Algorithm Evolution for Face Recognition:
What Makes a Picture Difficult

Astro Teller and Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
astro@cs.cmu.edu   mmv@cs.cmu.edu

ABSTRACT

One of the classic problems in computer vision is the face recognition problem. In general this problem can take on a wide variety of forms, but the most common face recognition problem is "Who is this a picture of?" Evolutionary computation has, in the past, been applied indirectly to this problem through techniques like learning Neural Networks. This paper introduces a Genetic Programming style approach to learning algorithms that directly investigate face images and are coordinated into a face recognition system. Through a series of experiments, we will show that evolved algorithms can accomplish the face recognition task. We will also highlight several pitfalls and misconceptions surrounding face recognition as a learning problem.

1. Introduction

Face recognition has been recognized for years now as an important task in computer vision and an excellent area in which machine learning can participate. In the past, works like [2] and have used evolutionary computation to help another learning technique learn face recognition. The purpose of this paper is three fold. First, we aim to show that genetic programming can be used to learn algorithms that directly discriminate among face images for a face recognition application. Second, we will show that images that appear to be extremely difficult to classify, can have underlying and unwanted simplicity. And third, we will suggest, through the fourth of our four experiments, that the traditional method of removing this unwanted underlying simplicity can also remove part of the interest of the problem.

These goals stem from two basic beliefs of ours. The first is that computer vision can, in many cases, benefit from the aid of machine learning. We believe this both because machine learning has made significant advances in computer vision problems (e.g., [7]) and because there seem to be more problems with more task specific details than humans will ever be able to understand and write individual specific programs to conquer. Filling in such non-intuitive details is exactly the sort of way machine learning can be of help to the field of computer vision. Our second belief is that machine learning, especially in the service of computer vision, has not yet well explored a wide variety of learning techniques on problems where the learning ramifications are well understood. For example, the overwhelming majority of past work in learned face recognition has been images cropped to just the face and recognized by a Neural Network. Both of these choices, while not wrong, have other options. These two beliefs gave birth to this paper subject: the application of a GP-variant called PADO to a face recognition study across several related databases.

This paper reports on experiments using a relatively new database of face images developed at CMU. Face databases such as the Olivetti, Usenix, or MIT image sets are publicly available and have all been the topic of multiple papers. A good mosaic address for getting both the data and related papers is *http://www.cs.rug.nl/~peterkr/FACE/face.html*. In our work, none of these "older" databases were used because the faces they contain have a restricted set of head movements and facial expressions. The new CMU face images have a broader range. In addition, most of the "older" face databases need to be preprocessed in order to make appropriate learning experiments. This point will be discussed more in Section 4. For the rest of this paper, the MIT face database will be used as the dominant example of an established database. It was chosen for this purpose only because it has probably been the heaviest source

of face recognition papers in the last 5 years.

Our face recognition work is done in our PADO learning architecture. PADO (Parallel Algorithm Discovery and Orchestration) is a learning architecture that incorporates a form of Genetic Programming(GP) [11]. Section 2 describes PADO's process of algorithm evolution and algorithm orchestration for signal understanding. There have been some examples of GP applied to bitmaps (usually font bitmaps) in order to do classification (e.g.,[1, 5]). In between bitmaps and full resolution images are projects like [4] that applied GP to a subset of black and white silhouettes of a person and tried to learn where one of the hands was. Even in the domain of full video, learned aids to object recognition can be seen in works like [6] and [9].

This paper is organized as follows: Section 2 sketch PADO's process of algorithm evolution and orchestration. Section 3 gives a few details on the language in which PADO algorithms are evolved. Section 4 proceeds through a series of 4 experiments using PADO to accomplish face recognition in full video images. Finally, Section 5 draws conclusions.

2. The PADO Architecture

Pictures of faces are image signals. Classification can be done by algorithms that analyze signals. PADO learns these algorithms through evolution. The purpose of this section is to provide formal explanation to support the following experiments. We will first sketch the PADO architecture and its extension of genetic programming (GP). Then, Section 3 will detail exactly what kind of programs PADO is evolving and what access these programs have to the experimental signals.

The goal of the PADO architecture is to learn to take signals as input and output correct class labels. When there are $\mathcal{C}$ classes to choose from, PADO starts by learning $\mathcal{C}$ different "systems". System$_I$ is responsible for taking a signal as input and returning a confidence that class $\mathcal{I}$ is the correct label. System$_\mathcal{I}$ is built out of several programs learned by PADO. Each of these programs does exactly what the system as a whole does: it takes a signal as input and returns a confidence value that label $\mathcal{I}$ is the correct label. PADO performs object recognition by orchestrating the responses of the $\mathcal{S}$ programs within each system and then the $\mathcal{C}$ systems. System$_\mathcal{I}$ is built from the $\mathcal{S}$ programs that best (based on the training results) learned to recognize the instances of class $\mathcal{I}$. The $\mathcal{S}$ responses that the $\mathcal{S}$ programs return on seeing a particular image are all weighted and their weighted average of responses is interpreted as the confidence

that System$_\mathcal{I}$ has that the signal in question contains an object from class $\mathcal{I}$. How these orchestration weights (also used to orchestrate the $\mathcal{C}$ systems) are actually changed and used can be seen in [11]. Figure 1 summarizes the main functionality of PADO's evolutionary learning of signal understanding algorithms.

```
function PADO(Pop, signals, C, S)
  inputs:
     Pop, a set of P randomly generated algorithms
     signals, a set of training signals
     C, the number of classes
     S, # of algorithms from each class to return
  Repeat
    Loop over signals
          EvaluateFitness(Pop,signal_i)
    Split Pop into C distinct subpools of size P/C
                                   based on fitness
    Loop i from 1 to C
          MatingPool_i ← Reproduce(SubPool_i)
          NewSubPop_i ← Recombine(MatingPool_i)
       population ← ΣNewSubPop_i
  Until return requested
  return the most fit S algorithms from each subpool
```

Fig. 1: PADO's algorithm evolution learning process.

PADO evolves programs in a PADO-specific graph structured language. At the beginning of a learning session, the main population is filled with $\mathcal{P}$ programs that have been randomly generated using a grammar for the legal syntax of the language. All programs in this language are constrained by the syntax to return a number that is interpreted as a confidence value between some minimum confidence and some maximum confidence. Crossover and mutation in PADO are more complicated than their standard forms in genetic algorithms or GP. Both the crossover and mutation operators are "SMART" operators that are co-evolved with the main population, as we describe in [10, 11].

3. PADO Program Language

Figure 2 sketches the structure of a PADO program. Each program is constructed as an arbitrary directed graph of nodes. As an arbitrary directed graph of N nodes, each node can have as many as N outgoing *arcs*. These arcs indicate possible flows of control in the program. In a PADO program each node has two main parts: an *action* and a *branch-decision*. Each program has an *implicit* stack and an indexed memory. All *actions* pop their inputs from

this implicit stack and push their result back onto the implicit stack. These actions are the equivalent of GP's terminals and non-terminals. For example, the *action* "6" simply pushes 6 onto the parameter stack. The *action* "Write" pops *arg1* and *arg2* off the stack and writes *arg1* into Memory[*arg2*] after pushing Memory[*arg2*] onto the stack. Evaluating a GP tree is effectively a post-order traversal of the tree. Because there are many arcs coming into a particular node in the PADO language we evaluate a part of the graph (indeed, the whole graph) as a *chronological*, not structural, post-order traversal of the graph.
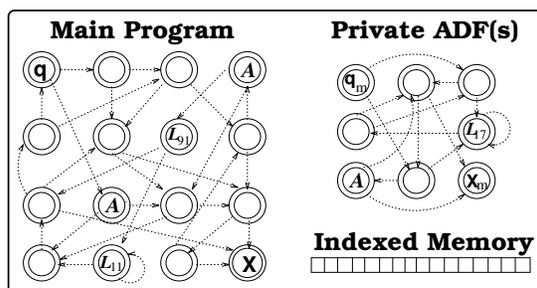

Fig. 2: The general structure of a new PADO program.

After the action at node i is executed, an arc is taken to a new node. The branch-decision function at the current node makes this decision. Each node has its own branch-decision function that may use the stack top, the temporally previous node action type, the memory, and constants to pick an arc.

There are several special nodes shown in Figure 2. Node q is the start node. It is special in no other way than it is always the first node to be executed when a program begins. Node X is the stop node. When this node is reached, its action is executed and then the program halts. When a program halts or is halted at the time-threshold, its response is considered to be the current value residing in some particular memory location (e.g., response = Memory[0]). If a program halts sooner than a preset time threshold, it is started again at its start node (without erasing its memory or stack) to give it a chance to revise its confidence value. A weighted average of the responses that the program gives on a particular execution is computed and interpreted as the answer.

Node A executes the private *ADF* program (starting at q_m) as its action. It then executes its branch-decision function as normal. The *ADF* programs associated with each *Main* program bear similarity to the concept of ADF's (automatically defined functions) [5]. However, PADO ADFs do not take a specific number of arguments but evolve to use what it they need from the incoming argument stack. In addition, they have internal loops and recursion.

Here is a brief summary of the language primitives and their effects:

Algebraic Primitives: {+ - * / NOT MAX MIN}
Memory Primitives: {READ WRITE}
Branching Primitives: {IF-THEN-ELSE PIFTE}
PIFTE is a probabilistic conditional that takes 3 arguments. A random number is chosen and if it is less than *arg1* then *arg2* is returned, else *arg3* is returned.
Signal Primitives: {PIXEL, LEAST, MOST, AVERAGE, VARIANCE, DIFFERENCE}
These are the language functions that can access the signals. In order to demonstrate PADO's power and flexibility, these same primitives were used for both image and sound data [12]. PIXEL returns the intensity value at that point in the image (or sound). The other five "signal functions" each pop the top four values off the stack. These four numbers are interpreted as (X1,Y1) and (X2,Y2) specifying a rectangle in the image. If the points specify a negative area then the opposite interpretation was taken and the function was applied to the positive area rectangle. LEAST, MOST, AVERAGE, VARIANCE, and DIFFERENCE return the respective functions applied to that region in the image. DIFFERENCE is the difference between the average values along the first and second half of the line (X1,Y1,X2,Y2).
Routine Primitives: {ADF LIBRARY[i]}
These are programs that can be called from the Main program. In addition, they may be called from each other. Because they are programs, and not simple functions, the effect they will have on the implicit stack and memory before completion (if they ever stop) is unknown. In particular, because both routine types are arbitrarily complex programs, each has an arbitrary number of arguments.

4. The Experiments

The images used in these experiments were taken from a new CMU face database set, created under the supervision of Tom Mitchell. These images are head and shoulder shots of 20 different people in a natural (i.e. extremely cluttered) office background. The database has 28 images of each person. So there are at total of 560 256x256 eight bit greyscale images in this database. Each set of 28 images has a wide range of facial expressions, head positions, and head rotations. This image variance range is, in fact, wider than the popular face databases such as the Olivetti and the MIT face sets that the following experimental databases are loosely compared against. Figures 3,

5, 6, and 7 show example face images from both the training and testing sets for the respective four experiments. As these figures show, each image contains exactly one person. The more difficult problem of multiple faces in a single image, while an important part of face recognition, is not the focus of this work.

As was just mentioned, the CMU faces database contains 28 images each for 20 different people. For each class (person) half the images were set aside as a training set and the other half were saved for testing. These sets were chosen randomly. In fact, of the 14 images of each person that were saved for testing, 4 were used for tuning the orchestration weights and the other 10 were used to actually measure PADO's generalization to completely unseen examples.

A population of 2800 PADO programs were used and each program was given 30 milliseconds to examine the training image and produce a confidence on which its fitness was then based. During each generation, the population is organized into C subpools, one for each of the C face classes. The confidence returned by a program X member of subpool I examining image U is the confidence X has that U is a member of class I. Each of the 5 runs that was done for each of the four experiments described in this paper was allowed to continue until generation 150. A single run (including testing at the end of each generation) took 2 days on a DECstation5000/20. At the end of each generation, the seven best algorithms from each of the 20 subpools were extracted and orchestrated into a complete PADO system which was then shown 200 images it had never seen before.

Experiment A

Initially, we tried PADO out on the unaltered CMU face database. Figure 3 shows two training and two testing example images (out of 560).

Figure 4 shows the percentage of **test** images that the orchestrated PADO system of most fit algorithms correctly classifies at the end of each generation. The four curves in Figure 4 refer to the four different experiments we report in this paper. The experiments are referred to as experiments **A**, **B**, **C**, and **D** and the four curves are marked accordingly. For example, curve **A** is the results we obtained in experiment **A**, using the untampered images.

Curve **A** in Figure 4 shows that by generation 150 PADO has learned to discriminate between the unseen face images with about 92% accuracy. These results are better than the only others currently reported on these images [8]. In fact, this 92% recognition rate is comparable to the best rates reported on the MIT face database which is, *to the causal observer*, a simpler set of faces to distinguish [3].
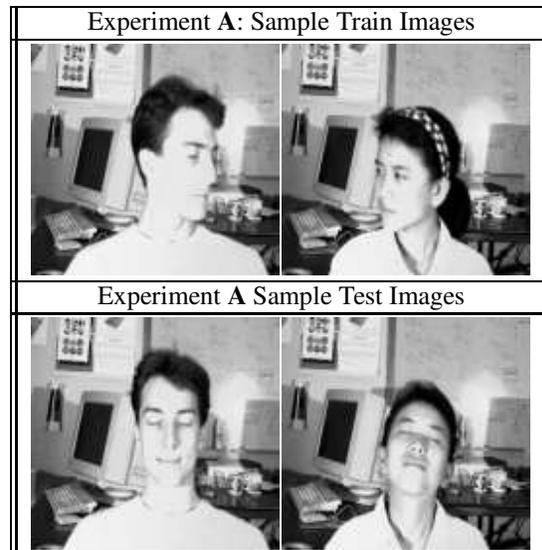

Experiment **A**: Sample Train Images

Experiment **A** Sample Test Images

Fig. 3: Random train and test images.

Experiment B

It is a well known evolutionary computation maxim that simple solutions are often evolved for hard-*looking* problems. In particular, many of the publicly available face databases suffer from this problem [3]. In order to test whether the same inadvertent simplicity occurs in the CMU face database, our second experiment involved reusing the same database but placing a black mask over the central half of each image (see Figure 5). Because faces vary so widely about the images, a portion of a face can occasionally be seen outside the black square. In general, however, this experiment tests how PADO learns to perform without the feature that was supposed to uniquely determine the image class.

As we can see from curve **B** of Figure 4, surprisingly, the faces do not matter much. By generation 150, the face recognition rate in this experiment is approximately 89%, only 3% lower than when the faces were included. These findings for the CMU face database are analogous to similar discoveries about other face databases in which the face is an unnecessary feature, (e.g., the MIT face set) [3].

Experiment C

The next logical step is one that many other experimenters have followed: isolate the faces to force the learner to concentrate on them (e.g., [13]). This approach has two obvious effects, one positive and one negative. The positive effect is that experiments do really concentrate better (if not perfectly) on the
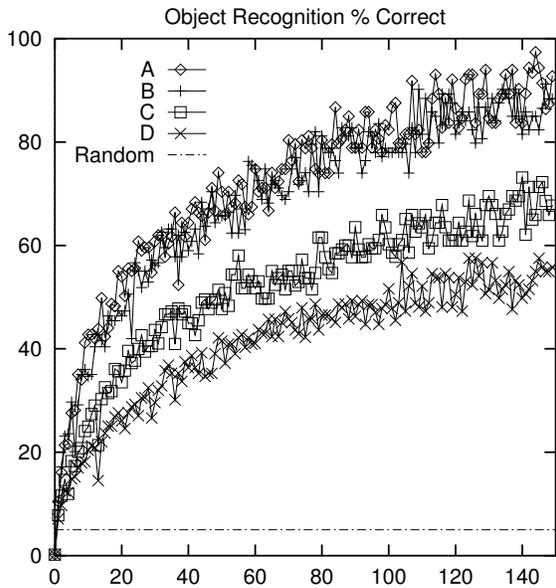
Fig. 4: The Face Recognition % correct for each generation. Test results of experiments **A**, **B**, **C**, **D**.

Fig. 5: Random train and test images.

faces. The negative effect, rarely discussed, is that part of the learning problem should be to find the face in the image. This aspect of the problem has been removed when the images are cropped to remove the background. If the background "gives away" the answer, one solution is to constrain the problem by removing the background. A more elegant solution is, perhaps, to get more realistic data.

Only because this face cropping is so often done for face recognition experiments, our third experiment did something similar for comparison. Instead of reducing the image to a smaller image that cropped the face perfectly, we simply reversed the mask from the previous problem so that background was masked and the faces were mostly visible (see Figure 6).

This problem is slightly harder than the cropping equivalent because the learner must learn to ignore the black area (absent in experiments like [3, 13]). As can be seen in curve **C** of Figure 4, the recognition rate climbs to approximately 70% by generation 150 under these conditions. Again, this is comparably to learned recognition rate results on cropped images like those from the MIT face set (e.g., [3]).

Experiment D

Given the results of experiments **A**, **B**, and **C**, our final experiment seemed a clear next step, though we have been unable to find analogies to it in the literature. This final step is: instead of masking the background, why not simply randomly excha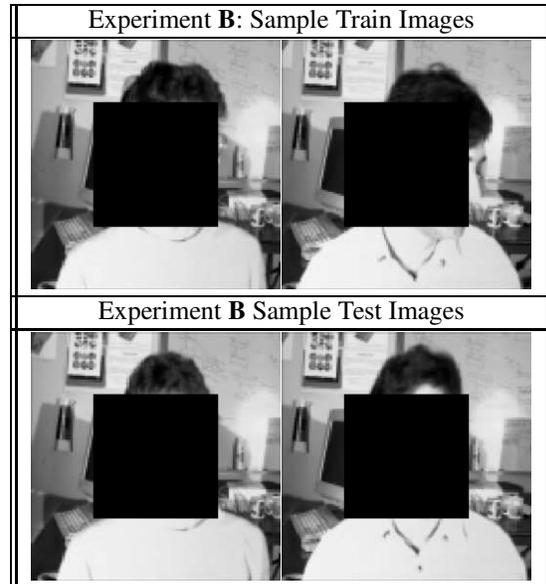nge them between images? The result of this is that the images are as complex as the original images (thereby avoiding the negative effect of cropping or masking mentioned above) but the answer is not "given away" by the background (thereby preserving the positive effect of cropping) (see Figure 7). While no substitute for more carefully taken face data, this background switching yields a more realistic face recognition problem than does image cropping to the faces.

Not surprisingly, this problem is harder than the task of experiment **C** even though in both cases only the signal data in the central square correlated to the correct answer. PADO manages to learn algorithms that recognize about 56% of the unseen faces by generation 150. This task is harder because, while the signal to noise ratio is the same in experiments **C** and **D**, there is no variation in the noise in experiment **C**, so there is no misleading correlations for the learner to find.

5. Conclusion

Previous work has shown that evolutionary computation can be used to help in the task of face recognition.

The first goal of this paper was to show that PADO, a variant of genetic programming, can produce algorithms that directly discriminate between face images and that PADO can perform on a par with past face recognition learning projects. This was shown in Section 4; PADO achieved a 92% recognition rate distinguishing among 20 different
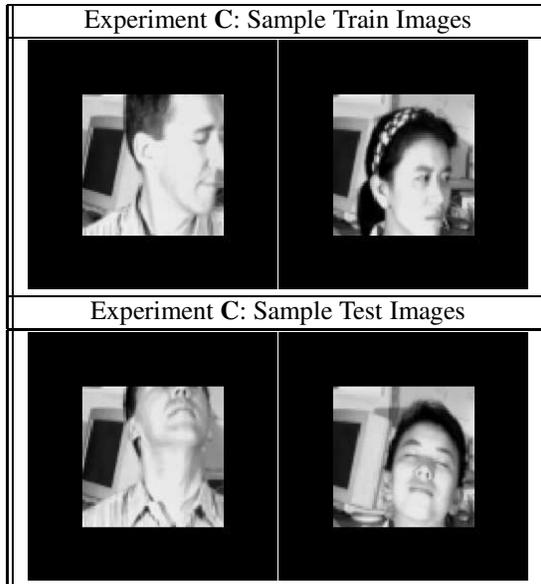
| Experiment **C**: Sample Train Images |
| Experiment **C**: Sample Test Images |

Fig. 6: Random train and test images.

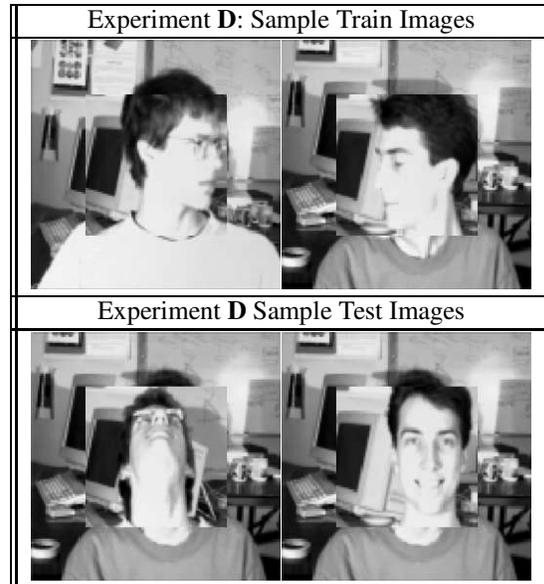| Experiment **D**: Sample Train Images |
| Experiment **D** Sample Test Images |

Fig. 7: Random train and test images.

face classes.

The second goal of the paper was to show that it is very difficult to correctly evaluate the difficulty of face images. The CMU face image sets looks more difficult than many of the traditional face databases. Experiment **B** showed that it is not.

And our third goal was to suggest, through experiment **D** and the difference between curves **C** and **D** in Figure 4, that traditional methods for "making the problem harder" (e.g., image cropping), have partially emasculated the problem.

We are currently producing a database of face images (http://www.cs.cmu.edu/~astro/FacePage.html) that really have the properties the face recognition task implicitly defines (i.e., vacation-style photos). We believe that, while still young, the marriage of genetic programming and computer vision promises a bright future for the face recognition task.

References

[1] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In K. Kinnear, editor, *Advances In Genetic Programming*, pages 477–494. MIT Press, 1994.

[2] P. J. B. Hancock and L. S Smith. GANNET: Genetic design of a neural net for face recognition. In H-P. Schwefel and R. Mǎnner, editors, *Proceedings of the Conference on Parallel Problem Solving from Nature*. Springer Verlag, 1991.

[3] N. Intrator, D. Reisfeld, and Y. Yeshurun. Face recognition using a hybrid supervised/unsupervised neural network. In *Proceedings of the Face and Object Recognition Conference*, 1995.

[4] M. Johnson. Evolving visual routines. In R. Brooks and P. Maes, editors, *Artificial Life IV*. MIT Press, 1994.

[5] John Koza. *Genetic Programming II*. MIT Press, 1994.

[6] T. Nguyen and T. Huang. Evolvable 3d modeling for model-based object recognition systems. In K. Kinnear, editor, *Advances In Genetic Programming*. MIT Press, 1994.

[7] Dean Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. PhD thesis, Carnegie Mellon University School of Computer Science, 1992.

[8] Jeff Shufelt and Tom Mitchell. Personal correspondence. In *Carnegie Mellon University*, 1995.

[9] Walter A. Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kauffman, 1993.

[10] Astro Teller. Evolving programmers: The co-evolution of intelligent recombination operators. In K. Kinnear and P. Angeline, editors, *Advances in Genetic Programming II*. MIT Press, 1996.

[11] Astro Teller and Manuela Veloso. PADO: A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*. Oxford University Press, 1995.

[12] Astro Teller and Manuela Veloso. Program evolution for data mining. In Sushil Louis, editor, *The International Journal of Expert Systems. Third Quarter. Special Issue on Genetic Algorithms and Knowledge Bases*. JAI Press, 1995.

[13] P. Viola. Feature-based recognition of objects. In *AAAI FSS on Machine Learning in Computer Vision*. AAAI, 1993.