Which Way Is Up?

Abstract

Determining the correct orientation of a photograph has often required human inspection of its various distinguishing elements like the location of the sky or the presence of faces or buildings. In this project, we use machine learning to automatically determine the correct orientation of a photo, helping to eliminate the need for humans to manually rotate digital photos. Specifically, we train a multilayer perceptron via backpropagation to take as input various features of a particular photo, such as the average color of particular regions of the photo, and produce an output indicating which edge of that photo is up. With such information, photos can be automatically and correctly rotated as soon as they are scanned or loaded onto the user's computer. Our variations of the features we used include adjusting the size and number of sample regions we draw from a photo, the type of color model we use, and which regions of the photo we sample. Using a training set of roughly 1000 photos and a test set of 500 photos, we achieve a final accuracy of 76%.

1 Introduction

Taking photographs has become easier over the past few decades. While photos are usually taken in a landscape orientation, sometimes a portrait orientation is used. Photos are transmitted to digital format in a variety of ways; physical photos are digitized using a scanning bed, and digital photos are imported from a camera or phone. When the photos are scanned or imported to a computer, it is often the case that the individual must go through and manually determine if and how to rotate each of their photos.

A machine learning algorithm could learn to correctly orient the photos so that users do not have to manually rotate the photos. This would save time and make importing pictures faster and more convenient.

2 Methods

2.1 Data Source

All three of the researchers contributed roughly 500 family photos to be used in the project. The pictures included a mixture of outdoor/indoor shots and shots with and without people. Small groups of photos looked similar because they were taken at a certain event. Out of the 1,616 photos used, about 78% of the photos had a horizontal orientation.

The images were all rotated to landscape orientation. Roughly 50% of the images were then rotated by 180 degrees. The images were then given the label of their correct orientation. The four label classes that our models will try to predict are ‘up’ (the image is already rotated correctly), ‘down’ (the image needs to be rotated 180 degrees), ‘left’ (the image needs to be rotated clockwise 90 degrees) and ‘right’ (the image needs to be rotated counter clockwise 90 degrees).

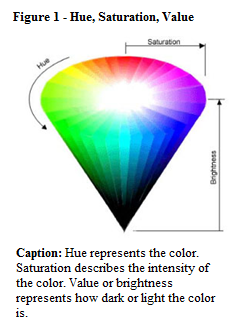
In the following experiments, photos contributed by Hansen and Kendall were used for the training set, while photos from Christensen were used as the test set. 1088 pictures were in the training set while the remaining 478 photos were in the test set. This was done to prevent data from leaking from the training into the test set. For instance, a series of family photos are often taken in the same position and location. If some of the similar photos appear in the training set while others appear in the test set, then it will be much easier for the model to classify the photos. Hence, the training and test sets were divided based on contributor.

The following are examples of photos used:



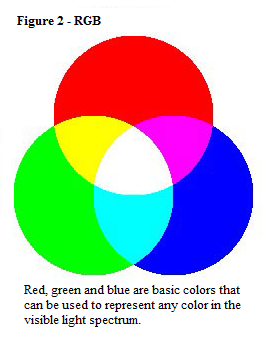
2.2 Attributes

The photos were divided into equally sized rectangles. The divisions used in our experiments include 4x4, 8x6, 16x12, and 16x24. This method divides the photo into blocks that we can inspect individually and use to draw conclusions.

For each of the rectangles, three main values were calculated: average hue, average saturation, and average value. Hue measures the color and ranges from 0 (red) to 120 (green) to 240 (blue) to 360 (red). Average hue is a useful feature because the model might use the hue to determine that blue sky is typically at the top of photos while green is usually at the bottom. Saturation refers to the intensity or dominance of the hue. The values range from 0 representing no saturation to 1 representing full saturation. Using average saturation as an input allows the model to learn patterns relating to hue intensity. Value measures the lightness of a color. Values range from 0 representing white

to 1 representing black. Average value enables the machine learning model to detect brightness in certain areas of the photo.

As an alternative to hue, we also tried two other representations of color: RGB (red, green, blue) and RGB percentages. The RGB color model contains three values ranging from 0 to 255 representing the amount of red, green, and blue in the color. RGB percentage is a custom color model we created that we felt would be best for the backprop model. It included four values. To calculate the percentage of red, green, and blue, the standard RGB values were totaled. Each indvidual color was calculated as a



percentage of that total. The resulting values ranged from 0 to 1. The fourth value was the total divided by 765 (255 \* 3), which represents the total light in the picture. We believed this would provide a more understandable form of color for a neural network.

2.3 Selected Models

A multilayer perceptron trained with backpropagation enabled the model to discover complex patterns for real-valued inputs. An instance-based model was also used to learn since photos with similar colors in certain areas likely have the same orientation. A single layer perceptron was incapable of dealing with the complexity of this problem, while a decision tree model was not selected because such a model is not ideal for continuous attributes.

3 Initial Results

For our initial model, a multilayer perceptron trained with backpropagation was used. The model had one hidden layer with twenty nodes, a learning rate of .1, and a momentum of 0. The model was allowed to train for 100 epochs without improving before training stopped.

The images were divided evenly into four rows and three columns. The average hue, saturation, and value was calculated for each of these rectangles and provided the only input into the machine.

The average resulting accuracy was 52% on the test set. The baseline for this problem would be classifying all the photos as landscapes in the standard upward rotation and would give us 42%. Therefore, the initial model did roughly ten percent better than our baseline.

4 Feature Improvements

4.1 Color Representation

Initially, the input features consisted of the average hue, saturation, and value for every rectangle. It was determined that hue would be difficult for the model to learn because values are circular, meaning that 0 is equivalent to 360. It was determined that a more continuous input should be used to represent the color.

The other attributes that were tested were the average RGB and average percent RGB (see section 2.2). The graph below shows the average percent accuracy over four different multilayer perceptron models trained with their respective attributes.

**<GRAPH>**

The RGB feature set performed marginally better than the rest of the features. Therefore, we selected RGB values to be the features in the experiments that followed.

4.2 Number of Rectangles

The next aspect of the features that was experimented with was the number of rows and columns that the image was divided into. When the image division was 1x1, the accuracy was 42.7%, which is the accuracy you would achieve if you predicted all images to already rotated correctly (the 'up' output class).

On average, the accuracy of a 4x3 division was 58% (as can be seen in 4.1). The following graph shows the accuracy of an 8x6 division trained with different numbers of hidden nodes.

<GRAPH>

The 8x6 division decreased the accuracy from 4x3 by roughly 4%. This was surprising since the model was being provided with more information to learn from. It was determined that the model might need more hidden nodes to process the more complex input. As demonstrated by the graph, the model requires 300 hidden nodes with an 8x6 division to obtain the same accuracy as a 4x3 division achieves (58%) with only 20 hidden nodes.

A model was also trained with a 16x12 division. A momentum of .9 and 300 hidden nodes were needed to achieve an accuracy of 56%. 4x3 appears to be the best division given the current input features.

4.3.1 Using Only Border Rectangles

During the iterations described in the previous sections, the images were divided into rectangles, and all of these individual rectangles were used as inputs. Upon reflection, we determined that solely using the borders might provide us with the same valuable information while only having to use a fraction of the inputs. The model could then rely more on using the ground and sky or ceiling to determine orientation.

The left part of the figure below shows the picture divided into 48 equal rectangles. The highlighted section of the right image shows the 24 border rectangles this variation would use.

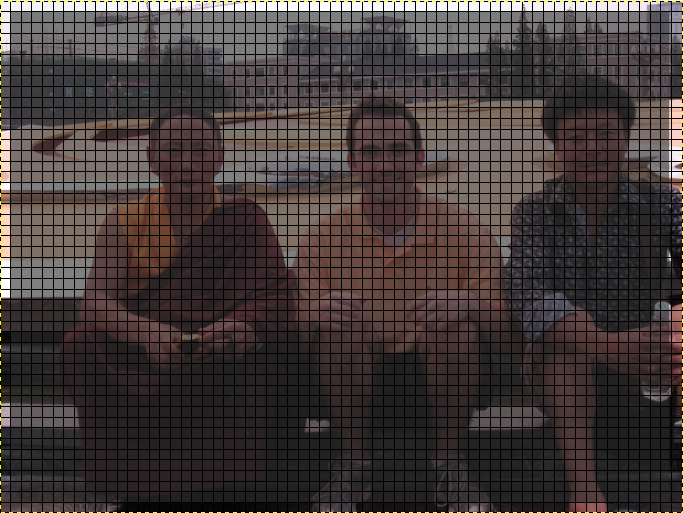


<pic\_grid\_orig.png> <8x6\_border\_hl>

By only sampling the border rectangles on a 8x6 division, we were able to achieve an accuracy of 58%. When we used a 16x12 division, again only sampling values from the border regions, we achieved a maximum accuracy of 61%. Thus, for the 8x6 division, there is no difference in accuracy from when all regions of the picture were sampled, versus when just the borders were used. For the 16x12 division, there was 3% increase in accuracy when using solely the border regions. Because the 8x6 accuracy with and without borders is identical, just using borders is the better choice because it reducing the space and time required by the learner. For example, in the 8x6 division example, only using border reduces the number of features from 48 rectangles \* 3 values/ rectangle = 144, to 24 \* 3 = 72, which is a 50% reduction. In the 16x12 instance, only using border rectangles reduces the number of features by 72% (192 \* 3 = 576 to 52 \* 3 = 156).

4.3.2 Border Variance

As a variation to the method described in 4.3.2, instead of using the normal RGB output values for each border region, we instead outputted a result that combined both the average red, green, and blue colors and the variance for each of those colors for each border (a total of 6 features per border). For example, an image that originally had 64x48 divisions would have four borders: the left and right borders have 48 rectangular regions each and the bottom and top borders have 64 rectangular regions each. For each border, we computed that border's average RGB and color variance by summing up the individual values for each in the individual regions composing that particular border and divided by the number of regions. Consider the image below:



<boravgvar.png>

The bottom border has 64 rectangular regions, each of which has their own value of red, green, and blue, and their own color variance of red, green, and blue. The bottom border's average red value would then be calculated using the equation:



<codecogseqn.png>

where *i* is a single rectangular region along the border. The calculations for the other 5 values for each border are computed similarly.

In this way, every image, regardless of the number of divisions or resolution, has 24 features (6 from each border, \* 4 borders). Using this combination of variance of each color and average color for each border, we obtained an accuracy of 69%. This was an 11% improvement over any other previous feature set.

4.4 Face Detection

As a final variation, face detection was used. We used the standard haar cascade classifier in OpenCV for facial detection. Each image was rotated in all four directions and the number of faces detected for each rotation was outputted as a feature; thus the total number of features was four. Intuitively, the face detection works because more faces should be detected in a correctly oriented image

Using the four face detection features alone, we achieved 68% accuracy on the test set. This was only slightly worse than our previously best obtained accuracy by using border average color

5 Other Models

5.1 K-nearest neighbor

An k-nearest neighbor model was also used. RGB values were calculated for each region of the picture, in addition to border region values, and used as the features. Running knn on this un-normalized data, an accuracy of 54% was achieved. When ran on normalized data, knn prodcued 51% accuracy.

6 Final Results

By using an ensemble that combined face detection features described in 4.4 with the border region average/variance features described in 4.3.2, a final accuracy of 76% was achieved. This combined accuracy surpasses the accuracy of either of those two techniques alone.

7 Conclusions

The multilayer perceptron trained with backpropagation was able to improve over the baseline 42% accuracy, to a final accuracy of 76%.

Our initial attempt used hue, saturation, and value of each region of the picture as input features, which resulted in 52% accuracy. After switching to using normalized red, green, and blue values for each region as input features, were able to improve our accuracy to 58%. By then adjusting the number of regions that we divided each picture into to 4x3, we were able to maintain the same accuracy as when we used a greater number of divisions, with fewer hidden nodes. Therefore, we found 4x3 to be the best division. Then, we experimented with only using the border regions of each picture, first using all of the rectangles making up the borders with their respective RGB values; this, however, did not result in any significant improvement or decrease in accuracy, but did simplify and speed up our learner. Focusing on just the border regions, we simplied the number of features further by only using as input features the average red, green, and blue values and variance among those colors for each border, reducing the number of total input features and increasing our accuracy to 69%.

Finally, we used a face detection algorithm to output the number of faces detected for each orientation as a feature. Using this alone resulted in 68% accuracy. Our final result combined both this face detection and the border region variance/average values as input features, resulting in a final accuracy of 76%. See the appendix for a graph showing the improvements in accuracy over each type of feature set.

8 Future Work

(TODO)Aquiring more data would be the most beneficial in improving our achieved accuracy.

Other object detection

<Detecting faces would be an easy way to determine orientation. A facial detection algorithm could be developed to add accuracy to classifying photos containing people.>

Acknowledgments

Louis Daguerre for the invention of the modern camera. Ray Kurzweil for the invention of the flat bed scanner.

References

[Jewett, 2009] Jewett, Tom. *Color tutorial.* California State University, Long Beach, California, 2009; <http://www.tomjewett.com/colors/hsb.html>.

Appendix: