KeepReading
ver. 1.30.58
12/06/2023

Table of Contents

Introduction

Mission

The goal of this software ("KeepReading"), is to provide security solutions for data destruction on hard drives for TechR2.

Strategy

This software ("KeepReading") enables TechR2 employees to be more efficient when it comes to clearing out hard-drive by speeding up the process of retrieving data from labels limited but not including: Serial numbers, company ownership of drives, model number, audit number, and PID.

User Manual's Scope

The goal of this user manual is to entail every detail of this project from architecture, design approach, maintenance, extension, and troubleshooting for the software ("KeepReading").

Software Overview

Architecture Overview

Providing a high-level view of the software's ("KeepReading") architecture, the primary programming language used for ("KeepReading") is Python. Besides Python, there are numerous Python packages used for developing this API ("KeepReading"). Many of the libraries used are licensed either as: BSD License, Apache License 2.0, or MIT License. Some of the most notable libraries that were frequently used are: *pytesseract*, *opencv-python*, *numpy*, and *Flask*. The figure below captures all of the 3rd party libraries used for ("KeepReading").

```
attrs==23.1.0
blinker==1.7.0
certifi==2023.7.22
cffi==1.16.0
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
coloredlogs==15.0.1
contourpy==1.2.0
cycler==0.12.1
dnspython==2.4.2
Flask==3.0.0
flatbuffers==23.5.26
fonttools==4.44.0
gTTS==2.4.0
h5py==3.10.0
humanfriendly==10.0
idna==3.4
imageio==2.32.0
importlib-metadata==6.8.0
importlib-resources==6.1.0
itsdangerous==2.1.2
Jinja2==3.1.2
jsonschema==4.20.0
jsonschema-specifications==2023.11.1
kiwisolver==1.4.5
lazy_loader==0.3
llvmlite==0.41.1
MarkupSafe==2.1.3
matplotlib==3.8.1
mpmath==1.3.0
networkx==3.2.1
numba==0.58.1
numpy==1.26.1
```

```
onnxruntime==1.16.2
opencv-python==4.8.0.76
opencv-python-headless==4.8.1.78
packaging==23.2
Pillow==10.0.0
platformdirs==4.0.0
pooch==1.8.0
protobuf==4.25.1
PyAudio==0.2.14
pycparser==2.21
pygame==2.5.2
PyMatting==1.1.11
pymongo==4.5.0
pyparsing==3.1.1
pyreadline3==3.4.1
pytesseract==0.3.10
python-dateutil==2.8.2
PyYAML==6.0.1
referencing==0.31.0
rembg==2.0.52
requests==2.31.0
rpds-py==0.13.0
scikit-image==0.22.0
scipy==1.11.3
six==1.16.0
srt==3.5.3
sympy==1.12
tifffile==2023.9.26
tqdm==4.66.1
urllib3==2.0.7
vosk==0.3.45
websockets==12.0
Werkzeug==3.0.1
zipp==3.17.0
zxing-cpp==2.1.0
```

Figure 1. List of all packages used with respect to their versions

Key Features

Notable features of this software ("KeepReading"), includes but not limited to:

- Object Character Recognition (image to text).
- Text to Speech.
- Voice Control.
- Database Catalog.
- Drive label analysis and information retrieval.

System Requirements

This software ("KeepReading") will require necessary hardware for successful operation. The main requirement is a camera peripheral with a working Windows operating system machine, preferable Windows 11 by Microsoft. In-addition, for software requirements, see *Figure 1* in Architecture Overview.

For maintainability, it may be advised to use a networked computer to download and update the packages over time in case of a vulnerability. You may need a thumb drive to import the updated files. The machine running the application will need to have python installed.

Getting Started

This section includes step-by-step instructions on getting started with using ("KeepReading"). Before getting started, be sure to have a compatible system as described in System Requirements. You will need Python and Git. To download all of the project files, you will need to clone the project repository listed on GitHub (Repository URL listed in the appendices section).

After cloning the project, you will need to set up the following packages listed in the 'Required Software' section, in addition, you will need to set up your database environment. Finally you can then launch the application.

Required Software

While other/newer versions of these software may work, the following are the only versions guaranteed to work.

- Node.js V20.10.0
- Python V3.11.4
- Tesseract V5.3.3
- MongoDB V7.0.2

Database Setup

In the MongoDB installation wizard, set up a local instance. On Windows, you will need to use your Windows username and password to setup. This is not used for logging into the database, but is used for MongoDB's permissions to setup.

Using command line tools or MongoDB Compass, connect using mongodb://localhost:27017 or a custom connection. Here, you can create a database named keepreading, and a collection named ocrtest (or use a custom name, instructions for making this modification can be found in the Technical Details section)

Python and Node Setup

After installing the above software, there are a few commands that need to be run in order to install the required python and node packages.

```
pip install -r requirements.txt
npm install
```

Following this, the application can be started by first starting the python server, then the node process.

```
python main.py
npm start
```

User Interface

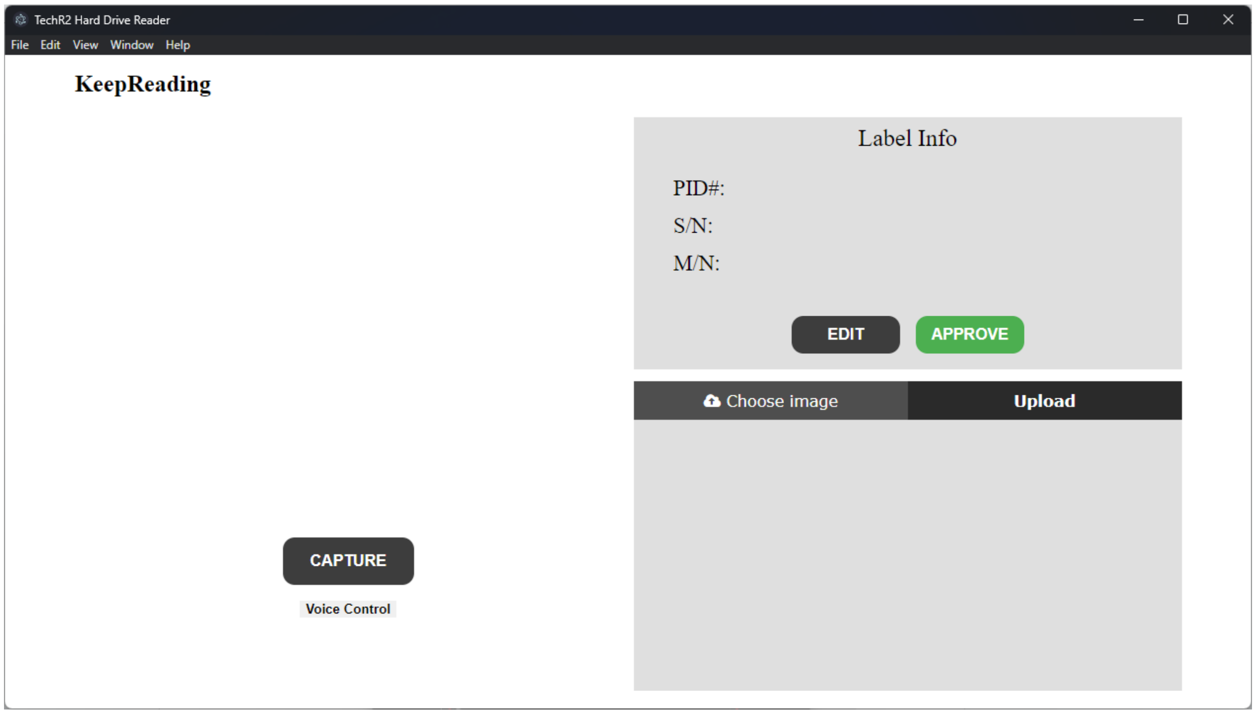

Figure 2. Main Window User Interface

Figure 2, above, displays the main window user interface. This window pops up as soon as the executable file is launched. The green rectangle in Figure 2, represents the feedback from the camera hardware. The 'capture' button simply takes a picture. This picture is then analyzed through object character recognition. This information is then passed to the machine learning models, discussed further in the manual.

Section 'Label Info' represents the information that the system successfully picked up from reading the hard-drive label. This section can display the 'PID', 'Audit #', 'Model Number', and 'Serial Number'. The OCR and machine learning model work hand-in-hand to display the current information in this section.

Beside the main window, the other navigation of this UI is through the buttons `EDIT` and `APPROVE` buttons. These buttons help correct and update the model in case of a mistake made by the machine learning algorithm. In addition to the `Capture` button, we have included an `upload image` for directly uploading an image as well.

Using the Software

Scanning a Live Image

Be sure to have the right equipment listed in the System Requirements section of this manual. With your camera set up, have the hard-drive appear in the camera-feedback section, and press the capture button, shown in *Figure 3*.

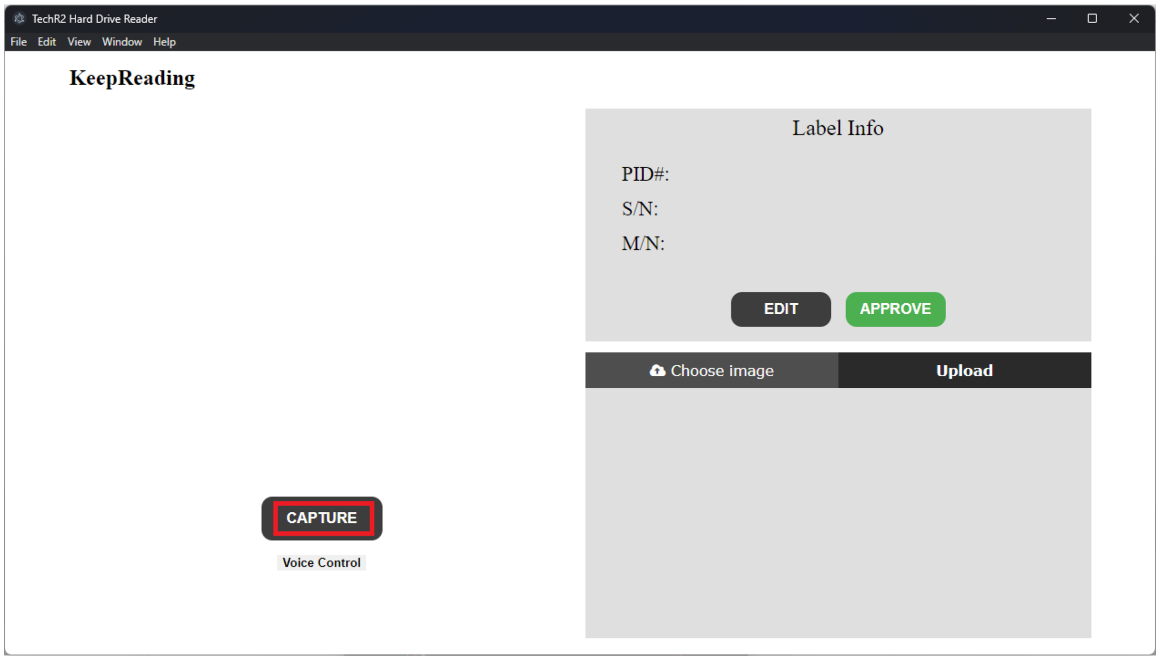

Figure 3. Main window page with 'Capture' button highlighted.

Scanning an Uploaded Image

Click the 'Choose image' button shown on the main window of the application in *Figure 2*, upon pressing this button, a mini window will pop up, shown in *Figure 4*. Locate the desired file, select the desired file, and finally press open.

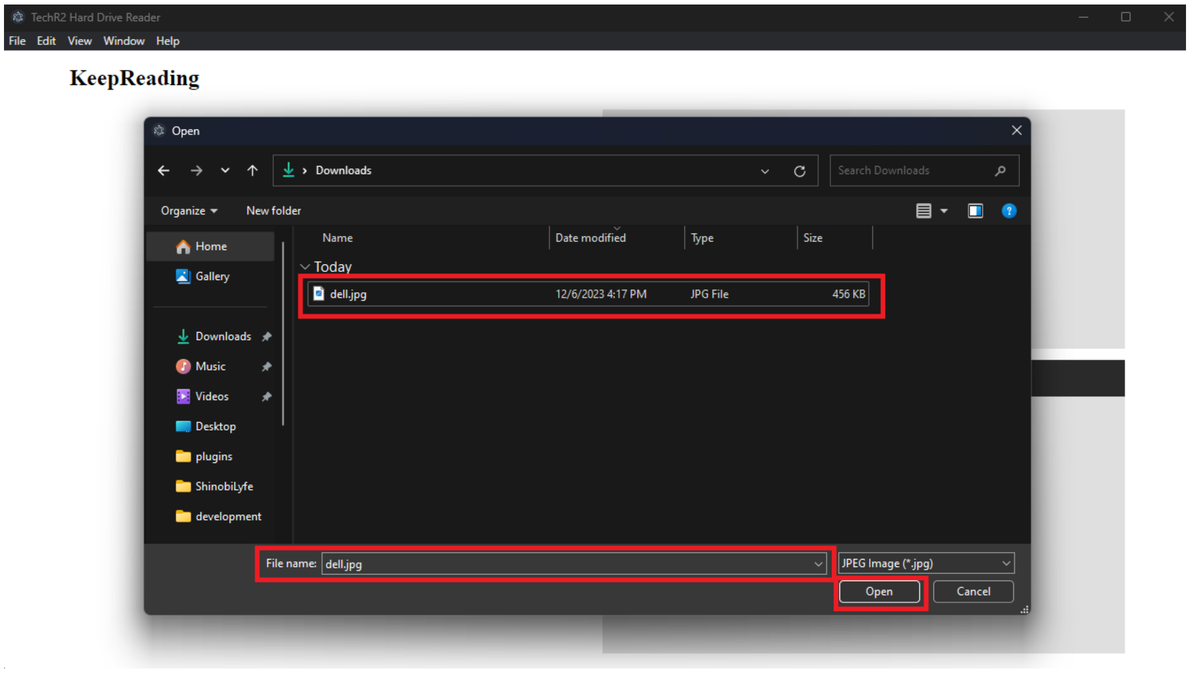

Figure 4. Choosing an image window pop-up

Training and Updating the Model

Prior to running the training script, ensure that there is a folder containing all of the images you wish to train the model on. **WARNING: If there are any images in this folder which are not meant to be used as training data, remove them from this folder–they will be turned into training data by the script.**

1) Open a terminal window and navigate to the directory which contains the script titled "drive_scanner_trainer.py" (this should be in the /lib/ folder within the project root directory).
2) Run the file in the terminal using python with the following arguments:
   a) -i "<*path to your folder containing the training images*>"
   b) -c "<*path to your folder containing a properly formatted csv file*>"
   c) -t "<*name of the folder where you want the model to be stored*>"
3) Wait until the script is complete. Your model will be stored in a new folder with the title supplied to the script in the -t parameter.

```
user> python ./drive_scanner_trainer.py -i
"C:/Path/To/Training/Images/Here" -c
"C:/Path/To/Formatted/CSV/File/Here" -t "training_folder_name"
```

OPTIONAL PARAMETERS:

1) –exclusions: This is used by the training and validation script to automate testing. This should not be used manually.
2) –extension: If the images you wish to use as training are not formatted using JPG, supply the appropriate file extension. For example, if your training images are formatted in PNG, enter the following:

```
user> python ./drive_scanner_trainer.py -i
"C:/Path/To/Training/Images/Here" -c
"C:/Path/To/Formatted/CSV/File/Here" -t "training_folder_name" -e
".png"
```

Testing Model Accuracy

If you add another classification algorithm to the list of models as described in the Technical Details section, you can test the accuracy of the new model by running the "train_and_validate.py" script followed by the "accuracy_check_KNN.py" script. **WARNING: If they already exists, make sure to delete/rename the folder "training_and_validation" and the files "test_results.txt" and "training_results.txt." If these elements exist prior to running the script, there will either be an error or the accuracy script will exhibit weird behavior.**

1) Navigate to the /lib/ folder in the project directory and run the train_and_validate.py script with the path to your training images as the first argument and the path to your formatted csv file with labels as the second argument.

```
user> python ./train_and_validate.py "C:/Path/To/Training/Images"
"C:/Path/To/Formatted/CSV/File"
```

2) Wait for the script to finish executing. Depending on the number of images you wish to train the model on, this may take a while.
3) When the script is finished executing, run the accuracy_check_KNN.py script with the path to your formatted csv file with labels as an argument. The training and test accuracies will be printed to the console.

```
user> python ./accuracy_check_KNN.py "C:/Path/To/Formatted/CSV/File"
```

Troubleshooting

General

- Invalid SOS parameters for sequential JPEG
    - This warning has to do with cv2's image processing and means that the image is not technically a valid JPEG. This does not affect the performance of the software and can be ignored

Camera Recognition

- Camera preview is not being displayed/Wrong camera is being displayed
    - A valid camera must be plugged into the device running the software
    - If there are multiple cameras, the default camera selected is the same as the default windows camera. Either disconnect/disable all cameras other than the one desired for use, or make sure the desired camera is the default

Prediction and Information Retrieval

- Cannot subtract size (X,) from size (Y,)
    - This error is likely caused by an outdated training model which is meant to run using an old configuration of the BOW text file. Try providing a different model when making predictions. If you are using the latest trained model, try training a new model and then specify its location in main.py in the variable "PATH_TO_TRAINED_MODEL."

Technical Details

Retrieving PID

Retrieving the PID from a passed image is completed in the “getPID” method in the “drive_scanner_updated” script. It is passed a list of text processed by the pytesseract OCR and the image to be scanned. Using this information, the function searches for the “PID” string in the list and, if found, returns the string supplied in the entry next to the located “PID” string. If the “PID” string is not found, the image is rotated ninety degrees, rescanned, and passed back into the “getPID” method. This repeats until the “PID” string is found or the image is rotated four times.

Retrieving the PID relies solely on the accuracy of the pytesseract OCR and does not depend on any image classification techniques. The team decided to implement the PID retriever (and *not* the serial number or model number retrievers) in this manner because the PID number is one of the largest and clearest labels on each hard drive. Further, the location of the PID number, unlike the locations of other pieces of information on the drive, is not predictable. The PID label can be placed on multiple different locations on two drives which are otherwise similar, making finding the location of the PID based on the type of drive difficult. Fortunately, the pytesseract OCR employed can pick these numbers out with some regularity due to their size and clarity compared to the rest of the information on the drive.

Retrieving Model Number

Retrieving the model number from a passed image is completed in the “getMOD” method in the “drive_scanner_updated” script. It is passed an image, a list of drive manufacturers, the processed OCR text, the name of the folder holding the model, and an optional “accuracy_mode” parameter. Using this information and a trained model, the image is classified and given a model number and manufacturer.

The model algorithm used for training is located in the models_updated.py script in the class “KNearest.” This class contains a series of methods designed to run the K-nearest neighbors classification algorithm for training and prediction. When the “predict” method is run, the image is converted to the appropriate passed-in data representation which is then compared with every image in the training set (which are stored with the same data representation used for the prediction inside of model.hdf5). Finally, the prediction is output.

In order to efficiently classify images by model number and manufacturer, distinguishing features need to be pulled out of said images. To accomplish this task, the team converts each image into two different data representations: a bag-of-words and a black-and-white histogram. The former is used because each drive differs slightly with respect to the words on the drive. For example, HP drives will have “hewlitt-packard” on the label, while DELL drives will not. This puts a limit on the possible drives that the passed image could be. The black-and-white histogram is used to take the different symbols and patterns on each respective drive type into account. Drive models have a certain structure with certain symbols in specific locations on the drive label–this information is captured by the black-and-white histogram representation. The black-and-white histogram representations of DELL drives of a certain model differs from the representations of DELL drives of a different model, allowing the team to receive more information about a passed-in drive label. The code for the bag-of-words representation can be found in bow.py in the BOW class. The code for the black-and-white histogram representation can be found in data_representation_abstracted_updated.py in the BWHistogram class.

The bag-of-words representation takes the passed OCR text and converts it to a vector containing a series of ones and zeroes. The structure of this vector is determined by the contents in the BOW.txt file such that the first index of the vector corresponds to the first word of the BOW.txt file and each succeeding index corresponds to each succeeding word in the BOW.txt file. When an image is converted to this representation, a 1 is placed in the index where the corresponding word is found (and 0 if the word is not found).

The black-and-white histogram first converts the given image to black-and-white, then produces a binary image using otsu’s method to determine the threshold. Then, a vector containing the sum of the pixels in each column is computed and normalized. Finally, the vector is standardized by removing any entries with a value of zero and then interpolated until the vector is of a specified size.

Retrieving Serial Number

Retrieving the serial number from a passed image is completed in the “getSER” method in the “drive_scanner_updated” script. It is passed with the pre-processed image and the model of the drive predicted from "getMOD". Pyterseearct OCR scans the pre-processed image and searches for "SER" or "SN" in the result from the OCR. If found, it returns the string supplied in the entry next to the located "SER" or "SN" string. However, if the keywords were not found, the image is rotated ninety degrees, rescanned, and passed back into the “getSER” method. This repeats until the "SER" or "SN" keywords are found or the image is rotated four times.

The pre-processed image is done in the "Preprocess" class, where the given image is altered by cropping the background, resizing the cropped image to 1000 x 1200 pixels, transforming it to black-and-white, and cropping the image again to the part where the serial number can be easily identified. Finally, the barcode remover is applied to the image to get a better accuracy for OCR reading.

The dimensions of the cropped image are defined in "crop_details.txt". Since every model of a drive has a similar placement for serial numbers, the team recorded the crop details for each model manually with the model number followed by the dimension. Then, during image pre-processing, the predicted model will determine the crop image dimension.

Database

For database installation and setup, see the README. After installing MongoDB and creating a collection to hold drive label information, go to line 8 in db.py and change “ocrtest” to match the name of your collection.

In the OcrDatabase class, there are properties that store the client, database, and the primary collection used. The API class contains wrappers for pymongo methods that are/can be used for the application, named find, find_one, insert, and find_most_recent.

The layout of the documents stored in the database is the OcrResult class, which contains properties for all the possible information on the drive label as well as important information for analytics. These expected JSON layout of data in the database is as follows:

```
{
      "pid": int,
      "auditId": string,
      "manufacturer": string,
      "model": string,
      "serialNumber": string,
      "confidence": string,
      "userReported": boolean,
"datetime": datetime
}
```

Development & Extension

Project Structure & Extension

This project can be extended by cloning the main repository listed privately on GitHub. The main coding structure for this project all starts with ‘main.py’. This file is the Flask server that runs in the background while also handling the user’s camera device for capturing and scanning an image. Many of the backend files are located inside the ‘lib’ folder. The ‘lib’ folder contains almost all of the API calls for backend functionality.

Another import file, ‘main.js’, handles the javascript segment of this project. Since the backend server is a Flask server, for front end, javascript and electron are utilized. For the front end, folders such as ‘node_module’, ‘static’, and ‘templates’ are used.

Adding a New Data Representation

To add another data representation to the predictor and trainer, add a class which inherits the TrainingRepresentation class in the data_representation_abstracted_updated.py file. This class should have one method titled “represent_data” which takes three parameters (itself, the image, and a dictionary of parameters), and returns a one-dimensional vector. To use this data representation when making predictions and training, see the following instructions for adding a new model. Whenever adding a model/adjusting the representations of existing models, change the data representation/training representation fields to the appropriate representation instead of adding a new model.

Adding a New Model

To add another model to the predictor, add a class which inherits the CModel() parent class in the models_updated.py file. Implement a load, predict, and train method. Then, add the following lines to the load_models() method in the drive_scanner_updated.py file:

```
algorithm_name = AlgorithmName()
algorithm_name.load(trained_data_loc,
MODELNAMEABBREVIATION_DATAREPRESENTATIONNAMEABBREVIATION_Dict_loc,
ConstantNames.DATAREPRESENTATIONNAMEABBREVIATION)
parameters = {} #dictionary should contain the parameters necessary
for your model
model_params.append([algorithm_name, parameters.copy()])
```

Then, add the following lines in the drive_scanner_updated.py file in the getMOD function:

```
model = trained_models[N] #where trained_models[N] holds the model
loaded in the above step
model_drive_N = predict(image, model[0], TrainingRepresentation(),
model[1])
Model_predictions["model_name"] = model_drive_N
```

In the models_updated.py file, edit the MODEL_WEIGHTS dictionary in the ModelUtils() class to the following:

```
MODEL_WEIGHTS = {"model_name": X} #where X is an integer that corresponds to the weight you want the model to have. The larger the number, the more likely its prediction will be prioritized over the other models.
```

Then, add the following lines in the models_updated.py file in the get_overall_prediction() function:

```
Model_Name = model_dict["model_name"]
for label in Model_Name:
            results[label] += ModelUtils.MODEL_WEIGHTS["model_name"]
```

Inside the "if ocrFlag" if-then block, add the following two lines:

```
for prediction in Model_Name:
            list_build[prediction] = strcomp(takeout_brand(prediction, drive_types), model_dict["OCR"].strip())
```

To add a model to the trainer, call the train_on method at the bottom of the drive_scanner_trainer.py file and pass training_data, ModelName(), TrainingRepresentation(), "MODELNAMEABBREVIATION", and "DATAREPRESENTATIONNAMEABBREVIATION" where the capitalized text in quotes can be changed to whatever you wish (but ensure that the occurrence of these names in the first step of this model addition guide is substituted with the text you decide to use in this step). The following line should be added:

```
train_on(training_data, ModelName(), TrainingRepresentation(), "MODELNAMEABBREVIATION", "DATAREPRESENTATIONNAMEABBREVIATION")
```

Appendices

- “Formatted CSV File”
    - This refers to the csv file necessary for training. All images in the folder used for training should be titled with the PID number and followed by an image extension. The csv file provided to the training scripts should be formatted as follows:

AuditId	PID	Manufacturer	Serial Number	Description
…	…	…	…	…

- “project repository”
    - This refers to the codebase uploaded to GitHub.com. With the power of Git & GitHub, all of the project files along with their respective commits can be found at https://github.com/DevFarid/KeepReading.

Revision History

- 11/20/23 — Version bump from 1.26.56 → 1.27.56
- 11/28/23 — User Interface update, 1.27.56 → 1.28.57
- 12/06/23 — Voice control, loading animation, 1.28.57 → 1.30.58