```
%%capture
!pip install torch torchvision transformers pdfplumber pymupdf pillow opencv-python pdf2image
!pip install -q git+https://github.com/huggingface/transformers.git@1931a351408dbd1d0e2c4d6d7ee0eb5e8807d7bf
!pip install -q qwen-vl-utils accelerate huggingface_hub pillow ipywidgets
!apt-get install poppler-utils -y
%pip install bitsandbytes
```

```
import os
import torch
import cv2
from PIL import Image, ImageEnhance
import fitz
import pdfplumber
from pdf2image import convert_from_path
from transformers import AutoProcessor
from transformers import Qwen2_5_VLForConditionalGeneration, AutoTokenizer, AutoProcessor
from qwen_vl_utils import process_vision_info
import json
import gc
from pdfminer.high_level import extract_text
from pdfminer.layout import LAParams
import torch, gc
from transformers import AutoProcessor, AutoModelForVision2Seq, BitsAndBytesConfig


INPUT_DIR = "/content/dataset"
OUTPUT_DIR = "/content/output"
os.makedirs(INPUT_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

```
print(torch.__version__)
print("CUDA available:", torch.cuda.is_available())
device = "cuda" if torch.cuda.is_available() else "cpu"

def clear_gpu_memory():
    torch.cuda.empty_cache()
    gc.collect()

# Load model and processor
clear_gpu_memory()
```

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

model_path = "Qwen/Qwen2.5-VL-7B-Instruct"
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
#=============== UTILITIES ===============
def is_text_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    for page in doc:
        if page.get_text().strip():
            return True
    return False

def sort_text_by_position(ocr_results):

    sorted_results = sorted(
        ocr_results,
        key=lambda x: (x['bbox'][1], x['bbox'][0])  # y1, x1
    )
    combined_text = "\n".join([res['text'] for res in sorted_results])
    return combined_text

def clean_image(image_path: str):
    original_pil = Image.open(image_path).convert("RGB")
```

```
        original_pil = Image.open(image_path).convert("RGB")
        image = cv2.imread(image_path)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray, (3, 3), 0)
        pil_image = Image.fromarray(blurred)
        enhancer = ImageEnhance.Contrast(pil_image)
        enhanced_image = enhancer.enhance(2.0)
        return original_pil, enhanced_image

    # def run_qwen_ocr(pil_img, prompt=None):
    #     if prompt is None:
    #                 prompt = (
    #     "You are an expert medical vision-language model.\n"
    #     "Extract all information from this medical image or report.\n"
    #     "- Extract all text including patient name, ID, doctor name, date, age, gender, and any other metadata.\n"
    #     "- If any part appears structured like a table (even without visible grid lines), extract it row-wise and column-wise.\n"
    #     "- Preserve the natural layout and hierarchy of the content.\n"
    #     "- Do not skip any heading or paragraph even if it's outside the table.\n"
    #     "- Extract all meaningful text regardless of format or placement."
    # )
    #     messages = [
    #         {
    #             "role": "user",
    #             "content": [
    #                 {
    #                     "type": "image",
    #                     "image": pil_img,

    #                 },
    #                 {"type": "text", "text": prompt},
    #             ],
    #         }
    #     ]
    #     text_prompt = processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
    #     image_inputs, video_inputs = process_vision_info(messages)
    #     inputs = processor(
    #         text=[text_prompt],
    #         images=image_inputs,
    #         videos=video_inputs,
    #         padding=True,
    #         return_tensors="pt",
    #     ).to(device)

    #     model.to(device)
    #     generated_ids = model.generate(**inputs, max_new_tokens=2048)
    #     generated_ids_trimmed = [
    #         out_ids[len(in_ids):] for in_ids, out_ids in zip(inputs.input_ids, generated_ids)
    #     ]
    #     output_text = processor.batch_decode(
    #         generated_ids_trimmed, skip_special_tokens=True, clean_up_tokenization_spaces=False
    #     )
    #     return output_text[0]
    def run_qwen_ocr(pil_img, prompt=None):
        if prompt is None:
            prompt = (
                "You are a highly knowledgeable medical assistant with expertise in understanding medical reports, even when the text is uncl
                "Instructions:\n"
                "- Extract all visible text from the image.\n"
                "- If any part of the image is blurry, faint, covered, or incomplete, use your medical expertise to intelligently guess or re
                "- Predict common medical terms if only partial characters are visible.\n"
                "- Include all metadata like Patient Name, Age, Gender, Doctor Name, ID, Date, Hospital Name, Report Type.\n"
                "- For tables or measurements, infer missing values based on format.\n"
                "- Do not skip any section.\n"
                "- Your goal is to produce a complete, human-readable version of the report, even if parts are unclear."
            )

        messages = [
            {
                "role": "user",
                "content": [
                    {"type": "image", "image": pil_img},
                    {"type": "text", "text": prompt},
                ],
            }
        ]
        text_prompt = processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
        image_inputs, video_inputs = process_vision_info(messages)
        inputs = processor(
            text=[text_prompt],
            images=image_inputs,
            videos=video_inputs,
            padding=True,
            return_tensors="pt",
```

```
        ).to(device)

        model.to(device)
        generated_ids = model.generate(
            **inputs,
            max_new_tokens=2048,
            do_sample=True,
            temperature=0.7,
            top_p=0.9
        )
        generated_ids_trimmed = [
            out_ids[len(in_ids):] for in_ids, out_ids in zip(inputs.input_ids, generated_ids)
        ]
        output_text = processor.batch_decode(
            generated_ids_trimmed, skip_special_tokens=True, clean_up_tokenization_spaces=False
        )[0]

        return output_text.strip()


    def classify_report_type(ocr_text):
        prompt = f"""
    You are an expert in clinical documentation. Identify the type of the following medical report.
    Use the content, structure, and terminology to decide.

    - Medical History
    - Physical Examination
    - Laboratory Test
    - Imaging
    - Pathology
    - Operative
    - Discharge Summary
    - Consultation
    - Progress Note
    - Emergency Department
    - Medication report
    - Prescription report

    Return only the exact report type name from the list. Do not explain.

    Report:
    {ocr_text}
    """
        messages = [{"role": "user", "content": [{"type": "text", "text": prompt}]}]
        text_prompt = processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
        inputs = processor(text=[text_prompt], return_tensors="pt").to(device)
        output = model.generate(**inputs, max_new_tokens=64)
        decoded = processor.batch_decode(output[:, inputs.input_ids.shape[-1]:], skip_special_tokens=True)[0]
        return decoded.strip()



    def generate_structured_json(ocr_text, report_type, image_path="", confidence=0.9):
        prompt = f"""
    You are a medical data extractor.
        Convert the following {report_type} into a structured JSON format.
        Arrange the extracted text logically:
        - For tables, extract data row-wise and column-wise as arrays.
        - For normal text, organize as paragraphs or sections.
        Handle potential OCR errors by making reasonable assumptions about handwritten text.
        Only return valid JSON. No explanations.


    Medical Report:
    {ocr_text}
    """
        messages = [{"role": "user", "content": [{"type": "text", "text": prompt}]}]
        text_prompt = processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
        # Changed DEVICE to device to match the defined variable
        inputs = processor(text=[text_prompt], images=None, return_tensors="pt").to(device)
        output = model.generate(
                                **inputs,
                                max_new_tokens=4096,
                                )
        decoded = processor.batch_decode(output[:, inputs.input_ids.shape[-1]:], skip_special_tokens=True)[0]
        return decoded.strip()
    def extract_and_store_json(image_path):
        pil_img = Image.open(image_path).convert("RGB")
        ocr_text = run_qwen_ocr(pil_img)
        report_type = classify_report_type(ocr_text)
        json_output = generate_structured_json(ocr_text, report_type, image_path=image_path)
        return json_output, report_type, ocr_text
```

```
def extract_and_store_json(image_path):
    _, cleaned_img = clean_image(image_path)
    ocr_text = run_qwen_ocr(cleaned_img)
    report_type = classify_report_type(ocr_text)
    json_output = generate_structured_json(ocr_text, report_type, image_path=image_path)
    return json_output, report_type, ocr_text


def process_input_file(file_path):
    name, ext = os.path.splitext(os.path.basename(file_path))
    if ext.lower() in [".jpg", ".jpeg", ".png"]:
        print(f"[IMG] Processing image: {file_path}")
        _, cleaned_img = clean_image(file_path)
        output_text = run_qwen_ocr(cleaned_img)
        # Detect report type for image files
        report_type = classify_report_type(output_text)
        structured_json = generate_structured_json(output_text, report_type=report_type, image_path=file_path, confidence=0.9)

        # Save OCR Text
        ocr_path = os.path.join(OUTPUT_DIR, f"{name}_ocr.txt")
        with open(ocr_path, "w", encoding="utf-8") as f:
            f.write(output_text)
        print(f"Saved OCR output: {ocr_path}")

        # Save Structured JSON
        json_path = os.path.join(OUTPUT_DIR, f"{name}_structured.json")
        with open(json_path, "w", encoding="utf-8") as f:
            f.write(structured_json)
        print(f"Saved structured JSON: {json_path}")

    elif ext.lower() == ".pdf":
        if is_text_pdf(file_path):
            print(f"[PDF] Text-based PDF: {file_path}")
            with pdfplumber.open(file_path) as pdf:
                for i, page in enumerate(pdf.pages):
                    output_text = page.extract_text()
                    if output_text:
                        # Detect report type for text-based PDF pages
                        report_type = classify_report_type(output_text)
                        structured_json = generate_structured_json(output_text, report_type=report_type, image_path=f"{file_path}_page_

                        # Save OCR Text for each page
                        # ocr_path = os.path.join(OUTPUT_DIR, f"{name}_page_{i+1}_ocr.txt")
                        # with open(ocr_path, "w", encoding="utf-8") as f:
                        #     f.write(output_text)
                        # print(f"Saved OCR output: {ocr_path}")

                        # Save Structured JSON for each page
                        json_path = os.path.join(OUTPUT_DIR, f"{name}_page_{i+1}_structured.json")
                        with open(json_path, "w", encoding="utf-8") as f:
                            f.write(structured_json)
                        print(f"Saved structured JSON: {json_path}")
                    else:
                        print(f"[PDF] No text extracted from page {i+1} of {file_path}")
        else:
            print(f"[PDF] Image-based PDF: {file_path}")
            images = convert_from_path(file_path)
            for i, img in enumerate(images):
                 # Resize image before processing
                max_size = 1024 # Define a maximum size for the longest side
                if max(img.size) > max_size:
                    ratio = max_size / max(img.size)
                    new_size = (int(img.size[0] * ratio), int(img.size[1] * ratio))
                    img = img.resize(new_size, Image.Resampling.LANCZOS)

                enhancer = ImageEnhance.Contrast(img.convert("L"))
                enhanced_img = enhancer.enhance(2.0)
                text = run_qwen_ocr(enhanced_img)
                # Detect report type for image-based PDF pages
                report_type = classify_report_type(text)
                json_result = generate_structured_json(text, report_type=report_type, image_path=f"{file_path}_page_{i+1}", confidence=(

                # Save OCR Text for each page
                # ocr_path = os.path.join(OUTPUT_DIR, f"{name}_page_{i+1}_ocr.txt")
                # with open(ocr_path, "w", encoding="utf-8") as f:
                #     f.write(text)
                # print(f"Saved OCR output: {ocr_path}")

                # Save Structured JSON for each page
                json_path = os.path.join(OUTPUT_DIR, f"{name}_page_{i+1}_structured.json")
                with open(json_path, "w", encoding="utf-8") as f:
                    f.write(json_result)
                print(f"Saved structured JSON: {json_path}")
```

```
                clear_gpu_memory()

        else:
            print(f"Unsupported file format: {file_path}")
            return


# =============== MAIN ===============
input_files = [
    os.path.join(INPUT_DIR, f)
    for f in os.listdir(INPUT_DIR)
    if f.endswith((".pdf", ".jpg", ".png", ".jpeg"))
]

for file in input_files:
    process_input_file(file)
```

Start coding or generate with AI.