Custom-Object Character Recognition(OCR) on AWS

Build a Custom OCR by combining YOLO and Tesseract, to read the specific contents of a Lab Report and convert it into an editable file. Use YOLO_V3 to trained on the personal dataset. Then the coordinates of the detected objects are passed for cropping the detected objects and storing them in another list. This list is passed through the Tesseract to get the desired output.

Model

- You can train a custom YOLO_V3 model using your custom dataset.
- Make a folder named model and put the weights file inside it.

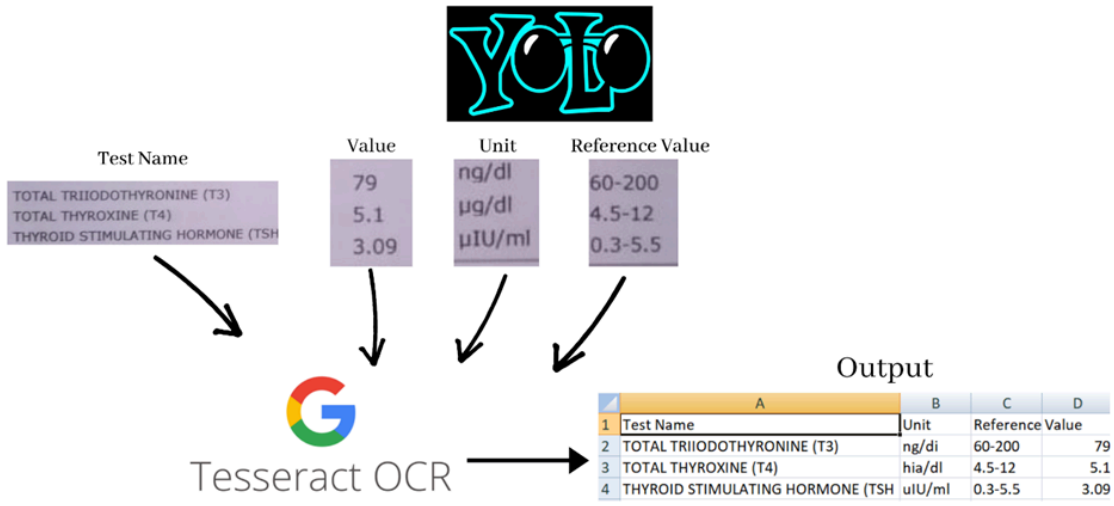
Data

- Link for the dataset https://drive.google.com/drive/folders/1uAc8xE6AS8YCb65iNMFyEfB5dzAWsgFN?usp=sharing

Dependencies

- Install Tesseract OCR Engine in the system
- Install Pytesseract library pip install pytessercat
- Install OpenCV pip opencv

Workflow



Getting Started

python Custom_OCR.py --image <yourimage.jpg>

Step by Step-by-step workflow

This section will highlight the steps I took to implement the Custom-OCR with YOLOv3 and potential areas to be worked on further.

This will show the step-by-step workflow on the following original image.

Detected regions

The first step of the process is taking the bounding box coordinates from YOLOv3 and simply taking the region within the bounds of the box. As this image is super small, we use cv2.resize() to blow the image up 3x its original size.

Then we convert the image to grayscale and apply a small Gaussian blur to smooth it out.

The image is then thresholded to white text with a black background and has Otsu's method also applied. This white text on a black background helps to find the contours of the image.

Then we apply a bitwise_not mask to flip the image to black text on a white background which Tesseract is more accurate with.

The preprocessed images are then passed over to Tesseract and the output is saved as a csv file.

Here's a breakdown of project tasks to do in Amazon SageMaker and S3 for your OCR project:

1. Set Up the Environment

- **Task 1.1: Create an S3 Bucket**
    - Create an S3 bucket to store your dataset, model weights, and output files.
    - Define the folder structure within the bucket for datasets, models, and results.
- **Task 1.2: Set Up SageMaker Notebook Instance**
    - Launch a SageMaker notebook instance.
    - Install necessary dependencies: Tesseract, PyTesseract, OpenCV, and YOLOv3.
    - Ensure that the notebook instance has access to the S3 bucket.

2. Data Preparation

- **Task 2.1: Upload Dataset to S3**
    - Download the dataset from the provided link.
    - Upload the dataset to the appropriate folder in the S3 bucket.

- **Task 2.2: Prepare the Dataset**
    - Use SageMaker to preprocess the dataset, such as resizing images and creating annotations for YOLOv3 training.
    - Store the preprocessed data back in the S3 bucket.

3. Model Training

- **Task 3.1: Train YOLOv3 Model**
    - Use SageMaker's training jobs to train the YOLOv3 model with your custom dataset.
    - Save the trained model weights to the S3 bucket under the 'model' folder.
- **Task 3.2: Model Validation**

Custom-Object Character Recognition(OCR) on AWS

Build a Custom OCR by combining YOLO and Tesseract, to read the specific contents of a Lab Report and convert it into an editable file. Use YOLO_V3 to trained on the personal dataset. Then the coordinates of the detected objects are passed for cropping the detected objects and storing them in another list. This list is passed through the Tesseract to get the desired output.

Model

- You can train a custom YOLO_V3 model using your custom dataset.
- Make a folder named model and put the weights file inside it.

Data

    - Validate the trained model using a subset of the data.
    - Upload validation results, including images with bounding boxes, to the S3 bucket.

4. Inference and Post-Processing

- **Task 4.1: Run Inference**
    - Deploy the YOLOv3 model on SageMaker for inference.
    - For each test image, store the detected object coordinates in a list and save the output to S3.
- **Task 4.2: Preprocess Detected Regions**
    - Use SageMaker to apply image preprocessing steps: resizing, converting to grayscale, applying Gaussian blur, and thresholding.
    - Store preprocessed images in S3.
- **Task 4.3: Extract Text Using Tesseract**
    - Pass the preprocessed images through Tesseract using SageMaker.

    - Save the extracted text as CSV files in S3.

5. Evaluation and Optimization

- **Task 5.1: Evaluate OCR Performance**
    - Analyze the accuracy of the OCR output by comparing it with the ground truth.
    - Store evaluation metrics and reports in S3.
- **Task 5.2: Optimize the Workflow**
    - Identify potential areas of improvement, such as fine-tuning the model, optimizing preprocessing, and improving Tesseract accuracy.
    - Implement and test optimizations.

6. Automation and Deployment

- **Task 6.1: Automate the Workflow**
    - Create a SageMaker pipeline to automate the entire process from data preparation to OCR output.
    - Store pipeline logs and outputs in S3.
- **Task 6.2: Deploy the Solution**
    - Deploy the entire solution on SageMaker for real-time OCR processing of lab reports.
    - Set up triggers to process new files uploaded to S3 automatically.

7. Documentation and Reporting

- **Task 7.1: Document the Workflow**
    - Create detailed documentation of the entire process, including model training, inference, and post-processing steps.
    - Store documentation files in S3.
- **Task 7.2: Generate Final Reports**
    - Compile a final report summarizing the project, including model performance, challenges, and next steps.
    - Upload the report to S3.

These tasks will guide you through the process of building and deploying your Custom OCR solution on Amazon SageMaker and S3.

Guidelines for Cost Management in Cloud Service:

Estimating the cost of the project for processing just two images involves considering the minimum usage of the services required. Here's a rough breakdown:

1. Amazon SageMaker

- **Notebook Instance:**
  - Assuming you use a small instance like `ml.t3.medium` for a few hours (say 2 hours) to set up your environment, train the model, and run inference.
  - **Cost:** Around $0.05 per hour, so approximately $0.10.
- **Training Jobs:**
  - Since you're working with just two images, the training time should be minimal. A short training job on a `ml.m5.large` instance could be enough.
  - **Cost:** Around $0.10 per hour, so approximately $0.10 for a short session.

- **Inference:**
  - Deploying the model for a brief period (say 1 hour) using a `ml.m5.large` instance.
  -
  - **Cost:** Around $0.10 per hour, so approximately $0.10.

2. Amazon S3

- **Storage:**
  - The storage required for your dataset, model weights, and output files is minimal.
  - Assuming you store 1 GB of data (which is likely an overestimate), the cost is around $0.023 per GB per month.
  - **Cost:** Less than $0.01 for the time you'll use it.
- **Data Transfer:**
  - Data transfer costs are minimal if you're not moving large amounts of data out of AWS.
  - **Cost:** Likely negligible for this small scale.

3. Additional Services (Optional)

- **Lambda/CloudWatch:**
  - If used, the cost for a few executions and logs will be minimal.
  - **Cost:** Less than $0.01.

Total Estimated Cost

- **Approximate Total Cost:** Around $0.30 - $0.50.

This estimate assumes minimal usage and the smallest possible resources to complete the task for just two images. The cost will scale up if you train larger models, process more images, or use higher-tier instances.

Project Timeline:1 month.

Deadline 20th March, 2025