Automatic Colorization using Auto-encoders

Colorization without Image compression

Naman Sood
18BCI0168
Vellore Institute of Technology, Vellore
naman.sood2018@vitstudent.ac.in

Shaik Haseeb Ur Rahman
18BCE0646
Vellore Institute of Technology, Vellore
shaikhaseeb.urrahman2018@vitstudent.ac.in

Lakshit Dua
18BCE0824
Vellore Institute of Technology, Vellore
lakshit.dua2018@vitstudent.ac.in

***Abstract*—This document is a model for a statistical-learning driven approach to approach Autocolorization through building an Encoder-decoder model with Convolutional neural networks. Learning Models: Keras, openCV, Numpy and Tensorflow. A direct function to convert grayscale into coloured images that can be coupled with various softwares or sensors.**

I. Introduction

A. Objective

Conversion of black and white images to colored images without human intervention using a unique convolutional neural network approach retaining the image properties and dimensions. Our secondary objectives involve improving the efficiency of the model through learning rate schedulers, different optimizers, dropouts and other techniques such as lasso and ridge regression.

B. Introduction & Motivation

In recent years, Convolutional neural networks have been emerging as one of the de-facto standards for solving various problems relating to image classification. It has come to limelight and has become popular because of the lower error rates (lesser than 4%) which it has achieved in ImageNet challenge. The main reason behind their success is due to their ability to learn and discern colors, shapes, and patterns within various images and associate them with object classes. This becomes one of the main reasons for such well-defined coloring carried out by CNNs as object classes, patterns, and shapes are generally correlated with color choice.

C. Background and previous research

Our project is mainly inspired in part by the CNN-based system of Ryan Dahl for automatically colorizing images. Dahl's system relies on several ImageNet-trained layers from mVGG16, which is integrated with an autoencoder like system with residual connections that merge intermediate outputs produced by the encoding portion of the network comprising the VGG16 layers with those produced by the latter decoding portion of the network. Since the connections link downstream network edges with upstream network edges, they purportedly allow for more rapid propagation of gradients through the system, which reduces training convergence time and enables training deeper networks more reliably. Indeed, Dahl reports much larger decreases in training loss on each training iteration with his most recent system compared with an earlier variant that did not utilize residual connections. Although Dahl's system performs extremely well in realistically colorizing foliage, skies, and skin, there are numerous cases in which the images generated by the system are predominantly sepia-toned and muted in color. It is due the use of a regression approach by Dahl. Although regression does seem to be well-suited to the task due to the continuous nature of color spaces, a classification-based approach using CNNs must work better.

II. Literature Review

There has been a great deal of progress in colorization of grayscale pictures without human mediation [20, 19, 18]. This issue isn't just fascinating from an aesthetic and materialistic perspective however it additionally has critical applications in significant exercises, for example, Video Restoration, Image Enhancement for better interpretability, colorization of representations done by craftsmen and artists and so on. The cycle can likewise be considered as a basic prehandling task for highlight learning in a self-administered mode, carrying on as a cross-channel encoder [20]. As of not long ago, colorization has consistently been a semi-mechanized cycle that depends on indications and highlights from the client. Levin et al. [14], for example, suggested that neighboring pixels in space with comparable pixel weights ought to have comparable hues. Through his work, these hints/indications are spread out in the types of harsh and incorrect scrawls on a grayscale picture. The algorithm can recognize these indications and create excellent colorization dependent on it. A few others have improved this algorithm further, including Huang et al. [13] (by tending to shading and draining issues) and Qu et al. [12] (by changing the cost function to account for the continuity of similar colors over matching textures in addition to similar intensities). Welsh et al. [11], then again, utilized a full shading model of comparative structure, in this manner decreasing human reliance significantly more.

One of the pioneers to propose a convolutional neural system which was trained for characterization of pictures to create full shading channels for the pictures provided as input was Dahl [17]. Fitting on the ImageNet information base with a L2 regularization loss applied on the chrominance esteems and resulting pixels, his methodology brought about normal

to below average yields and predicted results. The anticipated hues were consistently sensible and lied in the same root shade as the given picture but they were consistently desaturated, had muted colours or were tinted, an after-effect of the "averaging impact" of the L2 regularization. Later work in this space has moved toward this issue in a myriad of ways, the most well-known strategies comprising of regression onto a constant shading space [18, 17, 10] or order of quantized color esteems through classification [9]. There has additionally been progressions in present regularization techniques and objective functions to fold over and accurately order loss values for a multi-modular issue such as this. Hwang et al. [20] made an objective function exceptionally custom fitted for the issue of picture colorization. The methodology predicts an appropriation of potential hues for each given pixel and re-gauges the given misfortunes losses at the time of fitting to represent/accentuate on the uncommon hues (class weighting for rebalancing). Larsson et al. [8] and Iizuka et al. [7] have created comparable frameworks with negligible changes in the neural system architecture, techniques and models. While Hwang et al. [20] utilized a class weight classification model metric with rebalancing uncommon classes (pixels relating to uncommon hues), Larsson et al. utilized an un-rebalanced classification function, and Iizuka et al. utilized a regression objective function for the equivalent. Larsson et al. used hyper-sections [6] on a VGG architecture [5], Iizuka et al. utilize a two-stream design in which intertwined global and neighborhood weights and features.

GANs fill in as a great method to gain proficiency with the mappings between the sources of info and predictions just as the comparing loss and regularized values. It permits the model to be utilized in a more adaptable way for a wide assortment of circumstances. There has been a lot of analysts who have utilized GANs in a conditional setting too, polishing them on discrete labels [4] and text [3]. Isola et al. [2] expanded the utilization of GAN for picture to picture interpretation over a few areas, for example, blending photographs from name maps, recreating objects from edge maps, and colorizing pictures. All the more as of late, cycle GANs are being utilized for colorization [1] which diminishes the requirement for input-output sets. There has additionally been progress of picture colorization for craftsmen and comic essayists through the assistance of DRNNs (Deep residual neural systems) [22] as well as the blend of Vanilla neural systems with Adaptive picture clustering predictions [21]

III. Ease of Use

A. Abbreviations and Acronyms

- CNN - Convolutional Neural Network
- ReLU - Rectified Linear Unit
- CV - Computer Vision
- GPU - Graphics Processing Unit
- RGB - Red, Green and Blue color model
- IID - Independent and Identically Distributed

IV. Preliminaries

Google Collaboratory is the present working environment along with Python3. Collaboratory incorporates GPU support accelarating the fitting process of our Final Convolutional Neural Network.

A. Python3 Libraries

The present libraries incorporated in the project to obtain necessary outputs:

- **NumPy** - General Purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with arrays. It's the fundamental package for scientific computing with Python.
- **Matplotlib** - Plotting library for Python and its numerical mathematical extension: Numpy.
- **OS** - Provides functions to interact with the operating system in python. It provides a portable way of using OS dependent functionality.
- **cv2/OpenCV** - ItPython bindings developed to solve computer vision problems. All openCV arrays are structures converted to and from Numpy arrays. It allows easier integration with libraries such as SciPy and Matplotlib.
- **Keras with Tensorflow as Backend** - Keras is an open-source neural-network library written in Python. It can run on top of Tensorflow, Microsoft Cognitive Toolkit, Theano or PlaidML. It is designed to enable fast experimentation with Deep neural network while being friendly, modular and extensible.

B. Model Classes

- **Layers** - It refers to the neural network layers used by Keras. Our model uses the following layers
  - **Dense** - Builds layers such that each node of an input layer is connected to the output layer. It works as a vanilla neural network layer.
  - **Conv2D** - The layer that undergoes the convolution operation through several filters.
  - **Flatten** - Converts the 2D matrix into 1D Matrix. Reshaping matrix layers of the neural network.
- **Model** - The object that wraps around the neural network structure.
- **Model Checkpoint** - Saves weights+bias values per epoch to prevent data loss due to errors or failures

C. Dataset

The dataset used for training, validation and testing is the **CIFAR-10 dataset** which is one of the most diverse and data myraid datasets to use for image training of CNNs. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between

them, the training batches contain exactly 5000 images from each class. The images require pre-processing to convert them to black and white for training

Fig. 1. Classes with sample images - CIFAR-10 dataset

V. Proposed Methodology

A. Data Pre-processing

1) ***Coloured to Black and White Conversion****:* Since there is no proper image dataset for this task of automatic colorization, we will be using the CIFAR-10 image dataset wherein we will convert each and every image of the dataset into black and white and correspondingly through the help of openCV computer vision library and its pre-processing functions.

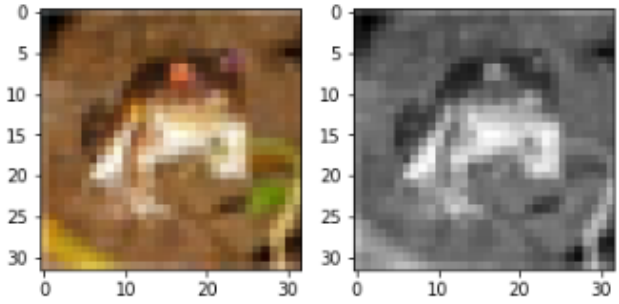

Fig. 2. Sample conversion from Coloured to Black and White

2) ***Pixel Normalization****:* An 8 Bit image is represented by 255 shades of colors. Therefore, it might happen that during the Training of the images an image having high values of the pixel is given more importance than an image having lower pixels values although the latter image being more important than the former image. Therefore, in order to prevent unwanted preference as well as improve the overall accuracy quality and training time we normalize the image by dividing each pixel by a value of 255. This will result in a model with value range of [0,1] which will boost our overall model performance and efficiency.

B. Objective Function

As brought up in [20, 7, 8], one of the most significant difficulties in auto-colorization is a cost function that represents the multi-modal idea of the issue. To explore this further, loss functions were tested.

The given equation represents the Smooth L1 loss: Introduced by Girshick[28]

$$\frac{1}{n}\sum_{n}\begin{pmatrix} 0.5\times(x_i-y_i)^2 \text{ if } |x_i-y_i|<1 \\ |x_i-y_i|-0.5 \text{ otherwise} \end{pmatrix} \tag{1}$$

This loss function is known for being robust with outliers. Given our data, outliers is not an issue. We are more interested in the model capturing finer details, which becomes a bigger challenge with L1 loss. It penalizes outliers less and would therefore not capture intricate details and predict far-off from the ground truth value.

The given equation below represents the L2 loss or the mean squared error loss:

$$\frac{1}{n}\sum_{n}(x_i-y_i)^2 \tag{2}$$

This is our primary loss function and we would be using this as our means for the final prediction accuracy and validation loss functions. This loss is also known as the MSE (mean squared error) or the quadratic loss function. This loss metric is not robust to outliers and penalizes them hard enough to capture the finer details in the final predictions made by our auto-encoder model. We will also experiment with the cross_entropy loss function to compare the predicted image colorization, hues and saturation levels to cross-verify and examine the best loss metric for our current task of auto-colorization.

C. Activation Function

We use the rectified linear unit (ReLU)[29] as the non linearity that follow each of our convolutional layers. Mathematically, ReLU can be defined as follows:

$$f(x)=\max(0,x) \tag{3}$$

ReLU helps with Training convergence speedups and is also extremely simple to compute the function. One drawback is that the model could be updated in such a way that the gradients for back-propagation always result in zero which effectively ”kills” neurons and prevents it from training. We would be using neuron initialization strategies to counter it.

D. Learning Rate

The learning rate is scheduled with the plateau function. After a set number of epochs, if the model does not show any improvement in validation accuracy, it will divide the learning rate by a set value.

E. Optimizer

The Adam optimizer[26] is used here. Adam stands for Adaptive moment estimation combines the ideas of momentum optimization and RMSProp: just like momentum optimization, it keeps track of an exponentially decaying average of past gradients; and just like RMSProp, it keeps track of an exponentially decaying average of past squared gradients.

Here, Beta1 refers to the momentum decay hyperparameter while Beta2 refers to the scaling decay hyperparameter. The epsilon parameter is the smoothing terms which is usually initialized to 10^{-7}.

As Adam is an adaptive learning rate algorithm like AdaGrad and RMSProp. It requires less tuning of the learning rate hyperparameter (n). The default value n=0.001 works for most cases, making it easier to use than gradient descent.

$$1.\quad \mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1-\beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$2.\quad \mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$3.\quad \widehat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-\beta_1^{\,t}}$$

$$4.\quad \widehat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\beta_2^{\,t}}$$

$$5.\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\, \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon}$$

Fig. 3. The Adam algorithm for parameter tuning (Theta)[26]

F. Neural Network Architecture and Flow

Our approach consists of training a CNN auto-encoder model with Black-and-white images and allowing it to learn coloured features without human supervision. To be more specific, we will convert the given coloured images into single-channel Black and white color space images that can provide us a grayscale input to our model. We then pass this model to our auto-encoder and it passes through the encoder schema first. The encoder will apply the convolution operation multiple times and return a vector of embeddings. These embeddings would be then passed to our decoder model which will apply the transpose convolution operation and return an output of the same height and width but with 3 different channels – RGB. The image dimensions and other properties would be retained throughout.

The dimensions of the input, output and embeddings vector are as follows:

- Input Image size – 32×32 pixel, single channel black and white images
- Output Image size – 32×32 pixel, three channel image (RGB)
- Encoding generation – [256] embedding per image generated in a latent vector

VI. Algorithm - Architecture and Flow

The model consists of an auto-encoder model which is a horizontal stack of the encoder and decoder models. The black and white image would first flow through the encoder model to be aggregated in an embedding vector. The process is as follows:

1) The image is passed to the encoder model
   a) The image undergoes convolution by 64 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 16×16×64
   b) The image undergoes convolution by 128 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 8×8×128
   c) The image undergoes convolution by 256 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 4×4×256
   d) The resultant vector is spread out into a 1D vector (dimension – [256])
2) The embeddings vector (latent_vector) is passed to the decoder model to apply the transpose convolution operation:
   a) A dense neural network layer is applied onto the latent vector. It results in a 1D vector (dimension [4096])
   b) The vector is reshaped into a three dimensional vector – (4×4×256)
   c) The image undergoes convolution by 256 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 8×8×256
   d) The image undergoes convolution by 128 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 16×16×128
   e) The image undergoes convolution by 64 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 32×32×64
   f) The image undergoes convolution by 3 filters (dimension – 3×3) with a stride of 2, no padding and sigmoid activation. It results in dimensions – 32×32×3

The resulting image from the encoder model is displayed as the output image. We successfully retain the height and width of the image (32×32) throughout the colorization process.

The architecture of the neural network (Encoder model) for Step 1

TABLE I
ENCODER ARCHITECTURE

Layer Type	*Input Shape*	*Number of Parameters*
encoder_input	(None,32,32,1)	0
conv2d_1	(None,16,16,64)	640
dropout	(None,16,16,64)	0
conv2d_2	(None,8,8,128)	73856
dropout_1	(None,8,8,128)	0
conv2d_3	(None,4,4,256)	295168
dropout_2	(None,4,4,256)	0
flatten_1	(None, 4096)	0
latent_vector	(None,256)	1048832

The architecture of the neural network (Decoder model) for Step 2

TABLE II
DECODER ARCHITECTURE

Layer Type	*Input Shape*	*Number of Parameters*
decoder_input	(None,256)	0
dense_1	(None,4096)	1052672
reshape_1	(None,4,4,256)	0
conv2d_transpose_1	(None,8,8,256)	590080
dropout_3	(None,8,8,256)	0
conv2d_transpose_2	(None,16,16,128)	295040
dropout_4	(None,16,16,128)	0
conv2d_transpose_3	(None,32,32,64)	73792
dropout_5	(None,32,32,64)	0
decoder_output	(None,32,32,3)	1731

The final aggregated model (Autoencoder)

TABLE III
AUTO-ENCODER ARCHITECTURE

Layer Type	*Input Shape*	*Number of Parameters*
encoder_input	(None,256)	0
encoder_model	(None,256)	1418496
decoder_model	(None,32,32,3)	2013315

A. Batch Normalization

The technique consists of adding an operation in the model just before or after the activation function of each hidden layer. This operation simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer: one for scaling, the other for shifting. In other words, the operation lets the model learn the optimal scale and mean of each of the layer's inputs.

- (mu)B is the vector of input means, evaluated over the whole mini-batch B (it contains one mean per input).
- (sigma)B is the vector of input standard deviations, also evaluated over the whole mini-batch (it contains one standard deviation per input).
- mB is the number of instances in the mini-batch.
- x(i) is the vector of zero-centered and normalized inputs for instance i.
- Y is the output scale parameter vector for the layer (it contains one scale parameter per input).
- the cross-circle represents element-wise multiplication (each input is multiplied by its corresponding output scale parameter).
- Beta is the output shift (offset) parameter vector for the layer (it contains one offset parameter per input). Each input is offset by its corresponding shift parameter.
- Epsilon is a tiny number that avoids division by zero (typically 105). This is called a smoothing term.
- $z^{(i)}$ is the output of the BN operation. It is a rescaled and shifted version of the inputs.

$$1.\quad \boldsymbol{\mu}_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\mathbf{x}^{(i)}$$

$$2.\quad \boldsymbol{\sigma}_B^{\,2} = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}_B\right)^2$$

$$3.\quad \widehat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^{\,2} + \varepsilon}}$$

$$4.\quad \mathbf{z}^{(i)} = \boldsymbol{\gamma} \otimes \widehat{\mathbf{x}}^{(i)} + \boldsymbol{\beta}$$

Fig. 4. Batch Normalization Algorithm[30]

VII. CONSTRAINTS

The major constraints observed throughout the process of fitting and validation of the model:

- Large Dataset Required
- High GPU power needed
- Overfitting if the parameters are not normalized and regularization isn't included
- Requires all features in the given image to be considered for accurate results

VIII. OBSERVATIONS

We explored different regularization techniques and strategies for tuning our models to produce the most robust and vivid images that resemble the pre-processed images. Through exploring different optimizers such as RMSProp, Nadam, Momentum and SGD, the ADAM optimizer produced the best results (least loss).

- Through the use of Batch Normalization and Dropout, images were desaturated and pale in nature. The validation accuracy was 53-55% while there was a loss of about 0.077. This model was overall decent in terms of the metrics due to all pixels being normalized and near

the mean values resulting in low loss and higher accuracy. This isn't our final chosen model

- We explored Dropout only models and saw a bump-up in the contrast and saturation of images while a decrease in accuracy and loss. Our model was less accurate for generalized images (high variance) but the image tones produced were much more vivid. This isn't our final chosen model
- We explored a model without any form of regularization. This allowed the model to produce highly contrasting and saturated images with minimal loss (0.3-0.4) while achieving a decent validation accuracy of 50-52%. This model also had properties of overfitting but was able to produce the most realistic images out of the trained models. This was our final chosen model.

IX. Conclusion

Through the amalgamation of decoder and encoder models, we were successful in exploring the space of convolutional neural networks and auto-encoders. Our model successfully converted a single-channel black and white image to its corresponding RGB layered colorized form automatically without any modifications to its inherent physical properties.

We were able to understand the importance of regularization techniques and also why they were more harmful towards our model rather that providing better results. This was due to their normalizing and mean-wrapped behaviour which resulted in muddy and desaturated images. Through a vanilla auto-encoder model, we were able to produce vivid images with low loss (0.3-0.4) and high accuracy (50-52%) that most closely resembled the original images before pre-processing.

References

[1] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycleconsistent adversarial networks." arXiv preprint arXiv:1703.10593 (2017).

[2] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint arXiv:1611.07004 (2016).

[3] Reed, Scott, et al. "Generative adversarial text to image synthesis." 33rd International Conference on Machine Learning. Vol. 3. 2016.

[4] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014).

[6] Hariharan, Bharath, et al. "Hypercolumns for object segmentation and fine-grained localization." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015

[7] Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification." ACM Transactions on Graphics (TOG) 35.4 (2016): 110.

[8] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning representations for automatic colorization." European Conference on Computer Vision. Springer International Publishing, 2016.

[9] Charpiat, Guillaume, Matthias Hofmann, and Bernhard Schölkopf. "Automatic image colorization via multimodal predictions." Computer Vision– ECCV 2008 (2008): 126-139.

[10] Deshpande, Aditya, Jason Rock, and David Forsyth. "Learning large-scale automatic image colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[11] Welsh, Tomihisa, Michael Ashikhmin, and Klaus Mueller. "Transferring color to greyscale images." ACM Transactions on Graphics (TOG). Vol. 21. No. 3. ACM, 2002.

[12] Qu, Yingge, Tien-Tsin Wong, and Pheng-Ann Heng. "Manga colorization." ACM Transactions on Graphics (TOG). Vol. 25. No. 3. ACM, 2006.

[13] Huang, Yi-Chin, et al. "An adaptive edge detection based colorization algorithm and its applications." Proceedings of the 13th annual ACM international conference on Multimedia. ACM, 2005.

[14] Levin, Anat, Dani Lischinski, and Yair Weiss. "Colorization using optimization." ACM Transactions on Graphics (ToG). Vol. 23. No. 3. ACM, 2004.

[15] Li, Jiwei, et al. "Adversarial learning for neural dialogue generation." arXiv preprint arXiv:1701.06547 (2017).

[16] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.

[17] Dahl, Ryan. "Automatic colorization." (2016).

[18] Cheng, Zezhou, Qingxiong Yang, and Bin Sheng. "Deep colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[19] Liang, Xiangguo, et al. "Deep patch-wise colorization model for grayscale images." SIGGRAPH ASIA 2016 Technical Briefs. ACM, 2016.

[20] Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European Conference on Computer Vision. Springer International Publishing, 2016.

[21] Deep Colorization Zezhou Cheng, Student Member, IEEE, Qingxiong Yang, Member, IEEE, Bin Sheng, Member, IEEE, arXiv:1605.00075v1

[22] Interactive Anime Sketch Colorization with Style Consistency via a Deep Residual Neural Network Ru-Ting Ye,Wei-Li Wang,Ju-Chin Chen Kawuu W. Lin

[23] Learning-Based Colorization of Grayscale Aerial Images Using Random Forest Regression Dae Kyo Seo, Yong Hyun Kim , Yang Dam Eo and Wan Yong Park

[24] Automatic Image Colorization Via Multimodal Predictions Guillaume Charpiat, Matthias Hofmann, and Bernhard Sch¨olkopf automatic image colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[25] Semi-Auto Sketch Colorization Based on Conditional Generative Adversarial Networks - Zijuan Cheng Fang Meng Jingbo Mao

[26] Kingma, Diederik Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations." ACM Transactions on Graphics (TOG). Vol. 25. No. 3. ACM, 2006.

[27] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1

[28] Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[29] Agarap, Abien Fred. (2018). Deep Learning using Rectified Linear Units (ReLU).

[30] Ioffe, Sergey Szegedy, Christian. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.