Ferramenta para Busca de Imagens usando processamento de linguagem natural

A ferramenta utiliza rede neural de codificador duplo (também conhecido como duas torres) modelo para procurar imagens usando linguagem natural. O modelo é inspirado em a abordagem CLIP, introduzida por Alec Radford et al. A ideia é treinar um codificador de visão e um texto codificador em conjunto para projetar a representação de imagens e suas legendas na mesma incorporação espaço, de modo que as incorporações de legenda estejam localizadas perto das incorporações das imagens que descrevem.

Configuração

```
!pip install transformers
!pip install git+https://github.com/openai/CLIP.git
```

```
import pandas as pd
import urllib.request
from multiprocessing.pool import ThreadPool
from PIL import Image
import math
import numpy as np
from IPython.display import Image
from IPython.core.display import HTML
import torch
import tensorflow as tf
from transformers import AutoProcessor, TFCLIPModel
import clip
```

Como dataset utiliza-se o conjunto de dados UNSPLASH para treinar o modelo de encoder duplo. UNSPLASH contém 25.000 imagens, cada uma das quais tem pelo menos 5 anotações de legendas diferentes. O conjunto de dados geralmente é usado para tarefas de legendagem de imagem, mas podemos redirecionar os pares imagem-legenda para treinar nosso codificador duplo modelo para busca de imagens.

Exatração do conjunto de dados, que consiste em duas pastas compactadas: um com imagens e o outro — com legendas de imagem associadas. A pasta de imagens compactadas tem 13 GB de tamanho.

```
!wget https://unsplash.com/data/lite/latest
```

```
!mkdir unsplash-dataset
!mkdir unsplash-dataset/lite/
!mkdir unsplash-dataset/lite/photos
!mkdir unsplash-dataset/lite/features
```

```
!unzip "/content/latest" -d "/content/unsplash-dataset/lite"
```

```
Archive:  /content/latest
  inflating: /content/unsplash-dataset/lite/colors.tsv000
  inflating: /content/unsplash-dataset/lite/photos.tsv000
  inflating: /content/unsplash-dataset/lite/README.md
  inflating: /content/unsplash-dataset/lite/collections.tsv000
  inflating: /content/unsplash-dataset/lite/TERMS.md
```

```
      inflating: /content/unsplash-dataset/lite/DOCS.md
      inflating: /content/unsplash-dataset/lite/keywords.tsv000
      inflating: /content/unsplash-dataset/lite/conversions.tsv000
```

```
from pathlib import Path
dataset_version = "lite"
unsplash_dataset_path = Path("unsplash-dataset") / dataset_version
```

```
# Lendo as imagens
photos = pd.read_csv(unsplash_dataset_path / "photos.tsv000", sep='\t', header=0)

# Extraindos as ID's e a URL das imagens
photo_urls = photos[['photo_id', 'photo_image_url']].values.tolist()

# Monstrando a quantidade de fotos prensentes
print(f'Photos in the dataset: {len(photo_urls)}')
```

```
     Photos in the dataset: 25000
```

```
# Realizando o download das imagens
photos_donwload_path = unsplash_dataset_path / "photos"

# Função para download de apenas uma foto
def download_photo(photo):

    photo_id = photo[0]


    photo_url = photo[1] + "?w=640"


    photo_path = photos_donwload_path / (photo_id + ".jpg")


    if not photo_path.exists():
        try:
            urllib.request.urlretrieve(photo_url, photo_path)
        except:

            print(f"Cannot download {photo_url}")
            pass


threads_count = 128
pool = ThreadPool(threads_count)

pool.map(download_photo, photo_urls)

display(f'Photos downloaded: {len(photos)}')
```

```
     Cannot download https://images.unsplash.com/photo-1578445700473-5d1cd2480a6a?w=640
     'Photos downloaded: 25000'
```

```
dataset_version = "lite"
photos_path = Path("unsplash-dataset") / dataset_version / "photos"

photos_files = list(photos_path.glob("*.jpg"))

print(f"Photos found: {len(photos_files)}")
```

```
Photos found: 24999
```

Importação de Bibliotecas:

As bibliotecas clip, torch, e Image são importadas para suportar o processamento de imagens e o uso do modelo CLIP. Determinação do Dispositivo de Execução:

A variável device é configurada para usar "cuda" (GPU) se uma GPU estiver disponível, caso contrário, usa "cpu" (CPU). Carregamento do Modelo CLIP:

O modelo pré-treinado CLIP "ViT-B/32" é carregado com a função clip.load. A função preprocess é definida para pré-processar imagens e o modelo é armazenado na variável model. Definição da Função compute_clip_features:

Uma função chamada compute_clip_features é definida para calcular recursos CLIP para um lote de fotos. Pré-processamento de Imagens:

As imagens são carregadas a partir de arquivos usando a classe Image do PIL e armazenadas em uma lista chamada photos. Pré-Processamento das Imagens e Movimentação para o Dispositivo:

As imagens são pré-processadas com a função preprocess e empilhadas em um tensor do PyTorch. O tensor é movido para o dispositivo (CPU ou GPU). Cálculo de Recursos CLIP:

Dentro de um bloco with torch.no_grad(), o modelo CLIP é usado para codificar as imagens pré-processadas, resultando em recursos relevantes armazenados na variável photos_features. Normalização de Recursos:

Os recursos são normalizados dividindo cada vetor pelo seu comprimento (norma L2) ao longo da última dimensão. Retorno dos Recursos:

Os recursos normalizados são retornados como um array NumPy, pronto para uso em cálculos adicionais ou em tarefas de IA.

```
import clip
import torch
from PIL import Image

device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)

def compute_clip_features(photos_batch):
    photos = [Image.open(photo_file) for photo_file in photos_batch]

    photos_preprocessed = torch.stack([preprocess(photo) for photo in photos]).to(device)

    with torch.no_grad():

        photos_features = model.encode_image(photos_preprocessed)
        photos_features /= photos_features.norm(dim=-1, keepdim=True)


    return photos_features.cpu().numpy()
```

```
100%|███████████████████████████████████████| 338M/338M [00:04<00:00, 74.0MiB/s]
```

O código processa imagens em lotes usando o modelo CLIP. Define o tamanho do lote como 32 e especifica o caminho para armazenar os recursos. Calcula o número de lotes com base no tamanho do lote e no número total de imagens. Em um loop, processa cada lote de imagens. Para cada lote, calcula os recursos CLIP das imagens. Salva os recursos em um arquivo NPY e os IDs das imagens em um arquivo CSV. Lida com exceções e relata problemas durante o processamento de lotes.

```
import math
import numpy as np
import pandas as pd

batch_size = 32

features_path = Path("unsplash-dataset") / dataset_version / "features"

batches = math.ceil(len(photos_files) / batch_size)

for i in range(batches):
    print(f"Processing batch {i+1}/{batches}")

    batch_ids_path = features_path / f"{i:010d}.csv"
    batch_features_path = features_path / f"{i:010d}.npy"

    if not batch_features_path.exists():
        try:

            batch_files = photos_files[i*batch_size : (i+1)*batch_size]

            batch_features = compute_clip_features(batch_files)
            np.save(batch_features_path, batch_features)

            photo_ids = [photo_file.name.split(".")[0] for photo_file in batch_files]
            photo_ids_data = pd.DataFrame(photo_ids, columns=['photo_id'])
            photo_ids_data.to_csv(batch_ids_path, index=False)
        except:

            print(f'Problem with batch {i}')
```

```
Processing batch 1/782
Processing batch 2/782
Processing batch 3/782
Processing batch 4/782
Processing batch 5/782
Processing batch 6/782
Processing batch 7/782
Processing batch 8/782
Processing batch 9/782
Processing batch 10/782
Processing batch 11/782
Processing batch 12/782
Processing batch 13/782
Processing batch 14/782
Processing batch 15/782
Processing batch 16/782
Processing batch 17/782
Processing batch 18/782
Processing batch 19/782
Processing batch 20/782
Processing batch 21/782
Processing batch 22/782
Processing batch 23/782
Processing batch 24/782
Processing batch 25/782
Processing batch 26/782
Processing batch 27/782
Processing batch 28/782
Processing batch 29/782
Processing batch 30/782
Processing batch 31/782
Processing batch 32/782
Processing batch 33/782
Processing batch 34/782
Processing batch 35/782
Processing batch 36/782
Processing batch 37/782
Processing batch 38/782
Processing batch 39/782
Processing batch 40/782
Processing batch 41/782
Processing batch 42/782
Processing batch 43/782
Processing batch 44/782
Processing batch 45/782
Processing batch 46/782
Processing batch 47/782
Processing batch 48/782
Processing batch 49/782
Processing batch 50/782
Processing batch 51/782
Processing batch 52/782
Processing batch 53/782
Processing batch 54/782
Processing batch 55/782
Processing batch 56/782
Processing batch 57/782
Processing batch 58/782
```

```
features_list = [np.load(features_file) for features_file in sorted(features_path.glob("*.npy"))]
```

```
features = np.concatenate(features_list)
np.save(features_path / "features.npy", features)
```

```
photo_ids = pd.concat([pd.read_csv(ids_file) for ids_file in sorted(features_path.glob("*.csv"))])
photo_ids.to_csv(features_path / "photo_ids.csv", index=False)
```

```
model = TFCLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = AutoProcessor.from_pretrained("openai/clip-vit-base-patch32")
```

```
Downloading (…)lve/main/config.json: 100%                     4.19k/4.19k [00:00<00:00, 243kB/s]

Downloading tf_model.h5: 100%                          606M/606M [00:02<00:00, 239MB/s]
All model checkpoint layers were used when initializing TFCLIPModel.

All the layers of TFCLIPModel were initialized from the model checkpoint at openai/clip-vit-base-patch32.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFCLIPModel for predictions without further training.
Downloading (…)rocessor_config.json: 100%                     316/316 [00:00<00:00, 22.6kB/s]

Downloading (…)okenizer_config.json: 100%                     568/568 [00:00<00:00, 32.8kB/s]

Downloading (…)olve/main/vocab.json: 100%                     862k/862k [00:00<00:00, 7.04MB/s]

Downloading (…)olve/main/merges.txt: 100%                     525k/525k [00:00<00:00, 4.32MB/s]
```

```
photo_ids = pd.read_csv('/content/unsplash-dataset/lite/features/photo_ids.csv')
photo_ids = list(photo_ids['photo_id'])
photo = np.load('/content/unsplash-dataset/lite/features/features.npy')
print(f'Photos Loaded: {len(photo_ids)}')
photo_features = tf.convert_to_tensor(
    photo
)
```

```
Photos Loaded: 24999
```

```
def encode_search_query(query):
  encoded_text = processor(query, return_tensors = 'tf')
  encoded_text = model.get_text_features(**encoded_text)
  encoded_text /= tf.norm(encoded_text, axis = -1,  keepdims = True)
  return encoded_text



def find_best_match(text_features, photo_features, photo_ids, count = 3):
  similarities = tf.squeeze(photo_features @ tf.cast(tf.transpose(text_features), tf.float16))
  best_photo_idx = tf.argsort(-similarities)
  return [photo_ids[i] for i in best_photo_idx[:count]]


from IPython.display import Image
from IPython.core.display import HTML

def display_photo(photo_id):

  photo_image_url = f"https://unsplash.com/photos/{photo_id}/download?w=320"

  display(Image(url=photo_image_url))

  display(HTML(f'Photo on Unsplash '))
  print()


def search_unslash(search_query, photo_features, photo_ids, results_count=3):

  text_features = encode_search_query(search_query)
  best_photo_ids = find_best_match(text_features, photo_features, photo_ids, results_count)

  for photo_id in best_photo_ids:
    display_photo(photo_id)
```

```
!pip install jupyter-dash
```

```
Collecting jupyter-dash
  Downloading jupyter_dash-0.4.2-py3-none-any.whl (23 kB)
Collecting dash (from jupyter-dash)
  Downloading dash-2.13.0-py3-none-any.whl (10.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.4/10.4 MB 37.1 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from jupyter-dash) (2.31.0)
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (from jupyter-dash) (2.2.5)
Collecting retrying (from jupyter-dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from jupyter-dash) (7.34.0)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter-dash) (5.5.6)
Collecting ansi2html (from jupyter-dash)
  Downloading ansi2html-1.8.0-py3-none-any.whl (16 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from jupyter-dash) (1.5.7)
Collecting Werkzeug<2.3.0 (from dash->jupyter-dash)
  Downloading Werkzeug-2.2.3-py3-none-any.whl (233 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.6/233.6 kB 23.3 MB/s eta 0:00:00
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash->jupyter-dash) (5.15.0)
Collecting dash-html-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash->jupyter-dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash->jupyter-dash) (4.5.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash->jupyter-dash) (67.7.2)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from flask->jupyter-dash) (3.1.2)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from flask->jupyter-dash) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from flask->jupyter-dash) (8.1.7)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter-dash) (0.2.0)
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter-dash) (5.7.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter-dash) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter-dash) (6.3.2)
Collecting jedi>=0.16 (from ipython->jupyter-dash)
  Downloading jedi-0.19.0-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 51.5 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (3.0.39)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython->jupyter-dash) (4.8.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->jupyter-dash) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->jupyter-dash) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->jupyter-dash) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->jupyter-dash) (2023.7.22)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from retrying->jupyter-dash) (1.16.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython->jupyter-dash) (0.8.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->flask->jupyter-dash) (2.1.3)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython->jupyter-dash) (0.7.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash->jupyter-dash) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash->jupyter-dash) (23.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->jupyter-dash) (0.2.6)
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-client->ipykernel->jupyter-dash) (5.3.1)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (from jupyter-client->ipykernel->jupyter-dash) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client->ipykernel->jupyter-dash) (2.8.2)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.0->jupyter-client->ipykernel->jupyter-dash) (3.10.0)
```

Inicia uma aplicação Dash chamada "app" para criar a interface da web.

Carrega um modelo pré-treinado chamado "openai/clip-vit-base-patch32" para processamento de texto e imagens.

Carrega IDs de fotos e recursos de fotos previamente processados.

Define funções para codificar consultas de texto e imagem usando o modelo CLIP.

Define uma função para encontrar as melhores correspondências entre recursos de consulta e recursos de fotos.

Define uma função para exibir fotos com base nos IDs das fotos.

Define uma função que executa a pesquisa com base no tipo de consulta (texto ou URL de imagem) e retorna as fotos mais relevantes.

Define a interface da web com entrada de texto, seletor de tipo de consulta, botão de pesquisa e área de exibição de resultados.

Define duas funções de callback que respondem ao clique no botão de pesquisa e exibem os resultados na interface da web.

Este código cria uma aplicação web que usa o modelo CLIP da OpenAI para realizar pesquisas de imagens com base em texto ou URLs de imagens e exibe os resultados de forma interativa na interface da web.

```python
import dash
from jupyter_dash import JupyterDash
from dash import dcc, html
from dash.dependencies import Input, Output, State
import tensorflow as tf
from transformers import AutoProcessor, TFCLIPModel
import numpy as np
import pandas as pd
from PIL import Image
import requests

app = JupyterDash(__name__)

intro = """
Welcome to TF! Application NLP-based image search engine. The database contains 25k images from the Unsplash Dataset. You can search them:
-using a natural language description (e.g., animals in jungle)
-using Image URL.
The algorithm will return the nine most relevant images.

"""

getting_started = "To get started, simply enter your query type and text/url in the text box inside the sidebar and hit the search button. Our search engine will then scan unslash database of 25k image

model = TFCLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = AutoProcessor.from_pretrained("openai/clip-vit-base-patch32")

photo_ids = pd.read_csv('/content/unsplash-dataset/lite/features/photo_ids.csv')
photo_ids = list(photo_ids['photo_id'])
photo = np.load('/content/unsplash-dataset/lite/features/features.npy')
print(f'Photos Loaded: {len(photo_ids)}')
```

```
photo_features = tf.convert_to_tensor(photo)

def encode_text_query(query):
    encoded_text = processor(query, return_tensors='tf')
    encoded_text = model.get_text_features(**encoded_text)
    encoded_text /= tf.norm(encoded_text, axis=-1, keepdims=True)
    return encoded_text

def encode_image_query(query):
    image = Image.open(requests.get(query, stream=True).raw)
    encoded_image = processor(images=image, return_tensors='tf')
    encoded_image = model.get_image_features(**encoded_image)
    return encoded_image

def find_best_match(features, photo_features, photo_ids, count):
    similarities = tf.squeeze(photo_features @ tf.cast(tf.transpose(features), tf.float16))
    best_photo_idx = tf.argsort(-similarities)
    return [photo_ids[i] for i in best_photo_idx[:count]]

def display_photo(photo_id):
    photo_image_url = f"https://unsplash.com/photos/{photo_id}/download?w=240"
    return html.Img(src=photo_image_url)

def search_unslash(query, query_type, photo_features, photo_ids, results_count):
    if query_type == 'Image URL':
        features = encode_image_query(query)
    else:
        features = encode_text_query(query)
    best_photo_ids = find_best_match(features, photo_features, photo_ids, results_count)
    photos = [display_photo(photo_id) for photo_id in best_photo_ids]
    return photos

app.layout = html.Div([
    html.H1("Tarefa final Luan"),
    dcc.Store(id='results'),
    dcc.Store(id='query_type'),
    dcc.Input(id='query-input', type='text', value='two dogs playing in the snow'),
    dcc.RadioItems(id='query-type', options=[{'label': 'Text', 'value': 'Text'}, {'label': 'Image URL', 'value': 'Image URL'}], value='Text'),
    html.Button('Search', id='search-button'),
    html.Div(id='search-results'),
])

@app.callback(
    [Output('results', 'data'), Output('query_type', 'data')],
    [Input('search-button', 'n_clicks')],
    [State('query-input', 'value'), State('query-type', 'value')]
)
def search_images(n_clicks, query, query_type):
    if n_clicks is None:
        return None, None

    features = encode_image_query(query) if query_type == 'Image URL' else encode_text_query(query)
    best_photo_ids = find_best_match(features, photo_features, photo_ids, 9)
    return best_photo_ids, query_type

@app.callback(
    Output('search-results', 'children'),
    [Input('results', 'data')]
)
def display_results(best_photo_ids):
    if best_photo_ids is None:
        return html.Div([intro, html.H3('Getting Started'), getting_started, html.H3("OPENAI's CLIP(Contrastive Language-Image Pre-Training)"), html.Img(src='https://raw.githubusercontent.com/openai/CL

    photos = [display_photo(photo_id) for photo_id in best_photo_ids]
    return photos
```

```
/usr/local/lib/python3.10/dist-packages/dash/dash.py:525: UserWarning:

JupyterDash is deprecated, use Dash instead.
See https://dash.plotly.com/dash-in-jupyter for more details.

All model checkpoint layers were used when initializing TFCLIPModel.

All the layers of TFCLIPModel were initialized from the model checkpoint at openai/clip-vit-base-patch32.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFCLIPModel for predictions without further training.
Photos Loaded: 24999
```

URL da ferramenta:

```
app.run(jupyter_mode="external")
```

```
Dash app running on:
```