

## **Azure Computer Vision API**

Es un servicio de Microsoft Azure que permite analizar imágenes y detectar objetos, personas, y más. Requiere una clave de suscripción y un endpoint.

### **Ventajas:**

Precisión alta.

Soporte para múltiples características como descripción de imágenes, detección de objetos, OCR, etc.

### **Desventajas:**

No es completamente gratuito (tiene un nivel gratuito limitado).

Requiere conexión a internet y configuración de una cuenta de Azure.

## **Hugging Face Transformers (Modelo facebook/detr-resnet-50)**

Es un modelo de detección de objetos basado en DETR (DEtection TRansformer) que se ejecuta localmente utilizando PyTorch. No requiere conexión a internet ni claves de API.

### **Ventajas:**

Completamente gratuito.

Se ejecuta localmente, lo que garantiza privacidad.

Fácil de usar con la biblioteca transformers de Hugging Face.

### **Desventajas:**

Requiere instalar dependencias adicionales como torch y transformers.

Puede ser más lento en máquinas sin GPU.

## **Clarifai**

Clarifai es una plataforma de inteligencia artificial especializada en visión por computadora y procesamiento de imágenes/videos, que ofrece APIs y herramientas para el reconocimiento de objetos, clasificación de contenido y más.

## **Ventajas**

### **Fácil integración**

Ofrece APIs bien documentadas y SDKs para múltiples lenguajes (Python, JavaScript, etc.), lo que facilita su implementación en aplicaciones.

Compatibilidad con plataformas como AWS, Google Cloud y Azure.

### **Modelos preentrenados**

Proporciona modelos listos para usar en detección de objetos, reconocimiento facial, moderación de contenido (NSFW), texto en imágenes, etc.

Útil para proyectos rápidos sin necesidad de entrenar modelos desde cero.

### **Personalización**

Permite entrenar modelos personalizados con tus propios datos, adaptándose a necesidades específicas.

## **Desventajas**

### **Costo elevado en escalamiento**

Aunque tiene un plan gratuito (limitado a 1,000 operaciones/mes), los precios suben rápidamente según el volumen de consultas.

Puede ser caro para startups o proyectos pequeños.

### **Precisión limitada en casos complejos**

Sus modelos preentrenados no siempre son tan precisos como soluciones custom (ej. detección de objetos en ángulos raros o imágenes con ruido).

### **Dependencia de la nube**

La mayoría de sus servicios son cloud-based, lo que puede ser un problema si necesitas procesamiento offline.

## Imágenes del código y funcionamiento

```
1 #
2 import cv2
3 from clarifai.grpc.channel.clarifai_channel import ClarifaiChannel
4 from clarifai.grpc.grpc_api import resources_pb2, service_pb2, service_pb2_grpc
5 from clarifai.grpc.grpc_api.status import status_code_pb2
6
7 # --- Configuración ---
8 CLARIFAI_USER_ID = 'me8es189f' # Reemplaza con tu ID de Usuario de Clarifai
9 CLARIFAI_APP_ID = '96959cabdd445793cc72a76238ff' # Reemplaza con tu ID de Aplicación de Clarifai
10 CLARIFAI_PAT = 'ac9ec5de72ba1da54886c1dffc694a1' # Reemplaza con tu Token de Acceso Personal de Clarifai
11
12 # Modelo de Clarifai para detección general (puedes elegir otros modelos)
13 MODEL_ID = 'general-image-detection' # Ejemplo: 'general-image-recognition' o un modelo personalizado
14 MODEL_VERSION_ID = '' # Opcional: establece una versión específica del modelo
15
16 IMAGE_FILE_PATH = 'imagen_capturada.jpg'
17
18 def capture_image_from_webcam(output_path):
19     """Captura una imagen desde la cámara web y la guarda."""
20     cap = cv2.VideoCapture(0) # 0 es usualmente la cámara web por defecto
21
22     if not cap.isOpened():
23         print("Error: No se pudo abrir la cámara web.")
24         return False
25
26     print("Presiona 's' para guardar la imagen, 'q' para salir.")
27
28     while True:
29         ret, frame = cap.read()
30         if not ret:
31             print("Error: No se puede recibir el fotograma (¿fin de la transmisión?). Saliendo...")
32             break
33
34         cv2.imshow('webcam - Presiona "s" para guardar, "q" para salir', frame)
35
36         key = cv2.waitKey(1) & 0xFF
37         if key == ord('s'):
38             # Guardar la imagen
39             cv2.imwrite(output_path, frame)
40             print(f"Imagen guardada en {output_path}")
41             break
42         elif key == ord('q'):
43             # Salir
44             cap.release()
45             cv2.destroyAllWindows()
46             return False
47
48     cap.release()
49     cv2.destroyAllWindows()
50     return True
51
52 def detect_objects_with_clarifai(image_path, user_id, app_id, pat, model_id, model_version_id):
53     """Envía una imagen a la API de Clarifai para detección de objetos e imprime los resultados."""
54     channel = ClarifaiChannel.get_grpc_channel()
55     stub = service_pb2_grpc.V2Stub(channel)
56     metadata = (('authorization', 'key ' + pat),)
57
58     try:
59         with open(image_path, "rb") as f:
60             file_bytes = f.read()
61     except FileNotFoundError:
62         print("Error: Archivo de imagen no encontrado en (image_path)")
63         return
64
65     post_model_outputs_response = stub.PostModelOutputsRequest(
66         service_pb2.PostModelOutputsRequest(
67             user_app_id=resources_pb2.UserAppIDSet(user_id=user_id, app_id=app_id),
68             model_id=model_id,
69             version_id=model_version_id, # Opcional
70             inputs=[
71                 resources_pb2.Input(
72                     data=resources_pb2.Data(
73                         image=resources_pb2.Image(
74                             data=file_bytes
75                         )
76                     )
77                 )
78             ],
79             metadata=metadata
80         )
81     )
82
83     if post_model_outputs_response.status.code != status_code_pb2.SUCCESS:
84         print(f"La solicitud PostModelOutputs falló, estado: {post_model_outputs_response.status.description}")
85         # print("Error detallado:", post_model_outputs_response.status) # Para más detalles
86         return
87
88     print("\nConceptos detectados:")
89     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.concepts:
90         for concept in post_model_outputs_response.outputs[0].data.concepts:
91             print(f"  {concept.name}: {concept.value:.2f}")
92     else:
93         print("No se detectaron conceptos o hay un error en la estructura de la respuesta.")
94
95     # Para modelos que proporcionan detecciones basadas en regiones (como detección de objetos)
96     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.regions:
97         print("Regiones detectadas (objetos):")
98         for region in post_model_outputs_response.outputs[0].data.regions:
99             # Imprimir el concepto con la mayor confianza en la región
100             if region.data.concepts:
101                 best_concept = region.data.concepts[0] # Asumiendo que está ordenado por confianza
102                 print(f"  Objeto: {best_concept.name} (confianza: {best_concept.value:.2f})")
103                 # Las coordenadas del cuadro delimitador son relativas al tamaño de la imagen (0.0 a 1.0)
104                 # top_row, left_col, bottom_row, right_col
105                 print(f"  Cuadro delimitador: Superior={region.region_info.bounding_box.top_row:.2f}, "
106                       f"  Izquierda={region.region_info.bounding_box.left_col:.2f}, "
107                       f"  Inferior={region.region_info.bounding_box.bottom_row:.2f}, "
108                       f"  Derecha={region.region_info.bounding_box.right_col:.2f}")
109             else:
110                 print(f"  Región detectada pero sin conceptos asociados.")
111     elif not post_model_outputs_response.outputs[0].data.concepts: # Si tampoco hay conceptos generales
112         print("No se detectaron objetos ni conceptos generales.")
113
114
```

```
10 def capture_image_from_webcam(output_path):
11     cv2.imwrite(output_path, frame)
12     print(f"Imagen guardada como {output_path}")
13     break
14
15     elif key == ord('q'):
16         print("Saliendo de la captura de la cámara web.")
17         cap.release()
18         cv2.destroyAllWindows()
19         return False
20
21     cap.release()
22     cv2.destroyAllWindows()
23     return True
24
25 def detect_objects_with_clarifai(image_path, user_id, app_id, pat, model_id, model_version_id):
26     """Envía una imagen a la API de Clarifai para detección de objetos e imprime los resultados."""
27     channel = ClarifaiChannel.get_grpc_channel()
28     stub = service_pb2_grpc.V2Stub(channel)
29     metadata = (('authorization', 'key ' + pat),)
30
31     try:
32         with open(image_path, "rb") as f:
33             file_bytes = f.read()
34     except FileNotFoundError:
35         print("Error: Archivo de imagen no encontrado en (image_path)")
36         return
37
38     post_model_outputs_response = stub.PostModelOutputsRequest(
39         service_pb2.PostModelOutputsRequest(
40             user_app_id=resources_pb2.UserAppIDSet(user_id=user_id, app_id=app_id),
41             model_id=model_id,
42             version_id=model_version_id, # Opcional
43             inputs=[
44                 resources_pb2.Input(
45                     data=resources_pb2.Data(
46                         image=resources_pb2.Image(
47                             data=file_bytes
48                         )
49                     )
50                 )
51             ],
52             metadata=metadata
53         )
54     )
55
56     if post_model_outputs_response.status.code != status_code_pb2.SUCCESS:
57         print(f"La solicitud PostModelOutputs falló, estado: {post_model_outputs_response.status.description}")
58         # print("Error detallado:", post_model_outputs_response.status) # Para más detalles
59         return
60
61     print("\nConceptos detectados:")
62     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.concepts:
63         for concept in post_model_outputs_response.outputs[0].data.concepts:
64             print(f"  {concept.name}: {concept.value:.2f}")
65     else:
66         print("No se detectaron conceptos o hay un error en la estructura de la respuesta.")
67
68     # Para modelos que proporcionan detecciones basadas en regiones (como detección de objetos)
69     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.regions:
70         print("Regiones detectadas (objetos):")
71         for region in post_model_outputs_response.outputs[0].data.regions:
72             # Imprimir el concepto con la mayor confianza en la región
73             if region.data.concepts:
74                 best_concept = region.data.concepts[0] # Asumiendo que está ordenado por confianza
75                 print(f"  Objeto: {best_concept.name} (confianza: {best_concept.value:.2f})")
76                 # Las coordenadas del cuadro delimitador son relativas al tamaño de la imagen (0.0 a 1.0)
77                 # top_row, left_col, bottom_row, right_col
78                 print(f"  Cuadro delimitador: Superior={region.region_info.bounding_box.top_row:.2f}, "
79                       f"  Izquierda={region.region_info.bounding_box.left_col:.2f}, "
80                       f"  Inferior={region.region_info.bounding_box.bottom_row:.2f}, "
81                       f"  Derecha={region.region_info.bounding_box.right_col:.2f}")
82             else:
83                 print(f"  Región detectada pero sin conceptos asociados.")
84     elif not post_model_outputs_response.outputs[0].data.concepts: # Si tampoco hay conceptos generales
85         print("No se detectaron objetos ni conceptos generales.")
86
87
```

```
79 def detect_objects_with_clarifai(image_path, user_id, app_id, pat, model_id, model_version_id):
80     """Envía una imagen a la API de Clarifai para detección de objetos e imprime los resultados."""
81     channel = ClarifaiChannel.get_grpc_channel()
82     stub = service_pb2_grpc.V2Stub(channel)
83     metadata = (('authorization', 'key ' + pat),)
84
85     try:
86         with open(image_path, "rb") as f:
87             file_bytes = f.read()
88     except FileNotFoundError:
89         print("Error: Archivo de imagen no encontrado en (image_path)")
90         return
91
92     post_model_outputs_response = stub.PostModelOutputsRequest(
93         service_pb2.PostModelOutputsRequest(
94             user_app_id=resources_pb2.UserAppIDSet(user_id=user_id, app_id=app_id),
95             model_id=model_id,
96             version_id=model_version_id, # Opcional
97             inputs=[
98                 resources_pb2.Input(
99                     data=resources_pb2.Data(
100                         image=resources_pb2.Image(
101                             data=file_bytes
102                         )
103                     )
104                 )
105             ],
106             metadata=metadata
107         )
108     )
109
110     if post_model_outputs_response.status.code != status_code_pb2.SUCCESS:
111         print(f"La solicitud PostModelOutputs falló, estado: {post_model_outputs_response.status.description}")
112         # print("Error detallado:", post_model_outputs_response.status) # Para más detalles
113         return
114
115     print("\nConceptos detectados:")
116     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.concepts:
117         for concept in post_model_outputs_response.outputs[0].data.concepts:
118             print(f"  {concept.name}: {concept.value:.2f}")
119     else:
120         print("No se detectaron conceptos o hay un error en la estructura de la respuesta.")
121
122     # Para modelos que proporcionan detecciones basadas en regiones (como detección de objetos)
123     if post_model_outputs_response.outputs and post_model_outputs_response.outputs[0].data.regions:
124         print("Regiones detectadas (objetos):")
125         for region in post_model_outputs_response.outputs[0].data.regions:
126             # Imprimir el concepto con la mayor confianza en la región
127             if region.data.concepts:
128                 best_concept = region.data.concepts[0] # Asumiendo que está ordenado por confianza
129                 print(f"  Objeto: {best_concept.name} (confianza: {best_concept.value:.2f})")
130                 # Las coordenadas del cuadro delimitador son relativas al tamaño de la imagen (0.0 a 1.0)
131                 # top_row, left_col, bottom_row, right_col
132                 print(f"  Cuadro delimitador: Superior={region.region_info.bounding_box.top_row:.2f}, "
133                       f"  Izquierda={region.region_info.bounding_box.left_col:.2f}, "
134                       f"  Inferior={region.region_info.bounding_box.bottom_row:.2f}, "
135                       f"  Derecha={region.region_info.bounding_box.right_col:.2f}")
136             else:
137                 print(f"  Región detectada pero sin conceptos asociados.")
138     elif not post_model_outputs_response.outputs[0].data.concepts: # Si tampoco hay conceptos generales
139         print("No se detectaron objetos ni conceptos generales.")
140
141
```

```
File Edit Selection View Go Run Terminal Help
C:\Users\ASUS> Documents > Proyectos python > El reto.py > detect_objects_with_clarifai
114
115 if __name__ == "__main__":
116     print("Iniciando captura de fotos con la cámara web...")
117     if capture_image_from_webcam(IMAGE_FILE_PATH):
118         print("Imagen capturada: (IMAGE_FILE_PATH)")
119         print("Enviando imagen a Clarifai para detección...")
120         detect_objects_with_clarifai(
121             IMAGE_FILE_PATH,
122             CLARIFAI_USER_ID,
123             CLARIFAI_APP_ID,
124             CLARIFAI_PAT,
125             MODEL_ID,
126             MODEL_VERSION_ID
127         )
128     else:
129         print("La captura de imagen fue cancelada o falló.")
130
131 print("El programa finalizado.")
132
```

