**Exploring CLIP: A Vision-Language Model (VLM) for Image Understanding**

[[CodeAddict](https://medium.com/@staytechrich?source=post_page---byline--9f1e506fe2fd---------------------------------------)](https://medium.com/@staytechrich?source=post_page---byline--9f1e506fe2fd---------------------------------------)

[CodeAddict](https://medium.com/@staytechrich?source=post_page---byline--9f1e506fe2fd---------------------------------------)

Follow

5 min read

·

Apr 1, 2025

In this tutorial, we’ll explore CLIP, a powerful Vision-Language Model (VLM) developed by OpenAI, which bridges images and text for tasks like scene description, classification, and more. We’ll use CLIP alongside YOLOv8 to analyze an image (cat-dog.jpeg), detect objects, match the scene to creative prompts, and annotate the result — all in a Jupyter Notebook.

**1. What Is CLIP?**

**1.1 Definition**

CLIP (Contrastive Language-Image Pretraining) is a **Vision-Language Model** that understands both images and text by learning to associate them in a shared space. Unlike traditional models that require specific training for each task, CLIP can perform “zero-shot” tasks — meaning it can handle new scenarios without retraining.

**1.2 What Kind of VLM Is It?**

* **Multimodal**: Combines vision (images) and language (text) processing.
* **Pretrained**: Trained on 400 million image-text pairs from the internet, making it versatile.
* **Zero-Shot Capable**: Uses text prompts to guide its understanding, no labeled data needed.

**1.3 What Does It Do?**

* Takes an image and a list of text prompts (e.g., “A calm scene,” “A wild adventure”).
* Outputs similarity scores, picking the best-matching prompt.
* Enables flexible tasks by simply changing the prompts.

**2. What We Do in This Tutorial**

We’ll:

* Detect objects in a sample image (cat-dog.jpeg) using YOLOv8.
* Generate a variety of prompts based on those objects (e.g., “A cat and dog sitting quietly”).
* Use CLIP to pick the best prompt.
* Annotate the image with boxes and the selected prompt.

Press enter or click to view image in full size

A screenshot of a dog

AI-generated content may be incorrect.

**3. Theory Behind CLIP**

**3.1 Architecture**

CLIP has two main components:

* **Image Encoder**: A **Vision Transformer (ViT)** or a ResNet variant (e.g., ViT-B/32 in this tutorial).
* Splits an image into patches (e.g., 32x32), processes them with Transformer layers, and outputs a single 512-dimensional vector.
* **Text Encoder**: A **Text Transformer**

**3.2 Contrastive Learning**

* **Training Goal**: Make vectors of matching image-text pairs (e.g., a dog photo and “A dog”) similar, and non-matching pairs dissimilar.

**Method**: Uses **contrastive loss**:

* For a batch of N pairs, it maximizes the cosine similarity of correct pairs while minimizing it for the N² — N incorrect pairs.

**4. What Can We Do with CLIP?**

* **Image Classification**: Label images with custom categories (e.g., “happy” vs. “sad”) using prompts.
* **Scene Description**: Match images to detailed narratives (e.g., “A cat napping on a sunny porch”).
* **Visual Question Answering**: Answer questions about images (e.g., “Is it raining?”).
* **Image Search**: Find images by text queries.
* **Content Moderation**: Detect inappropriate content with text prompts.

**5. Implementation**

* **Setup**: Install dependencies:

pip install torch torchvision ultralytics git+https://github.com/openai/CLIP.git opencv-python numpy pillow matplotlib

* **Import Libraries**

Get CodeAddict’s stories in your inbox

Join Medium for free to get updates from this writer.

Subscribe

import cv2  
import numpy as np  
import torch  
from ultralytics import YOLO  
import clip  
from PIL import Image  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
print("Libraries imported successfully!")

* **Load Models**

def load\_clip\_model():  
 model, preprocess = clip.load("ViT-B/32", device="cuda" if torch.cuda.is\_available() else "cpu")  
 return model, preprocess  
  
def load\_yolov8\_model():  
 return YOLO("yolov8m.pt")  
  
clip\_model, preprocess = load\_clip\_model()  
yolo\_model = load\_yolov8\_model()  
print("Models loaded: CLIP (ViT-B/32) and YOLOv8 (medium)")

* def load\_clip\_model(): Function to load CLIP.
* clip.load(“ViT-B/32”, …): Loads Vision Transformer (base, 32x32 patches), returns model and preprocessing function.
* **Generate Prompts**

def generate\_prompts(detected\_objects):  
 if not detected\_objects:  
 return ["An empty scene with no objects"]  
 base\_prompts = [  
 "A {objects} sitting quietly in a room",  
 "A calm scene with {objects} doing nothing",  
 "A {objects} resting peacefully",  
 "A {objects} standing still outdoors",  
 "A {objects} running around wildly",  
 "A busy scene with {objects} interacting",  
 "A {objects} jumping in excitement",  
 "A {objects} sleeping under a tree",  
 "A {objects} eating food together",  
 "A {objects} watching something intently",  
 "A {objects} surfing on waves",  
 "A {objects} flying in the sky",  
 "A {objects} performing tricks",  
 "A {objects} climbing a hill",  
 "A {objects} swimming in a pool"  
 ]  
 unique\_objects = set(detected\_objects)  
 object\_str = " and ".join(unique\_objects)  
 return [prompt.format(objects=object\_str) for prompt in base\_prompts]  
  
test\_objects = ["cat", "dog"]  
prompts = generate\_prompts(test\_objects)  
print("Generated prompts (sample):", prompts[:5], "...and", len(prompts)-5, "more")

* base\_prompts = […]: 15 diverse prompts.
* unique\_objects = set(…): Removes duplicates.
* object\_str = …: Joins objects (e.g., “cat and dog”).
* return […]: Formats prompts.
* test\_objects = …: Tests with “cat” and “dog”.
* prompts = …: Generates prompts.
* **Extract Image Features and Detect Objects**

def extract\_image\_features(image\_path, clip\_model, preprocess, yolo\_model):  
 image = cv2.imread(image\_path)  
 image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 results = yolo\_model(image\_rgb, verbose=True)  
 detected\_objects = [results[0].names[int(box.cls)] for box in results[0].boxes]  
 print("Detected objects:", detected\_objects)  
 image\_pil = Image.fromarray(image\_rgb)  
 image\_tensor = preprocess(image\_pil).unsqueeze(0).to(clip\_model.logit\_scale.device)  
 with torch.no\_grad():  
 image\_features = clip\_model.encode\_image(image\_tensor)  
 return image\_features, detected\_objects, image\_rgb, results[0].boxes  
  
image\_path = "cat-dog.jpeg"  
image\_features, detected\_objects, image\_rgb, boxes = extract\_image\_features(image\_path, clip\_model, preprocess, yolo\_model)  
  
plt.figure(figsize=(10, 6))  
plt.imshow(image\_rgb)  
for i, box in enumerate(boxes):  
 x1, y1, x2, y2 = map(int, box.xyxy[0])  
 plt.plot([x1, x2, x2, x1, x1], [y1, y1, y2, y2, y1], 'g-')  
 plt.text(x1, y1-10, f"{detected\_objects[i]}", color='green')  
plt.axis('off')  
plt.show()

* image = cv2.imread(…): Loads your image.
* image\_rgb = …: Converts to RGB.
* results = yolo\_model(…): Detects objects.
* detected\_objects = …: Extracts names.
* print(…): Shows objects.
* image\_pil = …: Prepares for CLIP.
* image\_tensor = …: Preprocesses and adds batch dimension.
* image\_features = …: Gets 512D vector.

**Output:**

Press enter or click to view image in full size

A dog and cat looking at camera

AI-generated content may be incorrect.

* **Match Scene to Prompt**s

def match\_scene(image\_features, detected\_objects, clip\_model, query\_prompts):  
 device = clip\_model.logit\_scale.device  
 text\_tokens = clip.tokenize(query\_prompts).to(device)  
 with torch.no\_grad():  
 text\_features = clip\_model.encode\_text(text\_tokens)  
 similarity = (image\_features @ text\_features.T).softmax(dim=-1).cpu().numpy()  
 best\_prompt\_idx = similarity.argmax()  
 best\_score = similarity.max()  
 print("All prompts:", query\_prompts)  
 print("Similarity scores:", similarity)  
 return query\_prompts[best\_prompt\_idx], best\_score, detected\_objects  
  
prompts = generate\_prompts(detected\_objects)  
query\_prompts = prompts + ["A chaotic scene", "An empty area"]  
best\_prompt, score, objects = match\_scene(image\_features, detected\_objects, clip\_model, query\_prompts)  
print(f"Best match: '{best\_prompt}' (Score: {score:.2f})")

**Output: (CLIP picks the best prompt.)**

Similarity scores (first 5 and last 5): [[ 0.0033512 0.034393 0.00050592 0.32129 2.3246e-06 0.0014639 3.612e-05 2.9802e-07 0.0028229 0.0033512 5.9605e-08 3.9637e-05 3.3379e-06 9.8944e-06 2.3842e-07 0.62891 0.0039787]] ... [[ 0.0033512 0.034393 0.00050592 0.32129 2.3246e-06 0.0014639 3.612e-05 2.9802e-07 0.0028229 0.0033512 5.9605e-08 3.9637e-05 3.3379e-06 9.8944e-06 2.3842e-07 0.62891 0.0039787]]  
Best match: 'A chaotic pet scene' (Score: 0.63)

* text\_tokens = …: Tokenizes prompts.
* text\_features = …: Gets 512D vectors.
* similarity = …: Computes scores.
* best\_prompt\_idx = …: Finds best match.
* best\_score = …: Gets score.
* prompts = …: Generates prompts.
* query\_prompts = …: Adds static prompts.
* best\_prompt, … = …: Matches scene.
* **Annotate and Save Image**

def annotate\_image(image\_rgb, boxes, best\_prompt, score, objects):  
 scale\_factor = 2  
 image\_rgb\_resized = cv2.resize(image\_rgb, None, fx=scale\_factor, fy=scale\_factor, interpolation=cv2.INTER\_LINEAR)  
 image\_bgr = cv2.cvtColor(image\_rgb\_resized, cv2.COLOR\_RGB2BGR)  
 for i, box in enumerate(boxes):  
 x1, y1, x2, y2 = [int(coord \* scale\_factor) for coord in box.xyxy[0]]  
 label = objects[i]  
 cv2.rectangle(image\_bgr, (x1, y1), (x2, y2), (0, 255, 0), 2)  
 cv2.putText(image\_bgr, label, (x1, y1-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)  
 text = f"Prompt: {best\_prompt} | Score: {score:.2f} | Objects: {', '.join(objects)}"  
 font\_scale = 0.7  
 thickness = 2  
 text\_size, baseline = cv2.getTextSize(text, cv2.FONT\_HERSHEY\_SIMPLEX, font\_scale, thickness)  
 text\_width, text\_height = text\_size  
 padding = 10  
 rect\_height = text\_height + 2 \* padding  
 cv2.rectangle(image\_bgr, (5, 5), (image\_bgr.shape[1] - 5, 5 + rect\_height), (0, 0, 0), -1)  
 text\_x = 10  
 text\_y = 5 + text\_height + padding // 2  
 cv2.putText(image\_bgr, text, (text\_x, text\_y), cv2.FONT\_HERSHEY\_SIMPLEX, font\_scale, (0, 255, 0), thickness)  
 output\_path = "/workspace/test/10\_vlm/pet\_analysis.jpg"  
 cv2.imwrite(output\_path, image\_bgr)  
 return cv2.cvtColor(image\_bgr, cv2.COLOR\_BGR2RGB)  
  
annotated\_image = annotate\_image(image\_rgb, boxes, best\_prompt, score, objects)  
plt.figure(figsize=(12, 8))  
plt.imshow(annotated\_image)  
plt.axis('off')  
plt.show()  
print("Annotated image saved to '/workspace/test/10\_vlm/pet\_analysis.jpg'")

Output:

Press enter or click to view image in full size

A screenshot of a dog and a dog with text

AI-generated content may be incorrect.

[Vlm](https://medium.com/tag/vlm?source=post_page-----9f1e506fe2fd---------------------------------------)

[Clip](https://medium.com/tag/clip?source=post_page-----9f1e506fe2fd---------------------------------------)

[Visionlanguagemodel](https://medium.com/tag/visionlanguagemodel?source=post_page-----9f1e506fe2fd---------------------------------------)

[Zeroshotlearning](https://medium.com/tag/zeroshotlearning?source=post_page-----9f1e506fe2fd---------------------------------------)