Deep Learning05: Texture Network

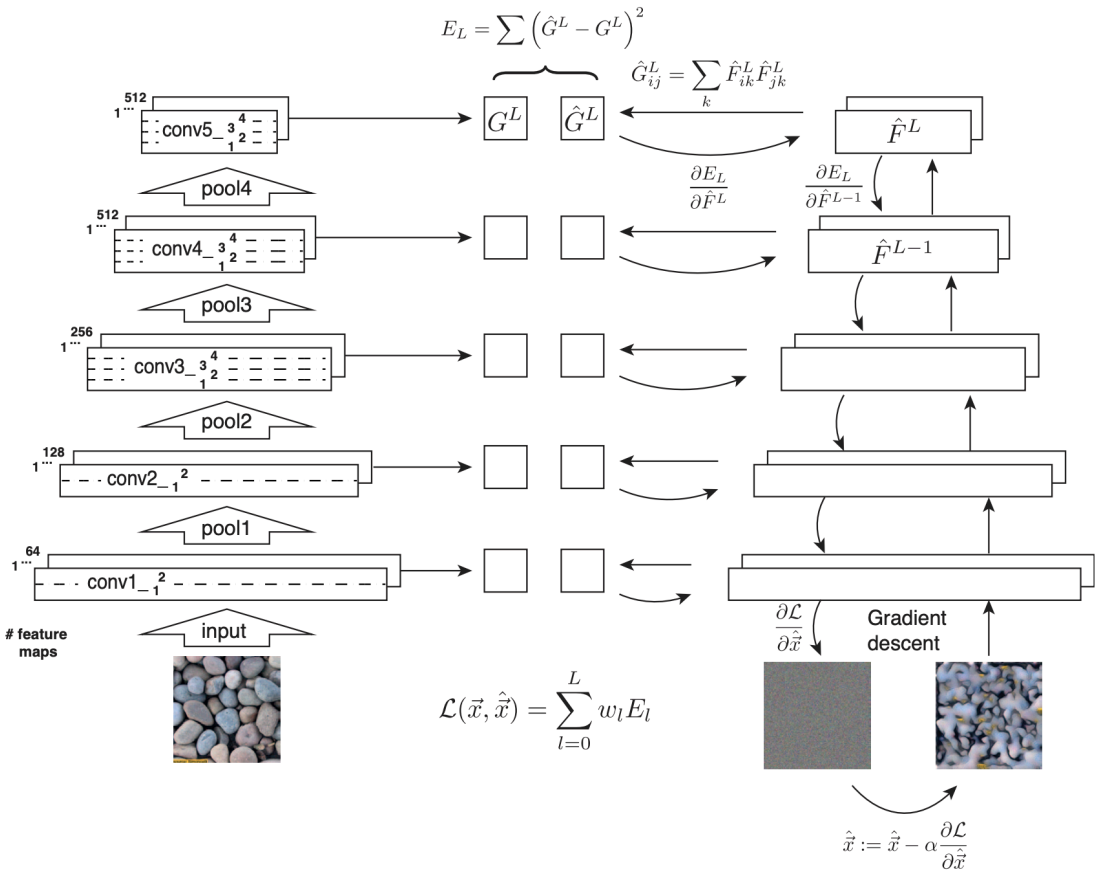
School of Computer Science and Engineering

17373240 Wanru Zhao

2020.6

Background

Textures synthesis can be viewed as approximating a texture image x from randomly initialized noise $\widehat{x}$.$\widehat{x}$ gets trained and optimized until its features extracted from the deep neural networks have the same statistics as that of the source texture image x.



Since textures are per definition stationary, the statistics of the features should be agnostic to spatial information. In this experiment, we use Gram matrix to describe the spatial-agnostic statistics of the source texture image and the generated texture image. The Gram matrix is defined as the correlation between different channel of features, which is shown as below,

$$G^l_{ij} = \sum_{m,n} F^l_{i,m,n} F^l_{j,m,n}$$

where Fi, m, n^l denotes the pixel with location (m,n) in the ith feature map in layer l.

So we can construct loss function as

$$\text{Loss} = \sum_l w_l \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \widehat{G^l_{ij}}\right)^2$$

where $G\ \widehat{G}$ sdenotes the Gram matrix of the texture image and the generated image, $N_l M_l$is the number of pixels of feature maps in layer l and w_l is the weighting factor of the contribution of each layer to the total loss.(we simply choose $w_l = 1$ for all layers in the baseline project.)

Tasks

Q1: Implementing Gram matrix and loss function.

Use the features extracted from all the 13 convolution layers, complete the baseline project with loss function based on gram matrix and run the training process.

$$G = A^T A = \begin{bmatrix} \mathbf{a}^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T \mathbf{a}_1 & \mathbf{a}_1^T \mathbf{a}_2 & \cdots & \mathbf{a}_1^T \mathbf{a}_n \\ \mathbf{a}_2^T \mathbf{a}_1 & \mathbf{a}_2^T \mathbf{a}_2 & \cdots & \mathbf{a}_2^T \mathbf{a}_n \\ \mathbf{a}_n^T \mathbf{a}_1 & \mathbf{a}_n^T \mathbf{a}_2 & \cdots & \mathbf{a}_n^T \mathbf{a}_n \end{bmatrix}$$

Codes and Results

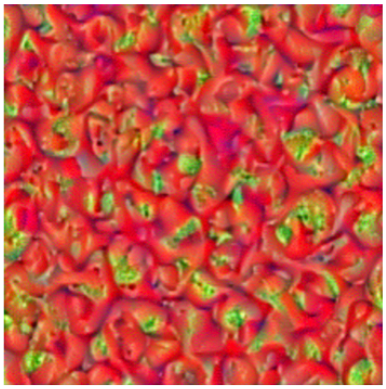
Without considering the weight w_l:

```
def get_gram_matrix(feature_map):
    shape = feature_map.get_shape().as_list()
    re_shape = tf.reshape(feature_map, (-1, shape[3]))
    gram = tf.matmul(re_shape, re_shape, transpose_a=True) / (shape[1]*shape[2]*shape[3])
    return gram

def get_l2_gram_loss_for_layer(noise, source, layer):
    Gram_s = get_gram_matrix(getattr(source, layer))
    Gram_n = get_gram_matrix(getattr(source, layer))
    return loss = tf.nn.l2_loss((Gram_s-Gram_n))/2

def get_gram_loss(noise, source):
    with tf.name_scope('get_gram_loss'):
        gram_loss = [get_l2_gram_loss_for_layer(noise, source, layer) for layer in
GRAM_LAYERS ]
    return tf.reduce_mean(tf.convert_to_tensor(gram_loss))
```

Origin



Output

Take the weight w_l into account, and adjust w_l to 1.

```
def get_gram_loss(noise_model, image_model):
    with tf.name_scope('get_gram_loss'):
        gram_loss = [get_l2_gram_loss_for_layer(oise_model, image_model, layer) for layer in
GRAM_LAYERS ]
        gram_weights=tf.constant([1. / len(gram_loss)] * len(gram_loss), tf.float32)
    weighted_layer_losses = tf.multiply(gram_weights, tf.convert_to_tensor(gram_loss))
    return tf.reduce_sum(weighted_layer_losses)
```

Analysis

For texture patterns with regular distribution, the network generation performance is good, especially after adding the weight.

Q2: Training with non-texture images.

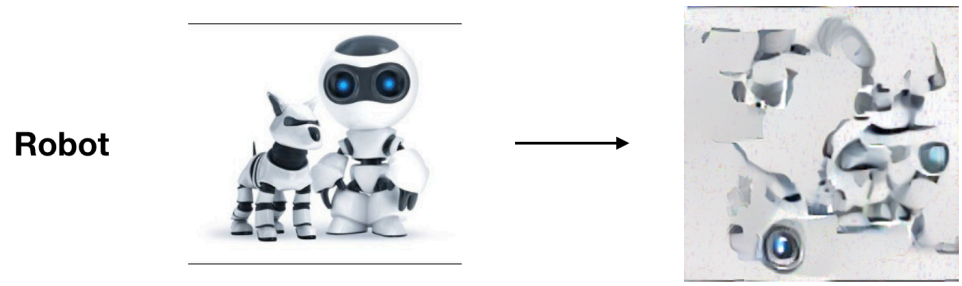
> *To better understand texture model represents image information, choose another non-texture image(such as robot.jpg in the ./images folder) and rerun the training process.*

Codes

The same as Q1.

Results

They differ significantly from the images used in the previous experiment in that there is no local texture information and the overall texture is very smooth.



Analysis

According to the experimental results, for non-textured images, the feature information will be broken up and then combined arbitrarily in the process of model generation.

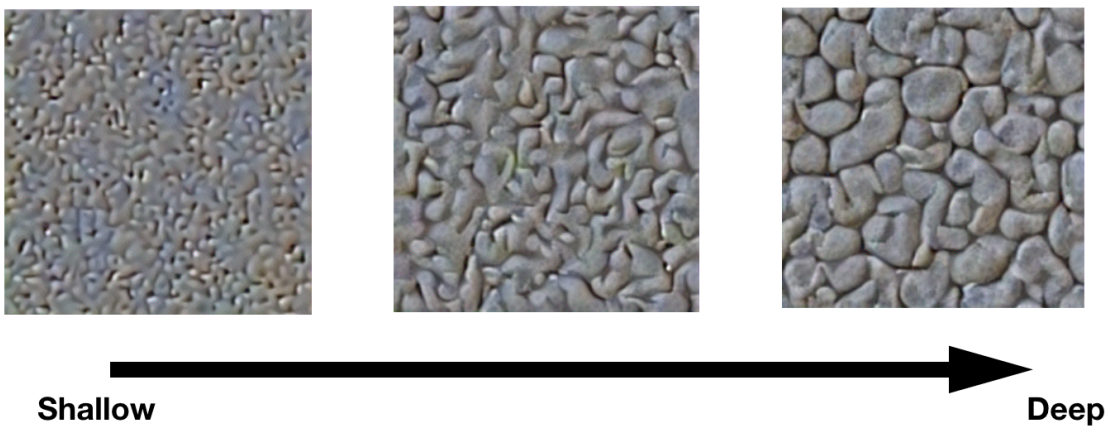
It can be concluded that this model is only applicable to textured photos.

Q3: Training with less layers of features.

> *The baseline takes the features from all the convolutional layers, which involves a large number of parameters. To reduce the parameter size, please use less layers for extracting features (based on which we compute the Gram matrix and loss) and explore a combination of layers with which we can still synthesize texture images with high degrees of naturalness.*

Codes , Results and Analysis

```
GRAM_LAYERS= ['conv1_1', 'conv1_2', 'conv2_1', 'conv2_2', 'conv3_1', 'conv3_2', 'conv3_3',
'conv4_1', 'conv4_2', 'conv4_3', 'conv5_1', 'conv5_2', 'conv5_3']
```

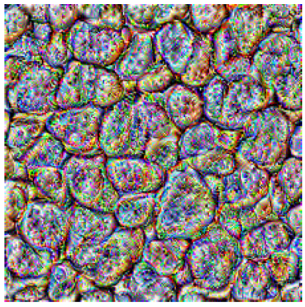


- As we can see, when the network depth is relatively shallow, the resulting image is similar to spectral noise. With the increase of depth, the structure of the image is gradually revealed.

```
# Only shallow layers:
GRAM_LAYERS= ['conv1_1', 'conv1_2', 'conv2_1', 'conv2_2', 'conv3_1', 'conv3_2', 'conv3_3']
# Only deep layers:
GRAM_LAYERS= ['conv4_1', 'conv4_2', 'conv4_3', 'conv5_1', 'conv5_2', 'conv5_3']
# One convolution for each layer:
GRAM_LAYERS= ['conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1']
```

One convolution for each layer

Only shallow layers



Only deep layers

- Shallow convolution extracts the abstract information of the image and can obtain better texture features. Deep convolution extracts the details of the image and can obtain better contour features.
- We can't get a good result simply by relying on the shallow layer or the deep layer.
- We get the best performance when we **only take the first convolution of each layer** and it is close to the baseline. Therefore, in order to integrate the contour information and detail information of texture, we can reduce the number of convolutional layers while **ensuring the existence of both shallow and deep convolutional layers**.

This trend is the same as that mentioned in the paper[3].

Q4: Finding alternatives of Gram matrix.

> *We may use the Earth mover's distance between the features of source texture image and the generated image. You may sort the pixel in each feature map of the generated image and the original image and compute their L2 distance to construct the loss, which is shown as below*

Codes

In statistics, the earth mover's distance (EMD) is a measure of the distance between two probability distributions over a region D. In mathematics, this is known as the Wasserstein metric. Informally, if the distributions are interpreted as two different ways of piling up a certain amount of dirt over the region D, the EMD is the minimum cost of turning one pile into the other; where the cost is assumed to be amount of dirt moved times the distance by which it is moved.[1]

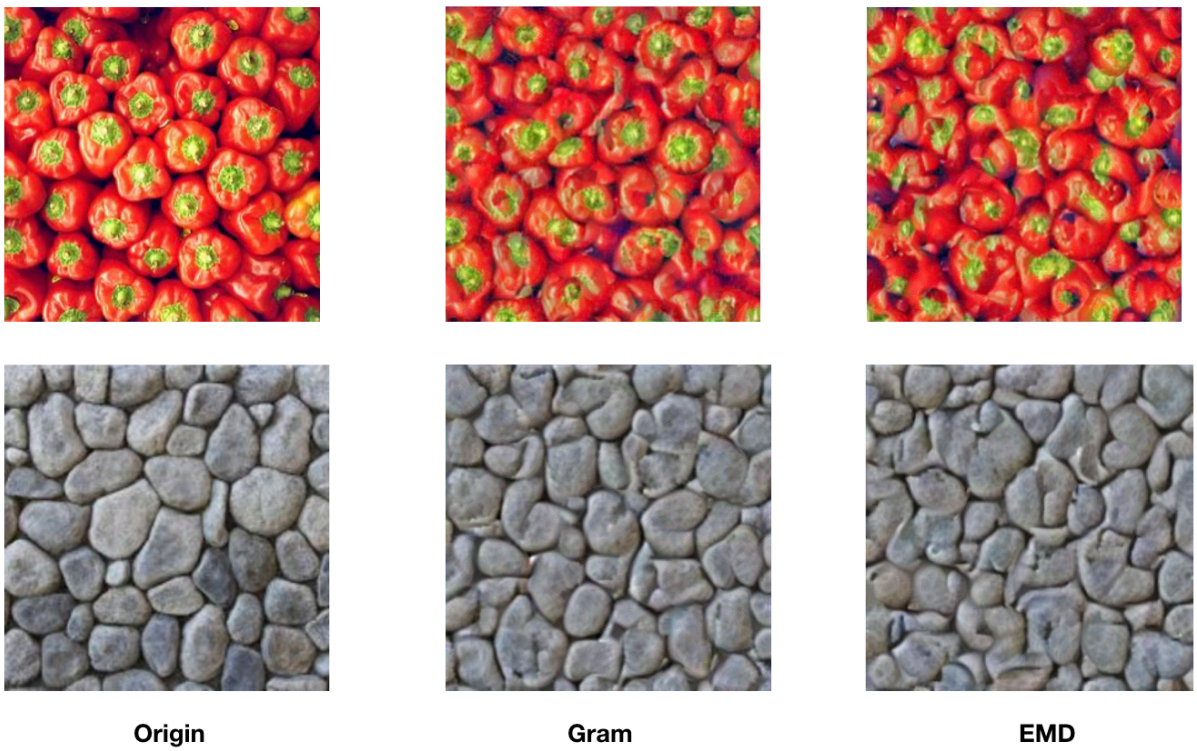
$$Loss = \sum_{l} w_l \sum_{i} (sorted(F_i) - sorted(\hat{F}_i))^2$$

where $F_i, \widehat{F_i}$ is the feature map of the original image and the generated image.

```
def get_l2_EMD_loss_for_layer(noise, source, layer):
    shape = tf.shape(getattr(noise, layer))
    noise_transpose = tf.transpose(tf.reshape(getattr(noise, layer), shape=(-1, shape[3])))
    noise_sort = tf.nn.top_k(noise_transpose, shape[1] * shape[2])[0]
    source_transpose = tf.transpose(tf.reshape(getattr(source, layer), shape=(-1,
shape[3])))
    source_sort = tf.nn.top_k(source_transpose, shape[1] * shape[2])[0]
    return config.gram_weight * tf.reduce_sum(tf.square(noise_sort - source_sort))


def get_gram_loss(noise, source):
     with tf.name_scope('get_EMD_loss'):
       EMD_loss = [get_l2_EMD_loss_for_layer(noise, source, layer) for layer in GRAM_LAYERS]
     return tf.reduce_mean(tf.convert_to_tensor(EMD_loss))
```

Results



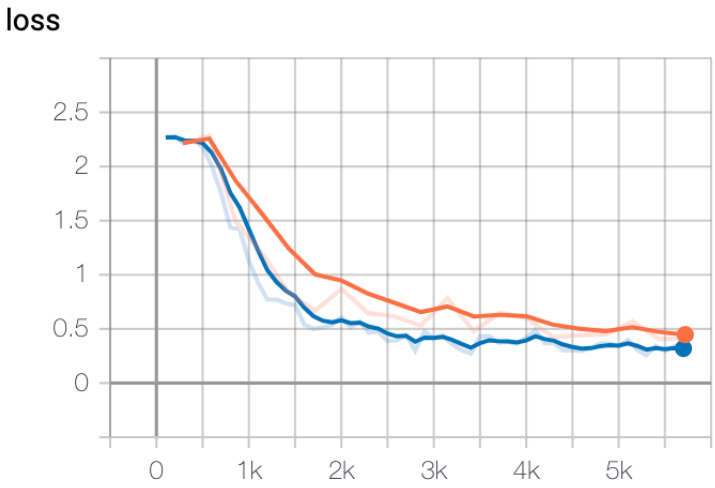
Origin **Gram** **EMD**

Analysis

- From the **experimental results**, our deep neural network learned the correlation between each feature vector of the original texture image, making the generated image and the original texture trend similar.

  But the sharpness is slightly worse than that achieved with the Gram matrix, and a lot of the color features are lost.

- From the **convergence speed**, EMD converges more slowly than the Gram matrix method.



Q5: Training with different weighting factor.

> *Use the configuration in Q3 as baseline. Change the weighting factor of each layer and rerun the training process.*

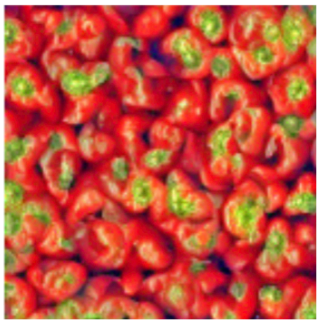
Codes

```
def get_gram_matrix(feature_map):
    shape = feature_map.get_shape().as_list()
    re_shape = tf.reshape(feature_map, (-1, shape[3]))
    return tf.matmul(re_shape, re_shape, transpose_a=True) / (shape[1]*shape[2]*shape[3])

def get_l2_gram_loss_for_layer(noise, source, layer):
    Gram_s = get_gram_matrix(getattr(source, layer))
    Gram_n = get_gram_matrix(getattr(source, layer))
    weight = ((GRAM_LAYERS.index(layer) + 1) / len(GRAM_LAYERS)) ** 2
    return weight * tf.reduce_sum(tf.square(Gram_n - Gram_s))
```

Results






Origin **Weight increases with depth** **Weight decreases with depth**

Analysiss

By observing the results, we concluded that **shallow convolutional layers learn the overall structural features, and deep convolutional layers learn the local details**.

Extensive Learning

ONNX

ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

Git LFS

I need to transfer the VGG16_onnx.nPY model to the remote server.

Because the model is so large, the common approach is not feasible at this time.

```
remote: error: GH001: Large files detected. You may want to try Git Large File Storage -
https://git-lfs.github.com.
remote: error: Trace: f30306bb9781d49439b957b656c30970
remote: error: See http://git.io/iEPt8g for more information.
remote: error: File vgg16_onnx.npy is 796.31 MB; this exceeds GitHub's file size limit of
100.00 MB
```

So I chose **Git LFS** to transfer the large model.

Git is a *distributed* version control system, meaning the entire history of the repository is transferred to the client during the cloning process. For projects containing large files, particularly large files that are modified regularly, this initial clone can take a huge amount of time, as every version of every file has to be downloaded by the client. **Git LFS (Large File Storage)** is a Git extension developed by Atlassian, GitHub, and a few other open source contributors, that reduces the impact of large files in your repository by downloading the relevant versions of them *lazily*. Specifically, large files are downloaded during the checkout process rather than during cloning or fetching.

References

[1] Earth mover's distance - Wikipedia

[2] https://www.atlassian.com/git/tutorials/git-lfs

[3] Gatys L, Ecker A S, Bethge M. Texture synthesis using convolutional neural networks[C]//Advances in neural information processing systems. 2015: 262-270.