Introduction

Estimating the subjective, human-perceived quality of an image can be very useful. For example, with an accurate measure of how interesting and attractive each photo from an event is, we could create high-quality, automated slideshows and summaries. Often people take multiple pictures of the same scene, and these quality metrics could also be used for deduplication: once duplicate images have been identified, we can choose to keep only the best one.

In general, this is a hard problem, because it’s not immediately obvious what makes a particular picture good or bad. While humans can make snap judgements about photos in a couple seconds, they use countless factors to do so, and draw upon years of experience and training.

There has been a reasonable amount of research done on this topic before. [2] is an example of an early series of experiments, while [1] was in the news only very recently. Modern methods have become very complex, for example [4] uses a complex layering of many feature extractors and multiple classifiers to categorize photos. Such complex methods are clearly outside the scope of a final project. Rather than attempting to match the power of these approaches, in this project I’ll try to answer a simpler question: how well can an algorithm perform this task while minimizing complexity and maximizing speed. While smartphones today are faster than ever, they are still far slower than computers, and an algorithm suitable for a mobile photo organization app will need to be able to process and categorize images fairly quickly to be practical.

The two-stage algorithm I developed uses a variety of image processing techniques to measure features of an image, then passes these features into a random forest classifier to make predictions about the quality of the image. The classifier was trained on images from redit.com, and the resulting algorithm is nearly as good as a human at predicting the quality of test images.

Method

Due to the complexity of the problem, I decided to use machine learning techniques to construct a system that can learn the qualities of a good image itself. As a result, the algorithm can be broken into two stages. Both stages are fairly simple, and could also be easily parallelized by splitting different features and decision trees across cores.

Stage 1: Image Analysis

The average image is far too large to pass into a classifier directly. Additionally, images can be similar in many high-level ways that can’t be recognized by looking only at the pixels, at least without massively large training datasets. Since making a large, high-quality dataset is difficult and for all these other reasons, the images are preprocessed before the classifier is run on the result.

The goal of the preprocessing is to extract measurements from the images that are salient to image quality either alone or in combination. Individual features don’t have to have direct correlation to quality to be useful. For example, perhaps image sharpness is more important for photos of people than for landscapes. Then while a feature that can distinguish people from landscapes may not be directly useful, it could improve the performance of the algorithm by increasing the usefulness of a sharpness feature. Deducing these relationships is left to the classifier. In all, I developed 36 features, which can be separated into six broad categories. Individual features will be explained in more detail in the implementation section.

1. Technical Quality includes features to measure things such as sharpness, contrast, and noise. Broadly, these are intended to measure the objective quality of the image, and identify mistakes such as photos that are out of focus, or underexposed.

2. Blur includes features to measure blur in an image. This is distinct from a measure of sharpness in two important ways: First, images with shallow depth of field, that isolate the subject and blur the background, are often considered high quality, and a heuristic feature to help distinguish “artistic” images from images that are just blurry is useful. Second, a measure of local contrast or high frequency detail in an image will be fooled
by a photo that is blurred along only one axis, due for example to camera shake. The algorithm uses an additional feature to detect blur along different axes to address this.

3. **Color** includes features to analyse the hue and saturation of an image. The effect of bright color on image quality is intuitive and borne out by the results (below). This section also includes a measure of the level of complementary color in an image.

4. **Composition** includes several features that attempt to extract structural information about the shapes in an image, including simple measures of image symmetry, and image alignment (the concentration and orientation of edges and lines in the image).

5. **Face detection** is self explanatory. People may prefer photos of other people to photos without people in them.

6. **Spatial envelope** is a collection of several features that attempt to classify scene types, based on [5]. These include “naturalness”, which measures the distribution of edge orientations: predominantly horizontal and vertical is less natural than an even mix, and “roughness”, which measures the overall complexity of the image, among others.

**Stage 2: Machine Learning**

The result of analyzing an image is a vector with 36 values. Most of these are continuous, with a few exceptions (for example, the number of faces is integral). This vector is fed through a random forest classifier, which predicts whether the image is “good” or “bad”.

Random forest classifiers, which use the consensus of many decision trees and were first described in [7], have a few significant advantages for this application over other types of supervised learning algorithms. First, decision trees are very flexible, and perform well on the mixed datatypes they receive here. Second, as an ensemble method, random forest on average does a good job of identifying complex, nonlinear relationships within the data. This also makes the classifier easily parallelizable, but in practice it’s much faster than image analysis so this isn’t important. Finally, random forest makes it easy to estimate feature importance, to analyze the contribution of each feature to prediction accuracy. For comparison, SVM (even nonlinear SVM with the kernel trick) performed several times worse on the same training data.

The classifier is trained on a dataset of 1800 images collected from reddit.com/r/itookapicture. This is far from an ideal dataset, but was far less work to collect than creating a suitable one from scratch would be. I did not find a suitable preexisting dataset available. Taking pictures from Reddit does have a few advantages: it results in photos of a wide variety of subjects and styles, and thanks to Reddit’s up/down voting system each image conveniently is already scored by Reddit users. These scores are also highly subjective, which makes classification difficult but is more representative of the goals of the algorithm. The itookapicture subreddit in particular was chosen for its focus on photographs (rather than memes and other images), its generality, and due to its reasonable size.

However, using Reddit as a datasource also has some unavoidable disadvantages. First, poor quality photos are inherently underrepresented: in general people only share their better photos, and as a result the photos that are uploaded are on average good. This makes classification significantly harder, because the difference between a good and a great image is in absolute terms smaller and more subjective than the difference between a good and a poor image. Second, the voting data is very noisy. For example, some good images just get unlucky and are never upvoted, while some mediocre images are upvoted more for the story behind them than for their quality. I avoided going through the dataset myself and manually removing images I thought were mislabeled because I didn’t want to introduce my own bias, but when I tried this did dramatically increase performance. Ultimately I only removed images that were clearly not photos, and left all others.

**Figure 1: An example of 10 images from the dataset.**

I decided to stick to a binary classification task for simplicity. Due to the noisiness of data in this dataset and consequent inherent limits on potential accuracy, regression or classification with more than two categories ended up being significantly more complicated to reason about and negligibly more informative. I selected images such that the log of the number of upvotes...
was approximately uniformly distributed, and I chose 50 upvotes as the threshold for a "good" image, which roughly split the dataset in half: 52% "bad" photos and 48% good photos. Hence, in practice my algorithm is predicting whether or not an image is likely to receive 50 upvotes, rather than quality directly. The algorithm can be trivially adapted to learn and predict continuous quality scores if desired.

Implementation

All code was written in Python, with the help of a number of libraries, including PIL (the Python Imaging Library), Numpy/Scipy, OpenCV, and scikit-learn.

Stage 1: Image Analysis

The image is first downscaled to a maximum of 800 pixels in either dimension, while preserving aspect ratio. This is done primarily for efficiency, but some measures like sharpness are also sensitive to image scale and benefit from uniform image size. I assume that features too small to detect on the downscaled image are unimportant, and experiments adjusting the scaling show little classification improvement at larger sizes. 800 pixels wide is also roughly the size that an image would appear on a computer screen.

Then, each feature extractor is run on the image in turn. All but the color features are computed from a grayscale version of the image. In the implementation of features, speed and efficiency were emphasized. Many of these features could be much more accurately measured with subclassifiers of their own, as used in for example [4], but that is slower and in any case beyond the possible scope of this project.

• Technical Quality

  – **Sharpness** is extracted by applying a high pass filter to the image (subtracting a blurred version from the original), then computing the 98th-percentile pixel in the resulting image. The 98th-percentile was chosen empirically as a good compromise between reliability and the goal of measuring peak image sharpness. An image is blurry if the entire image is blurry, but an image is sharp even if it contains a small amount of high frequency data - the rest of the image may naturally be smooth gradients.

  – **Contrast** is measured as the range and standard deviations of the image’s brightness histogram.

  – **Noise** is estimated as the difference between the image and a median filtered version of the image. This is obviously crude, but is trivial to compute and evidently useful.

• Blur

  – **Depth of field** is estimated by breaking the image into 25 segments (5 per side), and comparing the sharpnesses of the segments. A large difference between maximum and minimum sharpness presumably indicates a shallow depth of field. This is complicated by several common false positive, such as solid blue skies and high-key lighting. I found no pattern in the locations of the sharpest regions, so to combat this the algorithm uses the ratio between the sharpest and median regions in the image.

  – **Motion blur** is estimated by convolving the image with a number of different one-dimensional gaussian kernels at different orientations, and comparing the resulting image sharpness. An image blurred in only one direction will be sharper when convolved with a kernel in that direction than with a kernel in a perpendicular direction.

• Color

  – **Saturation** is simply measured as the mean of the saturation channel after the image is converted to HSV space.

  – **Hue** computes a rough histogram of hue values in the image. This leads to 6-12 features measuring the amount of blue, green, yellow, orange, etc. in the image.

  – **Complementary colors** are measured by taking the dot product of the hue histogram and the hues rotated 180 degrees. As a result, the blue bin is multiplied by the orange bin, red by green, etc. so a higher score corresponds to a higher fraction of the image composed of complementary color pairs. Partial credit is given for near misses, for example red and blue.

• Composition

  – **Symmetry** is estimated by measuring the differences between the image and reflected copies of it. The image is reflected horizontally and vertically around the centerline,
and around the four one-third lines (two of each orientation). Far more complex, accurate, and expensive methods are described in [6], but in practice this method is simple, fast, and still meaningful.

- **Lines** are computed by running a Hough transform on the image binarized by the Canny edge detector. The mean and spread of line orientations is reported.

- **Face detection**
  - Face detection is implemented using the Viola-Jones face detector built into OpenCV.

- **Spatial Envelope**
  - These features are loosely based on the ones described in [5], but significantly simplified.
  - **Naturalness** computes the gradient of the image, then investigates the distribution of edge orientations. Predominantly horizontal and vertical edges score low, while a more even distribution scores higher.
  - **Roughness** Measures the density of edges in the image. The Canny edge detector is run on the image, and returned result is proportional to the number and length of edges detected.
  - **Complexity** is similar to roughness, but measures the amount of high frequency detail in a manner closer to how sharpness is measured.
  - **Expansion** counts long diagonal edges, which presumably correlate with images with strong perspective and vanishing points.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forest</td>
<td>62%</td>
</tr>
<tr>
<td>Always guess “bad”</td>
<td>52%</td>
</tr>
<tr>
<td>Human</td>
<td>67%</td>
</tr>
</tbody>
</table>

Stage 2: Machine Learning

For the random forest classifier I used scikit-learn, a popular Python machine learning library. The parameters were tuned by hand.

Results

I tested the algorithm by performing randomized 10-fold cross validation of my 1800 image dataset (that is, tested on a randomly selected 10% of the dataset, trained on 90%, and repeated this 10 times, then averaged the results). I measured classification accuracy, the percent of images correctly classified. The results can be seen below, with two other reference points:

The algorithm performed 10% better than random, compared to my 15% better when I attempted to rate the images myself. Hence, by this measure the algorithm is about $\frac{4}{3}$ as good as I am at predicting the quality of the images in the dataset. Note that my 67% was the best after several tries (each on a different subset of the images), and trying very hard not to bias my guesses: I ended up guessing “good” 49% of the time, which compares well with the expected 48% proportion, so I believe this is a fair estimate of human performance.

These human and algorithm results are both disappointingly low. I believe this is almost entirely a result of the dataset, which is particularly difficult for the several reasons described in the Method section.

For a more subjective test, I also ran the algorithm on twelve of my own images from a trip to an air museum. The results are shown in Figures 2 and 3. We can see that the algorithm does a good job of identifying the obviously boring images as bad, and the images identified as good are certainly more interesting on average.

Figure 2: The six test images that the algorithm classified as “bad”.

Figure 3: The six test images that the algorithm classified as “good”.

4
Finally, I also empirically analyzed the contribution of each class of features to the overall classifier performance. This is based on the frequency of the features’ selection in the decision trees, and is calculated automatically by scikit-learn. The results can be seen in Figure 4. The most important features by far are the color features, followed by technical quality, composition, and spatial envelope features in that order. Among individual features, the most important are the hue features, then symmetry, complexity, depth of field, and contrast.

Surprisingly and conspicuously, face detection was almost useless on this dataset. There are a couple likely explanations for this. First, faces are not very common in the dataset, and likely far less common than they would be in a real sample of someone’s photos. Second, I did not spend much time tuning the OpenCV face detector, and in the limited time I had to experiment I could not keep the detection rate reasonable while totally eliminating false positives. Even with a false positive rate of 5%, if faces are only really present in 5% of the dataset then about half of detected faces will be incorrect. In actuality faces were detected in about 15% of images in the dataset.

Future Work

There are many ways that this work could be expanded. First, many more features could probably be added to the image analysis stage of the algorithm, and the accuracy of many of the existing features could definitely be improved, in particular face detections and noise.

Second, this project is almost certainly being held back by its dataset. A larger training dataset, with more low-quality images and more careful curation to remove outliers, would almost certainly instantly improve the performance of the algorithm. The feature extractors and the dataset are both extremely important to the quality of the algorithm’s results.

References


