorithm. A brief description of how each method is implemented is provided in the following two sections.

**Support Vector Machines**

A Support Vector Machine (SVM) is a type of supervised learning model. For the purposes of this research, a SVM model is trained to classify feature vectors generated by a SAENN as a means of evaluating their accuracy. In essence, the SVM model works by mapping each feature vector in an n-dimensional space [8]. It then attempts to create a set
of optimal boundaries to separate each class of data. The procedure of training an SVM to linearly classify a simple two-class dataset is described below [8].

While being trained, the SVM is presented with a set of training samples and their associated class labels. Equation 1 shows a simple two class dataset where each data sample is symbolized as $x_i$ and each sample class label is symbolized as $y_i$. The sample set $X$ exists in $n$ dimensions and the set of data class labels $Y$ assume a value of either +1 or -1 to depending upon which class the sample belongs to.

$$\text{Data} = \{(x_i,y_i) | x_i \in \{\mathbb{R}^n\}, y_i \in \{-1,1\}\}_{i=1}^{n}$$  \hspace{1cm} (1)

A SVM works by representing each data sample as a point in space. The machine then attempts to generate the best possible decision boundary to spatially separate each class of data while maintaining widest margin possible between the boundary and the nearest samples in each class [8]. The decision boundary is a hyperplane represented as $W^T x + b = 0$ where $W$ and $b$ are the vectors that define the hyperplane’s spatial position. The decision function is defined as $f(x) = sign(W^T x + b)$ so that all values greater than or equal to the decision boundary will have a label of +1 and all values less than the decision boundary will have a label of -1. The class label values are chosen as a way of shortening the notation so that the decision function can be re-written as $y(W^T x + b) \geq 1$. Two new parallel hyperplanes $W^T x + b = 1$ and $W^T x + b = -1$ are introduced to define each side of the margin separating the two data classes. The distance between these two planes shown in Equation 2 is the total width of the margin.

$$D = \frac{2}{\sqrt{W^T W}}$$  \hspace{1cm} (2)
The objective now is to maximize the size of the margin between the decision boundary and the nearest samples for each class (i.e. the support vectors) under the constraint that the decision function holds true for all samples. This can be represented as a minimization problem by inverting \( D \) (Equation 3).

\[
W_{opt}, b_{opt} = \min_{W,b} \left( \frac{W^T W}{2} \right): y_i (W^T x_i + b) \geq 1 \, (\forall x_i) \tag{3}
\]

Once the values of \( W \) and \( b \) have been calculated, they can then be used in the decision boundary equation to separate the two classes of data.

**K-Nearest Neighbors**

The K-Nearest Neighbors algorithm (K-NN) is a second popular supervised learning method. For the purposes of this research, the K-NN model will be used as an alternative to the SVM model for classifying feature vectors. The main advantage of K-NN is its intuitive simplicity and its ease of implementation [9].

Similar to the SVM algorithm, the K-NN is presented with a set of data samples and their associated class labels. Each data sample then is represented as a point in \( n \)-dimensions. Once this model has been established, new samples can be classified by representing the new sample as a point in space and determining the class labels of the \( K \) samples closest to the new sample. A class label is then assigned to the new point based upon which class labels are most frequent among the \( K \) samples [9]. \( K \) is a parameter whose optimal value varies depending on the type of dataset being classified. The
methods of Cross-Validation described in Chapter 5 can be used to determine the optimal value of $K$ for a given dataset.

**VIRAT Dataset**

The dataset form which both the SAENN and PCA algorithms will use to extract features from is known as the Video and Image Retrieval and Analysis Tool or VIRAT dataset. Developed by the Defense Advanced Research Projects Agency (DARPA) (an agency within the Department of Defense) [10], the VIRAT dataset is designed to provide a natural and diverse set of sample surveillance videos selected specifically for the development of content-based searching software. A set of annotation files corresponding to each video is provided to identify, and track events and persons/objects of interest throughout the video [10]. While there are a number of other classes of annotated objects in the VIRAT database, the subset of regions of interest which we use in this experiment consists of *people*, *cars*, and *bicycles*. These three classes supply many examples appropriate for training and testing our methods.
Figure 2.1 Sample Frames from the VIRAT Dataset
Chapter 3

Neural Networks and Autoencoders

A sparse autoencoder is a specific type of neural network. Therefore before describing a sparse autoencoder in its entirety, a bit of background into neural networks and their functionality is first required. This chapter will provide an overview of the functionality of neural networks and autoencoders.

Artificial Neurons and the Sigmoid Function

A neural network is a type of mathematical learning algorithm that can be used to recognize complex patterns in data. These algorithms are modeled as systems of interconnected neurons with weighted connections as shown in Figure 3.1. A neuron is an individual computational unit that maps the sum of its inputs \( x^{(i)} \) to a defined “activation function”.

![Figure 3.1 Model of an artificial neuron](image)

Figure 3.1 Model of an artificial neuron
The activation function being used in this project is the sigmoid function, shown in Figure 3.2, where $x$ are the neuron inputs, $W$ are the neuron weights, and $b$ is a bias term. The sigmoid function is one of the most commonly used curves used in complex earning systems. It is a smooth, monotonically increasing function with a value near zero for large negative inputs, and a value near 1 for large positive inputs. Colloquially, this function describes whether the neuron is turned “on” or “off”.

\[
a(W, x) = f \left[ \sum_{i=1}^{\alpha} W^{(i)} x^{(i)} + b \right]
\]  

(4)

**Figure 3.2** Plot of Sigmoid Curve
There are several reasons why the sigmoid curve is used as the activation function for the neurons within the network. The sigmoid function is a smooth curve that is differentiable over all real values. A second advantage is that the sigmoid curve’s derivative is directly based off of the original function (3). This is a useful property because the derivative of the sigmoid curve must be calculated multiple times in order to train the network using gradient descent [2].

\[ f'(x) = f(x)(1 - f(x)) \]  

(6)

**Neural Networks**

A neural network is formed by arranging neurons into layers and then connecting the output of each neuron in the \(n^{th}\) layer to the input of each neuron in the \((n+1)^{th}\) layer [2]. The significance of this type of algorithm is that it is capable of learning from the data it is presented with (a.k.a. a machine learning algorithm). The model works by passing data through the network’s layers to generate an initial approximation or guess of what the desired output will be. The difference between the desired output and the network’s approximation is determined and this data is used to adjust the weighted connections of the network. This process is repeated on multiple training examples until the error of the output approximation no longer improves. More detail on this process will be provided in the Sparse Autoencoders section.
One commercial example of an application of neural networks is with the U. S. postal service [11]. A backpropagating neural network was used to identify zip codes that were handwritten on various items of mail. The zip code data was presented to the neural network as an image of the handwritten numbers. This neural network could then be trained to identify the numbers and translate them into digital numeric characters.

**Sparse Autoencoders**

The autoencoder model being used in this project is a specific type of neural network that consists of 3 layers: one input layer, one hidden layer, and one output layer (shown in Figure 3.3). However, more complex autoencoders can use multiple hidden layers of neurons. An autoencoder is a neural network in which we desire the output to match the input as closely as possible, for many input data points which are drawn from a general class of data (i.e. patches from video clips, handwritten digits, natural images). Thus, as long as we insure that compression is occurring within the neural network, the overall structure of that dataset can be determined. The basis functions (i.e. \(W_1\) and \(b_1\) in Figure 3.3), which determine that structure, can then be used to calculate feature vectors which can be used for classification purposes.
In this project, we are using a method called gradient descent to adjust the weights of the autoencoding network. The weights are adjusted to optimize or find the minimum value of a cost function that uses the weighted connections of the network as variables. Gradient descent works by iteratively taking steps that are opposite to the gradient of the function being optimized (in this case the cost function). A method called backpropagation is used to efficiently compute the gradient of the cost function. In this method the training data is passed forward through the network to generate an initial reconstruction (shown in Equation 7). The function to produce this reconstruction is shown in Equation 7 where $x$ and $\hat{x}$ are $(m)\times1$ data vectors, $W_1$ and $W_2$ are $(m)\times(n)$ weight matrices, $b_1$ and $b_2$ are $(m)\times1$ and $(n)\times1$, respectively, sized biased vectors, and $f(\cdot)$ is the vectorized sigmoid function. $(m)$ is the number of dimensions of each input sample, and $(n)$ is the number of dimensions of the hidden layer. While performing this

![Autoencoder Neural Network Structure](image)

Figure 3.3 Autoencoder Neural Network Structure
step, one must be conscious of amount of available memory/computing speed when choosing a size for the training samples (see Creating Training Sets in Chapter 4).

\[ \hat{x} = f(W_2f(W_1x + b_1) + b_2) \]  

(7)

The cost function of a sparse autoencoder (Equation 8) incorporates three separate terms [2]. The first is a mean squared error term (Equation 9) where \( \hat{x}^i \) is the \( i^{th} \) sample in approximation data matrix with \( m \) samples, and \( x^i \) is the \( i^{th} \) original input sample in a data matrix of \( m \) samples. This term causes the networks weighted connections to adjust based on a comparison between the input data, and the output approximation. The second term is a regularization term (9) \( W_l \) is the weight matrix between layers \( l \) and \( (l + 1) \), and \( \lambda \) is a weight decay parameter of 0.0001 [2]. This forces the values of the networks weighted connections to stay at values near zero as the cost function is optimized.

\[ J_{sparse}(W, b) = J(W, b) + \beta \sum_{j=1}^{(n)} KL(\rho \| \hat{\rho}^{(j)}) \]  

(8)

\[ J(W, b) = \left[ \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \| \hat{x}^i - x^i \|^2 \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} (W_l)^2 \]  

(9)

The third term is a sparsity penalty term based off the Kullback-Leibler divergence of \( \hat{\rho}^{(j)} \) from \( \rho \) (Equation 11) [2], where \( \beta \) is a parameter used to control the Sparsity term’s weight within the cost function, \( \rho \) is a Sparsity parameter that controls the desired average activation of each hidden node, and \( \hat{\rho}^{(j)} \) is the \( j^{th} \) component of \( \hat{\rho} \), the actual average activation of the hidden node defined in Equation 10.

\[ \hat{\rho} = \frac{1}{m} \sum_{i=1}^{m} f(W_1x^i + b_1) \]  

(10)
This term is the most important for allowing the autoencoder to find features from within the data. The sparsity penalty term forces the average activation of all of the neurons in the hidden layer to stay at a specified value called the sparsity parameter. In other words this means that for a given example, only a certain number of the network’s hidden nodes are allowed to be “turned on”. By doing this the autoencoder is forced to determine any underlying structure in the input data as it creates an approximation [2].

\[
KL(\rho \| \hat{\rho}(j)) = \rho \log \frac{\rho}{\hat{\rho}(j)} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}(j)}
\]  

(11)
Chapter 4

Extracting Feature Vectors

This chapter gives a detailed description of the processes used to extract feature vectors from the video data set. There are three main steps in this process. The first is processing the video data so that it can be formed into training sets to be used with the Sparse Autoencoding Neural Network algorithm. This step involves getting rid of any excess data that is not relevant to the features we are trying to extract. It also involves identifying each object within the video and establishing a region of interest around that object. The second step involves training the network to produce a set of network weight values and bias terms. The final step is to pool over each region of interest clip to develop the key features within that data.

Video Processing

There are a number of operations that are performed on the VIRAT dataset before it can be used as training data for an autoencoder. The original dataset is composed of 308 full color 1080p stationary videos. Since we are only interested in extracting features that correspond to foreground objects within the video, we want to only pass the relevant information to the autoencoder. Much of the video data consists of background, which acts mainly as a distraction for the machine learning algorithms and is often unhelpful or
even a hindrance in making correct classification decisions. Thus, we desire to mask the objects under consideration and remove the background clutter [12].

There are three main operations involved in preprocessing the video data. The first is a background subtraction algorithm. The second is using the annotation files to only select data from regions within the video that contain the objects of interest. The third involves generating examples by extracting “patches” from the regions of interest and reshaping them into column vectors. A patch is a small sample of data extracted from one or more video frames. When extracting spatio-temporal features each patch is generated by sampling data from over a range of frames. Lastly, each example must be normalized. The details of each portion of the pre-processing code are covered in the following sections.

**Gaussian Mixture Model Background Subtraction**

The first operation is to apply a Mixture of Gaussian (MOG) background subtraction algorithm to highlight only the foreground objects in the video. This ensures that the only features that are extracted from the data are ones that correspond to objects in the foreground. To do this most effectively the image is first converted to grayscale. The histogram of the image is then equalized to enhance each frame’s contrast (Figure 4.1). This makes it easier for the MOG algorithm to pick up on changes between foreground objects and the background.
The Mixture of Gaussian algorithm itself works by modeling each pixel’s intensity value as an instance belonging to a Gaussian distribution composed of \( c \) classes [13].

\[
p(x^{(i)}) = \sum_{c=1}^{nc} w_c \cdot \eta(x^{(i)}, \mu_c, \Sigma_c)
\]

(12)

Each Gaussian distribution is described by a parameter vector \( \theta \) consisting of three terms where \( w_c \) is the weight of each class, \( \mu_c \) is the mean of each class, and \( \Sigma_c \) is the covariance matrix of each class.

\[
\theta = \{w_c, \mu_c, \Sigma_c\}
\]

(13)

The parameter vector \( \theta \) is iteratively re-estimated using an expectation-maximization (EM) algorithm. This is done by evaluating the probability that each input value belongs to a given class \( c \). The parameter vector \( \theta \) is then adjusted to account for the new value of \( x^{(i)} \). This process is repeated for each frame. During each iteration of the EM algorithm, values of \( x^{(i)} \) that do not represent this Gaussian mixture are set to a high value, while values that do are set to a low value. The result is a binary mask (shown in Figure 4.2) that is produced for each video frame.

A median filter is run over each frame of the mask to reduce any random “noise” that the MOG background subtraction algorithm may have missed (Figure 4.3). The object outlines in the foreground mask are then dilated using a structuring element and then eroded. This process helps fill in any minor holes in the mask located in the foreground object outlines. Lastly the binary mask is converted to a grayscale mask by multiplying each mask frame by the grayscale version of that frame (Figure 4.2).
Figure 4.1 Grayscale Image with Applied Mask

Figure 4.2 Left: Original Mask, Right: Modified Mask

Figure 4.3 Left: Grayscale Image, Right: Grayscale with Equalized Histogram
Establishing Regions of Interest

The second operation involves breaking each video file down into smaller clips that only include the regions of interest (ROI) around the objects that we wish to extract features from. The task of identifying each object and tracking its motion is beyond the scope of this project. The annotation files that are provided as part of the VIRAT dataset were exploited as a means of identifying moving objects. There are a few minor complications that come about when using the annotation files for the purposes of breaking each video into ROI clips that are suitable for this project’s needs.

One complication comes about from the fact that each ROI will later be used as part of a pooling algorithm and therefore must have a standard frame size. The annotation files do not provide uniform frame dimensions for tracking an object. The solution is to select the maximum frame width and height and use this for the entire duration that the object is in the video. To maintain a standard frame size when an object of interest moves close to the edge of the video frame, a special case must be implemented where a mask is created and the missing portion of the ROI frame is replaced with 0 values.

Training examples can then be generated by selecting a patch (i.e a \((h \times w \times t)\) sized block of pixels) from within each ROI file. If each training example is created using data from a single video frame, the autoencoder will only be able to extract spatial features from within the data. If each training example is created using data from multiple video frames, the autoencoder can potentially extract both spatial and spatio-temporal features from within the data. In this project each original patch size is 8 pixels wide, by 8 pixels high, by 8 frames \((8 \times 8 \times 8)\).
Creating Training Sets

Once each training example is selected, the data is reshaped into a column vector and then these column vectors are stacked together to form a training set matrix. This data matrix is the format used both for training the autoencoder and performing PCA. The matrix columns are the number of dimensions/data points within each example and the rows are the number of examples in the training set.

The last operation that must be performed on each training set before it is passed to the autoencoder is normalizing each data point to a value between 0.1 - 0.9. The standard range of intensities for grayscale images is 0 - 255. These values can be normalized between two arbitrary points $a$, and $b$ by using Equation 14; where in this case $a = 0.1$, $b = 0.9$, $X_{\text{min}} = 0$, and $X_{\text{max}} = 255$.

$$X' = \frac{(X - X_{\text{min}})(b - a)}{X_{\text{max}} - X_{\text{min}}} + a \Rightarrow \frac{(0.8)X}{255} + 0.1$$ (14)

This normalization step is required because the sigmoid function used by each neuron can only produce outputs within that range. The input values must be scaled to match these outputs since the backpropagation algorithm uses comparisons between the output approximations and the inputs in order to calculate the networks gradient values.

Tuning Sparse Autoencoder Parameters

To ensure that the best quality features are extracted using the SAENN, it is first necessary to determine the optimal amount of sparsity to use when encoding the input data. This optimal sparsity parameter can vary depending upon the type of training data.
being used with the autoencoder. To determine this optimal sparsity parameter, multiple training runs are performed using a range of sparsity values. A smaller dataset will be used to shorten the training time. The mean squared error (MSE) of the data reconstructions will be used as criteria to determine the optimal amount of sparsity to use for the dataset. Lower MSE values can be associated with better quality feature vectors. Nine separate training sessions were run on the same dataset using the training parameters shown in Table 1.

<table>
<thead>
<tr>
<th>SAENN Training Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td># Visible Nodes</td>
</tr>
<tr>
<td>$(8 \cdot 8 \cdot 8) = 512$</td>
</tr>
</tbody>
</table>

Table 1 Training Parameters for Determining Optimal Sparsity

A plot of the MSE of data reconstructions over several sparsity values is shown in Figure 4.4.

![Plot of MSE of data reconstructions vs. sparsity](image)

Figure 4.4 Plot of MSE of data reconstructions vs. sparsity
Based on the results from Figure 4.4, a sparsity of 0.45 corresponds to the lowest MSE value using 360 hidden nodes. This sparisty value was selected and used during the process of extracting features.

**Reconstructing/Visualizing Data**

Part of the autoencoder training process described in Chapter 2 involves generating reconstructions of the input data. These reconstructions, however, are in the form of a column vector and must be properly processed before they can be visually displayed. Visual reconstructions of the autoencoder’s approximations can be generated by inverting the process used to create each of the training examples. First, the reconstructions must be reshaped from a column vector back into the original patch size. This is done by simply reversing the shaping methods used to create each column vector training example. If spatio-temporal data is being used, each frame in the sample is plotted alongside one another on a single axis. A second module is called to set up an image grid that will display the desired samples and reconstructions. When implementing

![Figure 4.5 Visual Example of Data Reconstructions](image-url)
this step it is necessary to ensure that the intensity values for each image in the grid are normalized to the same range of values. Multiple instances of these image grids can be generated and saved using the network parameters from multiple training sessions in order to illustrate the change in accuracy of the reconstructions as the autoencoder is trained (Figure 4.5).

**Pooling**

For the purposes of activity and object recognition in video data, we wish to generate a feature vector representing an object appearing inside a moving ROI. Machine learning algorithms, such as a SAENN, a SVM, or the K-NN method, generally require that each sample be presented in the form of a fixed-length data vector. This becomes an issue because the ROI boundaries around objects we wish to classify can vary in both frame size and duration. Training a large neural network using gradient descent can be a very computationally costly procedure. Larger training samples require more computational power because the network size must grow to match the dimensions of the input data vector. Anything over a few hundred pixels in the input layer is both computationally infeasible, and also requires many more training examples. These restrictions make the ROI clips far too large to be converted into single training samples. The solution to each of these problems is to utilize the technique of pooling, non-linear method of down-sampling.

In the creation of a feature vector for input into a SVM or K-NN routine, a SAENN is first trained on samples created from much smaller “patches”. Each patch is a
small sample of data with dimensions of \((x \text{ pix} \times y \text{ pix} \times t \text{ frames})\) representing the size of a cube in pixels. These patches are extracted from points in various ROI clips where motion is occurring. Sets are then created using large numbers of patches and used to train the SAENN. Once the network has been trained, the network can then be used to extract a feature vector from any patch-sized cube inside the ROI clip. Equation 15 describes the function by which this feature vector is formed.

\[
a_2(x) = f(W_1 x + b_1)
\]

(15)

Where \(x\) is the patch input vector, \(a_2\) are the activation outputs of the network’s hidden layer, \(f(\cdot)\) is the activation function described in Chapter 3, and \(W_1\) and \(b_1\) are the first layer feature basis vectors and biases. Active neurons in the hidden layer (i.e. outputting a value near one, rather than zero) indicate that the corresponding basis vector appears somewhere within the given \((h \times w \times t)\) patch cube.

Therefore, a commonly used method which generates a feature vector that represents an entire ROI video cube is to ask the question of which basis vectors appear anywhere within the clip. This is done by extracting the maximum value of each basis vector component (i.e. the maximum value of activation, for each hidden node) taken over all possible patches within the ROI clip. This concept, that of determining the maximum basis vector values over all possible locations in the clip, is called pooling. It is the process of convolving each basis vector throughout the entire ROI clip to create a single feature vector.

The method of pooling allows us to use much smaller samples to when training the autoencoder. An additional benefit that pooling provides is shift invariance – the
location of the appearance of features no longer effects the output value of the feature vector. The following is a detailed explanation of how this method is implemented.

The feature set for each video file includes the maximum activations of the neurons in the hidden layer. There are several functions that were used when implementing the pooling code. The first function is used to set up and cycle through the object ROI clips. This function is also in charge of saving the maximum activations for each object ROI clip and the corresponding object class data.

The second module opens each object ROI file and sets up an initial “cube” of data within the video to be pooled over, as well as a smaller data cube that matches the dimensions of the training examples used with the SAENN (shown in Figure 4.6). In this experiment, each training example has the dimensions of \( (h = 8 \text{ pix}, w = 8 \text{ pix}, t = 8 \text{ frames}) \). This “patch” cube is iteratively passed throughout the larger ROI clip over the \( x \), \( y \), and \( t \) axes.

The third module is called during each of the iterations performed by the second module. This module takes the “patch” cube in its current position within the ROI cube and reshapes it into a column vector. This data is then passed half way through the network to evaluate the activations of the network’s hidden nodes.

As the “patch” cube is iteratively passed throughout the entire ROI cube, the maximum activations of the hidden nodes are saved. Once this code has completed, these activations are stored and are used as a feature vector representing that ROI clip.
Comparison to PCA method

Principal Component Analysis (PCA) is a method of compressing data by finding a linear projection of the data onto a lower dimensional subspace. The subspace is chosen such that maximum variance in the data is retained; this is also known as finding the principal components of the data.

PCA is implemented by performing the following operations. The data is first zero-men normalized. This is done by subtracting the mean, $\bar{X}$, from each of the
dimensions of the data. The data is then placed into a matrix to create the data set. The matrix rows consist of separate examples and the columns consist of the number of data points in each example. The covariance matrix of this dataset is then calculated and the eigenvectors and eigenvalues are determined. To discard the dimensions in which the data has the least variance, only the eigenvectors that correspond to the largest eigenvalues are used. This is represented as \( f(\cdots) \) in (8). The data is then projected into the subspace by performing the matrix multiplication in equation 4. \( X \) is a matrix with each of the normalized data items in each column, and \( G \) is a matrix of the calculated eigenvectors. This creates the compressed representation of the dataset, shown in equation 4 as \( Y \).

\[
Y = f(G^T) \cdot (X - \bar{X})
\]  

To generate a reconstruction of the input data, \( \hat{X} \), we simply transpose the matrix that contains the used eigenvectors, multiply it by the compressed representation, and add the mean back into each dimension of the data.

\[
\hat{X} = G \cdot Y + \bar{X}
\]  

The representation that is created will not be one hundred percent accurate due to the fact that some of the dimensions are nulled when some of the eigenvectors were discarded. The general idea behind PCA is that minimal information in the data should be lost because most of the data’s most important dimensions were retained throughout the compression process.
Feature Extraction with PCA

Using PCA as a method of feature extraction is a fairly straightforward process. It is based on determining the proper mean and eigenvector values ($\bar{X}$ and $G$) to transform sample into a given subspace. To determine both values, the PCA algorithm must be “trained” using as large a dataset as possible. This results in mean and eigenvector values that are representative of the entire set. Once these values have been determined the dimensionality of each feature is determined by selecting the number of least-variance eigenvectors to remove. The feature vectors ($Y$) are then extracted by applying Equation 16.
Chapter 5

Results

A two-fold comparison of the SAENN and PCA extracted features serves as a performance evaluation of the SAENN model for the purpose of temporal object recognition: reconstruction accuracy, and classification performance. Reconstructions will be compared using mean squared error to rate their accuracy. Each method’s extracted feature vectors are then used to train two classification algorithms. The first is a support vector machine model using a linear kernel and the second is a K-nearest neighbors algorithm. The theory behind each model is described in “Related Work” section of Chapter 2. The yielded classification accuracy from each algorithm is used as used to indicate the quality of each feature vector set. The following sections provide details on the obtained reconstruction MSE and classification accuracy values described in Chapter 1.

Cross Validation

The term “cross-validation” refers to any of a set of methods which allow the experimenter to determine the validity of a classifier. It is important to ensure that a classifier has not been over-trained on a particular set of data, resulting in good results for that specific dataset but poor results in general. Cross-validation is most importantly used in choosing parameters for particular learning models (e.g. $K$ in K-nearest neighbors, or the variance of the kernel in a Gaussian-based support vector machine).
Cross-validation can also be used to get better estimates of classification performance when only a small amount of training and testing data is available. Finally, in this work, cross-validation is used to obtain an estimate of the accuracy of our classification performance values. Detailed descriptions on two separate methods of cross validation are described in the following sections.

**K-Fold Cross Validation**

When implementing K-Fold cross validation, the feature set is first segmented into K equally sized blocks [14]. The supervised learning algorithm is then trained using \((K - 1)\) of the segmented blocks. The last block is used as the testing block to determine an initial classification accuracy value. The process of training the algorithm is then repeated \(K\) times. Each time a different block is used for testing and the remaining blocks are used to for training. Once the algorithm has been evaluated \(K\) times the average of the classification accuracy values is taken and used as the final value (Equation 18). The advantage of this method is that each feature in the set used for both training and testing the algorithm [14]. Also, by taking the standard deviation of the set of classification performance values, we can get an idea of the robustness of the algorithm; i.e. whether it performs at approximately the same capability over various data sets.

\[
E = \sum_{i=1}^{K} E_i
\]  

(18)
Random Sampling

Random sampling is implemented by randomly selecting feature set without replacement to generate a testing set of a given size [14]. The remaining samples are used for training the algorithm. The process is then repeated $K$ times (where $K$ is a parameter selected by the user) with new random samples selected on each iteration. Once again, the average of the classification accuracy values is used as the final value (Equation 18). The advantage to this method is that, unlike the K-Fold method, the number of training iterations ($K$) is independent of the size of the set selected for testing.

Reconstruction MSE

Reconstruction accuracy is the first condition used to evaluate the SAENNs spatio-temporal feature extraction performance. The number of dimension used in PCA compression is determined by accounting for sparsity restrictions placed on the compressed data in the SAENN model. To make a fair comparison, the number of compressed dimensions is based off of the number of active (i.e. with values not close to zero) dimensions utilized by the SAENN model in a single example. This number can be approximated by multiplying the sparsity parameter by the number of dimensions of the compressed data. Table 2 provides training parameters along with the resulting MSE values of reconstructions generated using both SAENN and PCA. The table indicates that the SAENN reconstruction is significantly superior to the reconstruction obtained via PCA. This can be shown by comparing the MSE values obtained from both methods. The
SAENN’s reconstruction error of 0.00267 is approximately twenty times lower than the MSE value obtained with PCA.

<table>
<thead>
<tr>
<th>Testing Parameters</th>
<th>SAENN</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patch Dimensions</td>
<td>((X=8, Y=8, T=8))</td>
<td>((X=8, Y=8, T=8))</td>
</tr>
<tr>
<td>Total Dims of Input Data</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Dims of Compressed Data</td>
<td>720</td>
<td>162</td>
</tr>
<tr>
<td>Sparsity Parameter</td>
<td>0.225</td>
<td>N/A</td>
</tr>
<tr>
<td>Number of Training Samples</td>
<td>300,000</td>
<td>300,000</td>
</tr>
<tr>
<td>Number of Testing Samples</td>
<td>20,000</td>
<td>20,000</td>
</tr>
<tr>
<td>Mean Squared Error of Reconstructions</td>
<td>0.00267</td>
<td>0.05577</td>
</tr>
</tbody>
</table>

Table 2 Mean Squared Error of Reconstructions

**Classification Accuracy**

Two separate learning algorithms are used to classify and evaluate the feature vectors extracted by the SAENN. The first is a support vector machine model using a linear kernel and the second is a K-nearest neighbors algorithm. The theory behind each model is described in “Supervised Machine Learning” section of Chapter 2.

Each feature vector set is generated using the testing parameters listed in Table 2. They are divided into three object classes: *people*, *cars*, and *bicycles* as described in VIRAT Dataset in Chapter 2. The number of examples in the dataset used for testing and training that are in each category is given in Table 3. Table 4 shows the confusion matrix for each feature extraction method using a SVM as a classifier. Table 5 shows the confusion matrix for each feature extraction method using K-NN as a classifier. Table 6 shows the standard deviation in accuracy across each of the K folds.
Table 3 Description of the Training and Testing Dataset

The values in each matrix are generated by summing the results from 10 k-fold cross validations. The dataset is divided into 10 subsets, trained on 90% and tested on the remaining 10%, and repeated over all 10 sets (a detailed explanation is provided above). K-fold cross validation was chosen out of the previously described two methods because it ensures that each sample is used for both training and testing. Note that the sum of predicted objects for a given class does not equal the total number of samples in that class. This discrepancy arises from a single non-uniform subset that is created during the K-fold cross validation process. More specifically, the number of samples in the class in the dataset is non-divisible by 10 resulting in a few training samples being discarded due to round-off.

Table 4 SVM Confusion Matrix

<table>
<thead>
<tr>
<th>Actual</th>
<th>SAENN-Features</th>
<th>Classification</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>People</td>
<td>People</td>
<td>371</td>
</tr>
<tr>
<td></td>
<td>Automobiles</td>
<td>Automobiles</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Bicycles</td>
<td>Bicycles</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Total Accuracy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actual</th>
<th>PCA-Features</th>
<th>Classification</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>People</td>
<td>People</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td>Automobiles</td>
<td>Automobiles</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Bicycles</td>
<td>Bicycles</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Total Accuracy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The parameter $K$ used in the K-NN algorithm is selected through methods of cross validation using different values of $K$. The optimal value of $K$ determined for this dataset is 9.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Classification</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>345 8 21</td>
<td>92.25%</td>
</tr>
<tr>
<td>Automobiles</td>
<td>63 149 105</td>
<td>47.00%</td>
</tr>
<tr>
<td>Bicycles</td>
<td>35 28 266</td>
<td>80.85%</td>
</tr>
<tr>
<td>Total Accuracy</td>
<td></td>
<td>74.51%</td>
</tr>
</tbody>
</table>

Table 5 K-NN Confusion Matrix

<table>
<thead>
<tr>
<th>Actual</th>
<th>Classification</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>342 9 20</td>
<td>92.18%</td>
</tr>
<tr>
<td>Automobiles</td>
<td>87 139 90</td>
<td>43.99%</td>
</tr>
<tr>
<td>Bicycles</td>
<td>39 57 237</td>
<td>71.17%</td>
</tr>
<tr>
<td>Total Accuracy</td>
<td></td>
<td>70.39%</td>
</tr>
</tbody>
</table>

Table 6 K-Fold Cross Validation Standard Deviation

<table>
<thead>
<tr>
<th></th>
<th>SAENN</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>2.4879564%</td>
<td>6.2783383%</td>
</tr>
<tr>
<td>K-NN</td>
<td>4.3800604%</td>
<td>2.7398409%</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusions

As previously stated in Chapter 1, the overall hypothesis of this research is that sparse autoencoding neural network models can effectively extract quality spatio-temporal feature vectors from video data for the purposes of object recognition. To prove this hypothesis, the autoencoder extracted feature vectors are compared to those of principle component analysis, a commonly used method of feature extraction. Two separate evaluations of the SAENN feature vectors are conducted to assess their quality. The first is through comparing data reconstructions created from the feature vectors. MSE values are used to gauge the overall quality of the reconstructions (i.e. what data is lost vs. what data is retained). Secondly, the feature vectors are used to train two supervised learning algorithms (K-nearest neighbors and a support vector machine) to determine which of the two sets allow the learning algorithms to provide better classification accuracy.

The anticipated outcome is that the results from the SAENN features, measured under both metrics, should be better than that using PCA. The distinguishing factor between the two methods is that the SAENN model forces sparsity into the feature vectors where PCA does not. Sparsity in encoding has been show in many applications to improve classification accuracy [3] because it provides the learning algorithm with a better representation of the underlying structure of the dataset. PCA assumes all data points are part of a Gaussian joint distribution and extracts orthogonal features corresponding to dimensions with the highest amounts of variance. More complex
features may require that additional restraints are placed on the data before revealing themselves. The sparse autoencoder does this by forcing a large percentage of the values in each feature vector to be near zero (see Sparse Autoencoders in Chapter 3) potentially drawing out patterns overlooked by other methods.

Results in each of the two categories provide additional validation of the proposed hypothesis. The reconstruction MSE values from the SAENN model are lower than PCA values by a factor of app. 20. SVM classification accuracy when trained on the SAENN feature set provided 25.78% higher classification accuracy than when trained on PCA-derived features. SAENN features clearly appear to outperform PCA features when used in spatio-temporal datasets, and thus deserve additional research in use in activity identification in video.

**Future Work**

Future work in this research involves further exploring and improving several of the methods described in Chapter 4.

Experimenting with color data to train the SAENN model may be an improvement in the video processing methods. A color image may contain additional features that are eliminated when each frame is converted to grayscale. Training the SAENN model with different sized patches is another potential area of exploration. This would adjust the level of complexity in the features that the autoencoder is being trained to extract (i.e., higher level features may classify better than lower level features). Another area of potential improvement is in the max pooling algorithm. Our pooling algorithm
creates a feature vector by convolving data from every 4\textsuperscript{th} pixel on the x and y axes and every 8\textsuperscript{th} frame on the t axis (see Pooling in Chapter 4). These increments were chosen to decrease the computational run-time of the code. Lowering these increments could allow the algorithm to construct a feature vector that more closely represents each ROI clip. A second approach would be to utilize multiple layers of pooling, similar to the HMAX algorithm for deriving features from static images, a separate method that would allow us to convolve data on multiple scales [15]. Lastly, employing algorithms to eliminate outlying samples may provide better classification accuracy. Marcks used Restricted Forward Selection (RFS) to determine the best performing feature vectors before classifying them [6]. Utilizing this algorithm may yield better classification values.
Bibliography


Academic Vita

Kyle Casterline  
191 Twin Brook Rd.  
Shickshinny, PA 18655  
KyleAC487@gmail.com

Education:
- The Pennsylvania State University, Spring 2014  
  - Bachelors of Science with honors in Electrical Engineering  
  - Thesis Title: Neural Network Feature Extraction for Activity Recognition in Video Data  
  - Thesis Advisor: Dr. Shashi Phoha

Honors and Awards:
- Schreyer’s Honors College Scholar  
- Member of Epsilon chapter of IEEE-Eta Kappa Nu Engineering Honor Society  
- Dean’s List  
- U.S. Steel Trustee Scholarship  
- Donald W. Hamer Electrical Engineering Scholarship  
- Ken Pollock Scholarship

Computer Skills:
- Programming experience in Python, MATLAB, and LabVIEW  
- Experience with NI Multisim/Ultiboard  
- Experienced with Microsoft Office  
- Some experience with 3D printing