Image Transformation via Neural Network Inversion

Asha Anoosheh Rishi Kapadia Jared Rulison

Abstract

While prior experiments have shown it is possible to approximately reconstruct inputs to a neural net given an output from one of its frontal layers, it becomes increasingly difficult as one moves toward the back layers, as the number of possible inputs that create the same output increases. In a convolutional neural net that performs image classification, only the end result label can be interpreted as a definite, concrete topic in the real world. Our experiment makes use of the information obtained by inverting based on a class label, the last layers output, to change an existing image to match said label rather than performing traditional inversion.

1. Introduction

1.1. Convolutional Neural Networks

An artificial neural network is a model used to transform a large number of inputs into a different representation that is more useful. It can learn the features to mutate data into a different basis that can be used to separate and classify different inputs. Convolutional neural networks (CNNs) are like traditional fully-connected neural networks, except they have fewer parameters and share weights to learn similar features for different parts of the input domain. For this reason, CNNs are better suited for the domain of images, where the number of inputs (pixels) is large.

1.2. Related Works

Feature inversion of CNNs first arose from Mahendran *et al*., who use deep feature representations to try to reconstruct original images [3].

We were inspired by the work of Gatys *et al*. in representing the style of an image using feature responses at various levels of a CNN [1]. They combine the artistic style of one image with the content of another. While we have different goals, our loss function includes style and content losses.

Wei *et al*. also uses a CNN to modify an image's detected class, but performs this by masking a key part of the image then using a specific instance of another class to fill the hole [5]. Our methods differ by using the network's general perception of a class rather than a single instance. We also use a total-variation regularization term implemented in their version, which is detailed later.

1.3. Goal

Given an image and a class label, our goal is to to visibly alter the image until it classifies as said label when passed through a trained CNN. Not only should the final modified image classify as desired, but it should classify as strongly as that class as possible. Additionally, incomprehensible noise should not be applied to the image to achieve this, but rather it should be changed in a visibly meaningful way that reflects attributes of what one expects from the given class label. We will exploit the non-linearity a neural network provides to perform this optimization task.

2. Method

2.1. Inversion

As in the prior works, we apply the gradient of our loss function to our input image, numerically altering it to minimize the error (The loss function itself will be detailed in the next section). Since we are minimizing loss L with respect to the input image I, we must obtain the gradient $\frac{dL}{dI}$, and backpropagation can give us exactly this. Backpropagation computes the gradient of each layer with respect to its input, and, using the chain rule, can compute the derivative of the end loss layer with respect to the beginning input.

We do not need to train a network to achieve this; in fact, we can utilize an existing network that has already been trained for image classification that uses the classification categories we are interested in. In our case we use the VGG-19 network trained on the ILSVRC-2012 image dataset [4]. The network's weights are thus held constant, and the calculation of the gradient with respect to weights can be omitted during backpropagation.

Our implementation uses Caffe for Python and drew heavily from Liu's PyCaffe implementation of Gatys' algorithm [2].

We use L-BFGS as our optimization algorithm instead of applying the gradient to the image directly, as it allows us to

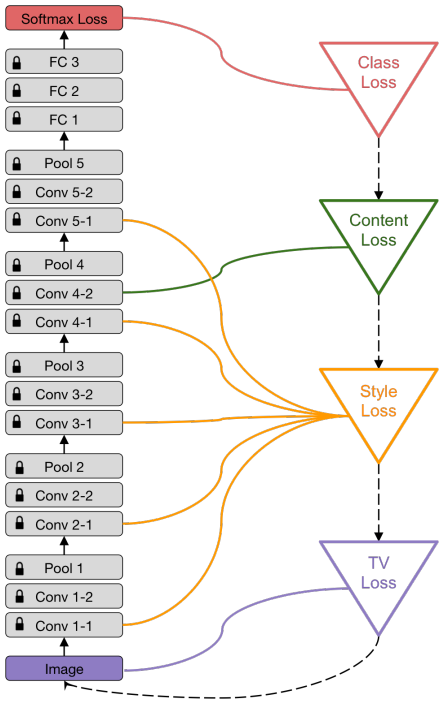


Figure 1: Our Optimization procedure using VGG-19

ignore learning rates and other hyperparameters associated with gradient descent.

2.2. Optimization Functions

Our main optimization function is the classification loss. We use the softmax loss over the classes, which computes the multivariate logistic cross entropy loss. We are not only want to make our target class the highest probability, but also want to push the probabilities for all the other classes down, to make our image look as much like one class and no other class as possible.

To modify our input image to match the desired class, we used hand-tuned ratios of the following constraints:

The content loss preserves the geometry of the image. It keeps the most important edges in the picture, with only slight distortions. It takes the Euclidean distance of the Gram matrix of the output image at the specified content layer in the network with that of the specified content image.

The style loss preserves the texture in the original image. This computes the Euclidean difference of the Gram matrices of the input image and the Gram matrix of the content image. More information about this loss function and the computation of the gradient can be found in Gatys *et al.* [1]

The total variation loss limits fluctuations between adjacent pixels. We took the total variation between both the horizontal and the vertical neighboring pixels, and summed both metrics into a single constraint function. The total variation of a function f over an interval $[a, b] \subset \mathbb{R}$ is given by

$$V_a^b(f) = \sup_P \sum_{i=0}^{nP-1} |f(x_{i+1}) - f(x_i)|$$

where the supremum runs over the set of all partitions of P of the given interval. Since our network deals with a discrete number of real numbers, the partition for which the supremum is achieved simply the entire interval of the image, or all the pixels. Thus, the formula becomes

$$V(I) = \sum_{i=0}^{N-1} |I(x_{i+1}) - I(x_i)|$$

where N is the number of pixels in the image. The image is flattened to form I, by both row-major and column-major, and the two resulting total variation metrics are added to form the total variation loss.

The similarity loss was added to mitigate the large color discrepancies between our input and resulting images. The network would occasionally produce an image that was offset to a completely different color than the input. To limit drastic color differences between the input and final images, this constraint computes the squared differences between the two images, but only considers the pixels for which the difference is greater than 30 percent.

2.3. Masking

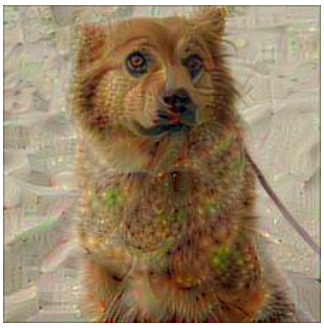
With just the implementations above, the network tries to alter the entire image to classify correctly. This may introduce unwanted artifacts to the background and any other objects that may be in the scene. To mitigate this problem for certain input images, we compute the gradients over whole image as before, but apply a manually-drawn mask to the gradient before applying the updates. We have seen that for some images, the mask tends to show improvement over the prior case, namely the images which are far away from their target class. However, for images which are close to their target class, such as an image of a cat turning into a tiger, there is not a significantly noticeable difference.

Deepdream also performed jittering of the input image as regularization to stop pixel-specific cheating. Following the same procedure, we randomly shift the image up and down as much as 16 pixels in each direction before each forward step. This should have the effect of mitigating the drastic discrepancies between neighboring pixels, as well as making the final image look more natural.

3. Results

We experimented with various combinations of the aforementioned loss functions and masking schemes, and we chose the following images as representative of the typical outcomes. The format is "new-class (content-loss multiplier, style-loss multiplier, TV-loss multiplier, color-loss multiplier)." The classification loss is always unscaled (x1), so all other losses are relative to it.

Original dog image



Tabby cat
(0,0,0,0)

Tabby cat
(0,0,0,1e-6)

Tabby cat
(1e-14,0,0,0)

Brown bear
(-1e-11,0,1e-6)

Brown bear
(-6e-11,1e-13,1e-7,1e-6)

Dalmation
(0,1e-11,1e-6,0)

Original cat image

Hyena [unmasked]
(0,1e-9,5e-6,1e-8)

Hyena [body-masked]
(0,1e-10,1e-7,1e-6)

Tiger [face-masked]
(0,0,0,0)

Tiger [face-masked]
(1e-10,1e-7,1e-9,1e-8)

Hole-filling with tiger label

Hole-filling with bear label

Hole-filling with gecko label

Brown bear [jittered]
(0,1e-11,0,0)

Tabby cat [jittered]
(0,0,0,0)

Tiger [jittered]
(1e-10,0,-1e-12,1e-11)

Tiger [body-masked, jittered]
(1e-10,-1e-12,0,1e-11)

Interesting accidental outcomes

4. Analysis

Using no constraints (all non-classification loss multipliers 0) tended to result in only somewhat-relevant noise being added to the image. Going from the first to second row in the first column demonstrates that the adding of Style and TV Losses reduces visually displeasing rainbow noise while maintaining desired alterations such as darker fur and emphasized stripes. Going from the second row to the third in the first column demonstrates that the adding of Content Loss smoothed the edges present in the original image, such as between the cat and the background and between different colors of fur.

We found the most success when modifying animals to related animals. We believe this is a result of the classes being distanced closely on the feature manifold, sharing shapes and textures. We were able to produce changes in texture such as stripes in the Tiger rows, darker fur in Tiger, Hyena, and Brown Bear, and spots in Dalmatian. The network was reluctant to produce changes in color beyond slight darkening or lightening.

We were unable to use the L-BGFS algorithm when using jittering, so we had to use simple gradient descent. Often, jittering helped to change the geometry of the image, but it also added undesirable colorful swirls to the image. The right-most "interestingi accidentl outcome" is an extreme example of the swirls. Sometimes it blurred the image due to the shifting and adding of the gradients.

5. Conclusion

Overall, we have obtained great results. We were unable to make progress modifying classes such as cucumbers, goldfish, and tennis balls. This may be due to a lack of closely related classes and easily modifiable textures in the original images. In addition, it was infeasible to significantly modify the geometry of the original image, since too negative of a content loss resulted in the included bloopers.

6. Further Work

One followup to this is abusing the constraints in unconventional ways to try to turn random noise into unpredictable artwork, similar to our failed examples above. Initializing the input image as random noise, and then trying to alter this image to classify as a specific class, we would hope to see the "average" representative of that class, without the blurring effects of a simple average image. With just the classification loss, it is more likely that the random noise will stay random but still classify correctly, since the random noise is not in the manifold of real images that the network was trained with. However, introducing and tuning our loss functions and others should constrain the resulting image to appear more along this manifold.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[2] F. Liu. style-transfer. `https://github.com/fzliu/style-transfer`, 2015.

[3] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[5] D. Wei, B. Zhou, A. Torralba, and W. T. Freeman. Understanding intra-class knowledge inside CNN. *CoRR*, abs/1507.02379, 2015.