

## Introduction

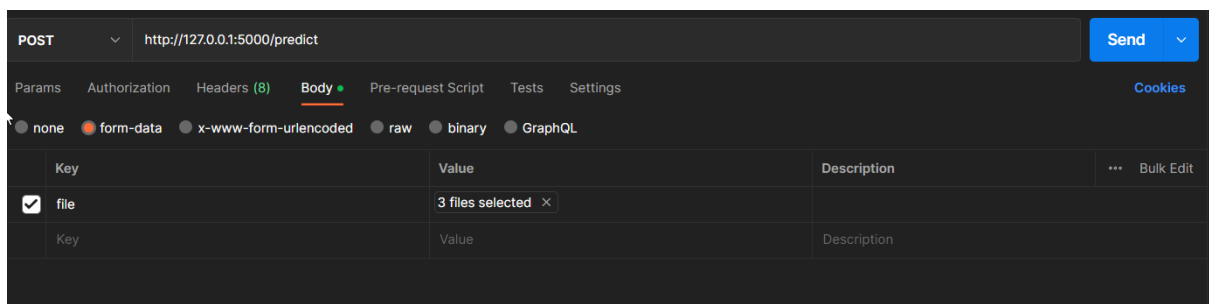
This documentation outlines the implementation of an image retrieval system using the Contrastive Language-Image Pre-Training (CLIP) model on the Flickr 30k dataset. The system is built using PyTorch and Hugging Face library, which is a transformer-based framework that leverages pre-trained deep learning models. The system is composed of two major components: training and evaluation on the dataset of the implementation of the CLIP model using BERT and ViT models for text and image embeddings, respectively, and finally, a RESTful inference API.

The goal of the project is to implement an image retrieval system that uses the CLIP model's similarity metric to find the most similar image and text for a given image. The model is trained on the Flickr 30k dataset, which consists of 31,783 images and captions, with each image having five captions that describe the image. The dataset is split into training and testing sets in an 80:20 ratio. The implementation is designed to handle multiple image inputs simultaneously.

The resulting implementation comprises an inference API and an inference script that uses curl to trigger the inference API. The script accepts one or more image files as input, sends them to the API, and as an output the most similar image and text for each image is returned. In addition, the script implements caching for the inference results to reduce the API's response time.

## Input

This is a POST API that accepts multiple image files as input using the form-data format. The URL for the API is <http://127.0.0.1:5000/predict>



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://127.0.0.1:5000/predict
- Body Type:** form-data (selected)
- Form Fields:**

Key	Value	Description
<input checked="" type="checkbox"/> file	3 files selected	
Key	Value	Description

## Output

Output will be provided in two forms :

- Json format :
  - o It will include the path of all the images

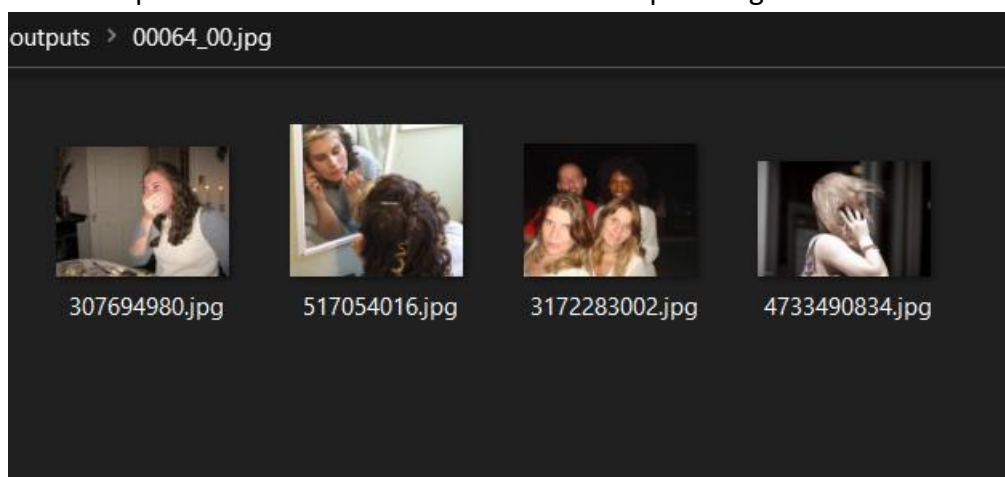
- Caption : Text provided by our model for the input image
- Matches : names of the images that are similar to input image and their most relevant text along with them according to our input image.

```

Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON
1
2 "image_path": "../flickr30k_images/flickr30k_images",
3 "output": {
4   "00064_00": {
5     "caption": "a woman in a black dress with a black bow tie",
6     "matches": {
7       "307694980.jpg": "A girl wearing a white sweater over a gray shirt is holding her hand over her mouth as she laughs .",
8       "3172283002.jpg": "Four people , two women in the front and a man and woman in the back , pose at night .",
9       "40425624.jpg": "A man in a short-sleeved white shirt is standing beside a man in a long-sleeved white shirt with the
10        sleeves rolled up .",
11       "4733490834.jpg": "A young woman with blond-hair and a colorful bracelet talks on a cellphone on a windy day .",
12       "517054016.jpg": "A young woman , with frosted brown hair , is putting on lipstick , while listening to someone on her
13        cellphone ."
14     }
15   }
16 }

```

- Storing the Output image in separate folder
  - Separate folder will be created for each input image.



## Setup

Create a Virtual Environment and and install all the packages using only one command i.e.

Pip install -r requirements.txt

Requirements.txt contains the following packages :

```
albumentations
Flask==2.1.0
flask_cors==3.0.10
Flask-Caching
matplotlib
numpy
opencv-python
pandas
pickle5
pillow
timm
torch==1.11.0 -f https://download.pytorch.org/whl/torch_stable.html
transformers==4.25.1
tqdm
```

**Dataset :** Download flickr30k\_images from the internet (kaggle)

**Clone :** git clone [https://github.com/shubhamMehla12/clip\\_proj.git](https://github.com/shubhamMehla12/clip_proj.git)

after cloning you just have to download two models named , clip\_bert\_vip.pt ,  
pytorch\_model.bin from gDrive and add them to the folder named “models” .

Gdrive link : [https://drive.google.com/drive/folders/1u2LmgzCWTlCdMW6q\\_XiE4cn-w7\\_wFsQB?usp=share\\_link](https://drive.google.com/drive/folders/1u2LmgzCWTlCdMW6q_XiE4cn-w7_wFsQB?usp=share_link)

Finally , to run the inference api run the following command in the terminal .

CMD : cd src

CMD : python main.py

## Technical documentation :

**Config.py :** in python scripts, config is a normal python file where I put all the hyperparameters. It is common convention to create a separate file, to store configuration parameters as variables. These variables can then be imported into your main Python script as needed.

**Utils.py :** It has AvgMeter class which is used to keep track of a metric's value, such as the loss or accuracy of a model during training or validation. It has three methods: \_\_init\_\_, reset, and update. The \_\_init\_\_ method initializes the metric name and calls reset to set the initial values to 0. The reset method resets the metric values to 0. The update method updates the metric with new values by adding the new values to the sum, incrementing the count, and calculating the average.

### Dataset.py :

- **CLIPDataset Class** : The CLIPDataset class is used to create a dataset of images and their captions for use in training or testing a Contrastive Language-Image Pre-Training (CLIP) model. It has two methods: `__init__` and `__getitem__`.
  - The `__init__` method initializes the dataset with a list of image filenames, a list of captions for the images, a tokenizer to encode the captions, and a set of image resize and normalization transforms.
  - The `__getitem__` method returns an item from the dataset at a given index. It returns a dictionary containing the encoded caption, the image tensor, and the original caption.
  - The `__len__` method returns length of the caption

### Train.py :

- Here are some functions to help us load train and valid dataloaders, our model and then train and evaluate our model on those. There's not much going on here; just simple training loop and utility functions

### Models.py :

- **Image encoder** : The image encoder code is straight forward. I'm using PyTorch Image Models library (timm) here which makes a lot of different image models available from ResNets to ViT and many more. Here I will use a `vit_base_patch32_plus_256` as our image encoder.
  - **ImageEncoder Class** : The ImageEncoder class has two methods. The `__init__` method initializes the ViT model specified by `model_name`. The `global_pool` argument is set to "avg", which means the output of the model is averaged across all spatial locations. The `trainable` parameter is used to set whether the model's parameters are trainable or not. If `trainable=True`, the parameters are set to be trainable, otherwise, they are not trainable.
    - The forward method takes an input tensor `x` representing the image and returns the encoded fixed-size vector obtained by passing the input through the ViT model.
- **Text encoder** : For the text encoder, I will be using DistilBERT. Similar to BERT, two additional tokens (CLS and SEP) will be included to indicate the beginning and end of a sentence. To obtain the complete representation of a sentence, as mentioned in the BERT and DistilBERT papers, I will extract the final representations of the CLS

token. The goal is to capture the overall meaning of the sentence or caption using this representation. Thinking it in this way, it is similar to what we did to images and converted them into a fixed size vector.

- In the case of DistilBERT (and also BERT) the output hidden representation for each token is a vector with size 768. So, the whole caption will be encoded in the CLS token representation whose size is 768.
- **TextEncode class :**
  - The `__init__` method initializes the model by loading a pre-trained model if `pretrained` is `True`, or creating a new model with default configuration otherwise. It also sets the `requires_grad` attribute of all the model's parameters to the value of the `trainable` flag. Finally, it sets the `target_token_idx` attribute to 0, indicating that the model's output will be based on the hidden representation of the first token ([CLS]).
  - The forward method is the main computation of the module, which takes in two inputs:
    - `input_ids`: a tensor of shape `(batch_size, sequence_length)` representing the input token IDs.
    - `attention_mask`: a tensor of shape `(batch_size, sequence_length)` representing the attention mask, where each value is either 0 or 1, indicating whether each token should be attended to or not.
  - The method feeds the `input_ids` and `attention_mask` tensors to the model and returns the hidden representation of the first token ([CLS]) of the last layer of the model. The returned tensor has a shape of `(batch_size, hidden_size)`, where `hidden_size` is the dimensionality of the hidden representation.
- **Projection Head :** I used Keras code example implementation ([https://keras.io/examples/nlp/nl\\_image\\_search/](https://keras.io/examples/nlp/nl_image_search/)) of projection head to write the following in PyTorch.
  - Now that we have encoded both our images and texts into fixed size vectors (896 for image and 768 for text) we need to bring (project) them into a new world with similar dimensions for both images and texts in order to be able to compare them and push apart the non-relevant image and texts and pull together those that match. So, the following code will bring the 896 and 768 dimensional vectors into a 224 (`projection_dim`) dimensional world, where we can compare them.
  - "`embedding_dim`" is the size of the input vector (896 for images and 768 for texts) and "`projection_dim`" is the size of the output vector which will be

224 for our case. For understanding the details of this part you can refer to the CLIP paper.

**Clip :**

- This code implements the main model for the image-text matching task and defines the loss function. The model encodes images and texts separately into fixed size vectors using the ImageEncoder and TextEncoder modules. Then, the ProjectionHead modules project these vectors into a shared space. The loss function measures the similarity between the two groups of vectors using the dot product (matrix multiplication) and cross entropy. The targets matrix is created by multiplying the transpose of text\_embeddings with image\_embeddings. Finally, the full matrix form of cross entropy is used to calculate the actual loss.
- In Linear Algebra, the dot product is a common way to measure the similarity of two vectors. The same principle is used here to measure the similarity of two groups of vectors (matrices). By transposing the second matrix, we can perform matrix multiplication and obtain a logits matrix with shape (batch\_size, batch\_size). This logits matrix is used to calculate the actual loss using the cross entropy function.
- Now that we've got our targets matrix, we will use simple cross entropy to calculate the actual loss. I've written the full matrix form of cross entropy as a function which you can see in the bottom of the code block. Okay! We are done! Wasn't it simple?! Alright, you can ignore the next paragraph but if you are curious, there is an important note in that.
- Here is why I did not use the simpler method for loss function :
  - I need to admit that there's a simpler way to calculate this loss in PyTorch; by doing this: `nn.CrossEntropyLoss()(logits, torch.arange(batch_size))`. Why I did not use it here? For 2 reasons.
    - The dataset we are using has multiple captions for a single image; so, there is the possibility that two identical images with their similar captions exist in a batch (it is rare but it can happen). Taking the loss with this easier method will ignore this possibility and the model learns to pull apart two representations (assume them different) that are actually the same. Obviously, we don't want this to happen so I calculated the whole target matrix in a way that takes care of these edge cases.
    - Doing it the way I did, gave me a better understanding of what is happening in this loss function; so, I thought it would give you a better intuition as well!

-

- **Clip Model loss:**

Training Loss	2.71
Validation loss	2.91