CLASSIFYING DIGITAL PHOTOGRAPHS TO IMPROVE AUTOMATED IMAGE ENHANCEMENT

ECS 192 Final Report

Patrick Sheehan

September 23, 2013

1 Background

The inherent numerical properties of the data collected by digital camera sensors allow software developers to fuse art with science. A large portion of this fusion lies in the digital darkroom where the pixels that compose photographs are altered mathematically to produce more pleasing images for the human eye. As the field of computational photography has grown, so has the desire to automate the adjustment process. In extreme cases like Instagram™, two buttons separate a user from observing a scene to posting an edited (albeit suboptimal) image online.

2 Objective

The main objective of my internship at Apple Inc. was to improve Apple's auto-enhance algorithm for digital images through use of binary classification.

3 What is Binary Classification?

Binary classification is the task of taking a set of objects and splitting them into two groups based on a set of properties.[1] Better put,

> A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one "target value" (i.e. the class labels) and several "attributes" (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.[2]

For my project, binary classification involved taking a set of digital photographs and computing various numeric features that would be used to train a support vector machine[3] (SVM) that would then separate an arbitrary image into one of two categories (any further information about the categories would break my contract / NDAs). The result of this classification would then be used to fine-tune auto-enhancement for that image.

A non-confidential example of binary classification with respect to automated digital image enhancement would be determining whether or not a photograph is of a sunset. Separating images

[1] "Binary Classification", http://en.wikipedia.org/wiki/Binary_classification

[2] "A Practical Guide to Support Vector Classification", http://www.csie.ntu.edu.tw/cjlin/papers/guide/guide.pdf

[3] support vector machines maximize the margin between the decision hyperplane and the data in the training set; more info at http://www.cs.ucf.edu/courses/cap6412/fall2009/papers/Berwick2003.pdf

into the categories "is a sunset" and "is not a sunset" might help automated image enhancement because one might, say, adjust the color temperature or white balance of a sunset image differently from a non-sunset image. To perform this classification, features such as image brightness might be used which perceptually differentiate between sunset pictures and non-sunset pictures.

4 Infrastructure for Data Collection and Classification

Using an internal photo-editing application and imaging library as frameworks, I created two tools; one to help me collect data and another to help with classification.

The data collection tool allows a user to load an image and set sliders to generate minimum, ideal, and maximum adjustment values which, upon close, are stored in a comma-separated values (CSV) file along with the image's filename.

The classification tool takes in two folders of manually separated images and adapts existing batch-processing code to calculate features for each image, processing the folders in parallel and outputting each image's label[4], feature vector, and filename into a single CSV file:

```
vp0102wa-dhcp57:Desktop patrick$ cat SVM_Training.csv
Label     ,Feature1  ,Feature2  ,Feature3  ,Feature4  ,Feature5  ,Feature6  # Filename
-1        ,0.992722  ,0.24426   ,0.165672  ,0.261739  ,0.07419   ,0.484369  # 0015STC_1602__aorg.jpg
1         ,0.00830725,0.107465  ,0.246383  ,0.0667503 ,-0.000130104,0.136704  # 001IMG_5362__aorg.jpg
1         ,0.0306647 ,0.208057  ,0.267188  ,0.386874  ,-7.53066e-05,0.683586  # 007Edinburgh 059__aorg.jpg
-1        ,1         ,0.898921  ,0.231872  ,0.0272536 ,0.917953  ,0.995515  # 100223_DamisWinter_AT_PHO-09-0102_0020.jpg
1         ,0.23798   ,0.29055   ,0.352376  ,0.683857  ,0.00354937,0.734357  # 100223_DamisWinter_AT_PHO-09-0102_0629.jpg
-1        ,0.999374  ,0.326257  ,0.235582  ,0.460981  ,0.0155318 ,0.734371  # 21 Beach Sunset in Santa Cruz  07.NEF
1         ,0.0066089 ,0.0537216 ,0.12248   ,0.0396121 ,0.00295415,0.0898427 # 2637928109_816e3f9bef_o.jpg
-1        ,0.999949  ,0.34076   ,0.22694   ,0.234407  ,0.148415  ,0.574217  # _MG_0793.CR2
-1        ,0.999981  ,0.313398  ,0.118129  ,0.292975  ,0.128904  ,0.574219  # _MG_0838.CR2
1         ,0.00757447,0.0560384 ,0.127196  ,0.0315088 ,0.00325501,0.0898298 # 31 DSC_6825.NEF
-1        ,0.955171  ,0.310732  ,0.274678  ,0.582037  ,0.0156126 ,0.667966  # _MG_0866.CR2
-1        ,0.808902  ,0.257077  ,0.251082  ,0.347673  ,0.0195101 ,0.539062  # AB18B133-02C8-4AA2-81B8-E70F97EA3797@apple
-1        ,0.999947  ,0.387222  ,0.232851  ,0.460938  ,0.0585817 ,0.699213  # Canon 1D Mk II N.CR2
```

After generating features, a second button in the classification tool begins a process which uses data in the CSV and an existing interface for LIBLINEAR[5] to train a SVM. Finally, a cross-validation[6] method is called which produces a confusion matrix and decision code corresponding to the training data:

```
True positives = 103
False positives = 21
True negatives = 103
False negatives = 10
Accuracy = 0.869198
Precision (+) = 0.830645
Precision (-) = 0.911504
Number of images = 237

static double decision(const double *features, size_t numFeatures, int *label){
    assert(numFeatures == 1);

    static const double normal[1] = { 2.43227605, };

    double result = -1.2954729;
    for(int f = 0; f < numFeatures; ++f){
        result += normal[f] * features[f];
    }
    int theLabel = (result < 0) ? 1 : -1 ;
    if(label){
        *label = theLabel;
    }
    return fabs(result) * theLabel;
}
(lldb)
```

[4] an image's label would equal "-1" if it belonged to one category and "1" if it belonged to the other

[5] "A Library for Large Linear Classification", http://www.csie.ntu.edu.tw/cjlin/liblinear/

[6] I chose to use K-fold cross-validation where I set K equal to the size of the training set. This means that, for each image, the rest of the images in the set are used to predict its label, more at http://en.wikipedia.org/wiki/Cross-validation_(statistics)

5 Collecting Useful Training Images

A large portion of my work this summer was focused on studying images through the perspective of both a photographer and an average customer to create a set of images that could clearly be separated between the two, purposefully unnamed categories.

For every 100 images I inspected, ≈ 2 out of 5 were usable as training images. Thus I individually inspected 600 images to arrive at my final training set of 244.

Putting in this work was necessary to both accurately grasp the impact of using various features in the classification and also to make clear the implications of any modifications to the enhancement algorithm. For any change to the algorithm to be considered an improvement, it must improve a substantive portion of the general case while preventing an increase in the number of failure scenarios.

6 Creating and Selecting Features

After experimenting with basic histograms and statistical measures such as mean pixel luminosity[7], I refocused my attention on higher-level image qualities. Using my background in photography, this lead to two useful metrics: one pertaining to detail and another to tonal range.

6.1 The Detail Metric

The detail metric began with a Sobel edge filter and ended with a modified version that more closely matched the level of perceptual detail in an image. Here's the result:

`Original Image` `Sobel` `Modified Sobel`

Using a gamma adjustment[8], Gaussian blur, and luminosity blending with a low-alpha[9] white image (and various other minor adjustments), I was able to alter the standard Sobel filter to better

[7] luminosity describes a weighted average of the red, green, and blue (RGB) color channels which gives green and red the highest weights in an effort to measure percieved brightness by the human eye

[8] gamma adjustments involve raising each pixel in an image to a power (in this case 0.4)

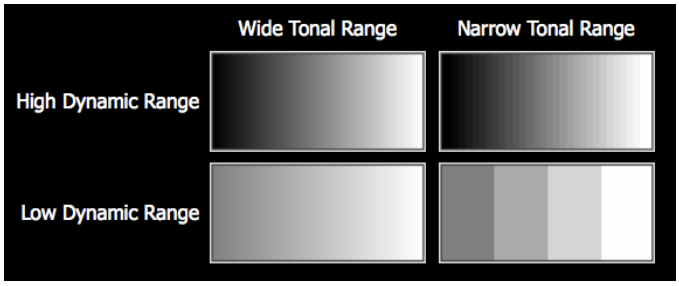
[9] the alpha of an image corresponds to translucency with 0 being transparent and 1 opaque

reflect all of the details that the human eye can detect in an image.

To translate the output image into a single metric, I summed pixel intensity[10] in the modified Sobel, weighted by a quadratic involving the pixel intensity in the original image at the same location. This weighted sum was then normalized by the size of the image and spread between $[0.0, 1.0]$ by my supervisor's favorite trigonometric function: a hyperbolic tangent.

6.2 The Tonal Range Metric

Tonal range is an especially important term in the field of photography. It describes the distribution of tones throughout an image and is limited by the dynamic range of the scene being photographed (and properly exposing to that range). In digital photography, the maximum amount of tonal range is ultimately limited by the dynamic range of camera sensors. But as with most concepts in photography, showing is better than telling:[11]



After discussing the concept of measuring tonal range with coworkers, using the image key (geometric mean of all pixel values) was suggested. To test this suggestion, I added a function to my classification tool which calculated the geometric mean of pixel luminosity for an image, offset by a small epsilon[12] to prevent black pixels from dominating the product. This data was again spread using a hyperbolic tangent.

6.3 Results

6.3.1 Why the Detail Metric Failed

While I was most excited about my work generating the detail metric, I soon realized that it failed to account for the global nature of auto-enhancement adjustments. Details in any photograph are generally important, but are inherently localized features. For auto-enhance to be successful, it has to prioritize global features like overall image brightness, contrast, and tonal range. So, unfortunately, I had to discard my work on the detail metric and focus my attention to features that might correlate with or measure these global properties.

6.3.2 Using the Tonal Range Metric to Improve Auto-enhance

Extensive testing and tweaking revealed that the tonal range metric alone was able to accurately separate 87% of the training images; an atypical result for linear classification. Going further, I compared tonal range values to other information in the current auto-enhance algorithm which revealed useful trends. Using this information (curve adjustment, contrast adjustment, etc.) as binary

[10] the counterpart to luminosity, a direct average of the three color channels
[11] http://www.dpreview.com/glossary/digital-imaging/tonal-range
[12] one "color bit" which is 1/255

features for the classification, many of the false negatives were remedied, improving overall accuracy to 99% which made the metric usable as a stand-alone indicator.

While this result deviates from standard instances of linear classification (which can involve hundreds of features), my end goal was met. Using the tonal range metric in conjunction with other auto-enhancement information produced better images in a substantive amount of cases while not generating any additional failure situations.

6.3.3 Using the Tonal Range Metric to Avoid Over-enhancement

A persistent problem for most (if not all) auto-enhancement algorithms is over-enhancement. Typically, if you enhance an image, save it, then enhance again you get a "double-enhanced" image. Repeating this process quickly leads to a failure scenario.

For some cases, enhancing an image once or twice would push the tonal range metric over the threshold. This result effectively limits over-enhancement in cases where the image's original tonal range value was not extremely low ($> .1$). This additional benefit was explored less extensively, but shows promise for at least slowing the rate of over-enhancement.

7 Additional Work

While most of my time was spent researching features, coding tools, looking for usable training images, and testing, I also spent a significant amount of time using my data collection tool (three adjustment sliders that could be set to 'min', 'max' and 'ideal' adjustment values) to provide a mapping between the current adjustment algorithm value and the new adjustment algorithm value.

To do this, the sliders would be set to reasonable values for one algorithm, then a checkbox would switch to the new algorithm and allow the sliders to be set again. Both slider sets would retain their values such that clicking the checkbox would simply switch the values of the sliders. Here's what the sliders look like (checkbox not included in the screenshot):

As you can see above, ideal is set first (discussions with coworkers revealed that setting the ideal was most natural) then both the min and max are adjusted relative to the ideal. The min slider starts from the right, subtracting from the ideal, the max from the left, adding to the ideal. At any point, an additional checkbox allows the user to make the sliders independent, setting each slider to its non-relative value. The buttons next to the sliders allow the user to quickly compare the min, ideal, and max image. Clicking one of the buttons re-adjusts the image based on the respective slider value. To make things more intuitive, clicking or adjusting the slider will also switch both the active button and the adjustment on the image to the currently clicked/moving slider.

8 Learning Experience

While I may have wished that I had experience with Objective C, XCode, and Core Image prior to my internship with Apple, I did not. This posed a great challenge that went hand-in-hand with a great learning experience.

Objective C is Apple's primary programming language. My experience with C and C++ helped me adapt quickly to the language itself; what became difficult was integrating what I needed to do with the existing structure of Apple's internal software.

Contributing to a large project with source control was perhaps the best experience I gained at Apple. Working with a group of people on the same application with many files, components, and possible source control issues was a great look into what coding in the "real world" is like. I used tools and plugged in code that others in my team had made and learned how to make my code usable in the same way. I also gained an understanding of the importance of code readability; both through negative and positive experiences trying to understand the code of others. To the best of my ability I used this as an opportunity to write readable, succinct, and commented code that others could understand fairly quickly (which is a statement I hope my supervisor will agree with).

With my introduction to Objective C came an introduction to XCode 5, Apple's IDE. This too was a positive experience, both because I want to use XCode for my CS projects at Davis, and because this gave me further insight into the real software development world; I had never "built" a program before and now I understand the settings, procedures, and power behind IDE-based software development (also how easy it is to ruin everything).

Core Image (CI) was my greatest challenge. While I had taken ECS 173 (image processing) prior to my internship, my experience was with ITK, a medical image processing library based in C++. Core Image, on the other hand, is "an image processing and analysis technology designed to provide near real-time processing for still and video images"[13]. While Core Image was more intuitive than ITK with its Objective C interface, it was entirely different from the image processing system that I was accustomed to. Using both internal tools and studying the CI library I was able to get a basic understanding of Apple's method for image processing and utilize it to calculate features for my classification project. I also helped integrate an adjustment kernel into the internal program which gave me an "under the hood" look at how CI is used to manipulate images.

Overall I find it difficult to concisely sum up what I learned in my three months at Apple. I've inevitably left out many of my day-to-day experiences and meetings that showed me the inner workings of one of the largest and most successful tech companies in the world (not to mention what it's like to be a Computer Scientist outside of school).

To me, my internship represented a productive merger between my love for Computer Science and my passion for Photography. I am proud of what I did and I am grateful to have been given the opportunity to do it.

[13] https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/CoreImaging/ci_intro/ci_intro.html