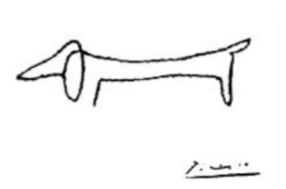
Lecture 18
Recognizing Dogs and Cats

Jason J. Bramburger

Now that we have done some image filtering, we are now going to talk about image recognition. For the next few lectures we will learn by doing instead of presenting some grand unified theory of image recognition. The goal will be to build a classifier that takes images and determines whether it is a dog or a cat. First, you should notice that this is a task you are very good at! You don't need perfect photos that are cropped or have optimal lighting to know the difference between dogs and cats. I could even give you a blurry picture and I'd bet you could still tell me the difference. Here's a question though: can you come up with a set of instructions for how to check if there is a cat or a dog in the picture? More importantly, could you provide them in such a way that a computer can follow them to determine if a cat or dog is in the picture? This is where things get hard.

This is a perfect example of a task where a data-driven approach (machine learning) is a good idea. Your experience with dogs and cats is what you are drawing on to differentiate them. Therefore, for our classifier, we are going to give the computer some experience (data) to help it to learn the difference too. We are going to use "training data" - 80 cat images and 80 dog images - that will allow the classifier to learn what is a cat and what is a dog. Once we train the model, we will (hopefully) be able to have the computer tell the difference between pictures of cats and dogs for us.

Your ability to recognize complex objects from poor data is best exemplified with the sketches below:


(a)

(b)

(c)

(d)

You can see that (a) is a dog, (b) is a bird, (c) is a camel, and (d) is Don Quixote and Sancho Panza. We can even tell that the dog is a dachshund and the bird is a dove. We got this all just from edges! So, based on our experience, it might be a good idea for our image classifier to be based on edge detection. Our strategy will be as follows:

1. Use a wavelet transform on each image to detect the edges.
2. Find the principal components of the wavelet transforms to see how dogs and cats differ in the principal component basis.
3. Use **linear discriminant analysis** to determine some threshold that separates cats and dogs.

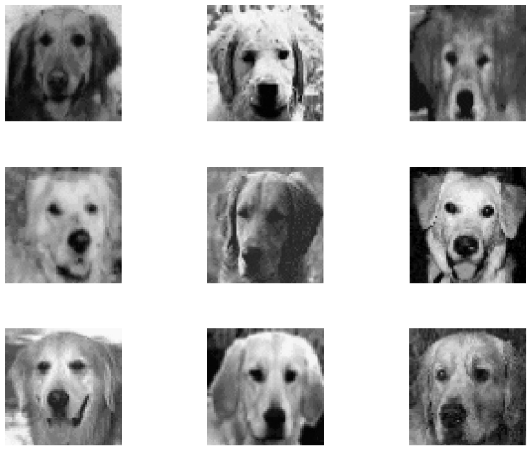
4. Test the algorithm on new data (test data) to see its accuracy.

The training data for this lecture will be contained in the MATLAB file dogdata.m and catdata.m. You will notice that these MATLAB files contain a single 4096×80 matrix. The 4096 data points correspond to a 64×64 resolution image of a dog or cat, while the 80 columns are 80 different images for the training set. We can load them with the following MATLAB command.

```matlab
clear; close all; clc
load('catData.mat')
load('dogData.mat')
```

Let's reshape and plot the first 9 dog images.

```matlab
figure(1)
for k = 1:9
   subplot(3,3,k)
   dog1 = reshape(dog(:,k),64,64);
   imshow(dog1)
end
```



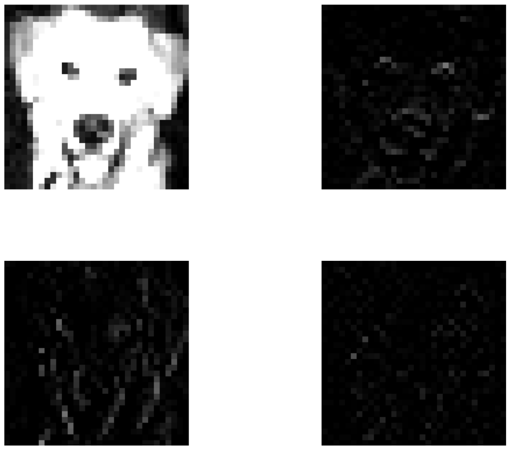
You should note that these are not very high resolution images. This will be an impressive aspect of this - we don't need perfect pictures to train on! Now, as mentioned above, we want to use wavelets to do some edge detection. We will start with a single-level discrete wavelet transform. Let's use Haar wavelets since we are most familiar with them. We will experiment with an image of one of the dogs to get our bearings. First, we have to convert the image to double precision.

```matlab
X = im2double(reshape(dog(:,6),64,64));
[cA, cH, cV, cD] = dwt2(X,'haar');
```

Here cA is the approximation (low frequency content), cH is the horizontal details, cV the vertical details, and cD the diagonal details. Note that each is only 32×32. Let's plot them to see what we have.

```matlab
figure(2)
subplot(2,2,1)
imshow(cA)
subplot(2,2,2)
imshow(cH)
subplot(2,2,3)
imshow(cV)
```

```
subplot(2,2,4)
imshow(cD)
```



The images are quite dark. The reason for this is that they aren't really scaled properly. Recall from Lecture 8 that if you give imshow something that is double precision, it expects a number between 0 and 1 (with 0 being black and 1 being white). If you look at the values inside our matrices, some are negative. So we need to rescale them to be between 0 and 1 to visualize the results of the discrete wavelet transform properly. But, we also need to keep the information about which values are large in absolute value (just like the fft), so we take an absolute value before rescaling. In order to do edge detection, we are also going to combine the horizontal details and the vertical details by adding them together.

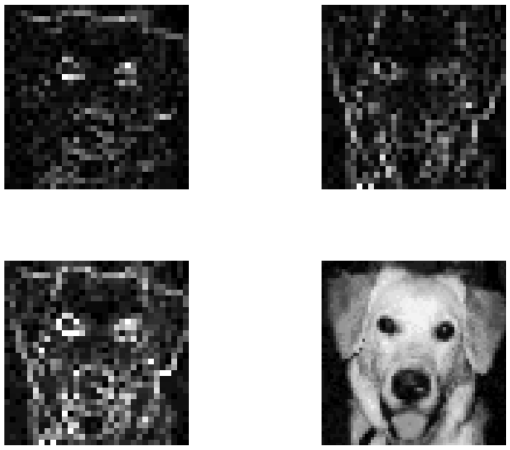
```
cod_cH1 = rescale(abs(cH));
cod_cV1 = rescale(abs(cV));
cod_edge = cod_cH1+cod_cV1;
figure(3)
subplot(2,2,1)
imshow(cod_cH1)
subplot(2,2,2)
imshow(cod_cV1)
subplot(2,2,3)
imshow(cod_edge)
subplot(2,2,4)
imshow(X)
```

In the bottom left corner (cod_edge) we can see a good tracing of the edges. This is what we will use for our classifier. Let's finish by building a function that will take in the matrix "cat" or "dog" and return the matrix of all the cod_edge vectors.

```
function dcData = dc_wavelet(dcfile)

    [m,n] = size(dcfile); % 4096 x 80
    pxl = sqrt(m);
    nw = m/4; % wavelet resolution
    dcData = zeros(nw,n);

    for k = 1:n
        X = im2double(reshape(dcfile(:,k),pxl,pxl));
        [¬,cH,cV,¬]=dwt2(X,'haar');
        cod_cH1 = rescale(abs(cH));
        cod_cV1 = rescale(abs(cV));
        cod_edge = cod_cH1+cod_cV1;
```



```
        dcData(:,k) = reshape(cod_edge,nw,1);
    end
end
```