Deep Image Colorization

Fadhil Mochammad
fadhilmch@kth.se

Muhammad Irfan Handarbeni
handa@kth.se

Sri Datta Budaraju
budaraju@kth.se

Abstract

With the motivation is explore applications of Deep Learning in Computer Vision, we have taken up the task of image colorization. In this project we try to build a Deep Neural Net to colorize the pixels in a gray-scale image. We came up with an approach that explores various concepts such as Convolutional Neural Networks, Autoencoders, Pre-trained Networks and also Generative Adversarial Networks. Our approach is the combination of the ideas from the papers let-there-be-color(1), deep-koalarization (2) which focus on Image Colorization using an Autoencoder architecture with the assistance of a classification network.

1 Introduction

Along with millions of colorful images there are millions of images that are simply black and white. These images are either the precious old images that are carefully stored to preserve history or they could also be the images from camera feed recorded in gray scale to save storage. Either way our aim is to *color 'em all!* and make the them beautiful. Colorizing these gray-scale images help bring those historic images back to life and also fetch important information from the security camera feeds. Right now these tasks are done by professionals who invest a lot of time in researching the history of the images like the estimated time period of the image, the place, their culture and climate and what materials were used and available then to restore antique images. This is undoubtedly a very challenging and requires lot of time and skill. With the help of Deep Neural Networks that are data-driven we could automate this process to some extent. Though gray-scale images miss capturing a lot of information and properties of the entities, we could use the millions of images that covers a wide range of scenarios and entities to enrich the gray-scale images. Thanks to the amazing computer vision research community, we have humongous collections like the Imagenet (6) which is exactly what we want for our purpose. Learning from these images is a very challenging, resource and time consuming task. We try to leverage the power of convolutions using an Autoncoder network along with support model, InceptionResNetv2 (8) that is pre-trained on Imagenet. We have experimented training our model on Tiny-ImageNet and different volumes of Imagenet and have tested them on some interesting gray-scale images.

2 Related Work

There are many papers that discusses various Deep learning architectures and techniques to solve this problem. In this section we talk about the 4 interesting papers that caught our attention. Each of these paper adds a little more twist to the previous one.

2.1 Autoencoder

The most popular architecture for image to image generation tasks like segmentation, colorization is the Autoencoder. The Autoencoder is the combination of the encoder and the decoder. The encoder takes in the input image and performs layers of convolutions to extract low level features of the image. The decoder then takes these low level features and again performs layers of de-convolutions or more

formally transpose convolutions to produce another higher level features i.e an another image. The suggest paper in the project ideas, Colorful Image Colorization by Zhang, R et al. (3) uses such an Autoencoder that learns to extract the most relevant features and use them produce a colorized image. This approach takes an gray-scale image as input and outputs 2 values for each pixel that define the color. These values correspond to the a,b channels in LAB color space. These values are then combined with the input image whose pixels correspond to the L channel thus giving a final output of 3 channel color image. The model's goal is to minimize the loss, which is the euclidean error between the predicted color and ground truth. As the task of image colorization is multimodal i.e a particular object in reality could be in different colors, the model predicts a distribution of colors for each pixel rather than just one. The authors devised a weighted-loss tailored to the task of colorization to emphasis rare colors. The final color is the annealed mean of the predicted distribution.

2.2 Autoencoder and simultaneous classification

Unlike the previous approach where the entire burden of extracting the features is given to the encoder, the paper Let there be color! by Iizuka et al. (1) proposes to use a helper classification architecture who shares the weights of the encoder but also learn deeper/global features. While the encoder learns the mid-level features of the image that has the knowledge of the local features of such as textures or objects in a smaller parts of the images, the classification network learns global features that has image level information like the environmental settings like outdoors or indoors, day or night etc. Combining these local and global features help to gain semantic knowledge of the image as a whole.

2.3 Autoencoder and Pre-trained Classification network

This paper Deep Koalarization by Baldassarre, F. et al. (2)is conceptually the replication of the previous paper (1) but leverages the use of pre-trained models that are trained on huge colored images from Imagenet (6). In this approach, as the weights are not shared between the encoder and the classifier there is more potential for the encoder to learn more local features with the same number of parameters. Additionally as the network uses the pre-trained model, the training time is drastically reduced.

2.4 Autoencoder and Generative Adversarial Networks

Deoldify by Jason Antic (4) which is also suggested in the project ideas is most popular and the most realistic colorization approach so far. This approach makes use of Self-Attention Generative Adversarial Network as proposed by Han Zhang et al. (5) but with a generator, Resnet34 (7) pre-trained on Imagenet (6). Though we get pretty good results using just an Autoencoder, there are many inconsistencies in the colorization say, blue apple, orange skinned people or some random glitches or abnormal patches. GANs help to solve this problem by making the Autoencoder predict colors in a way that a discriminator/critic network is unable to distinguish it from real image. Trying to minimize both the generator loss and the critic's loss, GANs take a lot of time to train and get descent results. The most interesting part of Deoldify is its GAN training technique called NoGANS. This method first trains the generator separately based on the feature loss. Followed by training the critic based on the trained generator and performing binary classification if it is a real image or not. Finally the separately trained generator and critic are trained in an actual GAN setting. By this method the author was able to complete the GAN training just after training over 1-3 percent of the Imagenet images after which the model reaches an inflection point after which the quality oscillates.

3 Approach

For our research in this project, we decided to replicate the approach and architecture in Deep-Koalarization proposed by Baldassarre et. al. (2). We came up with our own implementation of the architecture using Keras framework with Tensorflow backend.

Based on that approach, each image with size $H \times W$ in our dataset is represented as 3-D array in CIE $L^*a^*b^*$ color space. $L^*a^*b^*$ color space is used because it separates the luminance in L^* component and the color characteristics in a^*b^* components. From the luminance component $\mathbf{X}_L \in R^{H \times W \times 1}$, the model is to predict the remaining a^* and b^* components to generate prediction of fully-colored

image $\tilde{\mathbf{X}} \in R^{H \times W \times 3}$. Where the predicted colorized image is the combination of the original luminance component and the predicted a^* and b^* components, $\tilde{\mathbf{X}} = (\mathbf{X}_L, \tilde{\mathbf{X}}_a, \tilde{\mathbf{X}}_b)$.

The architecture for this approach is completely based on CNNs, which is widely used for a lot of computer vision research and projects. CNNs use convolutional layers which is a small filter that convolutes through the image to be able to learn the local patterns of the image. The first convolutinal layer would be able learn simple features such as small curve in the image and as the convolutional layer going deeper, the layers would be able to learn more complex features (9).

3.1 Architecture

Our model is based mostly on the architecture proposed in Deep-Koalarization paper with some minor modification. The model consumes grayscaled images and try to predict the a^* and b^* components. Then those predicted components are combined by the original luminance component to produce the final colorized images. We used the pre-trained Inception ResNet v2 for feature extraction and fuse the extracted feature from the last layer in the Inception network with grayscale training images. Figure 1 shows the overview of the model architecture.

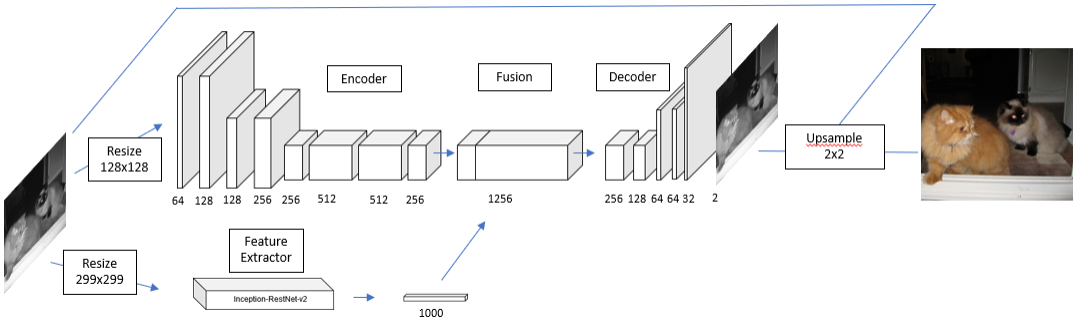

Figure 1: Overview of the model architecture

The network has four main components, which are encoder, decoder, feature extractor and fusion. Encoder component extracts the mid-level features of the images where the feature extractor component using Inception-RestNet-v2 extracts the high-level features. These features then are merged in the fusion component. This embedded features are then used by Decoder to predict the a^* and b^* components and produce the colorized images. We used the same network layers configuration as in Deep-Koalarization (2) with detail configuration in table 1, 2, and 3.

Layer	Kernel Size	Stride
conv	64 x (3 x 3)	2 x 2
conv	128 x (3 x 3)	1 x 1
conv	128 x (3 x 3)	2 x 2
conv	256 x (3 x 3)	1 x 1
conv	256 x (3 x 3)	2 x 2
conv	512 x (3 x 3)	1 x 1
conv	512 x (3 x 3)	1 x 1
conv	256 x (3 x 3)	1 x 1

Table 1: Encoder configuration

Layer	Kernel Size	Stride
conv	128 x (3 x 3)	1 x 1
upsample	-	-
conv	64 x (3 x 3)	1 x 1
conv	64 x (3 x 3)	1 x 1
upsample	-	-
conv	32 x (3 x 3)	1 x 1
conv	2 x (3 x 3)	1 x 1
upsample	-	-

Table 2: Decoder configuration

Layer	Kernel Size	Stride
fusion	-	-
upsample	256 x (1 x 1)	1 x 1

Table 3: Fusion configuration

The encoder consists of 8 convolutional layers with (3×3)-sized kernel that will generate mid-level features from the input images. This component receives grayscale image with size of $H \times W$ as

input and produce $H/8 \times W/8 \times 256$ feature representation. To reserve each layer's input size, padding is used on every layer. And to reduce the computational cost (10), we apply (2×2) strides on several layers.

The feature extractor components is used to extract high-level features of the images. High-level features contains information that can be useful for the colorization (2) We used pre-trained Inception-ResNet-v2 model in order to do that. To extract the high-level features, Inception network need to be feed with grayscale training images. Since Inception only accept specific size of image $(299 \times 299 \times 3)$, we need to resize the image into size if 299×299 and stack the grayscale image three times. The processed training images is then consumed by the Inception network to generate classification prediction, we then extract the output of the last layer before softmax to get $(1000 \times 1 \times 1)$ embedding.

The fusion component is where the network fuse the feature representation from the encoder and Inception network feature vector. The fusion component replicate the feature vector from Inception network $HW/8^2$ timees and fuse it with the feature vector from the encoder along the depth axis. This is based on the method proposed by Iizuka et. al. (1). This method produce a single feature vector with size of $H/8 \times W/8 \times 1256$. The last layer in the fusion component is a convolutional layer which consist of $256(1 \times 1)$ kernels which produce $H/8 \times W/8 \times 256$ feature vector.

Last component, the decoder component consumes the $H/8 \times W/8 \times 256$ feature vector from fusion component and applies some convolutional and up-sampling layers. This procedure is done in order to get the $H \times W \times 2$ final layer. This final layer is then up-sampled and combined with the original grayscale L^* component to produce the colorized image.

3.2 Training

All the networks were trained using the Adam optimizer with initial learning rate of 0.001. We use Mean Square Error between the actual color and the predicted a^* and b^* components to calculate the loss for the training. In order to reduce the training time, we resize all of the images before feed it to the network. Since our networks need to extract features from the pre-trained Inception ResNet v2 and embed it with the L^* component of the training images, it will cause bottleneck if it's done in the training phase. Hence, we did the image embedding beforehand and store the data in TFRecords format. The use of TFRecords also to reduce the training time.

Due to the complexity of the networks, we run all the experiments on GPU enabled virtual machine in Google Cloud Platforms. All the traninig were done using NVIDIA K80 GPU on the virtual machines. Training on GPU could reduce the training time, but a considerable amount of time is still needed to complete the training.

3.3 Data Collection

For our experiments in this research project, we rely on Imagenet dataset that is widely used for researches in computer vision (6). Imagenet has approximately 14 millions images with wide variety of classes. To reduce the computational cost and time, we decided to only use some subsets of the whole Imagenet dataset. We use three different Imagenet datasets for our experiments. The first one is the tiny Imagenet dataset which contain compressed version of the original Imagenet dataset. Tiny Imagenet dataset contains 100K color images of 200 classes. The second dataset we used is the first 12K images from Imagenet 2011 Fall Release. And we also used 5 classes from the Imagenet dataset, which are cat, obelisk, people, palm tree, and bullet train. For each class, we used 1.2K images. All the full-scaled Imagenet dataset were downloaded using python script with urllib library. The urls were obtained from the Imagenet website. Before send it to our network, all the images were resized to dimension of $128 \times 128 \times 3$. Examples of images can be seen in figure and.

4 Experiments & Results

As mentioned in the previous section, we have done several experiments using different dataset configurations on our model. We train our model on virtual machines in Google Cloud Platform with NVidia K80 GPU. During training, we save the model when there is an improvement in score. We also reduce the learning rate by a factor of 0.1 if no improvements for consecutive few epochs. In

addition, we use early stopping method if we are not seeing any improvement at all for considerable amount of time. For the optimizer, we choose to use Adam Optimizer. Before doing the experiment, we want to make sure that the model that we built is correct. We train and test our model with the same image and see if it can recover the gray-scaled image back to the original one. We get that our model is able to recover the image perfectly and hence we decide to train our model in a bigger dataset.

4.1 Tiny Imagenet

On the first experiment, we try to train our model using the 300 images from tiny-imagenet. From the result on the 2, we can see that our model is able to recognize some of the features and able to give the right colors on the features. However, it is still not able to recognize the borders between the object. This might be happening due to the small number of training data and low-resolution image from the tiny-imagenet.

Figure 2: Result using 300 data from tiny-Imagenet as training set

4.2 Imagenet

From our hypotheses on the previous experiment, we decided to use higher resolution images dataset and the one the widely used for image colorization implementation, which is the imageNet dataset. First, we try to train using 1200 images that belong to one class or group of images in imageNet dataset. As you can see from the 3, this model is able to perform better, especially it can recognized the borders between object better. However, our model is not generalized better since we only train our model only for specific classes.

On the second run, we try to increase the training data to 2000 images and also combine the training data from different classes of images. We train it on our cloud computer for around 15 hours. We can see from the result on 4 that our model perform best on this larger number of training data and able to generalize better. The results show that our model performs quite good for some images, especially for the images that have 'dominant' features, usually natural elements, for example, grass and sky. Our model is not able to correctly color some of the specific feature or object such as clothes.

4.3 Imagenet with TF Records

On the previous experiment, we are able to create the model that is able to colorize image well. However, the presentation of good result compared to the bad result is really low. We think that we can improve the performance of our model by increasing the number of training data. On the other hand, increasing the number of training data will also increase the training time. It will take days

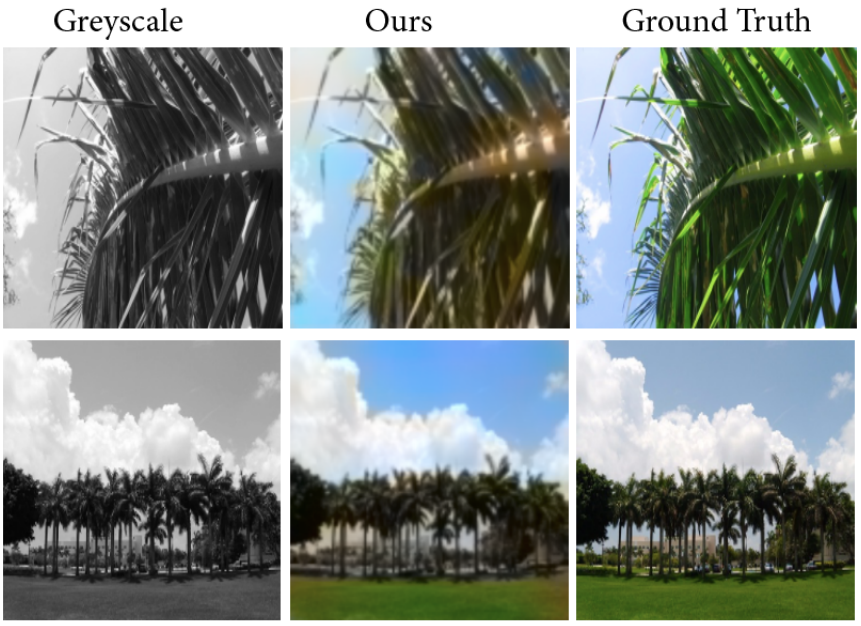

Figure 3: Result using 1200 data from only one class in Imagenet as training set

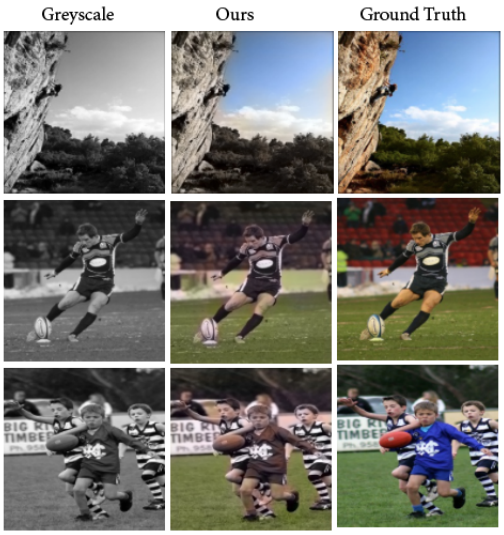

Figure 4: Result using 2000 data from Imagenet as training set

to train our model. Hence, we try to use another way to train our model by using the TF record. First, we need to resize and embed the features of all the training data and then save it on the TF record file. Then, when we want to train our model, we can generate the data from the TF record file. This method can significantly decrease the training time. It takes around 100 minutes to train 10.000 training data. 5 shows the result from this model. The result is not as we expected. We get a brownish image for all the test image. We presume that we did not implement the TF record method correctly. Thus, the model does not learn really well during the training.

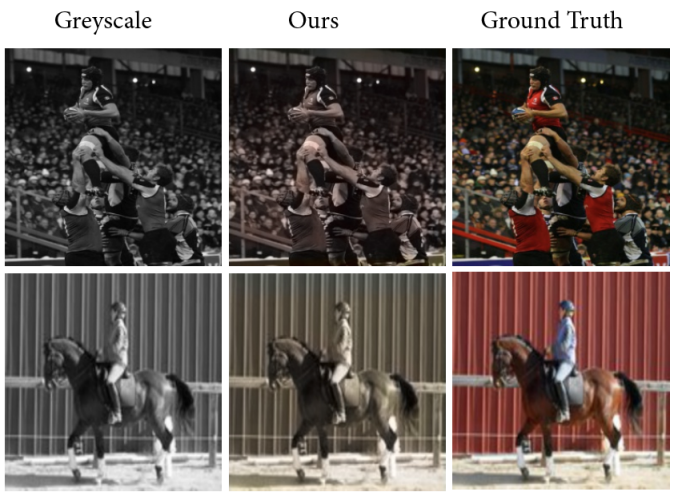

Figure 5: Result trained using 10.000 data from imagenet as training set and stored in TF Record

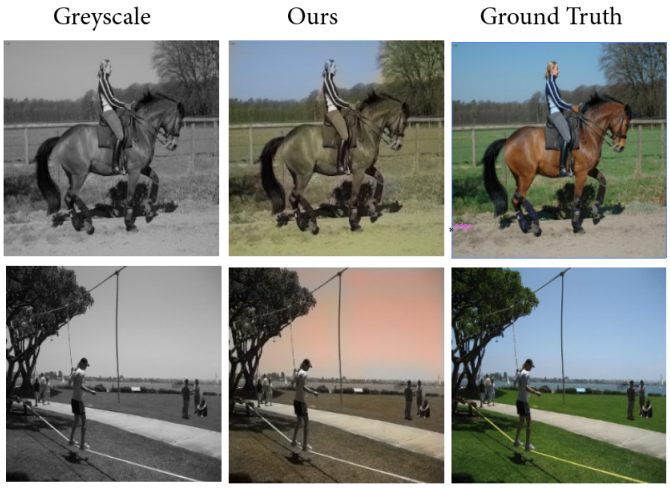

Figure 6: Miscolored result from trained model using 2000 training data from ImageNet

5 Conclusions & Future Work

Overall, get our hands dirty on implementing end-to-end deep learning project allow us to learn a lot of aspect in this subject, especially on its application in the field of computer vision. We successfully implemented the CNN-based approach of image colorization, and combine it with pre-trained Inception-RestNet-v2 to help the network to achieve better results. Unfortunately, as of the time we wrote this report, we're yet to get a model that can achieve excellent generalization performance. Our model managed to achieve decent estimation if we train it to overfit particular image classes. However, if we train our network with various image classes, it tends to produce brownish results. We also found that our network performs quite well on large object such as sky or trees, and performs poorly on smaller and more detail objects. We believe that results are due to the small size of our training dataset. By increasing the number of training data, the network might successfully achieve better generalization performance on image colorization. Due to resource and time limitation, we are not able to train our model using larger training set. The implementation TF record method is able to reduce the training time significantly and it will help to train our model more efficient when using larger dataset. However, we still not able to implement this method correctly and resulting in the model to not learn really well during the training.

As discussed in the related works section, there are a number of techniques to solve this task and we have taken the best and fairly easier methods and have obtain descent results as mentioned above. These results could be improved by training for longer periods on entire data. However this might not be able to overcome the inconsistencies and unnatural colors like the pink sky and the green horse in figure 6. One way this can be tackled is to introduce GANs by using the NoGAN method proposed in Deoldify (4). One other way to get better colors is to do layer-wise training as the current approach plateaus without giving desirable results.

References

[1] Iizuka, S., Simo-Serra, E. and Ishikawa, H. (2016). Let there be color!. ACM Transactions on Graphics, 35(4), pp.1-11.

[2] Baldassarre, F., Morín, D. and Rodés-Guirao, L. (2017). Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2. [online] arXiv.org. Available at: https://arxiv.org/abs/1712.03400 [Accessed 17 May 2019].

[3] Zhang, R., Isola, P. and Efros, A. (2016). Colorful Image Colorization. [online] arXiv.org. Available at: https://arxiv.org/abs/1603.08511 [Accessed 17 May 2019].

[4] GitHub. (2019). jantic/DeOldify. [online] Available at: https://github.com/jantic/DeOldify [Accessed 17 May 2019].

[5] Zhang, H., Goodfellow, I., Metaxas, D. and Odena, A. (2018). Self-Attention Generative Adversarial Networks. [online] arXiv.org. Available at: https://arxiv.org/abs/1805.08318 [Accessed 18 May 2019].

[6] Image-net.org. (2019). Imagenet. [online] Available at: http://www.image-net.org/ [Accessed 18 May 2019].

[7] He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition. [online] arXiv.org. Available at: https://arxiv.org/abs/1512.03385 [Accessed 18 May 2019].

[8] Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. [online] arXiv.org. Available at: https://arxiv.org/abs/1602.07261 [Accessed 18 May 2019].

[9] Zeiler, M.D., Fergus, R.(2014). Visualizing and understanding convolutional networks. In: European conference on computer vision, Springer 818-833.

[10] Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.A. (2014). Striving for simplicity: The all convolutional net. CoRR abs/1412.6806

A Learning Outcome

- Muhammad Irfan Handarbeni
  Convolutional neural network, autoencoder, fusing pre-trained network to help with training, data collection, data representation to decrease computational time.
- Fadhil Mochammad
  Implement deep learning method, mainly convolutional neural network for computer vision / image processing problem, know the work flow of creating deep learning model, Able to set up environment to develop deep learning model, Learn how to use pre-trained model to help the feature extraction process, know how to improve model performance,
- Sri Datta Budaraju
  First time ever implementing Autoencoders. Have used pre-trained networks before but it was new to fuse pre-trained network and self-made network. Learned a little more about GANS than I previously new. Though I have not finished the implementation by the time this report is written, I am working and learning about it.