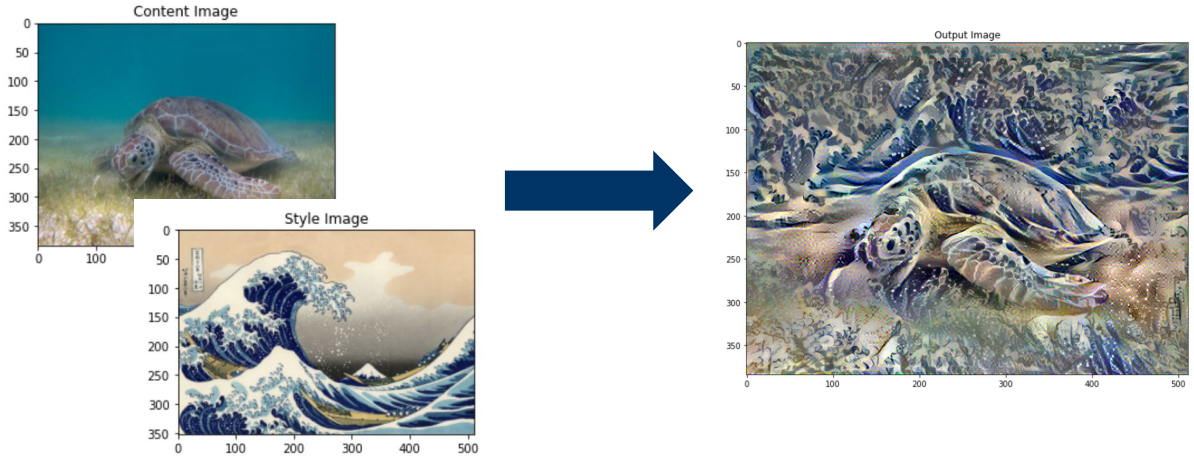
Berkeley
UNIVERSITY OF CALIFORNIA

Neural Style Transfer

Generating Art using Pre-Trained CNN Features and Gradient Descent

Sneak Preview: What We'll be Building



Berkeley

In this lecture we'll be talking about the design and implementation of an algorithm called "Neural Style Transfer," which can combine the "style" of one image with the "content" of another to produce new images, like this wave-turtle on the right. This might seem like an incredibly daunting task, but it's actually quite accessible. The key ingredient is that we don't have to start from scratch; we can build this style transfer algorithm on top of existing image classification model. We don't actually have to know much about how this pre-trained model works to use it for style transfer; we'll just be using its internal latent features as a black box.

Why Study Neural Style Transfer?

- Technical
  - Insight into how CNNs (convolutional neural networks) featurize images
  - Example of the power of pretrained models
  - Example of the power of iterative optimization
- Humanistic
  - Model of some aspects of human visual perception (perhaps?)
  - Generate new art!

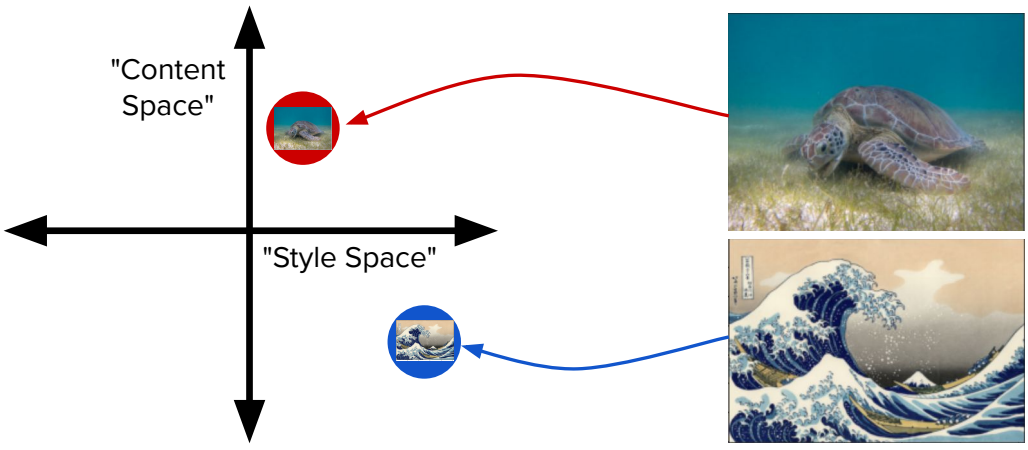
Berkeley
UNIVERSITY OF CALIFORNIA

Why are we talking about this algorithm in the first place? There are some technical reasons this algorithm is a useful example to study: it gives us some qualitative understanding of what image classification models based on convolutional neural networks "look like" internally, and it's a nice example of how pretrained models and iterative optimization algorithms can get you a long way towards implementing novel ML applications. But we're also studying style transfer because you might argue that the images it generates, and the techniques it uses to generate them, reveal something about what the intuitive human / aesthetic concepts of "style" and "content" actually *are*. It's also just a nice example of the fun things you can do with modern ML — you can

generate very compelling images and artworks using this algorithm and similar techniques.

Intuition

Decompose images into **style** features and **content** features

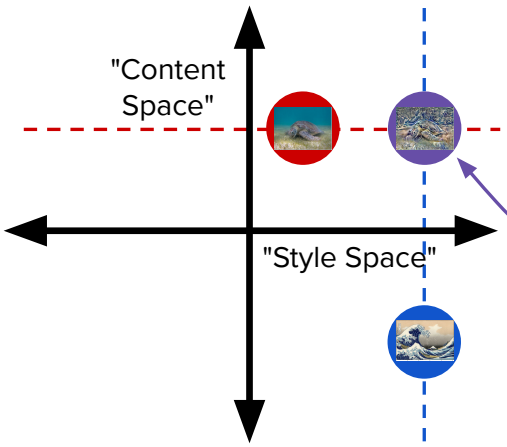


Berkeley

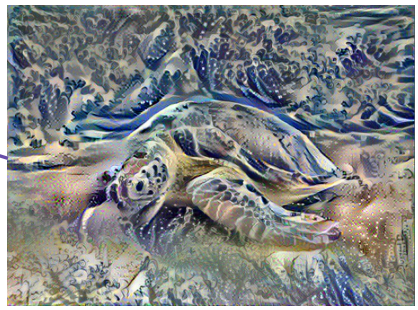
So, how might we go about actually designing an algorithm for style transfer? Well, imagine for the sake of argument that we had some magic black box that can decompose any input image into two vectors of features — "style" features and "content" features — and that these features actually correspond to our intuitive notions of "content" and "style". For simplicity, we can visualize the style feature space and the content feature space as both being one-dimensional, although of course in reality these spaces are very high-dimensional. In this simplified view, we can think of each image as being mapped to some point on a plane, where the "x coordinate" is the style of the image and "y coordinate" is the content.

Intuition

Decompose images into **style** features and **content** features

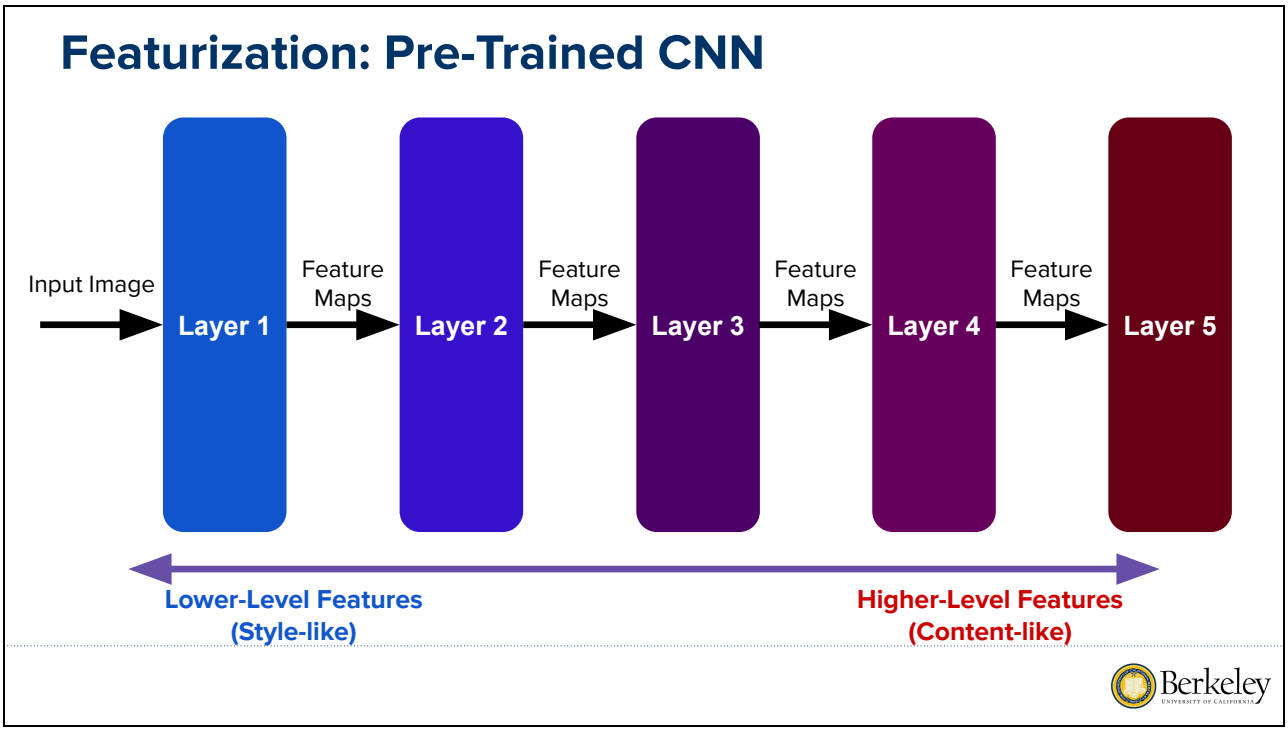


Generate a **new image** matching **style** features of one image and **content** features of another



Berkeley

Equipped with the ability to map images into this feature space, performing style transfer is actually quite straightforward. What we need to do is construct an image whose style features match those of the style source image as closely as possible, and whose content features match those of the content source image as closely as possible. In our simplified visualization, this is like finding an image whose "x coordinate" matches the featurization of the style image, and whose "y coordinate" matches the featurization of the content image.



So, can we actually build this magic black box that decomposes images into style features and content features? The answer is yes! The trick is to use a convolutional neural network model that's already been trained to perform an image-processing task, such as image classification, and extract the *internal* features into which it decomposes images at different layers in the model. We don't actually need to know much about how convolutional nets work to do this; all we need to know is that a convolutional neural net processes an image via a sequence of transformations, each of which transforms a representation of the image in one internal feature-space into another internal feature-space. In a trained model, we expect the feature spaces

of early layers in the network to represent fairly "low-level" features like texture and color, which intuitively correspond to "style." Moreover, in e.g. a classification model which has been trained to recognize semantic content in images, we expect the later layers in the network to represent fairly "high-level" features like object types and composite shapes, which intuitively correspond to "content." Thus, to decompose an image into style and content features, we can just run a classification model on it and record the internal feature maps it generates in the process. We then take the early feature maps to represent style, and take the later feature maps represent content.

Measuring Similarity: Content

- Want to measure **content** similarity between candidate image and content source image
- We want content similarity to consider:
  - At each content layer:
  - *Which* high-level features are present
  - *Where* in the image those high-level features are present
- Solution: per-pixel squared error

Source Image Feature Map

$$L^l_{content}(\vec{p}, \vec{x}) = \sum_{i,j}(F^l_{ij} - P^l_{ij})^2$$

Candidate Image Feature Map

Berkeley

Now suppose we have some candidate image, and we want to determine how similar its features are to the style and content features we want. How should we do this? Let's start with the content features. We want the generated image to contain the same "objects" — as modeled by high-level features — as the content source image, and we want it to have those objects in the same *place*. This means that to get a useful measure of content similarity, it suffices to extract content feature maps for the candidate image, content feature maps for the target image, and just measure the squared Euclidean distance between the two feature maps — in other words, the sum of the squared distances at each pixel.

Measuring Similarity: Style

- Want to measure **style** similarity between candidate image and style source image
- We want content similarity to consider:
  - At each style layer:
  - *Which* patterns of style features are present
  - *Regardless* of where these patterns occur
- Solution: compare squared distance btw. "Gram matrices" of each image
  - Gram matrix measures *correlations* between different features, across all pixels

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \qquad G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l,$$

(Normalization factor based on image size, number of features)

('A' is defined analogously to 'G', but for the target image 'P')

Berkeley

Now, how we measure similarity of style features? Unlike the content features, the style feature maps don't need to agree *at each position* in the image. Instead, we just want the *distribution* of all style features across both images to agree. To capture this, instead of directly comparing feature maps pixel-by-pixel, we compute something called a *Gram matrix* for the feature map of each image, which measures the prevalence of and relationship between different features in the feature map, but not where in the image those features occur. The Gram matrix thus gives kind of a statistical summary of all the features in an image, and which features tend to occur together with which other features. We then measure style similarity as the squared Euclidean distance

between the two images' Gram matrices, where the Gram matrices are generated from the style feature maps of each image.

Generating the Image

- Want to create which an image which measures as similar in **content** to the **content** source, and similar in **style** to the **style** source.
- Solution: optimization by gradient descent
  - Initially, start with generated image equal to content image
  - Iteratively update generated image to reduce style-space distance to style source, and content-space distance to content source
  - Perform updates with *gradient descent*: compute gradient of style and content distance w.r.t. pixels of generated image, then adjust pixels in the opposite direction.

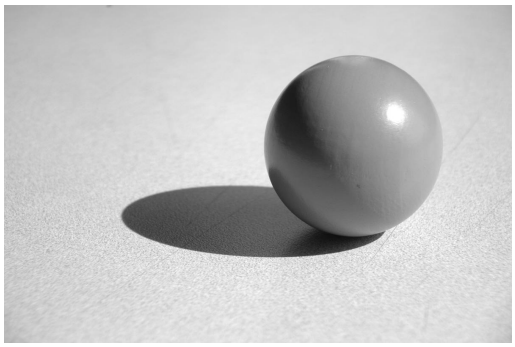
Berkeley
UNIVERSITY OF CALIFORNIA

So, we now have a way to extract the content and style features an image, and we have a way of measuring the style or content similarity (or equivalently, style or content distance) to each image. We want an image which minimizes content-distance to the content source, and minimizes style-distance to the style source. How do we find such an image? The trick is iterative optimization, using the technique of gradient descent. We can start with a candidate generated image which is just equal to the content source image — it doesn't have any extra style transformation applied yet. Then, we can iteratively improve it to minimize its content and style distance from the source images. This iterative improvement works by computing

gradients; at each step, we measure the gradient of the distance — really, some linear combination of the two distances we care about — and adjust the candidate generated image in the opposite direction of that gradient. This moves us gradually closer in style-space to the style source, while ensuring we don't stray too far in content-space from the content source. With a modern machine learning framework, we don't need to worry too much about how to compute the gradients; we can just specify the distance functions, and the framework does the rest.

Some Examples

Ball with Shadow + Starry Night



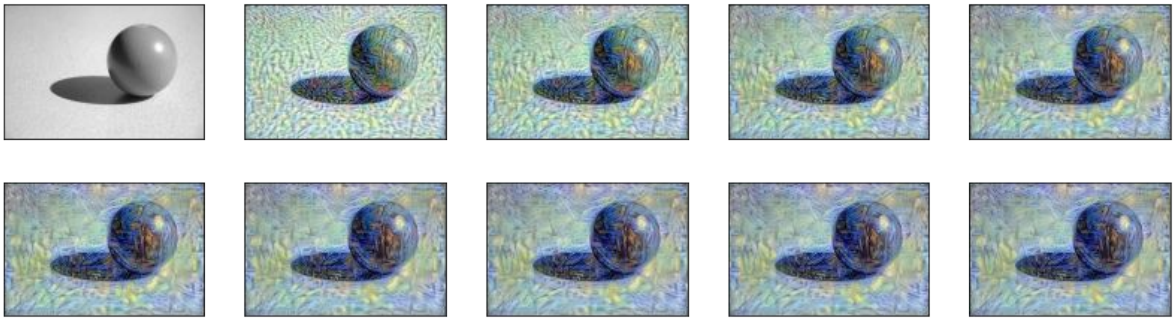
+

Berkeley
UNIVERSITY OF CALIFORNIA

Now we will look at some examples! Why don't we try something simple like a ball as our content image

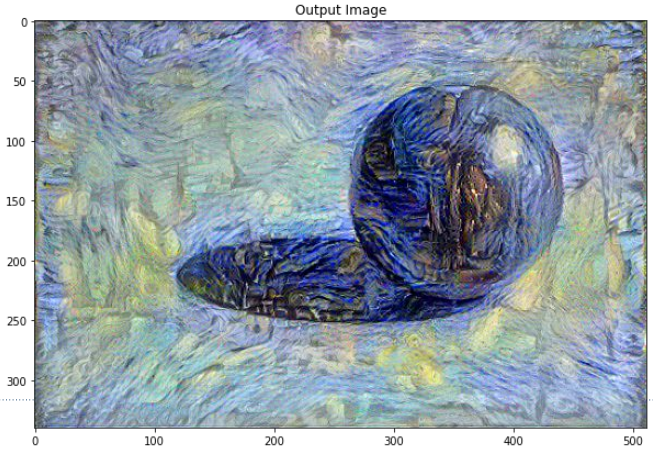
Some Examples

Ball with Shadow + Starry Night



Some Examples

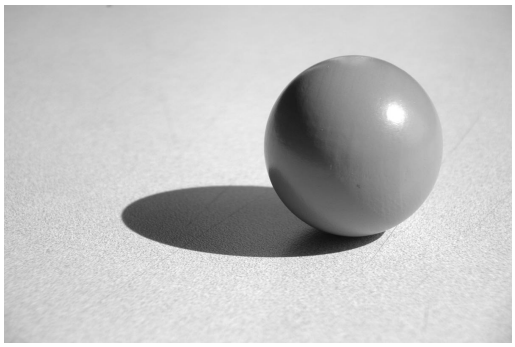
Ball with Shadow + Starry Night



Looks pretty cool!

Some Examples

Ball with Shadow + Pillars of Creation



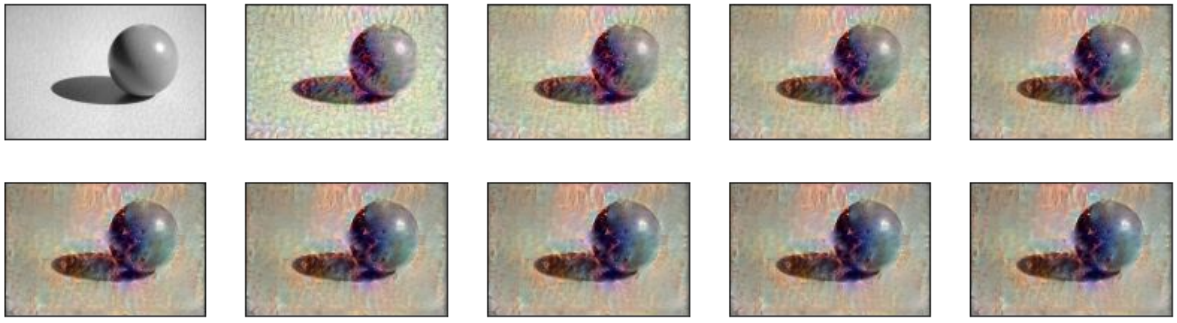
+

Berkeley
UNIVERSITY OF CALIFORNIA

Now lets try the transfer with a picture of the Eagle Nebula

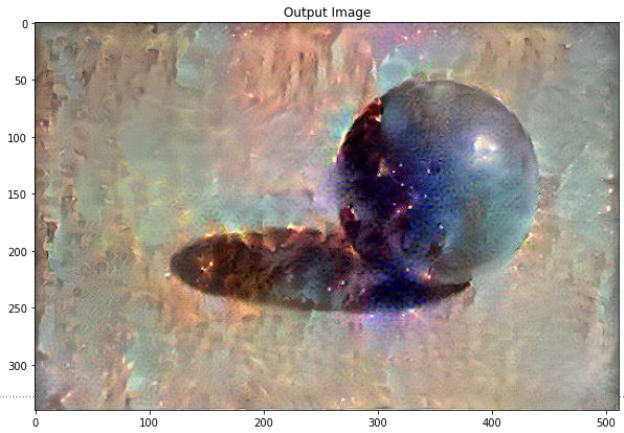
Some Examples

Ball with Shadow + Pillars of Creation



Some Examples

Ball with Shadow + Pillars of Creation



Berkeley
UNIVERSITY OF CALIFORNIA

Neat! This worked pretty well.

Can anyone guess when the algorithm doesn't work so well?

Some Examples

Oski + Pillars of Creation

+

Berkeley
UNIVERSITY OF CALIFORNIA

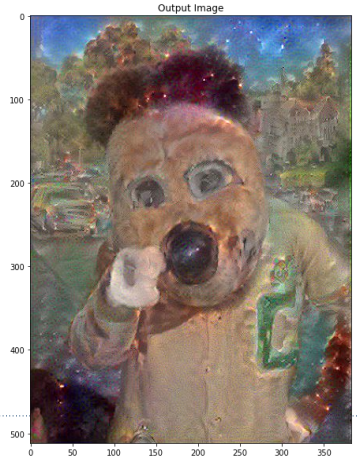
Now lets try the transfer with a picture of Oksi and the Eagle Nebula

Some Examples

Oski + Pillars of Creation

Some Examples

Oski + Pillars of Creation



This did not work as well.

Can anyone guess why the algorithm didn't work too well?

It might be because the style of the Pillars image was not easy to pick up or that the VGG-Network is not good at separating a background and the subject in the content.

References

1. Gatys, L. A., Ecker, A. S., Bethge, M. et al. "A neural algorithm of artistic style." arXiv preprint, 2015, https://arxiv.org/abs/1508.06576.
2. Simonyan, K., Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv preprint, 2015, https://arxiv.org/pdf/1409.1556.
3. Yuan, R. "Neural Style Transfer: Creating Art with Deep Learning using tf.keras and eager execution." Medium, 3, Aug. 2018, https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398.
4. Neural style transfer." TensorFlow.org, 2020, https://www.tensorflow.org/tutorials/generative/style_transfer.

Berkeley
UNIVERSITY OF CALIFORNIA