COLE OTT | MILESTONE WRITEUP

# GENERALIZABLE IMAGE ANALOGIES

## *OVERVIEW & RECAP*

### FOCUS

The goal of my project is to use existing image analogies research in order to learn filters between images.

### SIMPLIFYING ASSUMPTIONS

Filters being learned do not significantly distort the image. Pixel p in image A maps directly to pixel p in image A'.

Filters being learned do not behave differently based upon their location within the image. They depend entirely on the content of the pixel being examined and the neighborhood of pixels within a given radius, not on, for example, their proximity to corners.

Different color channels behave similarly. So if my algorithm performs well on just the brightness channel of an image (i.e. the grayscale image), it has the capacity to perform equally well in the presence of color. Because of this, I will focus my research entirely on working with grayscale images in an attempt to reduce the dimensionality of my data without compromising anything.

### EXISTING WORK

I am deviating fairly strongly from the original image analogies work done by Hertzmann et al., as well as the many derivative research projects I have found. Their algorithm is essentially K Nearest Neighbors classification, in which each possible pixel value [0-255] is a separate class and K=1. They then heavily tweak this very simple algorithm with some powerful heuristics.

However, their algorithm is nonparametric and, with K=1, is prey to over-fitting. The combination of these two qualities means that their algorithm can perform well (in terms of both error and run-time) only with a decently small training set that happens to be well suited to the test example. By attempting to use regression to solve this problem, I believe I can overcome these drawbacks, but as a result I'm discarding some powerful simplifications that their model allows (e.g. when dealing with missing features).

This has created some interesting problems for me, which I'm confident I can overcome with some clever solutions, but have, for the moment, delayed me from implementing more advanced regression algorithms until I can solve them.

## *PROGRESS*

At this point, I have implemented a working version of the basic algorithm using linear regression and both linear and binomial features. While I am technically on-schedule, I would have liked to have a working solution to the problem of incomplete vectors so that I could spend the rest of the project trying other regression approaches.

The issue of adding coherence features requires me to deal with hugely incomplete feature vectors (the first row of pixels is missing between 5/6 and 2/3 their total features, for instance). It seems my progress in terms of implementing more regression algorithms is largely stalled until I can solve this problem, but I am confident with the progress I've made so far and I predict moving past this within around a week.

## *RESULTS*

### PICTURES

Using just linear regression, I was able to learn toy filters with a low error rate (as well as trivial filters such as the identity filter and an inverse filter, with error 0). I have included images of Gaussian blur and emboss because they are two typical filters that behave very differently. The included results use 2 images: Lena (200 x 200 pixels. Offers a variety of details, shapes, and solid/smooth regions), and Flower (100 x 129 pixels. Less variety and a generally darker tone).

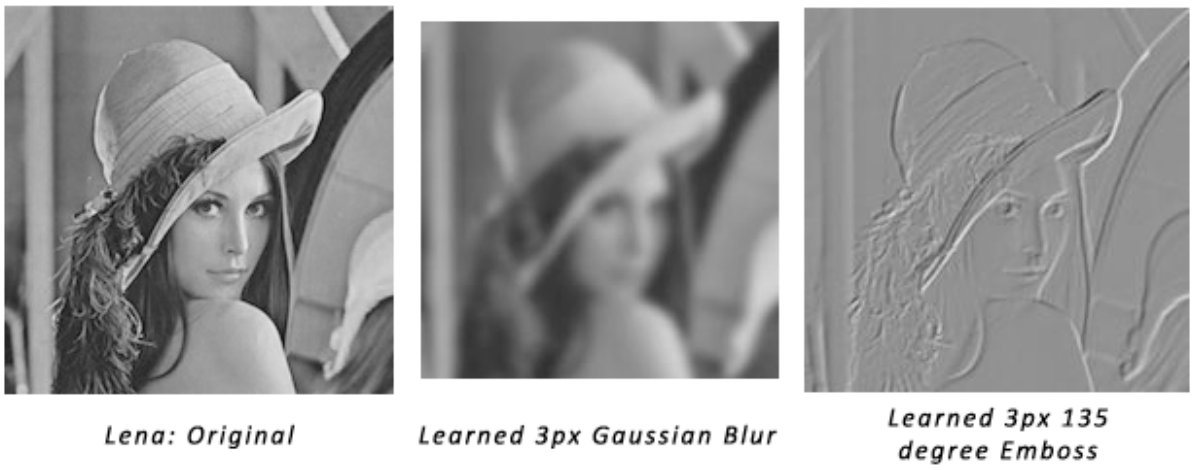

**Figure 1: Lena (Original, Blur, and Emboss)**
(training set: flower; sq. error rates 0.106 and 4.47 for Blur and Emboss, respectively)

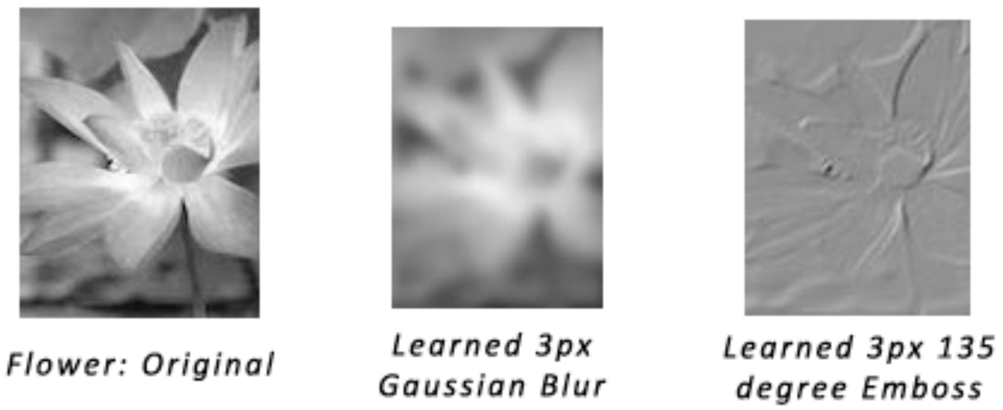

**Figure 2: Flower (Original, Blur, and Emboss)**
(training set: Lena; sq. error rates 0.101 and 2.87 for Blur and Emboss, respectively)

## EFFECTS OF FILTER SIZE

When being applied to the flower, my algorithm learned a 3 pixel Photoshop Gaussian blur effectively with a radius ≥5 pixels, and optimally with a radius of 8 pixels. Assuming Photoshop, which doesn't disclose the details of their algorithm, defines a 3 pixel Gaussian blur as having a variance of (.5 pixels + 3 pixels), as my searches have indicated, then my numbers seem ideal. The performance with filter sizes greater than 8 pixels increases very slightly, indicating a small amount of underfitting, which is unnoticeable to the human eye. For obvious reasons, overfitting is more prevalent when working from the smaller, less-varied training set (Flower).

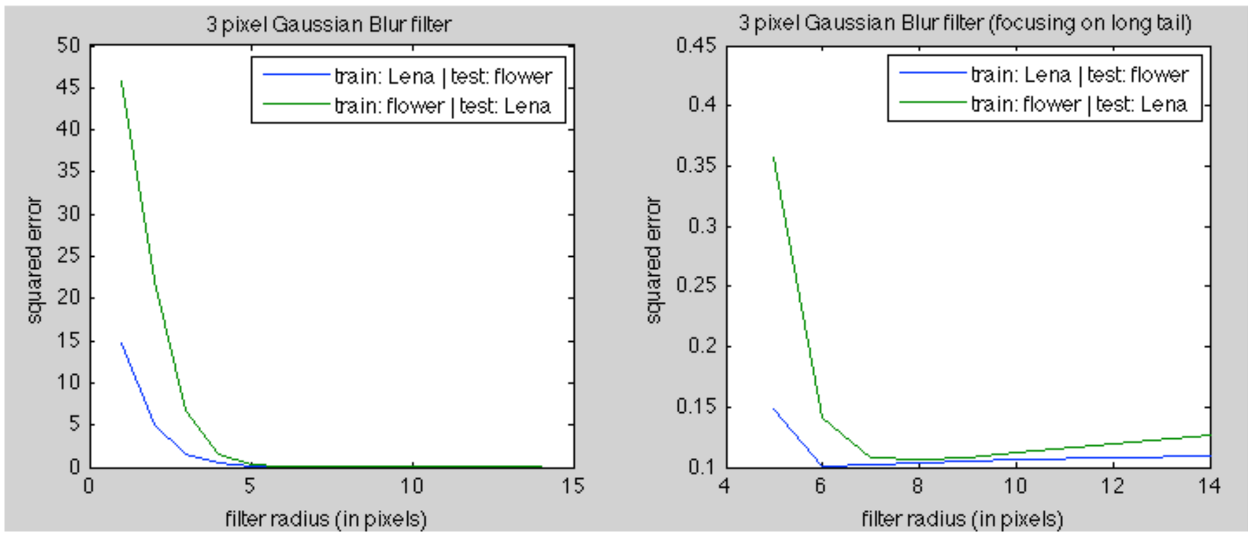

**Figure 3: Effects of Filter Radius on Gaussian Blur Examples**

Interestingly, the emboss filter performs differently depending on which image is used as the training data. When the image of Lena is used, it performs fairly predictably, although there is a much stronger effect of underfitting with larger filters than in the blur example. However, when the flower image is used as training data, the algorithm performs significantly worse at all radii, with a minimum error at, surprisingly, radius 1. Aside from the behavior of the latter example, the results behave fairly well — an emboss filter of 3 pixels at a 135 degree angle relies upon pixels in the regions at 135 degrees with a Euclidian distance of 3 away from the center pixel — which means a radius of 2 or 3 should be ideal.

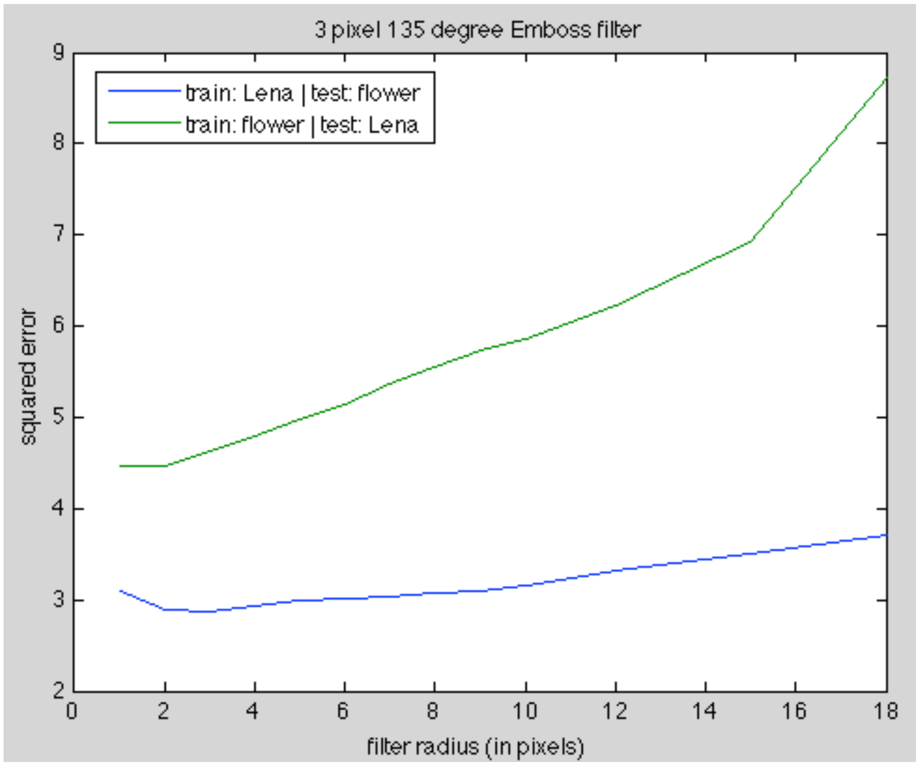

**Figure 4: Effects of Filter Radius on Emboss Examples**

### BINOMIAL FEATURE VECTORS

Using binomial feature vectors seemed to offer no improvements on learning the toy filters I focused on. However, I imagine that they will be helpful on more complex filters, which I will attempt after fixing my problems with coherence features. However, because of the high dimensionality of my data, binomial feature vectors are very costly with even reasonably-sized radii—a problem to which I outline one possible solution in the following section.

## *PROBLEMS*

### MISSING FEATURES

Dealing with huge amounts of missing features. This occurs along all edges and for the first "strip" (as wide as the filter radius) of the image.

My current solution is to tag invalid pixels by setting their value to negative one. I then effectively ignore invalid pixels at application-time by setting their corresponding theta values to zero, then normalizing the product of that pixel and theta (i.e. the output pixel) by a coefficient that represents the ratio of the relevant theta to the "whole" theta. In implementation, this looks like:

```
for each pixel p in A
     rTheta = theta .* (A(:,p) > 0); % where A(:,p) is p's vector
     c = abs(sum(theta) / sum(rTheta));
     A2(p) = c * rTheta * A(:,p);
end
```

As is shown in the images below, this has worked fairly well in some scenarios, allowing me to generate the entire image without having to ignore the edge. It generates just a small error around the image edges in the successful Gaussian blur example, but produces impossible values in the failed Emboss example (hence the odd rendering at the top of the image). I'm currently reevaluating my math in order to avoid these impossible values, but I haven't fixed the problem entirely yet. It is also possible that out-of-bounds output values are occurring partially because of a difference in histograms, which I am also looking into dealing with as well.

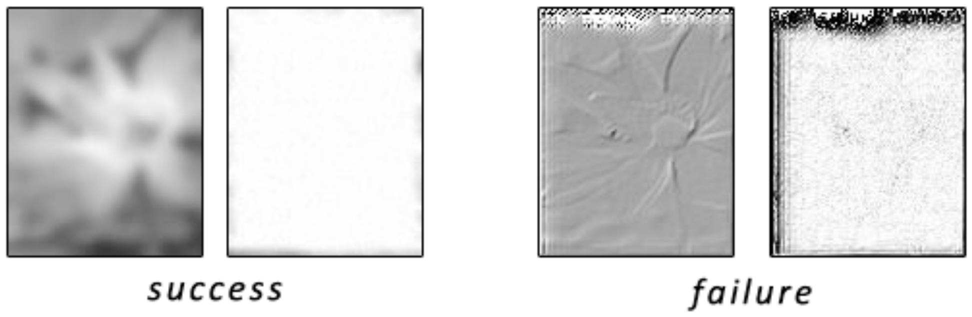

**Figure 5: Successful and Failed Applications of Coherence Features**
Both of these images were generated identically to the previously covered images, except that these also included coherence features of radius equal to the normal feature radius. Difference maps accompany each result to demonstrate where errors occur.

## HIGH DIMENSIONALITY

One problem that has presented itself is the explosion of dimensionality that accompanies even reasonably-sized filters—especially when it seems preferable to use binomial features. Even with a filter of only 11 pixels in width, there are 122 linear features and over 7000 binomial features.

Operating on the assumption that closer pixels are more relevant to the filter than further pixels (from the chosen pixels), I have devised (but not implemented) a method to reduce the dimensionality without sacrificing a great amount of effectiveness. My idea is to randomly select which pixels to treat as features with probabilities dependent on a gaussian distribution. This arose from two pieces of intuition: pixels further from the source pixel are less relevant to the results, and coherence among pixels allows us to simply pick pixels randomly without sacrificing much accuracy. An alternative approach to picking randomly would be to average certain pixel values out, which would effectively view pixels further from the target pixel in lower resolution. There are cases in which these intuitions fall flat, such as in a very noisy filter, but I believe they should hold on most filters I examine.

## *WHERE TO GO FROM HERE*

| | |
|---|---|
| **[ 1 week ]** | Solve the problem of impartial vectors (at least to the point of workability). |
| | Normalize histograms. (Hertzmann's research has some insight in this area) |
| **[ beyond ]** | Implement Support Vector Regression. |
| | Experiment with random feature selection along a Gaussian distribution. |
| | Ideally implement at least one other regression algorithm. |
| | Compare results to Herzmann's. |