CS231N Project: Coloring black and white world using Deep Neural Nets

Vivek Kumar Bagaria
Stanford University
vbagaria@stanford.edu

Kedar Tatwawadi
Stanford University
kedart@stanford.edu

Abstract

We address the problem of adding colors to gray scale images. We use convolutional neural-nets because color of a pixel is strongly dependent on the features of its neighbors. In this work, we use a pre-trained neural net (VGG) as a base-network to extract features. We then add a feature-compression layer and small conv-net on top of it to perform the task of coloring. Since the base-network is pre-trained, the rest of the network requires only 3 hours to train.

1. Introduction

Human intelligence will be eclipsed very soon. Sooner than *you* think.

History of science consists of many scientific revolutions and outcomes of these revolutions have completely changed our lives by substituting particular human task with machines. For example the energy revolution paved the path to eradicate human labor, whereas the information revolution connected the whole world and gave us tremendous computation power. The next age is the age of artificial intelligence and it will replace the job of the human brain - the task of thinking!

Neural networks have tremendously accelerated the field of machine learning by solving problems which were written off as (near) impossible 10 years ago. They have successfully infiltrated every field of science and have already conquered a few of them; one of them being computer vision/image processing. This field of computer vision is undergoing changes at an inconceivable rate and at the heart of these changes are Convolutional Neural Networks (CNNs). The reason for such a swift development is because neural networks are relatively simple when compared to the kind of tasks they accomplish. Unsurprisingly CNNs are performing tasks, which few years ago, exclusively belonged to the territory of human-creativity.

We attempt to solve one such task - the task of 'coloring' a gray-scale image 1.

If we consider an image in a YUV 2 format, then the Y channel corresponds to the intensity profile of the image, while the UV channels represent the color. Formally our problem corresponds to designing a neural-net framework (which we hereon call as *ColorNet*) which estimates the UV channels, given the Y channel as the input.

Figure 1: Input : Left image Ideal Output: Right image.

Problem statement: Given a gray-scale image like the left image, our goal is to produce a colored image like the image in the right.

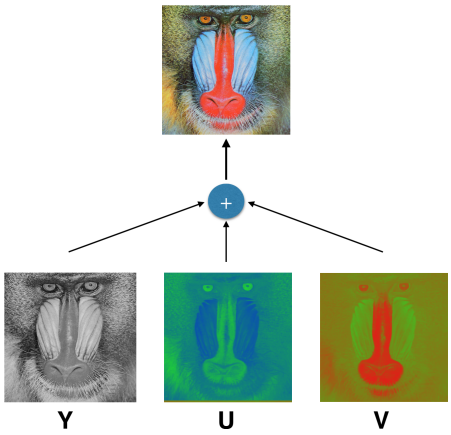


Figure 2: Dividing the colour image into 3 channels YUV. Where, **Y** - Intensity. **U,V** - Chrominance. The input to our model is the Intensity image and output is **U,V** images.

Generally, it is very difficult to achieve exact coloring

from the black and white image, as there is an inherent loss of information when one converts a colored image to gray-scale. Thus for this task, one cannot expect to obtain high accuracy (say) as compared to a classification problem. For example a gray-scale image containing a gray-colored car could correspond to car with various shades (like blue, red) and each of these would be a 'valid' image coloring in itself.

However, we can expect good colorization for natural images, such as images containing skies, water (which generally have shades of blue), or grass (shades of green), fruits, animals. We also expect good partial colorization of portraits, in which we expect the model get the skin-tone, hair color accurately but we do not expect it to get the color of surroundings correctly because they generally consists of dress, poster etc.

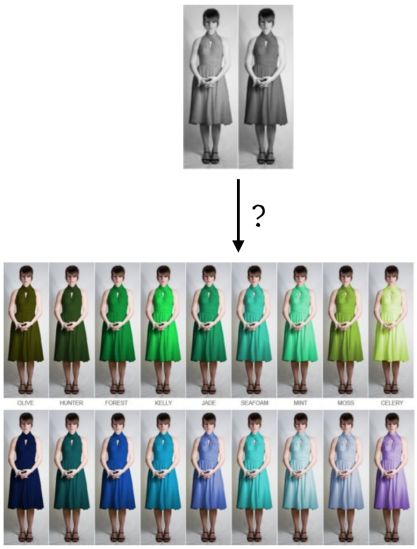


Figure 3: What is the dress color?

2. Existing Work

The traditional approaches to this problem can be divided into three categories :

1. Models in which the input consists of gray-scale image and color "seeds" (usually provided by a human). These kinds of approaches use convex optimization to achieve coloring.

2. Few approaches tackle the problem by coloring a gray-scale image by "using" similar colored images.

3. Semi-NN: In this approach, standard image features such as HoG, DAISY, color bins are extracted from the image. These features are then fed into a fully connected neural network, which produces a colored image.

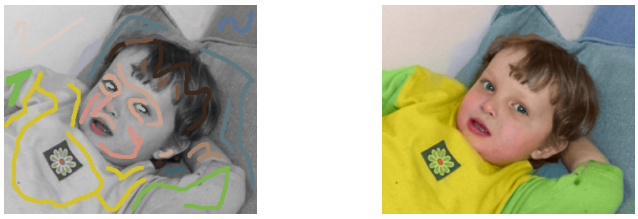


Figure 4: Scribble-based Colorization method

The first class of models use user input[2, 3], which forms the color palette. These 'scribble-based' methods require significant manual input. In these class of models, the model uses convex optimization techniques to color the image, however it does not 'learn' the colour palette, which happens to be the harder part.

Figure 5: [4]Transfer Colorization method

The second class of models, known as 'color transfer' models [4, 5] transfer color from a reference colored image onto the target gray-scale image. However, in such models finding a suitable reference image can be a difficult task. Also, the performance heavily depends upon finding a similar reference image.

The third class of models requires no input from the user. In these class of models [6], specially designed features such as patch feature, DAISY feature etc. are extracted from the gray-scale image, which serve as the input to a neural network which performs the colorization. These models achieve impressive colorization results, but are slightly rigid due to the specific custom-designed features used in the models.

A similar, but a very novel approach involves extracting the features from a Neural Network [1], instead of using custom features. These features, known as hypercolumns [10], are typically extracted from the intermediate layers of a pre-trained Neural Network. This increases flexibility and scalability of the model.

Our model builds upon this idea by choosing appropriate features for the task of coloring. We describe our model in the next section.

3. ColorNet Framework

Our model's framework consists of three parts:

- **Feature-Net:** Extracts features from the image - We accomplish this task by using a model pre-trained for

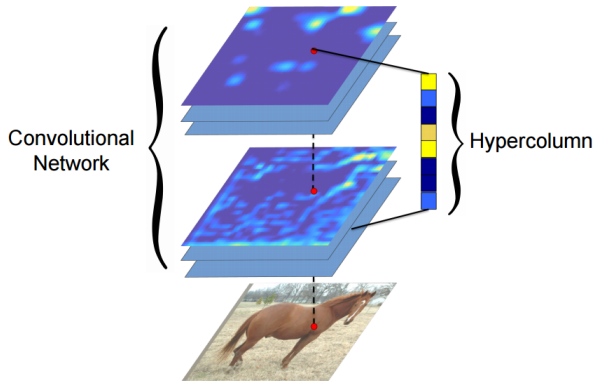


Figure 6: Extracting hypercolumns from the gray-scale image as features

classification. The extracted features contain redundant features and unwanted features (for task of coloring).

- **UpConv-Net:** Compresses the features obtained from the Feature-Net by removing redundant and unwanted features. This also has significant computational benefits.
- **Shallow-Net** This part obtains compressed features from the UpConv-Net and outputs **U,V** channels of the image.

The overall framework of the ColorNet is summarized in figure 7.

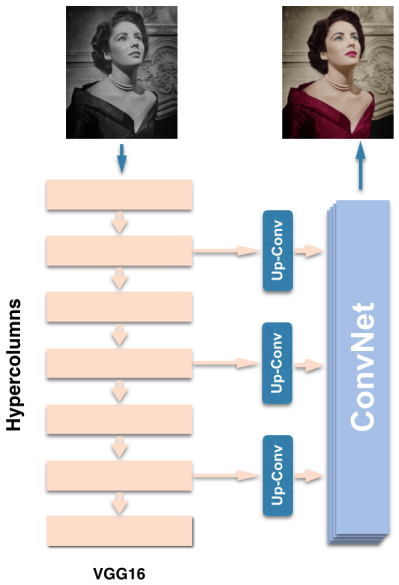


Figure 7: Overall Neural Net Model

We now go into the details regarding the motivation for choosing this framework for our ColorNet.

Feature-Net

For the task of colorization, the network requires local features to color nearby pixels with similar shades, on the other hand it needs global features to decide the macro-colors of the image.

We obtain this information by extracting the intermediate features of a pre-trained image classification net (example: VGGNet [11], ResNet [13]). These intermediate layers constitute the *hypercolumns* [10]. In this case, the lower layers of the Feature-Net give localized information, while the higher layers represent more global information. In our project, we use pre-trained VGG-16 net as our Feature-Net.

UpConv-Net

The UpConv-Net consists of a series of upconv layers, which scale the hypercolumns extracted from different layers to the same spatial dimensions. The UpConv-Net also reduces the effective number of feature layers, thus performing a lossy compression of the features. In a sense, the UpConv-Net learns an optimal set of features by removing redundancies and unwanted features.

Shallow-Net

The Shallow-Net consists of a small sequential CNN (2 or 3-layer CNN), which takes in the compressed features as an input from the UpConv-Net and outputs the UV channels, representing the color. The Shallow-Net architecture which we are using is given below:

We are using the *tanh* transfer function to restrict the output UV range to the valid UV-range $[-0.5, 0.5]$.

Input :	`#hypercolumns` $\times 224 \times 224$
(**Layer 1**) :	`SpatialConvolution` $(64, 3x3, 1, 1)$
($Layer\ 2$) :	`SpatialBatchNormalization`
($Layer\ 3$) :	`ReLU`
($Layer\ 4$) :	`SpatialMaxPooling`$(2, 2, 2, 2)$
(**Layer 5**) :	`SpatialConvolution` $(128, 3x3, 1, 1)$
($Layer\ 6$) :	`SpatialBatchNormalization`
($Layer\ 7$) :	`ReLU`
($Layer\ 8$) :	`SpatialMaxPooling`$(2, 2, 2, 2)$
(**Layer 9**) :	`SpatialConvolution` $(2, 3x3, 1, 1)$
($Layer\ 10$) :	`Tanh`[Transfer Function]
Output :	$2 \times 56 \times 56$
Loss :	Mean Squared Error/Abs. Error

Transfer Learning on VGG model

The pre-trained VGG model used is trained on color images and thus many of the features given by the VGG model contains information regarding the color of the image. However, our input is a gray-scale image. Therefore the features corresponding to the 'color' of the image has to be suppressed/altered. We do this by back-propagating the gradients of the loss into the VGG layers. In short, here, the VGG model trained on color images is transfer-trained to a VGG model trained on black images.

We are pruning the VGG-Net after layer 15 (as that is the last layer used for hyper-column construction). This leads to speed improvements in the forward pass and backward pass. We next discuss the different loss functions used for evaluating the performance of the colorization.

4. Loss Functions

We have tried the following loss functions, their combinations and their variants. Here $\mathrm{u}_{i,j}^{True}, \mathrm{v}_{i,j}^{True}$ are the true **U,V** values of the image at the pixel (i, j), whereas $\mathrm{u}_{ij}^{Op}, \mathrm{v}_{ij}^{Op}$ are the **U,V** values of the output image.

Abs. Loss

The loss is given by

$$Loss = \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\left|\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i,j}^{Op}\right| + \left|\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i,j}^{Op}\right| \right)$$

MSE Loss

The loss is given by

$$Loss = \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\left(\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i,j}^{Op}\right)^2 + \left(\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i,j}^{Op}\right)^2 \right)$$

Other loss function added in appendix 9

Performance

In this section we list out the pros and cons of using the above loss functions.

1. **MSE Loss:** The output pixels (using MSE loss) were good on average. None of the pixels were way off from the original pixels. However, the output image was blurred and the colors leaked out from the object - For example the sun's yellow color leaked out to the blue sky.

2. **Abs. Loss**: Compared to MSE loss, the output given by the Abs. Loss were sharp and they had no color-leakage problem. However, few pixels in the image were way off compared to the original image and these pixels (even though they were a small fraction) reduced the quality of the image.

3. TV-Loss - This loss function had very poor results and the results indicate that this loss function seems to wrong at the fundamental level. More detailed explanation provided in the later section.

5. Training and Validation Data-set

The Input dataset for the colorization problem is not a big problem, as any standard image dataset into a gray-scale dataset, which can be used to train the model. We used train images from the ImageNet dataset [12] and the LFW dataset [14].

We restricted our input images to the following classes

1. Natural scenes like mountains, sunset, beaches etc.

2. Fruits

3. Animals

4. Human portraits

6. Experiment details

We conducted experiments on two broad classes of datasets. The first class consists of the LFW dataset[14] containing human portraits. The second class of dataset comprised a sub-dataset of the ImageNet dataset. We picked a few different classes (3 to 6) from the ImageNet dataset, consisting of mainly natural images of scenery, sea, fruits, animals.

VGG16 has many layers from which features can be extracted and using features from all of the layers is not computationally feasible. Also there is lot of redundancy in the information across layers and hence we used a subset of layers. As expected, using features from more layers lead to slight improvement in the color performance. After balancing the trade-off between accuracy and computation-ability, we narrowed down 4 layers, each from different broad sections of the VGG16. In together they represented global information as well as local information, both of which are necessary for the task of coloring. One of our aim was to achieve good colorization results using a small ShallowNet.

We ran experiments with different Loss functions such as MSE loss, Abs loss and Total-Variation Loss. MSE Loss, Abs Loss gave a good results, whereas the TV loss gave poor results. We also experimented with different transfer functions, such as `sigmoid`, `tanh` and `linear-clipped` functions. Our results indicated that the tanh transfer function allowed for much brighter and better colors as compared with the sigmoid function. The

reason was because sigmoid is **not** a zero-mean function and hence it biases a few colours over other colors. The `linear-clipped` transfer function had poor performance because many pixels were colored with the colors at the clip.

We experimented with back-propagating the loss gradients through the Feature-net, along with the shallow net, and it led to significant performance improvements. As mentioned earlier, the reason behind this is the fact that the VGG16 network is pre-trained for the classification task on colored images. Thus, lot of neurons are sensitive to the color of the input image rather than the intensity texture. When the input is a gray-scale image, these neurons give out noisy features. By back-propagating into the VGG16 layer, we reduce this problem significantly by suppressing the weights of such neurons. .

Hyper-parameters

We used torch framework for our project. Since our network was not sequential, we designed our network using the *nngraph* from torch. We used *Adam* for updating the network parameters. We experimented with the learning rate hyper-parameter, and concluded that the optimal learning rate in the range of $[10^{-4}, 10^{-6}]$ with a decay factor of 0.85 per epoch. Our batch size was 32 on a CPU and 6 on a GPU

The training of the network clearly had two parts associated it with

1. **First 250 iterations** During the first 250 iterations the training loss decreased rapidly and after 250 iterations, the model produced images which has the 'sepia' tone to it.

2. **After 250 iterations** The training loss decreases slowly but now the model's output had more colors and it was learning to add more colors.

In the next section we shall present the output of our models on test cases. We also try explaining on why the model performs good/bad on the given test case.

Note: For all the results described below, same hyper-parameters were used.

6.1. Natural images colorization

We chose **six** different classes from the ImageNet dataset and trained on the model described in the section 7. We broadly ran two experiments, one using Abs. Loss function and the other using MSE Loss function. After comparing the results of both loss functions, we concluded that MSE Loss function gave better performance. Few of the outputs produced by the model on test data set can be seen in figures 8 and 9

(a) The output image misses the yellow color of the sunset, however it is a plausible image in itself

(b) The model colors the lemon perfectly. However, the 'half' apple in the right has been mistaken as lemon and colored yellow

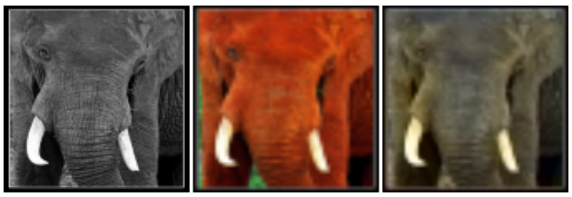
(c) The model completely misses the original image's color. It incorrectly gives a red shade to the whole picture.

(d) In this test case, the model performs very poorly, it is not able to re-produce the color of sky, land, waters.

(e) The model gets the elephant's color perfectly, however it misses the background by a large margin and ends up coloring it in 'sepia' tone

Figure 8: Test case outputs on a model trained on six classes of images from ImageNet dataset with Abs. Loss. The left figure is the gray-scale input, the center figure is the true colored image and the right figure is the model's output.

(a) The model misses the sunset's yellow color. But, the output image a plausible color-image in itself.



(b) The original photo of a African red-elephant. However, the model might not have seen red-elephant before and hence it gave the elephant a 'regular-elephant' color.

(c) The model misses the blue color of water and sky. The output mainly consists sepia colors.

(d) The model colors the image perfectly. There is no leakage of colors from tomatoes to lemons (or vice-versa)

(e) The model completely misses the original image's color. It gives a red shade to the whole picture instead of a blue, green shade. It seems like the output image is shot on mars

Figure 9: Test examples on a model trained on six classes of images from ImageNet dataset with MSE Loss. The left figure is the gray-scale input, the center figure is the true colored image and the right figure is the model's output.

6.2. Human portraits colorization

This projects' ultimate aim is to convert the gray-scale photos/videos of the 1800-1935 era to color photos. Thus training a neural network to color human portraits was the obvious next step. Before we started training the neural network for portraits, we had few expectations.

(i) The network will be able to learn the skin color perfectly **(ii)** However, it will perform poorly while coloring the dress and the background.

Our expectations were confirmed after we analyzed the output of the neural network. However to our surprise the performance of model with MSE loss was significantly better than the performance of model with Abs. Loss. In the interest of space, we have presented the MSE model's performance on test cases.

In the cases 10b, 10e, the color of the dress is not poorly re-produced, whereas the model colors skin very well. In the cases 10a, 10e, the background is poorly colored whereas the skin is color perfectly. Cases in 10c and 10d have been re-produced very well because the background is black.

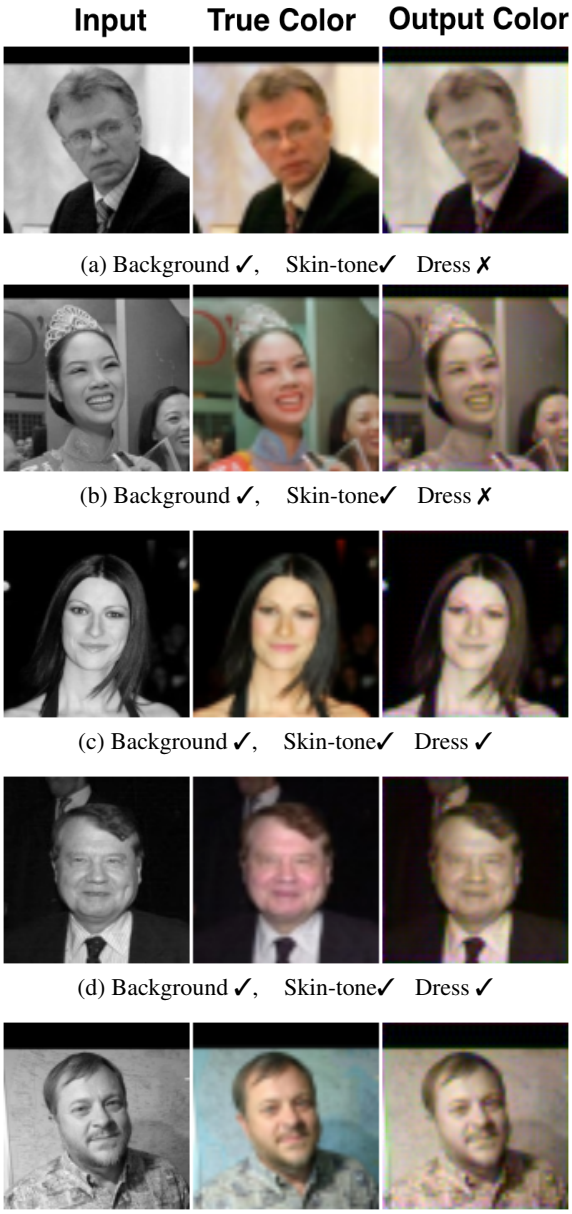
7. Ideas which did not work

We tried various modifications to the network, loss function etc. Some of them improved the results while the others did not improve. In this section we list the ideas which did not 'work'.

1. **TV loss + MSE Loss:** The output image using **only** MSE loss was a blur and to improve the 'sharpness' of the image, we tried combining TV loss and MSE Loss. **Reason:** We thought that penalizing the gradient of the output might increase the sharpness of the image (since TV loss would enforce sparsity in the gradient). However, the output images were 'pixelated'. Example shown in fig. 11

Figure 11: Pixelated output

2. **Weighted-TV Loss + MSE Loss :** We then tried weighing the gradient inversely proportional to the "Expected" gradient.
Reason: We thought that penalizing all the gradients



(a) Background ✓, Skin-tone✓ Dress ✗

(b) Background ✓, Skin-tone✓ Dress ✗

(c) Background ✓, Skin-tone✓ Dress ✓

(d) Background ✓, Skin-tone✓ Dress ✓

(e) Background ✗, Skin-tone✓ Dress ✗

Figure 10: Test examples on a model designed to color human portraits. The left figure is the gray-scale input, the center figure is the true colored image and the figure in the right is the model's output.

equally was unfair because we would ideally want the gradients to be high at few pixels and low elsewhere. However, the model output did not change significantly and it was still pixelated.

3. **Weighted-TV Loss + Abs. Loss :** In the above experiments, we replaced the MSE-Loss with Abs. Loss, but we got the same output from the model.

   Conclusion: Penalizing the gradient of the output image doesn't improve the quality of the image.
   Reason: Our model produces the color of a gray-scale image based on the intensity map and the **texture** of the image. When the gradient of the output image is penalized, indirectly we are penalizing the texture of the output image. The fact that we are penalizing the source of information might be the reason for the bad performance of the TV-Loss function.

4. **Abs. + MSE Loss**: Both MSE and Abs. loss functions individually produced good images. Each of the loss functions had a class of images where they failed. So we thought that combining the loss functions might help the loss function solve each others problem. However, using a combination loss functions had no significant impact. We could not figure out the reason for this observed phenomenon.

8. Conclusion

In our project we give a proof of concept of a neural network which adds colors to a gray-scale image with very good accuracy. For a specific class (say cats, dogs) our model is trained in 3 hours! This heavily indicates that one could potentially use a very large network to train a model which could color any image on the internet.

Coloring a gray-scale image is an important problem because this task is not an easy task for a average human brain and the fact that a neural network is able accomplish this task indicates possibility that today's neural networks are already performing certain complex tasks better than humans. This also supplements the theory that the intelligence of a neural network is different from that of a human brain.

9. Further Work

1. The current models seem to have a fundamental bottleneck - they favor 'sepia' colors more than the others. According to our experiments we concluded that blindly increasing the network size might not solve this problem and finding out the exact reason for this is a good direction to proceed.

2. Trying out classification instead of regression might improve the results for some classes.

3. Using a generative adversarial model over the existing mode might enable to the model to color objects such as cars, dresses etc.[9, 8]

References

[1] http://tinyclouds.org/colorize/

[2] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. In ACM SIGGRAPH 2004 Papers,pages 689694, 2004

[3] http://www.cs.huji.ac.il/ yweiss/Colorization/

[4] T. Welsh, M. Ashikhmin, and K. Mueller. Transferring color to greyscale images. ACM Trans. Graph.,21(3):277280, July 2002.

[5] R. Irony, D. Cohen-Or, and D. Lischinski. Colorization by example. In Eurographics Symp. on Rendering, volume 2. Citeseer, 2005.

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

[7] Lotter, William, Gabriel Kreiman, and David Cox. "Unsupervised Learning of Visual Structure using Predictive Generative Networks." arXiv preprint arXiv:1511.06380 (2015).

[8] Denton, Emily L., Soumith Chintala, and Rob Fergus. "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks." Advances in Neural Information Processing Systems. 2015.

[9] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in Neural Information Processing Systems. 2014.

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

[11] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[12] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[13] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." arXiv preprint arXiv:1512.03385 (2015).

[14] Gary B. Huang and Manu Ramesh and Tamara Berg and Erik Learned-Miller, Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.

[15] Collobert, Ronan, Koray Kavukcuoglu, and Clment Farabet. "Torch7: A matlab-like environment for machine learning." BigLearn, NIPS Workshop. No. EPFL-CONF-192376. 2011.

Appendices

A. Other Loss functions

Total-Variation (TV) Loss

The loss is given by

$$Loss = \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i-1,j}^{Output}\right| + \left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i,j-1}^{Output}\right| + \left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i,j+1}^{Output}\right| + \left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i+1,j}^{Output}\right| \right) + \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i-1,j}^{Output}\right| + \left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i,j-1}^{Output}\right| + \left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i,j+1}^{Output}\right| + \left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i+1,j}^{Output}\right| \right)$$

Weighted Total-Variation Loss

The loss is given by

$$Loss = \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\frac{\left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i-1,j}^{Output}\right|}{\left|\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i-1,j}^{True}\right|} + \frac{\left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i,j-1}^{Output}\right|}{\left|\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i,j-1}^{True}\right|} + \frac{\left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i,j+1}^{Output}\right|}{\left|\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i,j+1}^{True}\right|} + \frac{\left|\mathrm{u}_{i,j}^{Output} - \mathrm{u}_{i+1,j}^{Output}\right|}{\left|\mathrm{u}_{i,j}^{True} - \mathrm{u}_{i+1,j}^{True}\right|} \right) + \sum_{i=0,\ j=0}^{i=56,\ j=56} \left(\frac{\left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i-1,j}^{Output}\right|}{\left|\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i-1,j}^{True}\right|} + \frac{\left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i,j-1}^{Output}\right|}{\left|\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i,j-1}^{True}\right|} + \frac{\left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i,j+1}^{Output}\right|}{\left|\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i,j+1}^{True}\right|} + \frac{\left|\mathrm{v}_{i,j}^{Output} - \mathrm{v}_{i+1,j}^{Output}\right|}{\left|\mathrm{v}_{i,j}^{True} - \mathrm{v}_{i+1,j}^{True}\right|} \right)$$