Auto-Coloring Network

Coloring target image with color extracted from source image

20135067 Taewon Yoon 20145071 Junghoon Seo
20145108 Donghwa Lee 20145039 Jinwoo Kim

Summary

Usually, animator repeats tedious and boring work that makes similar pictures colored. Due to the indeterminate nature of the problem, image colorization techniques currently rely heavily on human interactions. This is a good problem to automate because training data is easy to get: any series color image can be desaturated and used as an example. We get establishment of the goal that the system takes the source image that guides feature-color mapping and it let the grayscale target image color. Our artificial intelligence system fundamentally based on CNN is set with several image processing techniques. Pre-processing includes FAST detector, canny edge detector, k-means clustering image segmentation and post-processing includes La*b* color space transformation. Our system works with 2 similar images, one color source image, and one gray-scaled target image. Pre-processed color source image goes into CNN for learning. After learning, the gray-scaled target image can be color-labeled with color probability on each pixel from the learned CNN. After the color label made by CNN combines with k-means clustered color label, we can get the color probability of the image. Then it goes through the post-processing – the La*b* color space transformation. From our approach, roughly 70~80% of result image color map was accurate.

Describing Problem

Our first problem was “To fix the fake or missed labels of the Pixiv’s image”. There are many images that contain wrong labels tagged by bullies or have right labels missed as uploaders forget to tag labels on their images. It is one of the big problems in Pixiv community, as needs of the people to search for the high-quality images, those fake and missed labels are huge obstacles. However, we need to change our target problem from correct label tagging to coloring because of the following reasons. First of all, auto labeling problem is not adequate for 4 people project, as it is hard to distribute workload equally. Also, the amount of images which contain correct labels on Pixiv was too small. Considering we use machine learning for the auto-labeling network, the size of image pool was too small as big size of the learning set is needed to apply. Lastly, even if the images are in the same category, or represent the same character, they were too different from each other. It was hard to determine which character is drawn for each picture even by our eyes. So we set our target problem as auto-coloring. Specifically, we set the main objective of our project “To color the gray image which has a similar image that is colored”. The coloring is one of the hard and tedious parts in drawing an image. For example, if you are making animation, you need to color many images even if they have similar composition and background. Our approach fills the colors of the target image from similar colored source image. By using our auto-coloring solution, animator doesn’t have to color the similar image again tediously. Auto-coloring also uses the machine learning concept we initially used, label the colors in the image to use it for coloring process.

Project objective

The original plan was to automatically label moe-factors for images found in Pixiv. However, there were some problems like difficult to get good source images, different pictorial style or too few properly labeled images. On this account, we change to a new plan that automatically colorizes grayscale images in anime or manga. We specified the category as anime or manga, because a boundary line of anime and manga images is adequate for color segmentation process. Anime and manga also have advantages that it is easy to get nice source images and grayscale images, and have resembled drawing styles. Our main objective is to colorize a grayscale image from a given source image that has similar grayscale image, and paint in colors. The auto-colorization algorithm can improve an efficiency of painting while sketching.

Approach and Methodologies Investigated

We get an idea from the paper 'Automatic grayscale image colorization using histogram regression'[1]. We research through some papers and finally get an idea from this paper. As we don't have enough learning set for using general CNN approach, we choose an idea that uses only 2 images for the colorization. The research for the colorization paper is equally distributed to 4 members and the list of most important and influenced papers we reviewed is in Appendix 1. So our basic approach is same as below

The basic concept of the paper is using a histogram. Automatic grayscale image colorization using histogram regression method is a novel method to perform automatic grayscale image colorization. Histogram-based image segmentation uses a histogram to select the gray levels for grouping pixels into regions. So, apply locally weighted linear regression to the luminance histogram of the source image and the grayscale image to realize histogram segmentation. Next, the zero-points adjustment algorithm is suggested to implement the pairing procedures between the histograms of the source image and the grayscale image.

However, we want to use Convolution Neural Network (CNN) method instead of histogram regression because we want to approach new method that different already exists. The main reason for change is to use the unintended learning method in the neural network. Just using feature extracted from target and source image, we can use original concept of CNN compare to paper. Also, by changing the histogram regression to CNN and some feature extracted from the source and target image, we can make our own unique coloring approach. We researched and use 4 difference method for designing the whole structure beside CNN. K-means, Edge detector, Corner & Blob detector for pre-processing to help to learn in CNN. La*b* color space for post-processing part to adjust the result. Each part is investigated by the individual. Below part are the theoretical background of each method.

CNN[4] investigated by Junghun

Deep networks are neural networks that are formed by many layers. These networks serve to predict continuous values from a given input. They consist of layers that realize a function of the form:

$$\mathrm{y} = \tilde{\sigma}(b + Wx)$$

where x and y are the n input and m output of layer, respectively, W is an m-by-n matrix of weights, b is a bias vector, and sigma is a non-linear transfer function, Both the weights and the bias are learnt through back-propagation, which consists of using the chain rule to propagate a loss to update the parameters. The loss consists in the error between the prediction of the network and the training data ground truth.

A simple convolution neural network is a sequence of layers, and every layer of a convolution neural network (CNN) transforms one volume of activations to another through a differentiable function. We use three main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly same with a regular neural network). We will stack these layers to form a full CNN architecture.

Convolution layers are special cases in which weights are “shared” spatially across an image. This has the effect of reducing the number of parameters needed for a layer and gaining a certain robustness to translation in the image. Typically, a layer is organized as a 2D image with multiple channels so that the components of the vectors x, y are pixels indexed by a cartesian coordinate and a channel number. The matrix W above is defined as a convolution of the image with a bank of filters, which, in the neural network view, means that the weights are “shared”.

The function of a pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. The depth dimension remains unchanged.

Feature detection investigated by Taewon

For edge Detection, we will use the Canny-edge filter[3]. Canny Edge detection is one of the famous Computer vison Algorithm which is made by John F. Canny. This algorithm is most famous and valuable edge detection algorithm for following reasons. It has the low error rate, well-localized edges and only one edge is a response to the real edge. But as it is a really heavy algorithm, so it uses some pre-conditioning process for optimizing it.

First of all, it uses the Gaussian filter before using canny-filter. As edge detection result is affected by the image noise, we need to reduce the image noise before using a canny-filter. Gaussian filter is well known and useful smoothing filter. The principle of the Gaussian filter is simple. It will make the filter kernel size (2k+1)*(2k+1) and calculate the matrix of the filter. The equation is same as below

$$H_{i,j} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right)\ where\ 1 \leq i,j \leq (2k+1)$$

After applying the Gaussian filter, it uses the Sobel mask filter to detect the Extrema which work as an edge. As the Sobel mask filter, we can derive the edge gradient and direction. From derived edge gradient and direction we can get the Extrema. We need to exclude some non-local Maxima which is called Non-maximum suppression. We can do the Non-maximum suppression by comparing the edge strength of current pixel and pixel that is around them. If it is largest compare to another pixel that is the mask with the same direction, it turns out to be local maxima and preserved. After Non-maximum suppression, Double Threshold compression is made. Applying two threshold value, which is a large and low threshold, we can get rid of the detected fake edge from noise. The edge that has gradient bigger than a high threshold is preserved and edge that has gradient smaller than small threshold they are dropped. The edge that has a gradient between high and low threshold, they are preserved only when it is related to the high threshold edges. Edge is tracked by hysteresis to find whether it is related or not. Edge tracking is using BLOB-analysis.

For Corner and Blob Detection, we choose the FAST corner detection algorithm[2] made by Edward Rosten at 2006. As it is the most powerful and fast algorithm in the corner and blob detection now. The compare result of the FAST corner detection and other is in Additional Figure 1[2]

FAST algorithm detects the corner by making the circle of a pixel which radius is 3 and check whether there are n-pixels that is continously darker(>p+t) or lighter(<p-t) than center pixel. The value of n is important for the speed and repeatability. It uses machine learning to drop out the detected corner that is not really a corner and makes it faster. It chooses the learning set and detects the corner by using the FAST algorithm and save the center pixel and 16 other around a center to P

vector. Each set of 16 pixels is classified to the 3 states which are darker, similar and brighter. Then boolean Kp is made and makes the entropy equation. ID3 is run recursively until entropy goes to zero. Then Decision tree for corner detection is made. To reduce the fake corners that are around real corner It uses the Non-maximal Suppression. Non-maximal suppression uses the score function V. V is a sum of the absolute of difference between center corner pixel and 16 other pixels around them. Compare the V of fake and true corner pixel, and drop out the smaller V valued corner pixel.

K-means algorithm investigated by Donghwa

The K-means algorithm is a method of vector quantization, one of the centre-based algorithms that attempt to find K clusters which pursue the minimum mean squared quantization error. The main idea of K-means algorithm is what observes K prototypes and relocates these K prototypes using a mean of a data set. The K-means algorithm repeats iteration of the main idea. Here is a summarization of the K-means clustering through the following steps. First, define the number of K prototypes and appoint a prototype(in this case, a vector quantity that is of the same dimensionality as the data) for each cluster in order to initialize. Second, allocate each data point to the nearest prototype. Now, that data point is a member of the partition confirmed by that prototype. Third, calculate the mean of all those members of the partition and find a new position for each prototype. Final, Observe the new prototypes' positions. If these prototypes' positions have not meaningfully changed over a certain number of iterations, finish the algorithm. If there are significantly changed over about values of prototypes' positions, then back to step second. However, the K-means algorithm has a problem that it is a dependency on the prototypes' initialization. If the initial prototypes are not chosen attentively the computation will run the chance of focused to a local minimum rather than the global minimum solution. The performance function for K-means algorithm is $J_{Km} = \sum_{i=1}^{N} min_{j=1}^{K} \left\| x_i - m_j \right\|^2$. K is a number of partitions, x are objects, m are mean of a data point.

La*b* color space investigated by Jinwoo

On the paper "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors"[5], it just compares when the input data set is given by RGB, YUV, and La*b*. Then it concludes that they should use La*b* as it has a slightly better result. The paper itself doesn't try to explain why this result happens. However, we can presume from the properties of each color space.

RGB doesn't have luminosity property. YUV does contain luminosity property(Y), but other properties (U, V) are not independent of luminosity property. Actually, its value can be easily calculated by simple linear multiplication of RGB value. In contrary to other color spaces, La*b* color space contains luminosity property, and other properties(a*, b*) is totally independent of lightness property, and shows the most similarity to the human eye, compared to other two color spaces. We can't briefly describe or predict how computer learn from data when we use Artificial Neural Network in a strict way, and we can't say computer learns the way how peoples do. However, considering the naturalness of the results were calculated from people in the paper, we can predict why La*b* color space shows the good results. So we are going to use La*b* space for post-process for adjusting the result of our network.

Implementation Detail

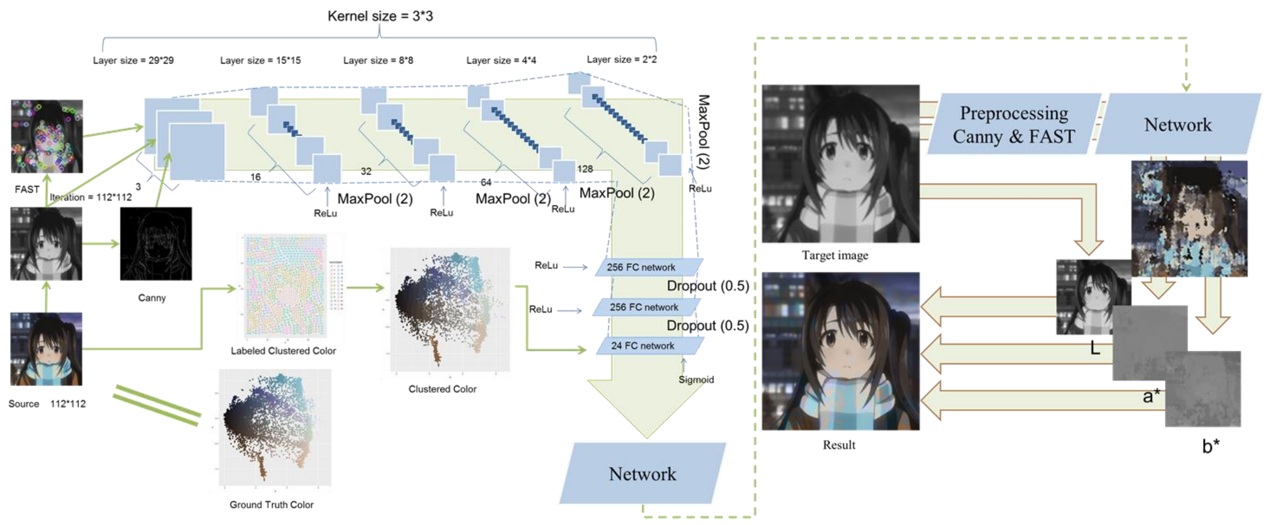


Figure 1 *Overview of our model for auto colorization of gray-scaled imaged based on similar color image.*

When source and target image comes to our coloring program, it goes through the Canny-edge detector and FAST corner and blob detector. Both filters is made by Taewon using OpenCV library with C++ and Python. Canny edge detector code is based on C++ with OpenCV library. It gets jpg format of source and target image and converts it into the matrix based on B,G,R channel. It makes the gray scaled image with gray-scaled filter and blurs it. The blurred gray-scaled image goes through the canny-edge filter. The threshold of our canny-edge filter is 20 and 80. As we test different threshold for the pre-process, we found out that 20 and 80 are a most reasonable threshold. As we use the animation image, threshold doesn't have to be big as it doesn't have noise compare to photos. The output of the canny-edge filter is resized to 112*112 with interpolation INTER_AREA type. By testing 5 different interpolation type for resizing, we choose the INTER_AREA type as it has the least noise made from resizing. Then the final output comes with jpg format. FAST filter code is based on Python code. Its input is jpg format and output is the csv format. The output of the FAST filter has an information of the key-points(corners) only. we also save the jpg result of the output for checking whether a corner is detected well. FAST filter code also uses INTER_AREA interpolation type for the resizing. By passing through the canny-edge and FAST filters, we can get gray-scaled target image, edge-detected target and source image, coordinated vale of corner and blobs in source and target image saved to csv format. So our first pre-processing filter has passed made by Taewon.

Concurrently with pre-process filter, source image go through k-mean clustering code made by Donghwa. K-mean clustering code is based on R. First, each pixel is merged as just one data frame. It consists of 112*112 by 3. The row is in-image row-wise sorted pixel and column is RGB channel value which has a value from zero to one. Based on merged pixel data, run k-means clustering. Cluster number is set 24. To adjust the center of cluster, center of mass critic Then, a table which let each cluster match with RGB color is saved as .rds(R Data Structure) file. Later, this is for regression from prediction label to image based on RGB color space. For visualization of a result of k-means clustering, principle component analysis technique is used. The ground truth color space and clustered color space is decomposed by same two principle component. The figure is shown in Fig.1 at a mid-left location. Also, for comparison between the original source image and clustered source image, they are exported by ggplot package. For ground truth data for convolution neural network, an 112*112-length array of which each element has its own clustered label number. Then, the array transformed to the data frame by one-hot coding. Additionally, the random sampling for network learning process is done.

Pre-process filter result goes into CNN network. CNN network code is written by Junghun. Python and Numpy, Tensorflow, TFlearn package is used for CNN. GPU acceleration with CUDA package is used for the computing power. The total input of the network is gray-scaled source, canny-edge detected source image and FAST corner & blob detected image samples. 3 image converted into 3 different 29*29 sizes 112*112 layers. each pixel in the image converted into 29*29 size layers. The total output of the network is 24 different clustered color label's relative probability. To reduce the over-fitting and to consider the difference between source and target image, learning processed source set is at most 10° rotated, and blued with Gaussian filter with at most 1.5 sigma setting. The mean of the weight parameter in a layer is 9 and the deviation is initialized to 0.02 normal truncated distribution. For addition, all bias parameter is zero initialized. The structure of the network is as below

Input layer(29-29-3) –> Conv layer(15-15-16) ->CL(8-8-32) -> CL(4-4-64) -> CL(2-2-128) -> Fully-connected layer(256)->FC layer(256)->FC layer(24)

In conv and input layer parenthesis contains (width-height-depth). FC layer's parenthesis means the number of nodes. Except the last layer, each layer used ReLU for activation function. In last Fully-connected layer, the soft-max function is used for activation function. Between Conv layer, max-pooling is used for the down-sampling, and to prevent overfitting batch normalization is used. In Fully-connected layer, to prevent overfitting, dropout technique with drop probability 0.5. For optimization, Adam, the modified version of SGD is used. Step epsilon is 1e-8 and learning-rate is 0.001. For loss-function cross-entropy is used. For each epoch, 112*112 iteration happens and total epoch is 100 times. In prediction period, Input is same as learning it use 29-29-3 layer is used. But in prediction period, the layer is made from the Target-image. The result table of prediction has 112*112 rows and 24 columns, each row-wise array means the relative probability of predicted color. The largest value of label goes into final table in each row. The result is 112*112 by 1 table and it is saved to .csv format.

The last process changes the predicted color label to RGB color and using La*b* color space for the post-processing. It is made by Jinwoo. R and C++ with OpenCV are used for this part. The result of K-means and the CNN color label prediction is combined to make the RGB result of it. Then the resulting image is up-sampled to 224*224 size with linear interpolation. We call this result palette. The palette has the information of target images predicted color and the coordinate position of it. But as it doesn't give enough information about the luminance, we combine La*b* transformation to the result. The target-image(gray-scaled) and palette is converted into La*b* color space. Then, the target image's Luminous value and a* and b* space of the palette image is combined into the total result image. We call this procedure, post-processing. Finally, the total result is 224*224 colored target image. It is the total result of our coloring system.

Result

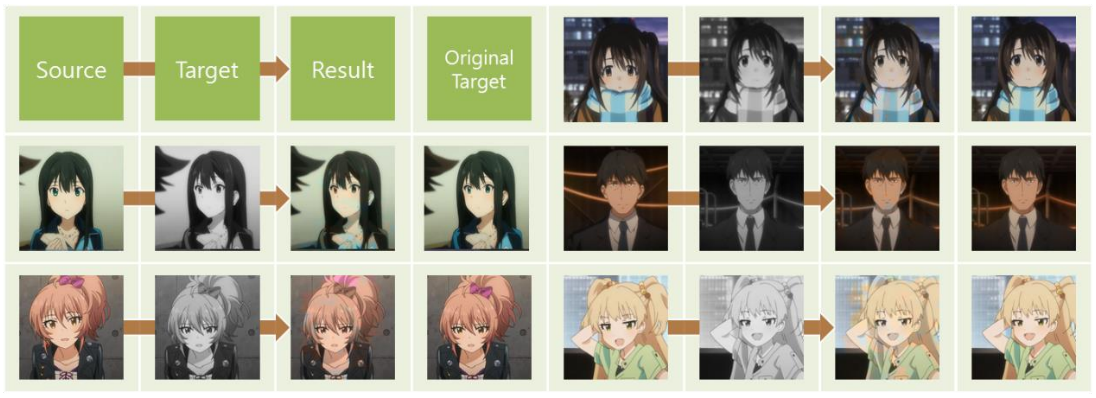


Figure 2 *The 5 different result of the our coloring system*

We made the system of machine learning the specific color image pattern and auto-coloring the gray-scaled image with the machine learned value. The computer used for the coloring system is GeForce GTX980Ti with 6GB VRAM, Intel i5-3690k @4.2GHZ, 16GB main RAM. The average time consummation for the CNN is about 13~15 mins for the learning period. The accuracy of our result compare to the original color target image, we saved for comparing, is about 0.7~0.8. Our result examples are in figure 2. In conclusion, we will discuss the better way to improve the system of our system.

Conclusion and Future work

We made a coloring system that first extracts the features and color from the source image, learns the pattern out of it, and at last colorizes the target image which is gray-scaled. We use 5 different techniques for our system - k-means, canny-edge detection, FAST corner and blob detection, CNN network and La*b* color space. Our accuracy of the result is roughly about 70% ~ 80%. If we can achieve the accuracy more than 95% like some papers at the top level has, we can liberate animators from tedious coloring procedures. To achieve this goal, we can try 3 ways in the future.

First of all, as the number of optimal K-means cluster is different by each picture, we need to make a decision algorithm which can suggest appropriate number of cluster labels. In our project, we fixed the size of color labels to 24 as we don't have the decision algorithm. However, the images which are anime-fashioned have lots of discrete and differences for each color distribution. Furthermore, the number of K-means cluster does a very important part in processes afterwards. For instance, if the number of clusters in CNN is low, matching rate is higher, but the transformed value of real color bound pretty ambiguous and different from the original. In contrast, if the number of clusters is high, the probability of right label for the color is decreased as the number of possible color labels is high. So, if we can make a proper algorithm for deciding the number of optimal color labels balancing these two aspects, we can get the better result image.

Second, if we do not assign the entire pattern matching algorithm to CNN, give weighting to unsupervised learning, and combine them, we might get a better result. The paper which brings an idea to our project uses unsupervised learning about extreme point of the histogram for pattern matching of two pictures and then infers color from regression. Similarly, pattern matching of CNN can adjust if our system uses the algorithm like SIFT descriptor.

Finally, if converting RGB space into La*b* space is conducted on the pre-processing level, it might improve the accuracy of the result as an information loss will occur less if we use two factors(a*, b*) rather than using three factors(R, G, B).

Reference

[1]S. Liu and X. Zhang, "Automatic grayscale image colorization using histogram regression", *Pattern Recognition Letters*, vol. 33, no. 13, pp. 1673-1681, 2012.

[2]E. Rosten, R. Porter and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105-119, 2010.

[3]J. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. -8, no. 6, pp. 679-698, 1986.

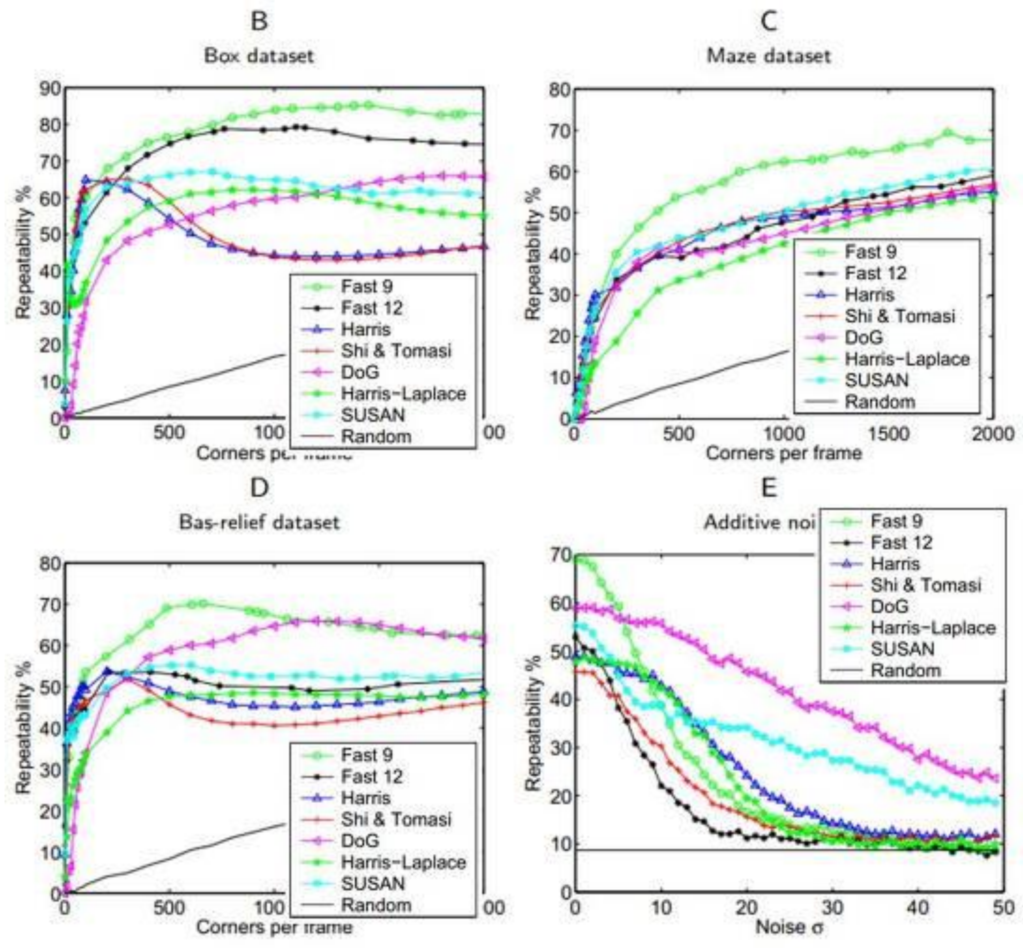
[4]"Stanford University CS231n: Convolutional Neural Networks for Visual Recognition", *Cs231n.stanford.edu*, 2016. [Online]. Available: http://cs231n.stanford.edu/. [Accessed: 09- Jun-2016].

[5] S. lizuka, E. Simo-Serra and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors," in *ACM SIGGRAPH*, Anaheim, CA, 2016

Appendix

[1]	S. lizuka, E. Simo-Serra and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors," in *ACM SIGGRAPH*, Anaheim, CA, 2016.
[2]	Y. Seldin, S. Starik and M. Werman, "Unsupervised Clustering of Images using their Joint," in *Proceedings of the 3rd International Workshop on Statistical and Computational Theories of Vision*, 2003.
[3]	A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional," in *Advances in Neural Information Processing Systems 25*, 2012.
[4]	G. E. Hinton, S. Osindero and Y.-W. Teh, "A fast learning algorithm for deep belief nets," in *Neural Computation*, 2006.
[5]	D. p. Tian, "A Review on Image Feature Extraction and Representation Techniques," in *International Journal of Multimedia and Ubiquitous Engineering*, 2013.

Additional Figure 1. List of the most important paper we reviewed before get the concept of approach



Additional Figure 2. Compare FAST algorithm with other corner detection algorithm