Machine Learning for Signal Processing
Automatic Image Coloring

Suhas Jagadish
School of Informatics and Computing
Indiana University
Email: jagadiss@iu.edu

Sid Thiruvengadam
School of Informatics and Computing
Indiana University
Email: sidthiru@umail.iu.edu

Arjun Rao
School of Informatics and Computing
Indiana University
Email: arsrao@umail.iu.edu

***Abstract*—Image coloring is a difficult problem, two objects with different colors can appear to be the same on grayscale layout and often requires human input to be successful. We present a method for colorizing grayscale images by transferring color from a reference image without direct user input. Our method colorizes one or more grayscale images, based on a segmented reference color image. The segmentation consists of one or more regions in the image, each assigned with a unique label. These regions need not cover the entire image, but each region should be uniform in color and texture. We classify the pixels of grayscale images to find out which region they match the best and consider only such pixels whose match has a high level of confidence to complete the colorization.**

1. Introduction

The meaning of term colorize according to Merriam-Webster is to add color to (a black-and-white film) by means of a computer, to put it simply, it means color it. There are many problems that are easy for a computer to solve but challenging to humans, example multiplying 10 digit number with 20 digit number, but there are other problems which are elementary to human being but very difficult for computers to solve, example coloring a black and white image. Even a three year kid identifies that sky is blue, grass is green and snowcapped mountain white, what about computers? How can computers identify and color a grayscale image?

In this project, our method colorizes one or more grayscale images, based on a user provided reference. This requires considerably less input from the user than scribbling-based interfaces, eliminating user intervention to a larger extent.

We use Discrete Cosine Transform(DCT) coefficients to build the feature space and KNN algorithm for classification. We represent images in the YUV color space, rather than the RGB color space. In this color space, Y represents luminance and U and V represent chrominance.

2. Background

Existing work with image colorization typically attempts to interpolate colors based off color scribbles supplied by an artist.

For example Levin et al. [2] proposed solution which was simple yet effective user-guided colorization method. In this method, the user is required to scribble the desired colors in the interiors of the various regions, which automatically propagates the scribbled colors to produce a completely colorized image. Other algorithms based on color scribbles have subsequently been proposed [3] [4].

The obvious draw back of the above suggested method is the manual effort required for the scribbles and the color pallet must be chosen ever so carefully to get somewhat desirable result, scribbling wrong would obviously result in wrong colorization, the onus is on the scribbler, which could either be good or not. One way to solve this is using reference image, in fact Welsh et al. [1] proposed an automatic colorization technique that colorizes an image by matching small pixel neighborhoods in the image to those in the reference image, and transferring colors accordingly.

Just finding a good match between pixel and its neighborhood image in gray-scale and the pixel image in the reference is not enough for satisfying colorization. Pixels with the same luminance value and similar neighborhood statistics may appear in different regions of the reference image, which may have different semantics and different colors.

To improve the results in such cases, Welsh et al. propose letting the user select pairs of corresponding swatches between the example and each input image, thus limiting the search for matching neighborhoods to particular regions. However, this user-assisted variant still relies on pixelwise decisions and does not enforce contiguity.

In this project, we try to leverage the advantages of these two colorization approaches, while avoiding their shortcomings. Our approach is largely based on the algorithm described in [7].

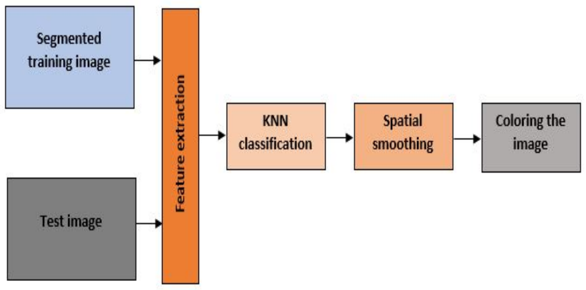
3. Method

Our approach consists of the following stages: (i) Training, (ii) Classification, and (iii) Color transfer.

In the training stage, we segment the luminance channel of the reference image and provide it as a training set to a supervised learning algorithm. The testing image can be a colored image, which will be converted to grayscale in out code.

In the classification stage we determine, for each grayscale image pixel, which region should be used as a color reference. We achieve this using KNN classifier by voting among pixel's nearest neighbors in the feature space determined in the previous step. We additionally employ voting in image space for better accuracy.

The matches found for each pixel also determine the color that should be assigned. In the final stage, colored pixels with high level of confidence are selected and colored. The overall flow of our approach is described below -



3.1. Building the feature space

Given the reference color image, our first task is to construct a feature space. We know that segmentation results in several regions, each associated with a unique label. Every pixel in one of these regions defines a labeled feature vector.

We need to classify the pixels from the test grayscale image and hence our classifier cannot rely on the colors of the pixels in the training image. The classifier should distinguish between different labels mainly based on texture. So each pixel should be associated with a feature vector representing its texture.

We decided to use DCT coefficients of a k by k neighborhood around the pixel as its feature vector. One of the advantages of using DCT coefficients is that they are simple texture descriptor, which is not too sensitive to translations and rotations. The DCT transformation is applied only to the luminance channel of the reference image.

This yields a k^2-dimensional feature space, containing labeled feature vectors corresponding to the training image pixels. Feature space construction is part of our "**Phase 1.py**"

3.1.1. Dimensionality reduction and feature transformation.

The KNN classifier computes the difference vectors between features obtained from the training image and the unseen features obtained from the test image. The difference vectors between features within the same segment are termed as *intra-differences*, and the difference vectors between features across segments are termed as *inter-differences*. Since our approach to labeling is segment based, it is important to tune the classifier such that it assigns labels mainly based on *inter-differences*. This is achieved by applying a transformation (using PCA and projections) to the feature space. This process can be described as follows:

- Randomly sample N intra-difference vectors(N = 700 here) from the original feature space(S).
- Perform PCA on these sampled vectors, and drop the eigenvectors that correspond to the first few highest eigenvalues. This is the first set of basis vectors(B1).
- Using the remaining eigenvectors, rotate the original feature space to get a new space (S1).
- Randomly sample N inter-differences from S1 and perform PCA on it. This time, keep the eigenvectors corresponding to the highest eigenvalues. These are the final basis vectors (B2).
- Transform S using B1 first and then B2 (B1 and B2 as obtained above). This time we are not sampling N vectors, but transforming the entire dataset.

Thus, the difference vector between two pixels p and q, computed by the KNN classifier is effectively the L2 norm of $T[f(p)] - T[f(q)]$, where T is the two-stage transformation described above, and $f(x)$ is the vector of DCT coefficients corresponding to the $k * k$ neighborhood of the pixel. This is part of our code in "**Phase 1.py**"

3.2. Classification

Given a previously unseen feature vector, the goal of our classifier is to decide which label should be assigned to it. We decided to use a classification approach based on the K-nearest-neighbor(KNN) rule. This classifier examines K nearest neighbors of the feature vector and chooses the label by a majority vote.

Once the feature space has been generated with the labeled vectors, the test feature vector can be classified by assigning it the label of its nearest neighbor.

When we applied the KNN algorithm to the high dimensional feature space, the results were not satisfactory. KNN classifier(or any classifier for that matter) when used on this feature space may fall prey to the curse of dimensionality. There will be significant improvements in the result if we use a low-dimensional subspace instead. We decided to use Principal Components Analysis(PCA) to achieve this. Further, PCA is also used to apply a basis transformation to the feature subspace that will improve the performance of the classifier.

3.2.1. Spatial smoothing.

The KNN classification on the transformed feature space is far more robust than KNN classification, but the labeling process till now has not made any spatial considerations. In other words, a pixel's neighbors haven't been directly used in its label assignment.

So, the classification can be enhanced by enforcing spatially coherent labeling.

Let $N(x)$, the k*k neighborhood around a pixel x in the input image. Often, this neighborhood will contain many pixels that have been labeled differently by KNN. In such situations, the labels are smoothed using the following idea: replace the label of x with the dominant label in $N(x)$. The dominant label is the label with the highest confidence. Confidence is defined as,

$$conf(x,l) = \frac{\sum_{y \in N(x,l)} W_y}{\sum_{r \in N(x)} W_r}$$

Here $N(x, L)$ is the set of pixels in $N(x)$ with the label L. The weight W_y depends on the distance between the pixel y and its best match among the training feature vectors. The best match refers to the nearest neighbor of y in the feature space, which has the same label as y.

Thus essentially, what we are doing is something like: ”if most of the surrounding pixels belong to segment 5, the given pixel should also belong to the same segment”. This is part of our code ”**Phase 2.py**”.

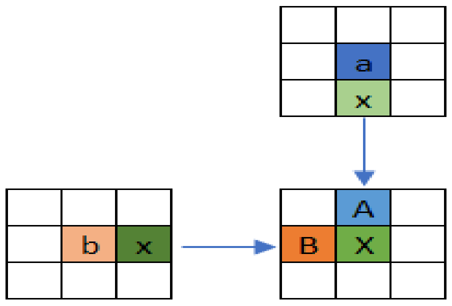
4. Coloring

As stated previously, we are converting all images into the YUV color space, where Y refers to the luminance channel and U and V refer to the chrominance channels.

Let $C(x)$ denote the chrominance of a pixel x. After classifying and labeling each pixel in the test image(as described in the previous two sections), the color values are obtained using a weighted average, shown below in this formula:

$$C(x) = \sum_{y \in N(y,l)} W_y C(M_y(x))$$

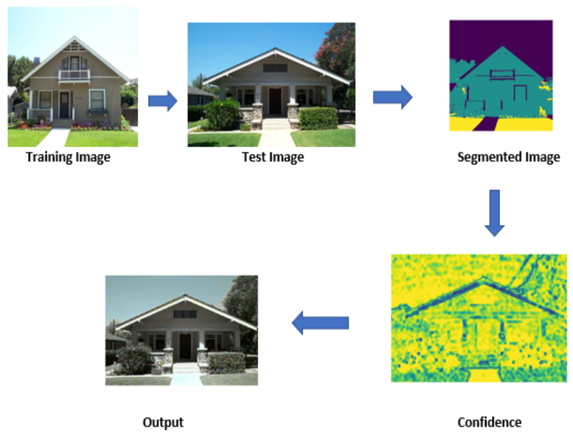
M_y is the best match of y in the training image. $M_y(x)$ refers to the pixel in the training image whose position relative to M_y is the same as the position of x relative to y. In this manner, the color coordinates are obtained for each pixel in the target image. This coloring process is described in the figure show below.

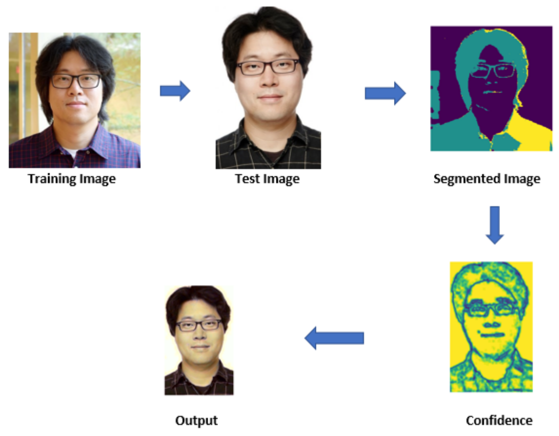


X is a pixel that needs to be colored, A and B are its neighbors. a is the best matches of A in the training images, and x is pixel whose position relative to a is the same as the position of X relative to A. This is part of our code ”**Phase 2.py**”.

5. Results

We chose 7*7 neighborhoods for our computations, so our initial feature space has 49 dimensions, which correspond to DCT coefficients. While the method does not reproduce all colors correctly, it in general produces plausible coloring results. The below figures illustrate the results on two different training images





6. Conclusion

In this project, we have presented and implemented a method for colorizing grayscale images using a reference image. The method uses a segmented reference color image as input and compute the DCT coefficients to build the feature space. We classify the pixels of the test image using KNN algorithm by assigning them to the label of its nearest neighbor. To further improve the accuracy, we employ voting in image space in addition to voting in feature space. We choose the pixels with highest level of confidence and finally color the entire test image.

Our algorithm was implemented in Python 2.7 using NumPy [5] and scikit-learn [6] libraries. It generally takes around 20 minutes to train and test our algorithm.

7. Future work

There are a number of measures than can be taken to improve on the performance of the algorithm.

- Currently we use only one training image to generate the feature space. We can use more than one reference images for training so that we get a much bigger feature space. This could also improve the accuracy of our approach.
- We currently use KNN algorithm for supervised classification. We can try CNN or GMM clustering or SVM and see whether the addition would further improve our results.
- In order to improve the coloring, we use image space voting in addition to feature space voting. We can further improve the accuracy using MRF smoothing of the classification results.
- We currently use the YUV color space to separate luminance and chrominance. Alternate color spaces exist that may do this more effectively (CIELAB).

Acknowledgments

We would like to thank Prof. Minje Kim for his support and guidance throughout the course. The concepts taught in the class were very clear and this helped us in better understanding the colorization problem. We sincerely believe that this course is by far the best Machine Learning course at Indiana University.

References

[1] T. Welsh, M. Ashikhmin, and K. Mueller, Transferring color to greyscale images, ACM Transactions on Graphics (TOG), vol. 21, no. 3, pp. 277 to 280, 2002.

[2] A. Levin, D. Lischinski, and Y. Weiss, Colorization using optimization, in ACM Transactions on Graphics (TOG), vol. 23, pp. 689694, ACM, 2004.

[3] SAPIRO G.: Inpainting the colors. IMA Preprint Series, Institute for Mathematics and its Applications University of Minnesota, 1979 (2004).

[4] YATZIV L., SAPIRO G.: Fast image and video colorization using chrominance blending. IMA Preprint Series, Institute for Mathematics and its Applications University of Minnesota, 2010 (2004).

[5] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, 2001.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research, vol. 12, pp. 28252830, 2011.

[7] Colorization by Example R. Irony , D. Cohen-Or, and D. Lischinski Tel-Aviv University The Hebrew University of Jerusalem