

✓ Importando as bibliotecas

Antes de começar, precisamos carregar algumas bibliotecas essenciais para o projeto:

- **Matplotlib (pyplot, gridspec):** usada para visualizar imagens e organizar os gráficos em diferentes layouts.
- **NumPy:** permite manipular matrizes e arrays numéricos, que são a base para trabalhar com imagens em visão computacional.
- **TensorFlow:** o framework de machine learning que nos dá as ferramentas para rodar redes neurais e processar dados.
- **TensorFlow Hub:** um repositório de modelos pré-treinados. Vamos utilizá-lo para carregar o modelo de *style transfer* sem precisar treinar do zero.

```
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
```

✓ Definindo os caminhos das imagens

Nesta etapa, indicamos os **arquivos que serão utilizados** no processo de *style transfer*.

- `content_image_path` → representa a imagem de **conteúdo** (a base que receberá o estilo).
- `style_image_path` → representa a imagem de **estilo** (a obra de arte cuja estética será aplicada).

No exemplo, escolhemos uma foto chamada **baina2.jpeg** como conteúdo e a pintura **O Grito** como estilo.

Esses arquivos precisam estar previamente carregados no ambiente do Colab, dentro da pasta `/content/`.

```
content_image_path = '/content/baina2.jpeg'
style_image_path = '/content/grito.jpg'
```



Função para carregar imagens

Aqui criamos a função `load_image`, que será responsável por **ler e preparar as imagens** para o modelo:

1. `tf.io.read_file(path)` → abre o arquivo da imagem a partir do caminho informado.
2. `tf.io.decode_image(..., channels=3, dtype=tf.float32)` → decodifica a imagem em formato RGB, convertendo os valores dos pixels para **float32** (necessário para TensorFlow).
3. `[tf.newaxis, ...]` → adiciona uma nova dimensão (batch size = 1), já que os modelos de deep learning esperam processar **lotes de imagens**.
4. `tf.image.resize(image, size, preserve_aspect_ratio=True)` → redimensiona a imagem para o tamanho desejado (por padrão 256x256), mantendo a proporção original.



Essa função garante que qualquer imagem lida esteja no **formato correto** para ser usada pelo modelo de *style transfer*.

```
def load_image(path, size = (256, 256)):  
    image = tf.io.decode_image(tf.io.read_file(path), channels = 3, dtype = tf.float32)[tf.newaxis, ...]  
    image = tf.image.resize(image, size, preserve_aspect_ratio = True)  
    return image
```



Carregando as imagens de conteúdo e estilo

Aqui usamos a função `load_image` (criada no bloco anterior) para abrir e preparar as imagens:

- `content_image` → carrega a imagem de **conteúdo** a partir do caminho definido em `content_image_path`, redimensionando para **384x384 pixels**. Esse tamanho maior ajuda a preservar mais detalhes visuais da imagem base.
- `style_image` → carrega a imagem de **estilo** a partir de `style_image_path`. Como não passamos o parâmetro de tamanho, ela será redimensionada pelo padrão da função (`256x256`), suficiente para capturar os padrões artísticos sem perder desempenho.



Dessa forma, as duas imagens já ficam prontas para serem processadas pelo modelo de *style transfer*.

```
content_image = load_image(content_image_path,(384, 384))
style_image = load_image(style_image_path)
```

✓ 🔍 Verificando as dimensões das imagens

Este comando exibe as **formas (shapes)** dos tensores que representam as imagens:

- `content_image.shape` → mostra as dimensões da imagem de conteúdo.
- `style_image.shape` → mostra as dimensões da imagem de estilo.

O formato retornado é algo como `(1, altura, largura, 3)`, onde:

- `1` → indica que temos apenas **uma imagem no lote** (batch size).
- `altura, largura` → representam o tamanho da imagem após o redimensionamento.
- `3` → corresponde aos **três canais de cor (RGB)**.

🔑 Essa verificação garante que ambas as imagens foram carregadas corretamente e estão no formato esperado para o modelo.

```
content_image.shape, style_image.shape
```

```
(TensorShape([1, 384, 288, 3]), TensorShape([1, 242, 256, 3]))
```

✓ 🖼️ Função para exibir imagens lado a lado

Aqui criamos a função `show_images`, que serve para visualizar as imagens de forma organizada:

1. `n = len(images)` → conta quantas imagens serão exibidas.
2. `fig = plt.figure(...)` → define o tamanho da figura principal.
3. `gs = gridspec.GridSpec(1, n, ...)` → cria uma grade (grid) com 1 linha e **n colunas**, ou seja, coloca todas as imagens lado a lado.
4. Loop `for i in range(n)` → percorre todas as imagens:

- `ax.imshow(np.squeeze(images[i]))` → mostra a imagem, removendo dimensões extras.
- `ax.set_xticks([]), ax.set_yticks([])` → remove os eixos para deixar a visualização mais limpa.
- `ax.set_title(titles[i])` → se títulos forem fornecidos, exibe-os acima de cada imagem.

5. `plt.show()` → renderiza a figura na tela.

🔑 Essa função será usada várias vezes para comparar a imagem de conteúdo, a de estilo e o resultado final.

```
def show_images(images, titles = []):
    number_images = len(images)
    plt.figure(figsize = (12,12))
    gs = gridspec.GridSpec(1, number_images)
    for i in range(number_images):
        plt.subplot(gs[i])
        plt.axis('off')
        plt.imshow(images[i][0])
        plt.title(titles[i])
```

✓ 👁 Visualizando as imagens de conteúdo e estilo

Aqui chamamos a função `show_images` para exibir as duas imagens principais lado a lado:

- `content_image` → é a imagem base, que servirá de **conteúdo**.
- `style_image` → é a imagem de **estilo**, de onde o modelo vai extrair os padrões artísticos.

Os títulos `['Content image', 'Style image']` ajudam a identificar cada uma na visualização.

🔑 Esse passo é importante para confirmar que as imagens foram carregadas corretamente antes de aplicar o *style transfer*.

```
show_images([content_image, style_image], ['Content image', 'Style image'])
```

Content image



Style image



✓ Carregando o modelo pré-treinado de Style Transfer

Neste bloco definimos e carregamos o modelo de *style transfer* a partir do **TensorFlow Hub**:

- `model_path` → contém o link para o modelo pré-treinado **Arbitrary Image Stylization v1-256**, desenvolvido pelo projeto Magenta do Google.
- `hub.load(model_path)` → baixa e carrega o modelo na memória, deixando-o pronto para uso.

Esse modelo já foi treinado em diversas imagens de arte, o que permite aplicar estilos variados em qualquer imagem sem a necessidade de treinar do zero.

🔑 Vantagem: economizamos tempo e recursos, usando um modelo robusto e testado pela comunidade.

```
model_path = 'https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2'  
model = hub.load(model_path)
```

✓ 🎨 Aplicando o modelo de Style Transfer

Aqui passamos as imagens carregadas para o modelo pré-treinado:

- `tf.constant(content_image)` → converte a imagem de conteúdo em um **tensor constante** do TensorFlow.
- `tf.constant(style_image)` → faz o mesmo para a imagem de estilo.
- `model(...)` → aplica o modelo de *style transfer*, gerando como saída uma versão estilizada da imagem de conteúdo.

O resultado é armazenado em `results`, que é um **tensor contendo a nova imagem já transformada**.

🔑 Esse é o momento principal do projeto: a junção entre **conteúdo** e **estilo** em uma única imagem.

```
results = model(tf.constant(content_image), tf.constant(style_image))
```

✓ 🧐 Inspeccionando o objeto de resultados

Aqui simplesmente digitamos `results` para visualizar o que o modelo retornou.

- O retorno não é a imagem diretamente, mas sim um **tensor do TensorFlow**.
- Esse tensor contém os valores numéricos (pixels normalizados entre 0 e 1) da imagem estilizada.
- Geralmente, o `results` é exibido como algo do tipo:

```
results
```

```
[<tf.Tensor: shape=(1, 384, 288, 3), dtype=float32, numpy=
array([[[[0.88455325, 0.690301 , 0.5053787 ],
        [0.87862027, 0.6721555 , 0.49283764],
        [0.90481526, 0.7134768 , 0.5491623 ],
        ...,
        [0.5812541 , 0.32032126, 0.16099231],
        [0.730587 , 0.48698083, 0.25199202],
        [0.75477576, 0.49209446, 0.26908016]],

       [[0.87687737, 0.66416377, 0.48129493],
        [0.86801785, 0.6467988 , 0.46174735],
        [0.9020329 , 0.7020096 , 0.52940065],
        ...,
        [0.58642364, 0.31620863, 0.15983275],
        [0.7235429 , 0.48277578, 0.24555 ],
        [0.749077 , 0.49474633, 0.26055956]],

       [[0.849659 , 0.6163755 , 0.42541593],
        [0.84391636, 0.600268 , 0.40830228],
        [0.89221114, 0.68010783, 0.49052972],
        ...,
        [0.52604455, 0.2572981 , 0.12173425],
        [0.62547237, 0.37159193, 0.1658686 ],
        [0.6699468 , 0.39346033, 0.18269347]],

       ...,

       [[0.77760214, 0.6056837 , 0.4101181 ],
        [0.8307236 , 0.7046526 , 0.49631006],
        [0.80186266, 0.6768198 , 0.42045048],
        ...,
        [0.595581 , 0.44921616, 0.411162 ],
        [0.73149025, 0.62386715, 0.5479021 ],
        [0.7128068 , 0.584705 , 0.48186162]],

       [[0.7789683 , 0.58773106, 0.40908805],
        [0.8305505 , 0.686993 , 0.4886381 ],
        [0.8094063 , 0.6705128 , 0.42074057],
        ...,
        [0.5885347 , 0.45350558, 0.4060379 ],
        [0.7279805 , 0.62011325, 0.5419871 ],
        [0.7131064 , 0.5803783 , 0.47452122]]],
```

```
[[0.79533666, 0.6030836 , 0.42951494],  
 [0.84304774, 0.6991737 , 0.49969643],  
 [0.81675905, 0.68528044, 0.43424067],  
 ...,  
 [0.5961308 , 0.46610036, 0.39684847],  
 [0.7329342 , 0.622434 , 0.53084874],  
 [0.7161103 , 0.5782915 , 0.46839574]]]], dtype=float32)>]
```

✓ 👁 Comparando conteúdo, estilo e resultado

Aqui usamos novamente a função `show_images`, mas agora exibindo **três imagens lado a lado**:

1. `content_image` → a imagem base de conteúdo.
2. `style_image` → a imagem de estilo escolhida.
3. `results[0]` → o **resultado do style transfer**, ou seja, a imagem de conteúdo estilizada.

Os títulos `['Content image', 'Style image', 'Result']` ajudam a identificar cada uma.

🔑 Esse passo é essencial para visualizar de forma clara como o modelo combinou **estrutura (conteúdo)** e **arte (estilo)**.

```
show_images([content_image, style_image, results[0]], ['Content image', 'Style image', 'Result'])
```


Content image



Style image



Result



✓ 🛠️ Preparando a imagem resultante

Aqui fazemos um ajuste no tensor gerado pelo modelo:

- `results[0]` → seleciona a primeira (e única) imagem do resultado.
- `tf.squeeze(...)` → remove dimensões extras desnecessárias do tensor, deixando apenas o formato padrão da imagem `(altura, largura, 3)`.

Sem o `squeeze`, a imagem ficaria com formato `(1, altura, largura, 3)`, ou seja, com um **batch size** de 1.

🔑 Esse passo é importante para que possamos salvar ou manipular a imagem como um arquivo comum.

```
result_image = tf.squeeze(results[0])
result_image = tf.clip_by_value(result_image, 0.0, 1.0)
```

✓ Salvando a imagem resultante

Aqui gravamos a imagem estilizada como um arquivo PNG:

- `result_image.numpy()` → converte o tensor do TensorFlow em um **array NumPy**, formato aceito pelo Matplotlib.
- `plt.imsave("result.png", ...)` → salva o array como uma imagem chamada **result.png** no diretório atual do Colab.

Depois de salvo, o arquivo pode ser baixado para o computador ou usado em outras aplicações.

🔑 Esse passo final transforma o resultado do modelo em um arquivo de imagem comum, que pode ser compartilhado facilmente.

```
plt.imsave("result.png", result_image.numpy())
print("Image saved as result.png")
```

Image saved as result.png

✓ Conclusão

Finalizamos o projeto de **Style Transfer** usando TensorFlow Hub! 🧠✨

- Carregamos e preparamos imagens de **conteúdo e estilo**.
- Aplicamos um modelo pré-treinado para combinar estrutura e arte.
- Visualizamos os resultados e salvamos a imagem final em arquivo.

Esse fluxo mostra como é possível usar modelos prontos para criar efeitos visuais impressionantes sem precisar treinar uma rede neural do zero.

```
print("✅ Projeto concluído com sucesso! A imagem resultante foi salva como 'result.png'.")
```