Old Image De-noising and Auto-Colorization Using Linear Regression, Multilayer Perceptron and Convolutional Nearual Network

Junkyo Suh (006035729), Koosha Nassiri Nazif (005950924), and Aravindh Kumar (006117685)

***Abstract*—We study about de-noising of old images using linear regression and auto-colorozing them using multilayer perceptron (MLP) and convolutional neaural networks (CNNs). Old images were mimicked by generating grayscale images with sparse black and white noise. It was found that implementing noise detection algorithm is critical to effectively reconstruct the original images and linear regression model performs surprisingly well with appropriate choice of threshold for the noise detection. For auto-colorization, we developed MLP as our baseline. It was observed that MLP significantly over-parameterizes the problem and hence overfits the training set. Thus, it performs poorly on the test-set, producing a test-error of ~3.4% (RMSE). However, CNN vividly colorizes images with the help of fusion layer accounting for local features as well as global features. Objective function was defined as mean square error (MSE) between the estimated pixel colors and ground truth, linked with classification results. As a result, images were colored with global priors and an excellent RMSE was achieved.**

***Index Terms*—Old image de-noising, noise detection, linear regression, auto-colorization, multi-layer perceptron (MLP), convolutional neural network (CNN).**

I. MOTIVATION

PEOPLE reminisce about precious moments in life with photos of family and friends. Some may have old pictures in which unwanted noise got easily incorporated from many sources of imperfection. Moreover, historic film footages were recorded by old devices in which case it is imperative to filter out unwanted signals to vividly and clearly reveal the scenes. On top of that, our application can be extended into restoring low resolution surveillance camera images, re-colorizing modern RGB images in different color schemes and Snapchat/Instagram filters. The objectives of this project are two-fold, artifact removal and auto-colorization.

II. METHODS

The IMAGENET data that contains 50000 photos (64×64 RGB, http://image-net.org/small/download.php) was initially used for de-noising. Due to the high computation cost of the auto-colorization algorithms and the relatively low computation power available, we switched to the CIFAR-10 data that contains 60000 photos of smaller size (32×32 RGB, https://www.cs.toronto.edu/~kriz/cifar.html). In both cases, the data was divided into 70% training and 30% testing datasets.

A. Generating Old Images

IMAGENET and CIFAR-10 images were converted to grayscale images with various types of noise incorporated to make them look old. As a baseline, we first dealt with grayscale images with sparse black and white noise.

B. . Linear Regression Model for De-noising

We used linear regression to de-noise images. A small scanning square (5×5) was delineated and moved throughout an image to learn our parameter. Feature selection and construction of matrices for input, output, and parameter are illustrated in Fig. 1.

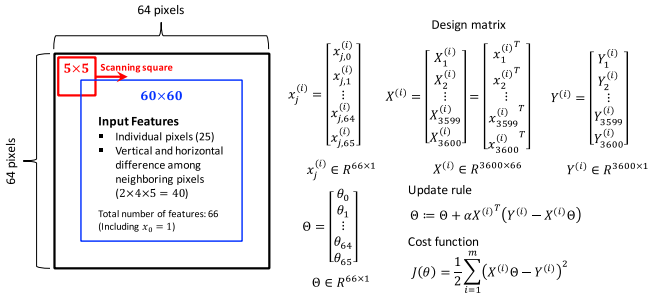

Fig. 1. Definition of scanning square for feature selection and construction of matrices for input, output, parameter.

Input features were chosen in the elements of a scanning square (5×5) as well as vertical and horizontal differences to the neighboring pixels to capture the surrounding information. For instance, in the case of 5×5 scanning square, the total number of input features is 66 including intercept term, individual pixels, and vertical and horizontal differences among neighboring pixels. Stochastic gradient descent was implemented in our algorithm such that parameter gets updated as the square scans not only within an image, but also throughout the whole training set.

C. Multi-Layer Perceptron Model for Auto-Colorization

We modelled the problem of auto-colorization as a regression problem and so each output neuron would predict the value of the pixel in the three channels [1]. A neural network with one hidden layer was used initially. The input is the grayscale image and the output would be the colorized imaged, which would be compared with the original colour image for training. The initial testing and validation of the model was carried out using the ImageNet dataset of 64×64-sized images, but later we moved on to the CIFAR-10 dataset, which contains 32×32 images, due to the huge computational cost of back-propagating through the fully-connected layers. since a CIFAR-10 image is 32×32 and has 1024 pixels, the input layer has 1024 inputs while the output

This report is a part of CS229 2017 autumn project.

Junkyo Suh is with the Department of Electrical Engineering, Stanford University, Stanford, California, CA 94305 USA (e-mail: suhjk@stanford.edu).

Koosha Naassiri Nazif is with the Department of Mechanical Engineering, Stanford University, Stanford, California, CA 94305 USA (e-mail: koosha@stanford.edu).

Aravindh Kumar is with the Department of Electrical Engineering, Stanford University, Stanford, California, CA 94305 USA (e-mail: akumar47@stanford.edu).

layer has $1024 \times 3 = 3072$ neurons. The back-propagation was carried out using mini-batch gradient descent. The neurons in both the hidden layer and the output layer are sigmoid neurons. This structure is shown in Fig. 2.

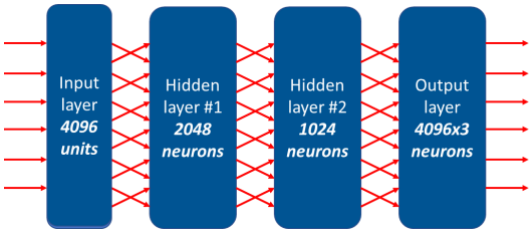

Fig. 2. The architecture used in our multi-layer perceptron model

D. *Convolutional Neural Network for Auto-Colorization*

In the recent years, convolutional neural networks (CNNs) have been a very successful model in many tasks, especially computer vision such as object recognition. Thus, we decided to implement CNN to auto-colorize grayscale images. In 2016, Ref. [3] proposed a novel approach to fuse local features with global features to capture semantic information in colorizing images. Our model is primarily inspired by their architecture such that our encoder is fused with high-level feature extractor to produce chrominance components of images. On top of that, we attempted to utilize the architecture in conjunction with one of the pre-existing models, Resnet.

First of all, we chose CIFAR-10 32×32 images as our dataset due to hardware limitations. The images were converted to CIE L*a*b* color space [4] because it divides RGB images into the color components and luminance information that contains image features. After that, the pixel values of all three image components are centered and scaled such that the obtained values are in between 0 and 1.

With the luminance component as an input, the chrominance (a*b*) is estimated through our model to restore fully colorized images, which is a mapping from luminance to chrominance. Our approach uses an encorder-decoder architecture and it consists of convolution layers with weight sharing and fully connected layers for classification and fusion. Here, the convolution layers are a set of small learnable filters that look for specific local patterns in input images and generate activation maps. Layers close to the input look for simple patterns such as contours, while the ones closer to the output extract much higher-level features [5]. ReLU was used as a transfer function and batch normalization [6] was performed at each layer to remove internal covariate shift.

The network architecture is illustrated in Fig. 3. The novelty of the proposed method in Ref. [3] comes from the fact that cross entropy loss is computed at the output of global feature extractor and the total loss is linked with classification results to capture semantic information for colorizing images.

Our network can be divided into four main parts that are encoding unit, classification unit, fusion unit, and colorization unit. For the output at the coloring unit, sigmoid transfer function is employed to output values in between 0 and 1, which will be scaled to RGB. In the middle of our network, the encoding unit and global feature extractor are fused to utilize not only specific local features, but also semantic information to boost performance of colorization.

- The encoder processes 32×32 gray-scale images and outputs a $8 \times 8 \times 128$ feature representation at the mid-level. It uses 6 convolutional layers with 3×3 kernels. Padding is used to conserve the size of input. Furthermore, instead of using max-pooling, different hyperparameter with stride of 2 was used to reduce the number of computations required while preserving spatial information.

- Global feature extractor is an important part of our network which has 4 convolution layers with 2 fully connected layers to classify 10 classes of CIFAR-10. The output of the global feature extractor is a 128-dimensional vector representation, which will be fused with mid-level feature vector for the input of colorization unit. Here, cross entropy loss is computed at the classification and the toal loss is linked with classification results.
- One of the most important pieces in our model is the fusion layer which was proposed by Ref. [3]. This is where global semantic information was fused with local features. This allows us to merge local information with global priors computed using the entire image. In our case, CIFAR-10 32×32 images were used along with the 10 class labels to learn the global priors. The output of the fusion layer for mid-level coordinates (x, y) as

$$y_{x,y}^{fusion} = \sigma\left(b + W\begin{bmatrix} y^{global} \\ y_{x,y}^{mid} \end{bmatrix}\right)$$
$$where\ y_{x,y}^{fusion} \in \mathbb{R}^{256}, y_{x,y}^{global} \in \mathbb{R}^{128},$$
$$y_{x,y}^{mid} \in \mathbb{R}^{128}, b \in \mathbb{R}^{256}$$

- Finally, the decoder takes the output of the fusion layer which is $8 \times 8 \times 256$ volume and applies a series of convolutional and up-sampling layers where 2D nearest neighbor up-sampling is performed.

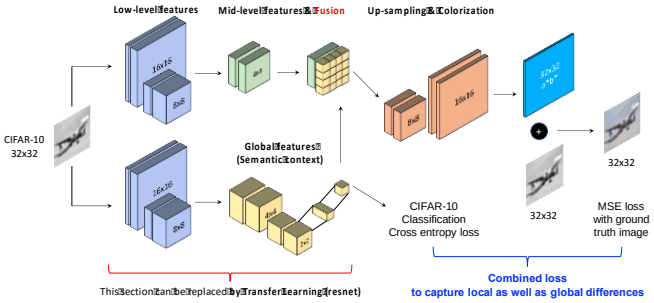

Fig. 3. An overview of our model architecture composed of low-level feature extractor, global feature extractor, fusion layer, and colorization network.

III. EXPERIMENTS

A. *De-noising Images Using Linear Regression*

Our de-noising linear regression algorithm only updates parameters when it detects noise in a pixel. It was found that how noise is detected and setting proper threshold for detection are crucial to effectively eliminate noise.

Our model was tested to unseen images with 4% pixel corruption with black and white noise. Fig. 4. (right) shows the predicted image where the image got blurry and cross patterns were observed because of averaging over surrounding pixels.

Here, it should be pointed out that it is important to embed noise detection algorithm to boost the accuracy of the model. Otherwise, prediction of a pixel is the same as the corresponding pixel in noisy image.

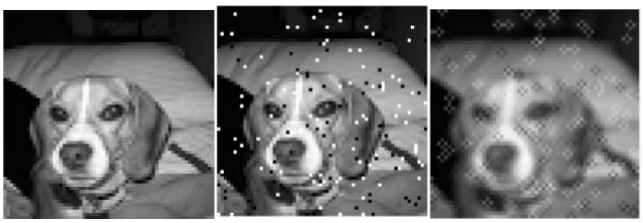

Fig. 4. Linear regression model without noise detection. Original (left), noisy (middle), predicted (right) images.

Our implementation of noise detection is as follows. When predicting the value of a pixel, 4 neighboring pixels (top, bottom, left, and right) were inspected and minimum difference was computed. If the minimum difference is greater than experimentally determined threshold, the pixel is considered to be corrupted with noise. Taking the minimum difference has the advantage that the model works at boundaries of objects. However, one can imagine that if noisy pixels cluster together, the model starts to lose its accuracy.

Fig. 5. Successful implementation of de-noising with noise detection. Original (left), noisy (middle), predicted (right) images.

Furthermore, we studied about the performance (test error, %) of our model dependent on the number of training images, the size of scanning square, and noise portion to image pixels. Our performance measure is defined as below (for instance, 5x5 scanning square).

$$error = \frac{1}{2m \times 3600} \sum_{i=1}^{m} \sum_{j=1}^{3600} \left(\frac{x_j^{(i)^T} \theta - y_j^{(i)}}{255} \right)^2$$

It was found that our model quickly captures sparse black and white noise with a great accuracy and test error plateaus after learning through ~100 images. On top of that, the model was working well with more severely corrupted images up to ~40% noisy pixels.

It is worth noting that our model is surprisingly robust against noise. Fig. 8. shows that it is able to restore concentric pattern where 64% of pixels were corrupted.

After successful de-noising of IMAGENET data set, we applied the linear regression model on the preprocessed grayscale CIFAR-10 images with 15% salt and pepper noise and then inputted the resulting de-noised grayscale images to the auto-colorization algorithms. The resulting test error was 0.12%. Fig. 9 illustrates the impressive performance of our de-noising algorithm on 16 sample images from the CIFAR-10 dataset, 4 from the training set (first row) and 12 from the test set (rows 2-4).

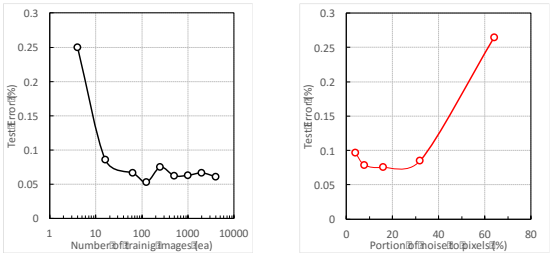

Fig. 6. Test error (%) as a function of number of images used for training (left) and portion of noise to pixels (right).

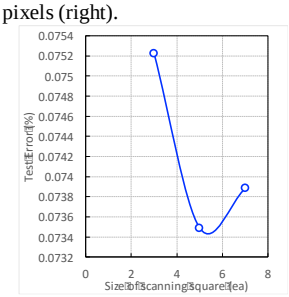

Fig. 7. Test error (%) as a function of size of scanning square. Smaller size is better to reduce blurriness of an image, however, it is in trade-off relationship with effectiveness of noise detection.

Fig. 8. Introduction of black and white noise to 64% of the pixels. The model still predicts the original image to some degree. Original (left), noisy (middle), predicted (right) images.

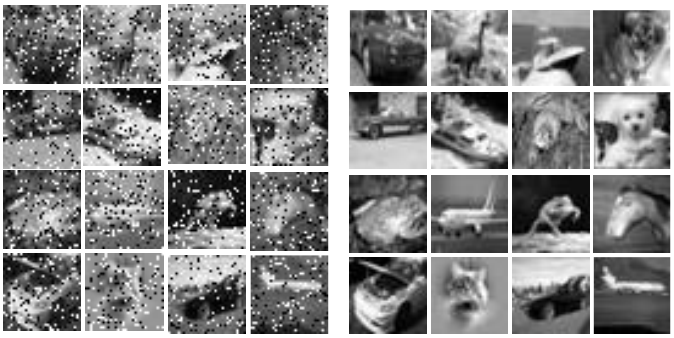

Fig. 9. De-noising of 16 images from CIFAR-10 dataset with 15% black and pepper noise. First row images are from the training set whereas the rest are from the test set. Noisy (left), predicted (right) images.

B. Auto-Colorization Using Multilayer Perceptron

In our first attempt, we initialized the weights and biases with a normal distribution. As a sanity check, the network was trained on a single example to see if it could over-fit the image. The results are shown in Fig. 10. As can be seen in figure, the network could not overfit the image. Moreover, the network did not learn beyond the first epoch, after which the gradients vanished. This indicates that either the sigmoid neurons were saturating towards 1 or 0, resulting in zero gradients, or the output of the earlier layers were vanishing. The solution to both of these issue is to standardize or normalize the inputs to each hidden/output layer. Another issue was that the network was

giving the same output for different inputs as can be seen in Fig. 11. This hinted that the biases probably dominated over the weights. One way to avoid this problem is to experiment with the initialization of weights and biases.

Fig. 10: Outputs of the 1-example sanity check of the MLP model

Fig. 11: The initial MLP model produces a very similar output for a very different input

To solve the above-mentioend issues, we normalized the data in the hidden and output layers by adding batch-normalization [1] (BN) layers with parameters γ and β, so that the network can also learn the distribution of the activations. The back-propagation method was modified since the BN layers have learnable parameters now and the network should learn them. The following equations describe the process:

1. $\mu_B = \frac{1}{m}\sum_{\{i=1\}}^{m} x_i$
2. $\sigma_B^2 = \frac{1}{m}\sum_{\{i=1\}}^{m}(x_i - \mu_B)^2$
3. $\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\{\sigma_B^2 + \epsilon\}}}$
4. $y_i = \gamma\hat{x_i} + \beta$

As can be seen in the equations above, the BN layer involves normalizing the data to a normal distribution and then a scale-and-shift using learnable parameters γ and β.

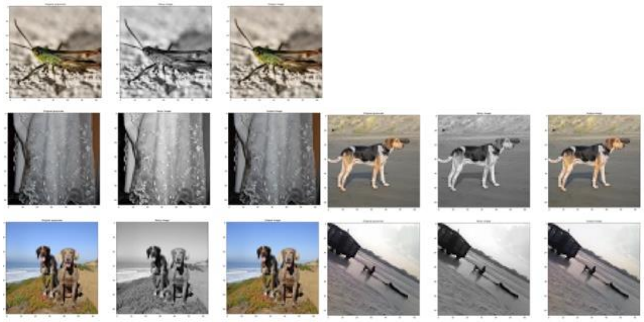

Fig. 12: The model resulting from the 5-example sanity check of the MLP model overfits all the 5 images and produces different outputs for different inputs

The network with BN could overfit one example but still suffered from the problem of vanishing gradients and also produced the same output for different inputs. To improve performance, we sampled multiple learning rates for the mini-batch gradient descent and this resulted in a much better

performance. We also tried different initialization of the weights and biases. Initialization with a normal distribution resulted in the weights blowing up to 104 in magnitude. The Xavier initialization was found to produce a more stable network.

Another sanity check, training on 5 examples, was run and the network could overfit all the 5 examples, which is shown in Fig. 12.

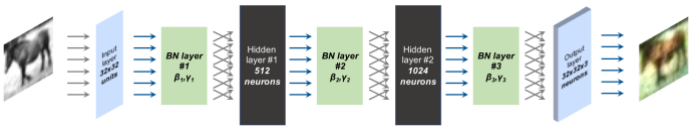

Fig. 13. Final structure of our multi-layer perceptron model.

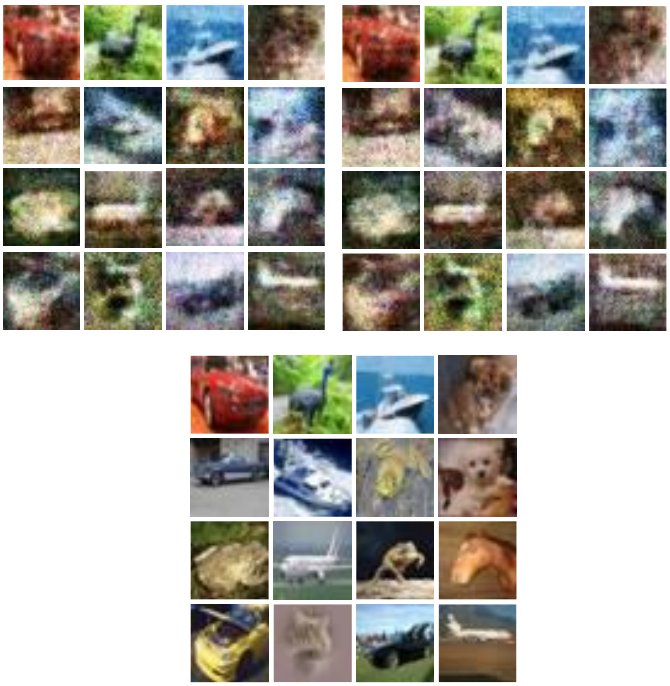

Fig. 14: Results of training MLP on 1000 examples and testing. The first row contains 3 images from the training set and the next 3 rows contain test images. The outputs of MLP without regularization (upper left), with regularization (upper right) and the ground truth (bottom) are shown.

The model was then trained on 1000 examples of the CIFAR-10 dataset over 500 epochs. The CIFAR-10 dataset was chosen because of the small image size (32×32) to reduce the training time. Also, the training set was intentionally chosen to be small since the MLP model has a large number of weights which makes training (backpropagation, especially) very computationally expensive. We tried a network with 1 hidden layer and one with two hidden layers, shown in Fig. 13. The results for the one-layer network is shown in Fig. 14. As can be seen, the MLP overfits the training set. This might be because the MLP model is over-parameterized since the hidden layers are fully connected layers. Also, the model does not capture the semantic information in the image as can be seen from the blurring of the edges and colors.

We added L2-regularization to mitigate the overfitting (Fig. 14), but it does not improve the test-error significantly as shown in Table 1. The two-layer network performed poorly on the test set and gave a larger test-error than the one-layer network. This might be because of the larger number of parameters of the two-layer network, which exacerbates the problem of over-fitting.

C. Auto-colorization Using CNN

The model parameters are updated by minimizing our loss objective function. It should be pointed out that for loss quantification, there are two different attempts made as expressed below. First, the loss is defined as mean square error

(MSE) between the estimated pixel colors in a*b* space and their ground truth. Second, this MSE was linked with classification results in order to reflect semantic information in coloring images.

$$MSE\ loss: \|y^{color} - y^{truth}\|^2$$
$$Total\ combined\ loss: \|y^{color} - y^{truth}\|^2 + \alpha\left(y^{truth\ class} - \log\left(\sum_i \exp(y_i^{class})\right)\right)$$

Once trained on 10000 images over 30 epochs (minibatch size of 30), images were fed into our network, which turned out to perform well. Our model can reproduce training data as shown in the top row of Fig. 15. More importantly, it performs very well on test dataset. As can be seen in Fig. 15, CNN with the combined loss (right in Fig. 15) performs better than the one with MSE only (left in Fig. 15).

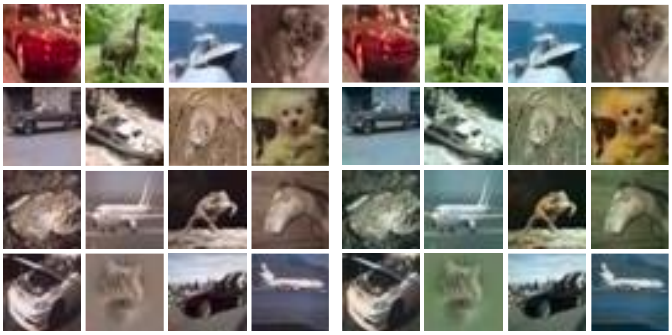

Fig. 15. Results of training CNN on 10000 examples and testing. The first row contains 3 images from the training set and the next 3 rows contain test images. The outputs of CNN with MSE loss only (left), with MSE and cross entropy combined (right), ground truth images are shown in Fig. 17 (bottom).

As shown in Fig. 15, adding semantic information helps colorize images more vividly with a rich variety of colors. In the total combined loss, by properly choosing α, the significance of global priors can be controlled on the outputs of colored images.

Another attempt that we made was combining CNN with semantic information extracted by using transfer learning. The pre-existing model, Resnet, was chosen. While the CNN was trained from scratch, the parameters of Resnet were fixed and used as a high-level feature extractor which provides information about image contents that can help their colorization. Since the model is compatible with 224×224 3-channel input, grayscale images were appropriately scaled and stacked to meet the requirements.

Fig. 16 shows that the performance wasn't as good as expected. This is because the model may have not learned its parameters enough due to time constraint and we speculate that Resnet may not work well with re-scaled images of 32×32. When the small amount of information was stretched out to fit into 224×224, the efficacy of the transfer learning may vanish.

Fig. 16. Transfer learning (Resnet) is integrated with CNN with MSE + cross entropy loss. The model may have not learned enough and Resnet may not work well due to small size dataset.

Finally, although the performance of auto-colorization is difficult to objectively quantify, average squared errors of the methods used in this study were computed and tabulated below for comparison. Among the methods that we attempted, CNN with the combined loss outperforms over the other methods.

Model	Avg. Squared Error
Linear regression for de-noising	0.12%
MLP without regularization	3.41%
MLP with regularization	3.30%
CNN with MSE loss	0.69%
CNN with combined loss of MSE + CE	0.61%
CNN with transfer learning integrated	0.72%

Table 1: Test errors for each method, calculated as the average RMSE over each pixel

IV. CONCLUSION

In this work, we studied about de-noising of old images using linear regression and auto-colorozing them using multilayer perceptron and convolutional neaural networks. We found that implementing noise detection algorithm is crucial to effectively reconstruct the original images and linear regression model performs surprisingly well with appropriate choice of threshold for the noise detection. Even with large amount of noise (64%) incorporated in an image, the model was able to recover the original image. We developed a multi-layer perceptron model as a baseline for colorization. The model was trained on CIFAR-10 dataset due to the huge computational costs involved in backpropagating through fully-connected layers. It was found that the model over-parameterizes the problem and hence overfits the training set. The model does not capture semantic information in the image and hence blurs the edges and imbues unnatural colours. L2-regularization does not mitigate the problem. CNN was implemented with architecture where local features are fused with semantic information and showed impressive performance with the combined loss of MSE and CE as an objective function.

V. FUTURE WORK

Once equipped with sufficiently high computation power, we would to implement our CNN auto-colorization model on a larger image dataset and observe how the results would change. We would also like to add image segmentation to our CNN model for better semantic understanding, using super pixel algorithm, and to improve the transfer learning CNN model.

VI. Contribution

Junkyo Suh: Data preparation and pre-processing algorithm, writing report, CNN implementation, debugging algorithms.

Koosha Nassiri Nazif: Linear regression algorithm with noise detection, debugging algorithms, writing report.

Aravindh Kumar: MLP for de-noising and auto-colorization, writing report, debugging algorithms.

VII. reference

[1] neuralnetworksanddeeplearning.com
[2] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International Conference on Machine Learning. 2015.
[3] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," ACM Transaction On Graphics (Proc. Of SIGGRAPH), 35(4): 110, 2016.
[4] Robertson, A.R.: The cie 1976 color-difference formulae. Color Research & Application, 2(1), 1977.
[5] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision, Springer, 2014.
[6] Ioffe and Szegedy, In 32nd International Conference on Machine Learning, Lille, France, 2015.

Download URL for our codes
https://drive.google.com/open?id=1P_lWvMka05hxtyE5hCSHkQYEV7OU3EW-

Library information
1. De-noising: This was done in MATLAB without using toolbox.
2. MLP: numpy, scipy, pickle, skimage
3. CNN: numpy, pytorch, skimage
4. Main references
Pytorch tutorials

- http://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- http://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html