XCS-based versus UCS-based Feature Pattern Classification System

Toktam Ebadi
Victoria University of Wellington, New Zealand
toktam.ebadi,

Mengjie Zhang
Victoria University of Wellington, New Zealand
mengjie.zhang,

Will N. Browne
Victoria University of Wellington, New Zealand
will.browne@vuw.ac.nz

ABSTRACT

Extracting features from images is an important task in order to identify (classify) the patterns contained. The Evolutionary Computation and Reinforcement Learning technique of Learning Classifier Systems (LCSs) has been successfully applied to classification tasks, but rarely to image pattern classification due to the large search space associated with pixel data. Recently, a Feature Pattern Classification System (FPCS), utilising Haar-like features has been introduced with promising results in the image recognition domain. This system used a confusion-matrix to direct learning to hard to classify classes, but due to its reinforcement learning nature was required to estimate the ground truth. The novel work presented here adopts a supervised learning (UCS-based) approach into the FPCS framework. This work is compared with the original XCS-based system, updated to include the known ground-truth of the confusion matrix to aid comparison, albeit no longer reinforcement learning. Results on the 10 class MNIST numerical digits recognition task show that the XCS-based FPCS produces better classification due to its complete mapping guiding learning. However, results on the 26 class NIST character recognition task show that the UCS-based scales better as it does not require the complete mapping. The human readable rules produced by each system, coupled with the competitive classification performance compared with similar techniques, supports future work on both the XCS and UCS-based FPCS.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics-Based Machine Learning, Learning Classifier Systems

General Terms

Algorithms, Performance

Keywords

Learning Classifier Systems, Pattern Recognition

GECCO'12, July 7–11, 2012, Philadelphia, Pennsylvania, USA.

1. INTRODUCTION

Image classification and pattern recognition remain a major challenge for machine learning. This is mainly due to the large-dimensionality of image data. Identifying the 'best' features of an image that represent a class is extremely difficult due to spareness of the features. Moreover, the image domain requires techniques that are efficient in training and testing stages, and yet are able to produce generalised and accurate solutions.

This work builds on top of the Kukenys et al. Feature Pattern Classification System (FPCS) [9], which was based on Wilson's XCS framework [19]. FPCS employs LCS for image pattern classification in reinforcement and online scenarios. The FPCS utilises Haar-like features for extracting information from images to manage the large search space. FPCS demonstrated competitive, albeit not optimal, classification performance on handwritten numerical image recognition, with the benefit of human rotated rules and operation in online, dynamic scenarios.

The aim of this work is to adopt a supervised learning approach into FPCS and compare its performance with the reinforcement learning FPCS. A handwritten numerical dataset, MNIST, will be used to compare the performance of the two systems. In addition, the impact of confusion matrix and 'divide and conquer' approaches on image classification will be investigated. The knowledge in the confusion matrix will be used by the reinforcement and supervised FPCS to provide a guided search for similar classes of the problem. Finally, the scalability of the supervised method will be measured by applying it to an alpha-numerical handwritten dataset.

Traditional ensemble classification methods such as bagging and boosting [13] have been used to address ambiguity between different classes of a problem. However, such methods follow a bottom-up approach by creating separate classifiers for each sample. These classifiers are then integrated to create one single output. This leads to a large and complex network of knowledge. In contrast, the pattern recognition system developed in this work follows a top-down approach where an LCS is formed for the entire system. The LCS utilises a 'divide and conquer' for resolving confusions by constructing separate classifier systems for problematic classes. Therefore, the LCS-based system is more efficient for large and complex problem domains. Moreover, the information in the proposed LCS-based system is human readable which is useful in understanding of complex systems.

The structure of the paper is as follows. Section 2 provides the related work including learning classifier systems concept and its application in the image domain. It also describes

a sUpervised LCS known as UCS. Section 3 describes the encoding details of the Haar-like features for construction of rule conditions. Moreover, it explains the reinforcement learning Feature Pattern Classification System (FPCS) and sUpervised Feature Pattern Classification System (UFPCS). Section 4 provides the details of the confusion matrix approach for resolving confusion between similar classes. Section 5 provides the experiments and their details. Section 6 discusses the results, and finally section 7 provides the conclusion and future work.

2. BACKGROUND

2.1 Learning Classifier Systems

Learning Classifier Systems (LCSs) model an agent interacting with an environment using a set of sensors for observing and collecting information about the environment. Once the current state is observed, the agent performs an action and receives a numerical reward from the environment. An agent in LCS learns by attempting to maximize its amount of reward.

LCS, benefits from the integration of evolutionary computing, reinforcement learning or supervised learning and heuristics for creating adaptive systems. Most implementations of LCS use the XCS formulation [19]. XCS utilises reinforcement learning and uses an accuracy-based fitness to map the states of the environment and agent's actions to reward. In order to produce maximally general rules, XCS requires a complete mapping from each state to the entire action set (one rule for each possible action) during the training period. Non-optimal actions for a given state are mapped to work as the optimum actions. This leads to an extremely large population in domains encompassing a large number of possible actions. In contrast, supervised LCS-based frameworks may only create one rule for every state of the environment based on the available 'correct' action (since the the ground truth data is available).

2.2 Supervised Learning Classifier Systems

This work employs the sUpervised Classifier System (UCS) developed by Bernadó-Mansilla and Garrell-Guiu [2]. The UCS framework is designed for supervised problems where the 'correct' action is available during the training.

UCS and XCS share the same principle since both systems use a niche GA and define fitness based on accuracy. The agent in the UCS framework has two modes of operation: explore and exploit. Algorithms 1 is executed by the agent in the explore mode.

In this algorithm, initially the agent observes the current state of the environment, s, and forms a *match set* (loop in line 3). If the current action is not present in the *match set* covering is triggered to create a rule that its condition matches the current state. Since the correct action (a) is known in supervised scenarios, the UCS framework suggests forming a *correct set*, Cs, that contains all classifiers of the match set, M, that advocate action a. Each classifier in the UCS framework has an additional field, *correct track*, that is increased every time the classifier is selected as part of a *correct set*. Moreover, each classifier has an accuracy field which replaces the prediction in the XCS framework. A classifier accuracy is calculated as:

$$accuracy = \frac{correctTrack}{experience} \quad (1)$$

Algorithm 1: explore

```
Data: s: observed state;a: correct action; P:
      population of rules; M: match set; c:
      classifier ∈ P;Cs correct set
1  s=observe();
2  foreach c in P do
3      if c.condition.matches(s.condition) then
4          M.add(c);
5  if !M.contains(a) then
6      P.add(createClassifier(s,a));
7  foreach c in M do
8      if c.action== a then
9          Cs.add(c);
10 foreach c in Cs do
11     c.correctTrack++;
12 foreach c in M do
13     c.experience++;
14     c.accuracy = c.correctTrack / c.experience;
15     c.fitness= c.accuracy^ϵ
16     updateActionSetsize(M.sum(numerosity));
17 runGA();
```

Where experience is the number of times that a classifier has been selected as part of a *match set*. In UCS calculation of fitness is different from XCS. UCS calculates fitness according to equation 2.

$$fitness = accuracy^{accuracy-threshold} \quad (2)$$

The agent updates the *size*of its *match set* by having the sum of the numerosity of all the classifiers in the *match set*. At the end of the explore mode, the agent executes a Genetic Algorithm (GA) to evolve the population of classifiers. Two classifiers are selected from the *correct set* and two offspring are produced by applying crossover and mutation on their conditions, such that both offspring match the current state of the environment.

The agent may also execute subsumption. Subsumption, combines specific classifiers and creates more general, accurate classifiers. In addition, it may delete classifiers from the population in situations where the number of classifiers has exceeded the defined limit.

In the exploit mode, the agent does not perform any learning and predicts the associated class for each input state. The best action for each example is selected via system vote, which represents weighted fitness of all the classifiers in the *matchset*.

2.3 Related work

LCS has been used in various fields including gene expression classification [17] and data mining [15] but rarely applied to the image recognition domain due to the large search space of making the feature extraction difficult. LCS has been used to address automatic target recognition [7, 14] to identify objects of interest based on data collected from sensors in the robotics filed. Ravichandran et al. [14] employed an XCS-based LCS in such a domain. They define three types of features including global, spatial and spectral to extract important information of an image. The global features were defined based on Principal Component Anal-

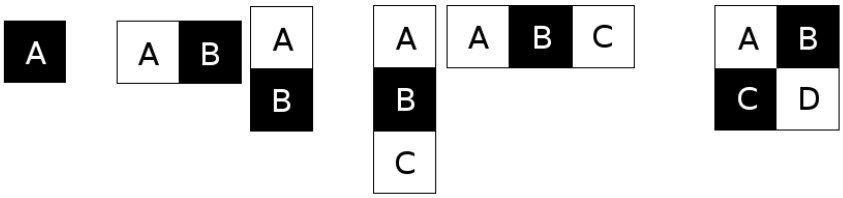

Figure 1: Haar-like features. The feature values are calculated by subtracting sums of pixel intensities in neighbouring rectangular regions A, B, C, D. Position and scale of features on images are important.

ysis (PCA) coefficients and represent the global information for an object. The spatial features captured region information, and spectral features represent the localised intensity based on dominant scatters in an image. These features were then encoded to an string that was used by the LCS for creating classifier rules. Frey and Slate [6] used LCS for handwritten letter recognition. The authors calculated 16 numerical attributes for creating each image condition where these values represent statistical features of the pixel distribution of images. The work of Frey and Slate was performed more that 20 years ago and their reported results were not promising.

All works described in this section use inflexible features that cannot be easily scaled to more sophisticated problems e.g., complex and large images.

3. CLASSIFICATION METHOD

3.1 Haar-like features

Traditional learning classifier systems use a 'don't care' symbol (#) in a classifier's condition in the ternary alphabet. The # symbol represents a field in the LCS condition that matches any attribute of the environment state. The # symbol enables LCS to produce generalised rules. Even so alphabets are not suited to encoding LCS conditions in the image domain since they are computationally expensive and intractable considering the large number of pixels in real images.

Haar-like features from the image domain are used to create rule conditions by extracting rectangular features from images and calculating the total pixel intensity in each rectangle. The final value is computed by measuring the difference between sums of the neighbouring rectangles. Figure 1 shows various types of Haar-like features used in both systems. The FPCS (and UFPCS) utilises the *integral image* proposed by [18]. When using the *integral image*, the value of each pixel in the image is replaced by sum of all pixel values to its left and above (equation 3). The *integral image*(II) is calculated once for each image.

$$II(x,y) = \sum_{i,j=1}^{x,y} I(i,j), \tag{3}$$

3.2 Encoding Haar-like features

Each Haar-like feature in the FPCS framework includes location $l = (x, y)$ and scale $u = (width, height)$ of the feature f. Thus the value of each feature $f(s, l, u)$ can be calculated with a few lookup calls in the integral image II.

In addition, each $f(s, l, u)$ includes a threshold between t_{low} and t_{high}. The threshold values enable the FPCS to form binary decision rules that identify whether the desired level of contrast between neighbouring rectangular regions exist. The rule condition in the FPCS can be constructed using the following formalism:

$$c_i = c(f, l, u, t_{low}, t_{high}),$$
$$c_i(s) = \begin{cases} \text{true, if } t_{low} < f(s,l,u) < t_{high} \\ \text{false, otherwise} \end{cases}$$

Although valuable information can be extracted from images using Haar-like features, a single feature is not sufficient for representing information required in complex and large images e.g., a single Haar feature on an image of digit '3' is not adequate for its identification. Therefore, we utilise a 'messy' encoding with appropriate operators [10], which allows for construction of complex features by joining multiple single features using a logical '*and*' operator. Such an approach creates comprehensive features and forms rule conditions that suit complex learning systems (equation 4).

$$c(s) = c_1(s) \wedge \ldots \wedge c_m(s) \tag{4}$$

3.3 Feature Pattern Classification System

The Feature Pattern Classification System (FPCS) employs Haar-like features, and is based on the XCS framework [19]. FPCS uses a prediction mechanism to predict a reward for a classifier, and a reinforcement learning to update agent's prediction. The classifier fitness in the FPCS is based on the accuracy of the predicted reward.

3.4 Supervised Feature Pattern Classification System

The sUpervised Feature Pattern Classification System (UFPCS) follows the same naming convention of the UCS. The UFPCS utilises supervised learning for the FPCS system based on the UCS framework [2]. The UFPCS framework performs covering in the explore mode if the selected action is not correct. This is to compensate for non-comprehensive representations of Haar-features since Haar-features do not contain information on the entire image.

4. CONFUSION MATRIX FOR DIVIDE AND CONQUER

Confusion matrices [8] are useful tools for identifying areas of confusion between correct (ground truth) and predicted classes in classifier systems. They facilitate resolving highly confused classes by providing quantifiable analysis of confusions. Such information can be used to create a guided learning with special focus on commonly confused classes.

By considering confusion matrices, pairs with the highest confusion values can be selected to resolve confusion. In

this work, we develop a two-level classifier system where the first level identifies the current state, and the second level resolves confusion based on the output of the first level. Such approaches are known as 'divide and conquer' or hierarchical systems in the field of machine learning [16].

4.1 Confusion matrix for FPCS

The information in the confusion matrix is not trivially available to the agent in the FPCS framework since the 'correct' action is not known by the agent. Therefore, the agent in the FPCS requires a mechanism that enables it to obtain such information (i.e., to estimate the correct state). In order to address this issue, the agent in the FPCS records a unique (*state,action,reward*) triplets for every state of the environment. Every time a new state is observed the agent selects the correct action by selecting the action with the highest reward for the state.

The 'divide and conquer' is implemented by considering the confusion matrix after the number of entries has reached a threshold. It picks pairs of actions ($action_A$, $action_B$) with the highest confusion error in the confusion matrix and sets up a separate classifier system for the confused classes.

4.2 Confusion matrix for UFPCS

UFPCS utilises a novel confusion matrix. Since the 'correct' action is known in the UFPCS, the confusion matrix is formed by comparing the system output against the actual class and recording the outcome. Therefore, the UFPCS is capable of finding confused classes by sorting the confusion matrix and forming separate classifiers for commonly confused classes.

5. EXPERIMENTS

5.1 Datasets

The UFPCS has been tested on MNIST and NIST datasets. Both datasets contain images of real world and thus provide a realistic test problem. In particular, MNIST has been widely used for evaluating performance of various methods on the handwritten-digit recognition problem, and serves as a standard for comparison between performance of different pattern recognition methods.

The MNIST benchmark [11] contains 60,000 training examples for handwritten digits: 0 ... 9. It contains binary image data of 28× 28 pixels. Note that no preprocessing is performed on the training set in our examples while preprocessing is known to improve the classification accuracy. In addition, MNIST provide 10,000 test images that are collected from a different group of people.

The NIST dataset contains alpha-numerical characters and provides total of 62 classes (10 classes for digits, 26 classes for uppercase and lowercase letters). It contains images of 32× 32 pixels. The data in the NIST is not in a common format. Therefore, it was converted to a binary format so it can be used for the classification task. Similar to MNIST, NIST has a separate independent test set.

5.2 Implementation details

Table 1 shows the parameters used in the experiments. These parameters are inherited from UCS project [2] except $\mu = 0.4$ which is higher than the UCS (0.04 in UCS) due to the variation in the Haar features. Six types of Haar-features presented in Figure 1 were used including a single rectangle sum, two, three and four rectangle difference. A horizontal and vertical features were considered for the two, three and four Haar types.

Table 1: Parameters used in the FPCS and UFPCS frameworks.

parameter name	symbol	UFPCS	FPCS
fitness fall-off	α	NA	0.1
accuracy threshold	ϵ_0	10	10
fitness exponent	ν	NA	5
learning rate	β	0.2	0.2
GA threshold	θ_{GA}	25	25
classifier threshold for deletion	θ_{del}	20	20
classifier threshold for subsumption	θ_{sub}	20	20
crossover probability	χ	0.8	0.8
mutation probability	μ	0.4	0.4

5.3 Comparing performance of the FPCS with UFPCS on MNIST dataset

The population size of rules is constrained to 60,000 classifier for the FPCS but it is decreased to 12,000 for the UFPCS as suggested in [2]. All the experiments are executed for 4,000,000 generations. In each generation a random image example is selected from the training set and presented as the current state to the LCS agent. All results presented here are averaged over 30 runs.

The FPCS and UFPCS were executed on the MNIST dataset for 4,000,000 generations. The FPCS and UFPCS required 15-20 and 2-6 hours receptively. The training time of the UFPCS is shorter than the FPCS due to lower number of rules in the population. The FPCS framework creates 10 (number of classes in the MNIST) rules when performing covering. However, the UFPCS only covers for the 'correct' known class resulting in a smaller population. Therefore, in UFPCS each example is compared against a relatively small population requiring less training time.

Figure 2 compares performance of the FPCS with UFPCS on the training data. The FPCS achieves 91% accuracy while the UFPCS reaches to 94% accuracy on the training set after 4,000,000 iterations. When testing the system using the test data, FPCS and UFPCS achieved 90±1% and 94±1% respectively.

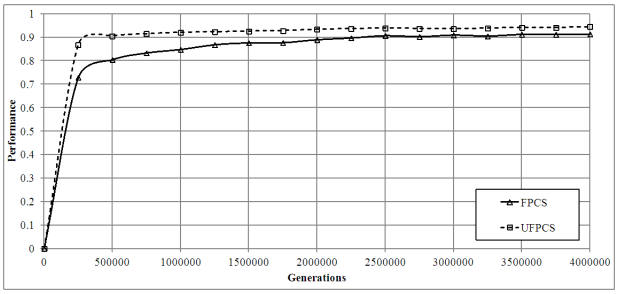

Figure 2: Comparison of accuracy rate of the FPCS and UFPCS on MNIST dataset.

Table 2 shows the confusion matrices recorded on the independent test set of the MNIST after 4,000,000 generations when applying FPCS. According to Table 2 the FPCS commonly confuses for example (3,8),(4,9),(2,7),(5,6). Note that

Table 2: Confusion matrix recorded on MNIST when applying FPCS. Rows correspond to system classification, columns correspond to actual classes (E estimated, A actual).

E\A	0	1	2	3	4	5	6	7	8	9
0	**96±0**	0±0	1±0	0±0	0±0	1±0	1±0	0±0	1±1	1±0
1	0±0	**98±0**	0±0	0±0	0±0	0±0	0±0	1±0	1±0	1±0
2	0±0	0±0	**90±1**	2±1	0±0	0±0	0±0	2±1	1±0	0±0
3	0±0	0±0	1±0	**89±2**	0±0	3±1	0±0	1±0	2±1	1±1
4	0±0	0±0	1±0	0±0	**94±1**	0±0	1±0	0±0	2±1	2±1
5	0±0	0±0	0±0	1±0	0±0	**86±2**	1±0	0±0	2±1	1±0
6	0±0	0±0	1±0	0±0	1±0	1±0	**94±1**	0±0	1±0	0±0
7	0±0	0±0	1±1	2±1	0±0	1±0	0±0	**91±1**	2±1	2±1
8	0±0	0±0	1±0	1±1	0±0	1±1	0±0	1±0	**86±4**	1±0
9	0±0	0±0	0±0	1±0	3±1	1±0	0±0	2±1	2±1	**88±1**

the confusion between the selected pairs is relatively symmetrical around the 'correct' diagonal e.g., FPCS estimates '9' when presented with a '4', and vice versa.

5.4 Confusion matrix guided learning

5.4.1 Divide and conquer in FPCS

In order to examine whether the knowledge captured in the confusion matrix (Table 2) can improve the performance of the FPCS framework, a simple 'divide and conquer' mechanism consisting of a two-level LCS is developed.

Figure 3 compares performance of the benchmark FPCS with the FPCS that utilises a 'divide and conquer' mechanism on the training set of the MNIST. It shows that using 'divide and conquer' improves the performance by 4±1% on the independent test set.

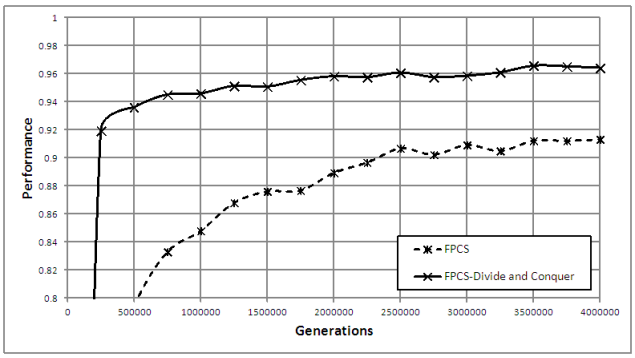

Figure 3: Comparison of accuracy rate of the benchmark FPCS and FPCS using 'divide and conquer' on MNIST dataset.

5.4.2 Divide and conquer in UFPCS

The divide and conquer is also applied to the UFPCS. The information in the confusion matrix can be made available to the agent. The agent ranks the confusion matrix and forms separate classifiers for classes that are commonly confused.

As figure 4 demonstrates, the 'divide and conquer' does not improve the performance in the UFPCS significantly. According to the confusion matrix in Table 3, clear confusion between classes is not present (except for (4,9)). For instance, when a digit '2' is presented to the system, the UFPCS highly confuses that with '7', although when a '7' is presented, the agent rarely estimates that as a '2'. Therefore, UFPCS produces less confusion but its performance is slightly better than the FPCS.

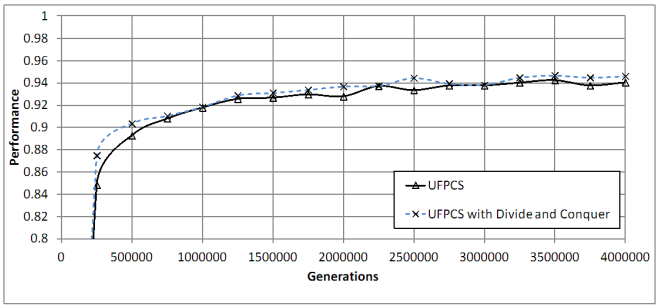

Figure 4: Comparison of accuracy rate of UFPCS and UFPCS using 'divide and conquer' on MNIST dataset.

5.5 Comparing performance of the UFPCS on NIST dataset

The NIST dataset contains 208,363 training examples for uppercase letters, 178,997 training example for lowercase letters, and 60,089 example for digits. The test set includes 11,941 uppercase, 12,000 lowercase and 5,646 digit examples. We created separate datasets for uppercase, lowercase and digits and trained the system separately on each file to limit the number of classes seen by the LCS.

5.5.1 FPCS performance on NIST

Applying FPCS to problems with a large number of classes is not efficient due to a need for very large population size. The FPCS required 60,000 rules to be trained properly for only 10 digit classes of the MNIST dataset. When considering problems with a larger number of classes, e.g., uppercase or lowercase letters, the population size must be increased to a very large number due to correct and incorrect rules e.g., 350,000 to accommodate rules required for different classes in addition to different niches within a class. Figure 5 shows examples of 'a' and 'g' from the NIST lowercase data file.

Several experiments were executed using the FPCS on lowercase and uppercase training set of the NIST. These experiments have been run for two weeks (on modern desktop PCs over a university grid) without convergence in the

Table 3: Confusion matrix recorded on MNIST when applying UFPCS. Rows correspond to system classification, columns correspond to actual classes (E estimated, A actual).

E\A	0	1	2	3	4	5	6	7	8	9
0	**98±0**	0±0	0±0	0±0	0±0	0±0	0±0	0±0	1±1	0±0
1	0±0	**99±0**	0±0	0±0	0±0	0±0	0±0	1±0	0±0	0±0
2	0±0	0±0	**94±1**	1±0	1±0	0±0	0±0	1±0	0±0	0±0
3	0±0	0±0	1±1	**91±1**	1±0	1±0	0±0	1±0	1±0	0±0
4	0±0	0±0	0±0	0±0	**94±1**	0±0	1±0	0±0	0±0	1±1
5	1±0	0±0	0±0	3±1	0±0	**92±2**	1±0	2±1	0±0	0±0
6	1±1	0±0	0±0	0±0	1±0	0±0	**94±1**	0±0	0±0	0±0
7	0±0	1±0	3±1	0±0	0±0	0±0	0±0	**91±2**	0±0	0±0
8	1±0	0±0	0±0	1±0	2±1	1±0	1±0	3±1	**86±4**	1 ±1
9	1±0	1±1	0±0	1±0	3±1	1±0	0±0	2±1	1±0	**89±2**

Figure 5: Examples of different ways in which letters 'a' and 'g' can be written.

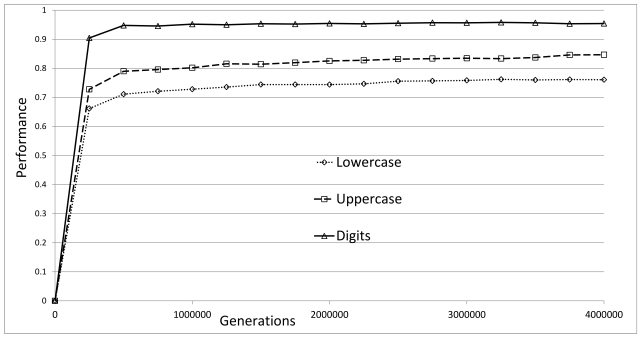

Figure 6: Comparison between accuracy rate of uppercase and lowercase letters, and digits in the NIST.

performance of the system, which was considered impractical. This questions efficiency of FPCS in the image domain for problems with a relatively high number of classes.

5.5.2 *UFPCS performance on NIST*

Due to JVM heap size memory limit on windows, training was performed on separate uppercase and lowercase files where each file was divided into four parts. The UFPCS was trained for 1,000,000 generations, 12,000 population, on each part before loading the next part of the data. After running the initial set of experiments, it was observed that the system was only able to sustain rules produced for the last part of the training. The deletion mechanism of the UCS removes rules that have not been used for a long period of time. This is mainly due to the fact that in the NIST dataset all examples of the same class are presented together e.g., all the examples of letter 'A' are presented in the first partition of the dataset and are not repeated again. These results confirm Butz and Sigaud 's finding [3] that deletion method of accuracy-based classifier systems do not suit problems where samples are non-uniformly distributed.

A sub-sampling was performed on each of the NIST uppercase and lowercase files. The examples were randomly selected from the original dataset. The number of examples in all the classes were approximately the same (1500 for each class). Figure 6 shows the results of the UFPCS when applied to uppercase, lowercase, and digits on the NIST (separately).

As Figure 6 depicts, digits accuracy rate is 95%, uppercase letters 83% and lowercase letters 76%. The rules produced by these systems were applied to two test sets. The first test set was formed by randomly selecting 10,000 examples from the training set, and the second test set was formed by using the hsf_4 dataset of NIST which contains unseen examples. Table 4 shows the accuracy of the UFPCS on NIST using the two test sets.

The difference between accuracy rate on training and testing of lowercase and uppercase classes on the test set (hsf_4) is due to lack of presence of some examples in the training set.

Table 4: Performance accuracy of UFPCS on the NIST dataset.

dataset	Training	Subsampled data	hsf_4
Digits	95%	95%	90%
Uppercase	83%	84%	78%
Lowercase	76%	76%	67%

6. DISCUSSION

The sUpervised Feature Pattern Classification System (UFPCS) achieves higher accuracy compared with the XCS-base, reinforcement learning, Feature Pattern Classification system (FPCS). The improvement in the accuracy of UFPCS is due to availability of 'correct' actions during the training phase that leads to construction of correct rules in the population.

X:28 N:2 F:0.3220 A:0.8929 HAAR2VER: 18,8-11x10 in (-10534.24, 49.95) && HAAR2HOR: 7,6-10x16 in (-3061.00,23848.10) && HAAR2VER: 6,6-4x16 in (10328.84,-2102.39) -> g

X:57 N:2 F:0.4831 A:0.9298 HAAR4: 10,0-8x18 in (-16830.00,-5610.00) && HAAR2VER: 0,5-30x26 in (9052.00,27157.00) -> a

Figure 7: Examples of character images and learnt matching Haar-like features. The classifier learnt for each character contains the Haar-based condition and action, as well as some statistics that allow humans to interpret rules.

Confusion matrices were employed to identify highly confused classes of the problem. The information in the confusion matrix facilitated construction of a hierarchical classifier system by utilising a 'divide and conquer' mechanism. The 'divide and conquer' improved the performance of the FPCS. The FPCS create a complete mapping between input data and all the possible classes of the problem thus introducing a large amount of confusion into the system. The reinforcement learning of the FPCS is in charge of improving the fitness of the correct rules while the deletion mechanism removes rules that have high rate of providing a wrong estimate. Since the 'correct' action is not known in the FPCS, the number of rules with incorrect actions in the system is high and there may be some cases that the reinforcement and deletion mechanism of the XCS do not remove such rules thus increasing the chance of confusion in the system. The UFPCS did not highly benefit from the 'divide and conquer'. It was observed that there was lack of clear confusion between classes when using UFPCS. This can be due to low rate of initial confusion in the system due to adding rules that advocate 'correct' actions.

Table 5: Number of rules for each class of the problem in the MNIST dataset.

Class	**0**	**1**	**2**	**3**	**4**
Number of rules	3476	3419	3623	3687	3123
Class	**5**	**6**	**7**	**8**	**9**
Number of rules	4063	4751	3664	2581	4005

By considering knowledge of the confusion matrix in Table 3, it can be inferred that the supervised method has provided stronger rules for certain classes of the problem for instance, '6' since whenever a '0' is presented, it frequently estimates that as a '6' while it never estimates '6' for a '0'. Table 5 shows the number of rules in each class when using UFPCS without having the 'divide and conquer' implementation. The UFPCS has produced 3,476 rules for class '0' and 4,751 rules for '6'.

Ensemble approaches have achieved high accuracy on the MNIST dataset in general. Convolutional networks [5] and Deep Belief Networks (DBN) [12] are example of such methods and have achieved 99.52% and 99.2% accuracy rate on the MNIST respectively (note that the authors of [5] have performed normalisation on the data). The best results on MNIST using Haar features was achieved by Casagrande [4] (98.69%).

NIST has not been widely used in the literature. The best results on the NIST is 83% and is reported in [1]. Please note that 83% accuracy is achieved on all of the 62 classes of the NIST dataset. The results reported in this paper were achieved without performing any form of preprocessing on the MNIST and NIST datasets.

6.1 Rule transparency

Figure 7 shows an example of the sample rules produced by the UFPCS for letters 'g' and 'a'. Note the some classifier conditions are intuitively interpretable while others are harder to interpret. The FPCS and UFPCS construct some statistical values for each classifier based on its performance for instance, the rule that corresponds to the letter 'g' in Figure 7 has been used 28 times (experience:X), and has a numerosity of 2 (N) meaning that the system has produced this rule twice. This rule has a relatively low fitness (F) and high accuracy (A). The HAAR values represent different types of Haar features. The rules for the letter 'a' can be interpreted similarly.

7. CONCLUSIONS AND FUTURE WORK

This work compared an XCS-based FPCS with an UCS-based (UFPCS) for image classification. Performance on the well-known MNIST dataset for numerical recognition (10 classes) was competitive when compared with other approaches that used Haar-like features. However, non-human readable approaches have achieved higher classification accuracies (e.g., [5, 12]).

In order to improve the classification accuracy performance the concept of confusion matrices to guide a divide and conquer approach was introduced into both systems. The complete map philosophy of FPCS led to a symmetric matrix, which enabled sub-LCS to resolve confused states to a degree (91±1% improved to 95±1%). Note that this approach requires the ground truth to be available to form the known confusion matrix, so is not pure reinforcement learn-

ing. Importantly, this work shows that the confusion matrix for UFPCS was asymmetric, so could not easily be used to guide a divide and conquer approach similar improvements (albeit performance did not suffer).

The complete map approach did not scale well to the character recognition task, due to the number of classes (26 classes per set) requiring multiple rules for both correct and incorrect action mapping. Analysis of the domain also showed that niches were present within a class further increasing the search space for rules. In contrast UFPCS performed well on these datasets (NIST uppercase and lowercase) as it did not attempt to form a complete mapping, only allocating the necessary rules for each class.

In domains with a small number of classes, the XCS-based system appears promising as the complete mapping highlights confusions, so can guided divide and conquer learning. As the number of classes increases, the number of required rules increases substantially, supporting the use of UCS-based systems.

Our future work will consider increasing the number of classes to 62 and test the performance of the UFPCS on the NIST dataset. In addition, we attempt to improve the accuracy of the system by replacing Haar-like features with more high-level features (e.g., features that capture curves and lines) that mimic biological pattern recognition mechanisms.

8. REFERENCES

[1] Y. Bengio, F. Bastien, A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache, et al. Deep learners benefit more from out-of-distribution examples. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 164–172. Journal of Machine Learning Research, 2011.

[2] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evol. Comput.*, 11:209–238, September 2003.

[3] M. V. Butz and O. Sigaud. Xcsf with local deletion: preventing detrimental forgetting. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 383–390, New York, NY, USA, 2011. ACM.

[4] N. Casagrande. Automatic music classification using boosting algorithms and auditory features. *Computer and operational research Department, University of Montreal, Montreal, Master Thesis*, 2005.

[5] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In T. Walsh, editor, *International Joint Conference on Artificial Intelligence*, pages 1237–1242. IJCAI/AAAI, 2011.

[6] P. Frey and D. Slate. Letter Recognition Using Holland-style Adaptive Classifiers. *Machine Learning*, 6(2):161–182, 1991.

[7] A. Gandhe, S.-H. Yu, R. Mehra, and R. E. Smith. Fused, multi-spectral automatic target recognition with xcs. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1874–1874, New York, NY, USA, 2007. ACM.

[8] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30(2):271–274, 1998.

[9] I. Kukenys, W. N. Browne, and M. Zhang. Transparent, online image pattern classification using a learning classifier system. In *EvoApplications (1)*, pages 183–193, 2011.

[10] P. L. Lanzi and A. Perrucci. Extending the representation of classifier conditions part ii: From messy coding to s-expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352, Orlando, Florida, USA, July 1999. Morgan Kaufmann.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, December 1989.

[12] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[13] J. Lu, K. Plataniotis, A. Venetsanopoulos, and S. Li. Ensemble-based discriminant learning with boosting for face recognition. *Neural Networks, IEEE Transactions on*, 17(1):166 –178, January 2006.

[14] B. Ravichandran, A. Gandhe, R. Smith, and R. Mehra. Robust Automatic Target Recognition Using Learning Classifier Systems. *Information Fusion*, 8(3):252–265, 2007.

[15] M. F. Santos, W. Mathew, T. Kovacs, and H. Santos. Supervised learning classifier systems for grid data mining. In *Proceedings of the international conference on Computational and information science 2009*, pages 416–424, Stevens Point, Wisconsin, USA, 2009. World Scientific and Engineering Academy and Society (WSEAS).

[16] R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, PPoPP '01, pages 34–43, New York, NY, USA, 2001. ACM.

[17] C. Vecchiola, M. Abedini, M. Kirley, X. Chu, and R. Buyya. Gene expression classification with a novel coevolutionary based learning classifier system on public clouds. In *e-Science Workshops, 2010 Sixth IEEE International Conference on*, pages 92 –97, December 2010.

[18] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages 511–518, 2001.

[19] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3:149–175, June 1995.