```
!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-packages (from pdf2image) (11.1.0)
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-utils amd64 22.02.0-2ubuntu0.6 [186 kB]
Fetched 186 kB in 1s (150 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"): # Changed to an absolute path within Colab
    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=50, maxLineGap=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 pipelines

    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)  # Draw red boxes for I&C
            cv2.putText(img_copy, "I&C", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    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_detected_images]
print(f"Detected I&C elements in {len(ic_detected_images)} images.")
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth" to /root/.cache/torch/hub/checkpoints/f
100%|██████████| 160M/160M [00:01<00:00, 141MB/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-eng all 1:4.00~git30-7274cfa-1.1 [1,591 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr-osd all 1:4.00~git30-7274cfa-1.1 [2,990 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tesseract-ocr amd64 4.1.1-2.1build1 [236 kB]
Fetched 4,816 kB in 3s (1,903 kB/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.deb ...
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.deb ...
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-dev amd64 3.6.0-1ubuntu1.3 [581 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libleptonica-dev amd64 1.82.0-3build1 [1,562 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libtesseract-dev amd64 4.1.1-2.1build1 [1,600 kB]
Fetched 3,743 kB in 2s (1,645 kB/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_extraction"):
    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": height},
                "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).replace(".png", "_extracted.png"))
    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(ic_detected_folder) if img.endswith(".png")]

    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, model)

    # 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_extraction' folder.")

# 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_detected_folder) if img.endswith(('.png', '.jpg', '.jpeg'))]

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

```
✅ Found 33 images: ['ic_detected/page_31.png', 'ic_detected/page_0.png', 'ic_detected/page_7.png', 'ic_detected/page_14.png', 'ic_
```

```
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_SIMPLE)

    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 box

    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_detected_folder) if img.endswith(".png")]

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': [(2408, 2607), (2196, 2607), (2173, 2607), (2105, 2607), (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_SIMPLE)
    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), thickness=2)

    # 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), thickness=2)

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

```
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.n)
    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), thickness=2)

    # 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}")
```

```
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), thickness=2)

    # 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}")
```

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

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
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-packages (11.1.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from pytesseract) (24.2)
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/dist-packages (4.11.0.86)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: pytesseract in /usr/local/lib/python3.11/dist-packages (0.3.13)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

```
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 needed

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 needed

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"), "UNKNOWN"))  # Get label from text file
    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, random_state=42)

# 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 for classification
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 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/page_11.png")  # Change to actual test image
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_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarning: You are using a softmax over axis -1 of a tensor of sh
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/losses/losses.py:33: SyntaxWarning: In loss categorical_crossentropy, expected y_p
  return self.fn(y_true, y_pred, **self._fn_kwargs)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 810ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 572ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 571ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 529ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 672ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 581ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 535ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 633ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 576ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 821ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
Predicted Label: UNKNOWN
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarning: You are using a softmax over axis -1 of a tensor of sh
  warnings.warn(
```

```
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"), "UNKNOWN"))  # Get label from text file
    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, random_state=42)

# 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 for classification
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 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_annotated.png")  # Change to actual test image
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)

NotADirectoryError: [Errno 20] Not a directory: '/content/Test (1).pdf'
```

```
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 i in range(len(pages))]
        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.listdir(image_folder)
                       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", ".txt"), "UNKNOWN"))

    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, random_state=42)

# 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 for classification
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 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_annotated.png")  # Change to actual test image
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_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarning: You are using a softmax over axis -1 of a tensor of sh
  warnings.warn(
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/losses/losses.py:33: SyntaxWarning: In loss categorical_crossentropy, expected y_p
  return self.fn(y_true, y_pred, **self._fn_kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 745ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 926ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 551ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 569ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 520ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 507ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 572ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 515ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 592ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 596ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
Predicted Label: UNKNOWN
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarning: You are using a softmax over axis -1 of a tensor of sh
  warnings.warn(
```

```
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"), "UNKNOWN"))  # Get label from text file
    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, random_state=42)

# 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 for classification
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 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")  # Change to actual test image
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_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 713ms/step - accuracy: 0.3401 - loss: 3.8044 - val_accuracy: 0.4500 - val_loss: 1.9941
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 995ms/step - accuracy: 0.5796 - loss: 1.9511 - val_accuracy: 0.7250 - val_loss: 1.7102
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 650ms/step - accuracy: 0.6413 - loss: 1.7909 - val_accuracy: 0.7250 - val_loss: 1.6969
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 669ms/step - accuracy: 0.6331 - loss: 1.7556 - val_accuracy: 0.7250 - val_loss: 1.7418
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 667ms/step - accuracy: 0.6589 - loss: 1.5995 - val_accuracy: 0.7500 - val_loss: 1.6934
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 667ms/step - accuracy: 0.6627 - loss: 1.5745 - val_accuracy: 0.7500 - val_loss: 1.8267
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 953ms/step - accuracy: 0.6820 - loss: 1.4911 - val_accuracy: 0.7500 - val_loss: 1.7837
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 666ms/step - accuracy: 0.6851 - loss: 1.3577 - val_accuracy: 0.7500 - val_loss: 1.7352
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 670ms/step - accuracy: 0.7127 - loss: 1.2227 - val_accuracy: 0.7500 - val_loss: 1.8169
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 858ms/step - accuracy: 0.7145 - loss: 1.1131 - val_accuracy: 0.7500 - val_loss: 1.6221
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
Predicted Label: UNKNOWN
```

```
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 scipy.spatial import distance

# Folder paths
OCR_IMAGES_FOLDER = "/content/extracted_files"
OCR_TEXT_FOLDER = "/content/extracted1_files"
ORIGINAL_DATASET_FOLDER = "/content/pdf_images"

# 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)

# Load images and extract text labels
def load_images(image_folder, text_data):
    images, labels, coordinates = [], [], []
    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)
            label = text_data.get(img_file.replace(".png", ".txt"), "UNKNOWN")
            labels.append(label)

            # Extract coordinates using OCR
            extracted_data = pytesseract.image_to_data(img, output_type=pytesseract.Output.DICT)
            if "text" in extracted_data:
                for i, txt in enumerate(extracted_data["text"]):
                    if txt.strip():
                        x, y, w, h = (
                            extracted_data["left"][i],
                            extracted_data["top"][i],
                            extracted_data["width"][i],
                            extracted_data["height"][i],
                        )
                        coordinates.append((label, (x + w // 2, y + h // 2)))  # Center of detected text
    return np.array(images), labels, coordinates

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

# Load images from the original dataset
X_original, y_original, coords_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
coordinates = coords_ocr + coords_original  # Merge coordinates

# 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, random_state=42)

# 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 for classification
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 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]

# Function to associate detected components with their correct labels using spatial alignment
def associate_labels_with_components(pipeline_segments, text_coordinates):
    associations = {}
    for text, text_pos in text_coordinates:
        nearest_segment = min(pipeline_segments, key=lambda seg: distance.euclidean(text_pos, seg))
        associations[text] = nearest_segment
    return associations

# Example pipeline component positions (to be replaced with actual detections)
pipeline_segments = [(200, 300), (400, 500), (600, 700)]  # Example detected pipeline coordinates
associations = associate_labels_with_components(pipeline_segments, coordinates)

# Print results
for label, coord in associations.items():
    print(f"Component: {label} -> Assigned to Pipeline at {coord}")

# Example usage for label prediction
test_img = os.path.join(OCR_IMAGES_FOLDER, "/content/pdf_images/page_12.png")  # Change to actual test image
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/PIL/PngImagePlugin.py in _save(im, fp, filename, chunk, save_all)
   1362     try:
-> 1363         rawmode, bit_depth, color_type = _OUTMODES[outmode]
   1364     except KeyError as e:

KeyError: 'F'

The above exception was the direct cause of the following exception:

OSError                                   Traceback (most recent call last)
                                          8 frames
/usr/local/lib/python3.11/dist-packages/PIL/PngImagePlugin.py in _save(im, fp, filename, chunk, save_all)
   1364     except KeyError as e:
   1365         msg = f"cannot write mode {mode} as PNG"
-> 1366         raise OSError(msg) from e
   1367
   1368     #

OSError: cannot write mode F as 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):
```

```
    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, pix.n)
        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/original_dataset_images")

# 🔹 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_folder)
                   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", ".txt"), "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("/content/original_dataset_images", ocr_texts)

# 🔹 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, random_state=42)

# 🔹 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 for classification
])

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

# 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")  # Change to actual test image
predicted_label = predict_label(test_img)
print("Predicted Label:", predicted_label)
```

```
Requirement already satisfied: PyMuPDF in /usr/local/lib/python3.11/dist-packages (1.25.3)
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 7s 742ms/step - accuracy: 0.4347 - loss: 3.3681 - val_accuracy: 0.7872 - val_loss: 1.5433
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 668ms/step - accuracy: 0.6939 - loss: 1.8545 - val_accuracy: 0.7872 - val_loss: 1.2641
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 880ms/step - accuracy: 0.6908 - loss: 1.7485 - val_accuracy: 0.7872 - val_loss: 1.2128
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 695ms/step - accuracy: 0.6645 - loss: 1.7723 - val_accuracy: 0.7872 - val_loss: 1.1585
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 698ms/step - accuracy: 0.6857 - loss: 1.4886 - val_accuracy: 0.7872 - val_loss: 1.1647
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 841ms/step - accuracy: 0.6798 - loss: 1.5066 - val_accuracy: 0.8298 - val_loss: 1.1273
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 623ms/step - accuracy: 0.6631 - loss: 1.5732 - val_accuracy: 0.8511 - val_loss: 1.1150
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 726ms/step - accuracy: 0.7392 - loss: 1.2274 - val_accuracy: 0.8511 - val_loss: 1.1284
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 645ms/step - accuracy: 0.6999 - loss: 1.3442 - val_accuracy: 0.8511 - val_loss: 1.1204
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 692ms/step - accuracy: 0.7344 - loss: 1.1675 - val_accuracy: 0.8511 - val_loss: 1.1418
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
Predicted Label: UNKNOWN
```

```
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, pix.n)
        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_IMAGES_FOLDER)

# 🔹 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]  # Improve OCR accuracy
        text = pytesseract.image_to_string(img).strip()
        extracted_texts[os.path.basename(img_path)] = text if text else "UNKNOWN"
    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_folder)
                   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"))  # Use extracted OCR labels

    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, random_state=42)

# 🔹 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=['accuracy'])

# 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 actual test image
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_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 707ms/step - accuracy: 0.6982 - loss: 0.6618 - val_accuracy: 0.9787 - val_loss: 0.2732
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 882ms/step - accuracy: 0.9786 - loss: 0.1658 - val_accuracy: 0.9787 - val_loss: 0.1562
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 663ms/step - accuracy: 0.9832 - loss: 0.1569 - val_accuracy: 0.9787 - val_loss: 0.1599
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 654ms/step - accuracy: 0.9734 - loss: 0.1588 - val_accuracy: 0.9787 - val_loss: 0.1490
```

```
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 882ms/step - accuracy: 0.9617 - loss: 0.1939 - val_accuracy: 0.9787 - val_loss: 0.1126
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 666ms/step - accuracy: 0.9591 - loss: 0.1372 - val_accuracy: 0.9787 - val_loss: 0.1139
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 869ms/step - accuracy: 0.9721 - loss: 0.0856 - val_accuracy: 0.9787 - val_loss: 0.1208
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 725ms/step - accuracy: 0.9911 - loss: 0.0470 - val_accuracy: 0.9787 - val_loss: 0.1020
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 661ms/step - accuracy: 0.9770 - loss: 0.0770 - val_accuracy: 0.9787 - val_loss: 0.0963
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 886ms/step - accuracy: 0.9865 - loss: 0.0709 - val_accuracy: 0.9787 - val_loss: 0.1103
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distribu
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, pix.n)
        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_IMAGES_FOLDER)

# 🔹 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]  # Improve OCR accuracy

        # 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_FOLDER, f)
                                                 for f in os.listdir(IC_DETECTED_FOLDER) if f.endswith(".png")])

# 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(extra)}

    return mismatches

# Perform validation
connection_mismatches = validate_ic_connections(original_ic_connections, detected_ic_connections)

# 🔹 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 aicatipte ten were ey ms cat', 'Oricarapennossncas w serene']
```