Colorize Black and White Images

Rajeev Meda
rmeda@umail.iu.edu

***Abstract*—For this project I investigate convolutional neural network based systems to faithfully colorize black-and-white images without any human intervention.**

***Keywords*—Convolutional neural networks (CNN); Residual connections; Autoencoders; Computer Vision;**

I. INTRODUCTION

Automated colorization of black and white images has been subject to research within the computer vision and machine learning communities. It has broad practical applications ranging from video restoration to image enhancement for improved interpretability.

Features based on convolutional networks (CNNs) have now led to the best results on a range of vision tasks such as image classification, object segmentation and detection, action classification etc. So I choose CNNs for this task. I design and build a CNN that accepts a black-and-white image as an input and generates a colorized version of the image as its output. Figure 1 shows an example of such a pair of input and output images. The system generates its output based solely on images it has learned from in the past, with no further human intervention.

One of the main reasons for the success of CNNs in recognition tasks is their ability to learn and discern colors, patterns, and shapes within images and associate them with object classes. These characteristics naturally lend themselves well to colorizing images since object classes, patterns, and shapes generally correlate with color choice.

II. RELATED WORK

Typically, recognition algorithms use the output of the last layer of the CNN. This makes sense when the task is assigning category labels to images or bounding boxes: the last layer is the most sensitive to category-level semantic information and the most invariant to nuisance variables such as pose, illumination, articulation, precise location and so on. However for fine-grained tasks such as segmenting, colorizing black-and-white images, estimating its pose etc. these nuisance variables are important. For such applications, the top layer is thus not the optimal representation.

Hypercolumns idea is introduced in [1]. A hypercolumn for a pixel in the input image is a vector of all the activations above that pixel. They hypostasize that is that the information of interest is distributed overall levels of the CNN and should be exploited in this way. In that regard the layers of a convolutional network can be thought of as a non-linear counterpart of the image pyramids used in optical flow and other vision tasks. By extracting the hypercolumns for an input image we can have lots of information about what's in the image. Using that information color information can be inferred. This is my initial idea.

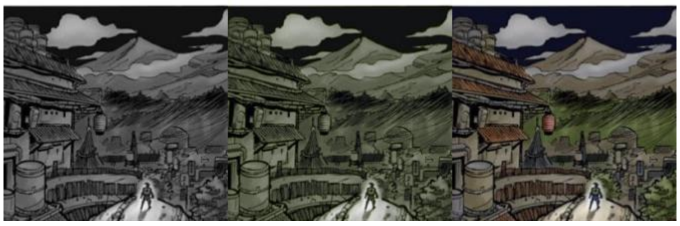


Figure 1: Sample input image left, output color image center and true color image right.

Later I explored the CNN based system for automatically colorizing images proposed by Ryan Dahl [2]. Dahl's system relies on several ImageNet-trained layers from VGG16 [3]. The VGG16 layers are integrated with 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. The residual connections are inspired by those existing in the ResNet system built by He et al that won the 2015 ImageNet challenge [4]. 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.

III. APPROCH

3.1 General Pipeline

During training time, the system reads images of pixel dimension 224 x 224 and 3 channels corresponding to red, green, and blue in the RGB color space. The images are converted to YUV color space. The Y component determines the brightness of the color (referred to as luminance or luma), while the U and V components determine the color itself (the chroma).

Then the image is converted to greyscale with 224 x 224 and 1 channel. This 1 channel is concatenated 3 times and sent into network as 224 x 224 x 224 tensor. The network returns U and V channels. An appropriate loss function is used to compute lose comparing these U and V channels with true U and V values for the image.

During the test time the model accepts a greyscale image of 224 x 224 x 1 dimension and is given as input. The network generates a color image as output, as well as a summary image as shown in Figure 1.

3.2 Transfer learning

I initialized parts of model with a VGG16 instance that has been pretrained on the ImageNet dataset. Since image subject matter often implies color palette, the network that has demonstrated prowess in discriminating amongst the many classes present in the ImageNet dataset would serve well as the basis for our network. This motivates the decision to apply transfer learning in this manner.

3.3. Activation function

The rectified linear unit (ReLU) has been empirically shown to greatly accelerate training convergence [5]. Moreover, it is much simpler to compute than many other conventional activation functions. For these reasons, the rectified linear unit has become standard for convolutional neural networks.

There is one downside of using the ReLU. They can be fragile during training and can die. the activation function in a neural network. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold.

To compensate this I use leaky ReLUs as the nonlinearity that follows each of the conv olutional and dense layers. I use a small negative slope of 0.01 as an attempt to attempt to fix the dying ReLU problem.

3.4 Weight Initialization

It is common to initialize the weights of the neurons to small numbers and refer to doing so as symmetry breaking. The idea is that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network. I initialize the weights using truncated normal function with standard deviation of 0.01 following the Xavier Initialization scheme proposed by [8].

3.5. Batch normalization

Ioffe et al introduced batch normalization as a means of dramatically reducing training convergence time and improving accuracy [6]. In the network I used batch normalization layer before every non linearity layer.

3.6 Hypercolumn model

This model is inspired from the Hypercolumns idea is introduced in [1]. Figure 2 shows the structure of the model.

In this model an image is passed through the VGG network and the layers before the max pooling layers (layers 4_3, 3_3, 2_2 and 2_1) are extracted. Each of these layers is resized to 224x224 using bilinear interpolation. This will give 963 hyper columns.

These 963 columns are then passed through the network as shown in figure 2 to finally output 224 x 224 x 2 output consisting U and V channels. These outputs are compared with the true U and V values of the image using an appropriate loss function and the network is trained to minimize this loss.

Intuitively this model is somewhat similar to the spatial pyramid model [7] where the image is divided into subblocks and features are extracted in each block. Here the network learns the distribution of image color representation at different scales in each of the neural network layer.

This model takes up a lot of memory as the network containing 993 hypercolumns does not shrink in size. I was not even able to run this model due to the lack of computational resources. Hence I turn towards a more efficient model explained in next section.

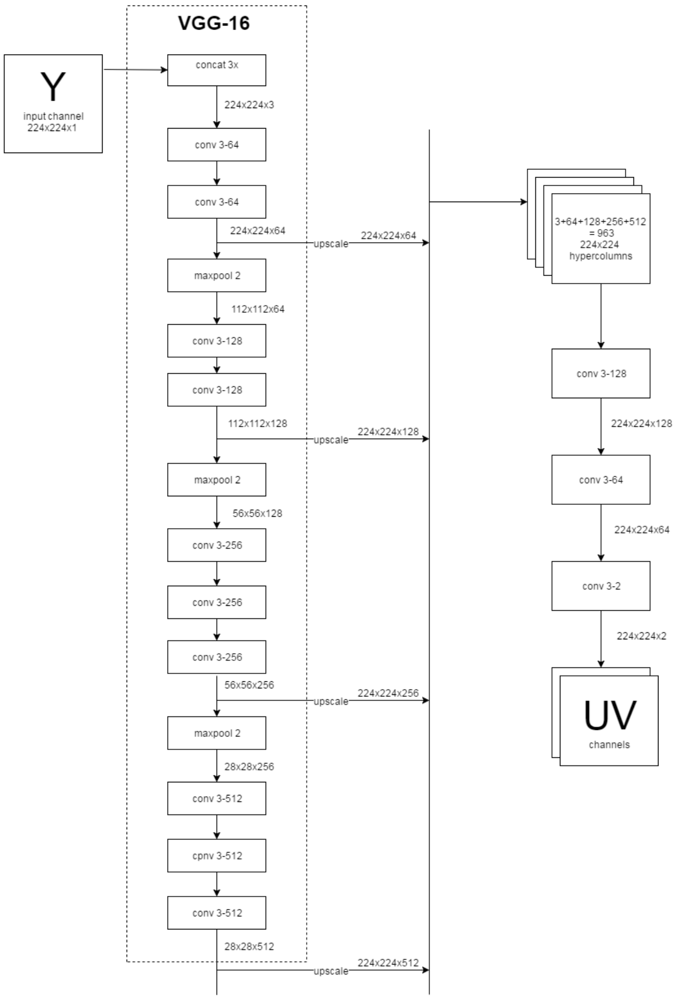


Figure 2: Hypercolumn inspired model

3.7 Residual encoding model

I use the model described in [2] with some changes. This model consists of a summarizing, encoding process on the left side followed by a creating, decoding process on the right side. Figure 3 shows the structure of this model.

The architecture of the leftmost column of layers is inherited from a portion of the VGG16 network. During this summarizing process, the size (height and width) of the feature map shrinks while the depth increases. As the model forwards its input deeper into the network, it learns a rich collection of higher-order abstract features.

The creating process on the right column is a slightly modified version of the residual encoder structure described in [2]. Here, the network successively upscales the preceding layer output, merges the result with an intermediate output from the VGG16 layer, and performs a two-dimensional convolution on the result. The progressive, decoder-like upscaling of layers from an encoded representation of the input allows for the propagation of global spatial features to more-local image regions. This trick enables the network to realize the more abstract concepts with the knowledge of the more concrete features.

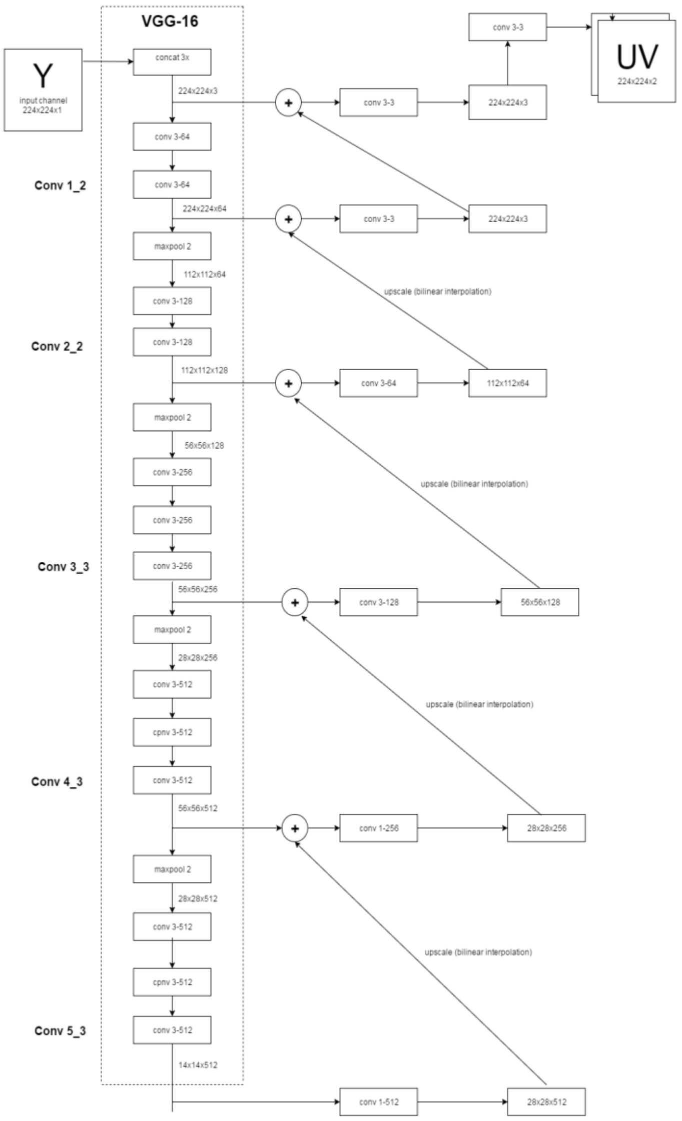


Figure 3: Residual encoding model

3.8 Objective Function

For the objective function I choose the most obvious Euclidean distance function between the network output U and V channels and the true color U and V channels.

However, I believe that more advanced loss functions would converge the network faster.

3.9 Optimizer

I use gradient descent optimizer with learning rate of 0.0001.

IV. Dataset

This model can be effectively used to train any images to colorize. I choose to colorize comics partly because they have simplistic color patterns and the network should be able to learn the color patterns pretty fast.

I choose 2685 images from Naruto comic. These images were colored manually. These images were in png format, which are about 1.2 MB per image. To reduce the image size I converted them into jpg format which are about 250 KB.

While training the images instead of resizing the images to 224 x 224 x 3 I choose to randomly take a crop the image which is of size 224 x 224 x3. I choose to crop them instead of resizing them because the images have high resolution and the images are taller than they are wider.

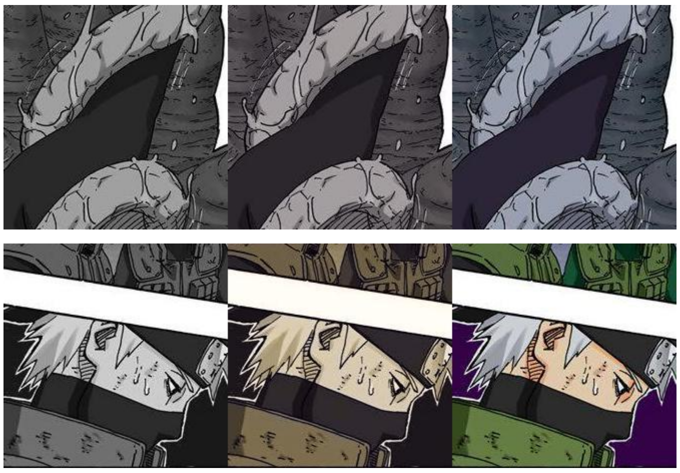
V. Experiments and Results

The network is implemented using tensorflow and was trained on Amazon g2.x2.large instances with NVIDIA GRID K520 GPUs.

I started by trying to overfit the model with 256 images to determine various hyper parameters. After few trails I choose image learning rate of 0.0001.

Due to the limited computational resources I used 1 image per batch and only ran 30,000 iterations. It took about 15 hours to run.

Figure 5 shows are some of the results I with this model after about 30,000 iterations.



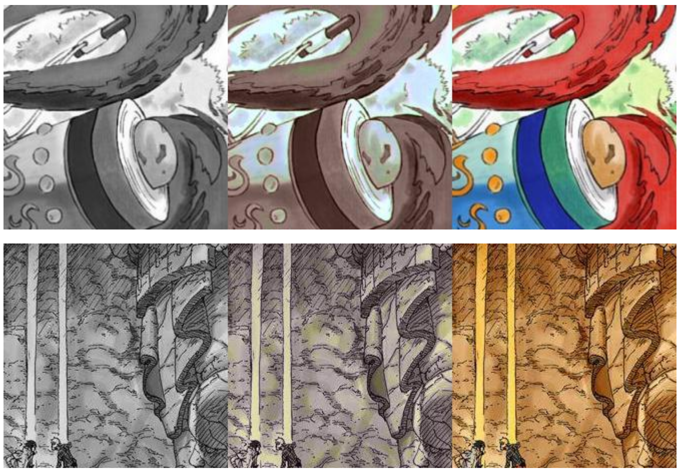


Figure 5: Example results

The major hurdle faced with this model is averaging problem. For example, consider a car, it have many colors, this network will take average of all those colors and give the same color to all the cars. This problem is caused because of using the regression based model.

This problem is especially evident in the dataset I choose. The comics have simplistic color representations and often times the same structures have different colors. For example rocks have brown or red or black color, so the network averages those and gives a dull color as shown in the 4th result in figure 5.

VI. Discussion and Conclussion

In conclusion I felt that I choose the wrong dataset for the model I am working with. Due to the averaging problem I was not able to get the outputs with vibrant colors.

To compensate this problem the final regression layer can be replaced with a classification layer. The UV color channels color space can be discretized into 64 equi-width bins using binning function. The function returns an array of the same shape as the original image with each U and V value mapped to some value in the interval [0, 63]. Instead of directly predicting the numeric values U and V channels the network can be trained the most probable bin numbers for the pixels, one for each channel. For loss function, the sum of cross-entropy loss on the two channels as minimization objective. This kind of classification network would improve the averaging problem.

From this project I learnt the many issues that rise out of training a neural network. Due to the lack of time and resources I was not able to implement the classification model. However my work lays a solid foundation for future work that can be done on this problem.

References

[1] B. Hariharan, P. Arbel´aez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 447–456, 2015.

[2] R. Dahl. Automatic colorization. http://tinyclouds.org/colorize, 2016

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

[7] Y. Jia, C. Huang, and T. Darrell, Beyond Spatial Pyramids: Receptive Field Learning for Pooled Image Features. CVPR 2012.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In International conference on artificial intelligence and statistics, pages 249–256, 2010.