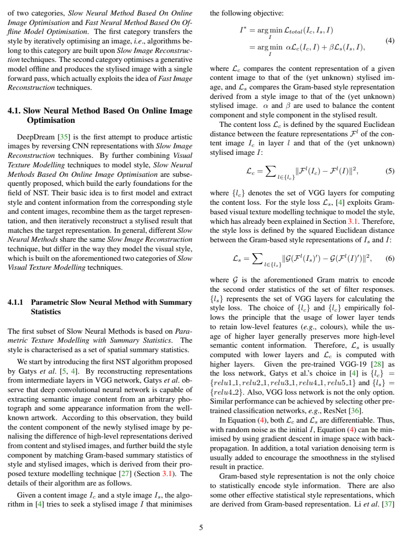
Hello World!



of two categories, *Slow Neural Method Based On Online Image Optimisation* and *Fast Neural Method Based On Offline Model Optimisation*. The first category transfers the style by iteratively optimising an image, *i.e.*, algorithms belong to this category are built upon *Slow Image Reconstruction* techniques. The second category optimises a generative model offline and produces the stylised image with a single forward pass, which actually exploits the idea of *Fast Image Reconstruction* techniques.

4.1. Slow Neural Method Based On Online Image Optimisation

DeepDream [35] is the first attempt to produce artistic images by reversing CNN representations with *Slow Image Reconstruction* techniques. By further combining *Visual Texture Modelling* techniques to model style, *Slow Neural Methods Based On Online Image Optimisation* are subsequently proposed, which build the early foundations for the field of NST. Their basic idea is to first model and extract style and content information from the corresponding style and content images, recombine them as the target representation, and then iteratively reconstruct a stylised result that matches the target representation. In general, different *Slow Neural Methods* share the same *Slow Image Reconstruction* technique, but differ in the way they model the visual style, which is built on the aforementioned two categories of *Slow Visual Texture Modelling* techniques.

4.1.1 Parametric Slow Neural Method with Summary Statistics

The first subset of Slow Neural Methods is based on *Parametric Texture Modelling with Summary Statistics*. The style is characterised as a set of spatial summary statistics.

We start by introducing the first NST algorithm proposed by Gatys *et al*. [5, 4]. By reconstructing representations from intermediate layers in VGG network, Gatys *et al*. observe that deep convolutional neural network is capable of extracting semantic image content from an arbitrary photograph and some appearance information from the well-known artwork. According to this observation, they build the content component of the newly stylised image by penalising the difference of high-level representations derived from content and stylised images, and further build the style component by matching Gram-based summary statistics of style and stylised images, which is derived from their proposed texture modelling technique [27] (Section 3.1). The details of their algorithm are as follows.

Given a content image I_c and a style image I_s, the algorithm in [4] tries to seek a stylised image I that minimises the following objective:

$$\begin{aligned} I^* &= \arg\min_I \mathcal{L}_{total}(I_c, I_s, I) \\ &= \arg\min_I \ \alpha\mathcal{L}_c(I_c, I) + \beta\mathcal{L}_s(I_s, I), \end{aligned} \tag{4}$$

where $\mathcal{L}_c$ compares the content representation of a given content image to that of the (yet unknown) stylised image, and $\mathcal{L}_s$ compares the Gram-based style representation derived from a style image to that of the (yet unknown) stylised image. α and β are used to balance the content component and style component in the stylised result.

The content loss $\mathcal{L}_c$ is defined by the squared Euclidean distance between the feature representations $\mathcal{F}^l$ of the content image I_c in layer l and that of the (yet unknown) stylised image I:

$$\mathcal{L}_c = \sum\nolimits_{l\in\{l_c\}} \|\mathcal{F}^l(I_c) - \mathcal{F}^l(I)\|^2, \tag{5}$$

where $\{l_c\}$ denotes the set of VGG layers for computing the content loss. For the style loss $\mathcal{L}_s$, [4] exploits Gram-based visual texture modelling technique to model the style, which has already been explained in Section 3.1. Therefore, the style loss is defined by the squared Euclidean distance between the Gram-based style representations of I_s and I:

$$\mathcal{L}_s = \sum\nolimits_{l\in\{l_s\}} \|\mathcal{G}(\mathcal{F}^l(I_s)') - \mathcal{G}(\mathcal{F}^l(I)')\|^2, \tag{6}$$

where $\mathcal{G}$ is the aforementioned Gram matrix to encode the second order statistics of the set of filter responses. $\{l_s\}$ represents the set of VGG layers for calculating the style loss. The choice of $\{l_c\}$ and $\{l_c\}$ empirically follows the principle that the usage of lower layer tends to retain low-level features (*e.g.*, colours), while the usage of higher layer generally preserves more high-level semantic content information. Therefore, $\mathcal{L}_s$ is usually computed with lower layers and $\mathcal{L}_c$ is computed with higher layers. Given the pre-trained VGG-19 [28] as the loss network, Gatys et al.'s choice in [4] is $\{l_c\} = \{relu1\_1, relu2\_1, relu3\_1, relu4\_1, relu5\_1\}$ and $\{l_s\} = \{relu4\_2\}$. Also, VGG loss network is not the only option. Similar performance can be achieved by selecting other pre-trained classification networks, *e.g.*, ResNet [36].

In Equation (4), both $\mathcal{L}_c$ and $\mathcal{L}_s$ are differentiable. Thus, with random noise as the initial I, Equation (4) can be minimised by using gradient descent in image space with backpropagation. In addition, a total variation denoising term is usually added to encourage the smoothness in the stylised result in practice.

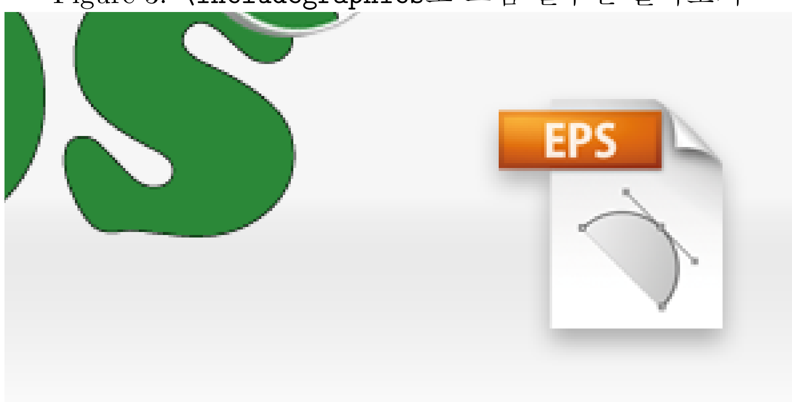
Gram-based style representation is not the only choice to statistically encode style information. There are also some other effective statistical style representations, which are derived from Gram-based representation. Li *et al*. [37]

5

Figure 1: `\includegraphics` 명령으로 부른 그램(축소/확대)

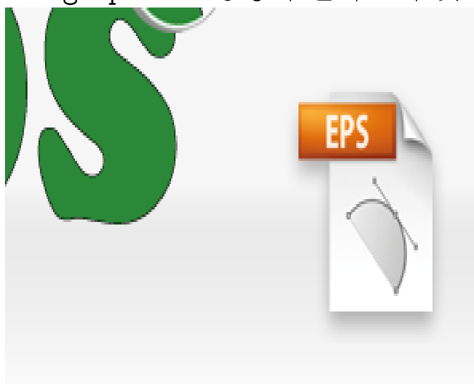
Figure 2: \includegraphics 명령의 그림 크기 조절

Figure 3: \includegraphics로 그림 일부만 잘라오기



오른쪽 그림과 같이, 함수 $f(x) = x^3 - x^2$의 그래프 위의 점 $(a_0, f(a_0))$에서 접선을 긋고 (단 $a_0 > 3$) x축과의 교점을 $(a_1, 0)$이라 한다. 다음에 점 $(a_1, f(a_1))$에서 접선을 긋고 x축과의 교점을 $(a_2, 0)$ 이라 한다. 이러한 방법으로 계속하여 일반적으로 점 $(a_{n-1}, f(a_{n-1}))$에서 접선을 긋고 x축과의 교점을 $(a_n, 0)$이라 한다. 이때 다음 물음에 답하여라.

Figure 4: `\includegraphics*` 명령의 잘라오기 및 그림 크기 변경



captionPostScript 그림 `eps.png`의 원본 및 반시계 방향으로 30 °회전한 결과

1. a_n을 a_{n_1}의 식으로 나타내어라($n = 1, 2, \cdots$).
2. $a_0 > a_1 > a_2 > \cdots > a_n > \cdots \geq \sqrt{3}$이 됨을 보여라.

왼쪽 문자–  –오른쪽 문자

Figure 5: `*` 있는 `\includegraphics*` 명령

Figure 9: `floatingfigure` 환경의 예 2

Hello wolrd!

외쪽 문자–


–오른쪽 문자

Figure 6: * 없는 \includegraphics 명령

(a) 원 그래프

(b) 막대 그래프

Figure 7: 원 그래프와 막대 그래프

(a) 회전 (b) 원래 크기

Figure 8: `subfigure` 패키지를 이용한 그림 1

Hello world!!!!

option	사용 결과
`l`	그림을 왼쪽에 넣을 경우에 설정한다.
`r`	그림을 오른쪽에 넣을 경우에 설정한다.
`p`	그림을 짝수면에는 왼쪽에 홀수면에는 오른쪽에 넣을 경우에 설정한다. (기본값)
`v`	`\usepackage` 명령에서 정해준 옵션을 따른다.

Table 1: `floatingtable` 환경의 옵