AI képek átalakítása tensorflow-val

Ez az oktatóprogram mélyreható tanuláson alapszik.

Egy képet egy másik kép stílusában komponál át.

Játékosan elérhető hogy hasonló kép készüljön mint Picasso vagy Van Gogh alkotása.

Ezt neurális stílusátvitelnek nevezük, és a technikát a művészi stílus neurális á

Megjegyzés:

- Ez az oktatóanyag bemutatja az eredeti stílusátviteli algoritmust.
- Optimalizálja a kép tartalmát egy adott stílushoz.
- A modern és antik megközelítések egy-egy modellt képezhet a stilizált kép közvetlen generálá

  Ez a megközelítés sokkal gyorsabb (akár 1000x). Egy előre kiképzett tetszőleges képstilizáló TensorFlow Lite-hoz.

A neurális stílusátvitel egy olyan optimalizálási technika, amelyet két kép - egy tartalomkép és egy festőművész alkotása) készítéséhez használnak, és összekeverik őket, így a kimeneti kép úgy néz referenciaképének stílusában.

Ezt úgy valósítják meg, hogy a kimeneti képet úgy optimalizálják, hogy megfeleljen a tartalom kép kép stílus statisztikájának. Ezeket a statisztikákat a képekből egy konvolúciós hálózat segítségéve

Minta forrása:

https://www.tensorflow.org/tutorials/generative/style_transfer

Kiinduló kép egy mokus a farönkön

Stilus kép:

[stilus](),

Eredmény kép:

Vizualizációs bemenetek

Itt határozzuk meg a kép betöltésének funkcióját, és korlátozzuk annak maximális méretét 512 pix

Több képkezelő eljárás kerül meghatározásra

```
1 from __future__ import absolute_import, division, print_function, unicode_literals
```

```
1 ## tensorflow és más modulok betöltése
```

```
try:
  # %tensorflow_version ellenőrzése
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf
import IPython.display as display

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12,12)
mpl.rcParams['axes.grid'] = False

import numpy as np
import PIL.Image
import time
import functools

## Veziok kiírása
print('TF verzio    : ', tf.__version__)
print('KERAS verzio : ', tf.keras.__version__)
```

```
TensorFlow 2.x selected.
TF verzio    :  2.1.0
KERAS verzio :  2.2.4-tf
```

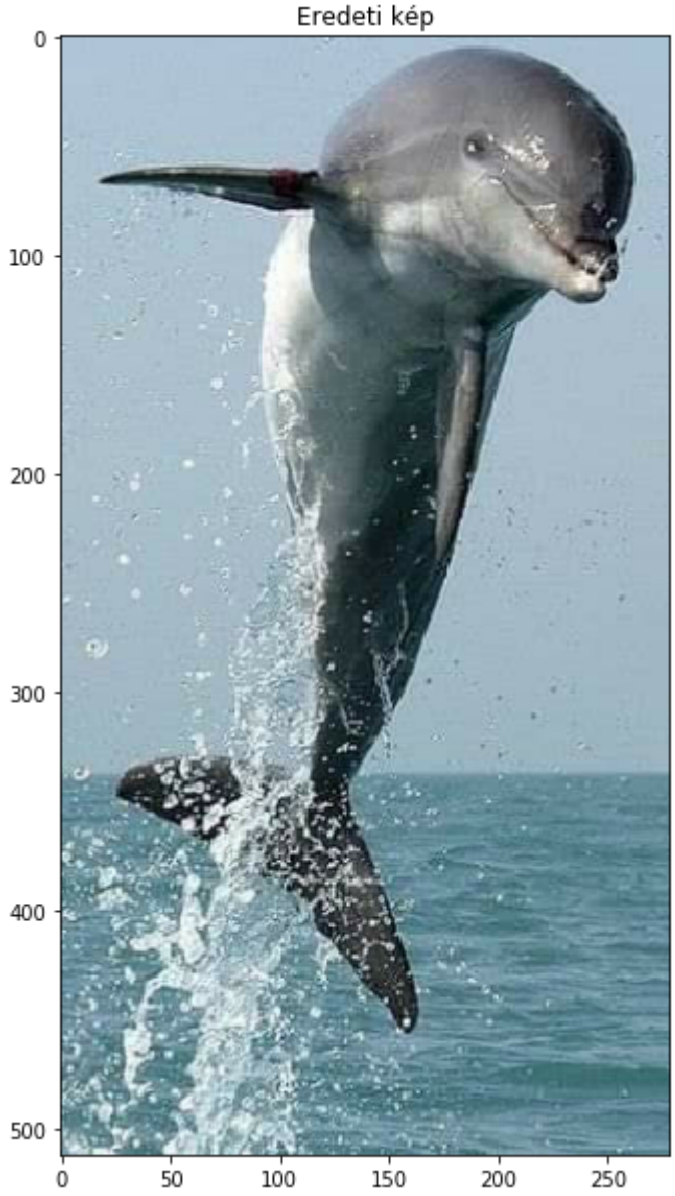
```
## képállományok elérése a WEB -en
## mokus
# content_path = tf.keras.utils.get_file('forras_kep.jpg', r'https://1.bp.blogspot.c
## sas
# content_path = tf.keras.utils.get_file('forras2_kep.jpg', r'https://1.bp.blogspot.
# delfin
content_path = tf.keras.utils.get_file('forras3_kep.jpg', r'https://1.bp.blogspot.cc
## stilus
style_path   = tf.keras.utils.get_file('cel_kep.jpg', r'https://1.bp.blogspot.com/-F

```

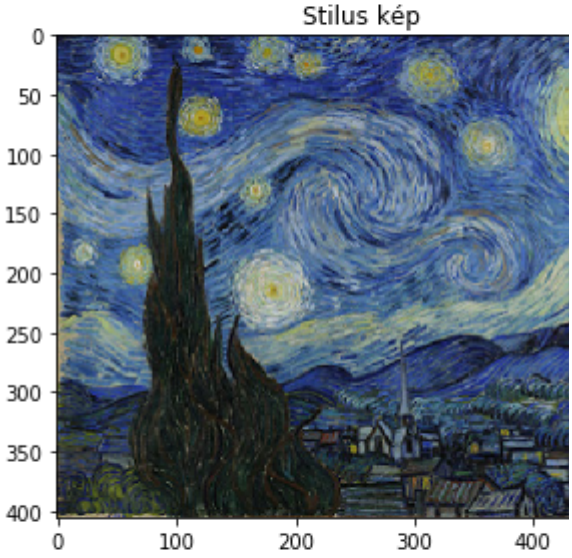
```
Downloading data from https://1.bp.blogspot.com/-fhBWj_lpjVg/XmIwn9JjYRI/AAAAAAAAa
32768/25866 [====================================] - 0s 0us/step
```

```
## kép brtöltő eljárás definiciója
def tensor_to_image(tensor):
  tensor = tensor*255
  tensor = np.array(tensor, dtype=np.uint8)
  if np.ndim(tensor)>3:
    assert tensor.shape[0] == 1
    tensor = tensor[0]
  return PIL.Image.fromarray(tensor)

def load_img(path_to_img):
  max_dim = 512  ## max
  img = tf.io.read_file(path_to_img)
  img = tf.image.decode_image(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)
```

```
  shape = tf.cast(tf.shape(img)[:-1], tf.float32)
  long_dim = max(shape)
  scale = max_dim / long_dim

  new_shape = tf.cast(shape * scale, tf.int32)

  img = tf.image.resize(img, new_shape)
  img = img[tf.newaxis, :]
  return img

## kép megjelenítés
def imshow(image, title=None):
  if len(image.shape) > 3:
    image = tf.squeeze(image, axis=0)

  plt.imshow(image)
  if title:
    plt.title(title)
```

```
## eljárások meghívása
content_image = load_img(content_path) ## képfile bolvasása
style_image = load_img(style_path)     ## stilus képfile bolvasása

plt.subplot(1, 2, 1)                   ## megjelenítési pozició
imshow(content_image, 'Eredeti kép') ## megjelenítés

plt.subplot(1, 2, 2)                   ## megjelenítési pozició
imshow(style_image, 'Stilus kép')     ## megjelenítés
```





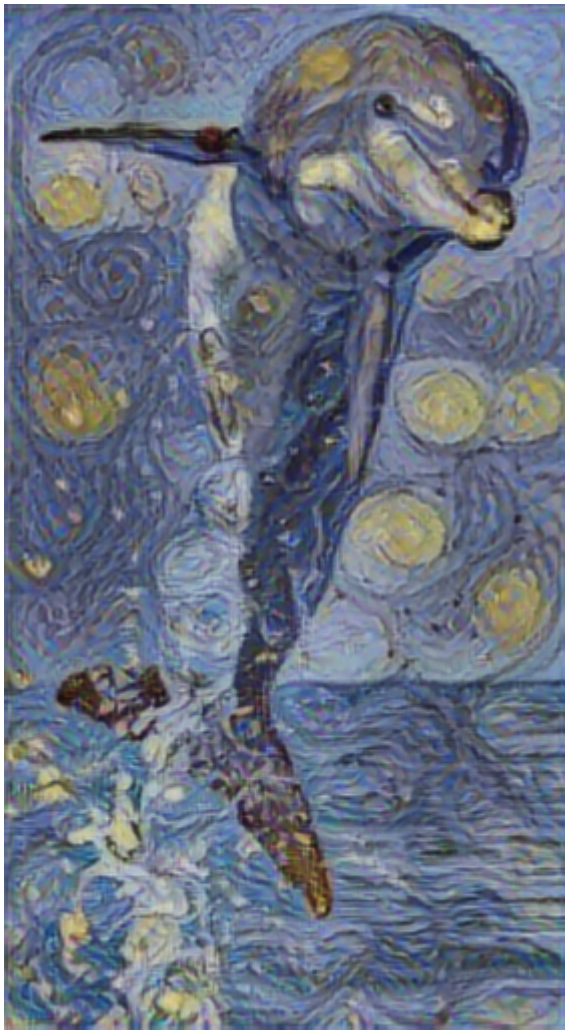
Gyors stílusátvitel a TF-Hub segítségével lehet megtámogatni

Elérése:

[TensorFlow Hub]

```
## A hub használathoz installálni kell a modult (ezt csak egyszer kell megtenni)
# !pip install "tensorflow_hub>=0.6.0"
```

```
import tensorflow_hub as hub
#hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylizatior
hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-
stylized_image = hub_module(tf.constant(content_image), tf.constant(style_image))[0]
tensor_to_image(stylized_image)
```



A tartalom és a stílus ábrázolás paramétereinek megadása

Használja a modell köztes rétegeit a kép tartalmának és stílusának ábrázolásához.

A hálózat bemeneti rétegétől kezdve az első néhány réteg aktiválás alacsony szintű funkciókat kép

A hálózaton való áthaladás során az utolsó néhány réteg magasabb szintű funkciókat képvisel - o vagy a szemét.

Ebben az esetben a VGG19 hálózati architektúrát használjujk egy előre képzett képosztályozó háló

Ezekre a köztes rétegekre van szükség a tartalom és a stílus képek ábrázolásának meghatározásá

A bemeneti képhez próbáljuk meg egyeztetni a megfelelő stílus- és tartalomcél reprezentációkat ez

Model elérése: VGG19

```
## model betoltés és paraméterezés
x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
x = tf.image.resize(x, (224, 224))
vgg = tf.keras.applications.VGG19(include_top=True, weights='imagenet')
prediction_probabilities = vgg(x)
prediction_probabilities.shape
```

```
## Prediktiv modell
predicted_top_5 = tf.keras.applications.vgg19.decode_predictions(prediction_probabil
[(class_name, prob) for (number, class_name, prob) in predicted_top_5]
```

```
[('great_white_shark', 0.58453816),
 ('albatross', 0.21685721),
 ('grey_whale', 0.13315526),
 ('tiger_shark', 0.025578123),
 ('gar', 0.014428519)]
```

```
##  VGG19-et betöltése az osztályozó fej nélkül, és  rétegnevek felsorolása nélkül
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

print()
for layer in vgg.layers:
  print(layer.name)
```

```
input_10
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_conv4
block4_pool
block5_conv1
block5_conv2
block5_conv3
block5_conv4
block5_pool
```

Choose intermediate layers from the network to represent the style and content of the image:

```
## Válasszunk köztes rétegek a hálózatból, hogy hogyan ábrázolja a kép stílusát és t
## Tartalmi réteg, ahova behúzza a funkciótérképeket
content_layers = ['block5_conv2']

# Stilus réteg megadása
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

num_content_layers = len(content_layers)
```

```
num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

Középső rétegek a stílus és a tartalom számára

Ezek a közbenső kimenetek engedik meg az előre képzett képosztályozó hálózatunkban a stílus és

Magas szinten annak érdekében, hogy egy hálózat elvégezhesse a kép osztályozását (amelyet erre képet.

Ehhez a nyers képet bemeneti képpontként kell venni, és létre kell hozni egy olyan belső ábrázolást tulajdonságok komplex megértésévé alakítja.

Ez is az oka annak, hogy a konvolúciós ideghálózatok képesek jól általánosítani: képesek megraga meghatározó jellemzőket (pl. Macskák vagy kutyák), amelyek a háttérzajt és más kellemetlenségel

Így, valahol a nyers kép beillesztése a modellbe és a kimeneti osztályozási címke között, a modell k modell köztes rétegeinek elérésével leírhatja a bemeneti képek tartalmát és stílusát.

Ezek után készítsuk el a modellt

A tf.keras.applications hálózatait úgy tervezték meg, hogy a Keras funkcionális API segítségével kö értékeit.

A modell meghatározásához a funkcionális API használatával adja meg a bemeneteket és a kimer

Ez a következő függvény egy VGG19 modellt épít fel, amely visszaadja a közbenső réteg kimenetei

```
def vgg_layers(layer_names):
  # # Töltsünk be modellünket. Töltsük be az előképzett VGG-t, a képzett az imagenet
  vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

  outputs = [vgg.get_layer(name).output for name in layer_names]

  model = tf.keras.Model([vgg.input], outputs)
  return model
```

```
style_extractor = vgg_layers(style_layers)
style_outputs = style_extractor(style_image*255)

##  Nézzük meg az egyes rétegek kimeneti statisztikáit
for name, output in zip(style_layers, style_outputs):
  print(name)
  print("  shape: ", output.numpy().shape)
  print("  min: ", output.numpy().min())
  print("  max: ", output.numpy().max())
  print("  mean: ", output.numpy().mean())
  print()
```

```
block1_conv1
  shape:  (1, 406, 512, 64)
  min:  0.0
  max:  643.19025
  mean:  21.583544

block2_conv1
  shape:  (1, 203, 256, 128)
  min:  0.0
  max:  2553.161
  mean:  130.00684

block3_conv1
  shape:  (1, 101, 128, 256)
  min:  0.0
  max:  7192.019
  mean:  126.93232

block4_conv1
  shape:  (1, 50, 64, 512)
  min:  0.0
  max:  15073.25
  mean:  469.53122

block5_conv1
  shape:  (1, 25, 32, 512)
  min:  0.0
  max:  3081.0842
  mean:  38.43216
```

Stilus számítások

A kép tartalmát a közbenső jellemző térképek értékei képviselik.

Kiderült, hogy a kép stílusát leírhatjuk az eszközökkel és a különféle térképek közötti összefüggése

Számítson ki egy Gram-mátrixot, amely tartalmazza ezt az információt, úgy, hogy az objektumvekt helyekre, és átlagolja ezt a külső terméket az összes helyre.

Ez a Gram-mátrix egy adott rétegre kiszámítható:

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

Ez tömören megvalósítható a `tf.linalg.einsum` függvény használatával:

```
def gram_matrix(input_tensor):
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Kivonatolt stílus és tartalom

Készítsünk egy modellt, amely visszatér a stílus és a tartalom információkkal tenzoraihoz.

```
class StyleContentModel(tf.keras.models.Model):
  def __init__(self, style_layers, content_layers):
    super(StyleContentModel, self).__init__()
    self.vgg =  vgg_layers(style_layers + content_layers)
    self.style_layers = style_layers
    self.content_layers = content_layers
    self.num_style_layers = len(style_layers)
    self.vgg.trainable = False

  def call(self, inputs):
    "Expects float input in [0,1]"
    inputs = inputs*255.0
    preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
    outputs = self.vgg(preprocessed_input)
    style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                      outputs[self.num_style_layers:])

    style_outputs = [gram_matrix(style_output)
                     for style_output in style_outputs]

    content_dict = {content_name:value
                    for content_name, value
                    in zip(self.content_layers, content_outputs)}

    style_dict = {style_name:value
                  for style_name, value
                  in zip(self.style_layers, style_outputs)}

    return {'content':content_dict, 'style':style_dict}
```

Amikor képre hívnak egy képet, ez a modell visszatér a "stílus_rétegek" grammátrixa (stílusa) és a "

```
extractor = StyleContentModel(style_layers, content_layers)

results = extractor(tf.constant(content_image))

style_results = results['style']

print('Styles:')
for name, output in sorted(results['style'].items()):
  print("  ", name)
  print("    shape: ", output.numpy().shape)
  print("    min: ", output.numpy().min())
  print("    max: ", output.numpy().max())
  print("    mean: ", output.numpy().mean())
  print()
```

```
15
16 print("Contents:")
17 for name, output in sorted(results['content'].items()):
18   print("  ", name)
19   print("    shape: ", output.numpy().shape)
20   print("    min: ", output.numpy().min())
21   print("    max: ", output.numpy().max())
22   print("    mean: ", output.numpy().mean())
23
```

```
Styles:
   block1_conv1
    shape:  (1, 64, 64)
    min:  0.0
    max:  33396.195
    mean:  512.9968

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  121439.21
    mean:  14256.908

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  245691.77
    mean:  10863.229

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  2920080.0
    mean:  169023.64

   block5_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  56942.824
    mean:  1094.9308

Contents:
   block5_conv2
    shape:  (1, 32, 17, 512)
    min:  0.0
    max:  1521.962
    mean:  12.249769
```

```
1 style_targets = extractor(style_image)['style']
2 content_targets = extractor(content_image)['content']
3
4 image = tf.Variable(content_image) ##  tartalmazzák a képet az optimalizálás érdekéb
5
6 def clip_0_1(image):  ## Mivel ez úszó kép, definiáljon egy funkciót a pixelértékek
7   return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)
8
9 opt = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)  ## optimali
10
```

```
style_weight=1e-2
content_weight=1e4
```

```
def style_content_loss(outputs):
    style_outputs = outputs['style']
    content_outputs = outputs['content']
    style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])*
                           for name in style_outputs.keys()])
    style_loss *= style_weight / num_style_layers

    content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[r
                             for name in content_outputs.keys()])
    content_loss *= content_weight / num_content_layers
    loss = style_loss + content_loss
    return loss
```

```
@tf.function()
def train_step(image):  ## kép változtatási lépések
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)

  grad = tape.gradient(loss, image)
  opt.apply_gradients([(grad, image)])
  image.assign(clip_0_1(image))
```

futtassünk pár válzoztató lépés közvetlenül és nézzük meg a hatását

```
train_step(image)      ## 1 lépés
train_step(image)      ## 2 lépés
train_step(image)      ## 3 lépés
train_step(image)      ## 4 lépés
train_step(image)      ## 5 lépés
tensor_to_image(image) # megjelenítés
```

Mivel eddig működik, végezzünk optimalizálást is

```
import time
start = time.time()

epochs = 5            ## léptető paraméter
steps_per_epoch = 80  ## léptető paraméter

step = 0
for n in range(epochs):
  for m in range(steps_per_epoch):
    step += 1
    train_step(image)
    print(".", end='')
  display.clear_output(wait=True)
  display.display(tensor_to_image(image))
  print("Train step: {}".format(step))

end = time.time()
print("Total time: {:.1f}".format(end-start))
```

```
Train step: 400
Total time: 12.8
```

Teljes variációs veszteség

Ennek az alapvető megvalósításnak az egyik hátránya, hogy sok magas frekvenciájú művet hoz lé

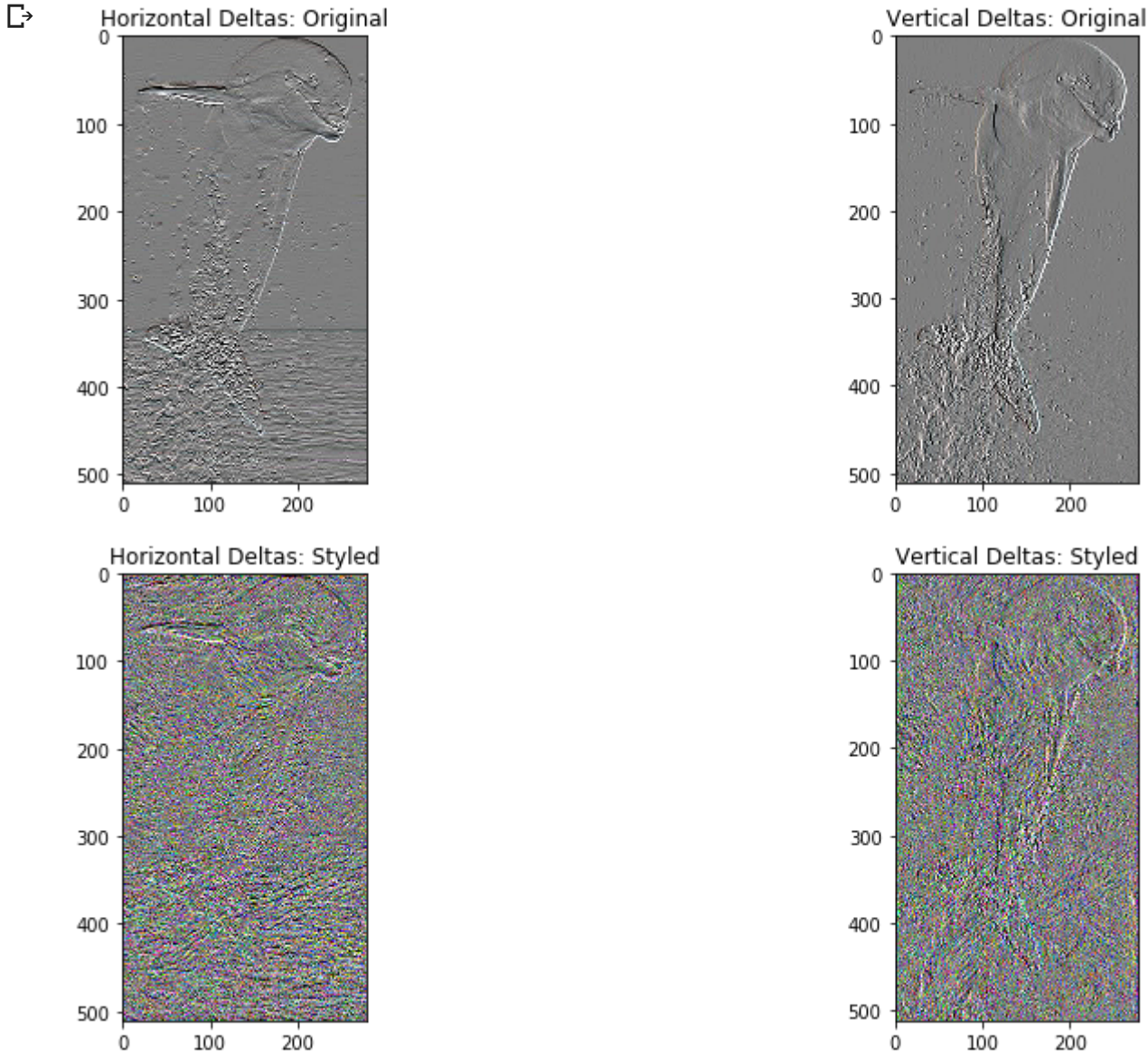
Csökkentse ezeket a kép nagyfrekvenciájú összetevőinek kifejezett szabályzási kifejezés használa

A stílusátvitel során ezt gyakran így hívják: *total variation loss*

```
def high_pass_x_y(image):
  x_var = image[:,:,1:,:] - image[:,:,:-1,:]
  y_var = image[:,1:,:,:] - image[:,:-1,:,:]

  return x_var, y_var
```

```
x_deltas, y_deltas = high_pass_x_y(content_image)

plt.figure(figsize=(14,10))
plt.subplot(2,2,1)
imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Original")

plt.subplot(2,2,2)
```

```
imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Original")

x_deltas, y_deltas = high_pass_x_y(image)

plt.subplot(2,2,3)
imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Styled")

plt.subplot(2,2,4)
imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Styled")
```



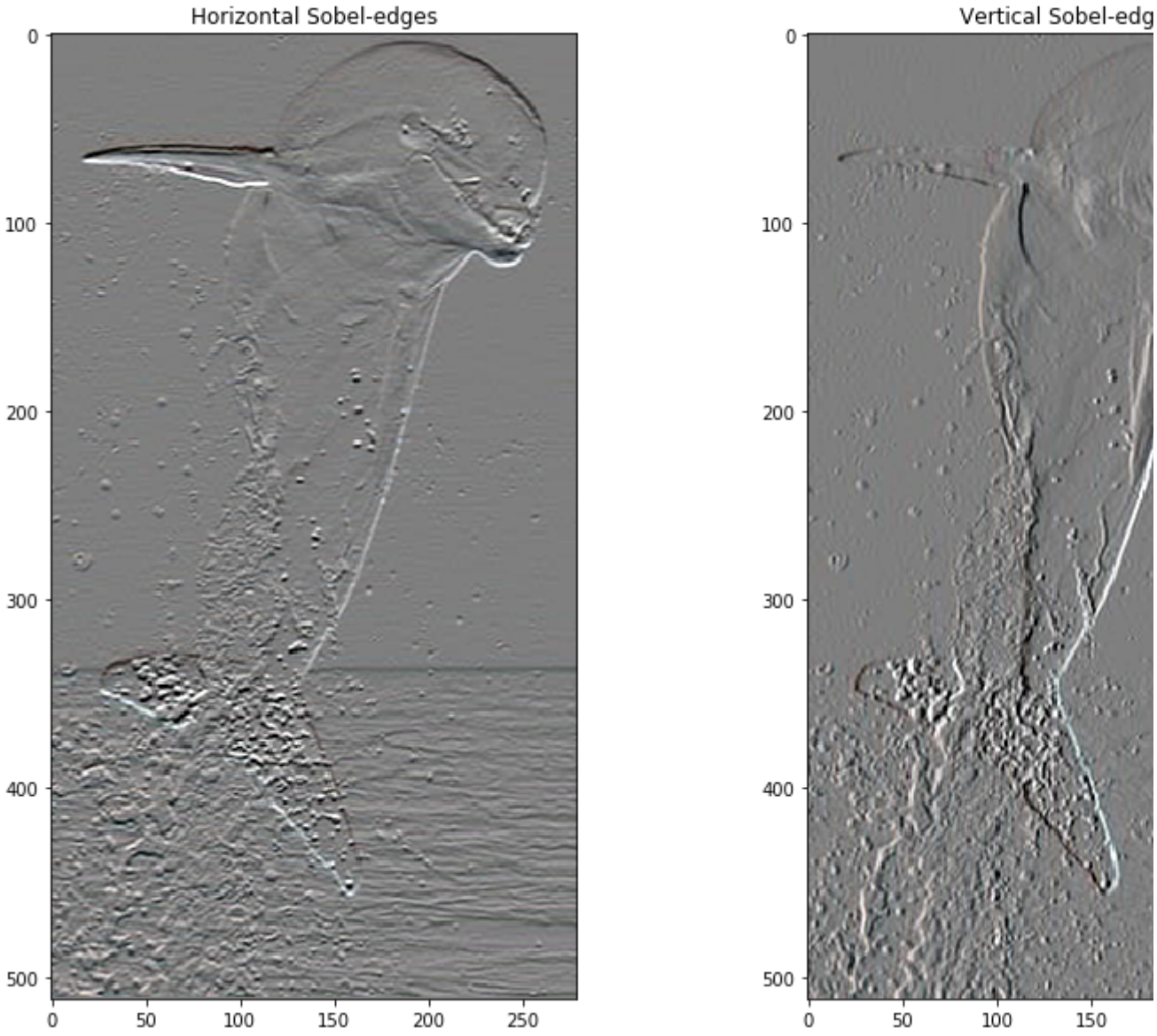
Most megmutatjuk, hogyan növelhető a magas frekvenciájú kompo

Ez a magas frekvenciájú elem alapvetően egy éledetektor is.

Hasonló kimenetet kaphatunk például a Sobel éldetektorról, például:

```
plt.figure(figsize=(14,10))

sobel = tf.image.sobel_edges(content_image)
plt.subplot(1,2,1)
imshow(clip_0_1(sobel[...,0]/4+0.5), "Horizontal Sobel-edges")
```

```
6 plt.subplot(1,2,2)
7 imshow(clip_0_1(sobel[...,1]/4+0.5), "Vertical Sobel-edges")
```



The regularization loss associated with this is the sum of the squares of the values:

```
1 def total_variation_loss(image):
2   x_deltas, y_deltas = high_pass_x_y(image)
3   return tf.reduce_sum(tf.abs(x_deltas)) + tf.reduce_sum(tf.abs(y_deltas))
4
5 total_variation_loss(image).numpy()
6
7 tf.image.total_variation(image).numpy()
```

```
array([72955.53], dtype=float32)
```

Optimalizáció ujra futtatása

Válasszon súlyt `total_variation_loss`:

```
1 total_variation_weight=30
```

```
 2
 3 @tf.function()
 4 def train_step(image):
 5   with tf.GradientTape() as tape:
 6     outputs = extractor(image)
 7     loss = style_content_loss(outputs)
 8     loss += total_variation_weight*tf.image.total_variation(image)
 9
10   grad = tape.gradient(loss, image)
11   opt.apply_gradients([(grad, image)])
12   image.assign(clip_0_1(image))
13
14 image = tf.Variable(content_image)
```

```
 1 import time
 2 start = time.time()
 3
 4 epochs = 10
 5 steps_per_epoch = 100
 6
 7 step = 0
 8 for n in range(epochs):
 9   for m in range(steps_per_epoch):
10     step += 1
11     train_step(image)
12     print(".", end='')
13   display.clear_output(wait=True)
14   display.display(tensor_to_image(image))
15   print("Train step: {}".format(step))
16
17 end = time.time()
18 print("Total time: {:.1f}".format(end-start))
```

```
Train step: 1000
Total time: 33.1
```

végül mentsuk az eredményünket

```
file_name = 'stylized-mokus.png'
tensor_to_image(image).save(file_name)

try:
  from google.colab import files
except ImportError:
   pass
else:
  files.download(file_name)
```