Accelerating Classifier Training using AdaBoost within Cascades of Boosted Ensembles

A thesis presented in partial
fulfillment of the requirements

for the degree of

Master of Science in Computer Sciences

at Massey University, Auckland, New Zealand

Teo Susnjak

2009

Contents

List of Figures

List of Tables

List of Notations

α_t	Factor associated to the error of h_t 15
d	Number of dimensions representing a sample instance 2
D_n	The training dataset consisting of n samples 2
$D_t(i)$	Weight of sample i at round t 15
$f(x)$	Classification function applied on a sample x 3
F	False detection rate 42
H	Strong hypothesis - strong classifier function 15
h_t	Weak hypothesis -weak classifier at the t^{th} boosting round 15
i^{th}	Position of an element in a sequence 2
m	Number of intervals used to discretize an input domain. .16
N	Negative samples 42
P	Positive samples 42
$\mathbb{R}$	Real numbers 2
T	Total number of boosting rounds 15
$\mathcal{V}(f(h_t), x, y)$	Multi-class, confidence vote resolution function 19
x_i	Feature array 2
y_i	Class label array 2

Abstract

This thesis seeks to address current problems encountered when training classifiers within the framework of cascades of boosted ensembles (CoBE). At present, a significant challenge facing this framework are inordinate classifier training runtimes. In some cases, it can take days or weeks (Viola and Jones, 2004; Verschae et al., 2008) to train a classifier. The protracted training runtimes are an obstacle to the wider use of this framework (Brubaker et al., 2006). They also hinder the process of producing effective object detection applications and make the testing of new theories and algorithms, as well as verifications of others research, a considerable challenge (McCane and Novins, 2003).

An additional shortcoming of the CoBE framework is its limited ability to train classifiers incrementally. Presently, the most reliable method of integrating new dataset information into an existing classifier, is to re-train a classifier from beginning using the combined new and old datasets. This process is inefficient. It lacks scalability and discards valuable information learned in previous training.

To deal with these challenges, this thesis extends on the research by Barczak et al. (2008), and presents alternative CoBE frameworks for training classifiers. The alternative frameworks reduce training runtimes by an order of magnitude over common CoBE frameworks and introduce additional tractability to the process. They achieve this, while preserving the generalization ability of their classifiers.

This research also introduces a new framework for incrementally training CoBE classifiers and shows how this can be done without re-training classifiers from beginning. However, the incremental framework for CoBEs has some limitations. Although it is able to improve the positive detection rates of existing classifiers, currently it is unable to lower their false detection rates.

Acknowledgements

I would like to thank my supervisor Dr. Andre Barczak for his invaluable input, encouragement and patience over the last year. Thanks also to my colleagues at the Massey University Computer Science Department who helped me with technical issues and provided a fun and stimulating environment to work in. Thanks to the Computer Science faculty members at Massey University for always being encouraging and generous with their feedback and assistance.

Thanks to my friends for still being there despite not seeing much of me. My gratitude goes to Brian and Beverly for the genuine care and strength they have given me and also to my dear parents who have always been unwavering in their support. Thanks to Sarah for being so understanding and for providing me with much motivation during the last year.

Chapter 1

Introduction

In recent years, object detection applications have become more widespread and have found their way into many everyday electronic devices. Applications designed for character recognition through to devices like digital cameras, capable of detecting faces as well as facial expressions, are now commonplace. The likelihood of this technology proliferating further is increasing. This technology has in part been made possible through innovative machine learning algorithms which produce classification results in real-time, that can accurately detect target objects from various streams of input data.

However, training classification rules to perform such tasks is not trivial. Aside from the fact that there are many tunable parameters in the training process, whose optimal configurations are not known *a priori*, the training process itself can take days or even weeks (Viola and Jones, 2004; Verschae et al., 2008) to produce an effective classification rule. This is particularly true in real-world problems with large and difficult training datasets that are necessary as input into learning algorithms. The inordinate training runtimes are not only an impediment to producing effective object detection applications, but also make the testing of new theories and algorithms, together with verifications of others' research, a significant challenge (McCane and Novins, 2003).

In addition to protracted classifier training runtimes, there are currently limited capabilities within existing frameworks to train such classifiers in an incremental manner. Often, the consequence of this requires classifiers to be re-trained when additional datasets containing new information become available. The re-training guarantees that the new information is integrated into the classification rule, but this results in a loss of all previously

learned knowledge.

Barczak et al. (2008) proposed a novel framework for training classifiers which demonstrated potential for significant reductions in training runtimes of classifiers. The framework termed, *parallel strong* classifier within the same *layer* (PSL), indicated a capability to produce efficient classifiers, but was insufficiently tested for robustness concerning their accuracy. The PSL framework also exhibited opportunities for further extensions. One of those extensions enabled the introduction of a framework as a proposed solution to the still open problem of incremental learning.

1.1 Brief Introduction to Machine Learning

Machine learning is a field of study belonging to artificial intelligence. A major focus within it, is the design of algorithms that are capable of automatically learning rules or discovering patterns within data. *Supervised learning algorithms* and more specifically *classification algorithms*, used in this thesis, are a special group of machine learning algorithms. Classification algorithms operate by being exposed to data with the task of learning to map each instance of a given set to a specified label from a closed set. Because this group of algorithms is provided with a predefined and limited number of labels to which it can map input, it is also said to belong to a broader category of *supervised learning algorithms*.

In order to train a classification algorithm to learn to perform a categorization task, a training set is required consisting of n samples with a corresponding number of labels. A training set D_n is defined as $\{(x_1, y_1), ...(x_n, y_n)\}$. Each sample consists of a feature vector $x_i = (x^{(1)}, ..., x^{(d)}) \in \mathbb{R}^d$, that comprises of d number of dimensions and an associated label $y_i \in \{-1, 1\}$ for each vector x_i. In this example the label y_i represents a binary classification problem in which a sample or feature vector x_i is labelled as 1 if positive and -1 if negative.

Given the training set D_n, the task of a learning algorithm is to analyze a training set and according to its *model*, produce a function $f(x)$, or in this instance a *classifier*, which is able take as input an *unseen* sample instance x and return a binary decision -1 or 1 which accurately matches the class it belongs to. The ability of a classifier to correctly predict class labels of unseen samples instances is referred to as the classifier's

generalization ability. In the same manner, machine learning algorithms are assessed on how well their resulting functions $f(x)$ or classifiers are capable of *generalizing*.

For classification algorithms, the goal during classifier training is to generate a function $f(x)$ that best approximates the boundary which separates positive from negative training samples. More often than not, formulating a function that can do this on large and difficult training sets is not trivial, regardless of the type of *model* used. What is more, attempting to devise a single, complex function using a given *model* that is designed to simultaneously process multiple dimensions, is intractable.

Due to this problem, this thesis employs machine learning algorithms whose *model* (boosting) for finding an optimal separation boundary between different classes employs the divide and conquer strategy. In this *model*, the final classifier is referred to as the *strong classifier*, composed of numerous (boosted) ensembles called *weak classifiers*. Each weak classifier on its own is only slightly correlated with a true classification but when combined with the remaining *weak classifiers*, the resulting *strong classifier* attains an accurate correlation.

Although using the boosting *model* for training classifiers has made the learning task more tractable on large and difficult datasets, there is still no escaping the fact that a large *strong classifier*, consisting of many thousands of *weak classifiers* has to be generated. A problem arises at detection time for detectors with real-time constraints when the *strong classifier* is required to evaluate a sample x_i. In order to produce an evaluation, a *strong classifier* needs to calculate all the *weak classifiers* in its ensemble. When a classifier is employed in a computationally demanding environment like computer vision, the calculation of every *weak classifier* in an ensemble becomes prohibitive if real-time constraints are in place.

Viola and Jones (2001*b*) solved this problem by arranging *weak classifiers* into a cascading structure in which each layer of a cascade contains only a portion of all the *weak classifiers*. They found that in order to reject a large percentage of samples as negatives, only a small number of *weak classifiers* needed to be calculated. By grouping *weak classifiers* into layers, they achieved a cascading structure which required all layers to be evaluated only for positive and for most difficult negative samples. This arrangement of *weak classifiers* enabled real-time classifier evaluation and came to be known as a *cascade*

of boosted ensembles (CoBE). The CoBE framework and the issues arising with training classifiers in this context, are the focus of this research.

This thesis will seek to make modifications to the standard CoBE framework. Due to this, a method for comparing and assessing the performances of different frameworks has to be formulated. The tendency is to simply ask which framework produces classifiers that *generalize* best on unseen data, or in other words, which of the classifiers are more accurate on a given test dataset? This however, is inadequate.

Wolpert and Macready (1997) have shown through their "No Free Lunch" (NFL) theorem for machine learning algorithms, that "any two algorithms are equivalent when their performance is averaged across all possible problems" (Wolpert and Macready, 2005). In other words, on a particular problem, some algorithms may produce divergent results, but over all problems their performance is indistinguishable. It follows that matching algorithms to specific problems will yield superior results, rather than employing the same algorithm on all. Hence, conclusive statements regarding classifier performances in experiments conducted in this research can only be made for the specific datasets which they were trained on.

There is also another perspective from which all experimental results in this research need to be evaluated. Occam's razor and a law of parsimony, attributed to the 14^{th} century English logician, William of Occam, states that precedence ought to be given to simplicity. Thus, in the presence of two competing theories, the simplest explanation of an entity is to be preferred. Occam's razor, which is largely uncontroversial (Domingos, 1998), was formulated by the same authors into the context of machine learning algorithms with its original intent preserved to state "given two models with the same generalization error, the simpler one should be preferred because simplicity is desirable in itself". Therefore, the application of Occam's razor in this research will give precedence to CoBE frameworks which produce strong classifiers comprising of fewer weak classifiers when similar performances exist.

Therefore, this research will be mindful when analyzing performances of different algorithms and will take into account the fact that some algorithms are more suited to certain datasets than others. The price of superior performance on one dataset will be reciprocated by an inferior performance on another. The analyses will also place emphasis

on the simplicity of the resulting classification rules and the ability of the classification algorithms to generate them both efficiently and rapidly.

1.2 Research Scope and Objectives

The scope of this research is to consider classification problems that deal only with binary outputs. This means that the classifiers are trained to evaluate a sample instance only on whether it is a target object (a positive), or not (a negative). By reducing problems to a question of binary-classification, the performance capabilities of proposed classifier training structures will be more emphasized and focused. The aim is to limit the research to working in a supervised learning environment using the Discrete AdaBoost machine learning algorithm, while producing boosted ensembles consisting of degenerate decision trees. Neural networks and Support Vector machines are not considered within the scope of this research. Experiments will be conducted on datasets which comprise only of pre-extracted features instead of low level data, in order to focus explicitly on the learning framework.

Within this scope, the objectives for this research are to:

- test the training runtimes and the generalization ability of the PSL classifiers against standard structures of the monolithic ensemble (Schapire, 1999) and the cascaded as proposed by Viola and Jones (2001*b*)
- explore further extensions to the PSL training structure and to perform experiments on it
- to propose a novel approach to incremental learning within the scope of CoBE using incremental PSL (IPSL)
- to explore the feasibility of the adapted PSL training structure to the problem of incremental learning and implement parallel cascaded classifiers to test against

The primary research questions posed by this thesis are:

1. is the PSL training structure efficient at producing classifiers which are robust and capable of real-time execution?

2. is it possible to extend the PSL training structure in order to realize further performance gains?

3. can CoBE be extended in such a way that existing weak classifiers need not be retrained during subsequent rounds of incremental training, and can additional weak classifiers be appended in order to integrate new information, while improving the generalization ability?

4. is the IPSL capable of incremental learning and of producing robust classifiers?

1.3 Contributions

The principal contributions of this research are experimental results confirming the capabilities of the PSL classifier training structure, as well as further extensions to the original PSL structure, together with supporting experimental results. Lastly, a novel classifier training structure is proposed with the ability to incrementally train a classifier without losing previously learned knowledge.

This thesis uses part of the work published by the author in:
Susnjak, T. and Barczak, A. L. C. (2008). The Influence of the PSL Cascade Structure on Training Time and Accuracy. In *IIMS Postgraduate Conference*.
Susnjak, T. and Barczak, A. L. C. (2008). Accelerated Classifier Training using the PSL Cascading Structure. In *International Conference on Neural Information Processing of the Asia-Pacific Neural Network Assembly*.

1.4 Structure of the Thesis

Chapter 2 introduces the theory of classification and seeks to highlight the primary components that comprise classifier training structures. Each component is explored together with different methods available to implement them. Those methods are investigated from the perspective of how the different implementations affect the training and execution phases of classifiers, as well as their eventual accuracy on *unseen* data.

Chapter 3 investigates the primary causes of protracted classifier training runtimes. Recent research and proposals to address each of the root causes are discussed with their overall effectiveness.

The principal contribution of this research is presented in Chapter 4. A proposed solution to drawn-out classifier training runtimes in the form of alternative training frameworks is presented. A novel approach is introduced to incrementally training CoBE classifiers with datasets containing unique information as they become available after the initial classifier has been trained.

The nature and the details of the experiments are described in Chapter 5, while their results are reported in the two succeeding chapters. Chapter 6 discusses experimental results of the proposed training structure as well as those of a modified version of PSL, while Chapter 7 examines the results of experiments carried out on the novel incremental classifier training structure. The final chapter closes with the conclusion.

Chapter 2

Classification Theory

This chapter explores the theory of classification within supervised learning. It does this by examining the primary components that comprise a training structure designed to produce classifiers. Chapter 1 briefly introduced the process of training a classifier. The process was presented as an algorithm which constructs rules based on an input training set D_n that is composed of samples $\{(x_1, y_1), ...(x_n, y_n)\}$, where x is a feature vector $x_i = (x^{(1)}, ..., x^{(d)})$ and y is a label $y_i \in \{-1, 1\}$. The goal of the rules is to learn to discriminate samples belonging to different classes by identifying common features in the positive set which set them apart from the negatives. This chapter examines the issues concerning the feature vector x_i, and the relevance of each feature within it. The various *models* available to the learning algorithm are also described, as well as the different types of weak classifiers that can be generated. Multi-class classification problems, in which the label y is an element of a larger set as in $y_i \in \{A, B, ..., Z\}$, is discussed together with different architectures for arranging weak classifiers h, that make up a strong classifier H. Finally, training procedures are discussed that involve different strategies available for sequences in which a training set D_n is *shown* to a learning algorithm. The various components of a training framework are discussed with their influences on the overall performance and accuracy of classifiers.

2.1 Features

In pattern recognition, a *feature* or a *descriptor*, is extracted quantitative information from data which highlights attributes of interest that can be used to differentiate one class from

another (Gonzalez and Woods, 2002). In the classification domain, the use of features as input into a learning algorithm is preferable to data that has only undergone low level processing. The preference of using features over low level data lies in the ability of the former to reduce the variability within samples of the target object class, while increasing the variability between the negative and target object classes. Features also tend to have the capacity to encode specific knowledge about the domain if they are appropriately chosen (Lienhart and Maydt, 2002). In certain domains such as computer vision, feature based systems are also much faster than pixel-based systems (Viola and Jones, 2001*b*).

Selecting the right feature representation plays a critical role in classification (Yilmaz et al., 2006) and there are a number of factors to be considered. In general, the most desirable property of a feature is its discriminatory ability to distinguish negative training samples from the positive training samples in the feature space, whilst providing robustness in noisy datasets and changing operating environments (Gonzalez and Woods, 2002). The decision on what types of features to use for the learning task is problem specific and requires *a priori* domain knowledge.

Although a strong discriminatory ability of a feature is vital to the learning process, it can also be detrimental to the generalization ability of the final classifier if a feature is too effective. When a feature is found to be too aggressive in its ability to separate positive samples from the negatives during training, the end result is overfitting of the classifier to the training data and a compromise in its generalization ability. Conversely, if a feature is discriminately too weak, then the potential generalization ability may not be realized through underfitting.

The ability for rapid feature evaluation is another property that is becoming more important (Lienhart and Maydt, 2002). This property is critical for real-time detection systems, because the speed of each feature evaluation will have a direct bearing on whether the final detector can execute in real-time. This can be best appreciated when considering video image detection programs that usually rely on brute force raster scanning of an entire image frame using a sliding window. With up to 15 frames per second to scan though, the calculation overhead of each feature has to be at a minimum.

Lastly, the ability of a chosen feature representation to behave consistently under various environmental conditions is also of importance. It is vital that feature values being

extracted from low level data in real world settings are robust to changing conditions, as this will directly affect the performance of a detector. An example of a type of variance that may be experienced in low level input data may be in the form of shading and lighting intensity fluctuations in image detectors, or voice-over-speech interference in music broadcast classification.

2.1.1 Feature Types

There exists a wide spectrum of feature types. They are highly domain specific and are chosen with respect to the environment that a detector is to operate in. Since some feature types are more expensive to calculate than others, they are also chosen in respect to the performance and accuracy requirements of the final detector.

Haar Wavelet features are common types of features used in vision detection systems whose target object displays consistent geometrical properties. Papageorgiou et al. (1998) introduced their usage to machine learning by modifying the original Haar Wavelet decomposition to allow them to perform object recognition at a finer resolution. The Haar-like Wavelets have since been used successfully by Viola and Jones (2001*b*) to create an accurate and first real-time face detection system. Haar-like features are a mixture of square and rectangle filters, each divided into two main regions. The filter is applied to an image at different scales and the value for each filter is the difference of the two regions, with each region being the sum of all its pixel intensity values. The complete Haar-like feature set is applied to an entire image frame, producing an over-complete feature space. The amount of information that accompanies this type of a feature set is many times greater than the amount of actual low level image data. Gabor Wavelet features have also been used in vision detection systems and likewise produce a rich and an over-complete feature space. They are seen as being more powerful than Haar-like features since they display a functional similarity to the cells of the visual cortex of a human (Shen and Wang, 2005), but consequently they are considerably more expensive to calculate (Whitehill and Omlin, 2006; Wu et al., 2004). In audio detection systems on the other hand, features are based on the sound spectrum and can comprise types such as energy statistics, silence ratios or pitch to identify target objects (Wang et al., 2003).

In this research however, feature types were not explicitly chosen and the machine

learning algorithms were not designed to extract features themselves. The focus in this research was on comparing performances of machine learning algorithms. The input for the learning algorithms were datasets comprised of values representing pre-calculated features, applied on various datasets. Thus, a measure of decoupling was achieved in the training process between feature extraction and the actual learning, allowing more emphasis to be placed on the machine learning algorithms.

2.1.2 Feature Extraction

Once feature types have been defined for a learning task and a set of features has been designed, they can then be applied to low level input data. The process of transforming low level input data into a set of feature values is called feature extraction. The resulting set of feature values is placed into a feature vector $x_i = (x^{(1)}, ..., x^{(d)})$. A feature vector in this case becomes a multi-dimensional vector representation of a sample object. The dimension d of each vector is determined by the number of features used to describe a target objects. The collection of various feature vectors forms the total feature space out of which the most discriminant features are selected.

The process of feature extraction can either lead to the simplification of the amount of resources required to accurately describe a large set of data or, to the burdening of it. When the process of feature extraction produces more feature values than the actual volume of low level data, then the feature space is said to be over-complete. As mentioned earlier, this occurs when Haar-like features are employed in vision detectors. Over-complete feature spaces tend to complicate learning by imposing large memory and computational demands involved in maintaining and searching through immense data volumes. However, authors like Viola and Jones (2001*b*) and Lienhart and Maydt (2002) found that this is balanced out by added domain knowledge, and eventual improved accuracy of a final classifier.

An extreme case of over-complete, feature space generation is referred to as "the curse of dimensionality" (Bellman, 1957). In this scenario, the feature extraction process leads to an exponential increase of the feature space. The learning algorithm becomes flooded with irrelevant and redundant features, thus bringing the learning process to a grinding halt. In such instances, the problem is tackled by utilizing methods of feature selection

which reduce the size of the feature space. However, when these methods are employed, care needs to be taken to ensure that features which effectively discriminate the target sets are not removed.

2.1.3 Feature Selection

Feature selection is one of the most important and difficult problem-specific elements of learning (Kulkarni et al., 1998). Once a feature space has been generated, feature selection algorithms attempt to identify and remove from it features which are unhelpful or destructive to the learning task. The outcome is a more relevant sub-set of the original feature pool. The process of feature selection alleviates the effect of the "curse of dimensionality". If this phase is completed with an appropriate algorithm, it can enhance the generalization capability as well as the speed of the learning process.

In real-world situations, relevant features are often not known *a priori* (Dash and Liu, 1997). Due to this, it becomes necessary to introduce many candidate features to more accurately represent the problem domain. The introduction of many features almost guarantees that a large number of them will be irrelevant or redundant in respect to the target object. A feature is deemed to be irrelevant when it does not affect the target object in any way, and termed as redundant if it does not add anything new to the target object (Dash and Liu, 1997). The reduction of redundant and irrelevant features becomes crucial to the shortening of the runtime of a learning algorithm.

Feature selection can be reduced to a search problem which ideally finds the optimal subset. Finding an optimal subset is computationally prohibitive in instances where massive feature spaces are generated (Pudil et al., 1994). In such cases, resorting to sub-optimal approaches such as filtering methods or greedy search algorithms is the norm. In this research, the greedy search algorithm is used which is built into AdaBoost and discussed further in Section 2.2.1.

2.2 Boosting

Boosting represents a *model* available to machine learning algorithms for building classification rules. It is a general methodology that describes a category of machine learning algorithms (Schapire, 1999), which convert "rough rules of thumb" into highly accurate

prediction rules. Since building single, highly accurate classification rules from large, real-world datasets is not trivial, boosting algorithms base themselves on the notion that building classification rules out of many "rough rules of thumb" or weak classifiers, is more feasible.

In order to implement boosting, a method for extracting weak classifiers must first be devised. Such a method is referred to as a *weak* or a *base learner*. A weak learner is called repeatedly to extract a weak classifier from the training dataset. Following each call on a weak learner, the distribution or weighting that is placed on individual training samples is altered, thus ensuring that a different weak classifier is selected at each round. After a certain number of iterations, all weak classifiers generated by the weak learner are combined into a single classification rule.

A further two fundamental concepts need to be defined before boosting can be implemented. The first deals with how the distribution of the training samples should be altered after each round of weak learning. The second addresses how all the weak classifiers are to be combined into a single strong classification rule. The different variants on these two key concepts determine what type of a boosting algorithm is produced. In this work, the Discrete AdaBoost boosting algorithm was used for all the experiments. For completeness, it is discussed more formally below, together with a brief summary of other common boosting variants.

2.2.1 Discrete AdaBoost

Since its introduction in 1995 by Freund and Schapire (Freund and Schapire, 1995), Discrete AdaBoost (or simply AdaBoost) has been widely used in different kinds of classification problems. Its proliferation can be directly attributed to its simplicity, high performance and speed (Verschae et al., 2008). AdaBoost's theoretical foundations have received much research, and its effective generalization ability has been demonstrated by Schapire (1999). Its introduction solved many practical difficulties associated with previous boosting algorithms and contributed to its popularity. One of these difficulties was the requirement that the accuracy, or the error rate of the weak learner, be known *a priori*.

AdaBoost solved this shortcoming by introducing a confidence parameter α. The confidence parameter changes values at each t^{th} iteration of weak classifier generation

according to the error rate of a weak learner. Hence, due to the ability of AdaBoost to change the error rate of its weak learner, it is said to be *adaptive*. A key feature of AdaBoost, is the distribution of weights it maintains over all samples in its training set. $D_t = \{1, ..., n\}$ denotes the distribution of n weights for a t_{th} round of boosting, on a training set D_n. AdaBoost manipulates this distribution after each round of generating a weak classifier, by placing a larger weight on misclassified samples, whilst reducing the weights of correctly classified samples. Using this strategy, it is able to focus more on samples that have previously been misclassified, thus in the process driving down the training error rate. Algorithm 1 shows the details of AdaBoost.

Input : A set of training examples $D_n = (x_1, y_1), (x_2, y_2), ...(x_n, y_n)$
Output: Strong classifier H comprised of a linear combination of weak classifiers $h_t(x_i)$.
Data: x_i is a feature $\in$ X and y_i is a class $\in \{-1, +1\}$, n is the number of elements. T denotes total number of boosting rounds. H is the final strong classifier composed of h_t weak classifiers. Function TrainWeakClassifier generates a new weak classifier per round of boosting.

$D_1(i) = 1/(n)$ `/* Initialize weights for each element */`
for $t = 1$ **to** T **do**
  $h_t =$ `TrainWeakClassifier(`D_t`)` `/* train so that` $h_t \in \{-1, +1\}$ `*/`
  $error_t = \sum_i (D_t(i) h_t(x_i) y_i)$ `/* compute error of weak classifier */`
  $\alpha_t = 0.5\ ln(\frac{(1-error_t)}{error_t})$ `/* compute alpha value */`
  for $i = 1$ **to** n **do**
    $D_{t+1}(i) = D_t(i)\ exp(-\alpha_t y_i h_t(x_i))$ `/* update weights */`
  end
  $D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_i D_{t+1}(i)}$ `/* normalize weights */`
end
return $H(x) = sign(\sum_t \alpha_t h_t(x))$

Algorithm 1: ***Discrete AdaBoost***

AdaBoost Feature Selection Section 2.1.3 discussed feature selection as a form of dimensionality reduction which produces a subset of the original feature pool. Greedy search algorithms and feature filtering were mentioned as two methods that achieve this reduction with suboptimal properties in respect to finding a final solution. By default, AdaBoost has a feature selection process built into it which comes in the form of a greedy search algorithm. It is classified as a greedy search algorithm since, at every boosting round, it is only concerned with selecting a feature for the best *current* solution, without

taking into account the ramifications on the succeeding subproblems. In this way, by not factoring into its deliberation future choices or all the solutions to the subproblem, it in effect limits itself to a subset of the possible total feature space. This limitation of the possible total feature space, indirectly accomplishes feature selection.

2.2.2 Variant Boosting Algorithms

The popularity of AdaBoost has seen numerous research efforts to extend it further, and to address some of its shortcomings. The variant AdaBoost algorithms listed below represent some of the more common and successful efforts to produce robust alternatives.

Real AdaBoost Real AdaBoost behaves much the same as Discrete AdaBoost, but differs primarily in the type of outputs it produces when each weak classifier is calculated. Weak classifiers under Real AdaBoost, are real-valued functions instead of binary ones of Discrete AdaBoost. The real-valued results of its weak classifier correspond to a confidence-rated prediction as opposed to only 1 or -1 of Discrete AdaBoost.

However, in current works the outputs of the weak classifiers are discretized to a certain extent (Le and Satoh, 2006). This means that the input domain is divided up into m intervals where each interval is termed a *bin*. The implication is that any two weak classifiers with different outputs, falling into the same bin will, register the same real-valued output. This approach has become common practice due to the possibility of overfitting when training on noisy data (Liu et al., 2002) and due to concerns over storage space and computation overheads.

Gentle AdaBoost Criticisms have been labelled against Discrete AdaBoost that it is not resilient to outliers and consequently produces weaker classification rules when trained on noisy datasets (Friedman et al., 2000). Gentle AdaBoost (Friedman et al., 2000) was designed to counter this problem. Gentle AdaBoost performs like the discrete version in every way, but differs in the amount of emphasis it places on misclassified samples. Its policy is to place a diminished emphasis on misclassified samples where the increase in the weight of each one is quadratic in the negative margin, rather than exponential in Discrete AdaBoost.

Asymmetric AdaBoost Viola and Jones (2001*a*) discuss the need for a learning algorithm that is capable of training with the target of achieving extremely high detection rates instead of primarily focusing on minimizing the total classification error. They proposed a new algorithm called Asymmetric AdaBoost, whose intended purpose is to retain a very high positive rate while accepting a modest false positive rate if necessary. Asymmetric AdaBoost achieves this by changing the re-weighting scheme of AdaBoost, so that greater weight is placed on positive training samples. Thus in essence, the "asymmetric" title refers to the skewed nature of re-weighting where the positive samples are favored.

Float Boost Li et al. (2002) introduce a variant of AdaBoost called Float Boost. The motivation behind this variant, was the supposed suboptimal nature of strong classifiers created by conventional AdaBoost (Bhlmann and Yu, 2000). Float Boost employs a unique backtracking algorithm to remove weak classifiers after each round of boosting according to a criterion. The criterion dictates that a weak classifier be removed from the ensemble if it is harmful or not useful to minimizing the error rate. This has the effect of creating strong classifiers which have fewer weak classifiers and perform at better detection rates. The improvement comes at a price of training runtimes being extended up to five times than normal boosting. In all other respects the algorithm proceeds as Real AdaBoost (Brubaker et al., 2006).

2.2.3 Weak Learner

In Section 2.2, the concept of a weak learner was introduced as a function which produces a rough rule of thumb or a weak classifier. This weak classifier is required to be only slightly better than chance, thus lending it great flexibility (Schapire, 1999). If a weak learner is found to be too strong in its ability to discriminate, then it is likely to overfit the training data. On the other hand, Mita et al. (2005) mention that the problem lies with using weak learners that are too ineffectual. They discuss that the problem is experienced particularly in latter rounds of training when the learning process becomes difficult and each additional weak classifier ceases improving the generalization. Two types of binary weak learners will be discussed here, namely decision stumps and decision trees.

Decision Stumps Decision stumps are degenerate trees consisting of only two leaves. They are the simplest of weak learners and are the ones used in this research. At every t_{th} iteration of boosting on a distribution D_t, decision stumps become associated with only one feature and together they form a weak classifier h_t, which comprises the final strong classifier $H = \{h_1, ..., h_T\}$. The chosen feature is one that produces the lowest weighted error.

This simple linear division of the sample space may not be discriminative enough for some less powerful feature types, or for very difficult datasets, thus leading to underfitting. However, two advantages can be attributed to this approach; speed being one of them. Also, by associating one feature with each weak classifier, the learning methodology under decision stumps inadvertently becomes an automatic feature selection algorithm.

Decision Trees A decision tree is a multi-level structure which can have two child nodes per parent node. This structure is used in instances where the sample space is more complex and a simple linear decision stump threshold is insufficient in its accuracy to discriminate between two classes. Lienhart, Kuranov and Pisarevsky (2003) and Brubaker et al. (2006) found that by using *classification and regression trees* (CART) they were able to improve accuracy and achieve faster convergence in training. They found that whereas weak classifiers formed by decision stumps are too weak in their discriminating power in latter stages of training, decision trees based on CART continue to improve the generalization rate.

2.2.4 Strong Classifier Type

At the conclusion of the learning process, all weak classifiers h_t are are assembled to form a final strong classifier H. There are two types of strong classifiers. The simplest is a binary-class classifier, while the more complex is a multi-class classifier. The output of the two classifier types determines what category they fall under.

Binary-Class Classifiers Binary-class classifiers are trained with the limited ability of learning the difference between what is a target object and what is not. This type of a classifier is the subject of the proposed research. During the detection phase of a candidate sample, the result of each weak classifier is multiplied by its confidence rating

α. The products are then summed as shown below, and if the final sum is above a predefined threshold, then the classification is deemed a positive, otherwise a negative.

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

Multi-Class Classifiers Multi-class classifiers have the capacity to map an input to n number of class labels instead of mapping only to a positive or a negative class label. However, it is not trivial to train a multi-class classifier. In machine learning, it is easier to work with classification problems that distinguish only between two classes (Allwein et al., 2001). A popular strategy to dealing with multi-class classification has been to extend a simple binary-class classification to a multiple binary class classification problem. In designing such a system, each class is trained against others in a one-versus-all training approach. Once all binary classifiers are created, they are then combined using a variety of approaches to comprise a multiple class classification rule (Allwein et al., 2001) as shown below:

$$H(x) = \max_{y \in Y} \mathcal{V}\left(f(h_t), x, y\right)$$

where the return value is a label y which has the highest confidence vote calculated in $f(h_t)$.

2.3 Classifier Training Structures

A classifier training structure refers to the approach employed to dealing with several key components of the learning process. These components can be summarized below as:

- the decision on when to halt and resume the boosting process with distribution weights reset to zero
- which training samples are to be *shown* to the learning algorithm at different times
- the level of independence and separation between the different boosting rounds
- the policy used for combining weak classifiers for the detection phase

This section will examine commonly used approaches to the above questions, and what the resulting classifier training structures look like. Since the focus of the proposed research involves devising training structures which accelerate runtimes of training phases, particular attention will be given to how various policies affect this phase. However, the ramifications on the detection runtime will also be examined since the goal of the research is to train classifiers that retain their real-time execution property.

Training phase issues involve two aspects. The first considers how effective a training structure is at converging to a target training error rate. The total number of weak classifiers needed to generate a convergence at training are considered. A more effective and efficient framework requires fewer weak classifiers to converge to a target error rate. The time required to calculate each weak classifier is also important and is indicative of a training structure's effectiveness. The second aspect considers a framework's ability to produce optimal classifiers for the needed task. Frameworks which do not rely on an empirical and iterative process of re-training classifiers with different tunable parameter configurations are more favorable.

Detection runtime issues deal specifically with computational complexity involved in ascertaining whether or not a given candidate sample is a target object or not. One challenge for each framework is how to create as few weak classifiers as possible in order to determine a positive object classification. While another challenge is to determine how few of the total weak classifiers need to be calculated in order to classify a candidate sample as a negative. Both properties are crucial to real-time detection systems.

An important issue referred to as "rare event detection", arises at detection time in some classification domains. This scenario concerns a highly disproportionate presence of negative samples in respect to positive samples. This asymmetric property is often found in image detection and poses difficulties. The reason for its existence in image detection for example, is due to the fact that the standard approach employed in this domain is a brute force methodology, whereby an entire image is scanned in raster using a squared sub-window of $l*x$ pixels over multiple scales (Pham and Cham, 2007). Consequently, millions of sub-windows per image undergo an evaluation, with each sub-window representing a candidate sample. Even if many target objects are present in an input image, they are proportionally rare to the number of negative sub-windows that are processed.

The introduction of a large number of negative samples into a data distribution poses two challenges. Firstly, in order for a detector to function in this environment in real-time, it needs to be trained to specifically know how to reject negative samples rapidly. This is the case since an overall speed of a detector is determined by the computational cost of a single sub-window (Brubaker et al., 2008). Secondly, the skewed nature of such domains makes the accuracy of a detector difficult to calibrate. Employing a strategy to maintain a low error rate will likely result in all positive sub-windows being rejected as negative (Wu, 2008), while relaxing the strategy would likely introduce significant numbers of false positive detections.

2.3.1 Monolithic Ensemble

Organizing weak classifiers into a single layered monolithic ensemble was the first and intuitive strategy when boosting was initially introduced. This straightforward approach to structuring a strong classifier involves appending each weak classifier to the ensemble horizontally until the required training error is satisfied.

The effects of this approach on the training phase is that there are very few tunable parameters involved, thus making the process simple. Also, the boosting process executes uninterrupted from start to finish without any re-setting of data distribution weights. Lastly, the learning algorithm is *shown* the same set of training samples throughout, meaning that more difficult negative training samples are never introduced. This raises questions as to whether very low false positive detection rates are attainable, even though Schapire (1999) has demonstrated that the generalization ability of classifiers trained in this manner have strong bounds.

There is a significant impact of a large number of weak classifiers assembled into a single layer on the detection phase. The result is that oftentimes classifiers cannot execute in real-time, since each classification requires the evaluation of all its weak classifiers.

2.3.2 Cascaded

The cascaded structure mentioned in Chapter 1, and shown here in Figure 2.1, was introduced by Viola and Jones (2001*b*) in their seminal work which resulted in the creation of a first real-time face detection system. The cascaded structure simplified the training

process by making it more modular and tractable with only a slight compromise in accuracy. Its classifier training is broken up into layers. Each layer represents an independent round of boosting. This means that the distribution weights of each training sample are reset. Each layer is given simplified target rates to achieve. Once layer target rates are fulfilled, positive training samples that have been correctly classified are passed to succeeding layers for re-training, together with all the misclassified negative training samples. The targets Viola and Jones (2001*b*) used for each layer, were to learn to reject at least 50% of the negative training samples while correctly classifying 99%-100% of the positive training samples. In the end, the nal true positive rate (TPR) and the false positive rate (FPR) of the whole classifier become the product of the corresponding rates in each layer. Although, the generation of multiple layers adds tractability to the training phase, it also introduces difficulties. Additional tunable parameters for each layer, is one issue that arises which subsequently makes classifier optimization a difficult task. Besides this, there is also an introduction of layer threshold adjustments, whose purpose is to assist in maintaining high hit rates per layer but at a considerable computational cost. The significance of both issues are discussed in more depth in Chapter 3. Algorithm 2 shows the cascaded method in more detail.

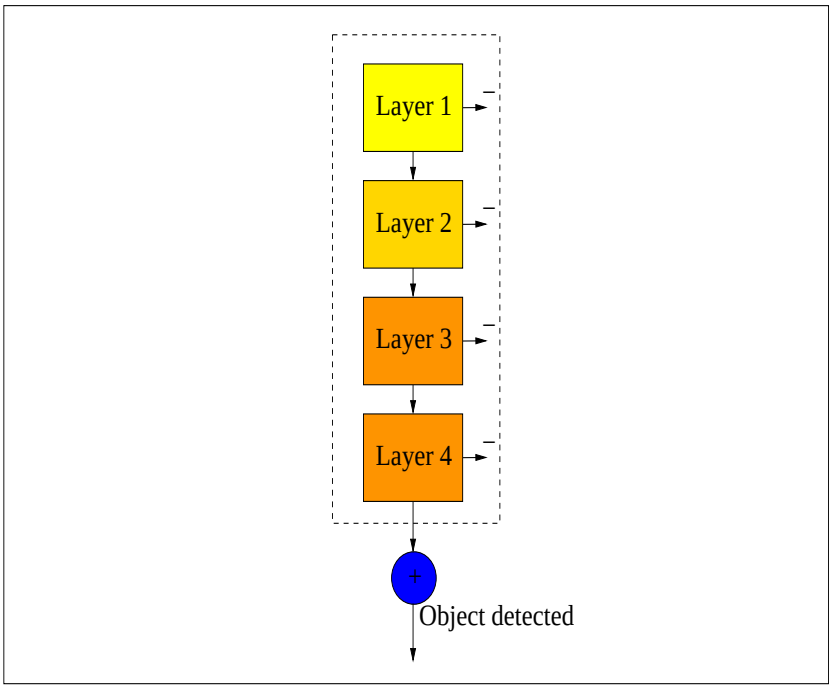

Figure 2.1: Cascaded classifier in the form proposed by Viola and Jones (2001*b*).

Input : P,N,f,d
Output: cascaded classifier
Data: P = set of positive samples, N = set of negative samples, f denotes the maximum acceptable false positive rate per layer, d denotes the minimum acceptable detection rate per layer, F_{target} is the overall positive detection rate.
TrainWeakClassifier() produces a new weak hypothesis.
TestClassifierOnValidationSet() evaluates the accuracy of a new weak weak classifier.
DecreaseClassifierThreshold() lowers the layer threshold to enable a higher positive detection rate.
EvaluateSampleAgainstCurrentClassifier() searches out new negative training samples which are misclassified by the current classifier.

```
1  F_0 = 1.0
2  D_0 = 1.0
3  i = 0
4  while F_i < F_target do
5  |  i = i + 1
6  |  n_i = 0
7  |  F_i = F_{i-1}
8  |  while F_i > f * F_{i-1} do
9  |  |  n_i = n_{i+1}
10 |  |  TrainWeakClassifier(P,N,n_i)
11 |  |  TestClassifierOnValidationSet(F_i,D_i)
12 |  |  DecreaseClassifierThreshold()
13 |  end
14 |  N = 0
15 |  if F_i > F_target then
16 |  |  foreach Image I in the set of nonTargetImages do
17 |  |  |  Res = EvaluateSampleAgainstCurrentClassifier(I)
18 |  |  |  if Res == Reject then
19 |  |  |  |  N = I
20 |  |  |  end
21 |  |  end
22 |  end
23 end
```

Algorithm 2: Viola-Jones standard cascaded structure

By learning to reject most negative samples in the first few layers, the detection runtime becomes rapid since an entire ensemble of weak classifiers does not need to be evaluated. An entire ensemble is evaluated only for the very difficult samples which are close to the decision boundary separating the classes. The performance accuracy of this system has been proven to be effective by a number of researchers, though there is a slight degradation in regards to positive detections. Criticism for the slightly lower hit rates has been

directed at how this structure handles positive training samples which are misclassified during training. Once a positive training sample is misclassified, it is removed from training. As the number of layers increases during training, so does the number of misclassified positive samples, meaning that the generalization ability of a final cascaded classifier becomes permanently compromised the longer the training continues. However, the training runtime decreases compared to the monolithic ensemble structure. It accelerates exponentially as the cascade building process advances because at each layer, there are 50% or fewer negative training samples available, and possibly up to 1% of the most difficult positive training samples that failed to be trained correctly in previous layers.

2.3.3 Parallel

The single cascaded approach works well for domains where a target object is relatively simple. However, its discriminative power between the target object and negatives becomes compromised when its complexity increases. In this instance, a single cascade is overstrained to accommodate all the in-class variability of the target object (Lienhart, Liang and Kuranov, 2003). The intuitive approach to deal with this problem is to divide the target training samples into subsets with each one representing a sub-pattern class (Zhang et al., 2002). The next step involves training individual classifiers for each subclass. Once all the individual sub-pattern class classifiers are trained, they are deployed at runtime by executing them in parallel with their results being merged, shown in Figure 2.2. The merged result indicate which sub-pattern class has been detected with highest probability.

This method was first proposed by Schneiderman and Kanade (2000) who created an array of parallel detectors for multi-view faces and vehicles. Their research enabled the first successful algorithm for detecting faces with an out-of-plane rotation, however the detection runtime was in the order of minutes per frame thus being too slow for real-time systems. Zhang et al. (2002) employed the same training methodology to create a multi-view face detector though they achieved real-time detection speeds through a novel arrangement of their detectors into an efficient pyramid scheme. Their system still underwent parallel executions of detectors whose results had to be merged in order to discern a classification.

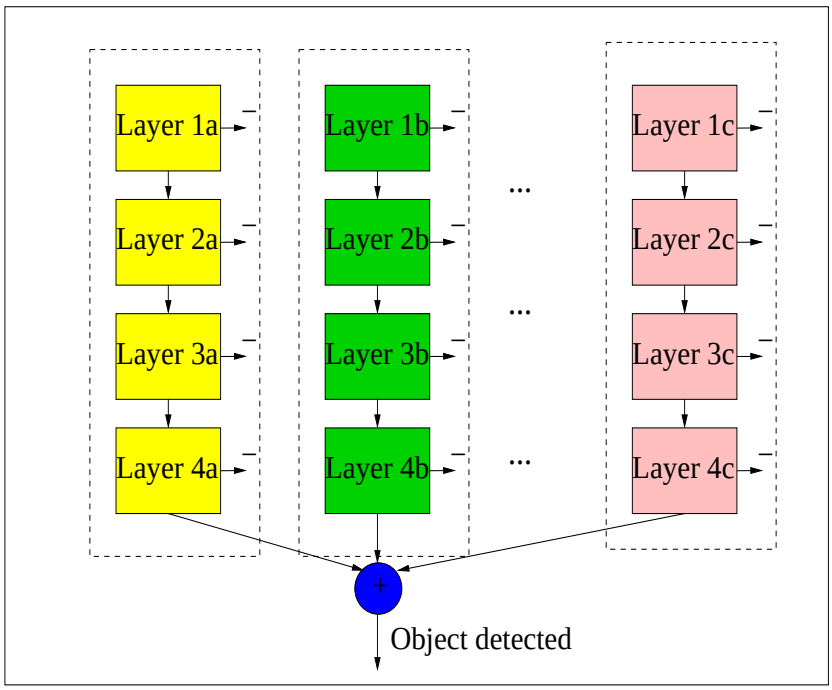

Figure 2.2: The structure for parallel cascaded classifiers experimented with by Lienhart, Liang and Kuranov (2003).

The parallel structure outperforms single cascaded systems in terms of accuracy due to the specific attention given to each sub-pattern class by training them individually. Naturally, the detection runtime is slower than that of the single cascaded structure. This is because of extra classifiers involved as well as the overhead involved in merging results from all the classifiers (Lienhart, Liang and Kuranov, 2003). Although the authors of the above systems did not mention training runtimes specifically, it can be assumed that significant complexity was encountered in having to train multiple cascades with all parameter tuning and optimization problems inherent with a single cascaded structure.

2.3.4 Tree

In order to address the problem of in-class variability for target objects, as well as the shortcomings of building a parallel cascaded classifier, Lienhart, Liang and Kuranov (2003) proposed a tree classifier training structure shown in Figure 2.3. The tree training structure produces a single classifier. At each node within the tree structure, a *k-nearest* clustering algorithm is employed to construct branches into a classifier. The branching has the effect of dividing up the sub-pattern classes of the target object into its constituent cate-

gories, and thus enabling the learning algorithm to produce more discriminant classifiers. The branching itself is efficient since it is followed through only if the resulting classifier's accuracy is improved and the detection runtime complexity does not deteriorate.

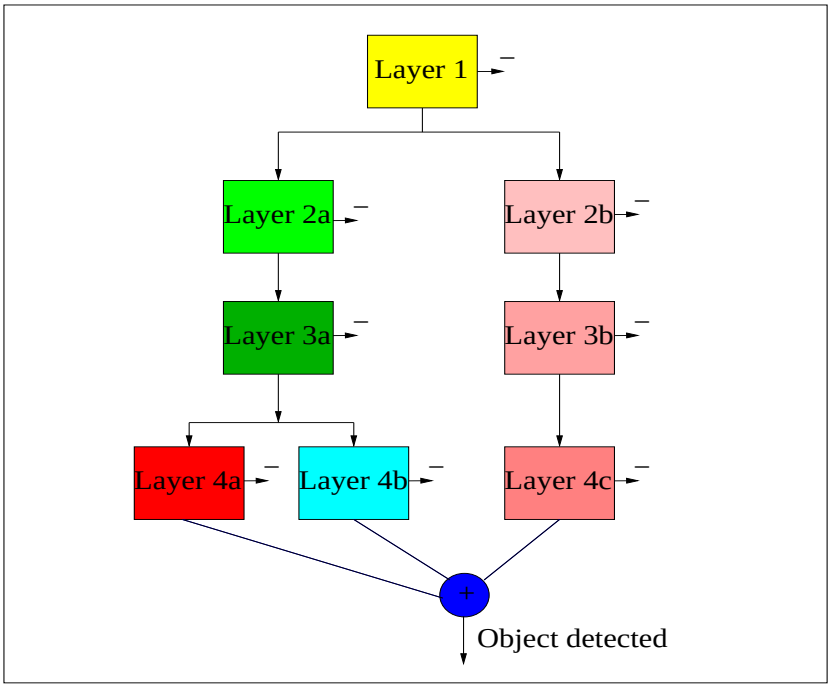

Figure 2.3: A detector tree of boosted classifiers proposed by Lienhart, Liang and Kuranov (2003).

This methodology has been shown to be twice as fast at detection runtime than two parallel classifiers, and even faster than a single classifier with accuracies either preserved or improved (Lienhart, Liang and Kuranov, 2003). The complexity involved with individually training multiple classifiers is removed as is the empirical approach of selecting sub-pattern classes during training. However, there are still disadvantages with this structure. Threshold adjustments associated with the cascaded structure remain. The root node still has to undergo training with an entire positive training sample set. This may be difficult, if not impossible for a complex dataset with large in-class variability (Barczak et al., 2008). In such a scenario, the number of weak classifiers may increase significantly (adversely affecting both the training and detection runtimes), or the threshold adjustments may result in substantially large false detection rates. Lastly, the computational overhead of the *k-nearest* clustering algorithm is also likely to increase the training runtime.

2.3.5 Soft Cascades

A soft cascade structure (Bourdev and Brandt, 2005) is a hybrid between a single layered and a cascaded structure. It makes the most of the advantages from both structures in order to achieve construction of classifiers which can be trained with more simplicity. A soft cascade structure simplifies training by requiring fewer training parameters, and by removing threshold adjustments for each layer. In spite of explicitly removing the layering of classifiers, the soft cascade classifiers still achieve real-time detection.

Bourdev and Brandt (2005) successfully demonstrated the efficacy of soft cascade classifiers. They created a soft cascade classifier by first training a classifier as a single layered monolithic ensemble. Subsequently, they augmented the classifier with rejection thresholds. The rejection thresholds assumed the behavior of a cascade by enabling rapid rejection of negative samples without evaluating an entire classifier. Thus, not only was high accuracy attained but real-time detection was also achieved.

However, Zhang and Viola (2007) point out that it is difficult to set rejection thresholds for a single layered classifier as part of post-processing. Finding optimal tradeoffs between true positive and false positive detections is not trivial. Setting rejection thresholds too conservatively increases true positive detections, but also increases false positive detections and slows down detection runtimes. The other aspect of the tradeoff enables real-time detection, but at the price of accuracy (Zhang and Viola, 2007).

2.4 Training Procedures

Training procedures can be defined as the approaches used in compiling training datasets and policies used in deciding what sequence the training data is to be *shown* to a learning algorithm. These training procedures are as important as the learning algorithm. Having a powerful learning capability without adequate training procedures and data is useless (Verschae et al., 2008). For binary classification problems, the process begins with compiling two training sample sets $\{(x_1, y_1), ...(x_n, y_n)\}$, where each sample instance x is either a positive or a negative y class label. In working with discriminating methods, it is important to have training samples that correctly define a classification boundary and are therefore statistically significant. It is especially important to use non-target object

patterns that are similar to a target object (Verschae et al., 2008). To ensure this takes place, the aim is to have as large a training set as possible in order to gain the necessary comprehensive sampling of the input space (Sung and Poggio, 1998) with respect to computational complexity and memory space requirements.

Once training datasets have been compiled, it is essential to consider and formulate a strategy which defines what order the training samples will be *shown* to a learning algorithm (Verschae et al., 2008). The simplest strategy makes all the training samples from both datasets available to a learning algorithm. In this strategy, as the learning process progresses no alteration or intervention with respect to the datasets is made. In the context of training CoBEs where each layer is designed to learn to reject 50% of the negative training samples, it means that each succeeding layer in a cascade receives 50% fewer negative training samples than the previous layer thus simplifying the learning process as it advances.

Sung and Poggio (1998) introduced what they termed a *bootstrap* strategy whereby at each iteration of training, incorrectly classified samples are added to the training set. Viola and Jones (2001*b*) used this strategy with great success together with numerous other authors in the context of CoBE training. At the end of each layer, all correctly classified negative samples are removed from training and replaced by new negative training samples which must have been misclassified by the classifier up to the last layer. This strategy makes the learning process difficult, but it facilitates in steering a classifier away from its current mistakes (Sung and Poggio, 1998). This method has longer runtimes due to the testing phase of candidate negative samples and has the property of requiring a much larger negative dataset in proportion to the positive dataset. The importance of order in which the training samples are *shown* to the learning algorithm can be seen in the context of cascade training. The core idea behind the cascaded structure was explained as the ability to build classifiers that can rapidly reject negative samples in the early layers in order to operate under real-time conditions. Hypothetically, if the negative training samples *shown* to the learning algorithm in the final layers of the classifier were to be *shown* in the first layer, then the resulting number of weak classifiers for the first layer would be enormously larger than otherwise and the classifier would be unable to perform in the real-time environment. This may occur if the designer of a detector has performed several

prior rounds of training, in which case the more difficult negative training samples would be identifiable. Under normal circumstances, an even distribution of negative training samples occurs which guarantees that the difficult negative training samples propagate to the final layers where the training is eventually forced to learn them.

Verschae et al. (2008) take the *bootstrap* strategy in cascade training one step further. They not only apply the *bootstrap* procedure between layers (external *bootstrap*), but additionally introduce the idea of an internal *bootstrap* that is affected within a layer. The external *bootstrap* is modified from the original in that an entire negative training set is replaced after each layer with a new one. This involves new negative samples which have all been misclassified up to the new layer. The internal *bootstrap* procedure becomes an iterative process of re-training a layer after it has reached the desired FPR and TPR. At each iteration of layer re-training, the negative samples which have been correctly classified are replaced with new misclassified samples. This approach to multi level *bootstrapping* is a computationally expensive strategy, but an effective way of aggressively defining more clearly the statistical boundary between positive and negative classes and therefore producing accurate classification rules. The detector produced by Verschae et al. (2008) achieved noteworthy accuracies in comparison to other state of the art detectors.

2.5 Summary

Designing a detection system is a complex task. At each step, a designer of a detector faces decision points which have significant ramifications on the chosen machine learning framework and its output. These decisions affect the ability of a chosen machine learning framework to converge to target error rates and to do so in reasonable timeframes. They also affect detection runtimes of final classifiers as well as their ability to produce effective accuracy rates.

These decisions involve the choice in the types of features used with respect to their computational overheads associated with their extraction, through to policies on how to select most optimal features from large feature spaces. The decision on which boosting algorithm to use is driven and affected by the quality and the distribution of training datasets, as well as the proportion of target objects to negative samples that the detector is likely to see at detection time. Whilst the choice of a weak learner, in conjunction

with the boosting algorithm, is likely to play a large role in determining whether the final classifier generalizes well or proceeds to under-fit or over-fit the training data.

Both training runtimes and detection runtimes of a classifier are significantly affected by the type of a training structure used to put together their weak classifiers into a final classification rule. A decision in this area can balance out and compensate for other components of a detector which incur greater computational costs for a return in better accuracies. Lastly, training procedures underpin the entire process. They need to be robust and are likely to display a tradeoff between the training phase runtime and the accuracy of the final classifier.

Due to the fact that most of the components comprising a classifier training framework overlap in their ability to affect the accuracy, training runtimes and detection time behaviors of a final classifier, it is difficult to know ahead how best to select them. Although experience in building such systems is paramount, the process is often ad hoc.

Chapter 3

Training Classifiers

This chapter examines the primary causes of protracted training runtimes for classifiers in the context of CoBE. There are a variety of contributing factors. They are:

- the size of feature spaces
- the calculation of cascade threshold adjustments
- slow convergence rates to target training errors
- the process of cascade cascade optimization
- limited incremental training capabilities

The degree to which any one of the above factors above affects the training runtimes, is domain specific. This means that, for example, a vision classification systems and audio classification systems are likely to be affected to a different extent by identical factors.

Since the seminal work of Viola and Jones (2001*b*) that introduced the first real-time face detector using a cascading structure, efforts to produce an efficient training structure have seen proposals which have targeted and modified all major components of the boosting training framework with varying degrees of success. This chapter will seek to explore the components which prolong training runtimes, as well as recent research which has attempted to find solutions to it. Lastly, the chapter will summarize the recent research proposals and compare their merits.

3.1 Feature Space Size

Pham and Cham (2007) identify the size of a feature space as being a primary bottleneck at training. In this regard, vision detection systems are more likely to suffer from large feature spaces due to the types of features they tend to employ. Often these features produce over-complete feature spaces which slow down the learning process. To demonstrate the problem, Viola and Jones (2001*b*) cited that their face detector produced 160,000 features per image and used a conservative number of around 5,000 training samples. The resulting features space, of nearly a billion features, demonstrates the magnitude of a search space as well as feature extraction overhead for every boosting round. Pham and Cham (2007) cite a runtime of up to six minutes to train a single weak classifier in the context of a feature space of this size. Considering that a viable detector requires thousands of such weak classifiers, the resulting runtimes lasting into days or even weeks (Viola and Jones, 2001*b*) come as no surprise.

In order to address computational problems posed by massive feature spaces, various authors have turned their attention to modifying the feature selection component of AdaBoost. Wu et al. (2003) significantly modify this component with notable results by proposing a filtering method that results in a training phase speed-up by two orders of magnitude whilst preserving accuracy. Their method uses forward feature selection (FFS) in which features are precomputed before each layer, meaning that this method trains weak classifiers only once per layer, as opposed to Viola and Jones (2004) who re-train for each weak classifier in a layer. In effect, FFS alters the greedy search algorithm within AdaBoost to one that is likely to be less optimal, but with large runtime reductions. Pham and Cham (2007) criticize FFS as actualizing its speed-up in part, by ignoring the weight distribution of a boosting process. They claim that features selected by FFS are not compatible with boosting, since FFS does not have the same hypothesis space as AdaBoost.

In similar vein, Verschae et al. (2008) and Baluja et al. (2004) propose to also precompute the feature space only once before the boosting process of each layer takes place. However, they go one step further by sampling it. In this manner AdaBoost still has the chance to select features from a large set, allowing a large diversity of selected features. Using this method, they observed a reduction in training times as being proportional to

the percentage of features considered at each iteration. Baluja et al. (2004) design a system providing AdaBoost with a predetermined percentage of the total feature space to search through. Depending on the percentage of a total feature space being searched through, they observed considerable speed-ups at a moderate loss in performance accuracy. For example, with a feature space being limited to 1% of an entire feature space, a speed-up of a hundred fold over a complete feature space was observed.

Wu (2008) approaches the problem of large feature spaces by tackling the computational demand required to re-calculate it at each round of boosting. They make use of the fact that in different rounds of boosting, the weight distribution of features changes, but not the feature values of a training set. This means that they do not need to be continuously computed, but can instead be cached. They sort all feature values for each of the samples in a pre-processing stage which enables them to rapidly compute thresholds, regardless of weights. They reduce the time from $O(NTlogN)$ to $O(NT)$ with a pre-calculation phase of $O(NTlogN)$ where N is the number of samples and T denotes total features. Using the caching strategy, they realize a speed-up by a factor of 15-60 and is thought to be one of the fastest boosting frameworks for face detection training using Haar-like features. The downside of their strategy is the sizable memory requirement. Pham and Cham (2007) mention that for a training sample size of N = 10,000 generating a total feature space of T = 40,000, with 2 bytes used to store an index, one would require 800MB of memory. Similarly, Pham and Cham (2007) use caching to produce a training methodology that is even faster for scenarios involving massive feature spaces that consist of local features which are linear projections of a random vector. They employ statistics to train a classifier based only on feature values and class. They report a reduced training complexity of $O(Nd^2 + T)$ with an improved accuracy over most recent detectors in the face detection field.

3.2 Cascade Threshold Adjustments

Cascade threshold adjustments, mentioned in Section 2.3.2, are a necessary component in any classifier whose arrangement consists of a layered structure. These thresholds are artificially increased at end of each layer in order to allow positive training samples to meet the overall cascade target hit rates. Without them, large numbers of positive

samples would be removed from each subsequent layer and the generalization ability of the final classifier would be compromised. This artificial adjustment is computationally expensive to evaluate as its formulation takes place after each round of boosting. Before it is adopted, all positive and negative training samples need to be evaluated against a current arrangement of weak classifiers to which the threshold adjustment is added. This produces revised hit and false detection rates which determines the threshold adjustment's performance. Its calculation becomes a significant overhead when there are large number of training samples involved, and continues to compound as the number of weak classifiers increases within a layer.

An elegant solution to computational overheads of threshold adjustments can be found within the training structure of "soft cascades", mentioned in Section 2.3.5. Authors, Bourdev and Brandt (2005), Grossmann (2004) and McCane et al. (2005), resort to training a single layered, monolithic ensemble classifier and in the process they altogether remove the need for threshold adjustments. In order to realize real-time runtime capabilities otherwise not offered by the single layered classifier, they insert rejection thresholds into a classifier allowing rapid rejection of negative samples early on in a detection phase.

3.3 Slow Learning Convergence

Classifier training commences with target training error rates being set to which the learning algorithm is required to converge. Ordinarily, this is set to a zero error rate. The more weak classifiers that the learning algorithm requires to reach this target, the longer the runtime will be. More powerful and efficient learning frameworks require fewer weak classifiers to converge to target training error rates.

Viola and Jones (2001*b*) discuss convergence issues of boosting algorithms and mention that the training process becomes more difficult particularly in latter layers of cascade training. They point out that in those layers, weak classifiers tend not to be powerful enough in their ability to discriminate positive samples from negative training samples. An observation is made that in early layers, it is possible to see error rates of 10% to 30%, whereas in latter layers this rises to 40% to 50%. This weakness in the discriminatory ability, found in latter layers, prolongs the convergence of the final layers towards their cascade target rates and ultimately delays the overall training process itself.

In order to realize faster convergence speeds, some authors have targeted modifications to the types of features used at learning. Xiao et al. (2003) and Withopf et al. (2007) attempt to reduce the number of weak classifiers by using more discriminatory features. The more powerful feature type they select is the output of previous layers, which becomes incorporated into a current layer as an additional stronger learner. By using historical information from a previous layer, Withopf et al. (2007) achieved a reduction of up to 58% in the weak classifier count, which translated to a 15% reduction in their training time. Lienhart and Maydt (2002) on the other hand, introduce a novel set of rotated Haar-like features on top of the set already used by Viola and Jones (2004), thus boosting the filter set to nearly twice the original size. Though an increase in additional rotated feature types for Lienhart and Maydt (2002) lead to a decrease in false alarm rates for a given hit rate, it didn't decrease training times, but instead prolonged them. The lengthening of their training phase can be squarely attributed to an upsurge in the size of the feature space. This demonstrated that a richer feature space with more powerful discriminants does not necessarily equate to a shortening of a training phase.

Similarly, Mita et al. (2005) employ more powerful features in the form of joint Haar-like features for their cascaded face detector. Their usage of co-occurrence of multiple Haar-like features accelerated the convergence and generated fewer weak classifiers by associating each weak classifier with multiple features. However, their system is susceptible to overfitting depending on the number of co-occurrent features they specify per weak classifier. This parameter is different depending on the problem at hand and cannot be known *a priori*. This means that training needs to be re-run in order to determine the optimal configurations. Training phase durations for a single run were not discussed in their research. Consequently it is fair to assume, that with an additional calculation of the co-occurring features, a substantial computational penalty is incurred which is not likely to contribute to a shortening of the training phase.

Also in the area of image detection, researchers Wu et al. (2004) and Whitehill and Omlin (2006) experimented with using the more powerful Gabor Wavelet features. Wu et al. (2004) report a faster convergence to a target training error rate using these feature types but fail to observe a marked improvement in classifiers' generalization over the weaker Haar-like features. However, other studies (Wu et al., 2004; Whitehill and Omlin,

2006), have shown that using more powerful Gabor features results in a prolonged feature extraction process which contributes to training runtimes as well as a detection runtimes increasing by orders of magnitude.

Other researchers have put forward modified versions of the original AdaBoost with the aim of not only achieving better accuracy, but also a faster training convergence. Lienhart, Kuranov and Pisarevsky (2003) compared three main AdaBoost algorithms, namely Discrete AdaBoost, Gentle AdaBoost and Real AdaBoost (Friedman et al., 2000) described in Chapter 2.2. They found that by using slightly more complex weak learners, Gentle AdaBoost outperformed the other two on their datasets. It not only needed fewer weak classifiers to converge to its target training error rate, but its accuracy on test sets was also better. Nevertheless, the authors do not state whether using Gentle AdaBoost also corresponded to a reduction in training runtimes.

Another variant of AdaBoost, termed FloatBoost was proposed by Li et al. (2002). It utilizes a backtracking algorithm which has the capacity of removing weak classifiers that are deemed harmful or not useful in minimizing the overall error rate. Consequently, strong classifiers with fewer weak classifiers were created in their research, but in the end their training runtime proved to be considerably longer than that of AdaBoost and its variants.

Lastly, Viola and Jones (2001*a*) introduced Asymmetric AdaBoost which altered the weighting distribution of the boosting process by placing greater weight on misclassified positive training samples. It demonstrated a greater capacity to reject false positives for a given hit rate, but there were no reports of a speed up in convergence runtimes. If anything, the proposed boosting algorithm appeared to extend the training phase since it required a larger number of cascades than normal AdaBoost to achieve its target training error rates.

3.4 Cascade Optimization

Section 2.3.2 described the cascaded training structure and listed parameters which have to be provided to the learning algorithm. Whether a resulting cascaded classifier operates optimally, is determined by these parameters whose ideal configuration is not known *a priori*. Currently, these values can only be derived from empirical testing. Such a process

requires that a classifier be re-trained for an undetermined number of times until the training is successful in converging, and a classifier can perform to a desirable speed for a given accuracy rate. The problem is that not only does the empirical approach to parameter setting fail to formulate an optimal solution, but contributes to protracted training runtimes. As mentioned earlier, a single round of training can take days or even weeks. When this process is multiplied by a number of times it takes to find a satisfactory solution, then the process becomes even more time intensive. To complicate matters, when using more difficult training datasets, the training process may stop converging at a given layer. In such cases, either the training for that particular layer can be restarted with different parameters thus preserving prior layers, or the entire cascade can be re-trained with a different combination of parameters. Either way, cascade training involves a high level of supervision and intervention which comes at a considerable cost to time resources.

Luo (2005) propose a system which is capable of optimizing a cascaded classifier once it has been trained in an arbitrary manner using any boosting methodology. It adjusts layer thresholds as part of post-processing and produces notable accuracy and performance results. This makes a modest contribution to simplifying the training process overall. Using their approach, the goal at training is reduced to only formulating a parameter combination which converges and succeeds, allowing the optimization to be carried out afterwards. However, their research does not address how layer targets at training should be set, whose values greatly influence the bootstrapping of the training data which ultimately affects the final accuracy of a classifier.

Brubaker et al. (2006) and also Brubaker et al. (2008), in their later and improved version regarding face detectors, propose a more robust solution to the problem of cascade optimization by devising an optimization function. In their research, an optimization function adaptively chooses target rates for individual layers based on user requirements for the final cascade true and false positive rates. The end result was a face detector which performed comparably against best current detectors. However, it is difficult to know how much of this can be attributed to the cascade optimization component since their training framework was modified significantly in other key components as well. Nonetheless, a viable approach to removing empirical cascade layer parameter setting has been proposed. Similarly, Sun et al. (2004) developed a framework for analyzing the impact of learning

a single layer in a cascade on an entire cascaded classifier through a novel cost function, which they claim yields an optimal learning goal for each cascade layer. In their system, however, it is not clear how to set parameters to optimize for detection runtime speed for a given detection rate.

An altogether different approach to finding a solution to the cascade optimization problems, is to remove them altogether by resorting to "soft cascades". The "soft cascades" achieve an ease of training by making the cascade parameters redundant and in conjunction with removing the threshold adjustments as mentioned earlier. In this framework, the classifier is guaranteed to have to undergo training only once and thereafter, an algorithm is employed to build into the monolithic ensemble classifier a quasi cascade-like behavior. Bourdev and Brandt (2005) successfully augmented such a monolithic ensemble classifier with rejection thresholds that can be calibrated for a specific detection rate and execution runtime. Their architecture effectively decouples the speed/accuracy tradeoff associated with cascade training. Once a classifier is trained, it is able to be rapidly adjusted for any point on the receiver operating characteristics (ROC) graph surface without iterative re-training.

3.5 Catastrophic Interference

Catastrophic interference, or forgetting (McCloskey and Cohen, 1989), is a term applied to a phenomenon in which all previous learning encapsulated within a classification rule is lost. This occurs when important additional datasets become available and need to be integrated after a classifier has been trained. This typically results in it having to be re-trained using a combination of previous and new data (Gangardiwala and Polikar, 2005). The process of re-training becomes unfeasible if the training data becomes massive, but also time consuming with no opportunity available for consolidating on previous classifier training successes.

Incremental learning presents a solution to catastrophic interference. Incremental learning, as defined by Gangardiwala and Polikar (2005), involves the capability of training a classifier without losing previously learned knowledge and doing this while acquiring novel knowledge on consecutive datasets, without requiring access to previously used data. In the field of multi-class classification they state also that this involves an ability to learn

new classes introduced by subsequent datasets as well.

However, there exists limited research on incremental learning using CoBEs. Oza and Russell (2001) introduced a version of on-line incremental AdaBoost which allows each weak classifier in an existing *base classifier* to be re-learned as new training samples are made available, one at a time. Authors like Roth et al. (2005) and Huang et al. (2007) have employed a similar on-line approach to incremental learning using AdaBoost and Real AdaBoost. In this approach individual weights of each weak classifier belonging to an original base classifier, are updated and modified using various approaches as the system is fed each training sample. A drawback of this approach is that its discriminative ability may be limited since the total number of weak classifiers remains fixed. Gangardiwala and Polikar (2005), on the other hand, propose an incremental learning approach for training boosted ensembles which generates additional weak classifiers. Their system also works in a multi-classification domain, however it is implemented through neural networks instead of decision trees.

3.6 Discussion

Each reviewed approach has made a contribution to the reduction in training runtime phases in instances where large training samples and massive feature spaces are involved, but none have provided a general framework that solves it outright and thus it remains an open problem. The most promising proposals come from Pham and Cham (2007) where caching and the simplification of the feature space is achieved through employment of statistics on the feature values and their classes, but this methodology may suffer in that it is not transferable to all feature types encountered in the classification domain. Other proposals which modify the feature selection component of AdaBoost, like feature filtering methods, are unreliable and ignore the distribution weights of boosting and thus undermine its theoretical foundations.

Stronger features have been found to be useful in producing weak classifiers in latter, more difficult stages of training. They have been observed to lower training errors more rapidly in those stages and create weak classifiers which generalize well, but can be prone to overfitting the data. Despite their contribution, they do not address the fundamental problems of protracted training times. In most cases, the more powerful and

discriminatory features are more computationally intensive thus further compounding the problem.

The emergence of a number of variant boosting algorithms has also not made a significant advancement in the reduction of training runtimes. Boosting algorithm modifications have mainly contributed to improvements in the accuracy of specific cases. GentleBoost has been found to be effective when used on datasets containing noisy data and outliers, while Asymmetric Boost was shown to be valuable in instances where a greater emphasis and weight needed to be placed on positive detections. Moreover, RealBoost has been found useful in scenarios where training sets proved to be difficult, and maximum information regarding training data needed to be retained.

Cascade optimization techniques have achieved some improvement and simplification in the training process of CoBEs. However, they still do not solve the problem of computational complexity inherent within the framework. In addition to that, some solutions do not automate all tunable parameters and leave behind others to iterative re-training. The soft cascaded approach at best preserves the runtime efficiency of the cascaded classifiers and removes artificial layer threshold adjustments, but retains the slowness associated with training monolithic ensemble classifiers. Meanwhile, cost functions designed to find optimal cascade parameters have been described as not having the ability to translate well into practical algorithms because they require assumptions or approximations that may not always hold (Masnadi-Shirazi and Vasconcelos, 2007).

Incremental learning in the context of CoBE is still an open question. Current approaches attempt to update and modify existing weak classifiers to force them to fit new incoming data, but do not expand an existing ensemble with additional weak classifiers. This has the potential of limiting the discriminatory ability of weak classifiers when new features are unable to be integrated for samples which have previously not been *seen*.

Chapter 4

Accelerated Classifier Training

This chapter presents the main contribution component of this research which addresses the limitations of current classifier training frameworks. Three alternative training frameworks are presented here. The first two frameworks focus exclusively on creating alternative approaches for training CoBE classifiers. The final framework describes a novel approach for incrementally training CoBE classifiers.

4.1 The PSL Structure

Introduced by Barczak et al. (2008), the Parallel Strong classifier within the same Layer (PSL) structure, primarily aims to address problems inherent in the CoBE structure. These problems are:

1. its inability to maintain 100% hit rates on positive training samples as cascade layer numbers increase, thus compromising generalization

2. the artificial adjustment of cascade layer thresholds which change the statistical margin that may consequently no longer represent classifiers' confidence (Barczak et al., 2008)

3. the framework's inability to succeed in training the initial first layer when presented with difficult datasets

4. the ad hoc nature of cascade optimization involving the tuning of each layer's positive hit rates

The PSL structure explicitly addresses the above issues and as a byproduct, it also achieves a significant reduction in runtimes of the training phase.

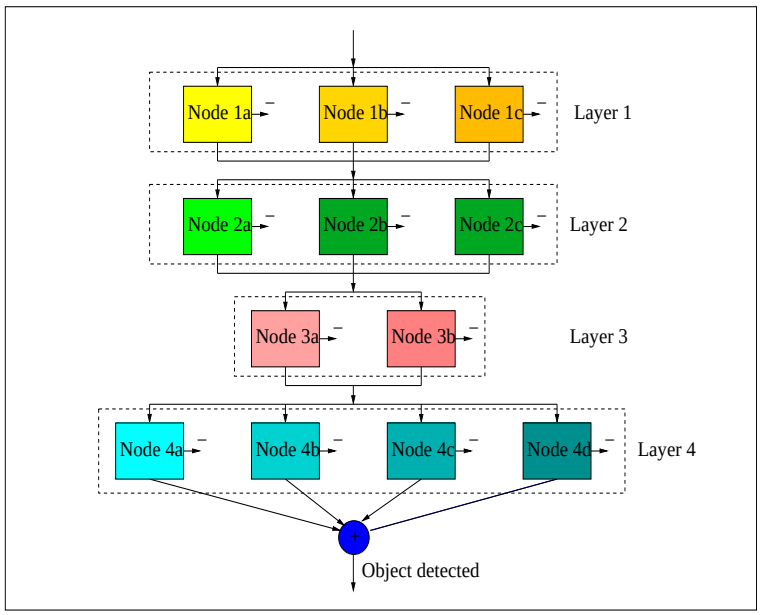

Figure 4.1: Structure illustrating parallel strong classifiers within the same layer (PSL).

The PSL structure shown in Figure 4.1 is a quasi cascading structure which fundamentally transforms intra-layer classifier training to the same degree that the Viola-Jones (Viola and Jones, 2001*b*) cascading structure transformed the monolithic ensemble structure. The PSL framework introduces a form of horizontal cascading within each layer which complements the standard vertical cascading structure. Just as independent AdaBoost rounds are executed by the Viola and Jones (2004) method for every new layer, the PSL structure provides a methodology for executing multiple, independent AdaBoost rounds (*nodes*) within the same layer. The construction of a PSL classifier can be seen in Algorithm 3.

The workings of a standard cascading framework was described in Section 2.3.2. It was shown there how a standard cascading framework operates at each layer of a cascaded classifier by executing a single independent round of AdaBoost until a layer's false alarm and hit rates have been met. The PSL framework decomposes layer targets into smaller and more achievable intermediate steps. It does this by specifying a *criterion condition* for each intra-layer AdaBoost node. The criterion condition places a limit on the number of boosting iterations each node can perform. Once a criterion condition is satisfied, the

Input : positive and negative training samples, layer targets
Output: PSL classifier
Data: P = positives, N = negatives, TS = training set, C = current, refers to misclassified samples up to that point of a classifier, cls = cascaded classifier, node = a PSL node of weak classifiers.
BoostNewPSLNode() initiate the boosting of a new PSL node until the criterion condition is met.
AppendNode() add the new PSL node to the layer.
TestNode() test the PSL node for a layer termination condition.
RemCorrectClassifNegatives() remove negative training samples that have been correctly classified.
RemCorrectClassifPositives() remove positive training samples that have been correctly classified.

```
1  CPTS = PTS;
2  CNTS = NTS;
3  foreach layer l in the cascade of max L layers do
4  |  node = BoostNewPSLNode(l,CPTS,CNTS);
5  |  AppendNode(l,cls,node);
6  |  targetsSatisfied = TestNode(l,cls,node,layerTargets);
7  |  if targetsSatisfied then
8  |  |  CNTS = RemCorrectClassifNegatives (l,cls,CNTS);
9  |  |  CPTS = PTS        /* set curr pos.  to all training pos.  */
10 |  else
11 |  |  CPTS = RemCorrectClassifPositives (l,cls,CPTS);
12 |  |  l = l − 1          /* create new PSL node for the same layer */
13 |  end
14 end
```

Algorithm 3: PSL classifier training structure

boosting process ceases, and a new AdaBoost node is started with weight distributions for all samples reset to an even spread.

A criterion for ending an AdaBoost node and beginning a new one can be application specific and may take a number of different forms. It may simply be a maximum number of weak classifiers that are permitted per node, as was used in this research; it can be a maximum percentage of negative training samples that must be rejected per node or a minimum percentage of positive training samples that must be accepted per node. The criteria can also be progressive and adaptive for the nodes within a layer. Depending on the criterion used, there is likely to be a tradeoff between training and detection runtimes which must be considered.

Once a node's criterion condition is reached and boosting is terminated, only positive training samples that have been misclassified till then are forwarded to the next node.

This process of starting new AdaBoost nodes with reduced positive training sample sets continues from node to node until all positive training samples have been correctly classified by a layer. The retention of high hit rates in each layer has been seen as crucial to maintaining a strong generalization ability of CoBEs, because once a positive sample is misclassified, it cannot be corrected by subsequent layers (Xiao et al., 2007). The same authors have claimed that the most intuitive way to lessen the degradation of generalization caused by misclassified positive training samples in early layers, is to use massive positive training sample sets (Xiao et al., 2007). However, being aware of the computational complexity and memory demands of this approach, they proposed to parallelize the processing in conjunction with sampling the data sets. They achieved comparable results to state of the art detectors, but the practicality of such a system remains questionable.

Correspondingly, once a criterion condition is met, 100% of the negative training samples are forwarded to subsequent nodes within the same layer. The negative training set remains constant throughout intra-layer training while the positive training set diminishes from one node to the next. A negative training sample is considered to be rejected by a layer only if a unanimous decision to reject it has been reached amongst all nodes within that layer. When a layer's target false alarm rate has been reached, all misclassified training negatives become the negative training samples for the next layer.

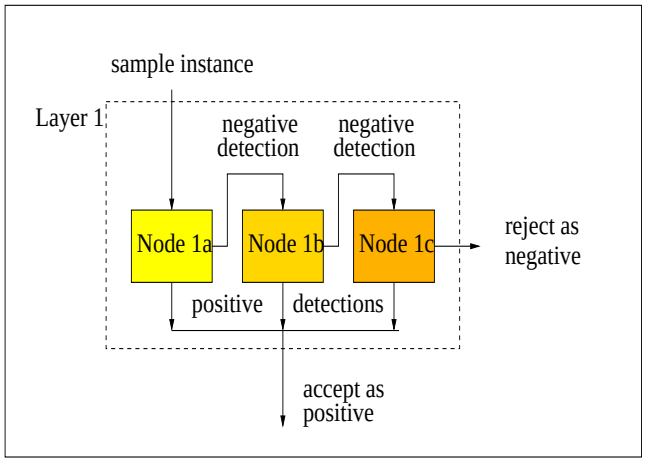

Figure 4.2: The flow of a test sample through the PSL structure during the detection phase illustrating the process of a positive detection and a rejection.

During a detection phase as seen in Figure 4.2, if a sample receives a positive vote from any one node within a layer, it is classed as a positive detection and immediately

forwarded to a next layer for evaluation without calculating the remaining nodes in that layer. On the other hand, in order for a sample to be classified as a negative sample and therefore require no further evaluation, it must obtain a unanimous decision by all nodes in the same layer that it is a negative.

The PSL approach of generating independent nodes per layer lends itself well to post-processing optimization of classifiers for detection runtimes. This can be realized if a detector's designer understands well the environment in which a detector is to function. As discussed in Section 2.3, rare event detection presents a significant challenge in real-time computer vision analysis. A PSL classifier can be optimized in such a way that it favors instances where an asymmetric data distribution is expected, or it can be calibrated to perform optimally on more even data distributions. Such a calibration can be performed by rearranging the PSL nodes within each layer.

Figures 4.3 and 4.4 show how negative sample rejection rates together with positive sample detection rates vary from one node to another. Using this variance, classifier nodes can be rearranged to favor a rare event detection environment. This is possible by ordering nodes which have the highest false detection rates first to last within a layer. Ordering in this way raises the probability that samples which are destined to be falsely detected by a layer, can be evaluated and passed to a next layer earlier, thus eliminating the need to compute the remaining nodes. In instances where a data distribution is more even, the balancing between true positive and true negative detection rates can be realized by arranging nodes in a manner in which precedence is given to nodes that have higher positive detection rates.

4.1.1 PSL Discussion

The PSL structure simplifies classifier training by eliminating the tunable parameter for positive hit rates for each layer. However, it does introduce an extra parameter for the node termination criterion which has the potential to affect runtime performances of a detector as well as its training duration.

Another issue associated with the node termination criteria is that there is likely to exist a variable and an unpredictable rejection rate between nodes in a layer. This means that the PSL structure has no control as to what negative samples will be rejected by each

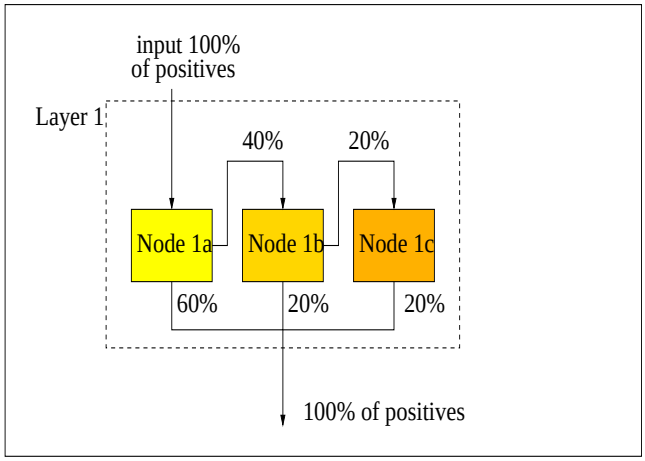

Figure 4.3: The flow of positive training samples through the PSL structure during the training phase.

node. Difficulties may arise in attaining layer false alarm rates if a boosting algorithm concentrates too much on rejecting negative samples which have already been misclassified by previous nodes and have therefore no chance of being rejected by a layer as a whole. A solution to this is presented in the succeeding section.

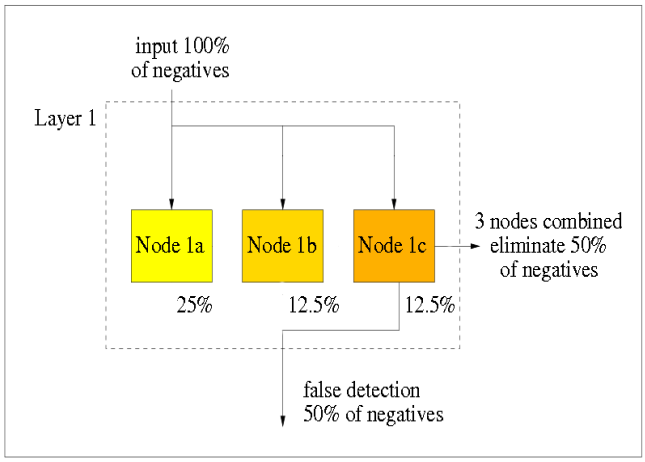

Figure 4.4: The flow of negative training samples through the PSL structure during the training phase.

4.2 Variant PSL Structure

In this section, a variant form of the PSL training structure is presented with the aim of realizing further reductions in training runtimes without compromising accuracy. This modification concerns limiting the size of a negative training sample set passed to each node within a layer. In Figure 4.4, it can be seen how a standard PSL approach allows each AdaBoost node in a layer to see 100% of negative training samples, irrespective of previous nodes' ability to correctly classify them. The modification to the PSL structure proposes that only correctly classified negative training samples from previous nodes be passed to subsequent nodes. Figure 4.5 shows this diagrammatically, while Algorithm 4 portrays the modification in more detail.

Input : positive and negative training samples, layer targets
Output: PSL classifier
Data: P = positives, N = negatives, TS = training set, C = current, refers to misclassified samples up to that point of a classifier, cls = cascaded classifier, node = a PSL node of weak classifiers.
BoostNewPSLNode() initiate the boosting of a new PSL node until the criterion condition is met.
AppendNode() add the new PSL node to the layer.
TestNode() test the PSL node for a layer termination condition.
RemCorrectClassifNegatives() remove negative training samples that have been correctly classified.
RemCorrectClassifPositives() remove positive training samples that have been correctly classified.

```
1  CPTS = PTS
2  CNTS = NTS
3  foreach layer l in the cascade of max L layers do
4  |  node = BoostNewPSLNode(l,CPTS,CNTS)
5  |  AppendNode(l,cls,node)
6  |  targetsSatisfied = TestNode(l,cls,node,layerTargets)
7  |  if targetsSatisfied then
8  |  |  CNTS = RemCorrectClassifNegatives (l,cls,CNTS)
9  |  |  CPTS = PTS        /* set curr pos.  to all training pos.  */
10 |  else
11 |  |  CPTS = RemCorrectClassifPositives (l,cls,CPTS)
12 |  |  CNTS = RemCorrectClassifNegatives (l,cls,CNTS)
   |  |         /* rem misclassified neg samples from training set */
13 |  |  l = l − 1          /* create new PSL node for the same layer */
14 |  end
15 end
```

Algorithm 4: Variant form of the PSL classifier training structure

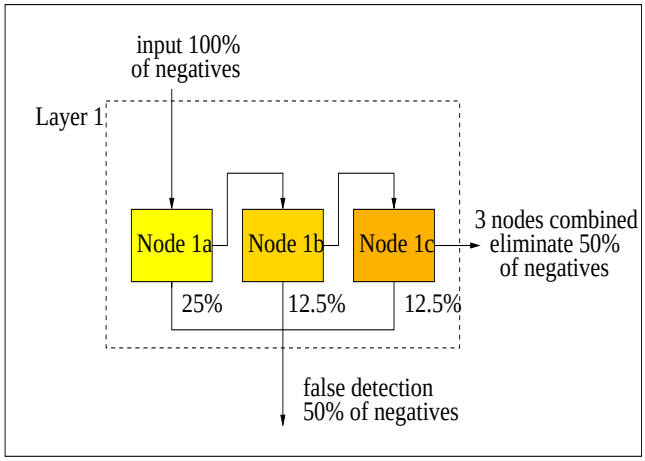

Figure 4.5: The modified flow of the negative training samples through the variant form of the PSL structure during the training phase.

By making available to each subsequent node negative training samples which have been misclassified by previous nodes, the process of learning to reject negative training samples that have until that point been correctly classified is made more difficult for the boosting algorithm. This means that a boosting algorithm will continue to concentrate on learning to reject negative training samples which a layer as a whole can no longer reject, since one or more previous nodes will misclassify them as positive. The result is that those negative training samples which have been correctly classified until that point may not get sufficient attention by subsequent nodes and may consequently be misclassified by them. The end result are higher false detection rates for a given layer.

In summary, the modification aim to simplify training by:

1. making a training set smaller for each subsequent node in a cascade layer

2. allowing a boosting algorithm to focus only on negative training samples which still have a possibility of being rejected by a layer, thus accelerating the convergence of layers' target false detection rates

3. creating fewer weak classifiers, since a boosting algorithm gains more control over which samples are rejected by each node

4.2.1 Variant PSL Discussion

By removing negative training samples from each subsequent node thus making the overall training set more limited, there is concern that the generalization of a final classifier might be compromised. However, this can be solved by employing the Viola and Jones (2001*b*) *bootstrapping* method if a training process possesses a large number of negative training samples. Each negative training sample that is removed from subsequent nodes can be replaced by a new negative sample for the next round. The only criterion which a new negative sample would have to meet, is that it is correctly rejected by all nodes in that current layer.

The variant PSL structure provides guarantees during training that ensure a requested percentage of negative training samples is rejected at each layer which the original PSL structure is not able to. When the original PSL structure completes training a layer, it does so when a single node achieves a specified hit rate and a minimum required false detection rate. However, a false detection rate for a node may not be the same as the false detection rate for the whole layer. It is likely that a false detection rate for an entire layer is considerably higher than that of individual nodes because various nodes might have learned to reject different subsets of negative training samples. Due to the fact that only negative samples that achieve a unanimous vote from all nodes are rejected, the consequence is that only the *intersection* of every negative subset rejected by all nodes will be correctly classified by a layer. The variant PSL structure on the other hand ensures that whatever false detection rate last node in a layer achieves, will be the false detection rate for an entire layer.

4.3 Incremental Learning with IPSL

Section 3.5 highlighted current shortcomings of the boosting paradigm in providing capabilities to train classifiers incrementally. This means that classifiers which have been previously trained on a given dataset must be entirely re-trained if new data becomes available which is deemed as necessary to incorporate into an existing classification rule. For classification domains where new and unique training data is made available in regular intervals, the consequences are that all previous efforts invested in training, tuning and

optimization are wasted with there being no way of capitalizing on previously successful training results. Also, since current boosting structures do not cope well with training classifiers on massive datasets due to memory and processing constraints, it means that classifiers cannot be trained incrementally on smaller partitions of datasets. Instead, the size of a dateset must be sampled and capped in line with hardware capabilities of a training environment. This section presents a novel approach to incrementally train classifiers of boosted ensembles using the IPSL framework.

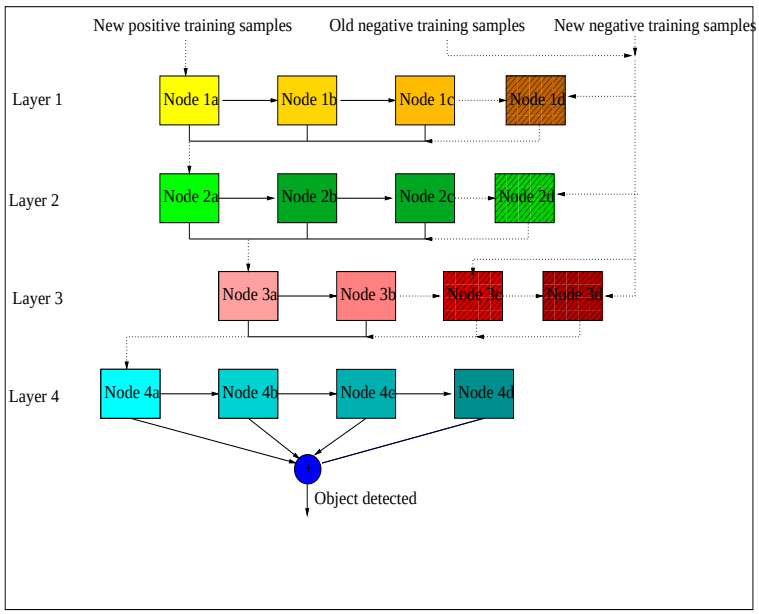

Figure 4.6: The incremental PSL (IPSL) training structure illustrating the creation of additional nodes to the existing *base classifier*.

Figure 4.6 illustrates the approach used by IPSL to build classifiers incrementally. IPSL uses the same methodology as PSL in its attempt to train cascade layers that produce 100% hit rates on positive training data. To achieve this, the IPSL structure takes as input an existing classifier (*base classifier*), together with a new additional positive training dataset. IPSL then evaluates the additional positive training samples against each layer of a *base classifier* and if any new positive training samples fail to pass a layer, a new node is trained on those misclassified positive samples. The new node is subsequently appended to the same layer of a base classifier. The misclassified positive training samples are trained against a combination of new negative training samples and all the old negative training samples used in prior training for that particular layer. Algorithm 5 shows the

details of IPSL.

The generation of new nodes for each layer continues until all positive training samples have been classified correctly and once a layer has learned to reject at least 50% of the total negative training samples. The latter is a modification to the standard PSL requirement in regards to rejecting negative training samples. The standard PSL permitted a layer to finish once a single node learned to reject 50% of the negative training samples. However, this did not necessarily translate to a 50% rejection of all negative training samples for an entire layer. It seemed necessary to create stricter constraints for rejecting negative training samples per layer, since the prospect of additional nodes gave the impression that overall false detection rates might increase rapidly. This was anticipated due to the fact that existing nodes of a base classifier had not been trained on new negative training samples. This means that any new negative training samples that did not attain a unanimous negative detection vote from existing nodes had no chance of being rejected by a layer through new nodes. Because of this, new nodes had to have the ability to preserve false alarm rates of base classifiers and therefore needed to have the old negative training samples available to them. On the other hand, all the positive training samples that were utilized in previous training rounds do not need to be used in subsequent incremental training since base classifiers' nodes would already have been trained to classify them as positives.

The characteristic of a PSL classifier which requires a unanimous negative detection vote by all nodes in a layer in order for a sample to be rejected by that layer outright, presents considerable difficulties and a limitation to the IPSL structure. The challenge for the IPSL structure arises in regards to producing a false detection rate which is lower than that of a base classifier. This issue comes up if the decision boundary of a base classifier has not been sufficiently defined in its training to classify correctly new negative training samples. In such a scenario, an IPSL classifier will be not be able to learn to reject new negative training samples even though subsequent new nodes appended to the base classifier have been trained to correctly reject the additional negative training samples. The IPSL structure in its present form appears to theoretically provide a mechanism, which at best preserves false detection rates of a base classifier whist improving its positive detection rates. The IPSL structure must therefore rely heavily on effective training of a

base classifier, particularly in regards to the usage of large and diverse negative datasets. Ideally, such datasets needs to fully encompass the decision boundary of the negatives, allowing additional IPSL positive training to only provide a refinement of the actual decision boundary.

Input : PSL classifier
Output: IPSL classifier
Data: MN = misclassified negatives, MP = misclassified positives.
BoostNewPSLNode() initiate the boosting of a new PSL node until the criterion condition is met.
AppendNode() add the new IPSL node to the layer.
ResetPositiveSet() re-initialize the positive training set all the positive samples.
PopulateCurrMPSet() fill the positive training set with positive samples that have been misclassified.
PopulateCurrMNSet() fill the negative training set with negative samples that have been misclassified.

```
1  foreach layer in the cascade do
2      totalMP = PopulateCurrMPSet(currMPSet);
3      if totalMP > 0 then
4          PopulateCurrMNSet(currMNSet);
5          while totalMP > 0 and !layerTargetMet and totalPSLNodes < MAX do
6              layerTargetsMet = BoostNewPSLNode(currMNSet,currMPSet);
7              if layerTargetsMet then
8                  currMNSet = ResetPositiveSet();
9              else
10                 totalMP = PopulateCurrMPSet(currMPSet);
11                 PopulateCurrMNSet(currMNSet);
12             end
13         end
14     end
15 end
```

Algorithm 5: IPSL learning

4.3.1 IPSL Discussion

When considering IPSL's approach to incremental training which involves continually appending new nodes to existing base classifiers, questions of scalability arise. Questions like, how many rounds of incremental learning can the IPSL structure perform before the number of nodes becomes too cumbersome to execute in real-time? Also, unlike the positive training samples, if all previously used negative training samples need to be used in every subsequent training run, how many rounds of incremental training can be performed before the negative training set becomes computationally too expensive?

A solution to the first issue might lie with an extension to the system that permits a total number of nodes to be capped. Such a system might have the capacity to replace existing nodes with new ones if a newer node displays higher accuracy rates than an older one belonging to a base classifier.

By limiting the size of a negative training set through careful sampling, the prospect of it becoming too massive for usage can be addressed. As long as a rich distribution of negative values, which accurately define the boundary between it and the positive samples is retained, then the generalization ought to be preserved despite its reduced size. An accurate distribution of negative values can be determined by evaluating candidate negative training samples against an existing base classifier. A diverse distribution can be obtained by selecting negative samples which are correctly classified and rejected by a base classifier at every layer in its cascade. By choosing a mixture of negative samples which are classified correctly by initial and final layers of a base classifier, a dataset which more accurately represents a decision boundary can be compiled.

4.4 Summary

The PSL framework proposes a solution to several problems encountered in training CoBEs and is promising in its theoretical ability to shorten overall classifier training runtimes. It offers additional tractability to the classifier training process over and above what is already offered by the standard cascaded approach. The framework guarantees that all positive samples are retained throughout a cascade, ensuring that the learning algorithm's generalization ability does not degenerate as cascade layer numbers increase. The arbitrary artificial threshold adjustments present in training of CoBEs at the end of each layer are removed, thus preserving classifiers' confidence. Lastly, the cascade optimization process is simplified by the removal of layer hit rate targets from training.

The PSL structure is notably different from other common structures such as the standard cascading or the tree structures. While PSL is a cascading structure of sorts, it differs from the standard cascaded version of Viola and Jones (2004) by allowing multiple independent boosting rounds per layer. It is also unlike the tree training structure in that it employs no clustering algorithms to split the positive training sample sets, but instead uses detection results from previous rounds produced by boosting to achieve the splits.

Whereas the tree structure possesses numerous decision branching paths from root nodes to leaf nodes, the PSL structure has only one and thus is still only a degenerate tree.

The extended PSL structure has the potential to add more robustness to the original PSL structure. It provides a methodology which steers the learning algorithm to focus more on training samples which are critical to layer targets being met. More control over individual nodes' accuracy performances is realized by minimizing the variance likely to be seen in the original PSL structure.

A novel approach to training CoBEs incrementally using the IPSL structure was put forward. It has sought to fill the gap that currently exists in the capabilities of CoBE structures. The primary limitation of existing CoBE structures is their inability to improve and build further on existing base classifiers when new training data is made available, without having to re-train classifiers from beginning. The IPSL structure is an intuitive extension of the PSL structure which achieves incremental training by appending additional nodes to base classifiers. The additional nodes undergo training with datasets that comprise of a combination of new positive and new negative training samples as well as the base classifier's negative training samples.

Chapter 5

Evaluating the PSL Structure

Chapter 4 introduced the PSL training structure as a solution to some of the problems that lead to prolonged training runtimes when constructing CoBE classifiers. It also proposed a novel IPSL framework for incrementally training CoBE classifiers. This chapter will describe methods undertaken to evaluate the efficacy of the PSL structure to train robust classifiers with reduced training runtimes as well as experiments which explored the ability of the IPSL structure to generate incrementally trained classifiers that improve the generalization ability of their respective base classifiers.

This chapter will be divided into sections that will describe the following aspects of the methodology:

- the types of training structures that the PSL structure was compared to
- the technical specifications regarding code and hardware used in training
- procedures used for training various types of classifiers
- dataset specifications
- the various measurements used for evaluating and comparing different classifiers

Four main experiments were conducted. The first two experiments dealt exclusively with the effectiveness of the PSL classifiers, while the third examined the plausibility of a variant form of a PSL training structure. The final experiment considered the efficacy of the IPSL training structure.

5.1 Comparative Experiments

In order to gauge the effectiveness of classifiers trained using the PSL structure, classifiers belonging to other structures were also trained and were compared to it. It was decided to compare PSL classifiers against those of the standard monolithic ensemble structure, comprising of a single layer as in the style of Freund and Schapire (1999) as well as those of the cascaded structure of Viola and Jones (2001*b*). The details of both structures are found in Section 2.3.

Classifiers of the monolithic ensemble structure served as a good comparison against PSL classifiers for two reasons. The monolithic ensemble structure classifiers were expected to produce the highest accuracy rates, particularly on hit rates since cascades generally lead to the lowering of them as layer numbers increase. High accuracy rates were expected to set a strong benchmark against which the accuracy of PSL classifiers was to be tested. Secondly, the monolithic ensemble structure was expected to be slowest in its convergence runtime during training. The intent was to place its high hit rates into context, while contrasting them against the training runtimes for the PSL structure. The expectation was that the tradeoff between high hit rates and training times together with real-time detection capabilities would become apparent.

Standard cascaded classifiers were chosen for comparison purposes against PSL classifiers because they provide real-time detection capabilities and because they have become a widespread and popular approach employed in training boosted ensembles (Brubaker et al., 2008). The cascaded structure was also expected to best highlight the difficulties associated with training classifiers of boosted ensembles and was thought to provide training runtimes that would be most comparable to those of PSL classifiers.

In order to evaluate IPSL classifiers, it was seen as more important to observe and compare the effects on the generalization ability of IPSL classifiers when trained with datasets of different sizes. The decision was made to train a *base classifier* using the standard PSL structure and thereafter, conduct three separate rounds of incremental training rounds on top of this initial base classifier. The three subsequent rounds of incremental training were carried out using different datasets which consisted of previously *unseen* samples. The three datasets differed in their sizes. The smallest, comprised of samples which totaled 10% in size relative to the size of datasets used to train a base

classifier. The other two totaled 20% and 30% in size relative to datasets used to train base classifiers.

The objective was to scrutinize the final IPSL classifiers for improvements in accuracy over their base classifiers and to ascertain the effects of datasets of different sizes on IPSL. Although a large component of testing involved comparisons between results of different IPSL classifiers, it was decided to also create parallel cascaded classifiers which closely simulate incremental learning since they do not require re-training and thus provide more objective comparisons.

The summary of the four main experiments are as follows:

1. **Preliminary experiments:** compare PSL classifiers to those of the monolithic ensemble and standard cascaded structures.

2. **Primary experiments:** use a larger dataset in order to produce more robust comparisons of PSL classifiers to those of the monolithic ensemble and standard cascaded structures.

3. **PSL vs the variant PSL structure:** compare classifiers of the original and the variant PSL structures introduced in Section 4.2.

4. **IPSL experiments:** compare the generalization abilities of IPSL classifiers trained on a base classifier using datasets of different sizes and compare them also against parallel cascaded classifiers trained using the same datasets.

5.2 Technical Specifications

The classifiers were trained using programs written in C/C++. The programs were an extension of work done by Barczak et al. (2008). Three programs were written for the first two experiments. They included a program for training PSL classifiers, monolithic ensemble classifiers and the cascaded classifiers. An additional program was written for the third experiment which involved training classifiers of the variant PSL structure. The final experiment required further two programs to be written; one for IPSL classifier training and the other for training parallel cascaded classifiers.

The output of the training programs were classifiers in a *.dat* file as well as training runtimes in seconds. All classifiers were trained in identical environments on a Dual Core

AMD Athlon 1.7GHz machine with 2MB RAM memory. Following the creation of each batch of classifiers, they were verified against positive and negative training datasets in order to ensure that the target training error rates were achieved.

Additional C/C++ programs were written to evaluate classifiers. A separate program was required for testing parallel cascaded classifiers. These testing programs were also specifically suited for one-vs-all modes of testing. The testing programs provided all necessary statistics as well as accuracy measurements. The same programs generated data used for the ROC graph curve analysis that enabled classifier optimization to take place.

The dataset preparation required the creation of numerous datasets for both training and testing purposes and this was automated using Python scripts. One set of training data was created for each classifier. This meant that every classifier was trained only once. The policy of creating multiple training datasets for same classifiers, which would have allow averaging of their final accuracies, was not pursued.

5.3 Datasets

The datasets for experiments were obtained from the machine learning repository of the University of California, Irvine (UCI). Instead of consisting of low level data that had undergone only minimal processing, the datasets comprised samples represented by feature values. These feature values were pre-calculated and extracted using various methods. The removal of a feature extraction process in these experiments allowed greater focus on machine learning frameworks.

5.3.1 UCI PenDigit and OptDigit Datasets

The PenDigit and OptDigit datasets (Asuncion and Newman, 2007) were used in the initial experimentation with the PSL structure. The overall characteristics for both datasets can be seen in Table 5.1. Both datasets represent handwritten digit features and therefore comprise 10 classes each, with one class for every digit. The details of individual digit classes for both datasets are shown in Table 5.2.

The PenDigit dataset was created by collecting 250 samples from 44 writers. Each writer was required to write 250 random digits on a LCD tablet device. The device sent

x and y tablet coordinates which represented sample features. Similarly, the OptDigit dataset involved 43 writers that contributed to its formation. In this case, the digits were written on a pre-printed form and extracted as normalized bitmaps using preprocessing programs. The 32x32 bitmaps were divided into non-overlapping blocks of 4x4 with the number of pixels counted in each block. This generated an input matrix of 8x8 where each element was an integer in the range 0-16.

Table 5.1: General details of UCI OptDigit, UCI PenDigit and UCI Letter datasets used for the experiments.

	UCI OptDigit	UCI PenDigit	UCI Letter
Classes	10	10	26
Test set	1797	3498	4000
Training set	3823	7494	16000
Features	64	16	16
Feature value range	0..16	0..100	0..15

Table 5.2: Details of the UCI OptDigit and PenDigit datasets featuring characteristics per class label.

Class	OptDigit training	OptDigit test	PenDigit training	PenDigit test
0	376	178	780	363
1	389	182	779	364
2	380	177	780	364
3	389	183	719	336
4	387	181	780	364
5	376	182	720	335
6	377	181	720	336
7	387	179	778	364
8	380	174	719	336
9	382	180	719	336

5.3.2 UCI Letter Dataset

The UCI Letter dataset (Asuncion and Newman, 2007) was used for all remaining experiments. This dataset comprises 20000 alphabet character samples in total. The characters are in uppercase and in 20 different fonts. The fonts were represented in a range of character types including script, italic, serif, as well as Gothic, while also being randomly distorted. Each image is represented by 16 integer features which were extracted using statistical moments and edge counts. The overall characteristics of this datasets can be seen in Table 5.1, while Table 5.3 presents details of each training dataset with respect to individual character classes.

Table 5.3: Details of the UCI Letter dataset featuring characteristics per class label.

Class	Positive Training	Negative Training	Positive Test	Negative Test
A	631	15358	158	3853
B	612	15377	154	3857
C	588	15401	148	3863
D	644	15345	161	3850
E	614	15375	154	3857
F	620	15369	155	3856
G	618	15371	155	3856
H	587	15402	147	3864
I	604	15385	151	3860
J	597	15392	150	3861
K	591	15398	148	3863
L	608	15381	153	3858
M	633	15356	159	3852
N	626	15363	157	3854
O	602	15387	151	3860
P	642	15347	161	3850
Q	626	15363	157	3854
R	606	15383	152	3859
S	598	15391	150	3861
T	636	15353	160	3851
U	650	15339	163	3848
V	611	15378	153	3858
W	601	15388	151	3860
X	629	15360	158	3853
Y	628	15361	158	3853
Z	587	15402	147	3864

5.4 Training Procedures

This section describes classifier training policies employed for each learning structure.

Dataset Training/Test Sample Split The decision on how to split the OptDigit and PenDigit datasets between training and test sets was predetermined because the datasets were distributed with the divisions already made. However, the Letter dataset was manually split with 80% of the samples used for the training set and the remaining 20% for the test set. The first 16000 samples were separated to form the training set and the last 4000 formed the test set as is the typical approach regarding this dataset according to its accompanying documentation (Asuncion and Newman, 2007).

The preparation of the Letter dataset for incremental learning experiments required additional splitting. As mentioned earlier, one training dataset was needed to train a base

classifier for each class. Additional three datasets were required to train incremental IPSL classifiers on top of their base classifiers. The incremental datasets varied in their total sizes. Their total training samples numbered 10%, 20% and 30% in size relative to the size of the datasets used to train their base classifiers. In order to achieve that, 75% of the samples from the training dataset were set aside to make up the training datasets for base classifiers of each class. The remainder was used to create the three subsets. The details of these datasets can be seen in Table 5.4.

Table 5.4: Size of the training sets for the UCI Letter dataset used in incremental learning.

	Base classifier dataset		10% Dataset		20% Dataset		30% Dataset	
Class	Pos.	Neg.	Pos.	Neg.	Pos.	Neg.	Pos.	Neg.
A	473	11514	52	1275	106	2573	158	3848
B	459	11528	51	1276	102	2577	153	3853
C	441	11546	49	1278	98	2581	147	3859
D	483	11504	53	1274	108	2571	161	3845
E	460	11527	51	1276	103	2576	154	3852
F	465	11522	51	1276	104	2575	155	3851
G	463	11524	51	1276	104	2575	155	3851
H	440	11547	49	1278	98	2581	147	3859
I	453	11534	50	1277	101	2578	151	3855
J	447	11539	50	1277	100	2579	150	3856
K	443	11544	49	1278	99	2580	148	3858
L	456	11531	50	1277	102	2577	152	3854
M	474	11512	53	1274	106	2573	159	3847
N	469	11518	52	1275	105	2574	157	3849
O	451	11536	50	1277	101	2578	151	3855
P	481	11506	53	1274	108	2571	161	3845
Q	469	11518	52	1275	105	2574	157	3849
R	454	11533	50	1277	102	2577	152	3854
S	448	11539	50	1277	100	2579	150	3856
T	477	11510	53	1274	106	2573	159	3847
U	487	11500	54	1273	109	2570	163	3843
V	458	11529	51	1276	102	2577	153	3853
W	450	11536	50	1277	101	2578	151	3855
X	471	11515	52	1275	106	2573	158	3848
Y	471	11516	52	1275	105	2574	157	3849
Z	440	11547	49	1278	98	2581	147	3859

Training Sample Bootstrapping Section 2.4 discussed different sequences available for *showing* training samples to a learning algorithm. The strategy employed in all experiments for this research was straightforward. The approach used here *showed* all available training samples to the learning algorithms from onset of the training process. For the

monolithic ensemble classifiers this meant that all training samples were present in the learning process from beginning to end. Meanwhile, the cascaded and the PSL structures retained all positive training samples, but removed negative training samples from each layer when a classifier learned to classify them correctly. The consequence of this was that the total number of available negative training samples diminished as layer numbers increased. The reason for using this approach instead of a more complex one, was in order to avoid diverting any emphasis from the PSL structure if performance gains were encountered and also due to the fact that the total number of negative training samples was insufficient for a *bootstrapping* strategy.

Boosting Algorithm The boosting algorithm employed for learning was Discrete AdaBoost. A slight departure was made from that of Viola-Jones (Viola and Jones, 2004) approach in constructing the weak learning algorithm. In this thesis, during boosting the weak learner attempted to minimize the total error rate instead of maximizing hit rates.

Weak Learner Type The decision on what type of a weak learner to use for the boosting process was driven by the desire to employ simple components. This was done so that accurate performance measures of the framework could be made, instead of allowing a situation to develop where a degree of uncertainty existed as to which part of a learning framework contributed mostly. For this reason and also due to the tendency of more powerful weak learners to overfit training data, a decision stump described in Section 2.2.3, was used. This decision was made while considering the research of Lienhart, Kuranov and Pisarevsky (2003) who reported on advantages of using more powerful CART decision trees which may have contributed to high detection rates in their experiments.

Tunable Parameters The target total error rates at training were set to 0% for all experiments. Also, 50% was set as a target FPR as well as 100% TPR for all layers in the training of CoBEs. The remaining tunable parameters are summarized in the Table 5.5.

Training Mode Since the scope of this research was limited to first ascertaining the feasibility of the PSL learning framework, a simpler form of binary-class classification introduced in Section 2.2.4, was chosen over multi-class classification. Due to complexities associated with multi-class classification, as well as it still being an open problem, it was

Table 5.5: The settings of tunable parameters for the training of the four experiments.

	Max Layers	Max Weak Classif. (per layer)	Max PSL Nodes (per layer)
Monolithic Ensemble	1	10000	NA
Standard Cascaded	100	50	NA
PSL	100	15 (per node)	50
Variant PSL	100	15 (per node)	50
Incremental Cascaded	75 (per round)	50	NA
IPSL	75	15 (per node)	50

decided that by pursuing this mode the challenges associated with it would have overshadowed the PSL structure and would have failed to accurately render its performance abilities. Therefore, the implications of this decision on training using a dataset consisting of multiple classes, was that the training mode had to be one-vs-many. This meant that the product of training was one classifier per class and in the case of the UCI Letter dataset, this resulted in 26 classifiers; one for each letter of the alphabet. An advantage of producing multiple classifiers was that performance results could be averaged out and it provided more protection against outliers. One-vs-all training mode was also suitable due to the fact that each classifier was exposed to a disproportionately larger number of negative training samples than positive training samples. Since classifiers were intended to perform in an environment where *rare event detection* (Section 2.3) was anticipated, it was prudent to replicate this in the training environment as well.

The training mode for parallel cascaded classifiers used in comparison against IPSL classifiers was somewhat different. The parallel cascades were trained with positive training samples that were misclassified by cascaded base classifiers. These misclassified positive training samples were trained against all new and old negative training samples from base classifiers to construct a parallel cascade. An increased number of negative training samples were used in order to ensure that the generalization ability was preserved.

Cascade Optimization Optimization of cascaded and the PSL classifiers in respect to the usage of different combinations of input parameters for the learning algorithm, was not pursued. This means that the best estimates for parameters regarding maximum number of weak classifiers per layer and per node were made at onset without further ad

hoc experimentation. It is possible that more optimal classifiers for both structures could have been trained with more experimenting.

On the other hand, classifiers for the cascaded and the PSL structures were optimized for accuracy tests to perform on optimal points on their respective ROC graph curves, which are discussed further in the next section. This form of optimization involved pruning layers away from CoBE classifiers. This is necessary because latter layers of classifiers can often reduce hit rates of a classifier without significantly lowering false detection rates and vice versa.

The optimization of IPSL classifiers was performed differently. The decision was made at the outset to train and optimize only base PSL classifiers. The intention was to prune away layers from base classifiers that were suboptimal, whose presence would have only served to slow down and not provide accurate training runtimes. The expectation was also that IPSL classifiers would operate optimally on same points on the ROC graph curves as their respective base classifiers.

5.5 Performance and Accuracy Testing Methodology

Once all classifiers were trained, an appropriate methodology had to be implemented for testing their performance runtimes and accuracies. This section will examine the types of measurements that were carried out on all classifiers.

5.5.1 ROC Graphs

Receiver operating characteristics (ROC) graphs were used throughout this research for analyzing classifiers. ROC graphs are a technique for visualizing and evaluating accuracy performances of classifiers (Fawcett, 2006). ROC graphs have their origins in the signal detection theory where they have been used to illustrate the tradeoff between hit rates and false detection rates of classifiers (Egan, 1975). In recent years there has been an increase in the use of ROC graphs amongst the machine learning community, which can be attributed to the realization that simple classification accuracy is an insufficient metric for measuring the overall generalization ability of classifiers (Fawcett and Provost, 2002).

As well as being a powerful tool for visualizing the generalization ability of classifiers, ROC graphs have the property of making them especially useful in domains with skewed

class distributions and unequal classification error costs (Fawcett, 2006). This characteristic is important in this research since an uneven number of positive and negative test samples were used due to the nature of one-vs-many classifier training (see Table 5.3). This means that the negative test datasets were many times larger than the positive test datasets. The consequence of this on simple classification accuracy measures which highlight total error, is to favor classifiers which achieve a lower false detection rate while placing less weight on their hit rate performances.

In order to properly interpret ROC graphs it is important to first understand the four different possible outcomes when a classifier is applied to an instance. If an instance is positive and it is classified as positive, then it is counted as a *true positive*; if it is classified as negative, it is subsequently counted as a *false negative*. If an instance is negative and it is classified as negative, it is counted as a *true negative*; otherwise if it is classified as positive, it is counted as a *false positive*. The ROC graphs show how the false positive rate of detection increases as the positive detection rate increases.

An example of ROC graphs is given in Figure 5.1. The figure shows three ROC curves with each one hypothetically representing a generalization of a different classifier. A point in the figure located at (0,1) represents perfect classification accuracy. The yellow curve illustrates the best generalization ability of the three curves since the points on its curve appear closest to the ideal (0,1) coordinate. From this, the optimal point for the classifier with the yellow curve can be selected as being in the approximate vicinity of (0.11,0.85). In contrast, the classifier represented by the blue line displays a behavior of randomly guessing a class and is therefore not of any practical use. The point at (1,1) on the graph represents a policy of pursuing unconditional positive classifications, while point (0,0) represents an opposite strategy of never issuing a positive classification.

Classifiers whose points on a ROC curve appear on the left-hand side of the graph, near the X axis, may be thought of as "conservative". This means that they make positive classifications only with strong evidence and so they produce few false positive errors. As a consequence, they often have low true positive rates. Classifiers whose points on a ROC curve appear on the upper right-hand side of the graph may be thought of as "liberal", meaning they make positive classifications with weak evidence and in turn classify nearly all positives correctly. Often this results in them having high false positive rates in return

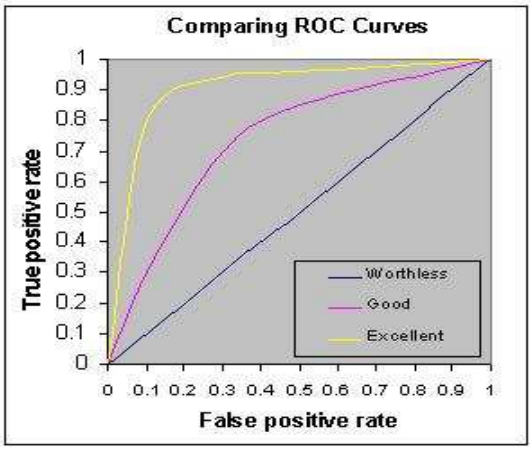

Figure 5.1: Example of ROC graph curve generalization with three different performances. (http://gim.unmc.edu/dxtests/ROC3.htm)

(Fawcett, 2006).

In this research, ROC graphs were also used to optimize classifiers. Each ROC graph produced markers on a curve that signified an end of each layer in a cascade. Using these points, optimal layer numbers for each of the cascaded and PSL classifiers were chosen. ROC curves for single layered monolithic ensemble classifiers are omitted in this research. This is due to the nature of the training method employed that does not generate standard ROC curves which portray increases in hit rates as false detection rates increase. The reason for this characteristic is that, in this research the weak learner attempted to minimize total error rates instead of maximizing hit rates, as was done by Viola-Jones (Viola and Jones, 2004) approach. Training a classifier based on minimizing total error rates involves a constant tradeoff between hit rates and false detection rates that results in erratic oscillations between the two axis on a graph. This property yields a graph that cannot be compared to standard ROC curves.

5.5.2 Classifier Training Convergence

The convergence of a classifier during training refers to its ability to attain its target training error rate. The target training error rate for all classifiers in this research was set to zero percent. The analysis of a training phase regarding its ability to reach a target training error rate reveals the strength of a learning framework used, but also the complexity of a dataset it is attempting to learn.

The capability and efficiency of a learning framework can also be measured by how quickly the learning process converges to a target training error rate. The analysis of experiments will take into account how many weak classifiers needed to be generated in order to produce final classification rules. Likewise, a faster convergence will become visible through fewer layers generated per classifier which will also be discussed.

5.5.3 Simple Classification Accuracy

Further generalization tests on classifiers were carried out through three simple classification accuracy measurements. The first measured test hit rates of all classifiers. Test hit rates are analogous to *true positive* detections. These figures highlight the proportion of positive detections on a positive test dataset. The second accuracy measurement compared *false detection* rates, which represent the proportion of negative test samples which have been misclassified as positive. Lastly, the accuracy of classifiers was examined through *total error* rates on test datasets. The total error rates represent the proportion of total *false negatives* plus *false positives* in relation to an entire test dataset. This figure, although arguably more informative than the previous two, is prone and sensitive to changes in class distributions. This means that as the proportion of positive to negative instances changes in a test set, so do the figures. As mentioned previously, because experiments in this research contained more negative test samples in proportion to positives test samples, the total error rates became more responsive to classifiers' *false positive* detection rates rather than to their *true positive* detection rates.

The accuracy testing for parallel cascaded classifiers was conducted in a different manner. A sample instance was first evaluated against a base cascaded classifier. If a result was positive, then the subsequent parallel cascaded classifier was not called to perform further evaluation. If a sample instance was rejected as a negative, then the second parallel classifier evaluated a sample instance and made the final decision as to its class.

5.5.4 Performance Runtimes

Performance runtimes took into consideration classifiers' training and detection phases. Training phase runtimes of various classifiers were compared in seconds. Detection phases on the other hand did not involve explicit runtimes. Instead, they were analyzed through

total numbers of weak classifiers which would have to be executed at detection time. Similarly, the total number of layers comprising cascaded classifiers also formed a part of the analysis.

Chapter 6

Experimental Results

This chapter will present results from the experiments outlined in Chapter 5. These experiments involve a preliminary evaluation of the PSL structure followed by more rigorous experiments using a larger dataset. The PSL classifiers are compared against those of the single layered monolithic ensemble and the cascaded structures. The last experiment compares the PSL classifiers with those of the variant form of the PSL structure.

The experiments will firstly analyze classifier training phases. The training phases will be analyzed by examining their runtimes, classifiers' convergence patterns on training sets and also through the generation of ROC graph curves. In addition to that, the total number of weak classifiers and the total number of layers for each strong classifier will assist in further analysis of the training phase and will also be serve as an evaluation tool for the detection phases. Secondly, classifiers' accuracy will be presented. The accuracy performance will be assessed by comparing the hit, false detection and total error rates on test datasets which the classifiers have not *seen* previously. Except for the first experiments, all accuracy figures express only the results of classifiers which have been optimized using ROC graph curves. The first set of experiments present results from unoptimized classifiers as well, in order to highlight the importance of the optimization process.

6.1 Preliminary PSL Experimental Results

The preliminary round of experiments were conducted in order to gauge the feasibility of the PSL structure as a framework for training robust classifiers. The ability of the PSL structure to achieve this was measured against the classifiers produced by the standard

cascade and the single layered monolithic structures using the smaller UCI PenDigit and the OptDigit datasets.

6.1.1 Training Phase Analysis

With a few exceptions, most of the classifiers trained on the OptDigit and PenDigit datasets converged to an absolute zero training error. Table 6.1 presents the total number of weak classifiers generated during training for each classifier class. In majority of cases, the figures show that the total number of weak classifiers comprising PSL classifiers is smaller than that of the single layered or of the cascaded classifiers. This is noteworthy because Zhang and Viola (2007) show, that in order for a cascade layer to reach a lower target false positive rate, more weak classifiers need to be generated. A training structure which has the ability to achieve effective false positive rates with a reduced amount of weak classifiers, ultimately has better convergence properties. This problem of convergence becomes even more pronounced as the training complexity increases in the later layers or in difficult training datasets. This tends towards a greater increase in the total number of weak classifiers and ultimately results in protracted training runtimes. The PSL structure appears to be more effective in handling this scenario.

Table 6.1: Total number of weak classifiers comprising each digit classifier for the PenDigit and OptDigit datasets comparing the single layered monolithic, standard cascaded and the PSL classifiers. The totals include weak classifiers for all the nodes in the PSL classifiers.

	PENDIGIT DATASET			OPTDIGIT DATASET		
Class	Monolithic	Cascade	PSL	Monolithic	Cascade	PSL
0	485	1003	147	58	90	54
1	4532	986	344	445	363	135
2	711	428	175	149	177	94
3	783	803	232	434	536	157
4	167	99	79	125	181	94
5	1674	679	212	200	390	110
6	90	236	92	95	129	87
7	282	446	125	90	158	94
8	1618	626	191	1097	737	300
9	1331	650	259	1713	731	350

Additionally, it can also be observed that a significantly smaller number of cascade layers were generated by the PSL structure for each of the classifiers, compared to the standard cascaded structure, as shown in Tables 6.2 and 6.3. This observation, together with the lower weak classifier count, strongly indicate to an inherently faster convergence

property of the PSL structure over the single layered monolithic ensemble and the standard cascaded structures.

Table 6.2: Total number of layers comprising each classifier for every digit in the OptDigit dataset. Total layer numbers required to train a classifier are shown with adjacent columns demonstrating the total number of layers per classifier after the optimization process is performed.

Class	Cascade	Cascade (optimized)	PSL	PSL (optimized)
0	27	16	6	6
1	31	18	11	8
2	23	9	7	5
3	28	20	8	4
4	18	7	6	4
5	28	10	7	3
6	18	16	6	4
7	23	14	5	3
8	27	16	7	4
9	23	22	8	4

Table 6.3: Total number of layers comprising each classifier for every digit in the PenDigit dataset. Total layer numbers required to train a classifier are shown with adjacent columns demonstrating the total number of layers per classifier after the optimization process is performed.

Class	Cascade	Cascade (optimized)	PSL	PSL (optimized)
0	12	10	4	4
1	20	20	6	4
2	20	11	6	4
3	22	12	7	5
4	19	17	5	3
5	20	14	7	6
6	15	14	6	4
7	16	8	6	2
8	22	12	11	5
9	27	17	13	6

The ROC graph curves for the standard cascaded classifiers and for the PSL classifiers are shown in Figures 6.1a-d representing PenDigit datasets while Figures 6.2a-d show the ROC graph curves for the OptDigit datasets. From these figures, it is evident that the cascaded classifiers generalize slightly better than PSL classifiers. This can be seen from their steeper curves and a denser concentration of points in the left had corner of the figures. This signifies higher hit rates with lower false detection rates. However, The figures also demonstrate that the PSL classifiers converge faster to their optimal point on the curve than the standard cascaded classifiers. The optimal points attained by PSL

Table 6.4: Training runtimes (seconds) of the monolithic, standard cascaded and the PSL classifiers on the PenDigit and OptDigit datasets.

	PENDIGIT TIMES			OPTDIGIT TIMES		
Class	Monolithic	Cascaded	PSL	Monolithic	Cascade	PSL
0	95	57	2	9	3	3
1	4567	75	4	92	9	5
2	170	7	3	27	5	4
3	202	63	4	89	22	5
4	22	2	1	22	5	4
5	728	35	3	37	10	4
6	10	3	2	16	3	4
7	43	7	3	16	3	4
8	688	29	3	301	33	8
9	495	28	4	583	64	9

classifiers are also comparable to those of the cascaded classifiers.

Table 6.4 presents the training runtimes for each class in the PenDigit and OptDigit datasets. Although training runtimes are mainly determined by the number of weak classifiers being generated, it is important to note that the training structures in these experiments are affected by their increasing numbers to different degrees. In order for the single layered monolithic structure to determine hit and false detection rates during training, the entire set of weak classifiers in an ensemble must be evaluated after each round. In contrast, the cascaded and the PSL structures need only evaluate a fraction of the total set of weak classifiers which comprise only those belonging to the layer for that particular round. Hence, the results demonstrate that the cascaded structure can generate vastly larger numbers of weak classifiers per classifier than the single layered monolithic structure, whilst producing reduced runtimes resulting from the savings brought on by the evaluation of hit and false detection rates of only one layer.

In many cases, the improvement brought on by PSL is orders of magnitude faster than the single layered and cascaded structures. The extent of the difference between the classifiers regarding training runtimes is smaller for the OptDigit dataset than for the PenDigit dataset. This can be attributed to the fact that the OptDigit dataset is less than half the size of the PenDigit dataset and therefore the performance differences are less pronounced. Consequently, it can be expected that as the size of training datasets increase, so will the training runtime performance advantage of the PSL structure over the remaining two structures.

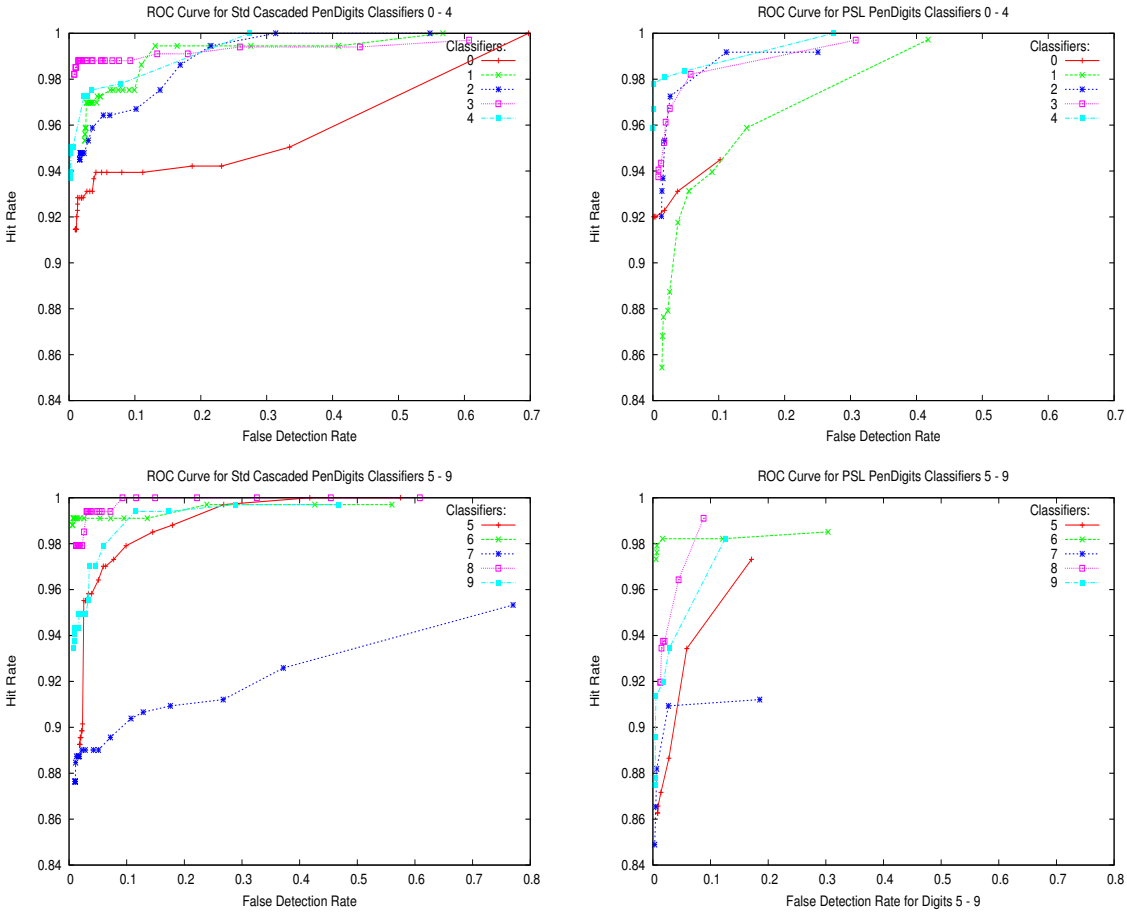

Figure 6.1: ROC graph curves for the PenDigit datasets showing the convergence of the standard cascaded and the PSL classifiers on the test dataset. a) cascaded classifiers 0 - 4. b) PSL classifiers 0 - 4. c) cascaded classifiers 5 - 9. d) PSL classifiers 5 - 9.

6.1.2 Hit Rate

Figures 6.3a and 6.3b present the hit rates for all three structures on both UCI test datasets. The hit rates in these graphs represent unoptimized classifiers. In these figures, the PSL classifiers clearly lost some accuracy, although in some cases the accuracy was comparable to that of the monolithic ensemble classifiers, which generally had the highest rates. Once the classifiers were optimized, the discrepancy between the PSL classifiers and others decreased. Figures 6.4a and 6.4b compare the classifiers of the three structures in their optimized state. From these figures, it is noticeable that the hit rate gap between the PSL classifiers and others has narrowed on both datasets and that the PSL classifiers operating at their optimal point on the ROC graph curves are generally more comparable than previously.

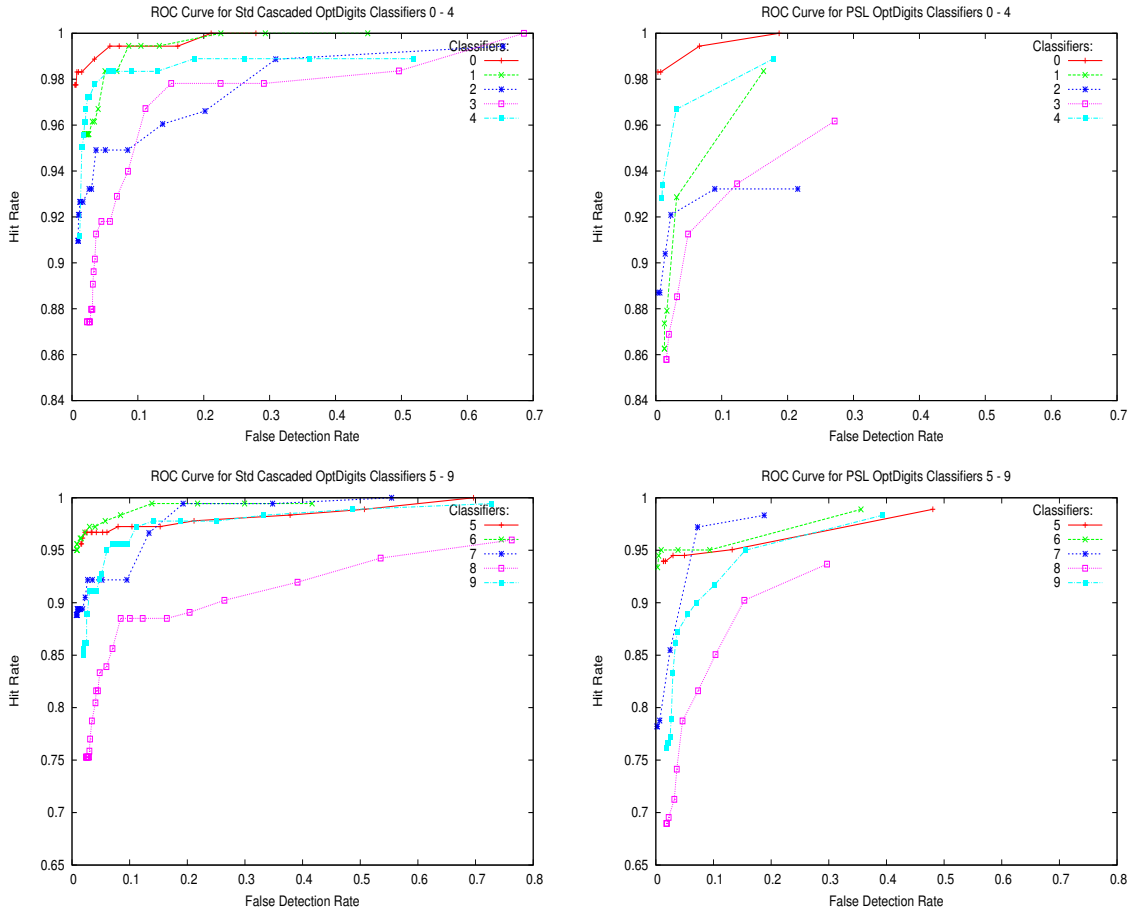

Figure 6.2: ROC graph curves for the OptDigit datasets showing the convergence of the standard cascaded and the PSL classifiers on the test dataset. a) cascaded classifiers 0 - 4. b) PSL classifiers 0 - 4. c) cascaded classifiers 5 - 9. d) PSL classifiers 5 - 9.

6.1.3 False Detection Rate

Figures 6.5a-b present false detection rates for all three structures on both UCI datasets. First of all, the figures show false detection rates for classifiers in their unoptimized state. PSL classifiers demonstrate lowest false detection rates. Since there is a tradeoff involved between hit rates and false detection rates, it can be concluded that the lower false detection rates of PSL classifiers account for lower hit rates. When the figures of the optimized classifiers are examined, the slight advantage of the PSL classifiers is still retained. Figures 6.6a-b demonstrate the optimized false detection rates and it is noticeable that for PSL classifiers they are for the most part still lower than those of other structures.

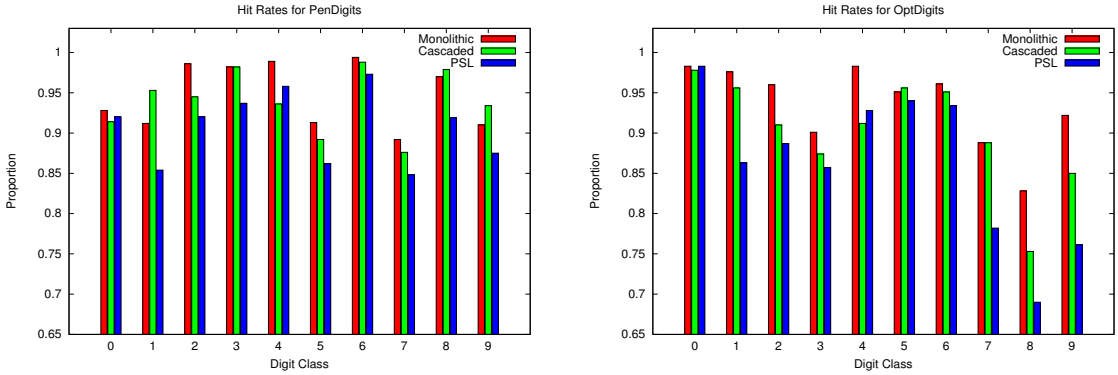

Figure 6.3: Hit rate results of the monolithic, standard cascaded and PSL classifiers prior to their optimization. a) PenDigit dataset b) OptDigit dataset.

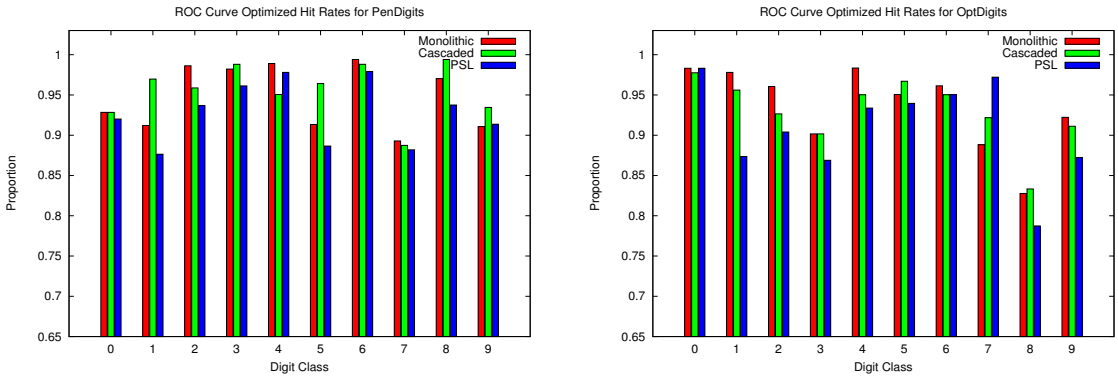

Figure 6.4: Hit rate results of the monolithic, and optimized standard cascaded and PSL classifiers. a) PenDigit dataset b) OptDigit dataset

6.1.4 Total Error Rate

The total error graphs highlight the proportion of false negative and false positive detections in respect to the entire test dataset. They are arguably the more important since they integrate the two perspectives into a single dimension. These graphs can be seen in Figures 6.7a-b which first of all show results of the unoptimized classifiers. The figures illustrate that the total error rates for the monolithic classifier are marginally better than of the standard cascaded and the PSL classifiers. The figures of the latter two are very comparable to each other. With the introduction of the total error rates for the optimized classifiers in Figures 6.8a-b, the overall picture changes little. The total error is seen to increase slightly for both the standard cascaded and the PSL classifiers in order to account

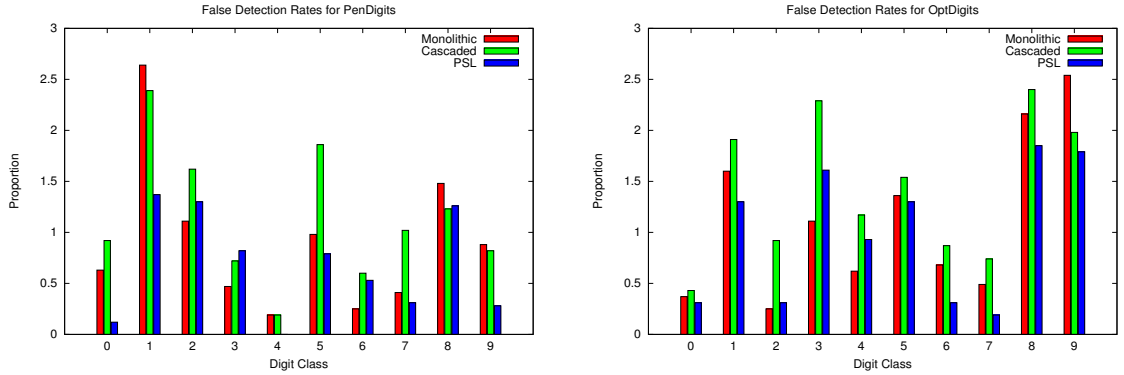

Figure 6.5: False detection rate results of the monolithic, standard cascaded and PSL classifiers prior to their optimization. a) PenDigit dataset b) OptDigit dataset

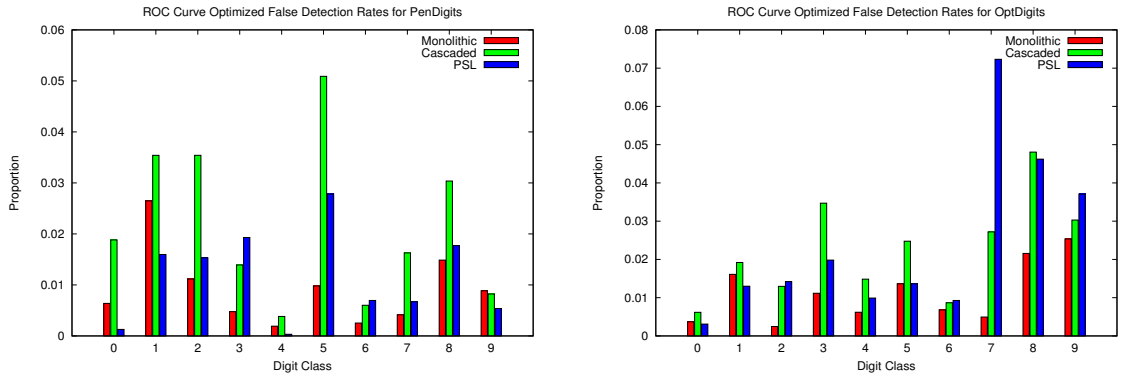

Figure 6.6: False detection rate results of the monolithic, and optimized standard cascaded and PSL classifiers. a) PenDigit dataset b) OptDigit dataset

for an increase in false detection rates which came as a consequence of an increase in hit rates.

6.1.5 Runtime Performance

Table 6.1 showed that the PSL structure significantly reduced the number of weak classifiers needed to build a final classification rule. This, together with a pattern of consistently generating fewer cascade layers, shown in Tables 6.2 and 6.3, demonstrates that PSL classifiers generated on these datasets are superior in their detection runtimes than the classifiers of the other two structures. PSL classifiers are also more efficient and faster at detecting positive samples than the other two structure because, depending on a sample, PSL classifiers may only need to evaluate the first nodes of each layer. In a best case

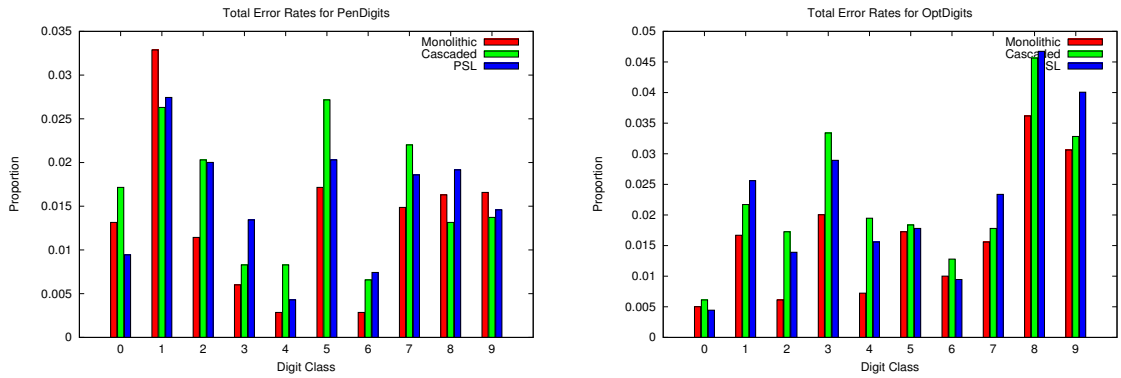

Figure 6.7: Total error rates of the monolithic, standard cascaded and PSL classifiers prior to their optimization. a) PenDigit dataset b) OptDigit dataset

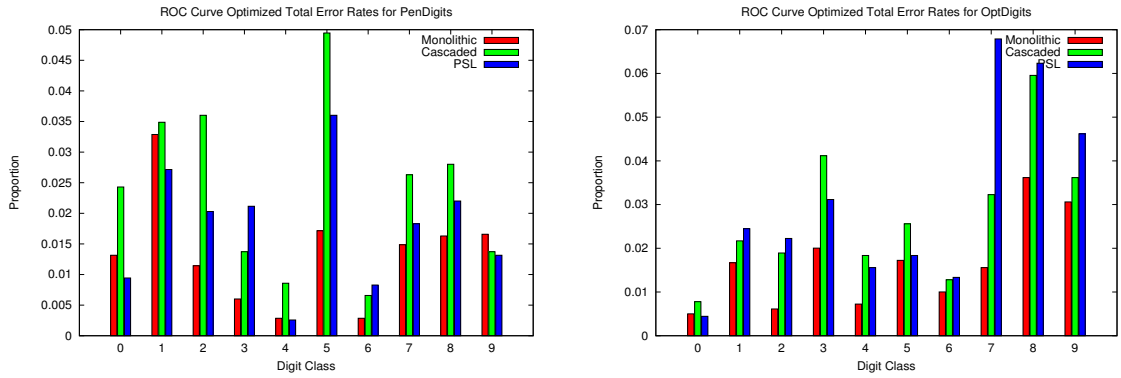

Figure 6.8: Total error rates of the monolithic, and optimized standard cascaded and PSL classifiers. a) PenDigit dataset b) OptDigit dataset

scenario, for a PSL classifier comprising of n layers and m number of nodes per layer, a positive detection may require only n number of nodes to be computed before a positive detection is made. The cascaded classifiers on the other hand, require all weak classifiers within a layer to undergo a calculation.

6.2 Further PSL Experimental Results

A second round of experiments were conducted on the PSL structure using the larger UCI Letters dataset whose details can be found in Tables 5.1 and 5.3. The aim of these experiments was to provide a more rigorous examination of the performances of the PSL classifiers. In much the same way as the preliminary experiments, this round of testing

compared the PSL classifiers to the single layered monolithic and the standard cascaded classifiers. This section will discuss the observations from those results.

6.2.1 Training Phase Analysis

All PSL classifiers converged to a zero training error as did the classifiers for the standard cascaded structure except for one. As was expected, a third of the monolithic classifiers failed to achieve the same. The reason for this lies in the fact that the monolithic training structure, unlike the cascaded or the PSL, retains all the training samples from the beginning of the training process to the end. This makes the separation between target objects and negative objects harder, especially in larger datasets. Also, the fact that the Letter dataset comprised of a relatively small number of dimensions, contributed to the convergence problem of the monolithic classifiers. Hence, convergence rates seen in the preliminary experiments were not replicated for the monolithic structure.

Table 6.5 illustrates again the ability of the PSL structure to train classifiers which are comprised of fewer weak classifiers. This confirms once more the faster convergence of PSL classifiers during the training phase. The figures show that the PSL classifiers consistently contain significantly fewer total weak classifiers than those of the other structures. In some cases, the PSL structure generated reductions in total weak classifier counts by up to 67% than the standard cascaded.

The ROC graph curves for the standard cascaded and the PSL classifiers are shown in Figures 6.9a-f. In agreement with the preliminary experiments, the figures confirm that the PSL reaches optimal accuracy in fewer layers than the cascaded structure, signifying a faster convergence. The steeper and more effective ROC graph curves of the cascaded structure witnessed in the preliminary experiments, are not seen in these datasets. Instead, the ROC graph curves illustrate a fairly similar generalization pattern for both structures.

Table 6.6 presents the training runtimes in seconds for each of the classes of the three training structures. The runtimes for the monolithic ensemble classifiers are characterized by protracted and unpractical runtimes. The longest training times were in excess of 15 hours for a single classifier. The cascaded structure performed much better than the monolithic ensemble structure and displayed an increase in speed that was orders of magnitude faster. However, the PSL structure performed even faster. Runtime improvements

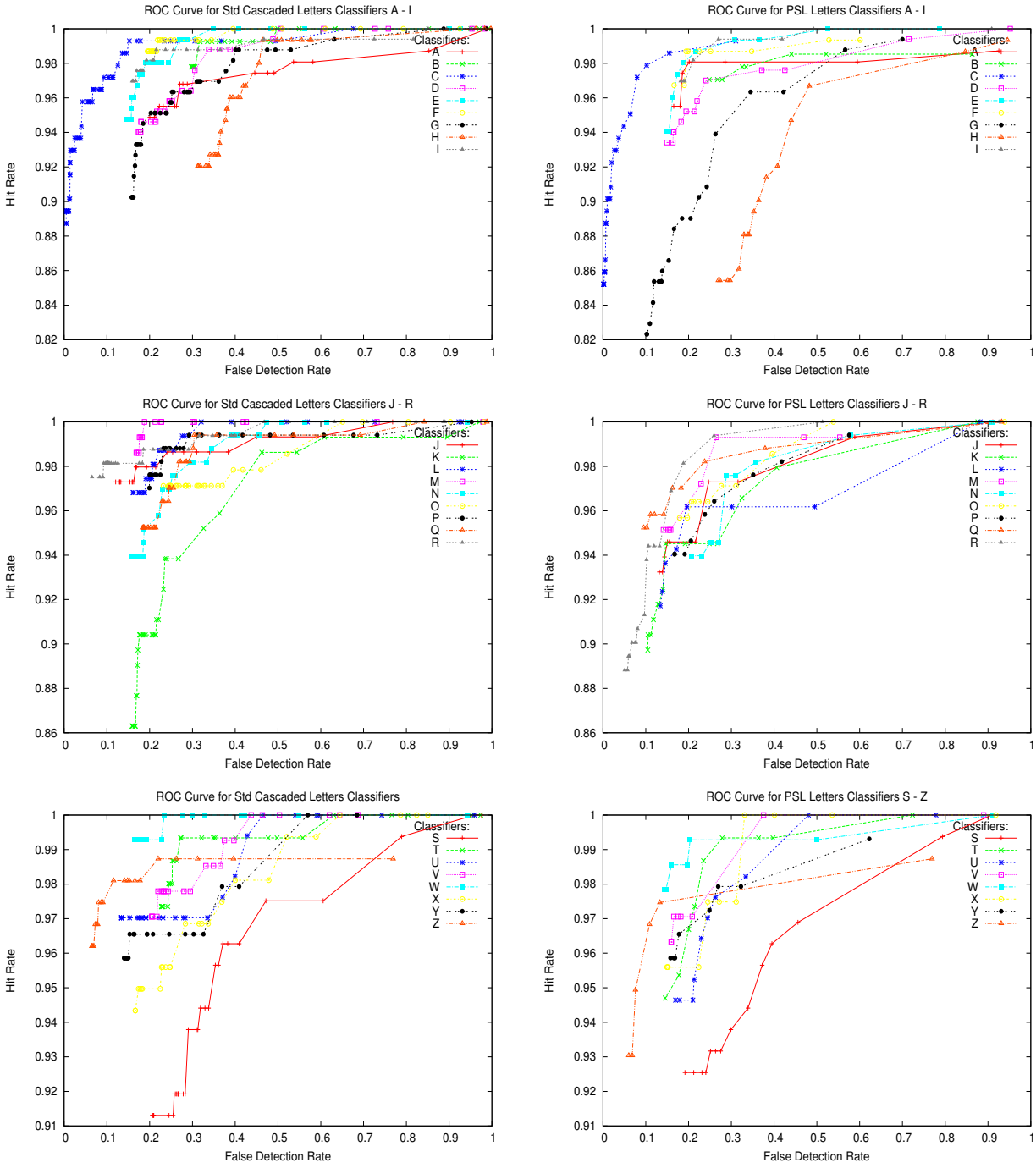

Figure 6.9: ROC graph curves for the Letters dataset showing the convergence of the cascaded and the PSL classifiers on the test sets. a) cascaded classifiers A-I. b) PSLclassifiers A-I. c) cascaded classifiers J-R. d) PSL classifiers J-R. e) cascaded classifiers S-Z. f) PSL classifiers S-Z.

Table 6.5: Total number of weak classifiers required to train classifiers for each of the UCI Letter character classes for the monolithic, standard cascaded, PSL and the variant PSL classifiers. The totals include weak classifiers for all the nodes in the PSL classifiers.

Class	Monolithic	Cascaded	PSL	Variant PSL
A	494	136	155	145
B	677	481	201	159
C	10000	2378	1298	738
D	10000	966	453	373
E	10000	906	351	299
F	3460	698	235	208
G	10000	1391	787	539
H	10000	911	683	471
I	1110	273	201	204
J	1579	449	310	236
K	10000	1192	519	353
L	10000	780	260	275
M	763	202	186	123
N	1506	669	279	244
O	10000	922	362	307
P	5165	806	259	205
Q	10000	987	330	214
R	10000	1247	752	487
S	10000	924	546	425
T	6813	740	264	218
U	3778	538	354	302
V	10000	556	237	244
W	510	228	142	140
X	2817	550	285	262
Y	1491	576	221	205
Z	1072	371	129	140

of up to an order of magnitude over the standard cascaded were witnessed in cases where difficult classifiers were encountered. PSL structure's superior training runtimes confirm the initial findings of the preliminary experiments.

6.2.2 Hit Rate

Figure 6.10 demonstrates hit rates for the three structures. The figure demonstrates that the monolithic ensemble classifiers outperform the rest of the classifiers in over 50% of the cases. The cascaded classifiers on the other hand perform slightly better than those of the PSL structure, though the PSL classifiers do exhibit a comparable rate. The results seen here also match the findings of the preliminary experiments in which the monolithic

Table 6.6: a) Training runtimes (seconds) of classifiers for each of the UCI Letter character classes for the monolithic, standard cascaded, PSL and the variant PSL classifiers.

Class	Monolithic	Cascaded	PSL	Variant PSL
A	223	2	6	6
B	345	7	5	5
C	53572	844	18	12
D	51927	61	8	8
E	54952	76	7	6
F	5924	25	5	5
G	52813	197	8	7
H	54384	98	9	8
I	792	4	5	5
J	1406	6	6	5
K	53774	115	7	7
L	53947	48	6	6
M	431	3	5	5
N	1344	16	6	6
O	52480	59	7	7
P	12804	40	6	5
Q	53633	92	6	6
R	52547	148	8	7
S	53347	63	7	7
T	21126	35	7	6
U	6972	18	6	6
V	53366	11	6	5
W	228	3	5	5
X	4002	22	6	6
Y	1303	14	5	5
Z	731	5	1	1

ensemble classifiers produced classifiers with higher hit rates.

6.2.3 False Detection Rate

Preliminary experiments showed how the monolithic ensemble and cascaded classifiers produced higher hit rates which were accompanied by elevated false detection rates. The same pattern is repeated in these experiments. Figure 6.11 shows that in 85% of instances, the monolithic ensemble classifiers produced highest false detection rates. The best false detection rates come from PSL classifiers which produced the lowest rates in 62% of the cases. Cascaded classifiers performed best in the remaining 38% of the cases. Therefore, the diminished hit rates of the PSL classifiers can once again be justified by lower false detection rates in majority of the cases.

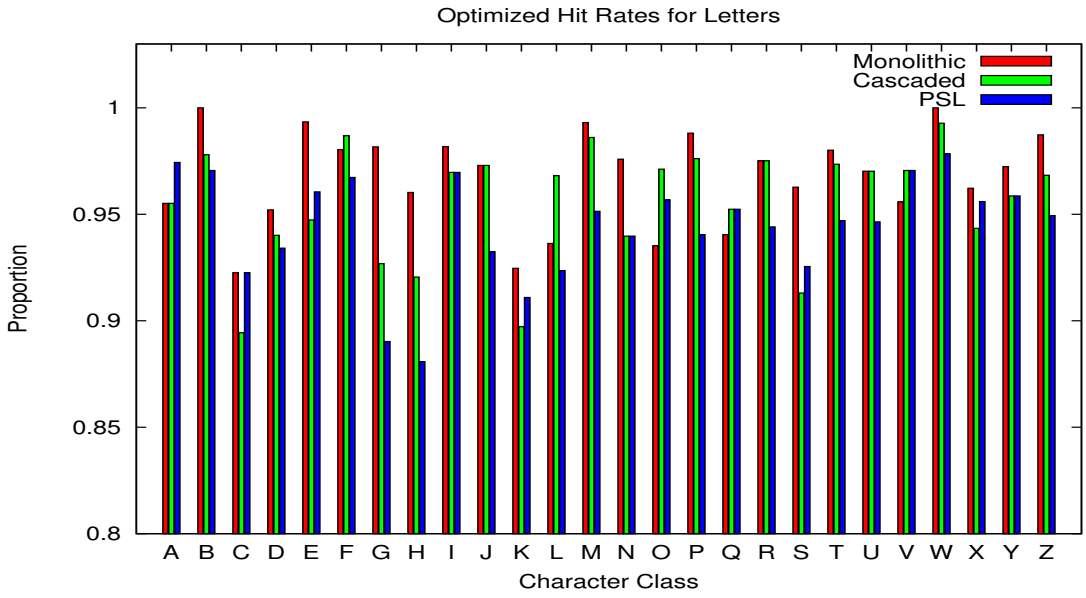

Figure 6.10: Optimized hit rates for the UCI Letter dataset comparing the monolithic, standard cascaded and the PSL classifiers.

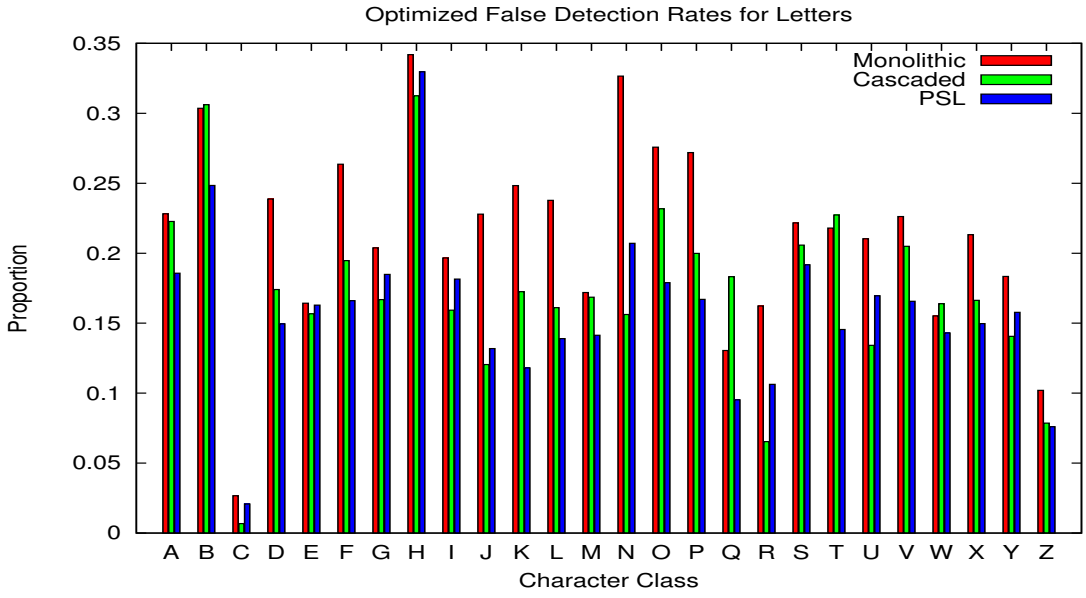

Figure 6.11: Optimized false detection rates for the UCI Letter dataset comparing the monolithic, standard cascaded and the PSL classifiers.

6.2.4 Total Error Rate

The total error rates in Figure 6.12 are almost identical to the false detection results. This shows that the discrepancy between the classifiers' hit rates and the false detection rates was not large enough to alter the patterns of the false detection figures.

Once more, the majority of the lowest total error rates are attributed to the PSL classifiers. The final figure affirms PSL classifiers' robustness and their ability to generalize as well as the other classifiers on this dataset.

6.2.5 Runtime Performance

The observations on the runtime performance of the PSL structure mirror the results found in the preliminary experiments as well. Table 6.5 points to a reduced number of weak classifiers comprising the final strong classifier. Likewise, Table 6.7 shows that a smaller number of layers are also generated during training. Importantly, it has also been shown that a reduction in total number of weak classifiers and layer numbers comes at no expense to the ability of the classifiers to generalize, therefore confirming the PSL structure's robustness.

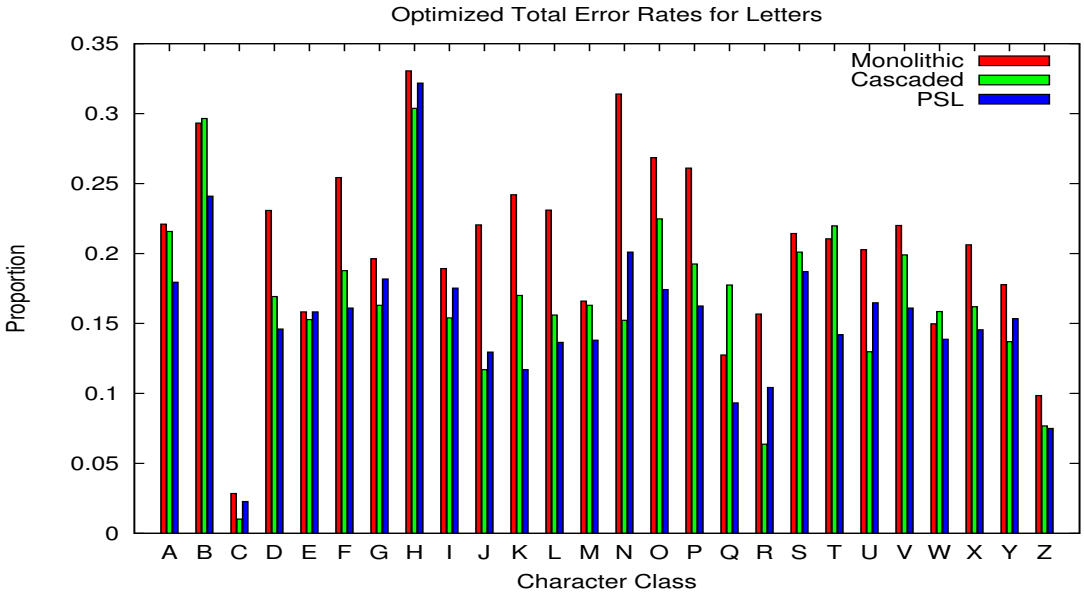

Figure 6.12: Optimized total error rates for the UCI Letter dataset comparing the monolithic, standard cascaded and the PSL classifiers.

6.3 Results of the Variant PSL Structure

Sections 6.1 and 6.2 examined the generalization ability and the training runtime phases of the PSL structure in comparison to the single layered monolithic and the standard cascaded structures. This section will compare the variant version of the PSL structure, first introduced in Section 4.2, to the original PSL. The comparisons between the two structures will be conducted based on their performances on the UCI Letters dataset.

6.3.1 Training Phase Analysis

Strong similarities between the two structures were observed in convergence rates at training. Classifiers of both structures succeeded in converging to an absolute zero error rate. Likewise in Figure 6.13, the ROC graph curves for the variant PSL structure yielded figures which showed a high degree of resemblance to those of the original PSL in Figure 6.9. This indicated that the generalization ability of the variant PSL structure remained intact and would display similar accuracy rates to the original PSL. The apparent lack of compromise in the generalization ability of the variant PSL dispelled initial concerns. The primary concern was that the modification to the structure, which saw a reduced number of negative training samples *shown* to the algorithm, would introduce higher false detection rates.

Nevertheless, the comparisons between the training phases revealed some interesting differences as well. Table 6.5 shows the total number features generated by the two structures during the learning process. The figures show that the variant PSL structure consistently generated fewer weak classifiers than the original, particularly as the difficulty of the classifier increased. A reduction in weak classifier numbers in the variant PSL classifiers can be seen as being up to 43%. Of all the classifiers, only 12% did not realize a reduction over classifiers of the original PSL structure. The classifiers which did not outperform the original PSL classifiers fall into the category of easier classifiers to train. This means that they required few weak classifiers to produce a classification rule, thus the difference in numbers was minor. In those instances, the original PSL classifiers outperformed the variant by only 1.5%-8% fewer weak classifiers.

An additional difference in the training phases of the two structures can be seen in the total number of cascade layers. Table 6.7 shows that in general, variant PSL classifiers

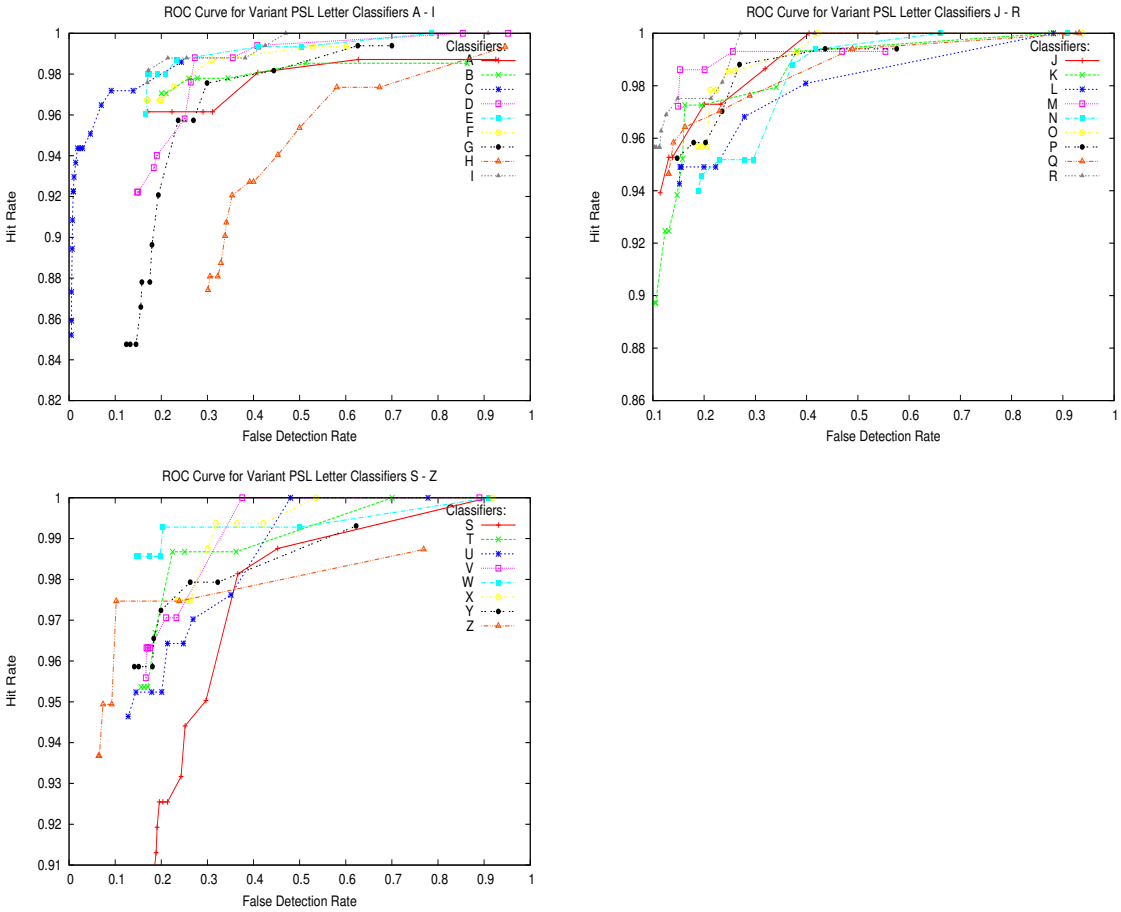

Figure 6.13: ROC graph curves for the UCI Letter dataset comparing the convergence patterns of the PSL and the variant PSL classifiers on a test set. a) classifiers A-I. b) classifiers J-R. c) classifiers S-Z.

required fewer layers to converge. This observation, together with the ability of the variant PSL structure to generate fewer weak classifiers demonstrates that it has a slight overall advantage. The variant PSL structure has demonstrated that it converges faster during training and produces classifiers which are likely to execute faster at detection time, while preserving the generalization ability.

The last two columns of Table 6.6 compare the runtimes of the two PSL structures, though in this case, they are inconclusive. On this dataset, the figures do not show that the variant PSL structure is significantly faster than the original. A faster training runtime might have been expected since fewer negative training samples are propagated in each succeeding node and fewer weak classifiers were generated. The lack of a speed up can be put down to the size of the dataset not being large enough to allow the runtime

Table 6.7: Total number of layers comprising the classifiers for the UCI Letter dataset. The figures show totals for the optimized PSL and the variant PSL classifiers.

Class	Cascade	PSL	Variant PSL
A	15	7	7
B	19	8	7
C	53	29	17
D	32	13	11
E	24	10	10
F	26	9	8
G	37	20	14
H	33	16	14
I	22	8	8
J	25	10	8
K	27	14	11
L	30	8	8
M	18	8	6
N	26	9	9
O	30	11	10
P	25	9	7
Q	34	10	8
R	39	18	15
S	31	14	12
T	28	9	8
U	26	10	10
V	24	8	9
W	21	8	8
X	25	11	10
Y	23	8	8
Z	15	6	7

performance differences to become apparent. The already very low runtimes of the original PSL structure do not leave room for substantial reductions. A hint of a potential training runtime acceleration can be seen when examining the class C, which was the most difficult to train for all structures resulting in longest training times. When training this sample, the variant PSL structure achieved a speed up of 33% whilst its generalization ability did not experience any degradation.

6.3.2 Hit Rate

In 61% of the instances, the variant PSL structure outperformed the original PSL in terms of the hit rates. The original PSL achieved higher hit rates in 11% of the cases with the remaining cases were even. Figure 6.14 highlights the hit rates of the two structures.

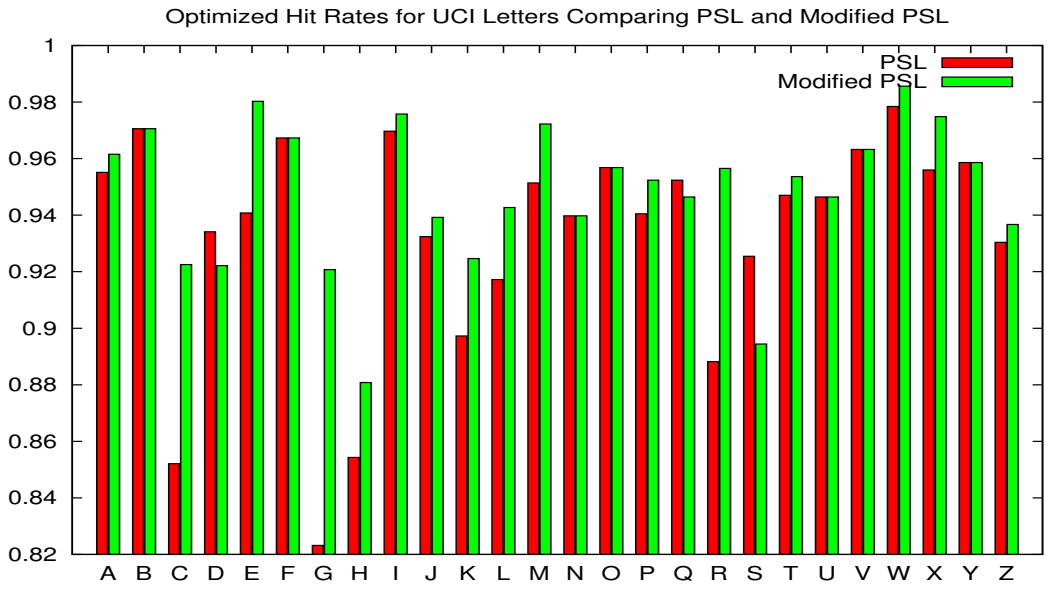

Figure 6.14: Optimized hit rates for the UCI Letter dataset comparing the PSL and the variant PSL classifiers.

6.3.3 False Detection Rate

In Figure 6.15, the false detection rates from the two structures can be seen. In 65% of the instances, the variant PSL has false detection rates which are higher than those of the original version. These figures account for the variant PSL classifiers' higher hit rates in Figure 6.14 and point to the fact that the classifiers from both structures are most likely operating at slightly different points on ROC graph curves. Nonetheless, an interesting trend can be witnessed when comparing the hit rates in Figure 6.14 for classifiers which achieved equal hit rates with how they compared in Figure 6.15 on the false detection comparison. The trend shows that 66% of the variant PSL classifiers have a lower false detection rate than the original PSL classifiers in instances where their hit rates were identical. What is more, experiments from earlier sections have shown that there is usually an associated high false detection rate that accompanies strong hit rates. In these comparisons, it can be seen that in 27% of the cases, the variant PSL classifiers have attained higher or equivalent hit rates in comparison to the original PSL whilst returning lower false detection results for the same classifiers. The original PSL classifiers achieved the same results in 12% of the cases. This shows that the variant PSL is comparable to the original PSL and that the generalization ability is not compromised with the diminished

size of the negative training datasets.

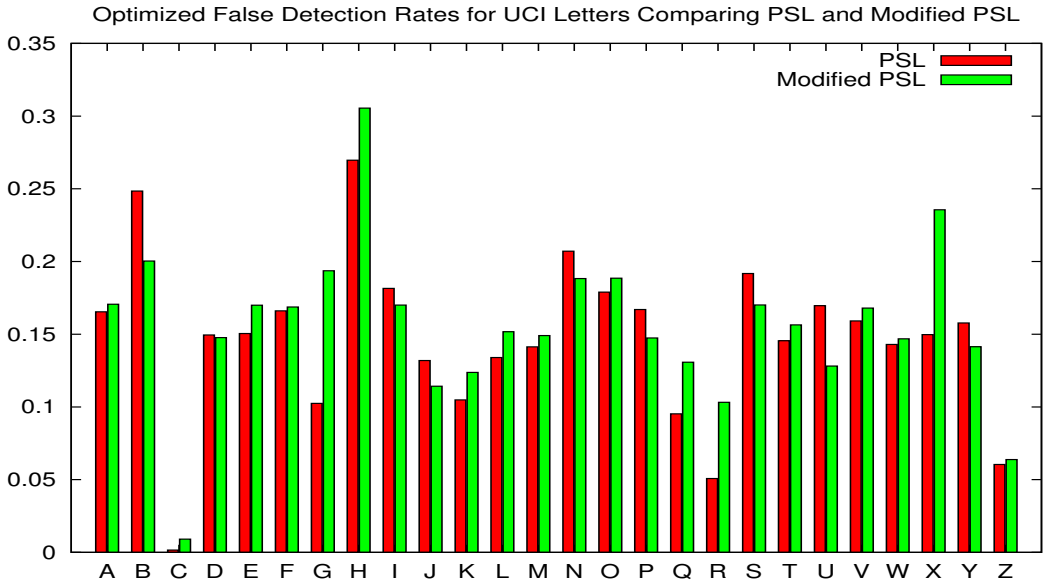

Figure 6.15: Optimized false detection rates for the UCI Letter dataset comparing the PSL and the variant PSL classifiers.

6.3.4 Total Error Rate

The figures showing the total error rates between the two structures do not change the overall picture but serve to affirm the figures provided by the false detection graphs. Figure 6.16 for total error rates, mirrors the figures of the false detection graph closely and the variance in results between the two structures seen in the false detection graph are closed only marginally by the factoring in of the higher hit rates of the variant PSL structure. It can therefore be concluded that the original PSL structure in general maintains slightly lower total error rates for that dataset.

6.3.5 Runtime Performance

The combined ability of the variant PSL structure to produce classification rules comprised of fewer weak classifiers (Table 6.5), which require a reduced number of layers (Table 6.7), demonstrates that they possess a faster runtime performance over the original PSL classifiers.

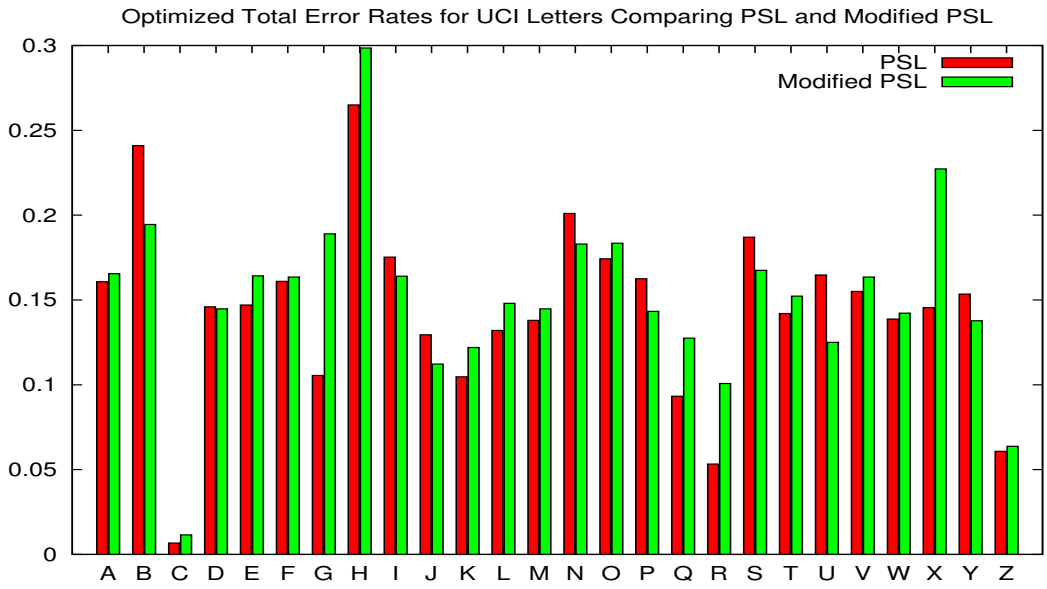

Figure 6.16: Optimized total error rates for the UCI Letter dataset comparing the PSL and the variant PSL classifiers.

6.4 Summary

In summary, the generalization ability of the PSL classifiers has been shown to be robust against the monolithic and the cascaded classifiers. The initial experiments showed lower accuracy rates of the PSL classifiers, but subsequent, more rigorous experiments on larger datasets pointed to accuracy rates of the PSL classifiers being as good, if not even better than that of the other two structures. The overall hit rates of the PSL classifiers tended to be slightly lower than that of the other classifiers but generally displayed lower false detection rates to account for them. The more important total error figures were closely aligned to those of the false detection rates. This pattern demonstrated that the methods of testing favored classifiers like PSL which produced lower false detection results and were not penalized as heavily for lower hit rates when the total error graphs were rendered. The reason for this is that during the one-vs-all mode of testing, many more negative as opposed to positive test samples were utilized, thus enabling the false detection results to affect the final total error graphs to a greater extent.

Both the preliminary and the subsequent experiments on the PSL structure showed consistent conclusions on the analysis of the training and detection phases. The PSL classifiers repeatedly demonstrated an accelerated runtime together with fewer weak classifiers

making up the final classification rule with a reduced number of layers. The latter two observations directly point to a faster convergence during training and a faster execution runtime during the detection phase.

The comparison between the two PSL structures also confirmed the effectiveness of the variant version of PSL. The variant form of the PSL produced classifiers which achieved higher hit rates but at a price of higher false detection rates than the original PSL classifiers. This was mirrored by the total error rates that pointed to a marginally better all round performance of the original PSL classifiers compared to the variant form. The data points to the fact that both sets of classifiers were operating at slightly different parts of the ROC graph curve which makes exact comparisons difficult. Despite this, it is still possible to conclude that the variant PSL structure is viable. What is more, the variant PSL classifiers generally produced classifiers with fewer classifiers pointing to a faster training phase convergence and an even faster detection runtime.

Chapter 7

Incremental Learning Results

This chapter presents results from experiments conducted on the IPSL training structure. The IPSL framework proposed a solution for enabling incremental training of CoBE classifiers. After the findings have been presented, the chapter will close with a discussion on the implications of the experimental results as well as a path for future research.

7.1 IPSL Incremental Learning Analysis

The comparisons of experimental results will be made between the base PSL classifiers and their respective IPSL classifiers. Each base classifier will be compared against three IPSL classifiers which will be referred to as the 10% increment, 20% increment and the 30% increment classifiers. The percentage refers to the size of the dataset used to train the IPSL classifiers in respect to the size of the dataset used to train the base classifiers.

The main aspect of comparisons will examine the effects of incremental training on the generalization ability of the resulting classifiers. Also, comparisons between the different IPSL classifiers will be made in order to ascertain the influence of different dataset sizes on the generalization. Lastly, for objectivity the IPSL classifiers will be compared to parallel cascaded classifiers which simulate incremental learning of the cascaded framework. Additional details of the experiments can be found in Chapter 5.

The experiments will examine the training phases of all the classifiers in which the training errors, the runtimes and the ROC graph curves form the bulk of the analysis. Subsequently, the accuracies of the classifiers will be presented, highlighting hit rates, false detection rates and the total error rates. Finally, the detection runtimes will be analyzed

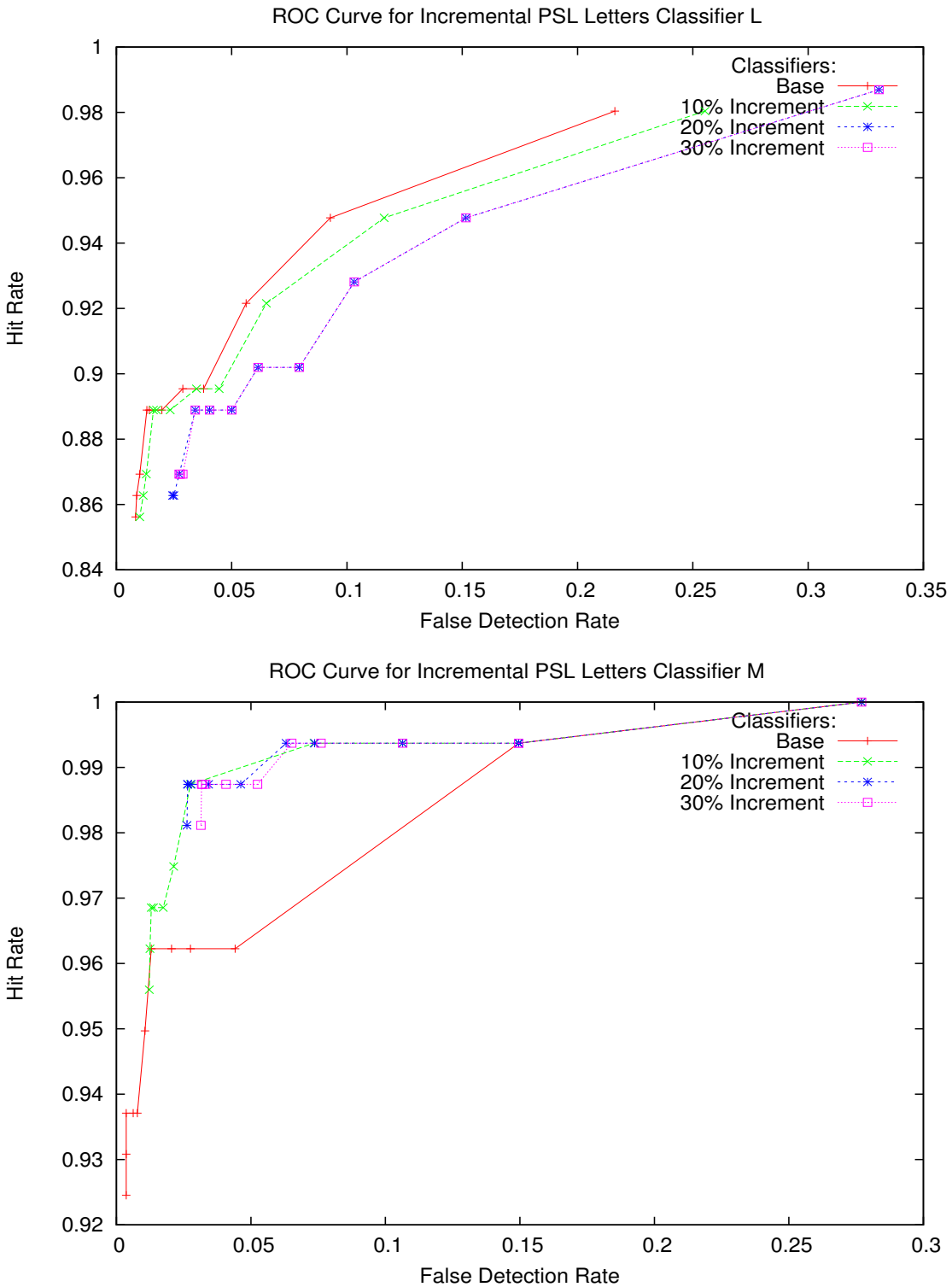

Figure 7.1: ROC graph curves for the Letters dataset showing the generalization of the IPSL classifiers on a test dataset, comparing the base classifier against the 10%, 20% and 30% increment classifiers. a) Class L, showing an example of a IPSL generalization affected by higher false detection rates and slightly improved hit rates. b) Class M displaying a successful example of IPSL classifiers improving the generalization of the base classifier with both false positive and true positive detections.

through weak classifier totals and cascade layer totals.

7.1.1 Training Phase Analysis

All base PSL classifiers converged to a zero training error. With a few exceptions, the cascaded base classifiers achieved the same training error targets. Once the incremental datasets were trained on top of the base classifiers, the training error increased for both types of classifiers. For IPSL and parallel cascaded classifiers, the error rate was influenced solely by false detections. The IPSL classifiers were observed to have the largest increases in the training error and the error rates themselves increased with the size of the incremental datasets.

There are two causes for this in the IPSL classifiers. Firstly, if a base classifier had achieved a high generalization ability for positive samples, then it was likely that no positive samples were left at each layer for further IPSL training. This meant that the classifiers could not learn to reject the remaining negative training samples. The second reason for this lies in the fact that the optimization of the layers was done on the base classifier prior to incremental training, meaning that fewer layers remained and thus diminished the possibility of learning to reject the outstanding negative samples.

The cascaded classifiers on the other hand were observed to generate a training error that was decreasing in magnitude as larger incremental training datasets were introduced. This is expected, as under the cascaded approach, additional horizontal new layers are guaranteed to be generated for outstanding misclassified samples which leads to the lowering of the training error.

A ROC graph curve was generated for each class in the UCI Letter dataset which compared the generalization of the PSL base classifiers against the 10%, 20% and 30% increment classifiers. Cascaded classifiers were omitted in order to focus more on the effects of IPSL training on base classifiers.

The analysis of the ROC graph curves for the IPSL classifiers produced mixed results, but showed a clear trend. Out of the 26 different classes in the dataset, 73% experienced improvement as a result of IPSL training through either one or all of the 10%, 20% and 30% increment classifiers. The remaining 27% of the classes observed a deterioration in the generalization by all three IPSL classifiers. Figures 7.1a-b show two representative forms of

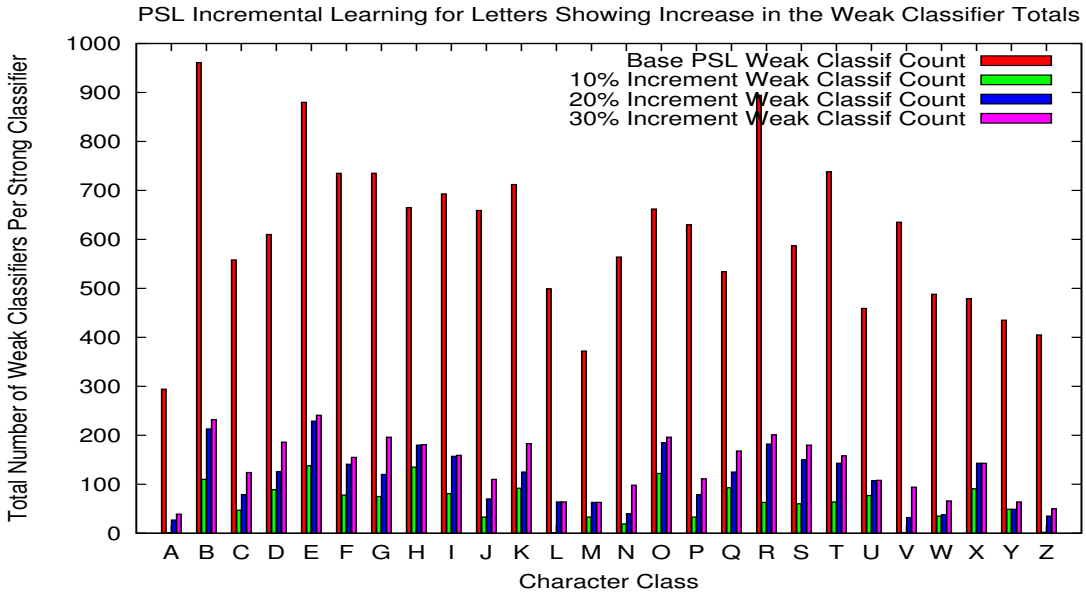

Figure 7.2: Total number of weak classifiers generates per base and IPSL classifier for each incremental dataset on the UCI Letter dataset.

generalization seen in the results. The first, demonstrates the deteriorated generalization. This graph is typical of roughly a quarter of cases, where it is observed that the hit rates at optimal points on the curve are slightly improved, but the false detection rates are elevated. The second graph is indicative of the trend witnessed in majority of classes. This graph displays a generalization curve in which false detection rates of base classifiers are preserved by IPSL classifiers, but the hit rates at corresponding optimal points on the curve are improved. The ROC graph curves for all UCI Letter classes can be found in Appendix A.

Table 7.2 presents training runtimes of the incrementally trained classifiers for both the IPSL and the parallel cascaded structures. The runtimes are shown for base classifiers and for each of the 10%, 20% and 30% increment classifiers. The training runtimes for the base classifiers of both structures were curious. In this experiment, both training structures required a considerably longer period to train on training datasets which were of reduced size compared to those of the experiments found in Section 6.2. A plausible explanation for this occurrence is that although the size of the dataset was reduced, thus raising expectations of a less complex and therefore faster training runtime, the distribution of the data may have been altered to a less advantageous spread. This altered data distribution

Table 7.1: Total number of layers generated per cascaded and IPSL classifiers for the UCI Letter dataset. Figures show layer totals for the unoptimized base classifiers as well as the final layer numbers after the optimization is applied.

	UNOPTIMIZED BASE CLASSIFIERS		OPTIMIZED CLASSIFIERS			
Class	Cascaded	IPSL	IPSL	10%(Casc)	20%(Casc)	30%(Casc)
A	36	10	10	25	15	18
B	71	41	21	39	34	34
C	50	23	12	23	17	21
D	54	29	14	39	39	39
E	54	50	20	28	28	28
F	54	39	16	30	24	30
G	54	53	16	33	30	30
H	54	57	16	25	19	22
I	47	18	16	25	17	25
J	40	20	15	15	19	29
K	54	35	16	25	30	25
L	41	14	11	28	18	28
M	34	12	12	30	23	30
N	54	21	14	29	29	33
O	54	34	16	44	30	36
P	54	22	14	33	33	23
Q	54	23	13	33	33	33
R	54	39	19	40	31	47
S	54	40	12	31	22	20
T	50	25	17	32	32	32
U	39	19	11	33	33	33
V	53	27	16	33	33	33
W	33	15	12	25	28	25
X	49	21	12	26	26	25
Y	35	12	10	18	30	18
Z	43	12	10	19	30	30

may have negatively affected the ability of the boosting process to discriminate between the target and the negative classes, resulting in longer runtimes. Nonetheless, the training runtimes on the PSL base classifiers in this experiment once more confirm earlier findings in Section 6.2 which show that the runtimes for the PSL classifiers are an order of magnitude faster than those of the cascaded classifiers.

The subsequent IPSL training done on top of the base classifiers produced rapid runtimes for both classifier structures. As expected, the runtimes increased in a mostly linear fashion in respect to the size of the training datasets. However, there is no substantial difference between the runtimes of IPSL classifiers and those of the parallel cascaded structures.

Table 7.2: Training runtimes for each classifier class on the UCI Letter dataset comparing the standard cascaded and the IPSL classifiers of 10%, 20% and 30% size increments.

	BASE CLASSIFIER		10% INCR.		20% INCR.		30% INCR.	
Class	Cascade	IPSL	Cascade	IPSL	Cascade	IPSL	Cascade	IPSL
A	93	6	1	1	1	1	1	1
B	1263	15	1	2	2	3	5	6
C	264	11	1	1	3	1	4	2
D	338	13	1	3	2	5	5	7
E	333	19	1	4	2	8	4	9
F	383	15	1	1	1	4	2	4
G	480	26	1	2	1	2	7	10
H	668	33	1	5	5	10	17	12
I	215	11	2	4	2	5	3	7
J	168	12	1	1	1	1	2	3
K	427	17	1	2	1	6	2	8
L	153	8	1	1	1	1	1	1
M	89	6	1	1	1	1	1	1
N	424	9	1	1	1	1	1	1
O	723	15	1	2	2	3	1	4
P	453	11	1	1	1	2	3	3
Q	650	11	1	1	1	4	4	4
R	740	17	1	1	1	3	2	3
S	515	20	1	4	5	7	10	8
T	291	13	1	1	2	6	2	6
U	190	10	1	1	1	2	1	2
V	310	12	1	1	1	1	1	6
W	87	5	1	1	1	1	1	1
X	266	14	1	2	1	6	2	6
Y	76	7	1	1	1	1	1	1
Z	194	6	1	1	1	1	1	1

7.1.2 Hit Rate

The effects of IPSL learning in respect to hit rates can be seen in Figure 7.3. The figure shows the positive detection rates in their succession from those of the base classifier to those of the classifier trained on a 30% sized dataset. The data demonstrates without exception that the hit rates increase over the base classifier when incremental training is performed. Apart from a few examples, the figures also show that the hit rates increase as the size of the datasets used in incremental training increase.

In order to place the performance accuracy of IPSL incremental learning into context, the hit rates of the cascaded incremental learning are also presented for comparison in Figures 7.4a-c. The comparisons show that the parallel cascaded classifiers have consistently attained higher hit rates. The improvement in the hit rates of the parallel cascaded

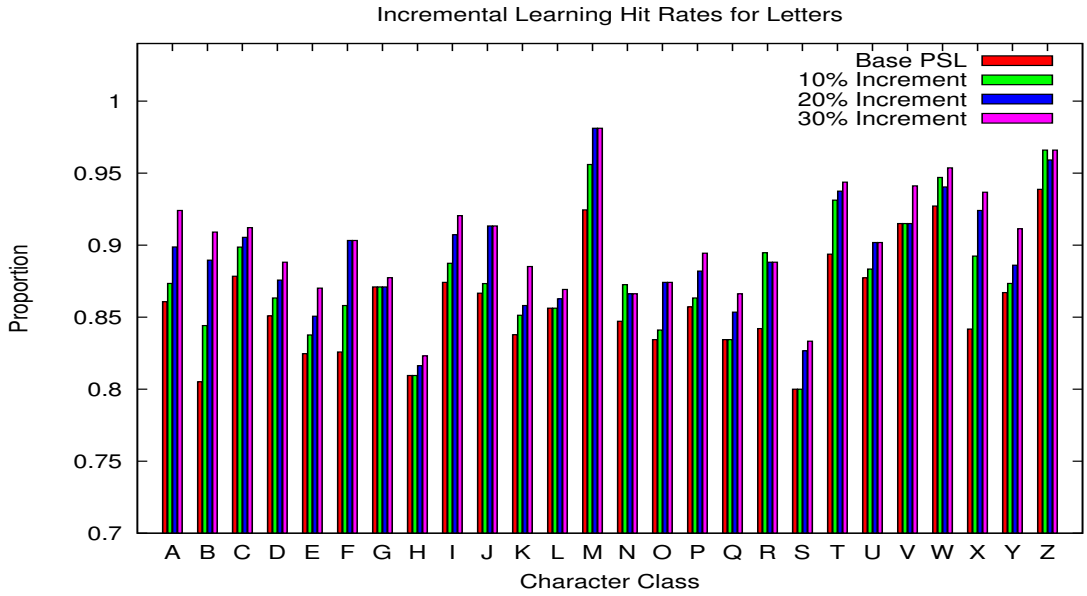

Figure 7.3: Optimized hit rates for the UCI Letter dataset comparing the base classifier and the classifiers for dataset increments of 10% - 30%.

classifiers over the IPSL classifiers is considerable with an increase of up to six percentage points. The discrepancy in performance between the classifiers of the two structures can be explained by the fact that they are operating on different points on the ROC graph curves. Whereas the cascaded classifiers have been individually optimized for accuracy the IPSL classifiers on the other hand have not. Only the PSL base classifiers were optimized by having suboptimal layers removed from their cascades.

7.1.3 False Detection Rate

As a consequence of increased hit rates for every incremental dataset, the false detection rates increased as well. Figure 7.6 demonstrates the correlation. The figure shows with few exceptions, that as the size of the incremental training dataset increased, so did the magnitude of increase in false detection rates.

The comparison of IPSL false detections to those of the parallel cascaded classifiers can be examined in Figures 7.5a-c. The figures demonstrate similar false detection accuracies between the two types of classifiers with neither the IPSL nor the parallel cascaded classifiers consistently achieving lower rates.

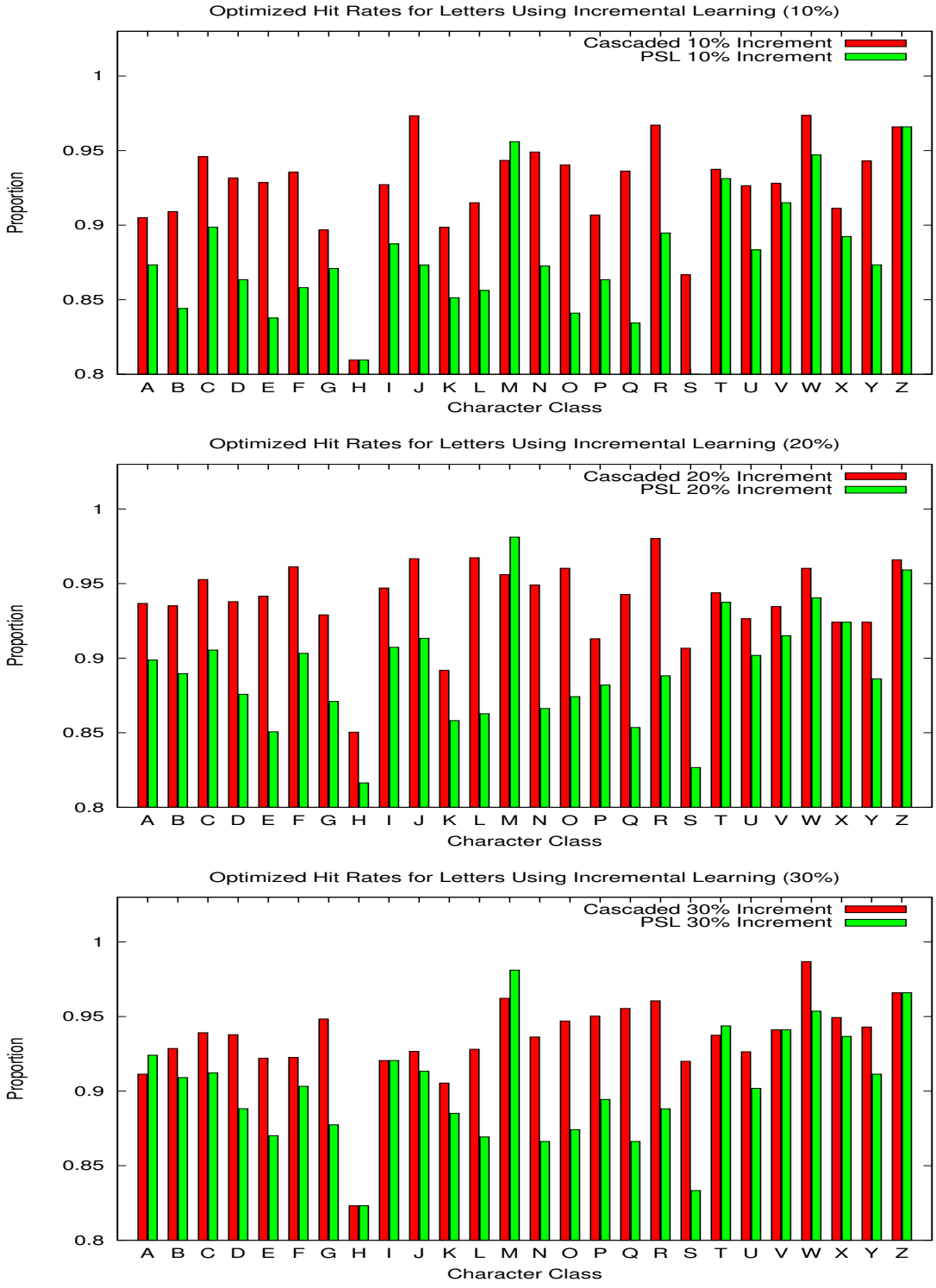

Figure 7.4: Optimized hit rates for the UCI Letter dataset comparing the standard cascaded and the IPSL classifiers. a) 10% increment dataset b) 20% increment dataset c) 30% increment dataset.

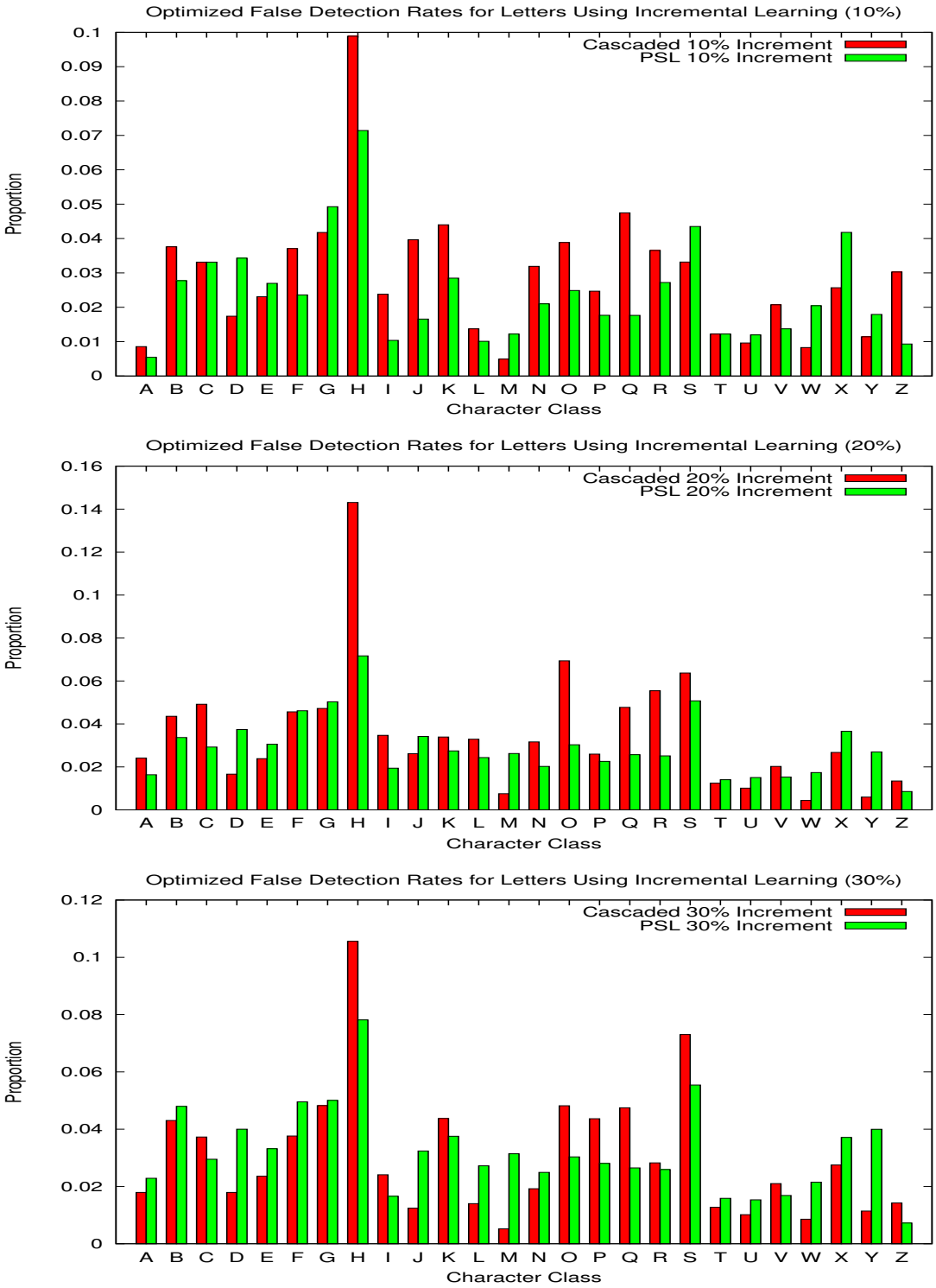

Figure 7.5: Optimized false detection rates for the UCI Letter dataset comparing the standard cascaded and the IPSL classifiers. a) 10% increment dataset b) 20% increment dataset c) 30% increment dataset.

7.1.4 Total Error Rate

The sizable differences between the IPSL classifiers' accuracies in regards to false detection rates are decreased when the hit rates are factored in to reveal total error rates in Figure 7.7. The overall comparisons of the total error rates between IPSL classifiers and parallel cascaded classifiers is seen in Figures 7.8a-c. The data from this perspective once more shows a similarity to the figures from the false detection graphs which confirms a strong resemblance in the overall generalization capability between IPSL and parallel cascaded classifiers.

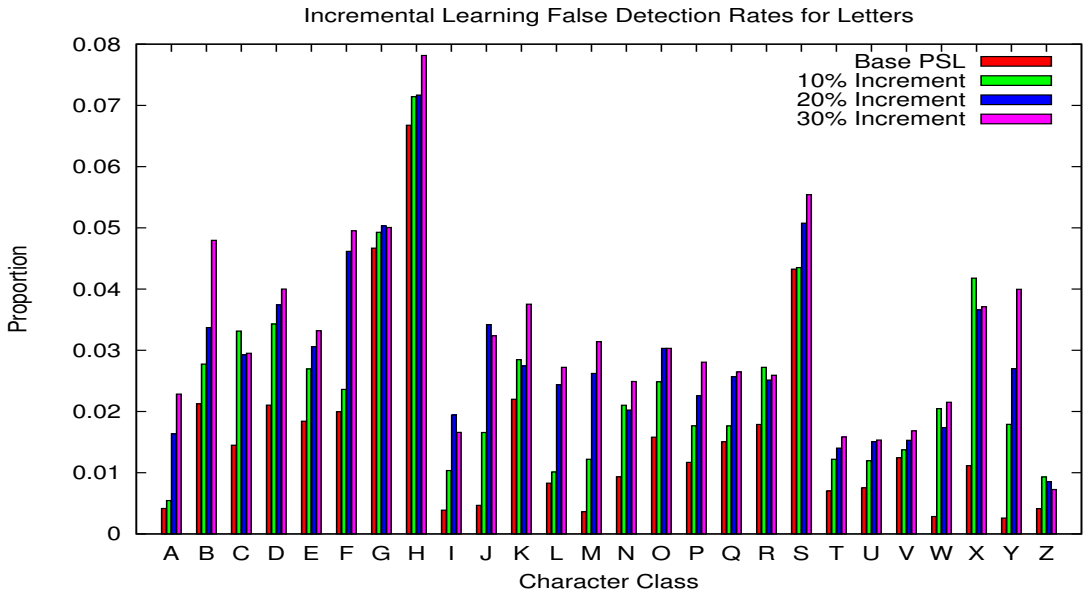

Figure 7.6: Optimized false detection rates for the UCI Letter dataset comparing the base classifier and the classifiers for dataset increments of 10% - 30%.

7.1.5 Runtime Performance

The runtime performance of IPSL classifiers will be analyzed by comparisons to the cascaded classifiers. The total number of weak classifiers generated for both types of classifiers can be seen in Figures 7.9a-c. The total number of weak classifiers generated by the IPSL structure is generally in the range of 25%-30% to that of the cascaded classifiers.

Table 7.1 demonstrates the same data from the perspective of the total number of cascades required for evaluation per classifier. The layer totals for the optimized IPSL classifiers are substantially lower than for those of the parallel cascaded classifiers, for all

three classifier increments.

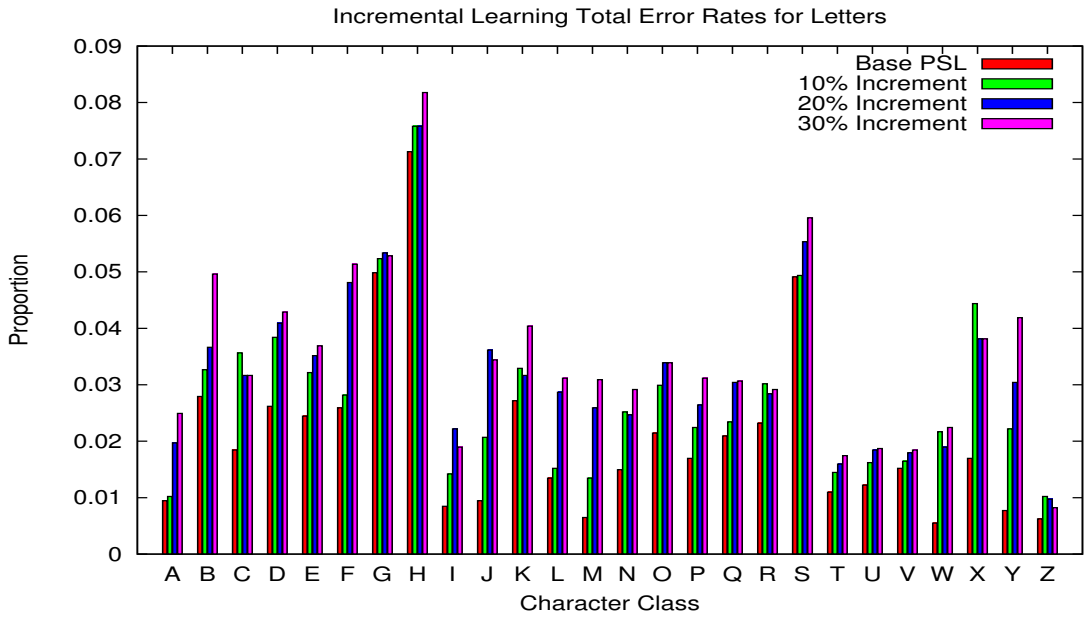

Figure 7.7: Optimized total error rates for the UCI Letter dataset comparing the base classifier and the classifiers for dataset increments of 10% - 30%.

7.2 Discussion

The experiments carried out on the UCI Letter dataset demonstrate that incremental learning using the IPSL structure within the context of boosting is plausible. The ramifications of this are that once a classifier has been trained and new training data is made available, a classifier need not be re-trained in order to accommodate additional training data. Instead, the base classifier can be improved by continuing incremental training on top of its prior training. Unlike the approach put forward by Oza and Russell (2001), the IPSL structure pursues the policy of generating additional weak classifiers, thus making IPSL classifiers more amenable to correctly classifying samples which exhibit new features. The ability to integrate new features into the classification rule indicates that the IPSL classifiers are able to expand and define more accurately the boundary between the positives and negatives. Also, the total number of additional new weak classifiers generated by each round of IPSL learning has been observed to increase linearly, and corresponds to the size of the incremental training set as seen in Figure 7.2.

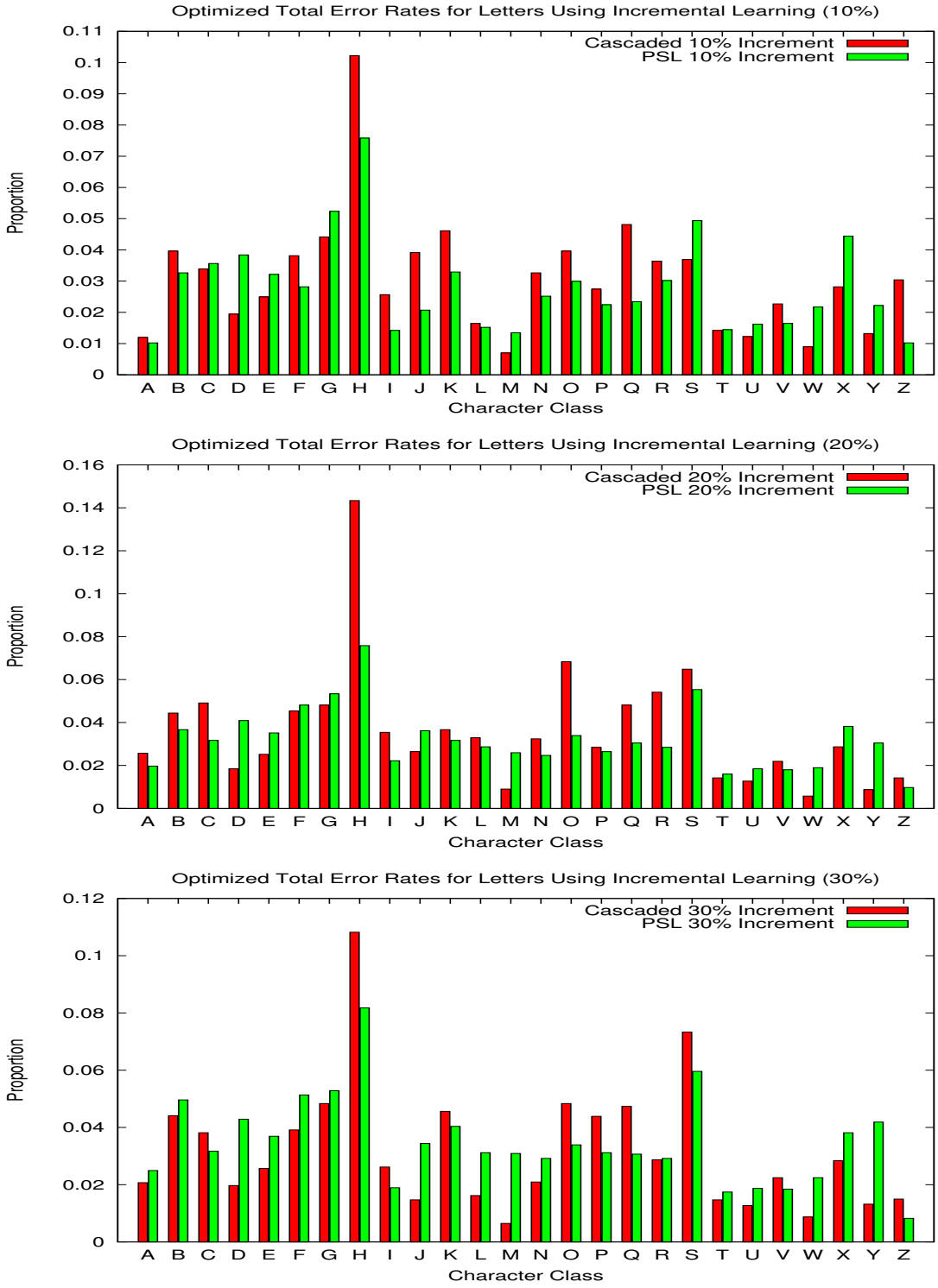

Figure 7.8: Optimized total error rates for the UCI Letter dataset comparing the standard cascaded and the IPSL classifiers. a) 10% increment dataset b) 20% increment dataset c) 30% increment dataset.

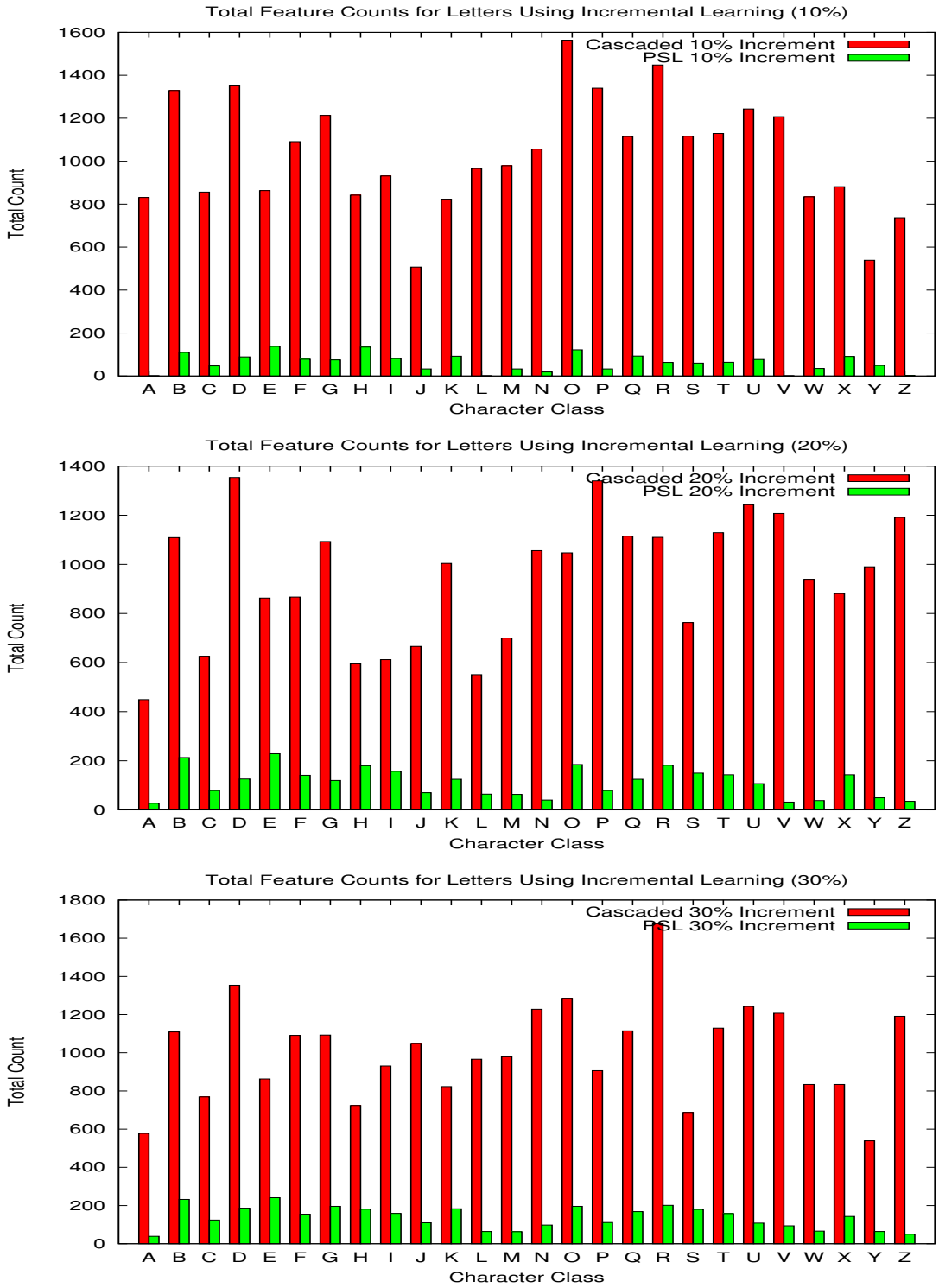

Figure 7.9: Total number of weak classifiers per classifier showing comparisons between counts of the standard cascaded and IPSL classifiers. a) 10% increment dataset b) 20% increment dataset c) 30% increment dataset.

The experimental data on the UCI Letter dataset in respect to the accuracy of the final IPSL classifiers, shows a clear trend, but also a current limitation. The IPSL framework has demonstrated a definite ability to raise hit rates of base classifiers, but has also shown a tendency to introduce higher false detection rates in return. The ROC graph curves in particular, plainly displayed this trend. At an *optimal point* on the ROC graph curve, in 75% of the cases, the IPSL classifiers succeeded in improving the hit rates of their base classifiers while at least preserving their false detection rates. In other instances, the hit rates *at a given point* on the ROC graph curves were improved only slightly over those of the base classifiers, while the false detection rates increased to a greater degree. The results indicate that the current limitation of the IPSL structure is its inability to lower base classifiers' false detection rates in subsequent incremental training. At best, the IPSL can preserve false detection rates of base classifiers and at worst, it generates additional nodes which increase them.

The overall accuracy of IPSL classifiers has shown to be comparable to those of parallel cascaded classifiers even though optimization for individual IPSL classifiers was not conducted. However, the advantage of IPSL classifiers over the cascaded is that the classification rules produced by IPSL are considerably simpler than those of the latter. The total number of weak classifiers that comprise IPSL classifiers are 25%-30% of the size of the parallel cascaded, thus indicating a simpler and a faster detection runtime. If Occam's razor is to be applied to this scenario, which states "if any two hypotheses are identical in their outcome, then the simpler one is to be classed as better", then the IPSL structure can be classified as the better one.

The IPSL training does however incur a computational penalty not present in the PSL training structure. The modification to the IPSL structure which specifies that layer training may terminate only when overall *layer targets* are satisfied, instead of only when *node targets* are met, forces the algorithm to calculate all weak classifiers in the layer after each new one is generated. The cost of calculating layer targets as opposed to only node targets imposes a computational burden which grows linearly in respect to the size of a layer.

Due to the versatility of the IPSL incremental learning structure, it is conceivable to perform IPSL incremental learning on classifiers which have been trained using different

boosting structures. The IPSL structure can convert an entire single layered monolithic classifier into a PSL node of a base classifier, and continue to train additional training data on top of it. IPSL can likewise accept cascades of boosted ensembles trained as non-PSL and perform incremental training by converting existing cascade layers into PSL nodes.

Further experiments need to be conducted on the IPSL learning structure in order to better understand how the classification rules are affected by multiple rounds of incremental learning and whether there is a point at which the generalization ability becomes adversely affected. Also, a solution to IPSL's inability to lower false detection rates needs to be formulated.

Chapter 8

Conclusion

The objectives for this research were to:

- test the training runtimes and the generalization ability of PSL classifiers against standard structures of the monolithic ensemble (Schapire, 1999) and the cascaded as proposed by Viola and Jones (2001*b*)
- explore further extensions to the PSL training structure and to perform experiments on it
- to propose a novel approach for incremental learning within the scope of CoBE using IPSL
- to explore the feasibility of IPSL training to the problem of incremental learning and implement parallel cascaded classifiers to test against

This thesis has implemented the PSL training structure as well as the monolithic ensemble and the cascaded classifier training structures for comparisons. Classifiers were trained for those structures and their performance runtimes and accuracies were compared. A variant form of the PSL structure was proposed, which sought to extend further the performance and accuracy of its original. Classifiers for the modified PSL structure were trained and contrasted in their capabilities to those of the initial PSL. This thesis has proposed a novel approach to training CoBE classifiers within the incremental learning paradigm. Experiments were conducted on IPSL classifiers and compared with parallel cascaded classifiers trained on same datasets.

The main research questions that this thesis has attempted to answer were:

1. is the PSL training structure efficient at producing classifiers which are robust and capable of real-time execution?

2. is it possible to extend the PSL structure in order to realize further performance gains?

3. can CoBE be extended in such a way that existing weak classifiers need not be re-trained during subsequent rounds of incremental training and can additional weak classifiers be appended to a base classifier in order to integrate new information, while improving the generalization?

4. is the IPSL capable of incremental learning and of producing robust classifiers?

The experiments have shown, that the PSL training structure is capable of efficiently producing classifiers in substantially reduced runtimes. The PSL classifiers are robust. They have shown similar generalization abilities to those of the monolithic ensemble and cascaded classifiers whilst exhibiting significant advantages over the latter in training as well as execution runtimes.

The original PSL structure has been modified and extended in such a way that the convergence during training has been made more efficient. This was confirmed through the resulting generation of fewer weak classifiers for each classification rule, whilst preserving the generalization abilities. The detection runtime phase of its classifiers was also increased through simpler classifiers.

Although the experimental results showed mixed results and highlighted some current limitations, the novel IPSL framework has shown how CoBE base classifiers can be improved through incremental learning. The IPSL proposed to perform incremental rounds of learning by appending new weak classifiers to the existing classification rule without altering previous weak classifiers. When presented with additional datasets which encapsulate new information, in majority of cases the IPSL classifiers have demonstrated the ability to improve the hit rates and preserve the false detection rates of base classifiers. The limitation of the current structure was discovered in its inability to lower the false detection rates of the base classifiers.

8.1 Future Work

In future research, it is intended to implement the PSL framework into a face detection system for which integration of a feature extraction component will be necessary. The goal is to compare the PSL framework with the face detection system which accompanies the OpenCV Computer Vision Library. In addition to that, a more rigorous comparison of the PSL framework will be pursued against other common machine learning framework such as Support Vector Machines and Neural Networks. The author also plans to make use of the WEKA (Witten and Frank, 2005) toolbench which has implemented and made available all current machine learning algorithms and will lend itself well to further performance comparisons involving the PSL framework.

The intention is to also pursue further experimentation and modifications on the IPSL framework. The author recognizes the need to not only test the robustness of the new framework with larger incremental datasets, but to also perform experiments on the IPSL structure which involve numerous, consecutive incremental training rounds on a single classifier, in order to observe the effects of long term incremental training on accuracy. Further modifications to the IPSL framework will involve its limitations regarding false detection rates. The revision of the IPSL framework will see an additional capability of the training structure to delete and replace existing nodes which contribute to increased false detection rates.

Appendices

Appendix A

ROC Graph Curves For IPSL on UCI Letter Dataset

Shown in this appendix are the ROC graph curves, generated for each class in the UCI Letter dataset. The graphs compare the generalization of the base PSL classifiers against the 10%, 20% and 30% increment classifiers. The ROC graph curves produced mixed results, but showed a clear trend. Out of the 26 different classes in the dataset, 73% experienced improvement as a result of IPSL training through either one or all of the 10%, 20% and 30% increment classifiers. The remaining 27% of the classes observed a measure of deterioration in the generalization through an introduction of higher false detection rates.

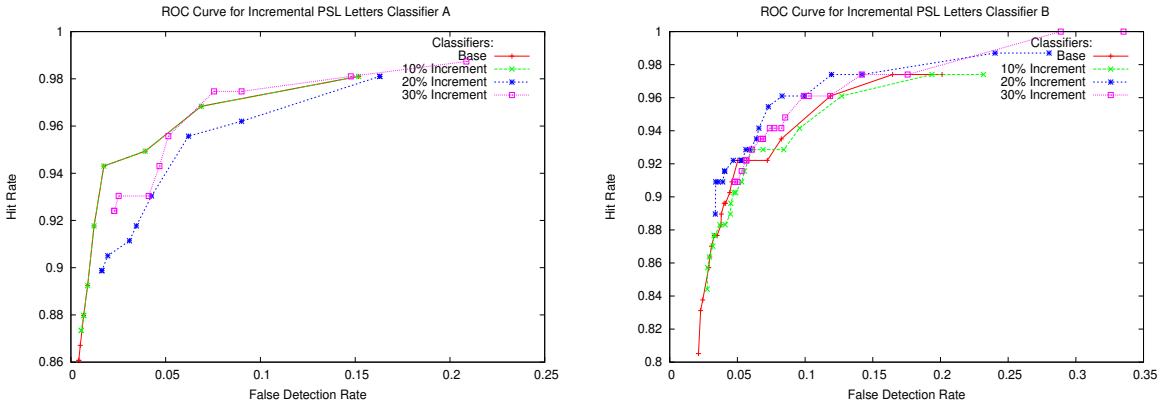

Figure A.1: ROC graph curves for the Letters dataset showing convergences of IPSL classifiers representing letters A-B.

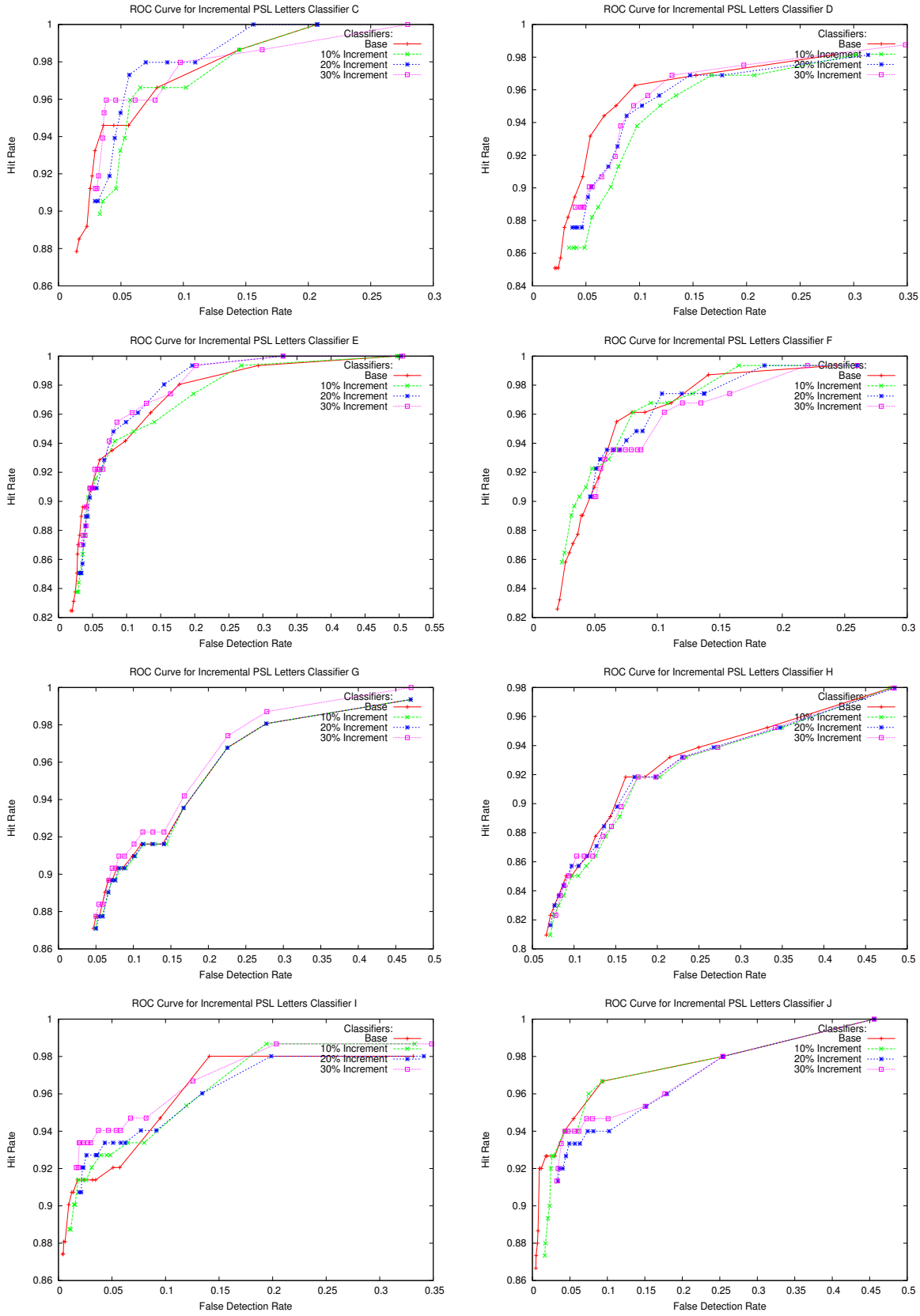

Figure A.2: ROC graph curves for the Letters dataset showing convergences of IPSL classifiers representing letters C-J.

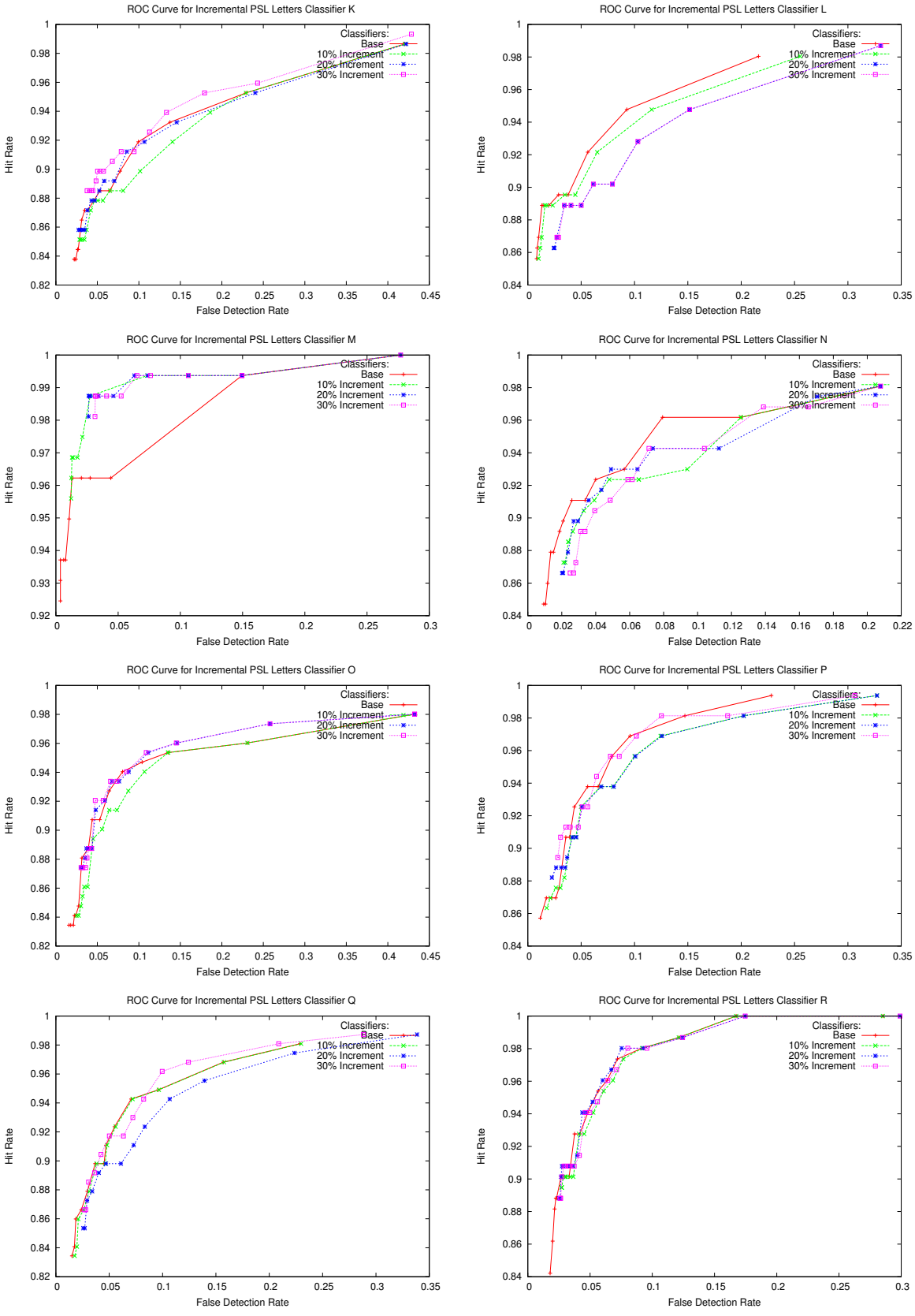

Figure A.3: ROC graph curves for the Letters dataset showing convergences of IPSL classifiers representing letters K-R.

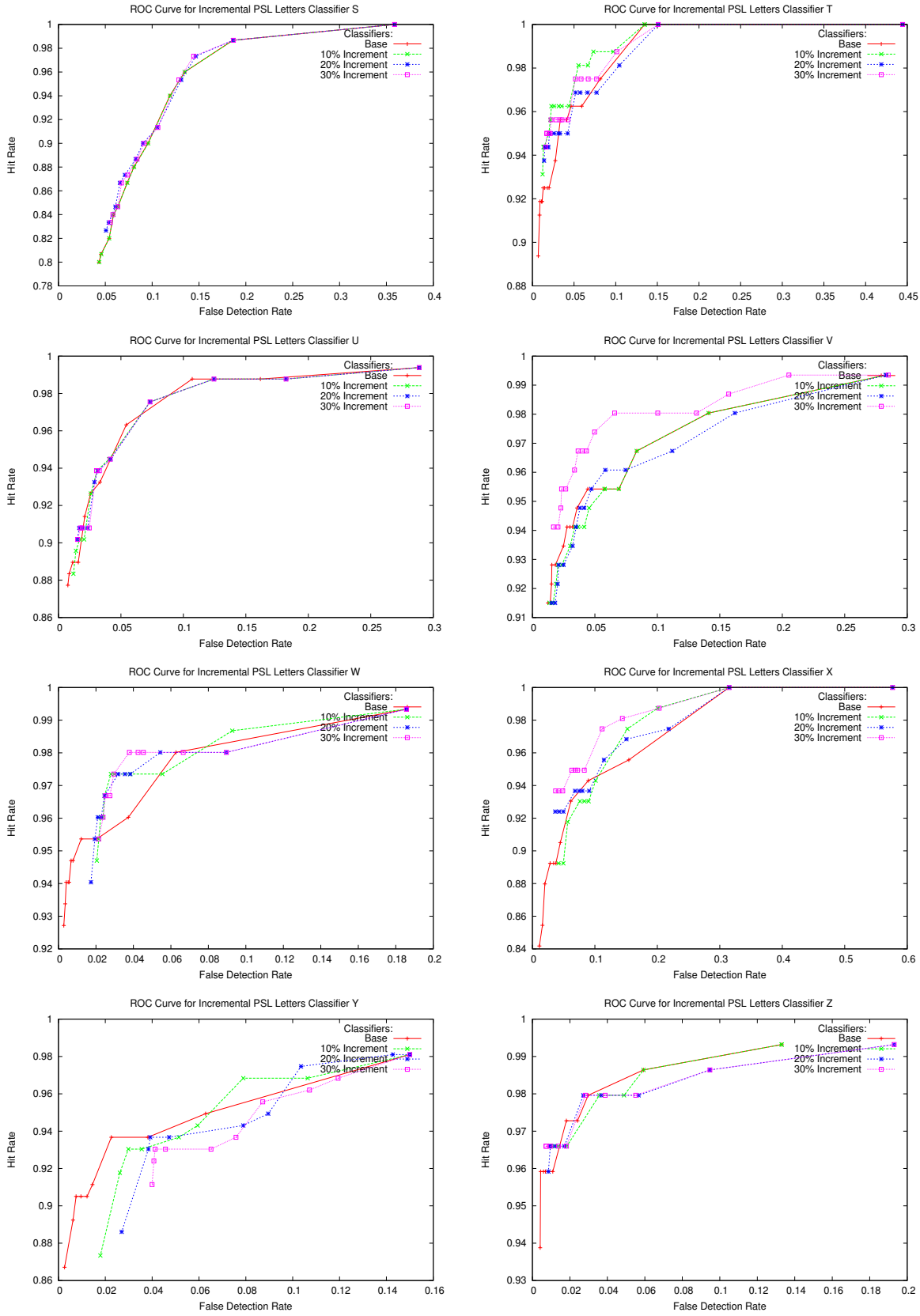

Figure A.4: ROC graph curves for the Letters dataset showing convergences of IPSL classifiers representing letters S-Z.

Bibliography

Allwein, E. L., Schapire, R. E. and Singer, Y. (2001), "Reducing multiclass to binary: a unifying approach for margin classifiers", *J. Mach. Learn. Res.* , Vol. 1, MIT Press, Cambridge, MA, USA, pp. 113–141.

Asuncion, A. and Newman, D. (2007), 'UCI machine learning repository', University of California, Irvine, School of Information and Computer Sciences. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
URL: *http://www.ics.uci.edu/∼mlearn/MLRepository.html*

Baluja, S., Sahami, M. and Rowley, H. (2004), Efficient face orientation discrimination, *in* 'Proc. International Conference on Image Processing ICIP'04', Vol. 1, pp. 589–592 Vol. 1.

Barczak, A. L. C., Johnson, M. J. and Messom, C. H. (2008), Empirical evaluation of a new structure for adaboost, *in* 'SAC '08: Proceedings of the 2008 ACM symposium on Applied computing', ACM, Fortaleza, Ceara, Brazil, pp. 1764 – 1765.

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Bourdev, L. and Brandt, J. (2005), "Robust object detection via soft cascade", *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , Vol. 2, pp. 236–243.

Brubaker, S. C., Mullin, M. D. and Rehg, J. M. (2006), "Towards optimal training of cascaded detectors", *Proc. of Computer Vision. ECCV 2006* , Vol. 3951, pp. 325–337.

Brubaker, S. C., Wu, J., Sun, J., Mullin, M. D. and Rehg, J. M. (2008), "On the design of cascades of boosted ensembles for face detection", *Int. J. Comput. Vision* , Vol. 77, Kluwer Academic Publishers, Hingham, MA, USA, pp. 65–86.

Bhlmann, P. and Yu, B. (2000), "Invited discussion on "additive logistic regression: a statistical view of boosting (friedman, hastie and tibshirani)".", *Annals of the Institute of Statistical Mathematics* , Vol. 52, pp. 287–315.

Dash, M. and Liu, H. (1997), "Feature selection for classification", *Intelligent Data Analysis* , Vol. 1, pp. 131–156.

Domingos, P. (1998), Occam's two razors: The sharp and the blunt, *in* 'In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining', AAAI Press, pp. 37–43.

Egan, J. P. (1975), *Signal Detection Theory and ROC Analisys*, Academic Press.

Fawcett, T. (2006), "An introduction to roc analysis", *Pattern Recogn. Lett.* , Vol. 27, Elsevier Science Inc., New York, NY, USA, pp. 861–874.

Fawcett, T. E. and Provost, F. (2002), "Fraud detection", Oxford University Press, Inc., New York, NY, USA, pp. 726–731.

Freund, Y. and Schapire, R. E. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, *in* 'EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory', Springer-Verlag, London, UK, pp. 23–37.

Freund, Y. and Schapire, R. E. (1999), "A short introduction to boosting", *Journal of Japanese Society for Artificial Intelligence* , Vol. 14, pp. 771–780.

Friedman, J., Hastie, T. and Tibshirani, R. (2000), "Additive logistic regression: a statistical view of boosting", *Annals of Statistics* , Vol. 28, p. 2000.

Gangardiwala, A. and Polikar, R. (2005), Dynamically weighted majority voting for incremental learning and comparison of three boosting based approaches, *in* 'IEEE International Joint Conference on Neural Networks', Vol. 2, Montreal, Canada, pp. 1131–1136.

Gonzalez, R. C. and Woods, R. E. (2002), *Digital Image Processing*, Prentice Hall.

Grossmann, E. (2004), *Structural, Syntactic, and Statistical Pattern Recognition*, Springer, chapter Automatic Design of Cascaded Classifiers, pp. 983 – 991.

Huang, C., Ai, H., Yamashita, T., Lao, S. and Kawade, M. (2007), Incremental learning of boosted face detector., *in* 'ICCV', IEEE, pp. 1–8.

Kulkarni, S., Lugosi, G. and Venkatesh, S. (1998), "Learning pattern classification-a survey", *Information Theory, IEEE Transactions on* , Vol. 44, pp. 2178–2206.

Le, D.-D. and Satoh, S. (2006), "Ent-boost: Boosting using entropy measure for robust object detection", *Pattern Recognition, International Conference on* , Vol. 2, IEEE Computer Society, Los Alamitos, CA, USA, pp. 602–605.

Li, S., Zhang, Z., Shum, H. and Zhang, H. (2002), Floatboost learning for classification, *in* 'Proceedings of The 16-th Annual Conference on Neural Information Processing Systems (NIPS)', Vancouver, Canada, pp. 9–14.

Lienhart, R., Kuranov, A. and Pisarevsky, V. (2003), Empirical analysis of detection cascades of boosted classifiers for rapid object detection, *in* 'DAGM03', Madgeburg, Germany, pp. 297–304.

Lienhart, R., Liang, L. and Kuranov, A. (2003), A detector tree of boosted classifiers for real-time object detection and tracking, *in* 'ICME2003', IEEE, pp. 277–280.

Lienhart, R. and Maydt, J. (2002), An extended set of haar-like features for rapid object detection, *in* 'ICIP02', Rochester, NY, pp. I: 900–903.

Liu, H., Hussain, F., Tan, C. L. and Dash, M. (2002), "Discretization: An enabling technique", *Data Mining and Knowledge Discovery* , Vol. 6, pp. 393–423.

Luo, H. (2005), "Optimization design of cascaded classifiers", *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* , Vol. 1, pp. 480–485 vol. 1.

Masnadi-Shirazi, H. and Vasconcelos, N. (2007), "High detection-rate cascades for real-time object detection", *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* , pp. 1–6.

McCane, B. and Novins, K. (2003), On training cascade face detectors, *in* 'Image and Vision Computing New Zealand', Palmerston North, pp. 239–244.

McCane, B., Novins, K. and Albert, M. (2005), Optimizing cascade classifiers. available at http://www.cs.otago.ac.nz/staffpriv/mccane/publications.html.

McCloskey, M. and Cohen, N. (1989), *The Psychology of Learning and Motivation*, Vol. 24, NY: Academic Press, chapter Catastrophic interference in connectionist networks: The sequential learning problem, pp. 109–164.

Mita, T., Kaneko, T. and Hori, O. (2005), Joint haar-like features for face detection, *in* '10th IEEE International Conference in Computer Vision (ICCV'05)', IEEE, pp. 1619–1626.

Oza, N. C. and Russell, S. (2001), Online bagging and boosting, *in* 'In Artificial Intelligence and Statistics 2001', Morgan Kaufmann, pp. 105–112.

Papageorgiou, C., Oren, M. and Poggio, T. (1998), A general framework for object detection, *in* 'International Conference on Computer Vision'.

Pham, M.-T. and Cham, T.-J. (2007), "Fast training and selection of haar features using statistics in boosting-based face detection", *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* , pp. 1–7.

Pudil, P., Ferri, F. J. and Kittler, J. (1994), Floating search methods for feature selection with nonmonotonic criterion functions, *in* 'In Proceedings of the Twelveth International Conference on Pattern Recognition, IAPR', pp. 279–283.

Roth, P., Grabner, H., Bischof, H., Skocaj, D. and Leonardist, A. (2005), "On-line conservative learning for person detection", *Visual Surveillance and Performance Evaluation of Tracking and Surveillance* , Vol. 0, IEEE Computer Society, Los Alamitos, CA, USA, pp. 223–230.

Schapire, R. E. (1999), A brief introduction to boosting, *in* 'Proc. of te 16th International Joint Conference in Artificial Intelligence'.

Schneiderman, H. and Kanade, T. (2000), A statistical method for 3d object detection applied to faces and cars, *in* 'IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2000)', pp. 1746–1759.

Shen, R., B. L. B. D. and Wang, Y. (2005), Gabor feature selection for face recognition using improved adaboost learning, *in* 'Advances in Biometric Person Authentication', Springer, pp. 39–49.

Sun, J., Rehg, J. and Bobick, A. (2004), "Automatic cascade training with perturbation bias", *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* , Vol. 2, pp. II–276–II–283 Vol.2.

Sung, K. and Poggio, T. (1998), "Example-based learning for view-based face detection", *IEEE Patt. Anal. Mach. Intell.* , Vol. 20.

Verschae, R., del Solar, J. R. and Correa, M. (2008), "A unified learning framework for object detection and classification using nested cascades of boosted classifiers", *Mach. Vision Appl.* , Vol. 19, Springer-Verlag New York, Inc., Secaucus, NJ, USA, pp. 85–103.

Viola, P. A. and Jones, M. J. (2001*a*), Fast and robust classification using asymmetric adaboost and a detector cascade, *in* T. G. Dietterich, S. Becker and Z. Ghahramani, eds, 'NIPS', pp. 1311–1318.

Viola, P. and Jones, M. (2001*b*), Robust real time object detection, *in* 'SCTV01', pp. xx–yy.

Viola, P. and Jones, M. (2004), "Robust real-time face detection", *International Journal of Computer Vision* , Vol. 57, pp. 137–154.

Wang, H., Divakaran, A., Vetro, A., Chang, S.-F. and Sun, H. (2003), "Survey of compressed-domain features used in audio-visual indexing and analysis", *J. Visual Communication and Image Representation* , Vol. 14, pp. 150–183.

Whitehill, J. and Omlin, C. W. (2006), Haar features for facs au recognition, *in* 'FGR '06: Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition', IEEE Computer Society, Washington, DC, USA, pp. 97–101.

Withopf, D., Withopf, D. and Jahne, B. (2007), Improved training algorithm for tree-like classifiers and its application to vehicle detection, *in* B. Jahne, ed., 'Proc. IEEE Intelligent Transportation Systems Conference ITSC 2007', pp. 642 – 647.

Witten, I. H. and Frank, E. (2005), *Data Mining: Practical machine learning tools and techniques*, 2nd edition edn, Morgan Kaufmann, San Francisco.

Wolpert, D. H. and Macready, W. G. (1997), "No free lunch theorems for optimization", *Evolutionary Computation, IEEE Transactions on* , Vol. 1, pp. 67–82.

Wolpert, D. H. and Macready, W. G. (2005), "Coevolutionary free lunches", *IEEE Trans. Evolutionary Computation* , Vol. 9, pp. 721–735.

Wu, B., Ai, H. and Liu, R. (2004), "Glasses detection by boosting simple wavelet features", *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* , Vol. 1, pp. 292–295 Vol.1.

Wu, J., Rehg, J. M. and Mullin, M. D. (2003), Learning a rare event detection cascade by direct feature selection, *in* 'NIPS (Advances in Neural Information Processing Systems) 2003', Vancouver, Canada.

Wu, Jianxin; Brubaker, S. C. M. M. D. R. J. M. (2008), "Fast asymmetric learning for cascade face detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence* , Vol. 30, pp. 369 – 382.

Xiao, R., Zhu, H., Sun, H. and Tang, X. (2007), "Dynamic cascades for face detection", *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* , pp. 1–8.

Xiao, R., Zhu, L. and Zhang, H.-J. (2003), Boosting chain learning for object detection, *in* 'ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision', IEEE Computer Society, Washington, DC, USA, p. 709.

Yilmaz, A., Javed, O. and Shah, M. (2006), "Object tracking: A survey", *ACM Comput. Surv.* , Vol. 38, ACM Press, New York, NY, USA.

Zhang, C. and Viola, P. (2007), Multiple-instance pruning for learning efficient cascade detectors, *in* 'NIPS 2007'.

Zhang, Z., Zhu, L., Li, S. Z. and Zhang, H. (2002), Real-time multi-view face detection, *in* 'FGR '02: Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition', IEEE Computer Society, Washington, DC, USA, p. 149.

Glossary

AdaBoost A classification algorithm that builds classifiers from smaller less accurate weak classifiers. 5

CoBE Cascades of boosted ensembles. A framework which constructs classifiers using boosting algorithms and arranges the weak classifiers in a layered structure. 3

FFS Forward feature selection. 32

FPR False positive rate. 21

IPSL Incremental Parallel Strong classifier within the same Layer. Training framework for incremental learning. 5, 49

OpenCV open source computer vision library 109

ROC Receiver operating characteristics. ROC graphs are a technique for visualizing and evaluating accuracy performances of classifiers. 38, 65

TPR True positive rate. 22

UCI University of California, Irvine. 58

accuracy Indicates the hit rates, false detection rates as well as the total error rates of a classifier 7

base classifier The initial classifier on top of which incremental training is performed using additional unique datasets. 39

binary-class classifiers Classifiers designed to match an input sample to only one of two possible class labels. 5

over-complete Situation in which the total number of information generated when extracting features is greater than the original amount of input low level data. 11

target object A sample instance of a positive set. The object which the detector is attempting to learn to recognize. 1

weak classifier/weak hypothesis A classifier that performs slightly better than 50% and is thus only slightly correlated with the true classification. 3

weak/base learner A method that extracts weak classifiers. 14