# Associating Images of Simple Objects With Labels Using Image Compression and an Evolved Neural Network

**Feng He and Kuzman Ganchev**

## Abstract

This paper investigates the problem of recognizing simple objects in real images, using neural network image compression and an evolved "black box" neural network. We present an algorithm for image compression that allows the compressed images to be used by a neural network. We also describe a genetic representation of neural networks that allows arbitrary feed-forward neural networks to be represented. While our completed system is not able to solve the problem of associating images with the objects they depict, we have limited success in doing this, and we succeed in compressing images to almost $\frac{1}{40}$ of their original size, and as a proof of concept are able to demonstrate that our genetic algorithm is able to solve a simple problem.

## 1 Introduction

The symbol grounding problem refers to the process of binding symbols to objects in the real world. One example of this is associating images of objects with a symbolic representation of the objects. We present a new approach to this problem that uses image compression to perform the first part of the abstraction, and an evolved "black box" to associate the compressed images with their symbols. We develop a feed-forward neural network to compress the images from 750 pixels to 20 activation values. Then we apply a genetic algorithm capable of producing arbitrary feed-forward neural networks to the task of finding a neural network that can produce the label of an image given its compressed representation. We find that our compressor succeeds in compressing the images and extracting recognizable images from the compressed representation. We also find that the genetic algorithm we developed for the "black box" can solve simple problems, such as associating ASCII-art images with their labels, but that it is not able to associate real compressed images with their symbolic representation, even when the compressed images are easily recognizable by a human.

There are several lines of motivation for this work. Firstly, in its full generality, the problem of associating images of objects to their "names" could be very useful for robotics as well as for other fields of Computer Science, such as Human-Computer Interaction. Secondly image compression can be useful in and of itself, and while there are many image compression algorithms exist, most existing algorithms do not generate an image that can be easily used by a neural network.

Section 2 describes some of the work related to our project. Section 3 gives an overview of the architecture of our system. Section 4 how we obtained our training data. Section 5 describes the specific experiments we performed and the results we obtained. Finally, Section 6 discussed the broader significance of our results as well as some of the future work that they may lead to.

## 2 Related Work

Plunkett et al [3] describe experiments to associate distortions of random dot pattern images (they calls these prototypes) with orthogonal artificial labels for those images. Plunkett et al do not elaborate on how they generate a distorted image form an image prototype, but they do state that the prototypes differ significantly and that the distorted images are selected from a Gaussian distribution centered around the image prototype. The authors discover that their neural network is able to distinguish between the images, albeit when they use extremely artificial data. They pre-process the images using a "retina" which basically scales the images down and unfocuses them slightly. The present work is an attempt to apply this approach to real data collected from the real world.

Our work also differs in our use of self organizing maps for preprocessing.

Widrow [5] provides a good overview of neural networks, and especially of the back propagation learning algorithm. Rather than re-implementing a neural network, we used the Con-x neural network module available in the the Pyro AI and Robotics System [1]. Con-x is a high level implementation of neural networks, which allowed us easy experimental setups.

Ganchev and He [2] attempt to reproduce the results of [3] using real images, rather than artificial images. They use SOMs to pre-process the images, and then investigate different ways to connect the SOM to a Plunkett style auto-association network. The main result of the paper is that SOMs are not a sufficient preprocessing tool for a Plunkett network to associate images of objects with object labels.

Schroeder [4] gives a brief outline of the JPEG compression algorithm. He describes the color space preprocessing, discrete cosine transform and quantization phases of JPEG compression. Unlike JPEG, our compression algorithm uses neural networks to perform all stages of compression, but does not deal with color images. We also do not use a symbolic compression algorithm such as Huffman codes or Ziv-Lempel encoding for the compressed images, allowing for easier interpretation by a neural network.

# 3 Implementation Overview

This section provides a brief overview of what components our system uses and how each one is implemented. Section 3.1 describes the architecture of our project, while the subsequent sections describe its components. Section 3.2 describes the implementation of the image compressor, while Section 3.3 describes the implementation of our genetic black box.

## 3.1 Project Architecture

We divide the problem of identifying an image into two parts. Firstly, we compress the 750 pixel images into 20 activation values. Once this is done, a genetic algorithm evolves an individual that can recognize the compressed image and produce the correct label. Figure 1 shows this architecture. The black box in the figure is an evolved feed-forward neural network. Preliminary results showed that our evolutionary algorithm is prohibitively slow when there are 750 input nodes. For this reason we compress the image to 20 activation values, and evolve an individual to recognize the symbol shown in an image based on the 20 activation values. This provides a performance improvement for two reasons. Firstly, each individual evaluation is considerably faster. Secondly, since the the input information has already been condensed, the evolutionary search space is considerably smaller, so we expect to find a solution in a smaller number of steps.

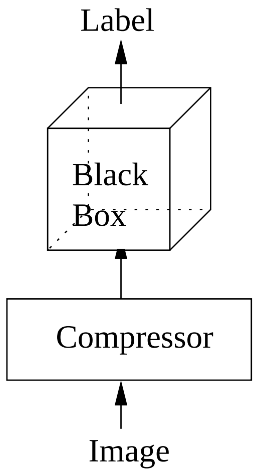


Figure 1: The project architecture. The image provides the activation values for the compressor network; the compressor output activations are the input for the black box, which produces a label.

## 3.2 Compressor Architecture

The compressor is derived from an auto-associative feed forward neural network shown in Figure 2. The network has 750 input and output nodes, but a hidden layer of only 20 nodes. Since the network is auto-associative, the hidden layer activations must contain enough data to reconstruct the image once the network is trained. The connection weights between the input and hidden layer provide a compression algorithm, while the connection weights between the hidden and the output nodes provides a decompression algorithm.

We evaluate the performance of the compressor both quantitatively and qualitatively. The quantitative results are the error values and the number of nodes within a threshold of correct values. We can also visually compare the uncompressed and compressed images to decide whether the compression performance is satisfactory. If we (as human observers) can recognize the compressed images, we consider compression to have been successful.

## 3.3 Black Box Architecture

Conceptually the black box recognizes a given image (in compressed form), and outputs the correct label. Because we are using genetic algorithm (GA) to evolve this network, the challenge lies in creating an architecture that enables us to add and re-

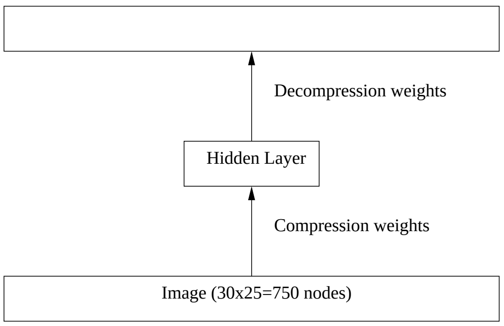


Figure 2: The compressor architecture. The auto-associative network shown has 750 input and output nodes, but only 20 hidden nodes. After training, the 20 hidden activations for a some image must contain enough information to reconstruct the image, and so enough to identify the symbol shown in the image.

move hidden nodes with ease.The obvious choice of an auto-associative feed forward neural network was rejected for this reason. Instead an architecture as shown in Figure 3 was used. The input and output layers have fixed sizes – determined by the image size and the number of labels. For our experiments the input layer has 20 nodes, while the output layer has 2. The hidden nodes are numbered from 0 through $n$. The connections are such that:

(i) Every input node is connected to every hidden node;

(ii) Every hidden node is connected to every output node;

(iii) Hidden node $i$ is can be connected to hidden node $j$ if and only if $i < j$.

We allow zero connection weights to represent connections that do not exist, but are described in the three items above.

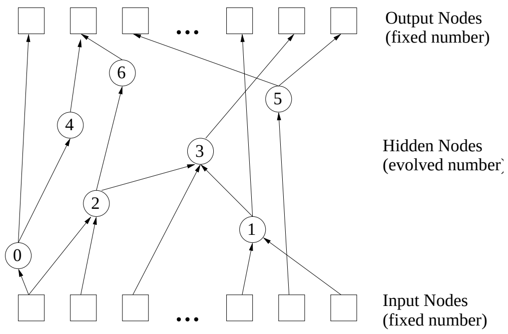


Figure 3: The black box architecture. The architecture was chose to enable the use of GA.

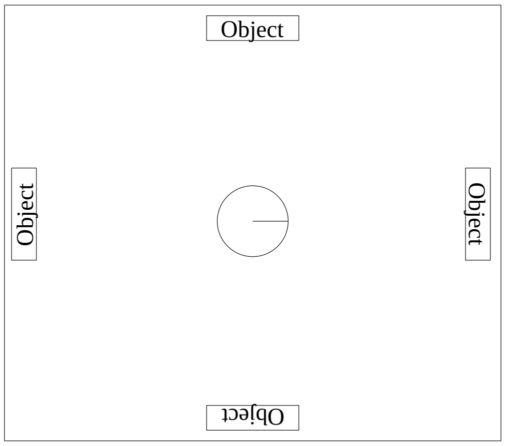


Figure 4: The setup for data collection. A Khepera robot moves around a world with the same object on each wall (or no objects for “blank” images). The images are then filtered to include only those that have contain a complete object.

## 4 Data Collection

Data was collected using a camera mounted on a Khepera robot. Figure 4 shows the data collection setup. The robot world is a 16” × 16” square world with white walls taller than the camera can capture. There were either four identical objects (constructed from red Lego blocks) in the places labelled “Object” or there were no objects in the world. The robot ran a simple wander algorithm with obstacle avoidance, taking a camera snap shot once every few seconds. We collected 1000 images at a resolution of $30 \times 25$ gray scale pixels for each of the configurations. After the images were collected, we filtered out approximately $\frac{3}{4}$ of them by requiring that images used for training have a white border around the outside, and have a certain number of dark pixels (the images with no objects were not filtered.) The white walls appear significantly whiter than the red blocks, and for most of the filtered images, a human can easily tell what objects is shown. We show images that the robot captured in each of the compression sections, since the objects differ between experiments.

## 5 Experiments and Results

We first performed several experiments to train a compressor capable of reproducing high quality images. This is important because if the compressed image is unrecognizable, then the hidden layer activations might not contain enough information to distinguish between the different objects. Compressing the images is a necessary step because running a genetic algorithm on the original images is not feasible on the available hardware. The experiments related to the compressor are described in Section 5.1. The second set of experiments we performed were on the genetic algorithm. We tested this on a toy problem and also on the task of distinguishing between the different objects. The experiments we ran relating to the genetic algorithm are described in Section 5.2.

### 5.1 Training the Compressor

The compressor and decompressor consisted of an auto associative neural network with a small hidden layer. An input layer of 750 nodes was fully connected to a 20-node hidden layer. This was fully connected to a 750 node output layer. This network was trained to reproduce its input activations on its output layer, using back propagation of error. The network was implemented using the Con-x neural network package. The learning parameter epsilon was set to 0.2, and momentum to 0.1. These values were chosen arbitrarily, but were consistent throughout the experiments. The error tolerance for an input and output node to be considered the same was 0.05, and was also consistent throughout the experiments. We chose this small value, because we found that it produced sharper compression – the compressed images had less blur.

#### 5.1.1 Experiment 1: Compressing Raw Data

Initially we attempted to compress the raw images, by assigning each of the 750 pixels to an input and output node. Since the image is gray scale, the activation value for each image pixel was affected by lighting conditions, wall proximity as well as the color of the object observed. Figure 5 shows a sample of the source images used for these initial experiments. There were also "blank" images presented in the training data. These are analogous to the images in the figure, except that no dark object is present. They were obtained by allowing the Khepera robot to take pictures of its environment when no objects were present.

Figure 6 shows the error and percentage of correct node activations over 32 epochs of training for a compression network with 20 hidden nodes. The error is the sum of the Euclidean distance from the input vector (input layer activations) to the output vector (output layer activations) for each of the 1250 images in the training set. The percentage of correct nodes is the percentage of the output nodes whose activation is within tolerance (0.05 units) of the input for the corresponding node.

The figure shows that no performance improvement was achieved after the first 12 epochs of training, and that the percentage of correct nodes is never more than 75%. While this might initially appear to be an acceptable performance, more careful analysis reveals that this is not the case. An image that has 25% of its pixels incorrect has more than 180 pixels wrong. Since there are normally much less than 100 pixels dark in each training image, this level of error allows the loss of all the information contained in the images! To confirm this hypothesis we looked at some compressed images (the output layer activations of the network reformatted into visual form). Figure 7 visually illustrates some examples of the performance. These images were selected because the object in the source image is larger than average, meaning that these examples illustrate some of the easiest to distinguish images from the data set.

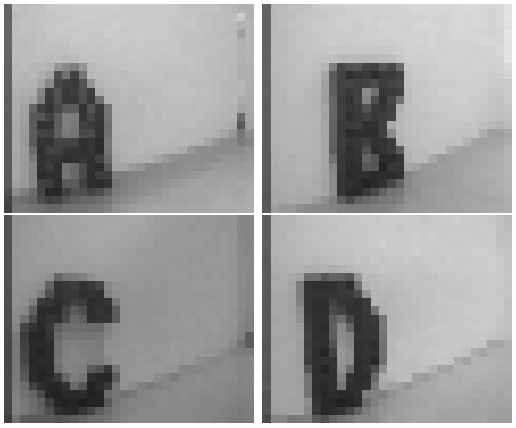

Figure 5: Sample raw images from the set of objects used in Experiment 1. Note the difference in shade of the background between different images and within each image. For example, the center of the "C", is much darker than the rest of the background.

We verified the disappointing results for Experiment 1, by performing several experimental runs. Due to space constraints we are only able to show the results for the initial run, but the results for the remaining runs are comparable. We also tried increasing the hidden layer size to 30, 40, and 50 nodes, and found that this did not give a noticeable performance improvement. The compressed images from the 50-node hidden layer network were still unrecognizable and no more than 80% of the pixels were correct. A 50-hidden node compressor network would also have produced larger compression images making our genetic algorithm prohibitively slow on the available hardware.

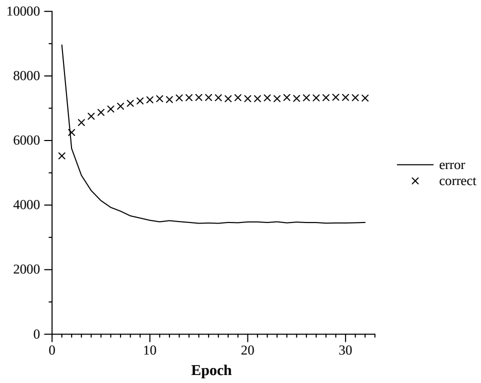


Figure 6: Results for Experiment 1 – The error and percentage of correct output layer activations over 32 epochs of training. The percentage has been scaled to 100 times its value in order to be visible on the same graph as the error. The maximal value on the $y$-axis represents 100%.

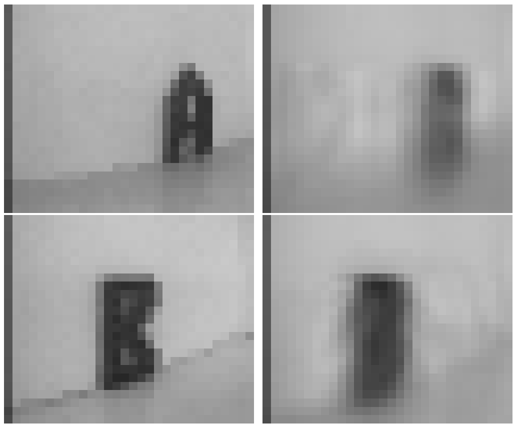

Figure 7: Results for Experiment 1 – Sample compression performance. The left column shows the uncompressed image, while the column on the right shows the corresponding image after compression and decompression has been performed. Note that this leaves most of the images unrecognizable even by a human.

#### 5.1.2 Experiment 2: Implementing a Pre-Preprocessor

In order to simplify the compression process, we converted the gray scale images to bit maps, where a pixel is either dark or light, but not somewhere in between. While computationally inexpensive, this step is a very big problem simplification for a number of reasons. Firstly, we are providing a hint to the neural network as to what information is important. It is not important to distinguish between the different shades of "light" in the background or shading on the object, but it is important to distinguish between the light background and dark objects. Secondly, the training data becomes more consistent, since there is only one way to represent a dark pixel. Consequently, the network does not need to have a way to produce a gray pixel, since this will never be required. Once we converted the training data in this way the experiment is the same as Experiment 1. Figure 8 shows the results.

The numerical results are much more promising than in the previous experiment. The network's performance exceeded 90%, and learning began slightly more quickly than before. For some objects – most notably the "B" and the "C", the performance difference is visibly better. Figure 9, shows a "B" and "C" that have been compressed. While the compressed images are not exactly true to the original, they are definitely recognizable.

We can contrast this improved performance with the weaker performance of the other two objects "A" and "D". Very often these were simply compressed as a fuzzy box around the position of the original. More interestingly, the compressed "D" shown in the figure looks like a "B", which would suggest that the compression network is abstracting from the pixels, and generalizing to objects. In this light, the corrupt "D" is a misinterpretation of a "D" by the network as a "B". A further problem that we discovered was that the bitmap images were often difficult to recognize even before compression, due to their size and the similarity between the shapes.

#### 5.1.3 Experiment 3: Changing the Objects

The results from the previous experiment showed that bitmap images were easier to compress than gray scale. This is not an unexpected result, since the images become more uniform in this way – each pixel can be one of two values, rather than one of 256 values. Unfortunately this initial pre-processing made many of the images in our data difficult to recognize even before compression. Our compression method preserves many large features of the images, but fails to preserve some small features, such as one or two white pixels in the center of a black cluster; however, this information became crucial for object recognition (measured visually by the authors). In order to gain better training data for the genetic algorithm experiments in Section 5.2, we collected new training data using the new objects. A bitmap for each of the objects used in this experiment are shown in Figure 11. We also found that blank images were always compressed correctly in the previous experiment, and since these are also disproportionately

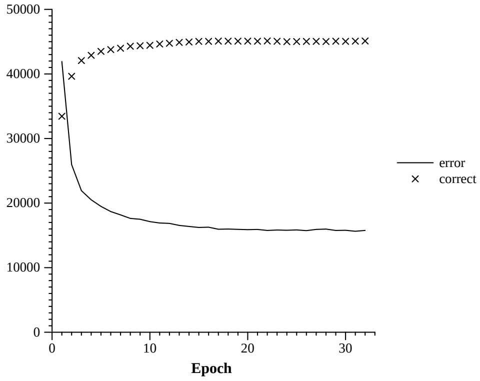


Figure 8: Results for Experiment 2 – The error and percentage of correct output layer activations over 32 epochs of training. The percentage has been scaled to 500 times its value in order to be visible on the same graph as the error. The maximal value on the $y$-axis represents 100%.

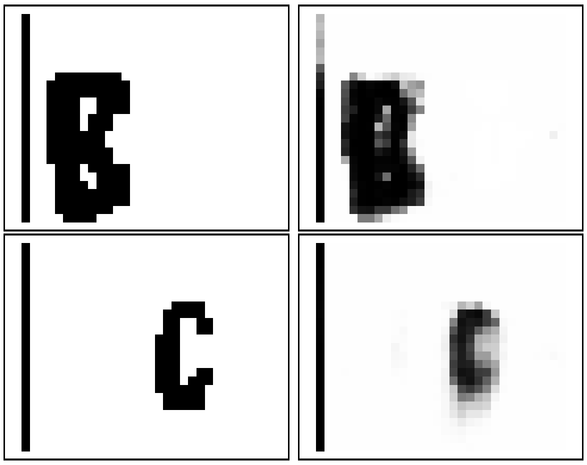

Figure 9: Results for Experiment 2 – Sample compression performance on bit map images. The left column shows the uncompressed image, while the right column shows the corresponding image after compression and decompression has been performed. "B" and "C" were compressed much better overall than "A" and "D".

easy to recognize, we eliminated them from the data set.

Because the source images were more easily distinguishable than in Experiment 2, the compressed images were also easier to distinguish. This was not reflected by the error rate and percentage of correct output nodes as shown in Figure 12. The percentage of correct output nodes is never more than 85.5%, compared to 90% achieved in the previous experiment. Repetitions of the experiment showed that this was a consistent result, and longer runs did not yield visible improvement over performance after 32 epochs of training.

While the numerical performance was worse in this experiment, the qualitative results were much better. Figure 13 shows the images in 11 after compression. The compressed images in the figure are much easier to recognize than the compressed image in Figure 10. While the images with smaller objects were more difficult to distinguish, only the triangles showed really poor performance, often appearing as another object after compression. Figure 14 shows some examples where the compression did not perform as well as we would have hoped.

There are a number of reasons for the apparent discrepancy between the quantitative and qualitative results. Firstly, because the new data set does not contain blank images present in present in Experiment 2, the percentage of correct nodes is not directly comparable. It is easy for the network to reproduce a blank image correctly, so the presence of blank images would affect the average number of correct output nodes. As an estimate of this effect, if every fifth image is blank, then the percentage of wrong output nodes would be $\frac{5}{4}$ times as much as without the blank images. This estimate suggests that without the blank images, the performance would have been 87.5% in Experiment 2, leaving a small discrepancy. This can be accounted for in that the objects were overall slightly larger than those in Experiment 2, meaning that the data would have more edges between dark and light, which is where most of the pixels differ.

#### 5.1.4 Experiment 4: Further Massaging the Data

The qualitative results in Experiment 3 were very encouraging, but some of the images were still impossible to recognize after compression. In particular, the "O" and the square were very similar and could be confused after compression, and the triangle shape was compressed poorly in many cases. In order to test the genetic algorithm used to generate the black box, we wanted to make sure that there was enough information in each compressed images to recognize the object. To this end, we collected a third data set. This differs from the data set if Experiment 3 in three major ways: the "O" object was modified so that it was sharper at the top and bottom; the triangle was replaced by a "+", and more data was collected, to allow more aggressive filtering. The final data set kept one out of about forty collected images, compared to about one out of four for the previous data sets. Figure 15 shows each of the objects used in this experiment.

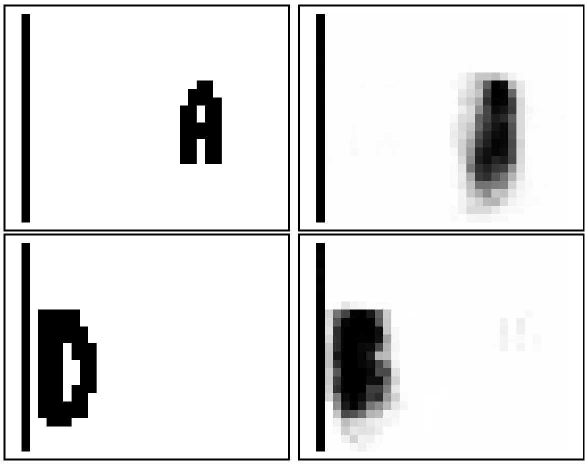

Figure 10: Results for Experiment 2 – Sample compression performance on bit map images. The left column shows the uncompressed image, while the right column shows the corresponding image after compression and decompression has been performed. “B” and “C” were compressed much better overall than “A” and “D”. Note that the compressed “D” looks a lot like a “D”.

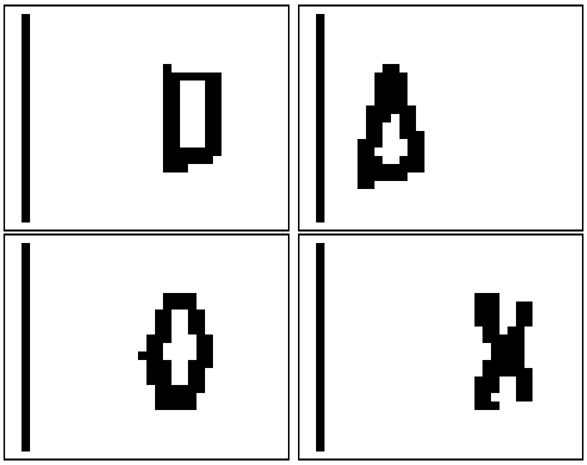

Figure 11: A sample image for each of the objects used in Experiment 3.

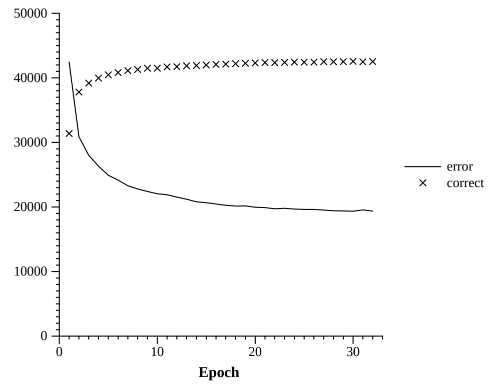


Figure 12: Results for Experiment 3 – The error and percentage of correct output layer activations over 32 epochs of training. The percentage has been scaled to 500 times its value in order to be visible on the same graph as the error. The maximal value on the $y$-axis represents 100%.

With this new data, the compressed images resemble their uncompressed counterparts very closely. Figure 16 shows the images in Figure 15 after compression. In each case, it is very clear from the compressed image which object is being shown. This means that the hidden layer activations contain enough information to distinguish between the images of different objects, which means that they contain enough information for our genetic algorithm to associate a compressed representation of an image with the object shown in the image. Due to space constraints, we do not show the learning curves for this experiment, but they do not differ significantly from those for the previous experiments. The network learns quickly at first, and performance improves rapidly until about the $10^{th}$ epoch of training, when the average percentage of correct nodes is almost 80%. Further training allows up to 83% of the nodes to be within tolerance of the correct values. The reason that this score is lower than the score for Experiment 3 is that with the larger images, there are more “edge” pixels – pixels that are just on the edge between white and black. The compression blurs the edges of the objects, so these pixels are the most difficult for the compression algorithm.

### 5.2 Evolving a Black Box

The black box is the second major component of our project architecture. Rather than hand-designing an algorithm to associate compressed images with the object they depict, we wanted to investigate the possibility of using a genetic algorithm to evolve a neural network that could do this. Our population consists of neural networks with the architecture described in section 3.3. Each network is represented using a list of floating point real numbers. The numbers record the weights and biases of each connection and node in the network. During each generation, the weights and biases are mutated, and a node might be added or removed at random from an individual. The number of hidden nodes is bounded within 1 to 80 inclusive. That is, no individual would have more than 80 hidden nodes or fewer than 1. Because individuals generally have different numbers of hidden nodes and cross-over would have been arbitrary, we did not allow cross-over in our experiments.

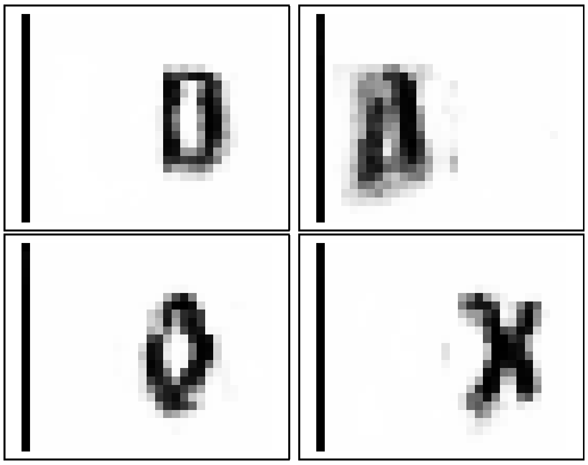

Figure 13: Results for Experiment 3 – The images in Figure 11 after compression.

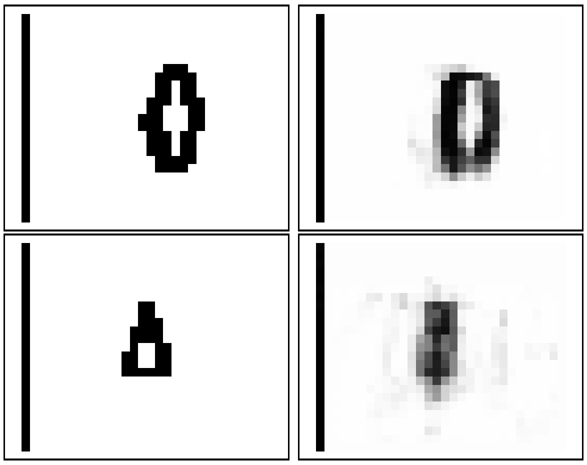

Figure 14: Results for Experiment 3 – Two examples where the compression was not very effective. The compressed images are to the left of the originals. The "O" could be mistaken for a square after compression, and the triangle has become completely unrecognizable.

### 5.2.1 Experiment 5: T and H

As a proof of concept for our genetic algorithm, we first evolved a simple network to distinguish between two ASCII art style letters: T and H. Each letter is represented by a 5x5 array of 1 and 0 as shown in Figure 17. To make the image easier to look at, the 0's have been omitted from the figure. The 25 elements of each array were fed to the input layer of the genetic black box. The output was either 1 or 0 to indicate T or H. Noise was introduced into the data by, randomly changing 0's to 1's and vice versa. Each input node had an 8% probability of being switched.

Our population consisted of 50 individuals and was initialized with random networks with four hidden nodes. Fitness is calculated using the Euclidean distance between the correct label and network output, so maximum fitness is 1, meaning the network gets every label correct. Figure 18 shows the results for this experiment. The fitness and number of nodes are for the best individual at each generation. The last generation fitness and number of hidden nodes of the best individual during each generation. The fitness at the end of evolution is 0.99, corresponding to the network getting the correct result 99% of the time. One interesting thing to note is that as the network approaches perfect fitness, the number of hidden nodes does not increase, but in fact decreases slightly. This is probably because of the simplicity of the problem we are using as a proof of concept.

### 5.2.2 Experiment 6: Real Images

In this experiment, we used 500 compressed images from the set described in Experiment 4. As in Experiment 5, we had a population of 50 individuals, initialized to random individuals with four hidden nodes. Unfortunately we did not observe any learning over 1000 generations – the fitness remained less that 0.01 for the best individual. Figure 19 shows the number of nodes in the best individual; we do not show the fitness, since that did not change during our training.

The number of hidden nodes fluctuated quite widely with no clear pattern, but fitness remained close to zero throughout the experiment. We hypothesize that the reason for the poor performance is the large number of images given to the network at the same time. Perhaps allowing the population to adapt to just four images, and slowly introducing more might allow learning to occur.

### 5.2.3 Experiment 7: Gentle Introduction

In order to test our hypothesis about the poor performance in the previous experiment, we designed a new experiment in which we gradually increased the number of images used in GA. As before, we created a population of 50, each with 4 hidden nodes. Initially only 4 images were used in obtaining the fitness value. The small number of images used implies small cumulative errors, which results in a reasonable fitness value. As fitness improves beyond a threshold of 0.9, we double the number of images used. Repeating this process allows us to achieve reasonable performance on even a large number of images. Unfortunately, due to computational computational and time constraints we were not able to extend this method to more than 64 images. Figure 20 shows

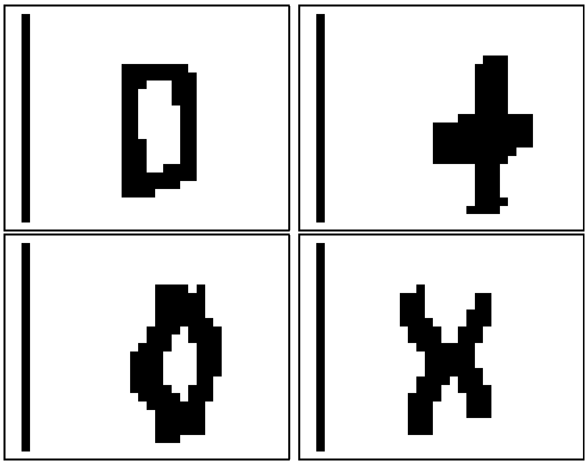

Figure 15: A sample image for each of the objects used in Experiment 4.

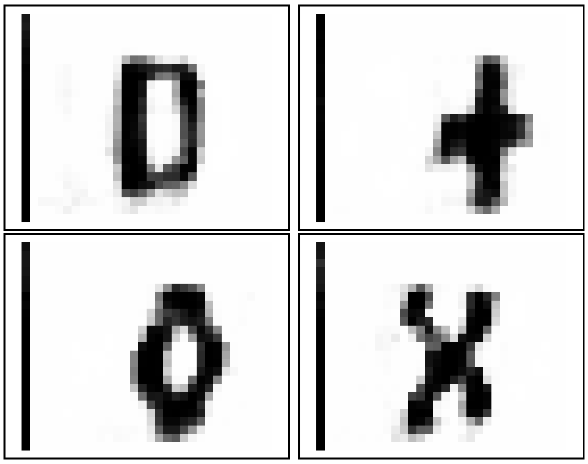

Figure 16: Results for Experiment 4 – The images in Figure 15 after compression. 4.

the results. We observe a pattern in the fitness, as it relates to the number of images. After the number of images is doubled, the fitness drops sharply, but there is a growth spurt immediately after this. For a small number of new images, this is enough to allow the network to reach a sufficiently good performance for the number of images to be doubled again. Once the number of images gets high enough, the initial spurt of growth is not enough to allow sufficient performance, and there is a plateau in the fitness. Eventually there is another spurt of growth that allows the individual to move to more images. This spurty growth could be likened to acquiring a new concept: when new data is introduced, generalizations can be made quickly, but eventually the immediate benefit of the new data is exhausted, and learning plateaus. After some time a new concept is

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | | 1 | | | | 1 |
| | | 1 | | | | 1 | | | | 1 |
| | | 1 | | | | 1 | 1 | 1 | 1 | 1 |
| | | 1 | | | | 1 | | | | 1 |
| | | 1 | | | | 1 | | | | 1 |

(T) (H)

Figure 17: Representations of T and H using arrays of 1 and 0 (not drawn).

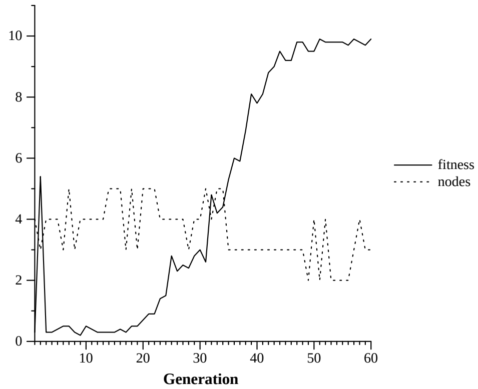


Figure 18: Results for Experiment 5 – The fitness and number of hidden nodes of the best individual after each generation. The fitness has been scaled by a factor of 10 to be more visible on the graph, so 10 represents a fitness of 1.

generated, and there is another spurt of growth. This analysis, however, is contradicted by the number of nodes in the black box, which increases significantly with the number of images, suggesting that the network is memorizing the images.

## 6 Conclusions and Future directions

The work that we present as well as our processing architecture is conceptually divided into two parts – the compressor and the genetic algorithm. We had some success with each part, but our system was not able to solve the task of associating images with labels. This leads us to the conclusion that perhaps the task needs to be divided into more than two parts in order to achieve better performance. Section 6.1 discusses our conclusions with regards to the compressor, while Section 6.2 discusses those with regards to the genetic algorithm. Finally Section 6.3 discusses the future work which our results may lead to.

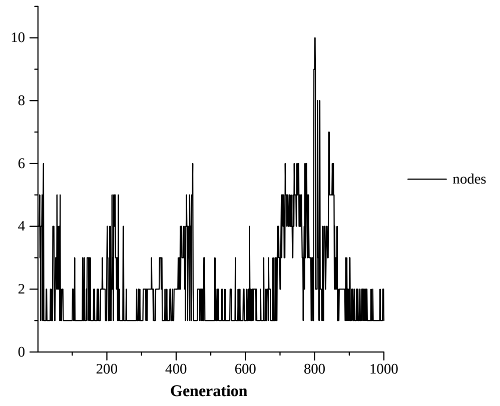


Figure 19: Using GA to evolve network. The network was unable to learn.

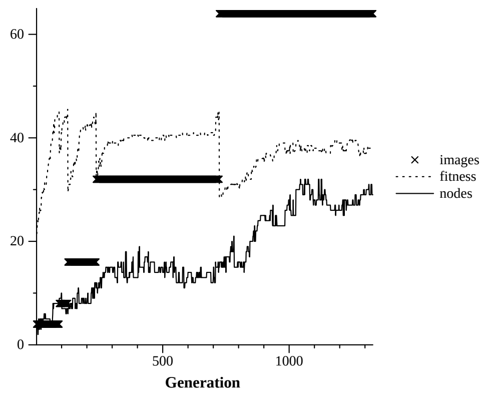


Figure 20: Results for Experiment 20 – The number of nodes and fitness of the best individual for each generation, as well as the number of images used. The fitness has been scaled to 50 times its original size, so that it is visible on the graph.

### 6.1 Compressor

We found that a feed forward Neural network with a narrow hidden layer can be an effective form of compression for images that will later be used by a neural network. We found also that this works better on bitmap images than on gray scale images. Since the overall effect of the compression is to blur the images, images with large objects were easy to tell apart even after being compressed, while images of small objects were often indistinguishable after compression. Surprisingly, the shape of an object can have an impact on how well it is compressed. For example, the "x" was compressed far better than the triangle in Experiment 3.

### 6.2 Genetic Algorithm

As a proof of concept for our genetic algorithm, Experiment 5 showed that we can evolve solutions to simple problems using this approach. We were also able to apply the genetic algorithm to the task of associating images with labels, but only if we introduced the images a little at a time. The necessity of doing this as well as the increase in the number of hidden nodes of the best individual suggests that the neural networks are not learning to generalize about objects in the world, but are just memorizing an increasing number of images. As a result, we conclude that our approach to a genetic algorithm works, but that the problem of associating real images with their labels is too difficult to be feasible using this technique.

### 6.3 Future Work

There are several lines of future work that stem from the present paper. Firstly, the compression experimental results leave a number of open questions that future work could explore. What images are hard to compress in the way that we describe, and why is it more difficult to compress these than other shapes? For example, why was the "X" easier to compress than the triangle? Also with respect to compression, it might be interesting to know why compressing bitmaps worked so much better than compressing gray scale images. The results for the genetic algorithm open the question about gently introducing more and more training data to a GA. Is this more likely to lead to memorization, as we suspect, or can this lead to abstractions that might have been otherwise elusive?

## References

[1] Douglas Blank, Lisa Meeden, and Deepak Kumar. Python robotics: An environment for exploring robotics beyond legos. *ACM Special Interest Group: Computer Science Education Conference*, 2002.

[2] Feng He and Kuzman Ganchev. Associating images of simple objects with labels using self organizing maps for preprocessing. Swarthmore College Developmental Robotics class midterm project, 2003.

[3] K. Plunkett, C. Sinha, M. Moller, and O. Strandsby. Symbol grounding or the emergence of symbols? vocabulary growth in children and a connectionist net. *Connectionist Science*, 4(3 and 4), 1992.

[4] Mark D. Schroeder. Jpeg compression algorithm and associated data structures. http://www.cs.und.edu/ mschroed/jpeg.html, 1997.

[5] Bernard Widrow. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9), 1990.