```
!pip install pdf2image
!apt-get install poppler-utils
```

```
Collecting pdf2image
  Downloading pdf2image-1.17.0-py3-none-any.whl.metadata (6.2 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-pac
Downloading pdf2image-1.17.0-py3-none-any.whl (11 kB)
Installing collected packages: pdf2image
Successfully installed pdf2image-1.17.0
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  poppler-utils
0 upgraded, 1 newly installed, 0 to remove and 29 not upgraded.
Need to get 186 kB of archives.
After this operation, 696 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 poppler-uti
Fetched 186 kB in 0s (757 kB/s)
Selecting previously unselected package poppler-utils.
(Reading database ... 124947 files and directories currently installed.)
Preparing to unpack .../poppler-utils_22.02.0-2ubuntu0.6_amd64.deb ...
Unpacking poppler-utils (22.02.0-2ubuntu0.6) ...
Setting up poppler-utils (22.02.0-2ubuntu0.6) ...
Processing triggers for man-db (2.10.2-1) ...
```

```
from pdf2image import convert_from_path
import os

# Convert PDF to images
def pdf_to_images(pdf_path, output_folder="/content/extracted_images"): # Chang
    os.makedirs(output_folder, exist_ok=True)
    pages = convert_from_path(pdf_path, dpi=300)  # High-resolution extraction

    image_paths = []
    for i, page in enumerate(pages):
        img_path = os.path.join(output_folder, f"page_{i}.png")
        page.save(img_path, "PNG")
        image_paths.append(img_path)

    print(f"Extracted {len(image_paths)} images from {pdf_path}")
    return image_paths

# Run the function
pdf_path = "/content/Test (1).pdf"
image_paths = pdf_to_images(pdf_path)
```

```
Extracted 33 images from /content/Test (1).pdf
```

Start coding or generate with AI.

```
import cv2
import numpy as np

def preprocess_image(image_path, output_folder="processed_images"):
    os.makedirs(output_folder, exist_ok=True)

    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

    # Apply adaptive thresholding to enhance lines
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)

    # Edge detection using Canny
    edges = cv2.Canny(thresh, 50, 150)

    processed_path = os.path.join(output_folder, os.path.basename(image_path))
    cv2.imwrite(processed_path, edges)  # Save processed image

    return processed_path

# Run preprocessing on extracted images
processed_images = [preprocess_image(img) for img in image_paths]
print(f"Processed {len(processed_images)} images.")
```

```
Processed 33 images.
```

```
def detect_pipelines(image_path, output_folder="pipeline_detected"):
    os.makedirs(output_folder, exist_ok=True)

    img = cv2.imread(image_path)
    edges = cv2.Canny(img, 50, 150, apertureSize=3)

    # Detect lines using Hough Transform
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=100, minLineLength=5

    img_copy = img.copy()

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(img_copy, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Draw pipe

    detected_path = os.path.join(output_folder, os.path.basename(image_path))
    cv2.imwrite(detected_path, img_copy)

    return detected_path

# Run pipeline detection
pipeline_detected_images = [detect_pipelines(img) for img in processed_images]
print(f"Detected pipelines in {len(pipeline_detected_images)} images.")
```

```
Detected pipelines in 33 images.
```

```
import torch
import torchvision
from torchvision import transforms
from torchvision.models.detection import fasterrcnn_resnet50_fpn

# Load pre-trained Faster R-CNN
def load_rcnn_model():
    model = fasterrcnn_resnet50_fpn(pretrained=True)
    model.eval()
    return model

def detect_instruments(image_path, model, output_folder="ic_detected"):
    os.makedirs(output_folder, exist_ok=True)

    img = cv2.imread(image_path)
    transform = transforms.Compose([transforms.ToTensor()])
    img_tensor = transform(img).unsqueeze(0)

    # Perform inference
```

```
    with torch.no_grad():
        prediction = model(img_tensor)

    img_copy = img.copy()
    for i, box in enumerate(prediction[0]['boxes']):
        score = prediction[0]['scores'][i].item()
        if score > 0.5:  # Confidence threshold
            x1, y1, x2, y2 = map(int, box)
            cv2.rectangle(img_copy, (x1, y1), (x2, y2), (0, 0, 255), 2)  # Drav
            cv2.putText(img_copy, "I&C", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLE

    detected_path = os.path.join(output_folder, os.path.basename(image_path))
    cv2.imwrite(detected_path, img_copy)

    return detected_path

# Load model and run I&C detection
rcnn_model = load_rcnn_model()
ic_detected_images = [detect_instruments(img, rcnn_model) for img in pipeline_c
print(f"Detected I&C elements in {len(ic_detected_images)} images.")
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: U
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: U
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_c
100%|██████████| 160M/160M [00:00<00:00, 190MB/s]
Detected I&C elements in 33 images.
```

```
!apt-get install tesseract-ocr
!apt-get install libtesseract-dev
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  tesseract-ocr-eng tesseract-ocr-osd
The following NEW packages will be installed:
  tesseract-ocr tesseract-ocr-eng tesseract-ocr-osd
0 upgraded, 3 newly installed, 0 to remove and 29 not upgraded.
Need to get 4,816 kB of archives.
After this operation, 15.6 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr-e
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr-c
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr a
Fetched 4,816 kB in 0s (15.4 MB/s)
Selecting previously unselected package tesseract-ocr-eng.
(Reading database ... 124977 files and directories currently installed.)
Preparing to unpack .../tesseract-ocr-eng_1%3a4.00~git30-7274cfa-1.1_all.de
```

```
Unpacking tesseract-ocr-eng (1:4.00~git30-7274cfa-1.1) ...
Selecting previously unselected package tesseract-ocr-osd.
Preparing to unpack .../tesseract-ocr-osd_1%3a4.00~git30-7274cfa-1.1_all.de
Unpacking tesseract-ocr-osd (1:4.00~git30-7274cfa-1.1) ...
Selecting previously unselected package tesseract-ocr.
Preparing to unpack .../tesseract-ocr_4.1.1-2.1build1_amd64.deb ...
Unpacking tesseract-ocr (4.1.1-2.1build1) ...
Setting up tesseract-ocr-eng (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr-osd (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr (4.1.1-2.1build1) ...
Processing triggers for man-db (2.10.2-1) ...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libarchive-dev libleptonica-dev
The following NEW packages will be installed:
  libarchive-dev libleptonica-dev libtesseract-dev
0 upgraded, 3 newly installed, 0 to remove and 29 not upgraded.
Need to get 3,743 kB of archives.
After this operation, 16.0 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libarchive-
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libleptonica-de
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libtesseract-de
Fetched 3,743 kB in 0s (12.9 MB/s)
Selecting previously unselected package libarchive-dev:amd64.
(Reading database ... 125024 files and directories currently installed.)
Preparing to unpack .../libarchive-dev_3.6.0-1ubuntu1.3_amd64.deb ...
Unpacking libarchive-dev:amd64 (3.6.0-1ubuntu1.3) ...
Selecting previously unselected package libleptonica-dev.
Preparing to unpack .../libleptonica-dev_1.82.0-3build1_amd64.deb ...
Unpacking libleptonica-dev (1.82.0-3build1) ...
Selecting previously unselected package libtesseract-dev:amd64.
Preparing to unpack .../libtesseract-dev_4.1.1-2.1build1_amd64.deb ...
Unpacking libtesseract-dev:amd64 (4.1.1-2.1build1) ...
Setting up libleptonica-dev (1.82.0-3build1) ...
Setting up libarchive-dev:amd64 (3.6.0-1ubuntu1.3) ...
Setting up libtesseract-dev:amd64 (4.1.1-2.1build1) ...
Processing triggers for man-db (2.10.2-1) ...
```

```
import os
import cv2
import torch
import json
import torchvision
from torchvision import transforms
from torchvision.models.detection import fasterrcnn_resnet50_fpn

# Load pre-trained Faster R-CNN model
def load_rcnn_model():
    model = fasterrcnn_resnet50_fpn(pretrained=True)
```

```
    model.eval()
    return model

# Function to extract pipeline structure from IC detected images
def extract_pipeline_structure(image_path, model, output_folder="pipeline_extra
    os.makedirs(output_folder, exist_ok=True)

    # Load image and convert to tensor
    img = cv2.imread(image_path)
    transform = transforms.Compose([transforms.ToTensor()])
    img_tensor = transform(img).unsqueeze(0)  # Add batch dimension

    # Perform inference
    with torch.no_grad():
        prediction = model(img_tensor)

    # Extract pipeline elements and connections
    pipeline_elements = []
    img_copy = img.copy()

    for i, box in enumerate(prediction[0]['boxes']):
        score = prediction[0]['scores'][i].item()
        if score > 0.5:  # Confidence threshold
            x1, y1, x2, y2 = map(int, box)
            width, height = x2 - x1, y2 - y1

            # Save element info
            element = {
                "id": i,
                "coordinates": {"x": x1, "y": y1, "width": width, "height": hei
                "confidence": round(score, 2)
            }
            pipeline_elements.append(element)

            # Draw bounding box
            cv2.rectangle(img_copy, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # Save the extracted image
    extracted_path = os.path.join(output_folder, os.path.basename(image_path).r
    cv2.imwrite(extracted_path, img_copy)

    return pipeline_elements

# Function to process all images in IC detected folder
def process_ic_detection_folder(ic_detected_folder):
    print("Loading Faster R-CNN model...")
    model = load_rcnn_model()
```

```
    print("Extracting pipeline structure and connections...")
    image_paths = [os.path.join(ic_detected_folder, img) for img in os.listdir(

    all_pipeline_data = {}
    for img_path in image_paths:
        img_name = os.path.basename(img_path)
        all_pipeline_data[img_name] = extract_pipeline_structure(img_path, mode

    # Save pipeline structure as JSON
    json_output = os.path.join("pipeline_extraction", "pipeline_structure.json'
    os.makedirs("pipeline_extraction", exist_ok=True)
    with open(json_output, "w") as json_file:
        json.dump(all_pipeline_data, json_file, indent=4)

    print(f"Pipeline structure extracted. Results saved in 'pipeline_extractior

# Set the IC detected image folder path
ic_detected_folder = "ic_detected"  # Update with your actual folder name
process_ic_detection_folder(ic_detected_folder)
```

```
Loading Faster R-CNN model...
Extracting pipeline structure and connections...
Pipeline structure extracted. Results saved in 'pipeline_extraction' folder
```

```
import os

ic_detected_folder = "ic_detected"

# List all images
ic_images = [os.path.join(ic_detected_folder, img) for img in os.listdir(ic_det

print(f"✅ Found {len(ic_images)} images: {ic_images}")
```

```
✅ Found 33 images: ['ic_detected/page_31.png', 'ic_detected/page_0.png',
```

```
import cv2
import os

def extract_ic_positions(image_path):
    """Extracts I&C element positions using contours (bounding boxes)."""
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    if img is None:
        print(f"❌ Error: Cannot read image {image_path}")
        return []

    # Apply threshold to detect shapes
    _, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    # Find contours of detected objects
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_

    ic_positions = []
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        center_x, center_y = x + w // 2, y + h // 2
        ic_positions.append((center_x, center_y))  # Store center of bounding b

    return ic_positions

# Extract I&C positions from all detected images
ic_detected_folder = "ic_detected"
ic_images = [os.path.join(ic_detected_folder, img) for img in os.listdir(ic_det

ic_all_positions = {img: extract_ic_positions(img) for img in ic_images}

print("✅ Extracted I&C positions for all images:", ic_all_positions)
```

```
✅ Extracted I&C positions for all images: {'ic_detected/page_31.png': [(24
```

```
import cv2
import numpy as np
import json
import os

def preprocess_image(image_path):
    """Load and preprocess image to enhance pipeline detection."""
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    blurred = cv2.GaussianBlur(img, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)
    return edges
```

```
def detect_contours(image_path):
    """Find contours in an image representing pipeline structures."""
    edges = preprocess_image(image_path)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_S
    return contours

def extract_coordinates(contours):
    """Extract geometric coordinates from contours."""
    coordinates = [contour.reshape(-1, 2).tolist() for contour in contours]
    return coordinates

def process_images(image_folder):
    """Process all images in a folder and extract pipeline structures."""
    pipeline_data = {}
    for filename in os.listdir(image_folder):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(image_folder, filename)
            contours = detect_contours(image_path)
            pipeline_data[filename] = extract_coordinates(contours)

    return pipeline_data

# Define image folder path
image_folder = "/content/ic_detected"

# Process images and extract pipeline structure
pipeline_structure = process_images(image_folder)

# Save extracted pipeline structures as JSON
output_json_path = "pipeline_structure.json"
with open(output_json_path, "w") as json_file:
    json.dump(pipeline_structure, json_file, indent=4)

print(f"Pipeline structure saved to {output_json_path}")
```

```
Pipeline structure saved to pipeline_structure.json
```

```
import json

# Load the extracted pipeline structure
json_path = "pipeline_structure.json"

with open(json_path, "r") as file:
    pipeline_data = json.load(file)

# Print a sample to verify correctness
print(json.dumps(pipeline_data, indent=4))
```

```
Buffered data was truncated after reaching the output size limit.
```

```
import cv2
import json
import os
import numpy as np
import matplotlib.pyplot as plt

# Load JSON pipeline data
with open("pipeline_structure.json", "r") as f:
    pipeline_data = json.load(f)

image_folder = "/content/ic_detected"

for image_name, contours in pipeline_data.items():
    image_path = os.path.join(image_folder, image_name)

    img = cv2.imread(image_path)

    if img is None:
        print(f"Error loading {image_name}")
        continue

    for contour in contours:
        contour = np.array(contour, dtype=np.int32)
        cv2.polylines(img, [contour], isClosed=False, color=(0, 0, 255), thickne

    # Convert to RGB for displaying in Matplotlib
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Display image with Matplotlib
    plt.figure(figsize=(8, 6))
    plt.imshow(img_rgb)
    plt.title(f"Pipeline Structure - {image_name}")
    plt.axis("off")
    plt.show()
```

```
import cv2
import json
import os
import numpy as np
from google.colab.patches import cv2_imshow  # Use this in Colab

# Load JSON pipeline data
with open("pipeline_structure.json", "r") as f:
    pipeline_data = json.load(f)

image_folder = "/content/ic_detected"
```

```
for image_name, contours in pipeline_data.items():
    image_path = os.path.join(image_folder, image_name)

    img = cv2.imread(image_path)

    if img is None:
        print(f"Error loading {image_name}")
        continue

    for contour in contours:
        contour = np.array(contour, dtype=np.int32)
        cv2.polylines(img, [contour], isClosed=False, color=(0, 0, 255), thickne

    # Display image in Colab
    cv2_imshow(img)  # ✅ Replaces cv2.imshow()
```

FUEL GAS CONDITIONING UNIT PU-2852
DETAIL "A"
FUEL GAS CONDITIONING UNIT PU-2852

GAS CHROMATOGRAPH
CONTROL UNIT
NOTES

NOTE:
OFF BASE ENCLOSURE
AUXILIARY BASEPLATE
RATCHET
MECHANICAL RATCHET
ROTATION
TORQUE
CONVERTER

OIL FILTER TROLLEY

NOTES
FUEL GAS CONDITIONING UNIT FCU-2862
DETAIL "A"
DETAIL "B"

DELETED

NOTES

NOTES
LIFT PUMP
DRIVEN PUMP
ACC GEAR

OFF BASE ENCLOSURE - TURBINE COMPARTMENT
PRIMARY GAS CONTROL VALVE
SECONDARY GAS CONTROL VALVE
PRIMARY MANIFOLD
SECONDARY MANIFOLD
COMBUSTION CHAMBER
TURBINE BASEPLATE
OFF BASE ENCLOSURE - TURBINE COMPARTMENT

NOTES

DETAIL "A"
DETAIL "B"

CONSTRUCTION SERVICES LIST
NOTES

NOTES
CORE PANEL
OFF BASE ENCLOSURE
AUXILIARY BASEPLATE
HPC
ACTUATOR ASSEMBLY

FILTER HOUSE
NOTES
AIR INLET
AIR INLET
AIR INLET
AIR INLET
AIR INLET
VENTILATION AIR
SILENCER
VENTILATION AIR
MCC

OFF BASE ENCLOSURE
AUXILIARY BASEPLATE PU12927
TORQUE CONVERTER
ACCESSORY GEAR
SAFETY SYSTEM
SELF STORAGE AND STARTING CAPTOR
MAIN ENGINE
NOTES:

NOTES
HX1N-1
HX1N-2
LUBE OIL/WATER HEAT EXCHANGER SKID
LO-ASS.
PA-ASS.

NOTES
JUBE ENGINE
G.T BASEPLATE
OFF BASE ENCLOSURE
TURBINE COMPARTMENT

FUEL GAS AUTOMATIC VENT VALVE TYPICAL
FUEL GAS AUTOMATIC SHUT-OFF VALVE TYPICAL
BLOWDOWN VALVE BDV TYPICAL TYPICAL
NOTES

FUEL GAS CONDITIONING UNIT PU-2054
NOTES
DETAIL "A"
DETAIL "B"

EXHAUST PLENUM
TURBINE ZONE
COUPLING ZONE
LOAD GEAR COMPARTMENT
ACCESSORY ZONE

VALVES
GLOBE VALVE
BALL VALVE
CLOCK VALVE
GATE VALVE
BUTTERFLY VALVE
SAFETY VALVE
AIR RELIEF VALVE
SPECIALTY ITEMS:
V-TYPE STRAINER
STEAM TRAP
VENT STACK
FLUID
MISCELLANEOUS EQUIPMENT
TANK ATMOSPHERIC
FUEL GAS HEATER
FUEL GAS SEPARATOR
MODEL INSULATION
DRAIN CONNECTORS
CONNECTORS
GENERAL MATERIALS
INSTRUMENT IDENTIFICATION LETTERS
FIRST LETTER
SUCCEEDING LETTERS
NOTES

CONTROL VALVE ACTUATOR
PIPING FITTINGS
MISCELLANEOUS
LIMIT OF SUPPLY
INSTR. LINES SYMBOL
CONTROL LOGIC LEGEND
FLOW CHARACTERISTICS
VALVE DESCRIPTION
FC FAIL-CLOSED
LO LOCKED OPEN
LC LOCKED CLOSED
FO FAIL-OPEN
PR PLUG RAISE

PU-2936
FIRE FIGHTING SYSTEM
(CO2 BOTTLES 2X100)
BANK "A"
(MAIN)

BANK "B"
(RESERVE)

SAFETY BLOCK
MC

AUX BASEPLATE PU-2927
OFF BASE ENCLOSURE
TURBINE BASEPLATE PU-2931

OFF BASE ENCLOSURE - TURBINE COMPARTMENT
NOTES

ON/OFF-LINE WASHING WATER TROLLEY

```
import json
import fitz  # PyMuPDF
import cv2
import numpy as np
from PIL import Image
import io

# Load extracted pipeline structure
with open("pipeline_structure.json", "r") as f:
    pipeline_data = json.load(f)

# Load the PDF
pdf_path = "Test (1).pdf"
doc = fitz.open(pdf_path)

# Process each page
for page_num in range(len(doc)):
    page = doc[page_num]
    pix = page.get_pixmap()

    img = np.frombuffer(pix.samples, dtype=np.uint8).reshape(pix.h, pix.w, pix.
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    # Draw pipelines on the PDF image
    for image_name, contours in pipeline_data.items():
        for contour in contours:
            contour = np.array(contour, dtype=np.int32)
            cv2.polylines(img, [contour], isClosed=False, color=(255, 0, 0), th
```

```
    # Convert OpenCV image to PIL Image
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(img_rgb)

    # Save PIL image to a bytes buffer
    img_byte_arr = io.BytesIO()
    pil_img.save(img_byte_arr, format="PNG")
    img_byte_arr = img_byte_arr.getvalue()  # Get bytes

    # Insert image into PDF
    rect = page.rect  # Full-page insertion
    page.insert_image(rect, stream=img_byte_arr)

# Save the updated PDF
annotated_pdf_path = "Annotated_Pipeline_Structure.pdf"
doc.save(annotated_pdf_path)
doc.close()

print(f"✅ Annotated PDF saved as {annotated_pdf_path}")
```

```
Collecting pymupdf
  Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_
Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x8
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.0/20.0 MB 52.5 MB/s eta 0:00
Installing collected packages: pymupdf
Successfully installed pymupdf-1.25.3
✅ Annotated PDF saved as Annotated_Pipeline_Structure.pdf
```

```
import cv2
import json
import os
import numpy as np
import matplotlib.pyplot as plt

# Load JSON pipeline data
with open("pipeline_structure.json", "r") as f:
    pipeline_data = json.load(f)

image_folder = "/content/ic_detected"
output_folder = "/content/pipeline_output"

# Create output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

for image_name, contours in pipeline_data.items():
    image_path = os.path.join(image_folder, image_name)
```

```
    img = cv2.imread(image_path)

    if img is None:
        print(f"Error loading {image_name}")
        continue

    for contour in contours:
        contour = np.array(contour, dtype=np.int32)
        cv2.polylines(img, [contour], isClosed=False, color=(0, 0, 255), thickr

    # Convert to RGB for displaying in Matplotlib
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Save the output image
    output_path = os.path.join(output_folder, image_name)
    cv2.imwrite(output_path, cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))

    # Display image with Matplotlib (Optional)
    plt.figure(figsize=(8, 6))
    plt.imshow(img_rgb)
    plt.title(f"Pipeline Structure - {image_name}")
    plt.axis("off")
    plt.show()

print(f"✅ All output images are saved in: {output_folder}")
```

Pipeline Structure - page_31.png

Pipeline Structure - page_0.png

```
pip install opencv-python numpy pytesseract pillow
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-pack
Collecting pytesseract
  Downloading pytesseract-0.3.13-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11
Downloading pytesseract-0.3.13-py3-none-any.whl (14 kB)
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.13
```

```
!pip install opencv-python numpy matplotlib pytesseract
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist
Requirement already satisfied: pytesseract in /usr/local/lib/python3.11/dis
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-p
```

Pipeline Structure-page_14.png

```
from zipfile import ZipFile

# Define the path where your ZIP file is uploaded
zip_path = "/content/OCRimages.zip"  # Change this to your file's path

# Extract the ZIP file
with ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall("/content/extracted_files")  # Change destination if nee

print("Extraction complete! Files are in /content/extracted_files")
```

```
Extraction complete! Files are in /content/extracted_files
```

```
from zipfile import ZipFile

# Define the path where your ZIP file is uploaded
zip_path = "/content/OCR_text_files.zip"  # Change this to your file's path

# Extract the ZIP file
with ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall("/content/extracted1_files")  # Change destination if ne

print("Extraction complete! Files are in /content/extracted1_files")
```

```
Extraction complete! Files are in /content/extracted1_files
```

```
import os
import cv2
import numpy as np
```

```
import json
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Folder paths (update these based on your directory structure)
IC_DETECTED_FOLDER = "/content/ic_detected"
PIPELINE_DETECTION_FOLDER = "/content/pipeline_detected"
OCR_IMAGES_FOLDER = "/content/extracted_files"
OCR_TEXT_FOLDER = "/content/extracted1_files"

# Function to load OCR text data
def load_ocr_texts(text_folder):
    text_data = {}
    for file in os.listdir(text_folder):
        if file.endswith(".txt"):
            with open(os.path.join(text_folder, file), "r") as f:
                text_data[file] = f.read().strip()
    return text_data

ocr_texts = load_ocr_texts(OCR_TEXT_FOLDER)

# Function to load and preprocess images
def load_images(image_folder, text_data):
    images, labels = [], []
    for img_file in os.listdir(image_folder):
        if img_file.endswith((".png", ".jpg", ".jpeg")):
            img_path = os.path.join(image_folder, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (128, 128)) / 255.0  # Normalize
            images.append(img)
            labels.append(text_data.get(img_file.replace(".png", ".txt"), "UNKN
    return np.array(images), labels

# Load images from IC detected and pipeline detection folders
X_ic, y_ic = load_images(IC_DETECTED_FOLDER, ocr_texts)
X_pipeline, y_pipeline = load_images(PIPELINE_DETECTION_FOLDER, ocr_texts)

# Combine datasets
X = np.concatenate((X_ic, X_pipeline), axis=0)
y = y_ic + y_pipeline  # Combine labels

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
y_encoded = to_categorical(y_encoded)  # One-hot encoding for classification

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer fo
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save model
model.save("pipeline_ic_classification_model.h5")

# Function to predict label for a new image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example usage (Replace with actual image path)
test_img = os.path.join(PIPELINE_DETECTION_FOLDER, "/content/pipeline_detected/
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

Pipeline Structure - page_4.png

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarnin
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/losses/losses.py:33: Synt
  return self.fn(y_true, y_pred, **self._fn_kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 810ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 572ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 571ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 529ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 672ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 581ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 535ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 633ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 576ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 821ms/step - accuracy: 1.0000 - loss: 0.0000e+0
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
Predicted Label: UNKNOWN
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarnin
  warnings.warn(
```

Pipeline Structure - page_16.png

```
import os
import cv2
import numpy as np
import json
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Folder paths (update these based on your directory structure)
OCR_IMAGES_FOLDER = "/content/extracted_images"
OCR_TEXT_FOLDER = "/content/extracted1_files"
ORIGINAL_DATASET_FOLDER = "/content/Test (1).pdf"

# Function to load OCR text data
def load_ocr_texts(text_folder):
```

```
    text_data = {}
    for file in os.listdir(text_folder):
        if file.endswith(".txt"):
            with open(os.path.join(text_folder, file), "r") as f:
                text_data[file] = f.read().strip()
    return text_data

ocr_texts = load_ocr_texts(OCR_TEXT_FOLDER)

# Function to load and preprocess images
def load_images(image_folder, text_data):
    images, labels = [], []
    for img_file in os.listdir(image_folder):
        if img_file.endswith((".png", ".jpg", ".jpeg")):
            img_path = os.path.join(image_folder, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (128, 128)) / 255.0  # Normalize
            images.append(img)
            labels.append(text_data.get(img_file.replace(".png", ".txt"), "UNKN
    return np.array(images), labels

# Load images from OCR image folder
X_ocr, y_ocr = load_images(OCR_IMAGES_FOLDER, ocr_texts)

# Load images from the original dataset
X_original, y_original = load_images(ORIGINAL_DATASET_FOLDER, ocr_texts)

# Combine datasets
X = np.concatenate((X_ocr, X_original), axis=0)
y = y_ocr + y_original  # Combine labels

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded = to_categorical(y_encoded)  # One-hot encoding for classification

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
```

```
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer fo
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save model
model.save("ocr_pipeline_classification_model.h5")

# Function to predict label for a new image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example usage (Replace with actual image path)
test_img = os.path.join(OCR_IMAGES_FOLDER, "/content/extracted_files/page_10_ar
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
---------------------------------------------------------------------------
NotADirectoryError                        Traceback (most recent call last)
<ipython-input-24-ec5f0c042e11> in <cell line: 0>()
     42
     43 # Load images from the original dataset
---> 44 X_original, y_original = load_images(ORIGINAL_DATASET_FOLDER, ocr_texts)
     45
     46 # Combine datasets

<ipython-input-24-ec5f0c042e11> in load_images(image_folder, text_data)
     29 def load_images(image_folder, text_data):
     30     images, labels = [], []
---> 31     for img_file in os.listdir(image_folder):
     32         if img_file.endswith((".png", ".jpg", ".jpeg")):
     33             img_path = os.path.join(image_folder, img_file)
```

```
import os
import cv2
import numpy as np
```

```
import json
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from pdf2image import convert_from_path # import to convert pdf to images

# Folder paths (update these based on your directory structure)
OCR_IMAGES_FOLDER = "/content/extracted_images"
OCR_TEXT_FOLDER = "/content/extracted1_files"
ORIGINAL_DATASET_FOLDER = "/content/Test (1).pdf"

# Function to load OCR text data
def load_ocr_texts(text_folder):
    text_data = {}
    for file in os.listdir(text_folder):
        if file.endswith(".txt"):
            with open(os.path.join(text_folder, file), "r") as f:
                text_data[file] = f.read().strip()
    return text_data

ocr_texts = load_ocr_texts(OCR_TEXT_FOLDER)

# Function to load and preprocess images
def load_images(image_folder, text_data):
    images, labels = [], []
    # Check if it's a PDF file and convert to images if necessary
    if image_folder.endswith(".pdf"):
        # Convert PDF to images and get paths to images
        pages = convert_from_path(image_folder, dpi=300)
        image_paths = [os.path.join('/content/pdf_images', f'page_{i}.png') for
        for i, page in enumerate(pages):
            os.makedirs(os.path.dirname(image_paths[i]), exist_ok=True)
            page.save(image_paths[i], "PNG")
    else:
        # If it's not a PDF, assume it's a directory
        image_paths = [os.path.join(image_folder, img_file) for img_file in os.
                       if img_file.endswith((".png", ".jpg", ".jpeg"))]

    # Load images and get labels
    for img_path in image_paths:
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (128, 128)) / 255.0  # Normalize
        images.append(img)
        labels.append(text_data.get(os.path.basename(img_path).replace(".png",
```

```
    return np.array(images), labels

# Load images from OCR image folder
X_ocr, y_ocr = load_images(OCR_IMAGES_FOLDER, ocr_texts)

# Load images from the original dataset (which is a PDF)
X_original, y_original = load_images(ORIGINAL_DATASET_FOLDER, ocr_texts)

# Combine datasets
X = np.concatenate((X_ocr, X_original), axis=0)
y = y_ocr + y_original  # Combine labels

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded = to_categorical(y_encoded)  # One-hot encoding for classification

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer fo
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save model
model.save("ocr_pipeline_classification_model.h5")

# Function to predict label for a new image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
```

```
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example usage (Replace with actual image path)
test_img = os.path.join(OCR_IMAGES_FOLDER, "/content/extracted_files/page_10_ar
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarnin
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/losses/losses.py:33: Synt
  return self.fn(y_true, y_pred, **self._fn_kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 643ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 382ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 458ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 403ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 551ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 376ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 375ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 1.0000 - loss: 0.0000e+0
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 375ms/step - accuracy: 1.0000 - loss: 0.0000e+0
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
Predicted Label: UNKNOWN
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarnin
  warnings.warn(
```

Pipeline Structure - page_21.png

```
import os
import cv2
import numpy as np
import json
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
```

```
# Folder paths (update these based on your directory structure)
OCR_IMAGES_FOLDER = "/content/extracted_files"
OCR_TEXT_FOLDER = "/content/extracted1_files"
ORIGINAL_DATASET_FOLDER = "/content/pdf_images"

# Function to load OCR text data
def load_ocr_texts(text_folder):
    text_data = {}
    for file in os.listdir(text_folder):
        if file.endswith(".txt"):
            with open(os.path.join(text_folder, file), "r") as f:
                text_data[file] = f.read().strip()
    return text_data

ocr_texts = load_ocr_texts(OCR_TEXT_FOLDER)

# Function to load and preprocess images
def load_images(image_folder, text_data):
    images, labels = [], []
    for img_file in os.listdir(image_folder):
        if img_file.endswith((".png", ".jpg", ".jpeg")):
            img_path = os.path.join(image_folder, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (128, 128)) / 255.0  # Normalize
            images.append(img)
            labels.append(text_data.get(img_file.replace(".png", ".txt"), "UNKN
    return np.array(images), labels

# Load images from OCR image folder
X_ocr, y_ocr = load_images(OCR_IMAGES_FOLDER, ocr_texts)

# Load images from the original dataset
X_original, y_original = load_images(ORIGINAL_DATASET_FOLDER, ocr_texts)

# Combine datasets
X = np.concatenate((X_ocr, X_original), axis=0)
y = y_ocr + y_original  # Combine labels

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded = to_categorical(y_encoded)  # One-hot encoding for classification

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# Split dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# Define CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer fo
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save model
model.save("ocr_pipeline_classification_model.h5")

# Function to predict label for a new image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example usage (Replace with actual image path)
test_img = os.path.join(OCR_IMAGES_FOLDER, "/content/pdf_images/page_11.png")
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 713ms/step - accuracy: 0.3401 - loss: 3.8044 -
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 995ms/step - accuracy: 0.5796 - loss: 1.9511 -
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 650ms/step - accuracy: 0.6413 - loss: 1.7909 -
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 669ms/step - accuracy: 0.6331 - loss: 1.7556 -
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 667ms/step - accuracy: 0.6589 - loss: 1.5995 -
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 667ms/step - accuracy: 0.6627 - loss: 1.5745 -
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 953ms/step - accuracy: 0.6820 - loss: 1.4911 -
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 666ms/step - accuracy: 0.6851 - loss: 1.3577 -
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 670ms/step - accuracy: 0.7127 - loss: 1.2227 -
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 858ms/step - accuracy: 0.7145 - loss: 1.1131 -
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
Predicted Label: UNKNOWN
```

Pipeline Structure - page_5.png

```
!pip install PyMuPDF
import fitz  # PyMuPDF for handling PDFs
import os
import cv2
import numpy as np
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Paths
IC_DETECTED_FOLDER = "/content/ic_detected"
OCR_IMAGES_FOLDER = "/content/extracted_files"
OCR_TEXT_FOLDER = "/content/extracted1_files"
ORIGINAL_DATASET_PATH = "/content/Test (1).pdf"

# 🔷 Function to Extract Text from OCR Text Files
def load_ocr_texts(text_folder):
    text_data = {}
    for file in os.listdir(text_folder):
        if file.endswith(".txt"):
```

```
            with open(os.path.join(text_folder, file), "r") as f:
                text_data[file] = f.read().strip()
    return text_data

ocr_texts = load_ocr_texts(OCR_TEXT_FOLDER)

# 🔷 Function to Extract Images from PDF using PyMuPDF
def extract_images_from_pdf(pdf_path, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    doc = fitz.open(pdf_path)  # Open the PDF
    image_paths = []

    for i, page in enumerate(doc):
        pix = page.get_pixmap()  # Render page as an image
        img_path = os.path.join(output_folder, f"page_{i}.png")

        # Convert to OpenCV format
        img = np.frombuffer(pix.samples, dtype=np.uint8).reshape(pix.h, pix.w,
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        cv2.imwrite(img_path, img)  # Save as PNG

        image_paths.append(img_path)

    return image_paths

# 🔷 Extract Images from the Original Dataset (PDF)
original_image_paths = extract_images_from_pdf(ORIGINAL_DATASET_PATH, "/content

# 🔷 Function to Load Images & Match with OCR Text
def load_images(image_folder, text_data):
    images, labels = [], []

    image_paths = [os.path.join(image_folder, img) for img in os.listdir(image_
                   if img.endswith((".png", ".jpg", ".jpeg"))]

    for img_path in image_paths:
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (128, 128))
        img = (img / 255.0).astype(np.float32)  # Normalize

        images.append(img)
        labels.append(text_data.get(os.path.basename(img_path).replace(".png",

    return np.array(images), labels

# Load images from IC detected and OCR dataset
X_ic, y_ic = load_images(IC_DETECTED_FOLDER, ocr_texts)
X_ocr, y_ocr = load_images(OCR_IMAGES_FOLDER, ocr_texts)
```

```
# Load images from extracted original dataset images
X_original, y_original = load_images("/content/original_dataset_images", ocr_te

# 🔷 Combine all datasets
X = np.concatenate((X_ic, X_ocr, X_original), axis=0)
y = y_ic + y_ocr + y_original

# 🔷 Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded = to_categorical(y_encoded)  # Convert to one-hot encoding

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# 🔷 Split Dataset into Training & Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# 🔷 Define CNN Model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer fo
])

# Compile Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train Model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save Model
model.save("ocr_pipeline_classification_model.h5")

# 🔷 Function to Predict Label for a New Image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example Usage (Replace with actual image path)
```

```
test_img = os.path.join(OCR_IMAGES_FOLDER, "/content/pdf_images/page_11.png")
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
Requirement already satisfied: PyMuPDF in /usr/local/lib/python3.11/dist-pa
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 7s 742ms/step - accuracy: 0.4347 - loss: 3.3681 -
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 668ms/step - accuracy: 0.6939 - loss: 1.8545 -
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 880ms/step - accuracy: 0.6908 - loss: 1.7485 -
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 695ms/step - accuracy: 0.6645 - loss: 1.7723 -
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 698ms/step - accuracy: 0.6857 - loss: 1.4886 -
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 841ms/step - accuracy: 0.6798 - loss: 1.5066 -
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 623ms/step - accuracy: 0.6631 - loss: 1.5732 -
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 726ms/step - accuracy: 0.7392 - loss: 1.2274 -
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 645ms/step - accuracy: 0.6999 - loss: 1.3442 -
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 692ms/step - accuracy: 0.7344 - loss: 1.1675 -
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
Predicted Label: UNKNOWN
```

Pipeline Structure_page_13.png

```
import os
import cv2
import numpy as np
import fitz  # PyMuPDF for handling PDFs
import pytesseract
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# 🔷 Define Paths
IC_DETECTED_FOLDER = "/content/ic_detected"
OCR_IMAGES_FOLDER = "/content/extracted_files"
ORIGINAL_DATASET_PATH = "/content/Test (1).pdf"
EXTRACTED_IMAGES_FOLDER = "/content/pdf_images"

# 🔷 Step 1: Extract Images from PDF
```

```
def extract_images_from_pdf(pdf_path, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    doc = fitz.open(pdf_path)
    image_paths = []

    for i, page in enumerate(doc):
        pix = page.get_pixmap()  # Render page as an image
        img_path = os.path.join(output_folder, f"page_{i}.png")

        img = np.frombuffer(pix.samples, dtype=np.uint8).reshape(pix.h, pix.w,
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        cv2.imwrite(img_path, img)  # Save image

        image_paths.append(img_path)

    return image_paths

# Extract images from original dataset (PDF)
original_image_paths = extract_images_from_pdf(ORIGINAL_DATASET_PATH, EXTRACTED

# 🔷 Step 2: Extract Text Using OCR (Instead of using text files)
def extract_text_from_images(image_paths):
    extracted_texts = {}
    for img_path in image_paths:
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1
        text = pytesseract.image_to_string(img).strip()
        extracted_texts[os.path.basename(img_path)] = text if text else "UNKNOW
    return extracted_texts

# Extract text from original images
ocr_texts = extract_text_from_images(original_image_paths)

# 🔷 Step 3: Load & Preprocess Images
def load_images(image_folder, text_data):
    images, labels = [], []

    image_paths = [os.path.join(image_folder, img) for img in os.listdir(image_
                   if img.endswith((".png", ".jpg", ".jpeg"))]

    for img_path in image_paths:
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (128, 128)) / 255.0  # Normalize

        images.append(img)
        labels.append(text_data.get(os.path.basename(img_path), "UNKNOWN"))  #

    return np.array(images), labels
```

```
# Load images from IC detected and OCR dataset
X_ic, y_ic = load_images(IC_DETECTED_FOLDER, ocr_texts)
X_ocr, y_ocr = load_images(OCR_IMAGES_FOLDER, ocr_texts)

# Load images from extracted original dataset images
X_original, y_original = load_images(EXTRACTED_IMAGES_FOLDER, ocr_texts)

# 🔷 Step 4: Combine Datasets
X = np.concatenate((X_ic, X_ocr, X_original), axis=0)
y = y_ic + y_ocr + y_original

# Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_encoded = to_categorical(y_encoded)  # One-hot encoding

# Reshape images for CNN input
X = X.reshape(-1, 128, 128, 1)

# Split Dataset into Training & Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2

# 🔷 Step 5: Define CNN Model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(label_encoder.classes_), activation='softmax')  # Output layer
])

# Compile Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train Model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Save Model
model.save("ocr_pipeline_classification_model.h5")

# 🔷 Step 6: Predict Component Labels for New Image
def predict_label(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128)) / 255.0
    img = img.reshape(1, 128, 128, 1)
```

```
    prediction = model.predict(img)
    return label_encoder.inverse_transform([np.argmax(prediction)])[0]

# Example Usage
test_img = os.path.join(EXTRACTED_IMAGES_FOLDER, "page_10.png")  # Change to ac
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 707ms/step - accuracy: 0.6982 - loss: 0.6618 -
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 882ms/step - accuracy: 0.9786 - loss: 0.1658 -
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 663ms/step - accuracy: 0.9832 - loss: 0.1569 -
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 654ms/step - accuracy: 0.9734 - loss: 0.1588 -
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 882ms/step - accuracy: 0.9617 - loss: 0.1939 -
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 666ms/step - accuracy: 0.9591 - loss: 0.1372 -
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 869ms/step - accuracy: 0.9721 - loss: 0.0856 -
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 725ms/step - accuracy: 0.9911 - loss: 0.0470 -
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 661ms/step - accuracy: 0.9770 - loss: 0.0770 -
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 886ms/step - accuracy: 0.9865 - loss: 0.0709 -
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` c
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step
Predicted Label: UNKNOWN
```

```
import os
import cv2
import numpy as np
import fitz  # PyMuPDF for handling PDFs
import pytesseract

# 🔷 Define Paths
IC_DETECTED_FOLDER = "/content/ic_detected"
ORIGINAL_DATASET_PATH = "/content/Test (1).pdf"
EXTRACTED_IMAGES_FOLDER = "/content/pdf_images"

# 🔷 Step 1: Extract Images from PDF (Original Dataset)
def extract_images_from_pdf(pdf_path, output_folder):
    os.makedirs(output_folder, exist_ok=True)
```

```
    doc = fitz.open(pdf_path)
    image_paths = []

    for i, page in enumerate(doc):
        pix = page.get_pixmap()  # Render page as an image
        img_path = os.path.join(output_folder, f"page_{i}.png")

        img = np.frombuffer(pix.samples, dtype=np.uint8).reshape(pix.h, pix.w,
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        cv2.imwrite(img_path, img)  # Save image

        image_paths.append(img_path)

    return image_paths

# Extract images from original dataset (PDF)
original_image_paths = extract_images_from_pdf(ORIGINAL_DATASET_PATH, EXTRACTED

# 🔷 Step 2: Detect IC Connections Using OCR
def detect_ic_connections(image_paths):
    ic_connections = {}

    for img_path in image_paths:
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1

        # Extract text
        text = pytesseract.image_to_string(img).strip()

        # Extract IC-related text (basic filter)
        ic_labels = [line for line in text.split("\n") if "IC" in line.upper()]
        ic_connections[os.path.basename(img_path)] = ic_labels

    return ic_connections

# Extract IC connections from detected IC images
detected_ic_connections = detect_ic_connections([os.path.join(IC_DETECTED_FOLDE
                                                 for f in os.listdir(IC_DETECTE

# Extract IC connections from original dataset images
original_ic_connections = detect_ic_connections(original_image_paths)

# 🔷 Step 3: Validate IC Connections
def validate_ic_connections(original, detected):
    mismatches = {}

    for image_name in original:
        orig_labels = set(original[image_name])
```

```
        detected_labels = set(detected.get(image_name, []))

        # Find missing or extra connections
        missing = orig_labels - detected_labels
        extra = detected_labels - orig_labels

        if missing or extra:
            mismatches[image_name] = {"missing": list(missing), "extra": list(e

    return mismatches

# Perform validation
connection_mismatches = validate_ic_connections(original_ic_connections, detect

# 🔷 Step 4: Print Validation Report
for image, issues in connection_mismatches.items():
    print(f"\n🔍 **Validation Report for {image}**:")

    if issues["missing"]:
        print(f"❌ Missing IC Connections: {issues['missing']}")
    if issues["extra"]:
        print(f"⚠️ Extra IC Connections Detected: {issues['extra']}")
    if not issues["missing"] and not issues["extra"]:
        print("✅ All IC connections are correctly detected.")
```

```
🔍 **Validation Report for page_1.png**:
⚠️ Extra IC Connections Detected: ['ON ic Rae']

🔍 **Validation Report for page_2.png**:
⚠️ Extra IC Connections Detected: ['Piescicmae ie']

🔍 **Validation Report for page_3.png**:
⚠️ Extra IC Connections Detected: ['ERAT icra', 'mficns']

🔍 **Validation Report for page_10.png**:
⚠️ Extra IC Connections Detected: ['Castaic']

🔍 **Validation Report for page_12.png**:
⚠️ Extra IC Connections Detected: ['Sicha tinny']

🔍 **Validation Report for page_17.png**:
⚠️ Extra IC Connections Detected: ['Me ee i Pica', 'Beet needed vt aicatipt
```

```
import fitz  # PyMuPDF for extracting images from PDF
import pytesseract
import re
```

```
import cv2
import numpy as np
from fuzzywuzzy import process

# Load valid pipeline labels (can be expanded or loaded from a database)
VALID_LABELS = ["FL-3", "VH-2", "PG-4", "FG-2", "P-101", "L-302", "T-1001", "V-

# Define regex pattern for valid pipeline labels (e.g., "P-101", "VH-2")
PIPELINE_REGEX = re.compile(r'[A-Z]+-\d+')

def extract_text_from_image(image_path):
    """Extracts text from an image using OCR after preprocessing."""
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_
    text = pytesseract.image_to_string(processed_img, config="--psm 6")
    return text.split("\n")

def correct_labels(extracted_lines):
    """Corrects misinterpreted pipeline labels using regex and fuzzy matching.'
    corrected_labels = []

    for line in extracted_lines:
        match = PIPELINE_REGEX.search(line)
        if match:
            label = match.group()
        else:
            # Apply fuzzy matching to find the closest valid label
            label, score = process.extractOne(line.strip(), VALID_LABELS) if li
            if score > 80:  # Set a confidence threshold for fuzzy matching
                label = label
            else:
                label = None  # Ignore if confidence is too low

        if label:
            corrected_labels.append(label)

    return corrected_labels

# 📌 **Run the pipeline**
image_path = ""  # Path to the uploaded image
extracted_text = extract_text_from_image(image_path)
corrected_labels = correct_labels(extracted_text)

# ✅ **Output Results**
print("Corrected Pipeline Labels:")
for label in corrected_labels:
    print(label)
```

```
import cv2
import pytesseract
import re
import numpy as np
from fuzzywuzzy import process

# Define valid pipeline labels (expandable)
VALID_LABELS = ["L-302", "PG-4", "T-1001", "V-205", "H-300", "C-450", "X-500",

# Regex pattern for pipeline labels (e.g., "PG-4", "T-1001")
PIPELINE_REGEX = re.compile(r'[A-Z]+-\d+')

def extract_text_from_image(image_path):
    """Extracts text from an image using OCR after preprocessing."""
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_
    text = pytesseract.image_to_string(processed_img, config="--psm 6")
    return text.split("\n")

def correct_labels(extracted_lines):
    """Corrects misinterpreted pipeline labels using regex and fuzzy matching.'
    corrected_labels = []

    for line in extracted_lines:
        match = PIPELINE_REGEX.search(line)
        if match:
            label = match.group()
        else:
            # Apply fuzzy matching to find the closest valid label
            label, score = process.extractOne(line.strip(), VALID_LABELS) if li
            if score > 80:  # Confidence threshold for fuzzy matching
                label = label
            else:
                label = None  # Ignore if confidence is too low

        if label:
            corrected_labels.append(label)

    return corrected_labels

# 📌 **Run the pipeline**
image_path = "/content/extracted_images/page_6.png"  # Use the uploaded image
extracted_text = extract_text_from_image(image_path)
corrected_labels = correct_labels(extracted_text)
```

```
# ✅ **Output Results**
print("Corrected Pipeline Labels:")
for label in corrected_labels:
    print(label)
```

```
Corrected Pipeline Labels:
E-2852
PU-2852
L-302
H-300
R-120
T-1001
FIT-237
PU-2852
```

```
!pip install pytesseract opencv-python fuzzywuzzy python-Levenshtein
```

```
Collecting pytesseract
  Downloading pytesseract-0.3.13-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/d
Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Collecting python-Levenshtein
  Downloading python_levenshtein-0.27.1-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/d
Collecting Levenshtein==0.27.1 (from python-Levenshtein)
  Downloading levenshtein-0.27.1-cp311-cp311-manylinux_2_17_x86_64.manylinu
Collecting rapidfuzz<4.0.0,>=3.9.0 (from Levenshtein==0.27.1->python-Levens
  Downloading rapidfuzz-3.12.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2
Downloading pytesseract-0.3.13-py3-none-any.whl (14 kB)
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Downloading python_levenshtein-0.27.1-py3-none-any.whl (9.4 kB)
Downloading levenshtein-0.27.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 161.7/161.7 kB 4.9 MB/s eta 0:0
Downloading rapidfuzz-3.12.2-cp311-cp311-manylinux_2_17_x86_64.manylinux201
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 25.2 MB/s eta 0:00:0
Installing collected packages: fuzzywuzzy, rapidfuzz, pytesseract, Levensht
Successfully installed Levenshtein-0.27.1 fuzzywuzzy-0.18.0 pytesseract-0.3
```