The Final Project "Identification and Synthesizing of Raphaels paintings from the forgeries"

Anonymous Author(s)
Affiliation
Address
email

Abstract

Artistic identification and image style transfer has long been considered as a challenging task due to the complex and unstructured nature of fine arts, especially painting. In this report, we explore algorithms for identification and synthesizing of Raphael's paintings from the forgeries. For identification problem, we first use Geometric tight frame and Gabor wavelet to extract features. With selected feature based on area under ROC curve(AUC), we apply several techniques such as SVM, Neural-Network and Decision-Tree for classification. Leave-one-out cross validation is used to ensure accuracy. As for synthesization problem, we follow the method proposed in [2]. With VGG network and well-defined content and style loss functions, style transfer is reduced to an optimization problem. The highlights of our report is that not only detailed discussion of these methods are presented, but we also endeavor to implement our own original thoughts to improve these models.

1 Introduction

The project is divided as two parts. The first part is a classification problem based on 28 Raphael paintings which have been labeled as authentic works and forgeries. We intend to use some feature extraction techniques to first record the main characteristic of the images by some simple statistics, and then apply the machine learning methods for classification problem such as SVM and Neural-Network. The second part will be somewhat more complicated. We need to transfer the style of Raphael paintings to another picture while maintaining the semantic content of it.

Task1

Distinguishing genuine paintings from fake ones is not easy task because forgeries tend to imitate characteristic of the genuine ones. Algorithms such as Neural Network, SVM are quite popular for classification. However, simply apply these algorithms seems like a bad idea because our data set, usually over 10 thousand pixels one image, can be computational costly. Besides, as is mentioned in [2], in higher layers of the network, detailed pixel information is lost, which is key information for art authentication.

As a result, feature extraction without loss of information containing particular traces of style is an important data preparation. The article [9] use 18 filters named geometric tight frame to convolve the image before further experiments. Gabor wavelet filters, mentioned in is another efficient tool for feature extraction. Leave-one-out cross validation shows our best performance reaches around 90% accuracy with Gabor wavelet based decision tree model.

Submitted to 30th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

Task2

So far, many algorithms has been designed to accomplish the second task. The article [8] is based on a bidirectional analogy of two semantically similar pictures, which is an important point since the two picture will have very similar content information at the coarsest layer of the VGG network. And the desired picture at this layer is initialized the same as the input. Then a nearest neighbor field search method helps to reconstruct the picture in the previous CNN layer.

Another approach described in [2] to address the problem is more straightforward. The content and style content are separately dealt with and a loss function is defined as a weighted sum of the ones from the content and the style based on the VGG network. Then apply gradient descent method to find the output image which minimize the loss function.

2 Related Work

Task 1

Li et al. [7] made an effort to extract those visually salient brushstrokes of van Gogh based on an integrative technique of both edge detection and clustering based segmentation. With the extracted brushstrokes, some definitions of brushstroke features for art authentication were given in distinguishing van Gogh paintings from forgeries.

Qi. et al. [11] use background selection and wavelet-HMT-based Fisher information distance for authorship and dating of impressionist and post-impressionist paintings. Two novel points were introduced in this work. The first point is that background information is much more reliable than the details of an intricate object which cannot represent the artist?s natural style because of multiple edits and corrections. The second point is that an artist?s style should be interpreted as a probability distribution over a set of possible textures, and not just simply from the textures themselves.

Task 2

Liao et al [8] propose an attribute transfer technique across images that may have different appearance but have perceptually similar semantic structure. They use a Convolutional Neural Network to construct a feature space in which to form analogies subject to bidirectional constraints. What's more, they adapt patch match and nearest neighbor field strategy to reconstruct the combined image.

Gatys et al [2] show how the generic feature representations learned by Convolutional Neural Networks can be used to independently process the content and style of natural images it is a texture transfer algorithm that constrains a texture synthesis method by feature representations from Convolutional Neural Networks. Since the texture model is also based on deep image representations, the style transfer method elegantly reduces to an optimization problem within a single neural network. New images are generated by performing a pre-image search to match feature representations of example images.

Isola et al [4] introduce a general solution to image transfer problem by training the network to not only learn the feature map but also a loss function to establish this map automatically. The goal is accomplished by Generative Adversarial Networks learning an appropriate loss function for a specific task.

3 Data Description

The data set provided by Prof.Yang Wang from HKUST consists of high resolution scans of 28 paintings, scaled to a uniform density of 200 dots per painted inch. The picture size vary from 1192*748 to 6326*4457 pixels. Of the 28 paintings, 12 have been attributed to Raphael, 9 are known to be non-Raphael, and others are currently questioned by experts.

Figure 1 is taken from our data set.

In section 4, for easier studying and feature capturing, we change them into grey-scale scans. Usually, turning a colored picture into grey-scale might lose a lot of information. In this specific problem however, the pictures given are all single-colored. So grey-scale doesn't make any difference.

(a) No.8 Raphael

(b) No.11 non Raphael

Figure 1: two pictures from data set

There are numerous ways for transformation. One simple and widely used formula is from BT.601, a standard issued in 1982.

$$\textbf{Grey} = \textbf{Red} * 0.299 + \textbf{Green} * 0.587 + \textbf{Blue} * 0.114 \tag{1}$$

It's worth noting that our data set is quite small, only 21 labeled pictures. For synthesization this is ok but not enough for classification. In an attempt to enlarge this data set, we cut each labeled image into 4 parts. Thus we get 84 labeled data in total.

4 Artistic Authentication of Raphael's Paintings

4.1 Feature Engineering

Artistic authentication requires delicate analysis of specific style and habits. Although perceptual meaningfulness is a subjective concept, it can be measured with a computer by using techniques derived from biological and psychological models of the human visual system. The key point is to find the appropriate features which give a good separation between the artists' paintings and those by his imitators. Next we will give thorough illustration of two analysis tools, namely, Geometric tight frame in [9] and Gabor wavelet transformation in [5].

Geometric Tight Frame Filters & Moment Statistics

The Geometric tight frame we use has 18 filters. They can capture the first and second-order differences in the horizontal, vertical and diagonal directions in every small neighborhood of the paintings. The first 2 filters are shown below. Refer to [9] to check all the filters.

$$\tau_0 = \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tau_1 = \frac{1}{16} \times \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2}$$

Given the i-th color painting, with m_i-by-n_i pixels, we represent its grey-scale intensity by an m_i-by-n_i matrix $\mathbf{P}_i$. Convolve $\mathbf{P}_i$ with each τ_j , $0 \leq j \leq 17$, to get the corresponding m_i-by-n_i tight frame coefficient matrices $\mathbf{A}^{(i,j)}$:

$$\mathbf{A}^{(i,j)} = \mathbf{P}_i * \tau_j = \begin{pmatrix} a_{1,1}^{(i,j)} & \cdots & a_{1,n_i}^{(i,j)} \\ \vdots & & \\ a_{m_i,1}^{(i,j)} & \cdots & a_{m_i,n_i}^{(i,j)} \end{pmatrix} \tag{3}$$

Therefore, there are 18 corresponding coefficient matrices for each painting after the decomposition.In an attempt to reduce dimensionality of data set, [9] proposes the following 3 statistics as features.

(i) the mean of the entries in the coefficient matrix $\mathbf{A}^{(i,j)}$:

$$\mu^{(i,j)} = \frac{1}{m_i n_i} \sum_{l=1}^{m_i} \sum_{k=1}^{n_i} a_{l,k}^{(i,j)}$$

(ii) the standard deviation of the entries in $\mathbf{A}^{(i,j)}$:

$$\sigma^{(i,j)} = \Big(\frac{1}{m_i n_i - 1} \sum_{l=1}^{m_i} \sum_{k=1}^{n_i} (a_{l,k}^{(i,j)} - \mu^{(i,j)})^2 \Big)^{\frac{1}{2}}$$

(iii) the percentage of the "tail entries", which are entries that lies outside one standard deviation from the mean:

$$p^{(i,j)} = \#(\hat{\mathbf{A}}^{(i,j)})/(m_i n_i).$$

where $\#(\hat{\mathbf{A}}^{(i,j)})$ is the number of nonzero entries in the tail matrix $\hat{\mathbf{A}}^{(i,j)}$ which is defined by:

$$\hat{a}_{l,k}^{(i,j)} = \begin{cases} a_{l,k}^{(i,j)}, & \text{if} \quad |a_{l,k}^{(i,j)} - \mu^{(i,j)}| \geq \sigma^{(i,j)} \\ 0, & \text{otherwise} \end{cases}$$

Thus the feature vector of the i-th painting is represented by

$$[\mu^{(i,0)}, ..., \mu^{(i,17)}, \sigma^{(i,0)}, ..., \sigma^{(i,17)}, p^{(i,0)}, ..., p^{(i,17)}] \in \mathbb{R}^{54}.$$

Gabor Wavelet Filters & Energy Values

Multiscale-oriented Gabor wavelet filters are sensitive to particular wave-like patterns. The Gabor wavelet filters come in pairs: $G_{even}(x, y, \sigma, \alpha, \omega)$ and $G_{odd}(x, y, \sigma, \alpha, \omega)$ are, respectively, the real and imaginary parts of the function:

$$e^{2\pi i \omega (x sin\alpha + y cos\alpha)} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x and y are the spatial coordinates assuming the filters are centered at the origin, α sets the spatial orientation and ω the spatial frequency of the filter pair.

In theory, there are infinite number of Gabor filters. In an attempt to balance efficiency and convenience, we apply a set of filters that covers a range of orientations and scales: six orientations ($\alpha = (k/6)\pi$, for $k \in \{0, 1, 2, 3, 4, 5\}$) and four scales (manually set ω and σ).

Figure 2 from [5] visualizes 24 filters.

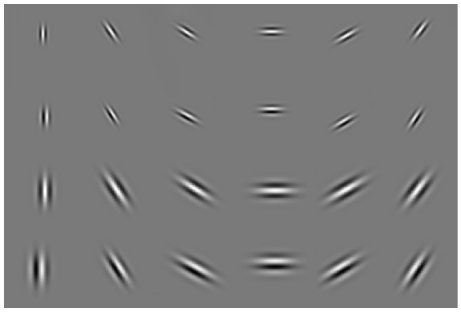


Figure 2: 24 Gabor filters

Convolving these filter with an image yields a decomposition of the image structure into "energy values", one energy value for each pixel, orientation, and scale. The Gabor wavelet energy value of an image patch is defined as the sum of the squared values obtained by convolving both components with an image patch. Figure 3 from [5] is a good illustration of the whole process.

4.2 Model Implementation

We use the following models to do feature selection and classification.

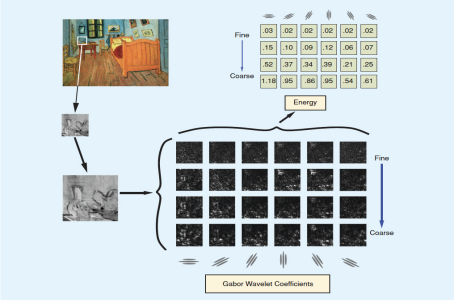


Figure 3: filters and energy value

Forward Stage-Wise Feature Selection

It is unlikely that all the 54 features obtained by Geometric frame are discriminative between Raphael and his contemporaries. If we include some noisy features in our classification task, the accuracy will deteriorate. That's why we adopt stage-wise method in [9].

To be precise, suppose there are 21(in this problem, we have 21 pictures labeled Raphael and non-Raphael). let the training dataset be $\mathcal{X} = \{\mathbf{x_1}, ..., \mathbf{x_{21}}\}$ and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_{21}} \end{bmatrix} \tag{4}$$

be the data matrix of $\mathcal{X}$and $\tilde{\mathbf{X}}$ be the normalization of $\mathbf{X}$ such that each column in $\tilde{\mathbf{X}}$ has a unit standard deviation. Let $\{1, ..., 21\} = \mathcal{T}_{vG} \cup \mathcal{T}_{nvG}$ where $\mathcal{T}_{vG}$ denotes the set of vG paintings in $\mathcal{X}$. For any feature subset $\mathcal{F}$, denote $|\mathcal{F}|$ the number of elements in the set $\mathcal{F}$ and $\mathcal{F} = \{i_1, ..., i_{|\mathcal{F}|}\}$. Define $\tilde{X}_{j\mathcal{F}} = (\tilde{X}_{ji_1}, ..., \tilde{X}_{ji_{|\mathcal{F}|}})$. Then we define the vG center w.r.t. $\mathcal{F}$ as the mean vector of vG on $\mathcal{F}$, i.e.

$$\mathbf{c}^{\mathcal{F}} = \frac{1}{|\mathcal{T}_{vG}|} \sum_{j \in \mathcal{T}_{vG}} \tilde{\mathbf{X}}_{j\mathcal{F}} \tag{5}$$

With this we define the distance between the j-th painting in $\mathcal{X}$ and the vG center $\mathbf{c}^{\mathcal{F}}$ by

$$d_j^{\mathcal{F}} = ||\tilde{\mathbf{X}}_{j\mathcal{F}} - \mathbf{c}^{\mathcal{F}}||_2 \tag{6}$$

For $\mathcal{F}$ to be a good feature set, $d_j^{\mathcal{F}}$ should be small for $j \in \mathcal{T}_{vG}$ and large for $j \in \mathcal{T}_{nvG}$ i.e. nvG should be far from the vG center and regarded as outliers.

We apply the theory of the ROC curve. Sort $\{d_j^{\mathcal{F}}\}_{j=1}^{21}$ in an ascending order. $d_{j1}^{\mathcal{F}} \leq ... \leq d_{j21}^{\mathcal{F}}$. For a given number ρ, we can use it as a binary classifier and calculate the TPR and FPR, then draw a ROC curve based on it. AUC denote area under the curve. Ideally, AUC($\mathcal{F}$)=1. We need to choose a subset $\mathcal{F}$ which maximize AUC, which can be hard to calculate. So we start from empty set $\mathcal{F}^{(0)}$ and greedy choose the next feature by

$$l_{j+1} = argmax_{l \notin \mathcal{F}^{(j)}} AUC(\mathcal{F}^{(j)} \cup \{l\}) \tag{7}$$

In this way, we are able to select our labels both fastly and accurately.

Support Vector Machine

Since we have learned this algorithm this semester, we will only briefly summarize what we have done. We use a linear model without kernel. Besides, we set the penalty coefficient equal 1.

Decision Tree

Python Machine Learning tool *scikit-learn* already has a library called *tree*. To implement this algorithm, we simply need to implement a line of Python code:

```
from sklearn import tree
```

Neutral Network

We use 1-hiden-layer neutral network with 7 hidden units, the connection between layers are liner, liner and softmax. In the Model fitting phase, we applied a dropout layer on the hidden layer(dropout probability 0.5) to avoid over fitting and we use various step size. More specifically, the model can automatically check the loss every 50 epochs, if the value not decrease for continuous 4 times, the step-size will reduce by half.

4.3 Experiments

The following are essential codes to conduct the experiment, partitioned by their applications.

4.3.1 Essential Codes

Packages to Incorporate

```
import numpy as np
import pickle
import tensorflow as tf
import sklearn
from sklearn.svm import SVC
from sklearn import neighbors
from sklearn import tree
import time
from Feature_model import Tight_frame_classifier
```

Moment Statistics and Energy Value

```
def statistics(image):
    """renturn the mean, std, precent of the tail and energy value"""
    a1 = image.mean()
    a2 = np.std(image)
    n, m = image.shape
    count = sum(sum(abs(image - a1) > a2))
    a3 = count / (m * n)
    a4 = sum(sum(image**2))
    return a1, a2, a3, a4
```

Gabor Wavelet

```
def gabor_fn(sigma, alpha, omega):
    """In our problem, sigma is std of Gaussian part,
      alpha is orientation, omega is wave frequency"""
    max = 30
    min = -max
    (y, x) = np.meshgrid(np.arange(min, max + 1), np.arange(min, max + 1))

    # Rotation
    x_alpha = x * np.cos(alpha) + y * np.sin(alpha)
    y_alpha = -x * np.sin(alpha) + y * np.cos(alpha)

    gb = np.exp(-.5 * (x ** 2 / sigma ** 2 + y ** 2 / sigma ** 2)) * np.cos(2 *
        np.pi *omega * x_alpha)
    return gb
```

Stage-Wise-Forward Feature Selection

```
def select_feature(train, label, size_feature, selected_feature=[]):
    """
```

```
    function:use stage wise forward algorithm to choose best
    n-dimension features

    input:Training matrix(list, 20*Feature_size), Training label,
    size of selected feature (integer from 1 to Feature_size),

    output:the selected feature(list with integer from 1 to Fe-
    ature_size), corresponding AUC(0 to 1), std of training
    matrix by column(np.array 1*Feature_size), mean vector of vG
   (np.array 1*Feature_size), and threshold of distance(a real number).
    """
    Feature_size = train.shape[1]
    # normalization
    std_train = train.std(axis=0)
    mean_train = train.mean(axis = 0)
    norm_train = (train-mean_train) / std_train
    norm_N = norm_train[label == 0, :]
    norm_T = norm_train[label == 1, :]
    center_true = norm_T.mean(axis=0) # mean vector of vG, i.e. vG center

    auc_select = []
    while (len(selected_feature)<=size_feature):
        index_auc_pairs = []
        unselected_feature = [i for i in range(Feature_size)
                              if i not in selected_feature] # optional feature
        for F in unselected_feature:
            temp_feature = selected_feature + [F]
            N_tempt = norm_N[:, temp_feature]
            T_tempt = norm_T[:, temp_feature]
            center_true_temp = center_true[temp_feature]
            # distance between fitures and vG center L2-distance
            dist_N = (((N_tempt - center_true_temp) ** 2).sum(axis=1)) ** 0.5
            dist_T = (((T_tempt - center_true_temp) ** 2).sum(axis=1)) ** 0.5
            dist = np.append(dist_T, dist_N)
            label_tempt = np.append(np.ones(T_tempt.shape[0]),
                                    np.zeros(N_tempt.shape[0]))
            # calculate and store AUC
            fpr, tpr, thresholds = metrics.roc_curve(
                label_tempt, dist, pos_label=0, drop_intermediate=False)
            index_auc_pairs += [[F, metrics.auc(fpr, tpr)]]
        # choose the best one feature
        index_auc = sorted(index_auc_pairs, key=lambda x: x[1])
        feature_best = index_auc[-1][0]
        auc_select = index_auc[-1][1]
        selected_feature += [feature_best] # added the best one into selected
            features

    # choose best threshold
    N = norm_N[:, selected_feature]
    T = norm_T[:, selected_feature]
    center_t = center_true[selected_feature]
    N_dis = (((N - center_t) ** 2).sum(axis=1)) ** 0.5
    T_dis = (((T - center_t) ** 2).sum(axis=1)) ** 0.5
    temp_label = np.append(np.ones(T.shape[0]), np.zeros(N.shape[0]))
    dist = np.append(T_dis, N_dis)
    accuracy = []
    for j in range(len(dist)): # test difference threshold
        thre = dist[j]
        predict = dist.copy()
        predict[dist > thre] = 0
        predict[dist <= thre] = 1
        accuracy += [accuracy_score(temp_label, predict)]
    tempt = np.argsort(accuracy)
    thres = np.mean(dist[tempt[-2:]]) # the best threshold
    return selected_feature, auc_select, mean_train,std_train, center_true, thres
```

Python Class for Tight Frame classifier

```
class Tight_frame_classifier(object):
    def __init__(self,best_size,Feature,mean_train,SD,Center,Threshold):\
        # store the best parameters of every possible dimensions of features
        self.selected_feature = Feature
        self.mean_train = mean_train
        self.std_train = SD
        self.center_true = Center
        self.thres = Threshold
        self.best_size = best_size

    def best_feature(self):
        return self.selected_feature[:self.best_size]

    def predict(self,test,feature_size = None):
        if (feature_size == None) :
            feature_size = self.best_size
        if (type(test[0]) == np.float):
            test = [test]
        prediction = []
        sd = self.std_train[feature_size-1]
        center = self.center_true[feature_size-1]
        selected_feature = self.selected_feature[:feature_size]
        thres = self.thres[feature_size-1]
        for ite in test:
            T = (ite-self.mean_train) / sd
            T = T[selected_feature]
            center_true_temp = center[selected_feature]
            dist = (((T - center_true_temp) ** 2).sum()) ** 0.5
            prediction += [dist < thres]
        return prediction
```

4.3.2 Results and Comparisons

Remember that in order to enlarge our data set, we have cut each labeled image into 4 parts and get 84 labeled image. Then we randomly split our data into training set and test set (use Python sklearn package). We adopt leave-one-out cross-validation on training set to select features, and tabular 1 shows our results.

Table 1: Experiment Results

Feature Extraction	model	TPR	TNR	Classification Accuracy
Tight Frame	Forward Stage wise	94.5%	34.7%	67.6%
	SVM	66.2%	80.9%	72.8%
	Decision Tree	75.6%	92.4%	83.2%
	KNN	47.3%	80.9%	62.4%
	Neural Network	76.2%	92.4%	84.0%
Gabor Wavelet	Forward Stage wise	78.0%	78.0%	78.0%
	SVM	100%	0	60%
	Decision Tree	95.3%	78.0%	88.4%
	KNN	95.3	0	57.2%
	Neural Network	95.3%	82.4%	90.1%

Analysis

The overall performance based on Gabor Wavelet seems better than Tight Frame filters, in that both the best classification method(Decision tree model) and Forward Stage Wise method have higher accuracy score. It's safe to conclude that Gabor Wavelet filters do a good jog in feature extraction.

However, results of SVM & KNN based on Tight Frame features are better than that of Wavelet features. One possible explanation is that both these two models' classification rule is based on distance, while the selection of Geometric Tight features are also based on distance.

Further Discussion: Gabor wavelet parameters

Gabor wavelet is used in our report to extract of wave-like patterns. Different pictures may have different scale, i.e. the reliability of our method largely depends on consistency of photography techniques. Thanks to our specially made data set, with a uniform density of 200 dots per painted inch, we don't need to worry about scale.

In our report, we manually check the scale of our data set and find that the stroke is around 5 to 20 pixels in width, ranging from the finest to coarsest. So we set a suitable parameter to ensure that all the wave lengths roughly match the stroke width.

5 Synthesization of Style Transferred Images by Raphael's Artworks

The main idea for the task is to extract the low level features like the texture and color from one image which is called the *style image* $\mathbf{s}$, and apply it to some more semantic and high level features on another *content image* $\mathbf{c}$ to derive the *style-transferred image* $\mathbf{x}$. Roughly speaking, the *output image* $\mathbf{x}$ should be the one preserving the objects and concepts in the *content image* $\mathbf{c}$ whilst deviating little from the *style image* $\mathbf{s}$ in terms of the style.

We intend to practice the mechanism depicted in [2] instead of [8] because the style and content images we want to extract feature from differ much both in their genre and semantic structure while the latter is an important requirement for the image analogy manipulation in [8].

So intuitively, we can introduce a loss function to measure the distinction between $\mathbf{x}$ and $\mathbf{s}$ & $\mathbf{c}$, all of which are rearranged as column vectors. All we need to do then is to produce the *style-transferred image* which minimizes the loss function. Consequently, the original style transfer objective reduces to an optimization problem:

$$\mathbf{x}^* = \underset{\mathbf{x}}{argmin} \left(\alpha \mathcal{L}_{content}(\mathbf{c}, \mathbf{x}) + \beta \mathcal{L}_{style}(\mathbf{s}, \mathbf{x})\right) \tag{8}$$

where α and β are two hyper-parameters that allow us to control how much emphasis we want to put on the content relative to the style, and $\mathcal{L}_{content}(\mathbf{c}, \mathbf{x})$, $\mathcal{L}_{style}(\mathbf{s}, \mathbf{x})$ are the content and style loss functions separately.

Now the crucial insight in [2] is that the definitions of these content and style losses are based not on pixel loss which is the Euclidean distance between the output $\mathbf{x}$ and the target $\mathbf{c}$ or $\mathbf{s}$, but instead in terms of some more perceptual differences between them. Such *semantic difference* are learnt by a convolutional neural network.

5.1 Feature Representations

It turns out that convolutional neural networks pre-trained for image classification have already learnt to encode perceptual and semantic information that we need to grasp the semantic differences. The key point of [2] is that we can apply the VGGNet model pre-trained on the *ImageNet* data set so that higher layers in the network capture the high level content in terms of objects and their arrangement in the input image (Figure 4). The feature representations of style and content are separable and the output image is synthesized by finding the one that simultaneously matches the content and the style representation based on the loss functions defined by the activations of the network.

We shall first introduce some denotation for the structure of the VGG network. The network we intend to work on is the 16-layer VGG network. Let $\phi^j(x)$ be the activations of the j-th layer of the network ϕ when processing the image x. From [2], the content and style loss functions can be defined as follows.

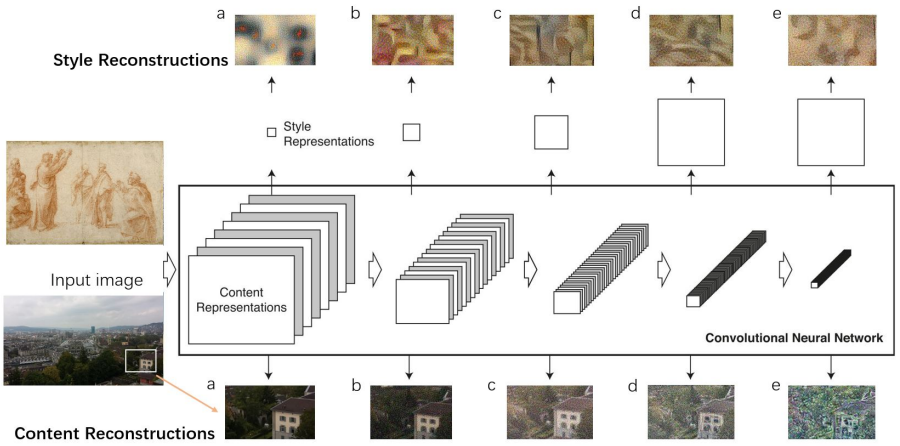


Figure 4: Image representation in a CNN. An input image is represented as a set of filtered images at each layer of CNN. The size of these filtered images is reduced by some down-sampling mechanism(e.g. pooling). To visualise the information in each layer, we reconstructed these filtered images.

5.2 Content Loss

For the content loss, the original choice in [2] is computed on the 4-th convolutional layer to be the squared Euclidean distance between feature representations of the content and combination images:

$$\mathcal{L}_{content}(\mathbf{c}, \mathbf{x}) = (\phi^j(\mathbf{c}) - \phi^j(\mathbf{x}))^2 \tag{9}$$

where $j = 4$. However, as stated above, the higher layer may deprive too much structure details of the input images and we will do experiments later to see which layer would be the most aesthetic choice.

5.3 Style Loss

The style representation will be more involved. According to [2], we first define a *Gram matrix* $G^l(x)$ which is proportional to the covariances of corresponding sets of features in the l-th layer. The elements of $G^l(x)$ is defined as:

$$G^l_{ij}(x) = \frac{1}{M^l} \sum_{k=1}^{M^l} \phi^l_{ik}(x)\phi^l_{jk}(x) \tag{10}$$

where ϕ^l_{ik} denotes the k-th position of the i-th vectorized feature map in the l-th layer and M^l is the number of elements in each feature map. The loss function of style representation in the l-th layer is then defined as the squared Frobenius norm of the difference between the Gram matrices of the style and combination images:

$$\mathcal{L}^l_{style}(\mathbf{s}, \mathbf{x}) = ||G^l(\mathbf{s}) - G^l(\mathbf{x})||^2_F. \tag{11}$$

And the total style loss is a weighted sum of the contribution from each layer:

$$\mathcal{L}_{style}(\mathbf{s}, \mathbf{x}) = \sum_{l=1}^{L} w^l \mathcal{L}^l_{style}. \tag{12}$$

To see why the Gram matrix works, we refer to the paper [1]. It is said that due to the fact that the style is static and uniform across the image, the Gram matrix is used to capture global statistics by averaging over positions. Besides, coherence among multiple features (computed by different maps) needs to be preserved to capture information about which features tend to activate together, which is measured by the off-diagonal terms in the Gram matrix.

5.4 Total Variation Regularization

If we are going to solve the optimization problem with only the two loss terms defined above, the output synthesized image would be quite noisy. According to the articles [6] and [10], we can add a regularization term called the total variation loss to encourage spatial smoothness.

The total variation encourages images to consist of piece-wise constant patches. For continuous functions $f : \mathbb{R}^2 \supset \Omega \to \mathbb{R}$, the total variation is given by:

$$\mathcal{R}(f, \delta) = \int_{\Omega} \left[\left(\frac{\partial f}{\partial u}(u, v) \right)^2 + \left(\frac{\partial f}{\partial v}(u, v) \right)^2 \right]^{\frac{\delta}{2}} \mathrm{d}u\mathrm{d}v. \tag{13}$$

For input images $\mathbf{x} \in \mathbb{R}^{H \times W}$, the total variation is replaced by the finite discrete approximation:

$$\mathcal{R}(\mathbf{x}, \delta) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\delta}{2}} \tag{14}$$

where δ is modified to balance the sharpness of the image against the removal of the artifacts called "spikes". According to the paper *Fast image deconvolution using hyper-laplacian priors*, regularizer with $\delta < 1$ is often used as a better match of the gradient statistics of natural images, while by choosing $\delta > 1$ changes in images are distributed across regions rather than concentrated at a point or curve.

In addition, when the target of the reconstruction is a colour image, the regulariser is summed for each colour channel.

5.5 Total Loss Function

Based on the previous knowledge, the total loss function is then refined as:

$$\mathcal{L}(\mathbf{c}, \mathbf{s}, \mathbf{x}) = \alpha \mathcal{L}_{content}(\mathbf{c}, \mathbf{x}) + \beta \mathcal{L}_{style}(\mathbf{s}, \mathbf{x}) + \gamma \mathcal{R}(\mathbf{x}, \delta) \tag{15}$$

where α, β, γ and δ are four hyper-parameters we will check during the experiments. And the objective is then to find the solution:

$$\mathbf{x}^* = \underset{\mathbf{x}}{argmin}\, \mathcal{L}(\mathbf{c}, \mathbf{s}, \mathbf{x}). \tag{16}$$

5.6 Experiments

5.6.1 Essential Codes

The following are essential codes to conduct the experiment, partitioned by their applications.

Packages to Incorporate

```
import numpy as np
import time
import scipy
from PIL import Image
from os import listdir, mkdir, getcwd
from keras.applications.vgg16 import VGG16
from keras import backend as K
```

Content Loss

```
def content_loss(content, combination):
    return K.sum(K.square(combination - content))
```

Style Loss

```
def gram_matrix(x):
    features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram

def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = height * width
    return K.sum(K.square(S - C)) / (4. * (channels ** 2) * (size ** 2))
```

Total Variation Loss

```
def total_variation_loss(x):
    a = K.square(x[:, :width-1, :height-1, :] - x[:, 1:, :height-1, :])
    b = K.square(x[:, :width-1, :height-1, :] - x[:, :width-1, 1:, :])
    return K.sum(K.pow(a + b, 1.5))
```

Total Loss Function

```
def total_loss(model, combination_image):
    loss = K.variable(0.)

    layers = dict([(layer.name, layer.output) for layer in model.layers])
    layer_features = layers['block2_conv2']
    content_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]

    # content loss
    loss += content_weight * content_loss(content_image_features,
                                          combination_features)

    # style loss
    feature_layers = ['block1_conv2', 'block2_conv2',
                      'block3_conv3', 'block4_conv3']

    for layer_name in feature_layers:
        layer_features = layers[layer_name]
        style_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_features, combination_features)
        loss += (style_weight / len(feature_layers)) * sl

    # total variation loss
    loss += total_variation_weight * total_variation_loss(combination_image)
    return loss # return total loss
```

Optimization

```
def eval_loss_and_grads(x):
    # @x: the combination image.
    x = x.reshape((1, width, height, 3))

    outputs = [loss]
    outputs += grads
    f_outputs = K.function([combination_array], outputs)
```

```
    outs = f_outputs([x])
    loss_value = outs[0]
    grad_values = outs[1].flatten().astype('float64')
    return loss_value, grad_values

def minimize_loss(combination):
    '''
    Using stochastic gradient descent, via fmin_l_bfgs_b algorithm.
    @combination: concatenated name of the two images used.
    '''
    x = np.random.uniform(0, 255, (1, width, height, 3)) - 128
    evaluator = Evaluator()

    print("\n\nProcessing: " + combination)
    for i in range(iterations):

        # print diagnostic information
        print('Start of iteration', i)
        start_time = time.time()
        x, min_val, info = scipy.optimize.fmin_l_bfgs_b(evaluator.loss,
                                                        x.flatten(),
                                                        fprime=evaluator.grads,
                                                        maxfun=20)
        print('Current loss value:', min_val)
        end_time = time.time()
        print('Iteration %d completed in %ds' % (i, end_time - start_time))
    return x
```

We will perform experiments on two sets of images: one is the works of Raphael as style images and the other is the test images, mostly landscape photographs served as content sources. First review the architectures of VGG networks. Since we are not interested in the classification problem, we don't

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 5: VGG Network Architectures

need the fully-connected layers or the final softmax classifier. We only need the part of the model marked green in the table.

Besides, when decoding the original style image into a vector, we find that the artworks are store in 4 channels, the last of which is the alpha compositing that represents the transparency. We will only use the first 3 channels to make the dimension consistent with the content image. Also, to speed up the reconstruction procedure, the original content image is condensed to a smaller size to relieve the computation burden. Moreover, if the height of style image is larger than its width, we will rotate it to fit the size of the content image.

The article [2] synthesizes the images by matching the content representation on layer 'conv4_2' and the style representation on layers 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1'. We now implement this model with settings $\alpha/\beta = 1 \times 10^{-3}$, $\gamma = 1$ and $\delta = 1$ to have a glance of the combined output. The combination image $\mathbf{x}$ begins its life as a white noise one and we use the L-BFGS algorithm to iteratively improve upon it.

When we use the Raphael's artwork numbered 2 as the style image, the combination image is produced in figure 6.

(a) (b)

Figure 6: Images that combine the content of a photograph with the style of Raphael's artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork. The ratio $\alpha/\beta = 10^{-3}$. The left one is the style image and the right one is the combination image.

Trade-off between Content and Style Representations When synthesizing an image that combine the content of one image with the style of another, there usually does not exist an image that perfectly matches both of them. However, since the loss function we minimise during image synthesis is a linear combination between the loss functions for content and style respectively, we can smoothly regulate the emphasis on either reconstructing the content or the style [2]. A high ratio α/β will result in images that effectively capture the texture of the style artwork but showing hardly any of the photograph's content.

Figure 7 shows the increasingly explicit content outline in the combination image by using larger ratios. The combination image is produced after 10 iterations of the algorithm. It is apparent that the output with ratio $\alpha/\beta = 10^{-2}$ produces the most aesthetic one that simultaneously maintains the content and transfers the style based on the other hyper-parameters set previously.

In these results it is clear that the pre-trained network is aware of the semantic content of images, especially objects like the parking cars. This is partly because the VGG networks are basically trained on the classification data set comprised of particular elements. So the style transfer will preserve these features more than the background [6].

5.6.2 Effect of Different Layers for Content and Style Representation

Another consideration in the image synthesizing process is the choice of layers for the extraction of the style and content. According to [2], using the content representation in higher layers in the

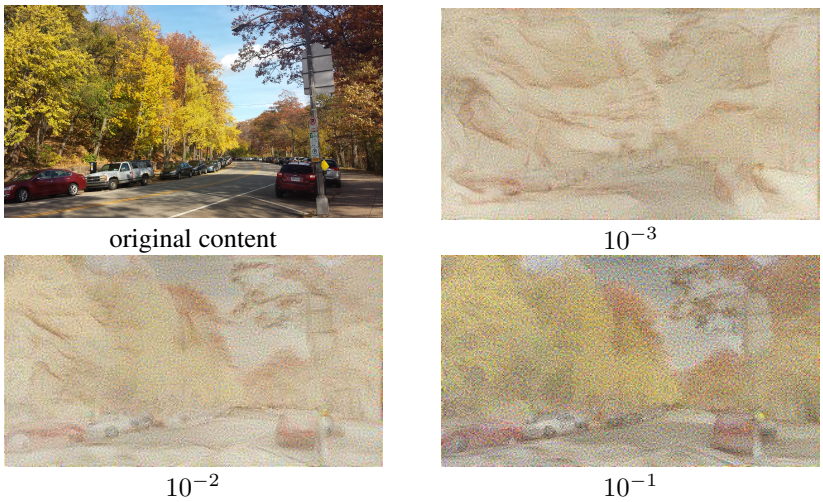


Figure 7: Output images synthesized by different settings of the ratio α/β. A high emphasis on the style produces combination difficult to recognize the content but excessively rich in the target style. A best choice for the model seems to be $\alpha/\beta = 10^{-2}$.

network decreases the object detail while matching the style representations up to higher layers preserves local textures within an increasingly large scale.

The previous experiment is conducted on the layer 'conv4_2' for content representation and 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' for style representation. Alternatively, we perform the style transfer with the same artwork and parameter configuration ($\alpha/\beta = 10^{-2}$) but matching the photograph on layer 'conv2_2'. Figure 8 demonstrates the comparison between this kind of difference.

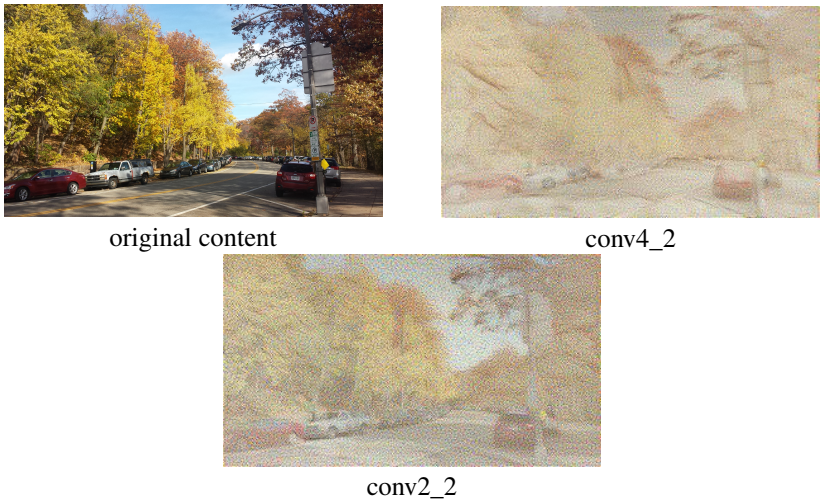
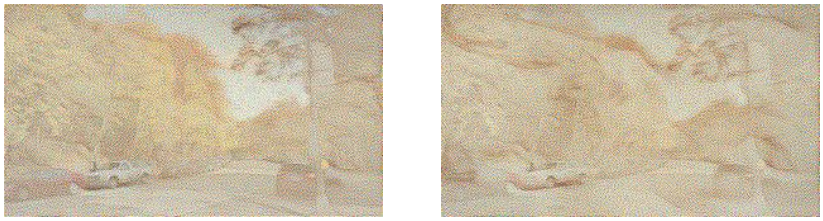


Figure 8: Matching the content on layer 'conv2_2' produces the combination image with more details but seems to hardly capture the style where the ratio $\alpha/\beta = 10^{-2}$.

A more aesthetic choice by *Johnson et al.* is that we further replace the layer for style representation by 'conv1_2', 'conv2_2', 'conv3_3' and 'conv4_3'. The result better expresses the texture of the style image since we are performing the style representation on deeper layers which incorporate information from an increasingly larger spatial extent. We also change the ratio α/β expecting for a better balance between the content and style (Figure 9).

However, it looks like the style over the combination image is still somewhat beyond recognition. One explanation is that the style and content image differ greatly in their characteristics. For example,



Johnson, $\alpha/\beta = 10^{-2}$ *Johnson*, $\alpha/\beta = 10^{-3}$

Figure 9: Reconstruction of the combination image with alternative layers for the style representation.

one is a sketch painting with sober beige color while the other is a colorful real-life photograph so that the texture and color from the style image may get blended with the content during the reconstruction process.

5.6.3 Selection of the Power Parameter δ of the Total Variation Regularization

To see how the total variation regularization term affects the combination image, we first draw a simple comparison between the output with and without this term. We will again use the style transfer structure performed in [6] and the ratio $\alpha/\beta = 10^{-2}$ and the power parameter $\delta = 1$. The regularization coefficient γ is set to 1. Figure 10 illustrates of the total variation regularization effect while the result seems to indicate the presence of the regularization term is out of usage. However, as we gradually increment the value of δ, the power parameter, the dramatic improvement will impress us.

original content original style

$\gamma = 0$ $\gamma = 1$

Figure 10: Reconstruction of style transferred image with and without total regularization when $\delta = 1$ shows the regularization produces little effect on removal of the 'spikes' which are sharp fluctuation of RGB value causing a discrete impression of image.

Figure 11 shows the effect of increasing value of δ. It is obvious that the combination image becomes smoother and abnormal background transients disappear with larger values of δ, exactly what we desire. Future experiment and discussion can be done to find the optimal option for γ, the regularization coefficient. [10] presents us with a concrete theory of method to implement. Also, the regularization term can be defined as the α-norm $\mathcal{R}(\mathbf{x}) = ||\mathbf{x}||_{\alpha}^{\alpha}$. By choosing a relatively large exponent the range of the image is encouraged to stay within a target interval instead of diverging.

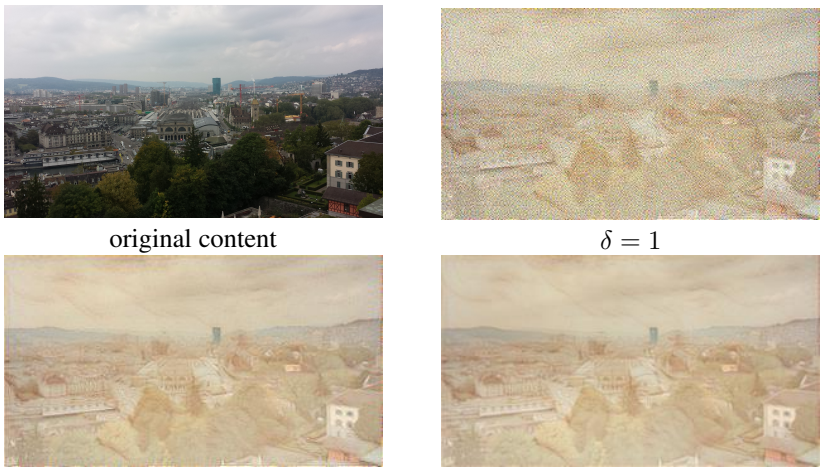


Figure 11: Increasing value of δ can bring about more spatial smoothness.

However, due to the limit of our personal laptops' computation ability (without GPU), we can only present a low-resolution combination image within a reasonable amount of time (i.e. one iteration of the output with 300 pixels of width costs over 10 minutes). So future improvement of this project will be focusing on finding a more efficient and time-saving algorithm, for example to build the style and content representation on the network architecture described in [12] and [3] and the feed-forward network in [6].

6 Conclusion

In task1, we use Tight Frame and Wavelet transform method to extract brushwork features, then use leave-one-out cross validation and stage-wise-forward algorithm to select features. After that, several models are used. From experiment result, it's obvious different feature extraction method significantly influence the efficiency of model and the dimension of features matters as well.

Of all models we trained in previous part, we did classification on the 7 pictures remained disputed (Pic 1/7/10/20/23/25/26). Basd on both *Tight Frame* & *Wavelet* features, most experiments show that Picture 1/7/23 are genuine, and Picture 10/25/26 are counterfeit. For Picture 23, predict results are various, so we have reservation about it.

In task2, to generate images with mixed content and style representations from two unrelated images, we used well-trained CNN network and *Gram matrix* to calculate the representations of content and style. We generate a new image from one noisy picture which minimize the pre-defined loss function. To enhance the effects of this model, we discussed different influence factors, including trade off between *style* and *content*, influence of different CNN layer etc.

References

[1] G. Berger and R. Memisevic. Incorporating long-range consistency in cnn-based texture generation. *CoRR*, abs/1606.01286, 2016.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[5] C. R. Johnson, E. Hendriks, I. Berezhnoy, E. Brevdo, S. Hughes, I. Daubechies, J. Li, E. Postma, and J. Z. Wang. Image processing for artist identification – computerized analysis of vincent van gogh's painting brushstrokes. *IEEE Signal Processing Magazine*, July 2008.

[6] J. Johnson, A. Alahi, and F. Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[7] J. Li, L. Yao, E. Hendriks, and J. Z. Wang. Rhythmic brushstrokes distinguish van gogh from his contemporaries: Findings via automated brushstroke extraction. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(6):1159–1176, 2012.

[8] J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang. Visual attribute transfer through deep image analogy. *CoRR*, abs/1705.01088, 2017.

[9] H. Liu, R. H. Chan, and Y. Yao. Geometric tight frame based stylometry for art authentication of van gogh paintings. *CoRR*, abs/1407.0439, 2014.

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

[11] H. Qi, A. Taeb, and S. M. Hughes. Visual stylometry using background selection and wavelet-hmt-based fisher information distances for attribution and dating of impressionist paintings. *Signal Processing*, 93(3):541–553, 2013.

[12] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.