Networks of Transform-Based Evolvable Features for Object Recognition

Taras Kowaliw
ISC-PIF, CNRS,
Paris, France
taras@kowaliw.ca

Wolfgang Banzhaf
Memorial University,
St. John's, Canada
banzhaf@mun.ca

René Doursat
Drexel University,
Philadelphia, USA
rene.doursat@drexel.edu

ABSTRACT

We propose an *evolutionary feature creator* (EFC) to explore a non-linear and offline method for generating features in image recognition tasks. Our model aims at extracting low-level features automatically when provided with an arbitrary image database. In this work, we are concerned with the addition of algorithmic *depth* to a genetic programming (GP) system, hypothesizing that it will improve the capacity for solving problems that require high-level, hierarchical reasoning. For this we introduce a *network superstructure* that co-evolves with our low-level GP representations. Two approaches are described: the first uses our previously used "shallow" GP system, the second presents a new "deep" GP system that involves this network superstructure. We evaluate these models against a benchmark object recognition database. Results show that the deep structure outperforms the shallow one in generating features that support classification, and does so without requiring significant additional computational time. Further, high accuracy is achieved on the standard ETH-80 classification task, also outperforming many existing specialized techniques. We conclude that our EFC is capable of data-driven extraction of useful features from an object recognition database.

Categories and Subject Descriptors

I.5.4 [**Computing Methodologies**]: Pattern Recognition—*Computer Vision*; 1.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*

General Terms

Algorithms, Design, Experimentation

Keywords

pattern recognition, computer vision, object, genetic programming, TEF, deep learning

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

1. INTRODUCTION

In order to achieve greater autonomy in pattern recognition, we need the ability to extract representations directly from databases. This is especially true for computer vision, where large dimensionality is the norm, and sample distributions tend to be dense and noisy. Operating directly on the original, *raw pixel space* allows for the extraction of useful low-level patterns and features not captured by traditional vision-based routines. This is now an active area of research, with many strategies being explored, including wavelets, histograms, and others [27, 26, 8, 35, 24].

Our present work concentrates on evolutionary computation (EC). There is a long history of the use of EC in image processing tasks [1, 2, 11, 12, 15, 19, 34, 36, 32]. Genetic programming (GP), in particular, is known to generate versatile and self-adapting solution structures [29, 10]. Flexible, rather than intuitive, representations are especially critical in applications for which techniques inspired by human vision might be suboptimal, such as satellite, multispectral, or medical imagery [30, 13, 31].

One approach to evolutionary image classification is our design and use of *transform-based evolvable features* (TEFs), the key ingredient in a larger *evolutionary feature creator* (EFC). Here, a collection of genetic programs are evolved as transformations on the space of images, which are evaluated for their ability to create descriptions of the images. This approach was first introduced in [17], then applied to various challenges such as the recognition of artistic styles [18], or a form of muscular dystrophy in biological cell images [16]. In the latter case, our technique matched the state of the art. The advantages of using TEFs are four-fold:

- the discovered transforms and their descriptions can be computed in a single pass over the image;
- transforms are easily visualized and interpreted;
- transforms are represented as a simple mathematical expressions, easily interrogated or modified;
- we can evolve any number of distinct features in a natural way.

Here we will apply our system to an object recognition benchmark, the ETH-80 dataset [22]. Object recognition involves learning to classify objects from any of several perspectives (for instance, an image of a mug from the top looks like a circle, while from the side it looks like a cylinder with a handle). As such, higher-level representations are generally beneficial, allowing for more primitive features to be recombined in non-linear ways. For this reason, object recognition is a common target of *deep learning* systems [4], a branch of neural computation often applied to computer vision. Neu-

ral networks are considered "deep" if there are more steps between the inputs and outputs—implying a greater capacity for abstraction—as opposed to "shallow" representations, where a dictionary of sub-representations are combined simply (e.g. as in support vector machines, SVM). In deep learning, a network will implicitly create low-level and high-level features simultaneously while learning, expressing them as sub-networks. For a wide class of problems, deeper representations are known to be exponentially more effective than shallower representations, but are also known to contain more local minima [5]. Deep representations have been highly successful recently in vision problems [21, 7].

Here we consider a new, deep version of our evolutionary system. In addition to our evolving sub-representations (the TEFs), we also co-evolve a *network superstructure* connecting those TEFs. This new superstructure will allow for the re-use of existing TEFs as inputs to more specialized (i.e. deeper) TEFs. We will contrast this new system against our original, "shallow" setup. It is not obvious that this addition of depth to a shallow GP-based system will be useful: GPs are more malleable than perceptrons or SVM kernels, and are innately capable of representing hierarchical knowledge. Yet, while GPs can theoretically recreate some depth on their own, our intuition is that such a representation will not be evolvable. Rather, our reasoning is that an imposed two-level system will allow for more primitive features to co-evolve with higher-level representations while retaining interim information as a working output.

We are not the first to use a deep network representation for GP-based image processing. GP-based network systems have already generated successful image segmentation methods [33, 23], which are partial inspiration for this work. There are several important differences with our model, however: firstly, we aim for feature extraction for classification; secondly, our system co-evolves both network structure and the component sub-representations simultaneously; and thirdly, our system uses a classifier as a wrapper, meaning that the extracted features need to support a classifier rather than solve the problem directly. This last point allows for different, specialized classifiers to be swapped in and out as desired—as we shall demonstrate—and for expertise to be located at a later stage.

2. THE DATA

The ETH-80 database, created by Leibe and Schiele [22], is an object database composed of eight categories (typically toys, mugs or fruits), each consisting of ten instances or "objects", photographed from 41 views. This makes $8 \times 10 \times 41 = 3280$ images in total. Figure 1 shows one particular view for each object, all from the same angle (here we preprocess the images by scaling them to 80×80 pixels and converting to grayscale). A typical recognition task consists of predicting the category of a previously unseen object.

For each run, we divide our data into three sets: a *training* set, which is used to train a classifier; a *validation* set, used by the fitness function to evaluate any particular classifier; and a *test* set to evaluate the final result of the evolutionary process. Each set contains a certain number of objects, where each object means a series of 41 images.

Initially (Section 4.1), we choose 24 objects as validation objects (i.e., $24 \times 41 = 984$ images), and 8 objects as test objects (i.e., $8 \times 41 = 328$ images). First, we evolve a good solution using the validation data as a fitness function. Then, we evaluate the highest fitness solution on the test data to obtain our test accuracy.

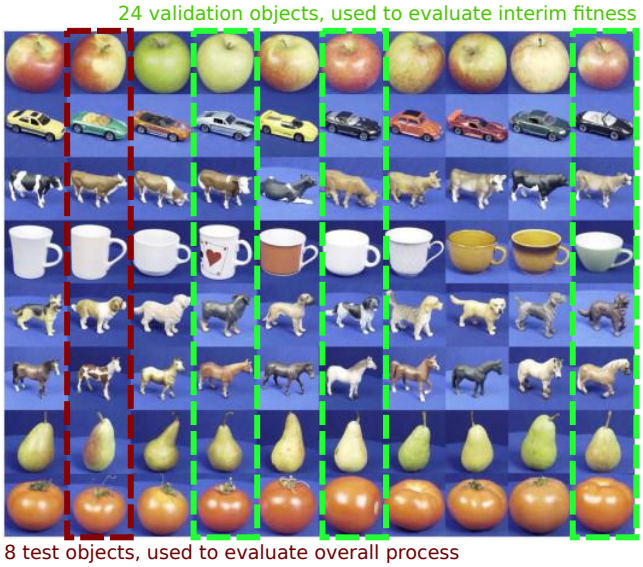

Figure 1: Samples from the ETH-80 database. There are 8 categories of 10 objects each. Each image shows a single view (from 41) of one object. One set of test objects (*column in dark red frame*) and three sets of validation objects (*columns in light green frames*) are highlighted. Each column contains a total of 8 × 41 views (other 40 not shown).

Later (Section 4.2), we also choose 24 objects as validation objects, leaving 8 test objects aside. Evolution proceeds as above, using the validation set to compute fitness. However, once evolution is complete and we have extracted our highest fitness solution, we compute the test accuracy for each of the 8 test objects separately. That is, for each test object, we take the remaining 79 objects to create a new training pool, and evaluate the accuracy on that test object. This latter measure, called the *leave-one-object-out accuracy*, is the traditional evaluation metric for the ETH-80 database.

3. THE MODEL

We begin with a high-level overview of our original model (the interested reader should consult [16] for details). Next, we discuss the new features.

Our model, the EFC, operates on a provided database of images, divided into a finite number of classes. The ultimate output of the system will be a collection of easily computed features well-suited for classification of the database by some particular classifier. The EFC consists of two parts: an evolutionary algorithm, and a collection of *feature-extracting individuals* or simply "individuals".

An individual is a tuple: $(n, T_1, ..., T_k, M, \sigma)$ where n is a neighbourhood size, T_i is a transform-based evolvable feature, M is a dimensionality-reducing function, and σ is a classifier. When presented with an input image, an individual will process it as follows (Figure 2):

1. compute a collection of transformations of the image using its TEFs T_i;
2. convert the transformed images to a *feature vector* via the moment function M;
3. feed the feature vector to classifier σ, which will return a class label.

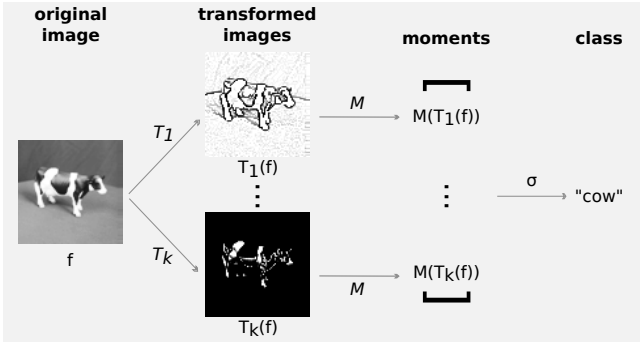

Figure 2: Overview of a feature-extracting individual. First, the original image is transformed by a collection of TEFs T_i. Next, each transform is converted to numerical values via the moment function M. Finally, the feature vector of moments is classified.

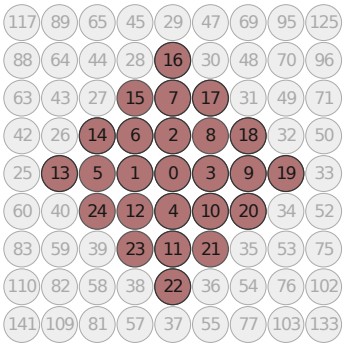

Figure 3: The neighbourhood of a pixel, VN-25, is mapped to a prioritized one-dimensional representation, which is fed to a genetic program.

A TEF is a genetic program which operates on the pixel space of the image as a sliding window. For each pixel in the source image, a neighbourhood is collected and passed to the TEF, which returns a real value in $[0, 1]$. This new value is used to define the intensity of the corresponding output image pixel. While neighbourhoods may vary (there is evidence that the EFC can self-select an appropriate neighbourhood size), here we simply use a neighbourhood of type VN-25 (Figure 3).

We use a single dimensionality-reducing function M, the first geometric moment. As in much previous work, informal experimentation convinced us that this was the best choice.

Our present work diverges from previous versions of the EFC in two ways: the chief modification is the inclusion of a *network superstructure* organizing the relations between TEFs. This allows for the generation of *depth* in the resulting transforms, as some TEFs can build upon the lower-level results of previous TEFs. A second change is the usage of Linear Genetic Programming (LGP) as the representation of the TEFs.

3.1 LGP representation for TEFs

There are two main motivations for using LGP: first, LGP is known to be naturally parsimonious, which is a desirable trait for our transforms; second, LGP is very easily converted to computer code—including, potentially, assembly code—which allows for the easy conversion of an LGP program to a compilable and fast machine-independent code listing. For reasons of verification and export of results from the system, this is also a desirable property.

Table 1: Pool of functions and actions on inputs (a, b).

func.	action	func.	action
plus	$a + b$	max	$\max\{a, b\}$
minus	$a - b$	min	$\min\{a, b\}$
times	ab	abs	$\lvert a\rvert$
sqrt	$\sqrt{\lvert a\rvert}$	square	a^2
inv	$1/a$, or 1 if $\lvert a\rvert < 0.0001$	thresh	1 if $a > b$, 0 otherwise
pow	a^b, if defined 1 otherwise	log	$\log a$, if defined 1 otherwise
dist	$\sqrt{a^2 + b^2}$		

Our implementation of LGP closely follows [6]. An LGP program consists of

- 25 initial registers, for the inputs from the VN-25 neighbourhood
- 5 additional registers, initialized to genetically controlled values in $[0, 1]$
- between 2 and 100 binary program statements
- a return statement, returning the contents of the first register (if necessary, the return statement is forced into the range $[0, 1]$ by using the closest extreme value).

Our pool of functions is shown in Table 1.

We can mutate an LGP program through two forms: micro-mutation and macro-mutation, where the former reinitializes existing elements with probability p_M, and the latter adds or removes a program statement. When mutating an LGP program, we choose macro-mutation with probability p_{mut}, otherwise we apply micro-mutation.

We will refer to the *effective size* of an LGP program as the number of non-neutral program statements. Note that this is an over-estimate of the complexity of a program, since some non-neutral program statements might be ineffective (e.g., $a = a + 0$). The *mean effective size* of an individual is the mean of the effective size of each of its transforms.

3.2 OrigEFC

We will refer to our original EFC system with LGP representation as OrigEFC. Like previous version, this is a *shallow* application of a series of transforms to the database.

As in previous work, we use four genetic operators: mutation, crossover, merger, and pruning. All are identical to previous work, except that intra-transform mutation now mutates an LGP program.

3.3 DeepEFC

Our primary extension of the EFC model will be termed DeepEFC, where we add a network structure to the original EFC representation. In this case, we view each transform as a linear list of functions, which is applied either to the raw images or to the outputs of previous transforms.

Thus, a DeepEFC individual can be written as

$$I = (n, (t_1, T_1), ..., (t_k, T_k), M, \sigma),$$

where n is a neighbourhood size, M is a statistical moment, σ is a classifier, and (t_i, T_i) is a target-TEF pair. The target t_i is a positive integer, interpreted either as the index of a preceding TEF T_j (in which case $t_i = j$) or as an indicator of the original image.

Given an image, denoted by f, a DeepEFC individual:

- applies the first TEF directly to f, producing image $f_1 = T_1(f)$
- for each subsequent target-TEF pair (t_i, T_i):
  - renormalizes the index: $j = (t_i \bmod i)$, so that it points to a preceding TEF T_j, where $j \in [0, i-1]$
  - gets the source image f_j to be used:
    - if $j = 0$, f_j is the original image f
    - if $j > 0$, f_j is as defined by previous iteration
  - applies the TEF to the source image:
    - $f_i = T_i(f_j)$
- computes the complete feature vector: $v = \{M(f),\ M(f_1),\ ...,\ M(f_k)\}$
- returns the class label $\sigma(v)$.

Thus, an individual now has its TEFs organized in a network, where one TEF might operate on the output of a previous TEF. By chaining the computation of the TEFs, we can compute a deep individual in the same amount of time as a shallow individual, assuming an equivalent complexity in terms of number and composition of TEFs.

A DeepEFC individual can be *initialized* in the same manner as an OrigEFC individual, with a random, positive integer chosen for each t_i value. With probability 0.5 we set $t_i = 0$ under the assumption that the majority of representations should be shallow.

All *genetic operators* applied to a DeepEFC individual are the same as for OrigEFC, where target-TEF pairs are always treated as a single unit. Thus, the crossover operator will swap target-TEF pairs between individuals, a merger will create an individual with a large list of target-TEF pairs from two source individuals, and pruning will delete a target-TEF pair from an individual. We can *mutate* a DeepEFC individual in the same manner as an OrigEFC individual, except that we also have some chance of mutating each t_i value. In these cases, the t_i value is reinitialized randomly.

3.4 Evolutionary Algorithm

Our system is driven by a steady-state evolutionary algorithm. A population of n_{pop} individuals is initialized and evaluated for fitness. Next, for $n_{\text{eval}} - n_{\text{pop}}$ iterations, we

(a) select an individual via a tournament of two randomly chosen individuals;
(b) select an operator from crossover, merger, or pruning, with probability p_{cross}, p_{merge}, and p_{prune} respectively. If none was selected, we choose mutation with strength p_{mut} (see [16] for operator definitions);
(c) in the case of crossover or merger, we select a second individual via tournament;
(d) we replace the worst individual from the initial tournament in (a) with the result of the operation.

Thus, we execute exactly n_{eval} evaluations in total.

The complexity of the training is linear in $O(n_{\text{eval}}\, n_{\text{im}} |f|)$, where n_{im} is the number of images in the training and validation sets, and $|f|$ is the number of dimensions in an image.

The *fitness* F of an individual I is the generalized sensitivity-specificity:

$$F(I, V) = \prod_{\text{categories } c} (1 - \text{FPR}(I, V, c))$$

where V denotes the set of validation images, and FPR is the False Positive Rate of the classifier over the images of category c. We use this product—rather than, say, simple classification accuracy—to discourage local minima when a classifier learns to recognize a single category but not others. To report results, however, we will use the classification accuracy. The winner of a tournament between individuals I_1 and I_2 is:

$$\text{win}(I_1, I_2) = \begin{cases} \arg\min_I n_{\text{T}}(I) & ; \text{ if } |F(I_1) - F(I_2)| < \delta_{\text{pars}} \\ \arg\max_I F(I) & ; \text{ otherwise} \end{cases}$$

where δ_{pars} is a parsimony threshold and $n_{\text{T}}(I)$ is the number of TEFs in individual I.

Following an informal parameter search of approximately 50 trial runs, we found the following parameter set to maximize the test fitness:

max number of initial TEFs	3	nbhd.	VN-25
max number of TEFs	20	δ_{pars}	0.001
n_{pop}	800	n_{eval}	15 000
p_{mut}	0.025	p_{cross}	0.33
p_{merge}	0.01	p_{prune}	0.025

Unless otherwise stated, these parameters will be used for all following experiments. Note that this choice was made based on the expected maximization of the OrigEFC model, not the DeepEFC, and hence, if any advantage exists, we expect it to favour the original technique.

4. RESULTS AND ANALYSIS

4.1 Model comparison

Initially, we compare the result from OrigEFC and DeepEFC on the ETH-80 database. For this section, we limited the test set to include the objects indexed "1" in each category (the second column in Figure 1), while the validation objects were randomly picked for each run. Constraining our choice of test objects to these particular ones is a setting we believe to be of greater than average difficulty based on the classifier accuracies that we computed in our early explorations. We do so to limit the variance due to selection of the test object and better see the stochasticity resulting from the use of an evolutionary algorithm. Note that this implies lower test accuracies than is generally possible, due to both the choice of test objects and less training data ($80 - 8 = 72$ left for training, instead of 79).

First, we looked at the performance of OrigEFC. The evolution of the classification accuracies for one typical run is shown in Figure 4. In general, evolution was successful, with individuals steadily increasing in fitness over time. Over about 15 independent runs, final test accuracies had a mean of 0.809 (s.d. 0.033) using the original first nearest neighbour (1-NN) classifier.

Generally speaking, test accuracy increased with training fitness, although there were occasional exceptions. The system did not, in general, overfit: on the contrary, generally there was substantial underfitting, with test accuracy approximately 20% higher than training accuracy. This means that our choice of validation technique (24 objects) was robust; it also suggests that running the experiments for more time might generate better results.

Next, we performed identical experiments on DeepEFC, using the same parameters as above, and again constraining the choice of test objects. In this case, evolution followed

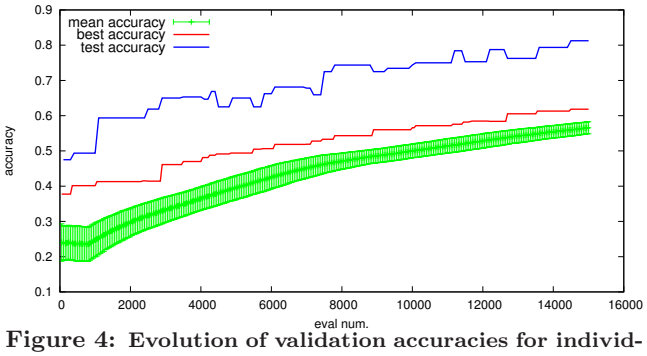

Figure 4: Evolution of validation accuracies for individuals in a typical OrigEFC run (*green*: mean, *red*: best). Test accuracy (*blue*) also computed to track overfitting.

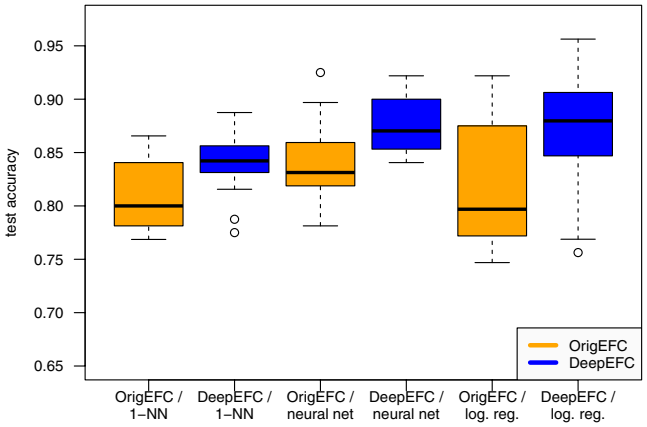

Figure 5: Comparison of test accuracies for several independent runs of OrigEFC (*orange*) and DeepEFC (*blue*) using several classifiers.

a very similar course, that is, a steadily increasing fitness over time, and substantial underfitting. The final mean test accuracy for about 15 independent runs of the DeepEFC system was 0.838 (s.d. 0.029) using the 1-NN classifier.

A comparison of the test accuracies achieved by OrigEFC and DeepEFC can be seen in Figure 5. We also consider the results re-evaluated via two additional classifiers: logistic regression and a neural network. Both are implemented via Weka [37] using default values. For all classifiers, a Welsch's two-value t-test confirms that the difference between mean values is significant ($p < 0.05$ in all cases), therefore that DeepEFC is an improvement over OrigEFC.

A second question we had was about the *complexity* of the evolved LGP programs. The mean effective size of the most

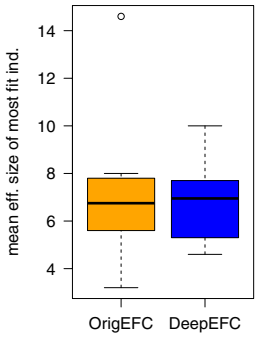

Figure 6: Comparison of the mean effective size of the TEFs in the most fit individual for several independent runs of OrigEFC and DeepEFC.

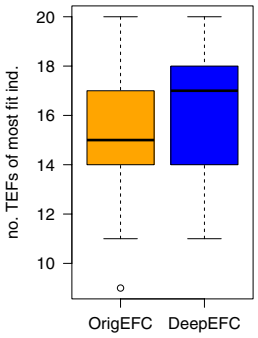

Figure 7: Comparison of the number of TEFs in the most fit individual for several independent runs of OrigEFC and DeepEFC.

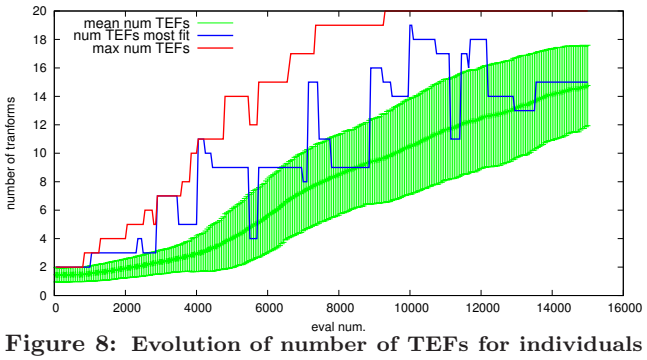

Figure 8: Evolution of number of TEFs for individuals in our example OrigEFC run.

fit individuals for each run is shown in Figure 6. Our first note is that our intuition regarding LGP was correct, the programs are parsimonious with an average size of six or seven program statements (see the appendix for samples): programs of this size can easily be analyzed by a human. This is a natural consequence of the use of LGP, as we did not put any pressure toward small LGP programs like we did toward a small number of TEFs.

A second note regarding the mean effective size is that there is no substantial difference between the OrigEFC and the DeepEFC program sizes. Indeed, with a mean program length of approximately six or seven statements, it is highly unlikely that the TEFs themselves are performing any computation that might be reasonably considered "deep". Thus if either system is capable of hierarchical reasoning, it most certainly comes from the network structure or the classifier.

A final note concerns the resulting size of the solutions. Since each TEF must be computed for each image, more TEFs imply greater computational complexity. We plot the number of TEFs of the most fit individuals for both groups in Figure 7. There seems to be some pressure toward larger solutions for DeepEFC than for OrigEFC, but this difference is not obviously significant ($p > 0.3$ by a Welsch's two-value t-test). Complicating this picture, it appears that our runs were constrained not by evolutionary pressure, but instead by our artificially imposed maximum number of TEFs. Figure 8 shows the evolution of the number of TEFs for our example run with OrigEFC. In this case, it is evident that evolution was pressing for larger solutions. Hence, it is difficult to predict what would happen if our solution size was fully unconstrained. We conclude that in this particular case, there was no significant additional computational cost to running DeepEFC compared with OrigEFC, but note that less restrained parameters might lead to different results.

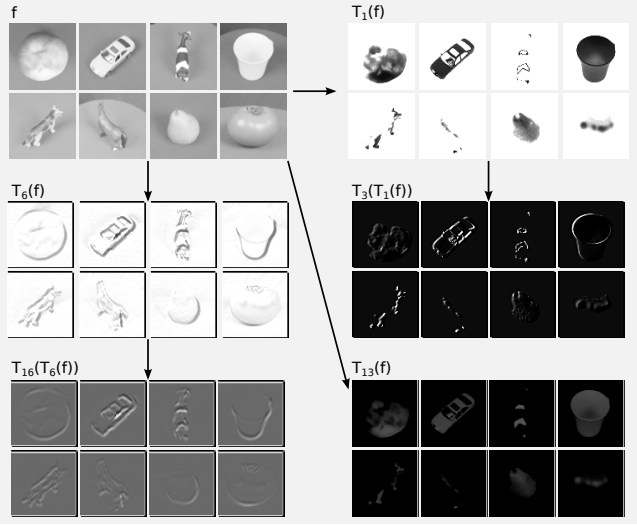

Figure 9: Illustration of part of the network comprising our exemplar individual: transforms T_6 and T_{16} show increasingly specialized edge detectors; transform T_1 generally serves to emphasize lighter areas of the image; transform T_3 reveals sharp diagonal transitions in the already segmented lighter areas.

4.2 Evaluation of DeepEFC using all objects

Here we re-evaluate the DeepEFC technique using randomly chosen test objects. As before, a collection of features is created having kept eight randomly chosen objects as test objects, one from each category. Once evolution is complete, however, we use the resulting collection of features to classify one test object, having re-trained the classifier using the other 79 test objects as training data. Thus, each run allows us to evaluate eight of the 80 test objects.

Since our technique is stochastic, we launched 20 independent runs (i.e. 160 evaluations in total, including at least one of each object). All parameters were the same as above, except for longer runs ($n_{\text{eval}} = 20\,000$). As before, these trials showed some underfitting, and generally increasing test accuracies with time (again, it is possible that running the algorithm for more time might yield better results). The mean leave-one-object-out classification accuracy using the original 1-NN classifier, a neural network, and logistic regression, are:

classifier	mean test accuracy	s.d. test accuracy
1-NN	0.839	0.201
neural net	0.890	0.171
logist. reg.	**0.902**	0.168

It should be noted that the ETH-80 contains several particularly difficult objects, which bring down the mean values. The *median* classification accuracy, using logistic regression as a classifier, is **0.950**.

4.3 Exemplar individual

Here we analyze one exemplar individual, drawn from one DeepEFC after 15 000 evaluations. A full listing of the exemplar individual's genome is in the appendix.

It is clear that although depth is used, it is not extensively used. No transform is more than two steps deep, and most transforms operate on the original images. This is typical of our DeepEFC runs.

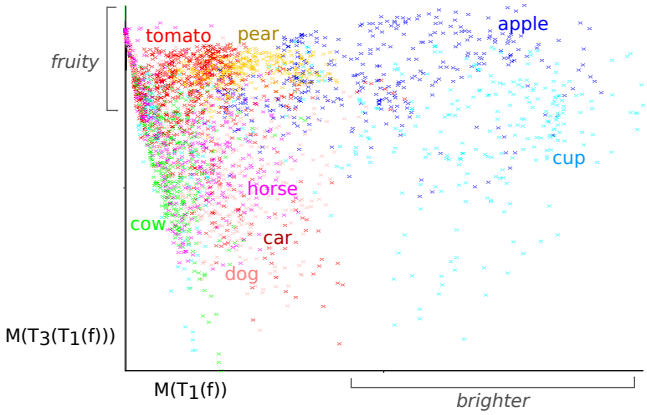

Figure 10: Illustration of the spread of values for transforms T_1, T_3. Best viewed in colour.

Figure 9 shows a sub-network of the exemplar individual, containing a couple of depth-two pathways. The benefits of increased specialization via depth can be seen here.

For instance, T_1 is a transform which (loosely) extracts the light-coloured sections of an image. Doing so discriminates between cups and apples on the one hand (the two brightest categories), and the remainder of the objects on other hand. The deeper T_3 transform highlights subregions from the bright regions already extracted, marking sharp transitions in the diagonal direction. This increased specialization serves to discriminate objects without sharp transitions in light-coloured regions, i.e. the fruits (Figure 10).

Similarly, T_6 is a rather general edge detector. While it has some discriminatory power, it does not favour any particular class. The deeper T_{16} transform, however, highlights a subset of these edges, which is effective at distinguishing between cars and pears.

As in our previous work with the EFC, TEFs range from very general to very specific. Edge detectors are common, as one would expect in an object recognition task. But some TEFs are also highly specialized: for the ETH-80, distinguishing between horses, dogs, and cows is the most difficult sub-problem. Our exemplar individual discovered a transform, T_{10}, which serves no other purpose than highlighting the spots on certain cow objects (Figure 11).

5. COMPARISON TO PREVIOUS WORK

The ETH-80 is a well-studied benchmark database with many results reported. Recent years, in particular, have seen rapid improvements in accuracy. Table 2 displays a subset of significant works on this database, including the most recent and the best, and contextualize our results. In all comparisons, however, it should be stressed that the EFC is a general-purpose system, and thus our application to this domain is rather naive. In particular, it should be remembered that our results are emergent, i.e., our model includes no preprogrammed low-level features or descriptors, no preselection of features, and no cognitive model of 3D objects. Our image sources were also constrained to grayscale.

Nilsback and Caputo utilized a multi-voting system under a decision tree classifier [28]. Bajramovic *et al.* considered several variants of nearest neighbour search in the context of optimizing an accuracy/runtime trade-off [3]. Ling and Ja-

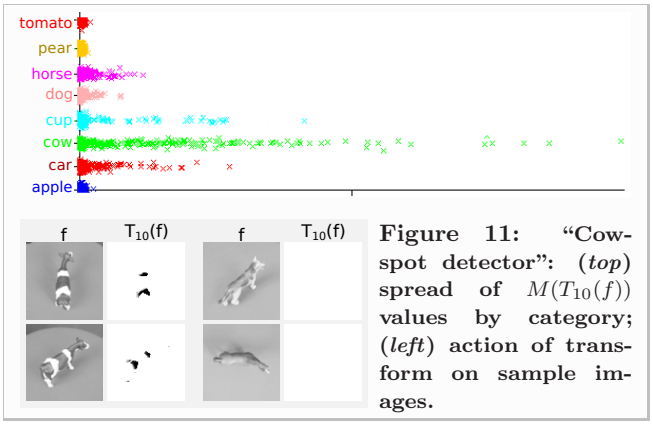

Figure 11: "Cow-spot detector": (*top*) spread of $M(T_{10}(f))$ values by category; (*left*) action of transform on sample images.

Table 2: Comparison of reported category recognition accuracies for the ETH-80 database by increasing values.

features (+ classifier)	accuracy
LDA [20]	0.652
biased discriminant analysis [20]	0.741
texture histograms + dec. tree [22]	0.798
nearest-neighbour variants [3]	0.840
colour, contour + dec. tree [22]	0.864
morphological + LCDP [14]	0.880
inner-distance, textures + dec. tree [25]	0.881
DeepEFC + logist. reg. (this work)	**0.902**
active sampling, metric learning + 1-NN [9]	0.903
colour, contour + multi-cue dec. tree [22]	0.930
Multivoting + dec. tree [28]	0.971
Complex histograms + SVM [24]	0.977

cobs designed a measure of inner distance in shapes, extracting a new means of representing the shape and augmenting with a texture description [25]. Kwak and Oh apply several forms of linear feature extraction techniques to PCA-reduced features [20]. Hu *et al.* use several morphological strategies to improve shape retrieval [14]. Ebert *et al.* rely on a combination of active sampling and metric learning. Their metric improves performance for many classifiers [9]. The best results to date come from Linde and Lindeberg, who defined a collection of low-level descriptors, which led to the definition of a multi-dimensional basis for histograms, further reduced by PCA. These histogram descriptions were then optimized via SVM [24].

6. CONCLUSIONS

In this paper, we applied a GP-based evolutionary feature creator to a benchmark object recognition problem. Our results were less than state-of-the-art, yet higher than many specialized techniques. Given that our system is a naively applied, general purpose machine, and that our features are entirely emergent from the original data source, we consider this to be a striking demonstration of the *capacity for artificial evolution to automatically extract features.*

Further, we contrasted a "shallow" and "deep" approach, showing that the latter generated improved results using the same computational time and solution complexity. This suggested that GP might benefit from deep representations in the same way that neural networks do, promising applicability to a wider range of problems.

In deep learning, however, it is well understood that increasing the depth of a chain of representations can generate more local optima. This problem almost certainly exists in the present system. It is an intriguing and unexplored question whether or not the solutions from the neural network community (e.g., pretraining via auto-association or sparse coding) could be applied to GP-based approaches as well.

7. REFERENCES

[1] E. Ahn, T. Mullen, and J. Yen. Evolutionary based feature extraction with dynamic mutation. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 409–416, 2011.

[2] H. Al-Sahaf, K. Neshatian, and M. Zhang. Automatic feature extraction and image classification using genetic programming. In *Conference on Automation, Robotics and Applications (ICARA)*, pages 157–162, 2011.

[3] F. Bajramovic, F. Mattern, N. Butko, and J. Denzler. A comparison of nearest neighbor search algorithms for generic object recognition. In J. Blanc-Talon *et al.*, editor, *Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 1186–1197. Springer-Verlag, 2006.

[4] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[5] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

[6] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer, 2006.

[7] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

[9] S. Ebert, M. Fritz, and B. Schiele. Active metric learning for object recognition. In *DAGM-OAGM*, 2012.

[10] P. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man and Cybernetics — Part C*, 40(2):121–144, 2010.

[11] W. Fu, M. Johnston, and M. Zhang. Genetic programming for edge detection: a global approach. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 254–261, 2011.

[12] H. Guo, L. Jack, and A. Nandi. Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man and Cybernetics — Part B*, 35(1):89–99, 2005.

[13] D. Howard, S. Roberts, and C. Ryan. Pragmatic genetic programming strategy for the problem of

vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, 2006.

[14] R.-X. Hu, W. Jia, Y. Zhao, and J. Gui. Perceptually motivated morphological strategies for shape retrieval. *Pattern Recognition*, 45(9):3222–3230, 2012.

[15] N. Kharma, T. Kowaliw, E. Clement, C. Jensen, A. Youssef, and J. Yao. Project CellNet: Evolving an autonomous pattern recognizer. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(6):1039–1056, 2004.

[16] T. Kowaliw and W. Banzhaf. The unconstrained automated generation of cell image features for medical diagnosis. In *Conference on Genetic and evolutionary computation (GECCO)*, pages 1103–1110, 2012.

[17] T. Kowaliw, W. Banzhaf, N. Kharma, and S. Harding. Evolving novel image features using genetic programming-based image transforms. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2502–2507, 2009.

[18] T. Kowaliw, J. McCormack, and A. Dorin. Evolutionary automated recognition and characterization of an individual's artistic style. In *IEEE Congress on Evolutionary Computation (CEC)*, 2010.

[19] K. Krawiec, D. Howard, and M. Zhang. Overview of object detection and image analysis by means of genetic programming techniques. *Frontiers in the Convergence of Bioscience and Information Technologies*, 0:779–784, 2007.

[20] N. Kwak and J. Oh. Feature extraction for one-class classification problems: Enhancements to biased discriminant analysis. *Pattern Recognition*, 42:17–26, 2008.

[21] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[22] B. Leibe and B. Schiele. Analyzing contour and appearance based methods for object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[23] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber. An integrated, modular framework for computer vision and cognitive robotics research (icVision). In *Biologically Inspired Cognitive Architectures 2012*, pages 205–210. Springer, 2013.

[24] O. Linde and T. Lindeberg. Composed complex-cue histograms: An investigation of the information content in receptive field based image descriptors for object recognition. *Computer Vision and Image Understanding*, 116(4):538–560, 2012.

[25] H. Ling and D. Jacobs. Shape classification using the inner-distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:286–299, 2007.

[26] G. Malathi and V. Shanthi. Wavelet based features for ultrasound placenta images classification. In *Conference on Emerging Trends in Engineering and Technology (ICETET)*, pages 341–345, 2009.

[27] D. Maturana, D. Mery, and A. Soto. Learning discriminative local binary patterns for face recognition. In *IEEE Conference on Automatic Face Gesture Recognition and Workshops (FG)*, pages 470 –475, 2011.

[28] M. Nilsback and B. Caputo. Cue integration through discriminative accumulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[29] R. Poli, W. Langdon, and N. McPhee. *A Field Guide To Genetic Programming*. Lulu Enterprises, 2008.

[30] P. Rosin and J. Hervas. Image thresholding for landslide detection by genetic programming. In *Analysis of multi-temporal remote sensing images,*, pages 65–72. World Scientific, 2002.

[31] B. Ross, A. Gualtieri, F. Fueten, and P. Budkewitsch. Hyperspectral image analysis using genetic programming. *Applied Soft Computing*, 5(2):147–156, 2005.

[32] L. Sekanina, S. Harding, W. Banzhaf, and T. Kowaliw. Image processing and CGP. In *Cartesian Genetic Programming*, pages 181–215. Springer, 2011.

[33] S. Shirakawa, S. Nakayama, and T. Nagao. Genetic image network for image classification. In *Applications of Evolutionary Computing*, pages 395–404, 2009.

[34] A. Song and V. Ciesielski. Texture segmentation by genetic programming. *Evolutionary Computation*, 16(4):461–481, 2008.

[35] A. Torralba, K. Murphy, W. Freeman, and M. Rubin. Context-based vision system for place and object recognition. In *IEEE Conference on Computer Vision (ICCV)*, page 273, 2003.

[36] L. Trujillo and G. Olague. Automated design of image operators that detect interest points. *Evolutionary Computation*, 16(4):483–507, 2008.

[37] H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, 3rd edition, 2011.

APPENDIX

Mathematical form of all target-TEF pairs from our exemplar individual. Note that some redundancy has been removed, including a redundant and an empty transform. Transforms are listed in order of the information gain of the resulting features.

TEF	source	expression
T_{13}	orig.	$\log\left(\left\|i_3 - 0.614\right\| + i_9\right)$
T_1	orig.	$\frac{(1-i_{22})}{(0.611-i_{22})}$
T_{17}	T_2	$(i_{19})^{(i_{24})^{0.749}}$
T_5	orig.	$thresh\left(\sqrt{\left\|i_5\right\|}, i_{12}\right) - (1 - i_{17}) + \log\left(i_{21}\right)$
T_3	T_1	$\left(\frac{i_{24}}{i_0} - 0.958\right) i_{18}$
$\emptyset$	–	Id
T_{12}	T_3	$i_{14} + \sqrt{(i_2)^2 + (i_{16})^2} \cdot i_1$
T_{14}	orig.	$\left(thresh\left(\sqrt{\left\|i_5\right\|}, i_{21}\right) - (1 - i_{17}) + \log\left(i_{21}\right)\right)$
T_{16}	T_6	$\frac{\log\left(\log\left(i_0\right)\right)}{\left(\left\|0.333+i_{21}\right\| - \log\left(0.333\right) \cdot \frac{i_4}{i_7}\right)}$
T_4	orig.	$\min\left\{i_{16}, 0.595\right\}$
T_9	orig.	$\max\left\{0.406 + \left\|i_3\right\|, i_{13}\right\}$
T_6	orig.	$\frac{\min\left\{i_6, i_9\right\}}{i_{23}}$
T_7	orig.	$\left\|(1 - \max\left\{i_2, i_{24}\right\})\right\| - \max\left\{i_3, i_0\right\}$
T_{10}	orig.	$\sqrt{\left\|1 - \frac{1}{i_{23}}\right\| + \left(\log\left(i_{20}\right) \frac{1}{i_{10}}\right)^2}$
T_2	orig.	$\left\|\min\left\{i_9, i_{19}\right\} - i_{10}\right\|^{i_{14}}$
T_{15}	T_{11}	$\left\|\min\left\{i_6, thresh\left(i_{16}, i_{20}\right)\right\} - \min\left\{\min\left\{\log\left(i_{20}\right), \left(i_{14} - i_7\right)\right\}, \sqrt{(i_3)^2 + (i_{19})^2}\right\}\right\|$