HEURISTICS

IMAGE COLORIZATION

Abhishek Panigrahi 14CS10001

Avikalp Srivastava 14CS10008

Ishaan Sang 14CS10022

Kushagra Goel 14CS10030

Sahil Rishi 14CS10041

Siddhant Singh 14CS10045

Vaishal Shah 14CS10059

Arijit Panigrahy 14CS30005

Dishank Agarwal 14CS30008

**Problem Statement**

Given a grayscale photograph as input, we shall attack the problem of hallucinating of plausible color version of the photograph.

The problem we want to solve is a fully automatic approach to colorization devoid of any human interference.

**Methodology**

1. **Naive Approach**

At first, we started with a naive approach with the idea that each combination of Red, Green, Blue has a unique grayscale shade. So, in order to learn the combination of colors for a particular shade, we used a shallow 3-layer convolutional neural network with the following structure.

*CNN structure:*

No. of layers = 3

Layer 1 = 224 \* 224 \* 50 neurons

Layer 2 = 224 \* 224 \* 25 neurons

Layer 3 = 224 \* 224 \* 3 neurons

Filter size is always kept as 1 \* 1 and we have given a stride of 1 \* 1 to each layer.

*Input:*

224 \* 224 \* 1 Grayscale image

*Output:*

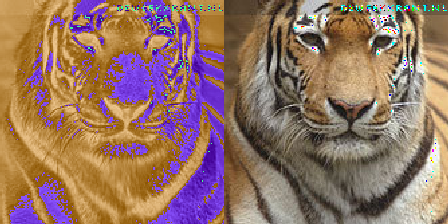
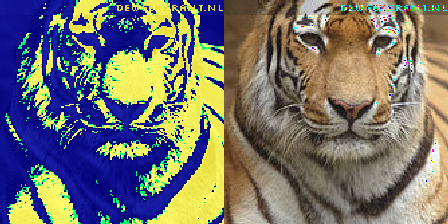
224 \* 224 \* 3 RGB image

*Output quality and explanation:*

The naive code fails to color black and white images with proper colors and rather produces a shaded image of the given input image with a prominent tint of a single color.

Reasons: Our CNN is made up of only 3 layers. Hence it's not able to learn properly each shade. Also we haven't taken care of the neighboring pixels of each pixel since we tried to learn from grayscale brightness. RGB value of [255,255,255] acts as a local optimum for such simple CNNs and so our loss function tends to converge to white image.

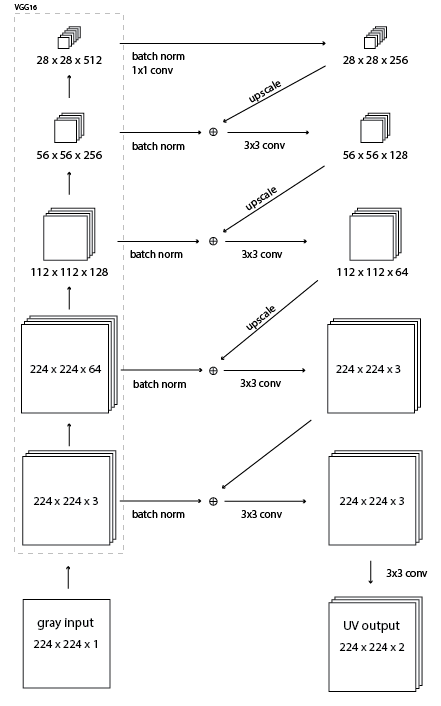
*Results:*



600 1200 2800



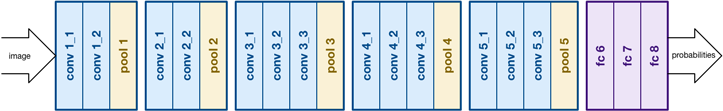
3600 Original



1. **Residual Encoder Approach**

The above model was used to utilize the intermediate layers of VGG-16 convolutional neural network and color a given black and white image.

The model is described below:



The intermediate layers of VGG-16 convolutional neural network are shown above. We pass a black and white image to a pre-trained VGG-16 convolutional neural network and take out the output of the following layers:

1. 224 x 224 x 64 tensor output of **conv1\_2** layer
2. 112 x 112 x 128 tensor output of **conv2\_2** layer
3. 56 x 56 x 256 tensor output of **conv3\_3** layer
4. 28 x 28 x 512 tensor output of **conv4\_3** layer

These tensors are up sampled to the size of original image and concatenated together to form hyper-columns.

So, now our CNN structure is as follows:

Layer 1:

Input:  Batch normalized output of conv4\_3(28X28X512).

Filter:  1X1

Output: 28x28x256 tensor.

Layer 2:

Input: Sum of the output of layer 1 tensor (after upscaling) along with the batch normalized output of conv3\_3 layer(56x56x256).

Filter: 3x3

Output: 56x56x128 tensor

Layer 3:

Input: Sum of the output of layer 2 tensor (after upscaling) along with the batch normalized output of conv2\_2 layer(112x112x128).

Filter: 3x3

Output: 112x112x64 tensor

Layer 4:

Input: Sum of the output of layer 3 tensor (after upscaling) along with the batch normalized output of conv2\_2 layer(224x224x64).

Filter: 3x3

Output: 56x56x128 tensor

Layer 5:

Input: Sum of the output of layer 4 tensor (after upscaling) along with the batch normalized output of conv1\_2 layer(224x224x3).

Filter: 3x3

Output: 224x224x3 tensor

Layer 6:

Input: Output of layer 5 tensor(224x224x3)

Filter: 3x3

Output: 224x224x2 i.e. an UV image of dimension 224x224.

We take the output of the conv4\_3 layer and use batch norm to get a 28x28x256 tensor. This was then up scaled to 56x56x256 and we did a 3x3 convolution with the output of conv3\_3 layer after applying batch.

On this model, we used a number of combinations of optimizer, batch size and error function to get a successful coloring. The two combinations that we heavily tested were

1. *Batch size = 1, Gradient Descent Optimizer and squared error function*

We first kept a batch size of 1 image to get faster training. The problem with this approach was that there was a lot of inconsistency in the weights learned in the process. This lead to images being colored inconsistently and shaded slightly with shades of green or red.

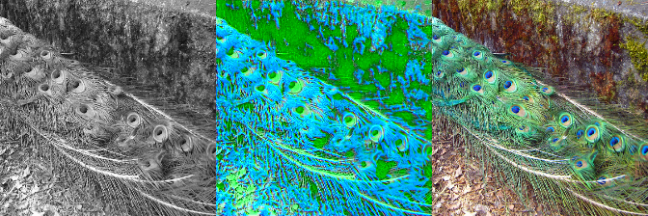
 

297000 298000

Again, Gradient Descent Optimizer tends to converge to a local optimum. So, we ran two different versions of the same code and got two different colorings for the same set of images.

*Examples:*









1. *Batch size=10, Adam Optimizer and Squared error function*

Increasing the batch size to 10 has the problem of being 10 times slower than the previous approach. So, we could only run 1000 iterations per hour. But the problem of inconsistency of training was somewhat minimized by doing this.

Adam optimizer uses momentum factor along with back propagation while weight update. This helped in dodging the local minimum problem that we faced by running the gradient descent optimizer.

*Some intermediate colorings are:*



64500

We can observe in the iteration 64500 that the machine is trying to learn the color of sky and we can see traces of blue near the buildings. The machine has also identified the field correctly and the tractor as a separate object from the field.



70000

Similarly, in iteration 70000 the machine correctly predicted the color of the tray (blue) and the ground beneath the tray. It identified all the objects on the tray correctly.



90000

We can also see that the machine is slowly learning to identify objects from one another as the orange color in 90000th iteration is limited to the face and skin only. The background and the plants are left away from range coloration.



2900000



4950000

. The machine keeps on getting better and recognizes correct colors as it iterates.

**Final results:**











