Neural Style Transfer

Progetto per il corso di Machine Learning

Autore: Andrea Baroni

Obiettivo

- Date due immagini S e C che rappresentano lo stile e il contenuto, studiare algoritmi per produrre in output un immagine O che abbia lo stile di S e il contenuto di O.

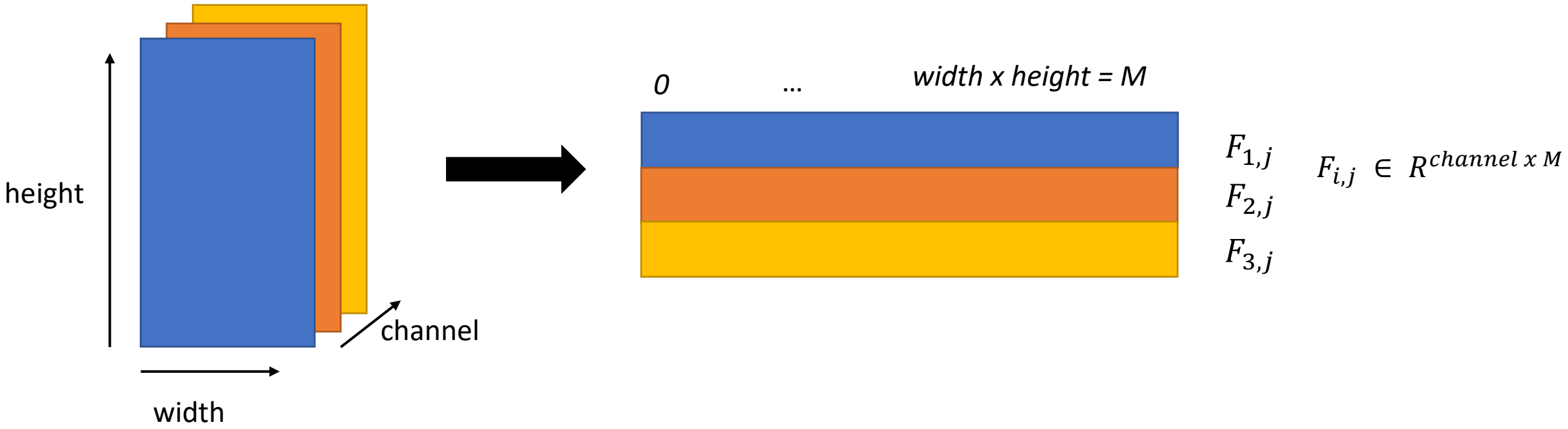
- Per affrontare il problema, sono state prese in considerazione due tecniche, per entrambe è stata studiata l'implementazione tramite Tensorflow 2.0.

1. L. A. Gatys, A. S. Ecker and M. Bethge, *"Image Style Transfer Using Convolutional Neural Networks,"* 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2414-2423, doi: 10.1109/CVPR.2016.265.

2. X. Huang and S. Belongie, *"Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization,"* 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1510-1519, doi: 10.1109/ICCV.2017.167.

Gatys et Al.

- L'articolo propone un metodo effettivo per l'estrazione del contenuto e dello stile da un' immagine avvalendosi di una rete neurale convoluzionale pre-addestrata.
- Si è dimostrato che ad ogni livello delle rete vengono codificate informazioni diverse man mano più complesse.
- Ogni livello contiene n attivazioni di dimensione *M = H x W,* risultato della convoluzione di n filtri.
- Le attivazioni in un particolare livello *l* possono essere memorizzate in una matrice $F^l \in R^{channel\ x\ M}$

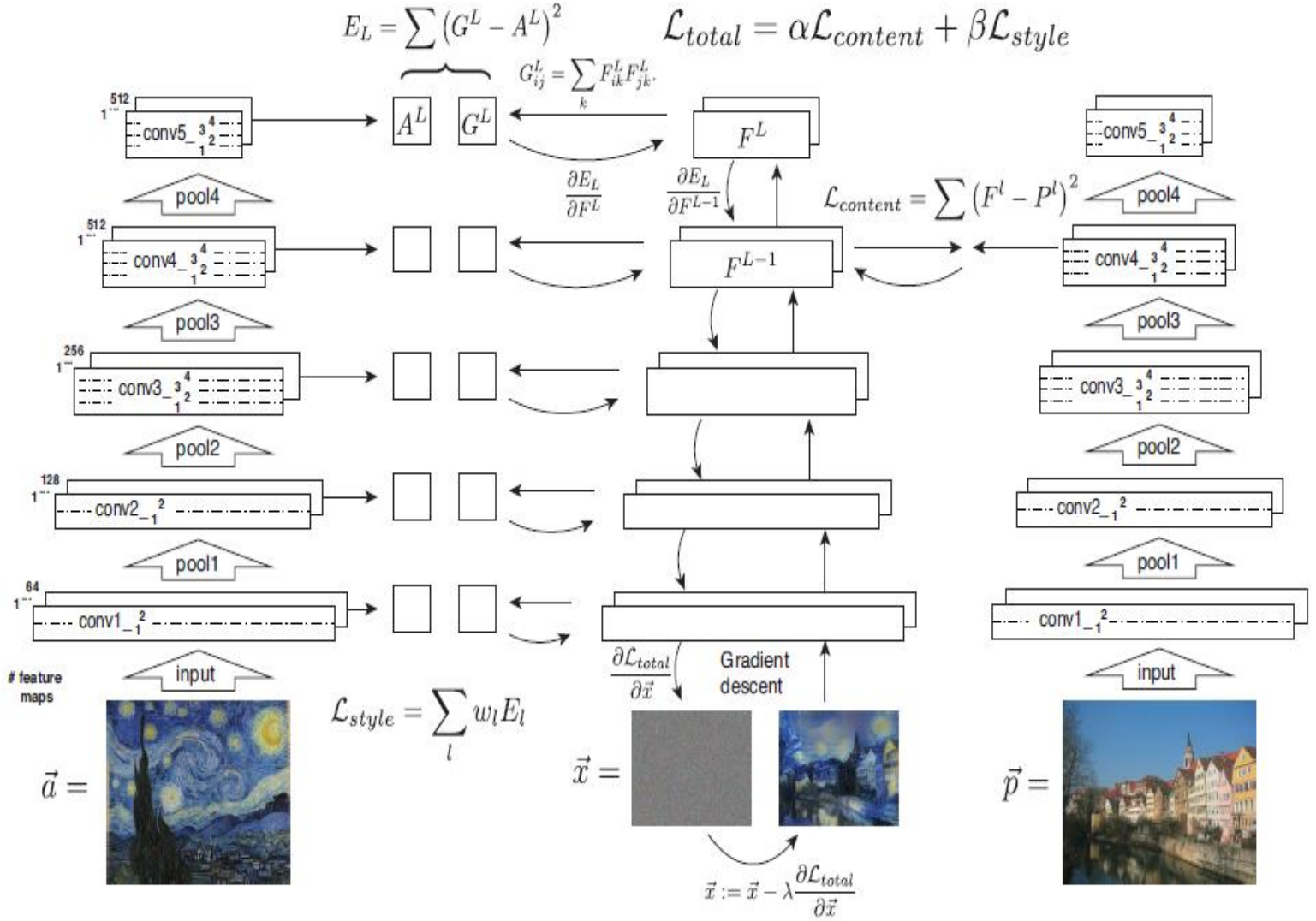


Gatys et Al.

- Per visualizzare il contenuto appreso da una rete in un particolare livello, per una data immagine in input, è possibile applicare la discesa del gradiente ad una immagine casuale in modo tale che l'immagine casuale dopo n iterazioni, veda avvicinarsi le proprie attivazioni a quelle dell'immagine in input.
- Per codificare il contenuto si sceglie un particolare livello della rete, si sceglie una funzione di perdita e si applica la discesa del gradiente alla funzione di perdita rispetto ai pixel dell'immagine da sintetizzare.
- Sia $\vec{p}$ e $\vec{x}$ l'immagine originale e l'immagine che si vuole sintetizzare e siano $F_{i,j}$ e $P_{i,j}$ le attivazioni in forma matriciale come descritto nella slide precedente.
- La funzione di perdita rispetto al contenuto è la seguente: $L_{content} = \frac{1}{2} \sum_{i,j} (F_{i,j} - P_{i,j})^2$
- Per vincolare $\vec{x}$ ad avere il contenuto di $\vec{p}$ si applicano modifiche ai pixel di $\vec{x}$ in accordo alla funzione di perdita in modo tale che le attivazioni di $\vec{x}$ e $\vec{p}$ si avvicinino.

Gatys et Al.

- Per codificare lo stile si procede in modo analogo a quanto fatto per il contenuto prendendo la seguente funzione di perdita:
- $E_l = \frac{1}{4\,N_l^2 M_l^2} \sum_{i,j} (G_{i,j} - A_{i,j})^2$
- dove $G_{i,j}$ è la matrice di Gram definita nel seguente modo: $G_{i,j} = \sum_k F_{i,k}\, F_{j,k}$.
- G ed A sono rispettivamente le matrici di Gram dell'immagine di cui si vuole imitare lo stile e dell'immagine da sintetizzare.
- Per ottenere una rappresentazione veritiera dello stile si prende la somma pesata di E_l in vari livelli.



Gatys et Al. – Implementazione pesi imagenet

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
```

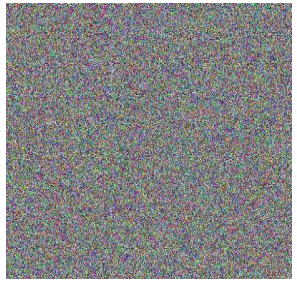
Livelli impiegati per la modellazione dello stile

Livello impiegato per la modellazione del contenuto

Gatys et Al. – Risultati ottenuti

contenuto

stile



Img da sintetizzare

Iter. : 200

Iter. : 400

Iter. : 600

Iter: 7800
Tempo 24 min,
colab gpu

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

- Il metodo precedente ha lo svantaggio di richiedere molto tempo per ottenere un risultato.
- E' emersa l'esigenza di addestrare una rete neurale per svolgere il trasferimento di stile.
- Il risultato sorprendente dell'articolo è che la media e la varianza (quindi la deviazione standard) delle attivazioni (calcolate indipendentemente per ciascun canale e per ciascun *batch*) sono sufficienti a modellare lo stile di un immagine.

$$\mu_{nc} = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

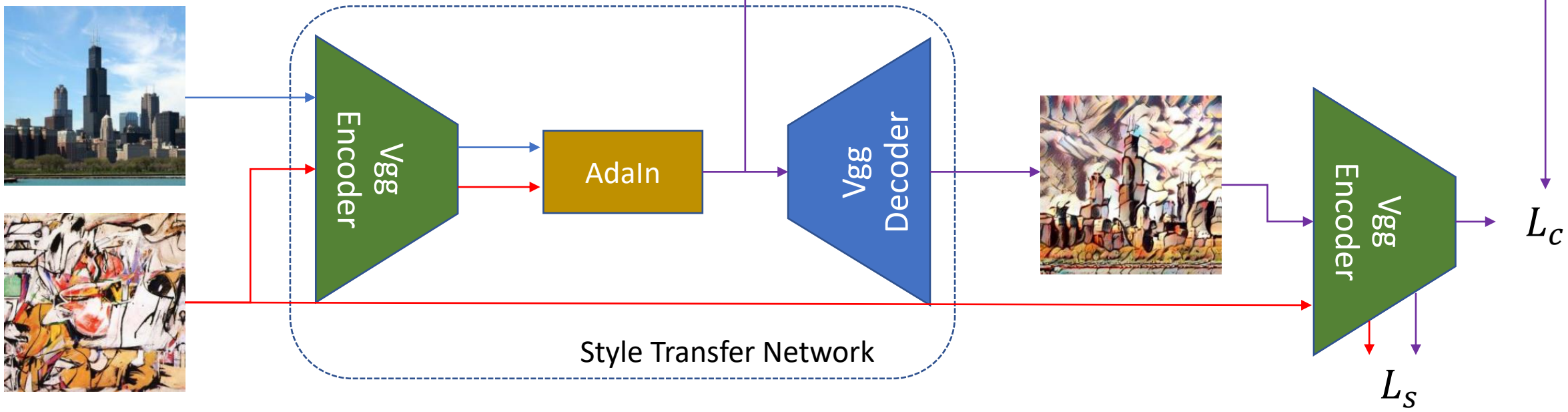
$$\sigma_{nc} = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_{nc}(x))^2 + \varepsilon}$$

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

- Data un' immagine x che rappresenta il contenuto e un' immagine y che rappresenta lo stile, AdaIn è un livello della rete che allinea la media e la varianza di x a quella di y.

$$AdaIN(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

contenuto stile



Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Encoder vgg19 up to block4_conv1

Pre-addestrato da imagenet

Layer (type)	Output Shape	Param #
input_7 (InputLayer)	[(None, None, None, 3)]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_conv4 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160

```python
class Encoder(tf.keras.models.Model):
    def __init__(self, content_layer):
        super(Encoder, self).__init__()
        vgg = VGG19(include_top=False, weights="imagenet")

        self.vgg = tf.keras.Model(
            [vgg.input], [vgg.get_layer("block4_conv1").output]
        )
        self.vgg.trainable = False

    def call(self, inputs):
        preprocessed_input = vgg19.preprocess_input(inputs)
        return self.vgg(preprocessed_input)
```

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

```python
def adaptive_instance_normalization(content_feat, style_feat, epsilon=1e-5):
    content_mean, content_variance = tf.nn.moments(
        content_feat, axes=[1, 2], keepdims=True
    )
    style_mean, style_variance = tf.nn.moments(
        style_feat, axes=[1, 2], keepdims=True
    )
    style_std = tf.math.sqrt(style_variance + epsilon)

    norm_content_feat = tf.nn.batch_normalization(
        content_feat,
        mean=content_mean,
        variance=content_variance,
        offset=style_mean,
        scale=style_std,
        variance_epsilon=epsilon,
    )
    return norm_content_feat
```

```python
def decoder():
    return tf.keras.Sequential(
        [
            ReflectionPadding2D(),
            Conv2D(256, (3, 3), activation="relu"),
            UpSampling2D(size=2),
            ReflectionPadding2D(),
            Conv2D(256, (3, 3), activation="relu"),
            ReflectionPadding2D(),
            Conv2D(256, (3, 3), activation="relu"),
            ReflectionPadding2D(),
            Conv2D(256, (3, 3), activation="relu"),
            ReflectionPadding2D(),
            Conv2D(128, (3, 3), activation="relu"),
            UpSampling2D(size=2),
            ReflectionPadding2D(),
            Conv2D(128, (3, 3), activation="relu"),
            ReflectionPadding2D(),
            Conv2D(64, (3, 3), activation="relu"),
            UpSampling2D(size=2),
            ReflectionPadding2D(),
            Conv2D(64, (3, 3), activation="relu"),
            ReflectionPadding2D(),
            Conv2D(3, (3, 3)),
        ]
    )
```

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization –Losses-

- Solamente il decoder deve venire addestrato
- Sia **c** l'immagine che rappresenta il contenuto, **s** l'immagine che rappresenta lo stile, **f** l'encoder (up to block4_conv1) , **t** il risultato dell'applicazione del livello AdaIN ovvero **t = Adain(f(c), f(s))** e **T** l'immagine stilizzata **T(c,s) = g(t),** dove g è il decoder.
- La funzione di perdita rispetto al contenuto è la seguente (dove $\|.\|_2$ è la distanza euclidea): $L_c = \left\|\left(f\left(g(t)\right) - t\right)\right\|_2$
- La funzione di perdita rispetto allo stile : $L_s = \sum_{i=1}^{L} \left\|\mu\left(\phi_i\left(g(t)\right)\right) - \mu\left(\phi_i(s)\right)\right\|_2 + \left\|\sigma\left(\phi_i\left(g(t)\right)\right) - \sigma\left(\phi_i(s)\right)\right\|_2$
- ϕ_i denota il livello nella rete VGG19 impiegato per calcolare la perdita dello stile e varia tra i seguenti:
  - "block1_conv1", # relu1-1
  - "block2_conv1", # relu2-1
  - "block3_conv1", # relu3-1
  - "block4_conv1", # relu4-1
- La funzione di perdita totale è la somma di L_c e L_s ovvero $L = L_c + \lambda L_s$

AdaIn – Controlli real time

- La rete descritta precedentemente permette di controllare a tempo di esecuzione il livello di stile che si vuole imporre tramite un parametro α. Sia $T(c, s, \alpha)$ l'immagine risultato dove c è l'immagine contenuto s è l'immagine che rappresenta lo stile e $0 \leq \alpha \leq 1$ il parametro che controlla il livello dello stile.

$$T(c, s, \alpha) = g\left((1-\alpha)f(c) + \alpha AdaIN\left(f(c), f(s)\right)\right)$$

encoder encoder encoder

decoder

- Per interpolare un insieme di k stili ciascuno con peso w_i dove $1 \leq i \leq k$ tale che $\sum_{i=1}^{k} w_i = 1$ è sufficiente calcolare $T\left(c, s_{1,2,\dots,k}, w_{1,2,\dots,k}\right) = g\left(\sum_{i=1}^{k} w_k AdaIN\left(f(c), f(s_k)\right)\right)$

encoder encoder

decoder

Risultati ottenuti

- Ho addestrato il modello tramite colab per 5h e 18min su 20000 style images e 20000 content images.
- Si è reso necessario dividere le 20000 immagini sia dello stile che del contenuto in 4 cartelle da 5000 per renderne possibile la lettura in colab.



- Il risultato ovviamente non è ottimale in quanto l'articolo originario prevedeva circa 80000 style images e 80000 content images.

Fonti

- https://github.com/AndreaBaroni92/Style-Transfer (mia sperimentazione sia dell' implementazione di Gatys che di AdaIn sulla base delle implementazioni citate sotto)
- https://ieeexplore.ieee.org/document/7780634 (articolo Gatys et Al)
- https://ieeexplore.ieee.org/document/8237429 (articolo Adain)
- https://keras.io/examples/generative/neural_style_transfer/ (implementazione keras di Gatys)
- https://github.com/emla2805/arbitrary-style-transfer (implementazione che ho studiato di AdaIn)