Methods for Feature Extraction on Tiny Images for k-Nearest Neighbors Classifiers

Eric Miller, University of Georgia at Oxford

27 June 2020

This will be the abstract.

1 Introduction

2 Methods

2.1 k-Nearest Neighbors Model

The k-nearest neighbors model was implemented as a Python class, with a methods for training and testing. The `train` method simply stores the training data, while the `test` method does the brunt of the work. When the test data is loaded in, L1 distances are computed for every pair of members between the testing and training sets. The `cdist` function, which uses C-level code, from `scipy` was used in order to optimize this process.

Next, for each member of the test set, distances to training data members are sorted increasing order, and training labels are sorted accordingly. Hence, the first k training labels in the sorted set can be used to vote for a label for the test data member. For simplicity and speed, k may be passed into the `test` method, and distances are not recalculated or resorted if not necessary.

A separate model, which interfaced with `sklearn`'s `NearestNeighbors` module, was implemented in order to provide a baseline for runtime.

2.2 Feature extraction

Originally, the images were fed in as raw pixel values. However, due to the sheer size of the dataset (3072 values for each image), as well as the nature of the model, which would compare the images pixel-by-pixel and not account for invariances, feature extraction methods provided notable increases in testing accuracy. These methods transformed the image arrays into various feature arrays, upon which the model would work and calculate distances.

2.2.1 Grayscale and Average-Value Images

The first feature extraction method was a simple grayscale converter. As human vision has different sensitivities to different colors, those with higher sensitivity have greater weight on the grayscale image. For this purpose, the brightness value was the dot product between the RGB array (consisting of 3 floating point values between 0 and 1) and the array `[0.2989, 0.5870, 0.1140]`.

Average-value images were also created, weighing each original color equally. Both of these methods shrank the 3072-value arrays down to 1024 values. However, they were still subject to the same constraints of k-NN models.

2.2.2 Spacial Histograms

One way to combat this issue is to create spacial histograms of pixel values. Within each specified window (e.g., the entire image, quadrants, sixteenths, etc.), and for each color (red, green, and blue), a histogram was calculated for the distribution of the color's brightness across all pixels in the window.

The variable parameters here are window size as well as bin size for the histograms. As window size and bin size increase indefinitely, the spacial histogram gets closer to the raw image, represented in a different (more verbose) format.

Theoretically, this fixes the earlier problem of using raw images (to an extent): instead of comparing raw pixels to pixels in the same locations, distributions of pixel values for each color are compared across general regions.

2.2.3 Histogram of Oriented Gradients

In image processing and computer vision, discrete derivatives/gradients are often used for edge detection. Large gradients indicate a strong linear shift in lighting, often indicating the presence of an edge or other feature. A Histogram of Oriented Gradeints (HoG) is a common way of storing the distribution of gradient vectors throughout an image.

The first step to this is determining a grid size, which was fixed at 8×8 pixel squares (forming a 4×4 grid). Within each square, the gradient is calculated for each pixel. The direction derivative is defined as

$$D_{\vec{u}} f(\vec{x}) \lim_{h \to 0} \frac{f(\vec{x} + h\vec{u}) - f(\vec{x})}{h}.$$

For $\vec{u} = \hat{\mathbf{i}}$, the directional derivative (also the partial derivative with respect to x) can be discretized as $(f((x, y) + (2, 0)) - f(x, y))/2$ with $h = 2$ (the smallest distance that avoids half-steps when centering). If we ingore the scale by half (since we care only about directions and relative magnitudes of the gradient), and shift by one, we get the partial derivative $f_x \approx f(x+1, y) - f(x-1, y)$, which can be represented by the convolution given by kernel $[-1\ 0\ 1]$. Similarly, for the y partial, the kernel $[-1\ 0\ 1]^\top$ may be used.

These convolutions are applied across the images (with 0-padded edges to retain shape) in order to obtain 8×8 matrices of the x and y gradients. These are converted to polar coordinates, producing a magnitude and direction for the gradient centered at each of the 64 pixels. Either signed gradients (between -180 and 180) or unsigned gradients (modding by 180, so opposite gradients are represented the same) may be used. This is done for each color, and, for each pixel, the polar tuple with the largest magnitude is used. A histogram is then computed for the directions throughout the square, weighted by their respective magnitudes.

Once each square in the grid is assigned a histogram, they are normalized in groups. Each adjacent 2×2 block of these squares (consisting of 16×16 pixels) is normalized. In a 32×32 resolution image (with a 4×4 grid) there are $3 \times 3 = 9$ possible placements for the normalization block. Within each block, the four histograms are concatenated and normalized (with an L2 norm) in order to account for differences in lighting. All of the normalized histograms are hence concatenated into a single array.

2.3 Concatenating Multiple Features

[Will fill in later]

3 Results and Analysis

3.1 *k*-NN Model

The custom k-NN completed training and testing for the entire dataset in a total of 2033 seconds, compared to 1984 seconds for the `sklearn` version.

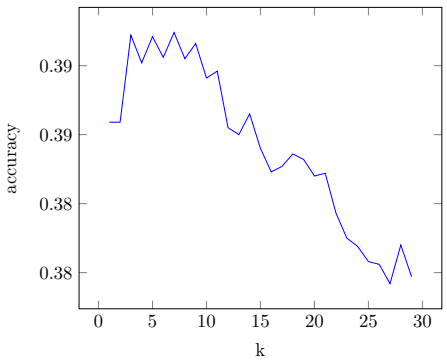


Figure 1: Accuracy of k-NN with raw pixel values.

In figure 1, we see that

3.2 Feature Extraction

3.2.1 Grayscale and Average-Value

The image below shows

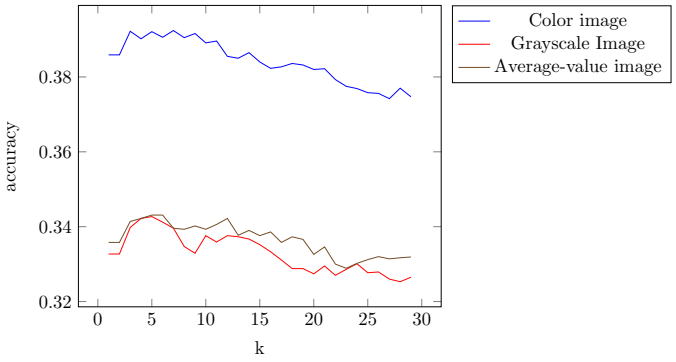
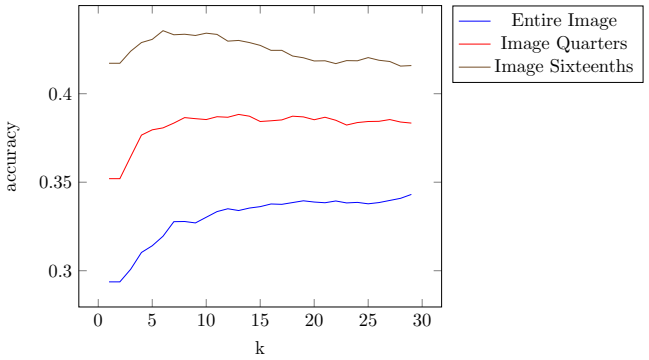


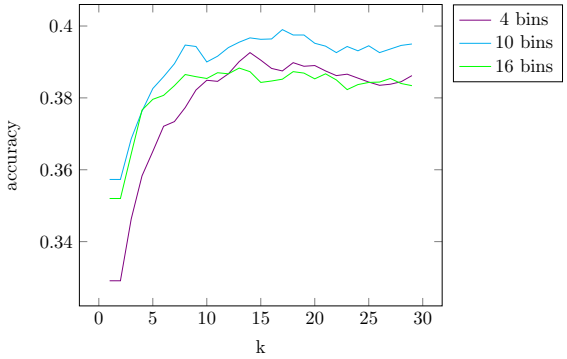
Figure 2: Accuracy on color versus grayscale and average-value images.

3.2.2 Spacial Histograms

The image below shows



(a) Fixed bin size of 16, with varying window size.



(b) Varying bin size, with fixed window size (image quarters).

Figure 3: Spatial histogram feature extraction results.

3.2.3 Histogram of Oriented Gradients

The following graph reveals

3.3 Concatenation of Multiple Features

The first graph

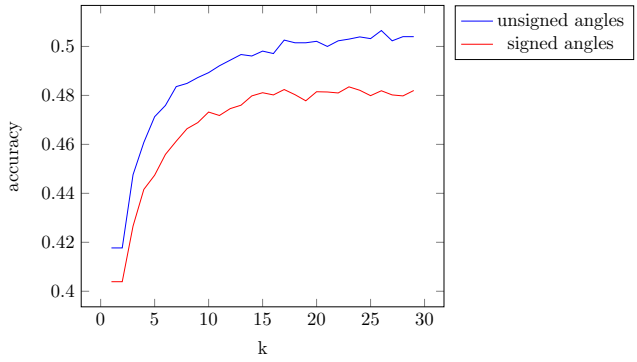


Figure 4: Histogram of Oriented Gradients Extraction.

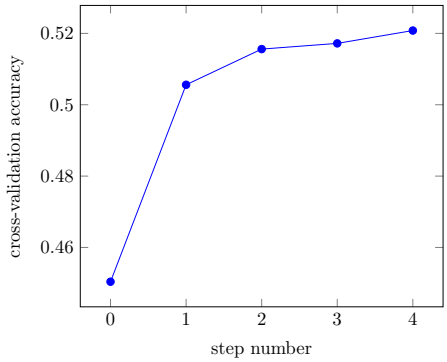


Figure 5: Steps of hill climbing search for concatenation weights.

The second graph

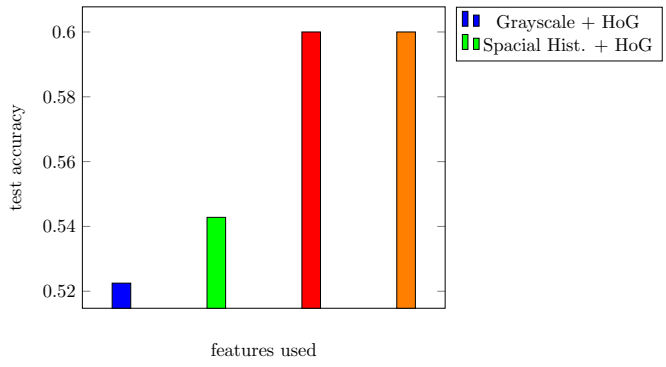


Figure 6: Accuracies of feature concatenation methods.

References

[1]

[2]