“Aadhar Card Text Extraction System Using Machine Learning and OCR”

PRATIK BHOUD
10/10/2024

Problem Statement:

Identity verification is a critical process in many sectors, including government services and insurance. Manual data entry from identity documents is prone to errors and inefficiencies. By leveraging machine learning algorithms, we can automate the extraction of key details from identity documents, ensuring accuracy and efficiency.

Key Objectives:

1. Automatically extract and validate text from identity documents like Aadhar.
2. Identify key details such as Name, Gender, Date of Birth, and Address with high accuracy.
3. Convert unstructured image data into structured formats like CSV or JSON for easy integration into databases.
4. Ensure security and privacy of sensitive information while processing identity documents.

Dataset Preparation:

The project requires a dataset of images of Aadhar cards, preferably from different states and different formats to cover variations. The dataset must include:

- **Synthetic Images** or real Aadhar card images with manual labelling of fields like Name, DOB, Address, etc.
- **Varied Image Quality:** Include different image resolutions, lighting conditions, and scanned quality (blurry, clear, etc.).

For testing, you can create a dataset by collecting various Aadhar card formats available online, ensuring data anonymization if necessary.

System Architecture:

1. **Image Acquisition:**
    - Input: Aadhar card image uploaded by the user.
    - Image Format: Supports PNG, JPG, and PDF formats.
    - Preprocessing: Convert images to grayscale and apply denoising techniques to improve OCR accuracy.
2. **Optical Character Recognition (OCR):**
    - Use **Tesseract OCR** to detect text in the image.
    - Extract all the text fields present in the Aadhar card using OCR and convert them into plain text for further processing.
3. **Field Detection and Extraction:**
    - After OCR processing, detect important fields such as Name, DOB, Address, and Gender using regex and Named Entity Recognition (NER).
    - **Regex-based Field Extraction:** For example, to identify dates and name patterns like 'DOB' or 'Name'.
    - **Named Entity Recognition (NER):** Use NLP-based entity detection models (Spacy) to identify address fields and names.
4. **Post-Processing:**
    - Cleaning the extracted text by removing unwanted characters, handling misspelled words, and normalizing the text for better accuracy.
    - **Validation:** Cross-check the fields like DOB to ensure valid date formats and check if the names are recognized by using pre-compiled lists of common names.
5. **Storage and Output:**
    - The extracted information is saved in structured formats (CSV, JSON) that can be exported or integrated into enterprise systems.
    - Generate a summary of extracted data for user verification.

Feature Engineering:

To improve the quality of information extraction, various features must be engineered:

- **Field-Specific Regular Expressions:** For each field like Date of Birth, Name, and Address, create specific regex to match standard formats.
- **Name Entity Recognition:** Leverage libraries like Spacy to detect person names and addresses in the text output.
- **Heuristic Rules for DOB and Gender Detection:** E.g., Look for specific keywords like "DOB" followed by date patterns, or identify "Male", "Female" for gender fields.

Implementation:

Step 1: Pre-processing

```
import cv2
import pytesseract
from PIL import Image

# Image Preprocessing
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Optional: Apply Denoising
    denoised = cv2.fastNlMeansDenoising(gray, h=30)
    return denoised

image_path = 'aadhar_card.jpg'
preprocessed_image = preprocess_image(image_path)
cv2.imwrite('preprocessed_image.jpg', preprocessed_image)
```

Step 2: Text Extraction Using Tesseract OCR

```
def extract_text_from_image(image_path):
    img = Image.open(image_path)
    text = pytesseract.image_to_string(img)
    return text

# Extract text from preprocessed image
extracted_text = extract_text_from_image('preprocessed_image.jpg')
print(extracted_text)
```

Step 3: Field Extraction

```
import re

# Extract Name, DOB, Gender using regex
def extract_fields(text):
    name = re.search(r'Name\s*:\s*([A-Za-z\s]+)', text)
    dob = re.search(r'DOB\s*:\s*(\d{2}/\d{2}/\d{4})', text)
    gender = re.search(r'(Male|Female|Transgender)', text)

    return {
        "Name": name.group(1) if name else None,
        "DOB": dob.group(1) if dob else None,
        "Gender": gender.group(0) if gender else None
    }

# Example of extraction
fields = extract_fields(extracted_text)
print(fields)
```

Step 4: Post Processing & Validation

Ensure the data is cleaned, processed, and validated before storing it:

```python
def validate_dob(dob):
    # Validate date format
    try:
        date = pd.to_datetime(dob, format='%d/%m/%Y')
        return True
    except:
        return False

if validate_dob(fields['DOB']):
    print("DOB is valid")
else:
    print("DOB is invalid")
```

Deployment:

The system can be deployed on **Streamlit** or **Flask** for a web-based interface where users can upload their Aadhar card image and get the extracted details.

```python
import streamlit as st

st.title("Aadhar Card Text Extraction")

uploaded_file = st.file_uploader("Choose an image file", type=["jpg", "png", "jpeg"])

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption='Uploaded Image.', use_column_width=True)

    # Extract text from the image
    text = extract_text_from_image(uploaded_file)
    st.write(text)

    # Extract specific fields
    fields = extract_fields(text)
    st.write(f"Name: {fields['Name']}")
    st.write(f"DOB: {fields['DOB']}")
    st.write(f"Gender: {fields['Gender']}")
```

Business Objective:

The primary objective of the Aadhar Card Text Extraction system is to create an automated solution for extracting and processing personal identification information from Aadhar card images. The solution can be scaled and monetized in various industries where identification is crucial, such as banks, insurance companies, and government agencies.

Target Customers:

- **Banks**: For KYC processes to streamline onboarding of new customers.
- **Insurance Companies**: To validate claims and verify policyholder information.
- **Government Agencies**: For various applications like subsidies, tax returns, etc.
- **E-commerce Platforms**: To verify user identity for high-value transactions.
- **Telecommunication Providers**: For SIM card activation and identity validation.

Revenue Streams:

- **SaaS Model (Subscription-based)**: Offer monthly/annual subscriptions for businesses based on the number of documents processed. For example, a basic plan for small businesses and enterprise-level pricing for larger organizations.
- **Pay-per-Document Model**: Charge a fee per document processed, which allows flexibility for smaller companies that may not need continuous document processing.
- **On-Premises License Fee**: For businesses that want to handle data in-house for privacy or legal reasons, offer one-time license fees for enterprise software installation.
- **Custom Development**: Provide custom solutions or integration with business systems for clients with specific needs.

Monetization Strategy:

1. **Subscription Pricing Tiers**:
    - **Basic Plan**: $50/month (up to 500 documents)
    - **Professional Plan**: $200/month (up to 5,000 documents)
    - **Enterprise Plan**: $1,000/month (unlimited documents)

2. **Per Document Pricing**:
    - $0.20 per document processed (for smaller organizations that don't need a subscription model)
    -
3. **Customization/Integration Fees**:
    - $5,000 to $10,000 for API integrations into existing systems (e.g., bank systems, insurance CRMs)

Financial Modeling for Aadhar Card Text Extraction System

Costs:

1. **Development Costs**:
    - Initial development (Software engineers, data scientists): $30,000
    - Testing and quality assurance: $5,000
    - Deployment costs: $3,000
2. **Operating Costs**:
    - Cloud Hosting: $500/month
    - Maintenance and bug fixing: $1,000/month
    - Salaries for ongoing technical support: $5,000/month
3. **Sales and Marketing**:
    - Marketing budget (advertising, customer acquisition): $2,000/month
    - Sales team salary: $7,000/month

Revenue Projections (Year 1):

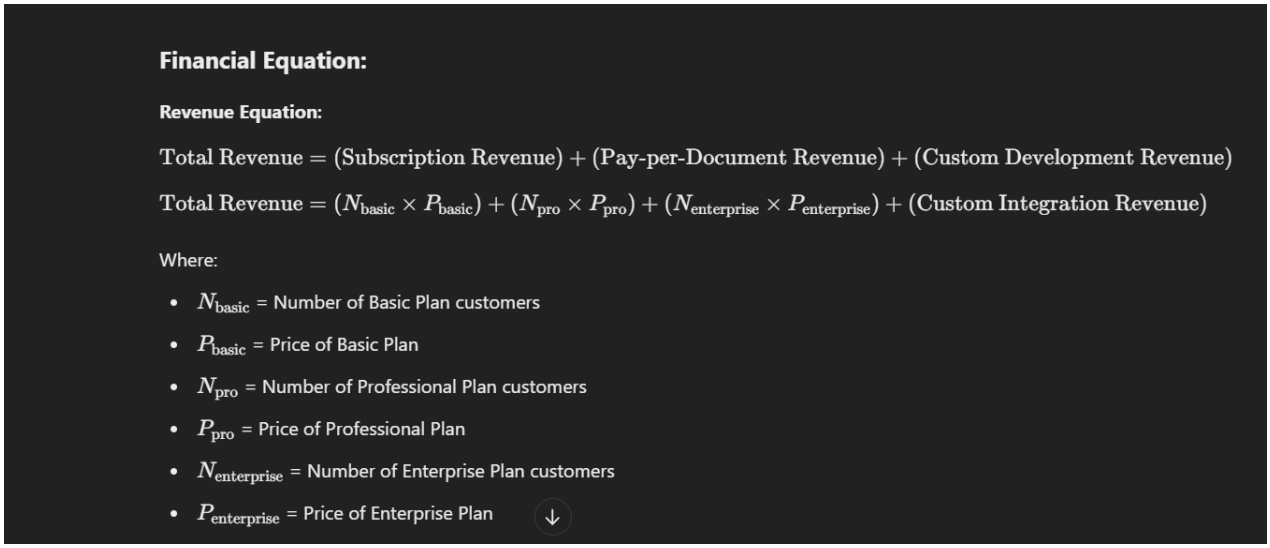
- **Subscription Model**:
    - Expected initial customers: 10 clients in the basic plan, 5 clients in the professional plan, and 2 in the enterprise plan.
    - Monthly Revenue from subscriptions:
        - Basic: 10 clients x $50 = $500
        - Professional: 5 clients x $200 = $1,000
        - Enterprise: 2 clients x $1,000 = $2,000
    - Total Monthly Revenue: $3,500
    - Total Annual Revenue: $3,500 x 12 = $42,000

- **Custom Integration**:
    - Estimated custom integrations: 3 clients at $7,000 each
    - Revenue from custom solutions: 3 x $7,000 = $21,000

- **Total Projected Revenue for Year 1**: $42,000 + $21,000 = **$63,000**

Financial Equation:

Revenue Equation:



Profit Equation:

Profit=Total Revenue−Total Costs\text{Profit} = \text{Total Revenue} - \text{Total Costs}Profit=Total Revenue−Total Costs

Where:

- **Total Costs** include development, operational, and marketing costs.

Example:

Let’s assume:

- 10 basic plan customers, 5 professional plan customers, and 2 enterprise plan customers.
- Custom Integration Revenue = $21,000.

The equation becomes:

Total Revenue
=(10×50)+(5×200)+(2×1000)+21000=500+1000+2000+21000=$24,500 per mo
nth.\text{Total Revenue} = (10 \times 50) + (5 \times 200) + (2 \times 1000) +
21000 = 500 + 1000 + 2000 + 21000 = \$24,500 \, \text{per
month}.Total Revenue=(10×50)+(5×200)+(2×1000)+21000=500+1000+2000+
21000=$24,500per month.

GitHub link - https://github.com/PratikMBhoud/Feynn-Lab