Report 3: Classification and Object Recognition

Eric Scott, supervised by Roy Villafañe, Andrews University

27 April, 2009

For CPTR 495, Independent Study in Computational Intelligence

1 Computer Assisted Taxonomy

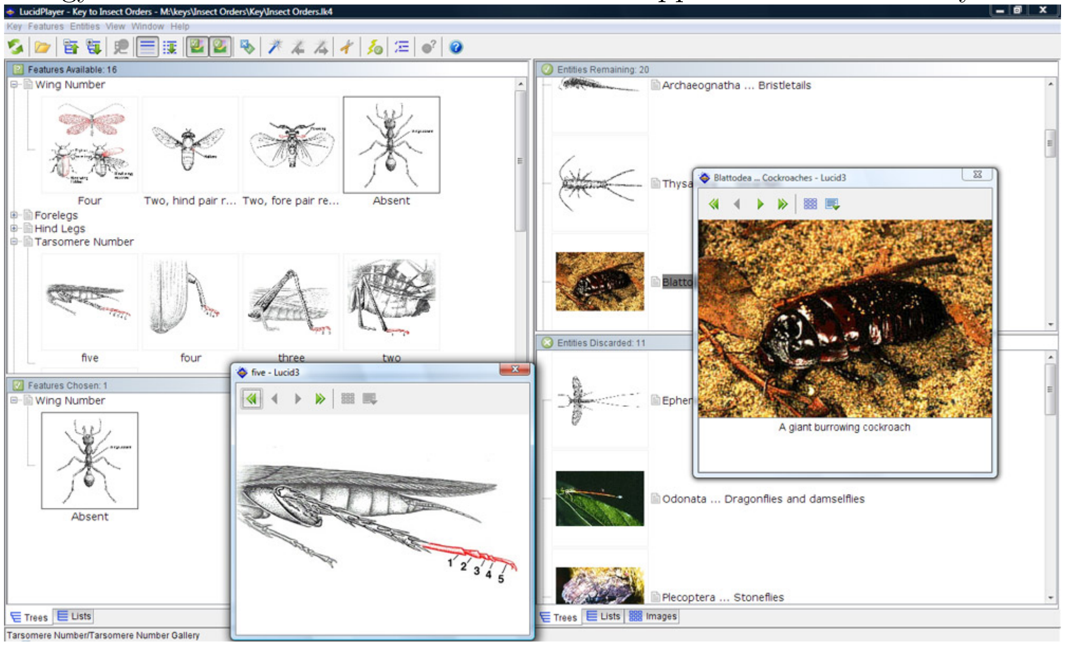
Some of Artificial Intelligence's greatest accomplishments are in the area of pattern recognition. Data mining, optical character recognition (OCR), and speech recognition have enjoyed relatively significant success over the past few decades. *Computer vision*, too – the study of object recognition and spatial awareness algorithms for images and video streams – is often seen as not only an application area but a subfield of AI and robotics. With the advent of the Internet, the concept of *content-based image retrieval* (CBIR) has become an important research fronteer, as the need to organize multimedia databases and make them easy to query has become quite pronounced. In a CBIR system searches are executed against the actual content of the images via automatically generated feature tags instead of human-defined metadata, since human classification of the images is expensive. In this sense CBIR can be seen as content-addressable memory.

These tools serve as a possible solution to what has been called the "taxonomic impedement" to biodiversity studies. In short, taxonomy is hard, with millions of known bug species, and with paper guides that make it difficult to identify specimens without being an expert already. The taxonomic workforce is small, making research on the distribution of bug populations difficult.[1] A sample of ten thousand bugs mechanically collected from a Mis-

[1]Kevin J. Gaston and Mark A. ONeill, "Automated Species Identification: Why Not?", *Philosophical Transactions fo the Royal Society* (2004) 359, 655667.

souri cornfield holds a lot of information regarding migrations, responses to pesticides, climate, etc, but it's a time consuming and expensive process to manually identify and record the specimens even on the most general level.

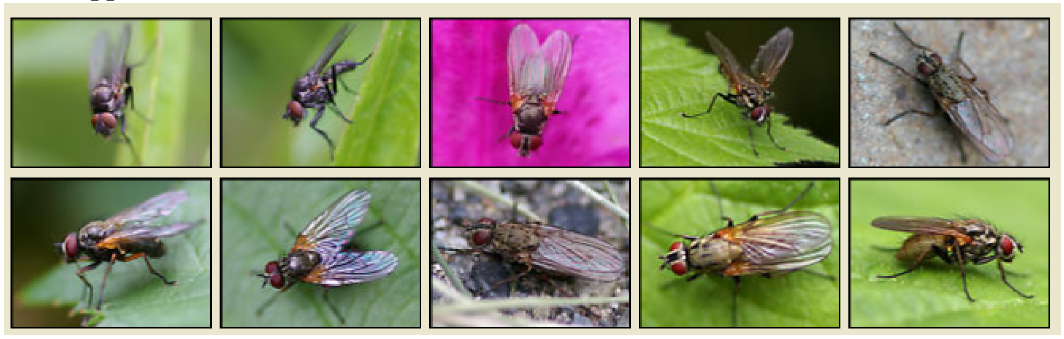
Figure 1: Expert systems such as the Centre for Biological Information Technology's Lucid 3 aim to streamline traditional approaches to taxonomy.


LucidPlayer - Key to Insect Orders - M:\keys\Insect Orders\Key\Insect Orders.lk4
Key Features Entities View Window Help
Features Available: 16
Wing Number
Four
Two, hind pair r...
Two, fore pair re...
Absent
Forelegs
Hind Legs
Tarsomere Number
five
four
three
two
Features Chosen: 1
Wing Number
Absent
five - Lucid3
Entities Remaining: 20
Archaeognatha ... Bristletails
Thysa
Blatto
Blattodea ... Cockroaches - Lucid3
A giant burrowing cockroach
Entities Discarded: 11
Epher
Odonata ... Dragonflies and damselflies
Plecoptera ... Stoneflies
Trees
Lists
Images
Tarsomere Number/Tarsomere Number Gallery

Probably the world's largest database of insect and arachnid photographs can be found in Bugguide.net, an online community hosted by the Entomology Department of Iowa State University. It contains nearly two hundred thousand images classified and discussed by experts and enthusiasts, assembled to form a field guide and record of bug locations. Quality control of the initial classification specified by the photagrapher is a difficult but pertinent priority for such a project, and the ratio of new images to experts is disproportionate. Furthermore, a specialist in one order or family may prefer to review images only in his area of expertise, making the process of skimming through new images much like filtering junk mail.

A CBIR-based classification system, i.e. *Computer assisted taxonomy* (CAT), could thus provide a significant ergonomic advantage to the community, even if it could only distinguish between orders, differentiating between, say, beatles, flies, and spiders. The high level of human involvement in the

Figure 2: Some of the roughly 23,500 images from the order *Diptera* (flies) on Bugguide.net



community also provides ample opportunity for supervised on-line learning of the system, as any misclassification could be readily noticed and corrected by a human. Current approaches to insect identification are based on expert systems or on classifiers which require a simple artificial background to the image. Applications of CAT technology extend beyond resources like Bugguide, as robotic collection and sampling systems can take significant measurements of bug populations much more quickly and at a much lower cost than humans. The Integrated Biological Control Program at New Mexico State University, for instance, has developed a fully automated bug sampling and identification system for informing agricultral pest control decisions.[2]

Bugguide poses a particularly interesting challenge for feature extraction. Some images are clear, well oriented, and on plain backgrounds, but as a rule the specimens are photographed in the wild on noisy backgrounds. Certain types of bees and wasps are often found perched atop flowers, and care must be taken to keep the classifier from learning to identify flowers or leaves instead of the insects sitting atop them. In some cases there are multiple species in a single image. Separating bug from background, then, is the primary task, the classification itself being relatively trivial.

These challenges, the available data, and the significant contribution offered by a CI-based classification system make Bugguide the perfect test bed for a student such as myself to tinker with pattern-recognition technology, and the code I write as a result of this paper will be applied to bug categorization. This project is ambitious and too grandiose to be completed for this independent study, but will likely continue on and become my senior

[2]http://www.nmsu.edu/biocontrol/

project for the John Nevins Andrews Honors Program.

2 Classification and Generalization

At the heart of a smart image database is the concept of *classification*, the process of algorithmically sorting complex and noisy data into meaningful categories. The algorithms and paradigms developed for such problems are an integral part of machine learning, and include various data mining tools, K-means algorithms, Bayesian methods, decision trees, linear regression models, kernel methods, and support vector machines, just to name a few. A comprehensive discussion of classification and clustering technologies would require an entire course in itself, but there is considerable overlap with CI tools: neural networks and fuzzy-rule-based (FRB) classifiers, for example, trained through evolutionary algorithms, are some of the most powerful solutions available.

A classifier, in its simplest definition, is a mapping from a highly dimensional feature space, in which vectors represent sample data (such as documents, images, or user-specific metadata), to a low-dimensional label space. A rudimentary classifier might define a correlation function, or distance metric, that defines how similar two vectors are (For example, Amazon.com compares users' purchase history to recommend books to users with similar interests). Often the metric is just the Euclidean distance between the vectors, but alternative methods such as the Pearson correlation coefficient or cosine distance allow the *length* of the vectors to be less important than their *direction*, effectively normalizing/disregarding the document length or image size. A more advanced system builds on these concepts to "cluster" vectors into discrete (or fuzzy) sets. K-means clustering, for example, determines center points to a pre-defined number of similar vector clusters, and a support vector machine (SVM) constructs hyperplanes in the space between clusters. Classification of new points is then determined by the nearest mean point or by which side of the plane they fall on, respectively.

These are examples of *unsupervised* learning, in which vectors in the label space are meaningful but do not necessarily correspond to pre-specified, human-defined categories such as "sci-fi fans," "pictures of cats," or "sunny days." Unsupervised learning is particularly useful for *knowledge discovery*, detecting patterns and categories that humans might not even have thought existed, and thus it is an important part of data mining.

Supervised learning, on the other hand, occurs when the classifier is provided with pairs of a sample datum *and* the human-defined label associated with it. The system is to conform to the mapping represented by the training examples (*minimize* the error, making this an *optimization* problem as discussed in report 2), building an internal model of the function it aims to imitate. This means A) no labels will be discovered that we don't already know about and explicitly request, and B) a significant quantity of training data must be provided for the system to be of any use. The advantage is that a properly supervised system can learn to *generalize*.

Generalization in CI is usually defined as producing the correct results on real-world data (as opposed to the samples it was trained with), i.e. accurately mapping vectors from the feature space to the label space. This process (and the term) hints at a deeper meaning, however, in the idea that the classifier is forming an abstract understanding of the problem at hand, as opposed to simply memorizing the answers to the training examples or building a statistical model (Which is relatively uneffective). The hidden layer of a neural network, for example, is sometimes called the "generalization layer," as the first layer of processing builds "meta-features" that help determine what is and is not important for the final decision in the second layer. In this way the computer can closely model human classification and even surpass human abilities (ex. learning to model complex nonlinear phenomena), and is thus what we want for a CBIR/CAT system (In which vectors in the label space are to correspond to potential human queries).

The road to generalization is marked by false-starts. Just when you think your system has the universe all figured out, a carefully crafted example might expose just how naïve it really is. Legend has it that in the 80's the Pentagon set out to train a neural network to recognize 100 camoflagued tanks out of 200 photographs. The network went through the training process on half the photos from each group with flying colors, but in the end gave random results when exposed to the remaining images. On closer examination of the training set, it was found that all the pictures with tanks had been taken on cloudy days, and all the ones without were from sunny days. The network was looking at the sky, not for tanks! For the classifier to develop an accurate mapping on real-world data points, it needs to develop a more general concept of what a "tank" is. More specifically, it needs to learn to distinguish between *relevant* and *irrelevant* features.

3 Feature Extraction

Many classification problems, if fed raw to the training algorithm, are simply intractable. Images, for example, may have millions of pixels. A classifier such as a neural network could conceivably learn to take the raw numerical bitmap and recognize such high-level entities as an individual's face. The number of features presented is immense, however, and it will take a very long time for the system to learn which pixels are relevant in a given context and which are not, not to mention the resources required to store and process large amounts of data. Without a large and sufficiently diverse set of training examples, too, the system can never achieve significant real-world success. Instance based learners (such as K-means), which rely on a distance metric, are particularly succeptible (exponentially, in fact) since irrelevant features cause large distances to form between members that are actually of the same category.

A very profitable shortcut, then, is to perform some preprocessing on the data, getting some of the generalization done ahead of time. Humans intuitively know, for example, that an object's location in space has no bearing on the object's type. We also think in high level terms about concepts such as curves, regions of similar color, texture, specularity, and so forth. A machine learning system works best when it can generalize data to a similar level of reasoning. It's inefficient for us to wait for even an adaptive classifier to develop this sort of generalization on its own: by far the most advantageous application of high-level features comes when humans can explicitly specify what is and is not important, discarding irrelevant data *before* feeding it to the classifier.

Feature extraction is a wide field composed of many clever techniques that help encode humans' common sense (and some not-so-common sense) into mathematics that can be automated during the preprocessing phase of classification. Below are some of the approaches useful to object recognition, among other things. I find these formulas to be particularly intriguing, as I've always been attracted to the visual-intuitive concepts that underly calculus, and here we have the ironic inverse: a "calculus of intuition."

3.1 Smoothing

We begin with noise removal. Small disturbances in the signal (i.e. large changes in color over a small area) are unimportant in most cases. If we were painting a picture, we might overcome such discontinuities by smearing or smudging the image a bit in the areas where the disturbance occurs, leaving only general features. Graphic designers often do just this, in fact, with digital image editing tools.

3.1.1 Gaussian Filter

One way to accomplish this mathematically is to consider a weighted sum of local pixel intensities (or color values). This is much like assigning each pixel (x, y) the average of its neighbors, with closer neighbors making a bigger impact than further ones. All we need is a defined way to describe how the relative importance of a neighboring pixel drops off with distance from (x, y). Many kinesthetic concepts like this can be modelled with simple mathematical functions that are easy to apply (a parabola for motion under constant acceleration, sinusoids for oscillations, etc). Smoothing, then, can be modelled nicely with the Gaussian function:

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x^2-y^2)}{2\sigma^2}} \tag{1}$$

where σ is the desired standard deviation (which is related to the width of the bell). A standard deviation of one or two pixels is usually sufficient to iron out small disturbances without destroying too much data.

3.1.2 Convolution

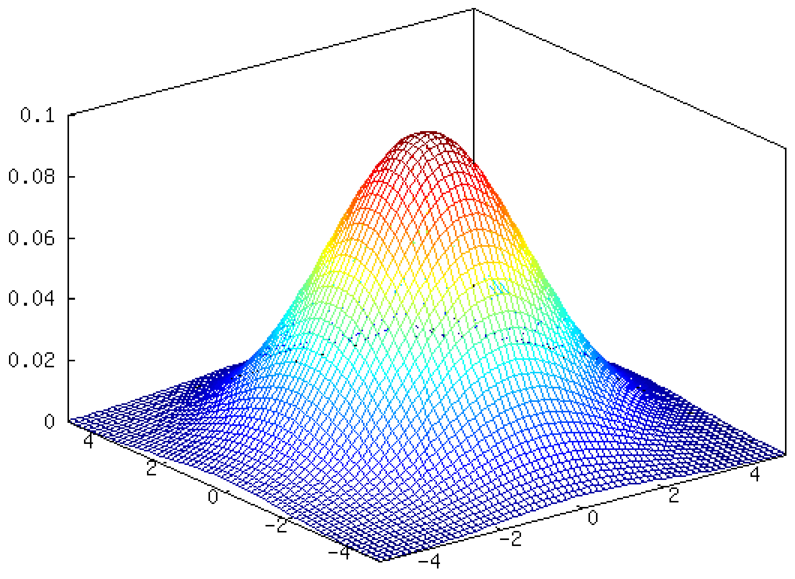
To apply the Gaussian filter we assign each pixel (i, j) the sum of $f(x, y)G_\sigma(i-x, j-y)$ over all (x, y), where $f(x, y)$ is the color value of the pixel (x, y). This type of weighted sum is known as the "convolution" of f with G_σ, denoted $*$. Convolution is a common technique used in many fields, from image processing to protein crystallography[3].

$$h(i, j) = f * G_\sigma = \sum_{x=-\infty}^{+\infty} \sum_{y=-\infty}^{+\infty} f(x, y)G_\sigma(i - x, j - y) \tag{2}$$

Because of the exponentially decreasing nature of the Gaussian, in practice using pixels more than a few standard deviations away from (i, j) has a negligable effect on the outcome. In the literature, then, the bounds of

[3]http://www-structmed.cimr.cam.ac.uk/Course/Convolution/convolution.html

Figure 3: A two-dimensional Gaussian function with $\sigma = 2$



summation are usually $[x = i - k\sigma]$ to $[i + k\sigma]$ and $[y = j - k\sigma]$ to $[j + k\sigma]$, respectively, where k is somewhere from 3 to 5, and $[\ldots]$ denotes the roundoff operation to the nearest integer.

3.2 Edges

Once small disturbances have been smoothed out, we can run another algorithm to detect *changes* in color distribution, or more specifically, *edges*. That is, if T is a prespecified threshold, a potential edge lies wherever

$$\frac{\partial f}{\partial x} \geq T \tag{3}$$

or

$$\frac{\partial f}{\partial y} \geq T \tag{4}$$

Of course, this method runs into trouble when, for example, a single pixel discontinuity is encountered. Thus, it is important to smooth the image first, since smoothing will eliminate small discontinuities, but not consistent differences like those found in a true edge. The two processes, smoothing and edge detection, can be combined in a highly advantageous way because of the following property of convolution:

$$(f * G)' = f' * G = f * G' \tag{5}$$

Here we see that we can avoid numerically sampling the derivative of the signal f by evaluating it for G instead, which is known (See equation 1):

$$G'_\sigma(x) = -\frac{x}{\sqrt{2\pi}\sigma^4} e^{-\frac{x^2}{2\sigma^2}} \tag{6}$$

Note that I've switched to the one-dimensional Gaussian. This is because, to detect edges at an arbitrary orientation (as opposed to purely horizontal or vertical), we will run the algorithm individually on the x and y dimensions and then combine the two like so:

$$R_v(x, y) = I(x, y) * G'_\sigma(x)G_\sigma(y) \tag{7}$$

$$R_h(x, y) = I(x, y) * G'_\sigma(y)G_\sigma(x) \tag{8}$$

$$R(x, y) = R_v^2(x, y) + R_h^2(x, y) \tag{9}$$

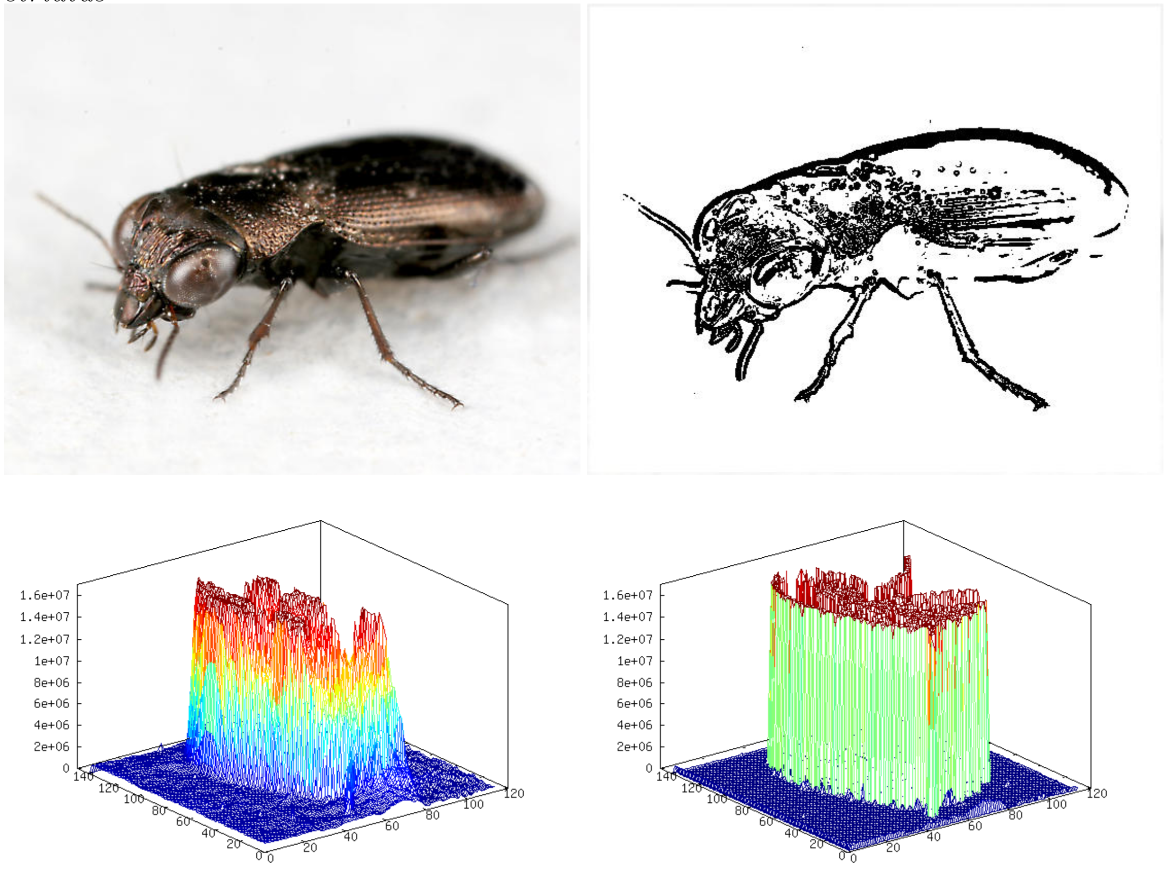
Then we mark any pixels where $|R(x, y)| > T$. Connected pixels can then be linked into unitary *edge curves* for further processing.

3.3 Fourier Transform

Image processing is closely related to signal processing, a field with wide applications and with which electrical engineers are particularly familiar. The Fourier transform is a special case of the bilateral Laplace transform, which converts a function $f(t)$ on the t domain into a function $F(s)$ on the s domain. That is, the Laplace transform is a function of the new parameter s, which was not present in the original function, and it is not a function of t. This is because the Laplace is an integral transform, and once evaluated all the terms with t in them become constant:

$$\int_{-\infty}^{+\infty} f(t)e^{-st}\,dt$$

Figure 4: The results of edge detection on a specimen of *Notiophilus Novemstriatus*



The disappearance of t implies that the transform represents *properties of the function as a whole* instead of only specific parts of it. The Laplace Transform is useful in situations where a problem is easier to solve in the s domain than the t domain. It often makes engineers' lives a lot easier, and for our current purpose is the same principle that motivates feature extraction.

The Fourier transform, then, comes about when we set $s = 2\pi i\xi$, $\xi \in \mathbb{R}$, effectively making F a function of ξ:

$$\int_{-\infty}^{+\infty} f(t)e^{-2\pi i\xi t}\,dt$$

This is important because, if you expand the exponential with Euler's formula and graph a few examples of the integrand, you should be able to convince yourself that the oscillations in the exponential resonate with certain frequencies in $f(t)$, depending on the value of ξ. In effect, the Fourier Transform produces a graph $|F(\xi)|$ vs. ξ that represents what frequencies ξ are present in $f(t)$ and how strong they are.

The discrete Fourier transform (DFT) can be used to process unknown functions by sampling them in discrete time and processing them digitally. One could decompose an audio signal, for example, into a Fourier series (A collection of individual sine waves) that could be used later to synthesize the sound of a trumpet.

$$F_k = \frac{1}{N}\sum_{n=0}^{N} x_n e^{\frac{2\pi i}{N}kn}$$

The DFT has a discrete domain *and* a discrete range, as opposed to the discrete time Fourier transform (DTFT) which has a discrete domain and a *continuous* range.

The fast Fourier transform (FFT) is a famous class of algorithms for computing the DFT efficiently. Technically, any implementation of the DFT which has $O(n \log n)$ complexity (as opposed to the $O(n^2)$ complexity of a naïve execution of the formula) is called a FFT, but the term usually refers to the Cooley-Tukey algorithm (re)discovered in the 60's (Thought it was known to Gauss as early as 1805). It can be used in one and multidimensional transforms (A multidimensional Fourier transform essentially involves running multiple single-dimensional transforms).

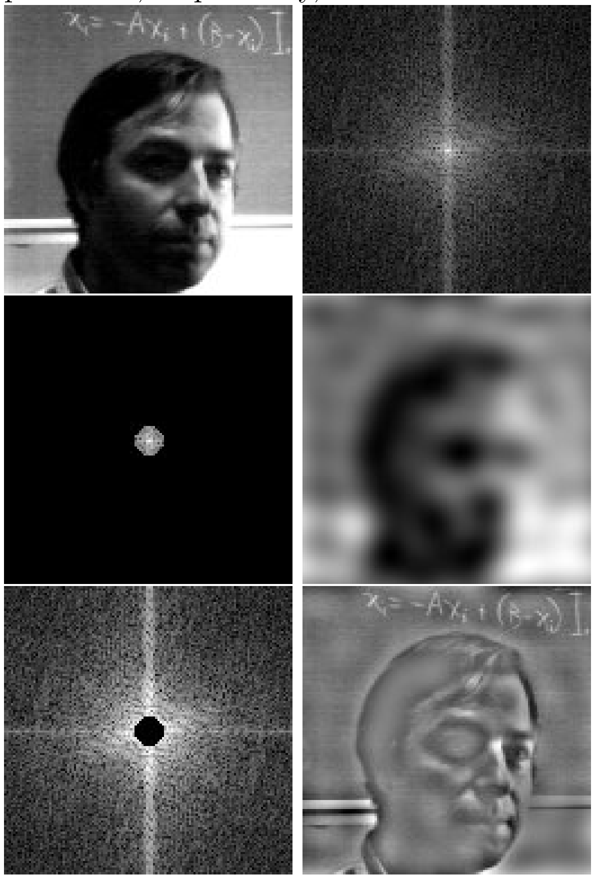
3.3.1 Frequency Filtering

We can use a Fourier transform to extract only general spatial features of the images (or, conversely, only fine features) by selecting only low (or high) frequencies from the output (See figure 5). Additional tools include band-pass filters (which specify a specific frequency range) and high-boost filters (which can help amplify edge features).

3.4 Wavelets

While the Fourier transform remarkably useful for a variety of image processing techniques, it is still somewhat clumsy where dimensionality is con-

Figure 5: An image and its Fourier transform, followed by a low and high-pass filter, respectively, with their inverses.



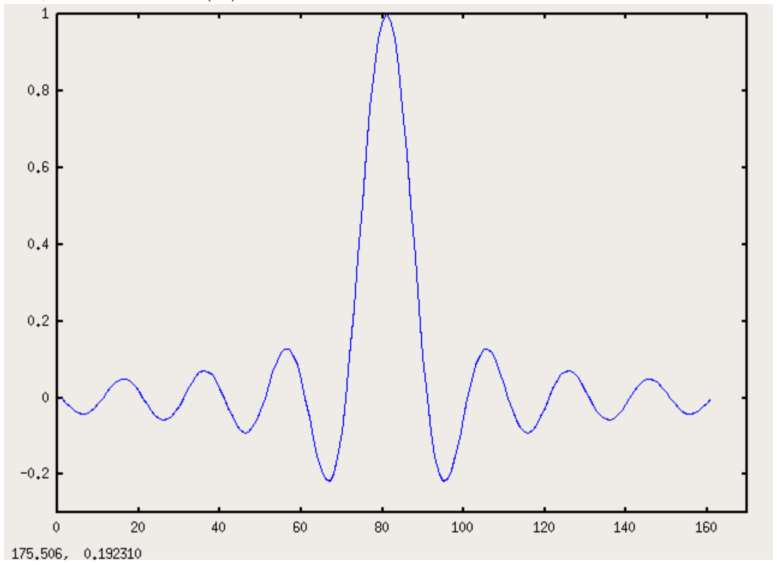
cerned since it represents images via a linear combination of general sinusoidal functions. While this makes representing repeating patterns trivial, isolated features are somewhat more difficult to model via this method. An alternative, then, is to choose a damped wave function ψ (such as *sinc*) and position it over the localized features in the function being modelled. This can be accomplished via the wavelet transform, which is the inner product of the wavelet basis function (sometimes called *child wavelets* $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}), (a \in \mathbb{R}^+, b \in \mathbb{R})$ and the signal $f(t)$:

$$W_\psi = < \psi_{a,b}(t), f(t) > \tag{10}$$

$$= \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} \psi^*(\frac{t-b}{a}) f(t) dt \tag{11}$$

Where $\psi^*(t)$ is the complex conjugation function of $\psi(t)$, and a and b are the scale and translation parameters, respectively.

Figure 6: $sinc(x)$ is a common "mother wavelet" for the wavelet transform.



3.5 Textures

One of the most important object recognition cues that humans use daily is visual texture. Abstracting these properties into feature labels would indeed give any computer vision system a significant advantage. This brings us to the domain of *texture analysis*, which deals with the supervised and unsupervised classification of different textures. Some approaches to the problem are based on principles we have already explored: fourier analysis, wavelets, and pre-defined primitives (such as edges) are all useful to texture classification. Other methods include stochastic and fractal modelling algorithms, and various statistical approaches.

David Mack, now a graduate of the Computer Science program at Andrews University, did his senior honors research project in the area of texture classification in 2003. He used a genetic programming approach to evolve classifiers for various textures, with statistical features extracted from the images' associated Grey Level Co-occurrence Matrix (GLCM).

The GLCM of an image A is an $\omega \times \omega$ matrix M_A, where ω is the maximum value of a pixel's intensity (”grey level”). A direction vector is selected as part of the definition of a GLCM, such as $(1, 0)$ (east), which defines a relationship between paired pixels. For example, for the eastern direction vector, every pixel $A(i, j)$ is paired with $A(i + 1, j)$. The value of the GLCM $M_A(x, y)$, then, records how many times a pixel with the value y is paired with a pixel with the value x. Note that in this case the rightmost elements are not paired with anything.

For example, if the elements of A are integers on $[0, 5]$, then

$$A = \begin{pmatrix} 0 & 5 & 2 & 3 \\ 1 & 3 & 3 & 4 \\ 0 & 2 & 3 & 3 \\ 5 & 3 & 2 & 2 \end{pmatrix}$$

yields the following GLCM for eastward pairing:

$$M_A = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

This can be read as follows: For the first row, we're told that pixels with the value of 0 are paired once with a pixel of the value 2 and once with the value 5. For the the third row, we're being told that pixels with the value of 2 are paired once with pixels of the value 2, twice with 3, and once with 4.

Equipped with this abstract statistical representation of the image, it is straightforward to compute several potentially useful features. The following table is taken from Mack's review:

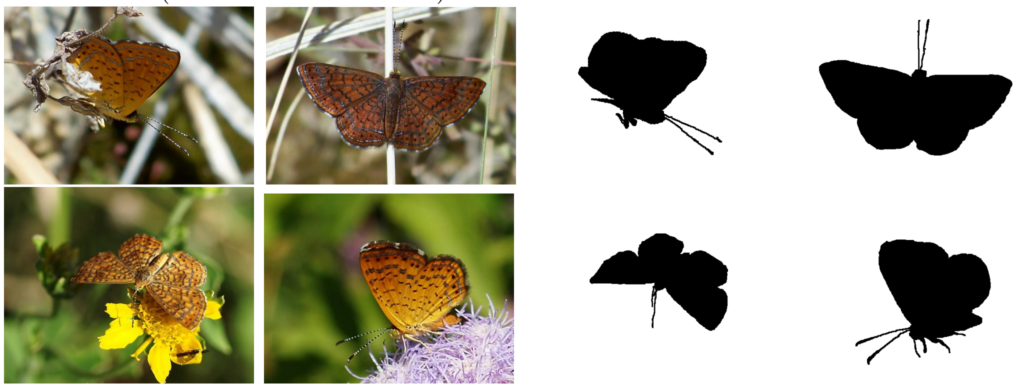
$\sum_i \sum_j (i-j)^2 P(i,j)$	**Contrast**: A measure of the local variation in the image
$-\sum_i \sum_j P(i,j) log P(i,j)$	**Entropy**: The inverse of uniformity. It is high when all elements of the GLCM have relatively equal values.
$\sum_i \sum_j (i-\mu)^2 P(i,j)$	**Variance**: The statistical variance of the distribution of value pairings.
$\sum_i \sum_j P^2(i,j)$	**Angular Second Moment**: A measure of textural uniformity. It is high when the distribution is constant or periodic.
$\sum_i \sum_j \frac{P(i,j)}{\lvert i-j\rvert^2}, i \neq j$	**Inverse Difference Moment**: The overall homogeneity of the image. It is high when the GLCM is concentrated along the main diagonal.
$\sum_i \sum_j \frac{(i-\mu_i)(j-\mu_j)P(i,j)}{\sigma_i \sigma_j}$	**Correlation**: A measure of linear dependencies within the image.
$\sum_i \sum_j [(i-\mu_i)+(j-\mu_j)]^3 P(i,j)$	**Cluster Shade**: Describes the clustering of pixel pairs using the third moment.
$\sum_i \sum_j [(i-\mu_i)+(j-\mu_j)]^4 P(i,j)$	**Cluster Prominence**: Describes the clustering of pixel pairs using the fourth moment.
$\sum_i \sum_j \lvert i-j\rvert[(i-\mu_i)+(j-\mu_j)]P(i,j)$	**Diagonal Moment**: A measure of the difference between correlation of high grey levels and low grey levels.

3.6 Segmentation

As our toolbox of features grows, so does our ability to differentiate between macroscopic components of an image. We can hope to distinguish between splotches of color on a butterfly's wings, which can help to identify its species, and to separate the insect from irrelevant background data. This process of dividing the image into chunks and evaluating macroscopic features is known as *segmentation*. Segmentation is vital to classification and models of associative memory. For example, models built to demonstrate how the brain might store and access visual memory use several Hopfield neural networks trained with Hebbian learning, each for a different feature such as color distribution, shape, and motion.[4] For our bug classifier we might begin by segmenting the image into regions of similar color or texture, or we might use wavelets to perform a more abstract differentiation.

Using these features as the input of a machine learning system, we can construct an ANN to perform *boundary extraction* on images of insects. Once the specimen's outline has been determined the background can be cropped out, making the classification problem much easier. This is what I'm currently working on, though as of yet I have done no tests to determine just how feasible this is.

Figure 7: Boundary extraction training images for specimens of *Calephelis Arizonesis* (Arizona Metalmark)



3.7 Boundary Curvature

Once the boundary of an object has been successfully determined, its general shape is one of the features that can be of use to our classifier. Its shape is, however, still defined by a rather low-level pixel representation. We consider the boundary to be a two-dimensional parametric equation:

$$\vec{r}(t) =< x(t), y(t) > \tag{12}$$

[4]Yaneer Bar-Yam, *Dynamics of Complex Systems* (1997), 343.

A more intuitive notion of the shape of an object can be provided by its *curvature*. Curvature is analogous to a higher-dimensional concept of the second derivate: it's the rate at which a trajectory's direction changes. In the present context it's simply a measure of how curved our boundary is at any given point. As any good Calc III student should know, curvature is defined as follows:

$$c(l) = \frac{|r'(t) \times r''(t)|}{|r'(t)|^3} = \frac{x'(t)y''(t) - y'(t)x''(t)}{[x'(t)^2 + y'(t)^2]^{3/2}} \tag{13}$$

Just as we did with edge detection, we can combine this process with smoothing by making the following substitutions:

$$\begin{aligned}
\dot{X}(t,\sigma) &= x(t) * \frac{\partial G_\sigma(t)}{\partial t} \\
\ddot{X}(t,\sigma) &= x(t) * \frac{\partial^2 G_\sigma(t)}{\partial t^2} \\
\dot{Y}(t,\sigma) &= y(t) * \frac{\partial G_\sigma(t)}{\partial t} \\
\ddot{Y}(t,\sigma) &= y(t) * \frac{\partial^2 G_\sigma(t)}{\partial t^2}
\end{aligned}$$

Where $*$ is the convolution operator. Then the smoothed curvature representation of the boundary can be computed as follows:

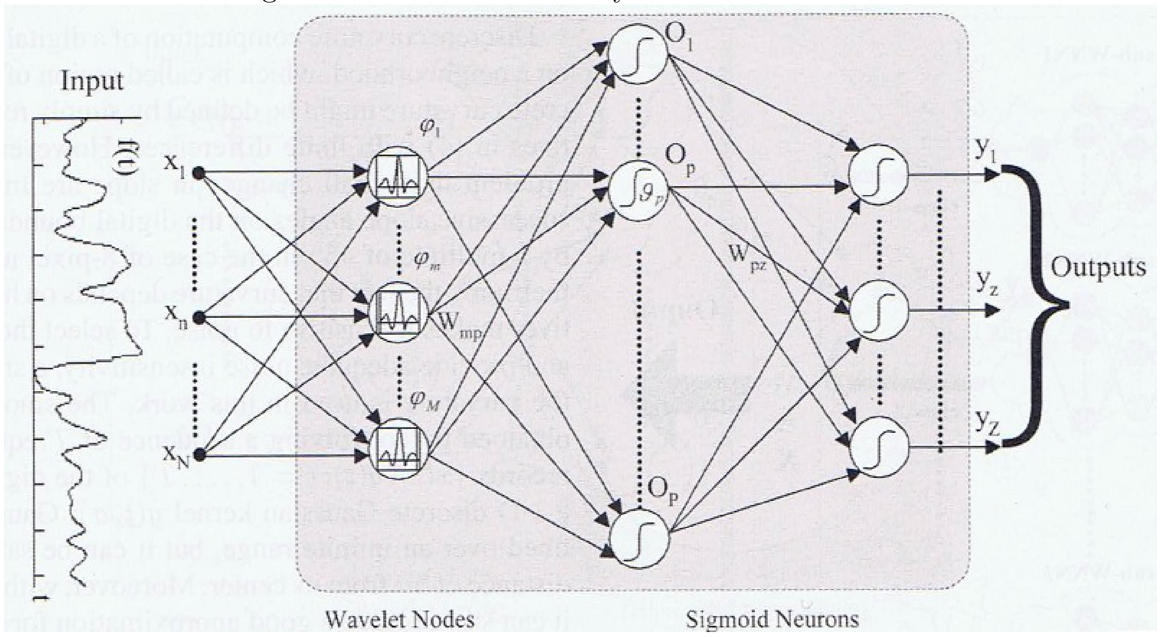
$$c(t,\sigma) = \frac{\dot{X}(t,\sigma)\ddot{Y}(t,\sigma) - \dot{Y}(t,\sigma)\ddot{X}(t,\sigma)}{[\dot{X}(t,\sigma)^2 + \dot{Y}(t,\sigma)^2]^{\frac{3}{2}}} \tag{14}$$

This function can then be processed via wavelets, etc, to produce an accurate, low-dimensional representation of the object's outline.

4 Classifier Architecture

Work utilizing many of the above tools was published this winter from Southeast University, Nanjing, China by Hong Pan and Liang-zheng Xia. They developed a wavelet neural network (WNN) that take the curvature representation of the boundary as its input. The first layer consists of "wavelet nodes" whose parameters are trained with the rest of the network via an iterative gradient technique (a backpropagation algorithm modified to handle the wavelet parameters). A separate sub-network is used to recognize each class of object. The system was tested on simulated outlines of aircraft, photographs of leaves from six species, and photographs of real-world tools. Gaussian noise and geometric transforms were used to artificially increase the number of sample images in each experiment to one to two thousand. All the images were, however, placed on a solid white background, making it trivial to distinguish the object border.

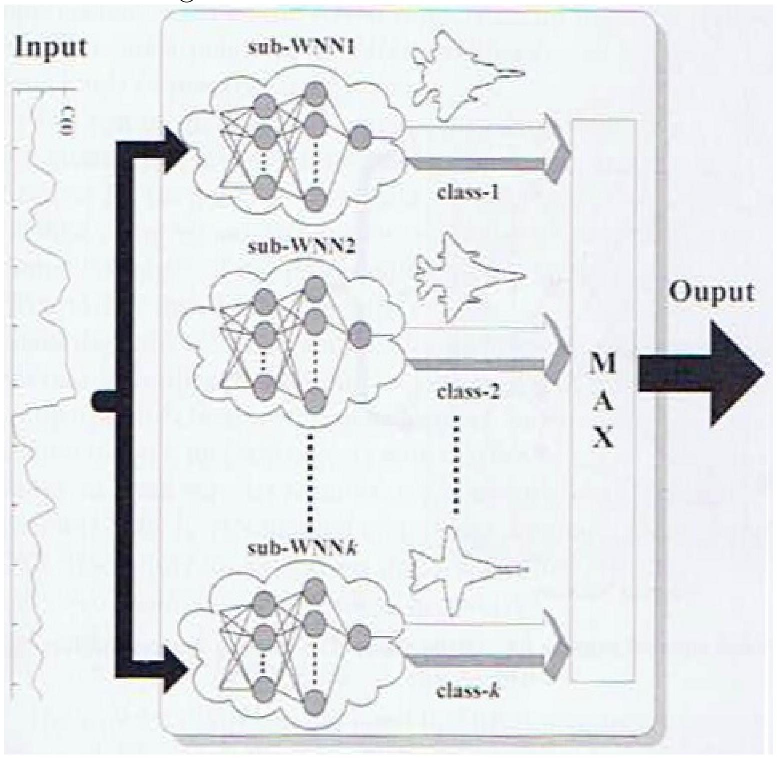
Figure 8: WNN described by Pan and Xia



Mack's project also used a separate classifier for each texture class. It is easier to train a classifier to provide a yes or no answer regarding a specific species of object than it is to train one large classifier to distinguish between several different species. This approach works fairly well when the specimen is neatly separated from its background.

For the images on Bugguide, however, boundary extraction will be a crucial and difficult process, and it is what makes the project relatively ambitious. One potentially useful feature for boundary extraction could be how blurry segments of the image are, since the background tends to be out of focus compared to the specimen. The large amount of data available from the website may be sufficient, too, to build a set of classifiers that learns to recognize common background objects such as leaves, sticks, etc, in effect building a system to memorize what is *not* a bug. These difficulties will be the focus of future work and experimentation.

Figure 9: Sub-network structure of the WNN classifier



5 Code Status

Edge detection has been implemented, though it's slow enough that I intend to research more efficient algorithms. I also need to experiment more with the threshold and standard deviation parameters to get more beautiful samples. Early attempts to feed raw image data into neural networks and genetic algorithms overloaded my code. This could imply that the linked structures I'm using in C# require too much overhead to be practical, but more likely the network was simply too large. Further feature extraction will reduce the dimensionality thousands of times over.

In short, there's still a lot of work to be done. Hitherto most of my time has been spent in literature review and learning about the mathematical tools at my disposal. More of that is needed as well.

6 Sources

I've learned about all of these topics from multiple sources over a long period of time, but in general most specific information was drawn from the following:

- Berman and Paul, *Algorithms: Sequential, Parallel, and Distributed* (2005): Fast Fourier Transform.
- Hall-Beyer, Mryka, "GLCM Texture Tutorial," v. 2.10, http://www.fp.ucalgary.ca/mhallbey/the_glcm.htm: Grey Level Co-occurence Matrix.
- Mack, David,"Visual Texture Classification Through Genetic Programming", senior Honors thesis at Andrews University, presented in 2003, supervised by James Wolfer of Indiana University South Bend: Texture Analysis.
- Pan and Xia, "Efficient Object Recognition Using Boundary Representation and Wavelet Neural Network", *IEEE Transaction on Neural Networks*, vol. 19, no. 12, 2132-2149: Boundary Curvature, Convolution, Wavelet Neural Network.
- Segaran, *Programming Collective Intelligence* (2007): Clustering, Classification.
- Russel and Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. (2003): Computer Vision, Gaussian Filter, Convolution, Edge Detection.
- Witten and Frank, *Data Mining* (2005): Clustering, Classification, Feature Extraction.
- *Wikipedia* and *Scholarpedia*: Everything else, give or take.

7 Time Spent

Date	Time Spent	Task
02/19/09	01:00	Read honors thesis of a previous AU student who worked with GAs
02/19/09	06:00	Debugged GA framework, implemented crossover, and created graphs of behavior with different parameters on a toy problem, directly and via setting weights on a neural network.
02/21/09	07:00	Implemented multithreading in the GA (Still buggy), tinkered with parameters more, realized the significance of squared and/or inverse errors vs. difference
03/19/09	02:00	Skimmed articles in IEEE Trans. on Neural Networks
03/20/09	02:00	Read parts of Yaneer Bar-Yam, Dynamics of Complex Systems, ch. 2, regarding Subdivided Hopfield Neural Networks and Content-Addressable Memory
03/24/09	05:00	Read more in Holland, and section 8.5 of Engelbrecht, and worked on draft of report 2
03/25/09	02:30	Worked on draft
03/25/09	02:30	Read D. Anderson et al, Modeling Human Activity From Voxel Person Using Fuzzy Logic, IEEE Transactions on Fuzzy Systems, vol. 17, no. 1, p. 39-49.
04/04/09	11:00	Read ch. 24 of Russel and Norvig, on Computer Vision and feature extraction for images. Read Wikipedia pages on object recognition, but found poor coverage of the topic. Read Pedrycz et al, Fuzzy Clustering With Partial Supervision in Organization and Classification of Digital Images, IEEE Transactions on Fuzzy Systems, vol. 16, no. 4. Sketched a design for Bugguide classifier. Started a report (3) on Classification and Object Recognition. Began an Sqlite backend for the Bugguide project, downloading and indexing 55 images for training.

Date	Time Spent	Task
04/04/09	04:00	Worked on draft of report 2.
04/05/09	03:00	Read Angelov et al., Evolving Fuzzy-Rule-Based Classifiers From Data Streams, Fuzzy Systems, vol. 16, no. 6.
04/05/09	09:00	Worked on draft of report 3. Implemented edge detection algorithm for feature detection as described in Russle and Norvig. Tested it on several bug photos and tinkered with the parameters.
04/06/09	02:30	Began manually highlighting silhouettes of bugs to provide training data for an ANN.
04/07/09	07:00	Indexed/Highlighted more bugs and wrote code to interface images with the net. Learned that pixel-by-pixel representation totally overloads both my GA and ANN code.
04/08/09	01:30	Read section in Engelbrecht on coevolution
04/09/09	02:30	Read up on Fourier transforms and low/high-pass filtering in image processing.
04/10/09	02:00	Wrote about Fourier transforms for report 3.
04/18/09	11:00	Read Pan and Xia, Efficient Object Recognition Using Boundary Representation and WNN, IEEE Transaction on Neural Networks, vol. 19, no. 12, 2132-2149. Wrote about smoothing, convolution, and edge detection for report 3. Started coding a smoothing function that's separate from edge detection.
04/19/09	08:00	Took a closer look at Pan and Xia. Re-read Mack's honors thesis on genetic programming. Wrote about texture analysis, segmentation, boundary extraction, curvature representation, and sub-classifier architecture for report 3. Had my code output data for edge detection to form 3D graphs for report 3. Installed the Weka datamining workbench and poked around.

Date	**Time Spent**	**Task**
04/26/09	03:00	Searched internet for articles relating to Computer Assisted Taxonomy Read Gaston and O'Neill, Automated Species Identification: Why Not?
04/27/09	03:30	Finished report 3.
For reports 2 and 3:	96:00	
To Date:	145:15	