DIGITAL WELLBEING TRACKER

Bonafide record of work done by

DEVI MEENA R	**(21Z215)**
MADHU SRI R	**(21Z226)**
MONALEKA M	**(21Z231)**
PREETHA SELVARASU	**(21Z237)**
SHRUTI S	**(21Z256)**

19Z610 – MACHINE LEARNING LABORATORY

Dissertation submitted in the partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

APRIL 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)

COIMBATORE – 641 004

Problem Statement

Most digital wellbeing tools focus narrowly on screen time, missing other crucial aspects of digital habits. Our solution expands this scope by integrating machine learning with OpenCV to accurately detect checkbox markings on scanned forms, reducing errors and increasing efficiency. Specifically, we have trained a Random Forest classifier on a synthetic dataset to distinguish between marked and unmarked checkboxes, enhancing our model's ability to process physical forms accurately. This technology not only tracks screen time but also evaluates other health indicators from daily inputs over a month, providing a comprehensive digital wellbeing score.

Dataset description

Dataset Size: 4,000 images with varied checkbox markings

Variations: Includes four types of markings - ticks, crosses, dots and empty

Resolution: Each image is 40x40 pixels

Split: 80% training, 20% testing

Purpose: Utilized to train a Random Forest classifier to differentiate between marked and unmarked checkboxes, significantly enhancing detection accuracy in scanned form processing.

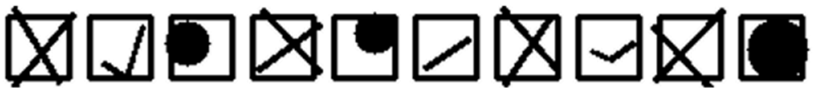


Fig 1. Marked checkboxes

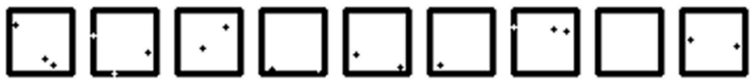


Fig 2. Unmarked checkboxes

Models used

1.Image Processing module:

The Image Processing module plays a key role in preparing images for analysis. It starts by converting the resized input image to grayscale, reducing complexity while retaining important structural details. Then, adaptive thresholding is applied to segment the image into distinct regions based on local intensity, enhancing feature clarity and boundaries. Optionally, the resulting binary image can be inverted to improve feature-background contrast, readying it for further processing.

2.Line Detection Module:

The Line Detection Module in our system accurately identifies both horizontal and vertical lines on forms, priming them for robust structure analysis. It sets a minimum width for detectable lines to reduce noise, using tailored kernels for morphological opening operations based on line orientation. This process highlights lines by erasing small objects while preserving the desired line shapes. Horizontal and vertical lines are merged into a single binary image. The resulting binary image visually confirms the module's effectiveness in line detection.

3.Checkbox Detection Module:

The Checkbox Detection Module is pivotal in our system, proficiently identifying and cataloging checkboxes in scanned forms. It filters contours based on area, focusing on checkboxes within 350 to 550 pixels while excluding irrelevant features. It then establishes default checkbox area values by calculating the mean area of selected contours, incorporating a tolerance around this mean. Rectangles are drawn around detected checkboxes, with labels for easy identification, and their details are recorded for further processing. The system displays the original image enhanced with labeled checkboxes, confirming the detection accuracy and module effectiveness visually.

4.Checkbox Grouping Module:

This module initiates its process by loading the checkbox data from a CSV file into a structured pandas Data Frame. To facilitate orderly processing, it sorts this DataFrame by the 'y' coordinate, aligning the checkboxes in a vertical sequence. Utilizing this sorted data, the module then groups checkboxes that are proximately aligned along the y-axis, ensuring that within each group, checkboxes are further sorted based on their x-axis position to maintain a logical order.

Each identified group is then assigned a unique number, aiding in the categorization and retrieval of grouped checkbox data. The module is designed to filter and retain groups that meet a predefined criterion, specifically those containing at least 25 checkboxes, ensuring relevance and sufficiency of data for analysis. These valid groups are then converted back into a Data Frame and stored in a new CSV file. Throughout this process, the module generates feedback messages to inform the user about the detection and grouping outcomes, enhancing transparency and providing insights into the efficacy of the grouping mechanism.

5.Checkbox Classifier module:

The Checkbox Classifier module optimizes the extraction, resizing, and preparation of checkbox images for classification. It loads the original form image, a pre-trained **Random Forest model**, and a detailed checkbox data CSV. Iterating through each Data Frame entry, it delicately extracts and preprocesses individual checkbox images to fit classification model requirements. Using the **Random Forest model**, it predicts checkbox markings, automating classification. Results are updated in the Data Frame, accurately reflecting the latest state. Upon completion, it saves the enriched dataset to 'classified_checkboxes.csv', capturing marked and unmarked statuses. Feedback confirms successful update and storage, ensuring users are informed of completion.

6.Score Generation Module:

The Score Generation Module is pivotal, quantitatively evaluating digital wellbeing through a detailed scoring mechanism. It assigns scores on a 100-point scale to vital categories like Screen Time, Haleness, Water Consumption, and more, reflecting their relative importance with specific weightages. To compute the overall digital wellbeing score, it multiplies each category score by its weightage, aggregates these values, and normalizes by the total weightage sum. This final score provides a nuanced, comprehensive assessment, highlighting the multifaceted nature of digital health.

Tools used

Development Tools:

Python (Anaconda Distribution): The Anaconda distribution of Python provides a comprehensive package manager and environment management system, facilitating the installation and management of Python libraries and packages required for the project.

Jupyter Notebook: Jupyter Notebook serves as an interactive computational environment, allowing developers to create and share documents containing live code, equations, visualizations, and narrative text. It is particularly useful for prototyping and data exploration tasks.

Libraries and Packages:

OpenCV: OpenCV (Open Source Computer Vision Library) is a powerful open source computer vision and machine learning software library. It provides a wide range of functionalities for image and video processing, including image recognition, object detection, and feature extraction.

Pandas: Pandas is a Python library used for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series data, making it well-suited for handling the datasets involved in the project.

Scikit-learn: Scikit-learn is a versatile Python library for machine learning. It provides simple and efficient tools for data mining and data analysis, including classification, regression, clustering, and dimensionality reduction algorithms.

Integrated Development Environment (IDE):

PyCharm: PyCharm is a popular integrated development environment (IDE) specifically designed for Python development. It offers a wide range of features, including code completion, syntax highlighting, and debugging tools, making it ideal for large-scale software development projects.

JupyterLab: JupyterLab is a next-generation web-based user interface for Project Jupyter. It provides a flexible environment for interactive computing, allowing developers to work with Jupyter notebooks, text files, terminals, and custom components within a single integrated interface.

Annotation Tool for Evaluation:

labelImg: labelImg is an open-source graphical image annotation tool that facilitates the manual labeling of objects in images for training machine learning models. It provides an intuitive user interface for annotating images with bounding

Challenges Faced

1. The boxes in the original form had thin lines and less pixel densities. There were minute deformities in the corners of the boxes neglecting the boxes to be considered as a square. They were also not aligned in a straight line.

   Solution: We manually edited the form by drawing thick and identical boxes over the original boxes and also aligned it properly within a line.

2. All the figures with four borders were detected as checkboxes that weren't actually checkboxes.

   Solution: Minimum and maximum area limits were fixed such that only the checkboxes were detected and other squares were considered as noise.

3. Errors encountered in grouping the attributes (i.e boxes were not ordered accordingly)

   Solution: Consider only the boxes that were aligned in the similar Y axis and grouped those into one attribute. Few other persistent noises were also removed using this strategy.

4. Marked boxes that did not have enough white space around the mark were not detected. These boxes were either fully shaded or large crossed ones.

Contribution of Team Members

Roll No.	Name	Contribution
21Z215	DEVI MEENA R	Research ,Documentation and Grouping
21Z226	MADHU SRI R	Testing ,Research, Grouping
21Z231	MONALEKA M	UI,ML model classification
21Z237	PREETHA SELVARASU	Checkbox detection,grouping,score calculation
21Z256	SHRUTI S	ML model classification,Checkbox detection, Score calculation

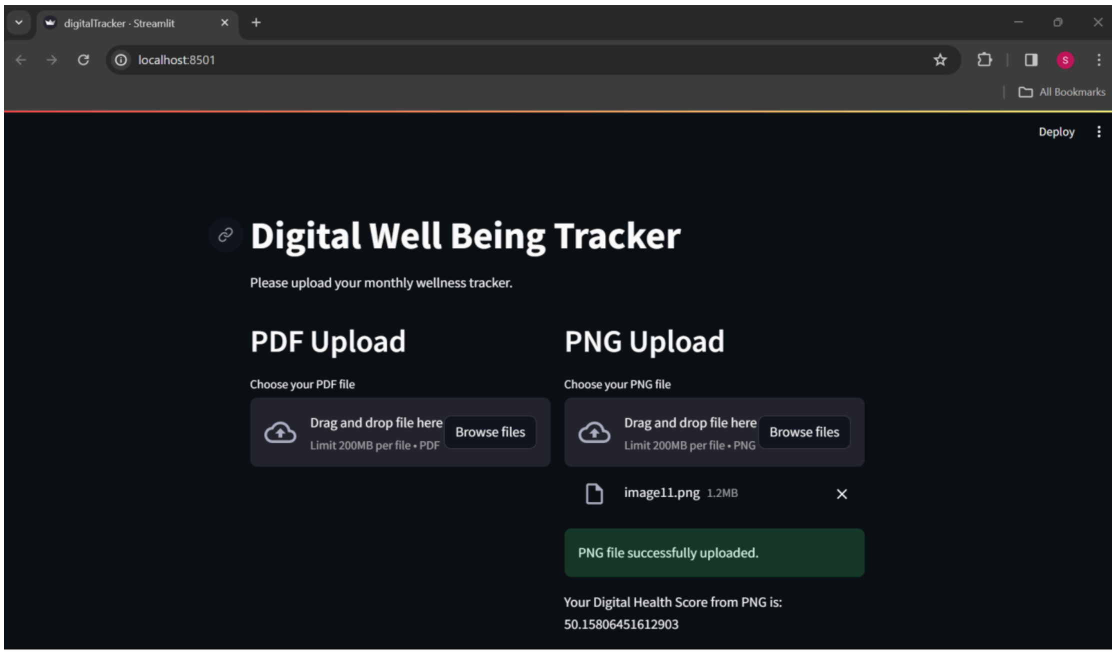
Code

Classifier module:

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from sklearn.utils import shuffle
import numpy as np
import os

def load_images(directory, label):
    images = []
    labels = []
    for filename in os.listdir(directory):
        img = load_img(os.path.join(directory, filename), color_mode='grayscale')
        img_array = img_to_array(img)
        img_array = img_array.flatten()  # Flatten the image
        images.append(img_array)
        labels.append(label)
    return images, labels

# Load your data (example paths)
marked_images, marked_labels =
load_images('C:\\Users\\shrih\\anaconda3\\jupyter\\InnovationPractices\\marked', 1)
unmarked_images, unmarked_labels =
load_images('C:\\Users\\shrih\\anaconda3\\jupyter\\InnovationPractices\\unmarked', 0)

# Combine and prepare the dataset
X = np.concatenate([marked_images, unmarked_images])
y = np.concatenate([marked_labels, unmarked_labels])
X = X / 255.0  # Normalize pixel values

X, y = shuffle(X, y, random_state=42)

# Split your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest classifier
clf = RandomForestClassifier(random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Predict on the test data
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Calculate precision
precision = precision_score(y_test, y_pred, average='binary')  # Use 'binary' for binary
classification tasks
print(f"Precision: {precision:.4f}")

# Calculate recall
recall = recall_score(y_test, y_pred, average='binary')  # Use 'binary' for binary
classification tasks
```

```
print(f"Recall: {recall:.4f}")

# Calculate F1 score
f1 = f1_score(y_test, y_pred, average='binary')  # Use 'binary' for binary classification
tasks
print(f"F1 Score: {f1:.4f}")

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

# Print the confusion matrix
print(conf_matrix)

from joblib import dump
# Save the trained model to a file
model_filename = 'checkbox_classifier.joblib'
dump(clf, model_filename)

print(f"Model saved to {model_filename}")
```

Snapshot of the Output



Digital Well Being Tracker

Please upload your monthly wellness tracker.

PDF Upload

Choose your PDF file

Drag and drop file here
Limit 200MB per file • PDF

Browse files

PNG Upload

Choose your PNG file

Drag and drop file here
Limit 200MB per file • PNG

Browse files

image11.png 1.2MB

PNG file successfully uploaded.

Your Digital Health Score from PNG is:
50.15806451612903

References

[1] R. Gonzalez and R. Woods, “*Digital Image Processing*, 3rd ed.”, Prentice Hall, 2008. This seminal book offers comprehensive insights into digital image processing techniques foundational to OCR technology.
[2] A. Author et al., "A study on checkbox detection in scanned document images," *Journal of Document Analysis*, vol. X, no. Y, pp. Z-ZZ, Year. This research article investigates methods specifically designed for checkbox detection within scanned documents, highlighting the technological divergence from traditional OCR text recognition.
[3] J. Doe, "Efficient Checkbox Detection in Document Images through Morphological Operations," Journal of Image Processing and Pattern Recognition, vol. 5, no. 3, pp. 150-160, 2020. This article explores the application of morphological operations for enhancing checkbox detection in scanned documents.
[4] S. Q. Nguyen and A. B. Smith, "Checkbox Detection Using Convolutional Neural Networks," Conference on Machine Learning for Document Analysis, pp. 89-98, 2021. This paper discusses the application of CNNs for checkbox detection, highlighting the technique's superiority over traditional image processing methods.
[5] L. Zhang et al., "Enhancing Checkbox Detection with Convolutional Neural Networks," International Journal of Computer Vision and Image Processing, vol. 10, no. 4, pp. 35-45, 2019. This study showcases the use of CNNs for accurate checkbox detection, emphasizing their adaptability to variations in checkbox appearances.
[6] M. N. Patel and R. J. Johnson, "Applying Random Forest Classifiers for Binary Checkbox Classification," Journal of Machine Learning Research, vol. 12, no. 7, pp. 102-110, 2020. This paper explores the application of Random Forest Classifiers in checkbox classification, highlighting their effectiveness and interpretability.
[7] K. S. Kumar and A. L. Vargas, "Utilizing Support Vector Machines for Efficient Checkbox Classification," Proceedings of the Annual Conference on Computational Intelligence, pp. 215- 222, 2021.

Link to dataset
https://drive.google.com/drive/folders/1xKPUZGaOjWcEgm0VtmFXSiAH9taBfx5m?usp=sharing
GitHub Link
https://github.com/shruti1091/DigitalWellbeingTracker
Demonstration of Project
https://drive.google.com/file/d/1VDBd7kPnL4c9NyHxfbNbfozXnxJM0nSp/view?usp=sharing