GROUP 3

FINAL PROJECT COMPLETE PAPERWORK

BURIAL RECORD OPTICAL CHARACTER RECOGNITION

EAI6010 APPLICATIONS OF ARTIFICIAL INTELLIGENCE

CRN: 72021

Done By

Anish Nitin Somaiah, NUID: 001894328, somaiah.a@husky.neu.edu

Pratibha Sridhar Iyengar, NUID: 001476161 , iyengar.p@husky.neu.edu

Ankit Arvind Khatavkar, NUID: 001476187, khatavkar.a@husky.neu.edu

Siddhant Sanjiv Sapte, NUID: 001494280 , sapte.s@husky.neu.edu

Srinivasu Narahari, NUID: 001824939, narahari.s@husky.neu.edu

Instructor: Mr. Sergiy Shevchenko



**Introduction**

One of the most significant applications of Artificial Intelligence is Optical Character Recognition. This phenomenon is defined as the process of scanning text into an image from an image/form/paper and translating the images obtained into a system manipulatable form. This is highly useful in applications where an archive of form data has to be manipulated and features in these forms are to be extracted and stored into a data frame. On an extension such OCR can be used in financial institutions to analyze cheques, credit cards, in healthcare providing systems to maintain patient databases from forms and so on.

**Project Objective**

Our project objective is similar to the process mentioned above. We are provided a set of burial forms. Our objective is to perform Optical Character Recognition over these forms and extract necessary features such as Name, Age and so on from the image given.

Initially, we are to classify the images based on their layout such as printed or handwritten images. After this classification, the computer performs feature extraction from the image and the model’s efficiency depends on how clear we input the image to the computer so that the features are properly extracted. For this purpose, we perform certain process on the images such as cropping, normalization, Gaussian filtering and so on depending upon the type of the image obtained as output from the classification.

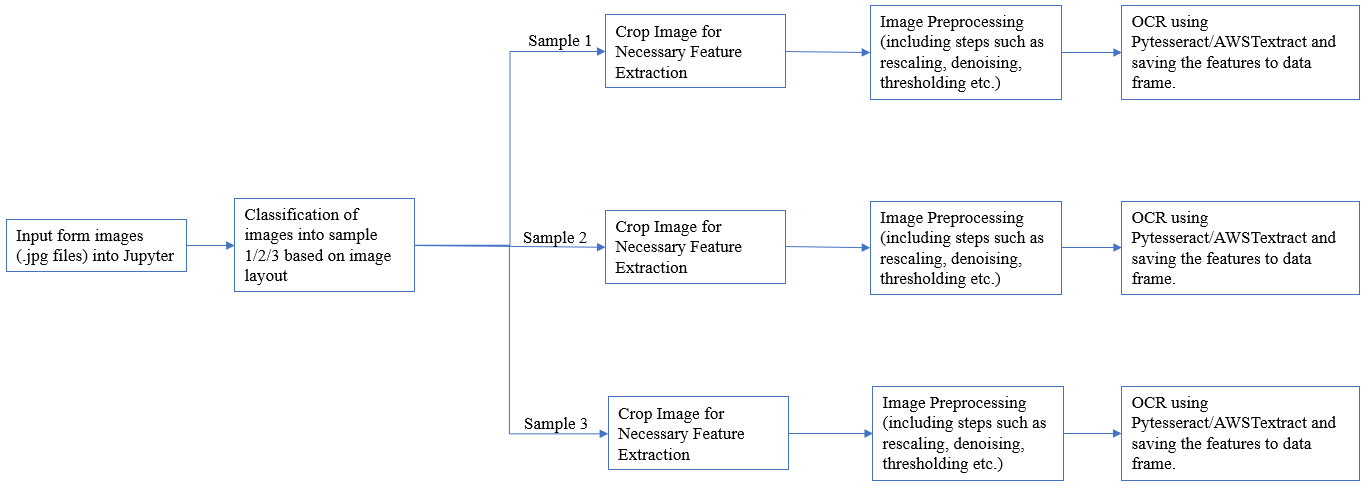
**Dataset Description**

Our dataset is obtained from an organization named “Plural Point” that has burial forms of 120 deceased patients provided to us as .jpg images. There are broadly 3 types of image categories based on image layout such as forms with column sections, forms with row sections and forms with empty sections which we use as base for the classification and image processing.

The forms contain essential attributes such as

* **Name**: name of the deceased person.
* **Age**: age at which person deceased.
* **Date of Burial**: date of burial.
* **LOT-TER**: Section of burial ground of the deceased person.

**Project Process Workflow**



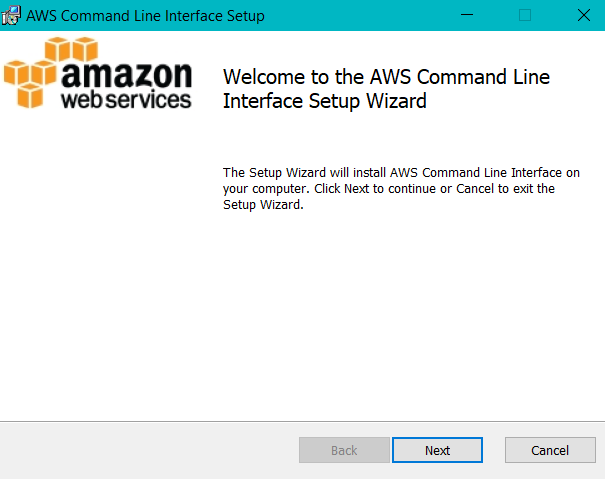
*Figure 1: Workflow of OCR process on burial form images*

* Input images from source to Jupyter Notebook using path directory.
* Perform classification of the images to identify the type of image based on layout into sample 1 or 2 or 3 as mentioned above.
* Crop image to take only necessary sections of the image to perform Optical Character Recognition.
* Perform Image Preprocessing steps such as Converting to Grayscale, Scaling the image, Denoising, Applying Dilation & Erosion and Thresholding onto the cropped image to improve the quality of the input image for the computer to efficiently extract features from the image.
* Finally perform OCR using AWSTextract or Pytesseract function on the image and store the necessary features of the image into a data frame which is presented to the user with the consolidated features from the image.

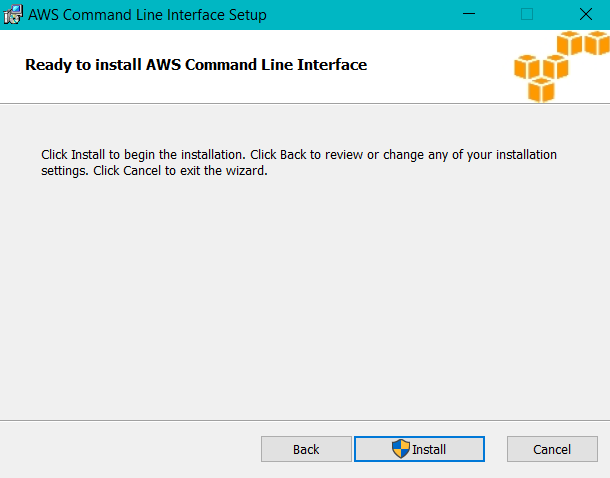
**STEPS AND ANALYSIS**

**Step 1: Integrate AWS Texteract to Local Computer**

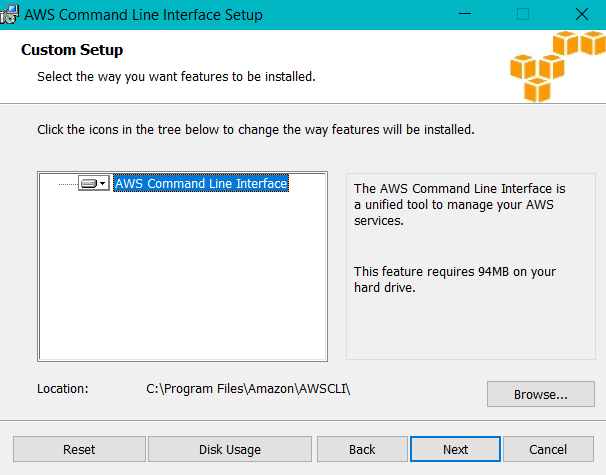
* In order to implement AWS Texteract, AWS command line interface or AWS CL2 needs to be installed into our local computer using the link from AWS tutorial page.
* Once the installation is complete, we check the aws version installed through command prompt and install boto3 package using Anaconda Command Prompt
* After successful installation, we configure our AWS environment into our local computer using the command prompt and using the access key ID for the IAM user that has been created under our AWS console.



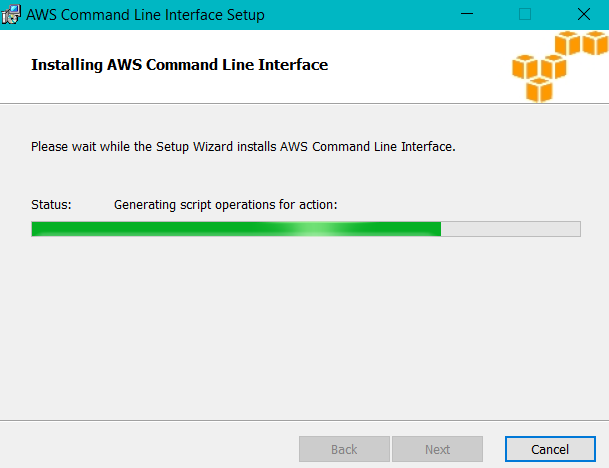
*Figure 2: AWS CL2 installation page 1*



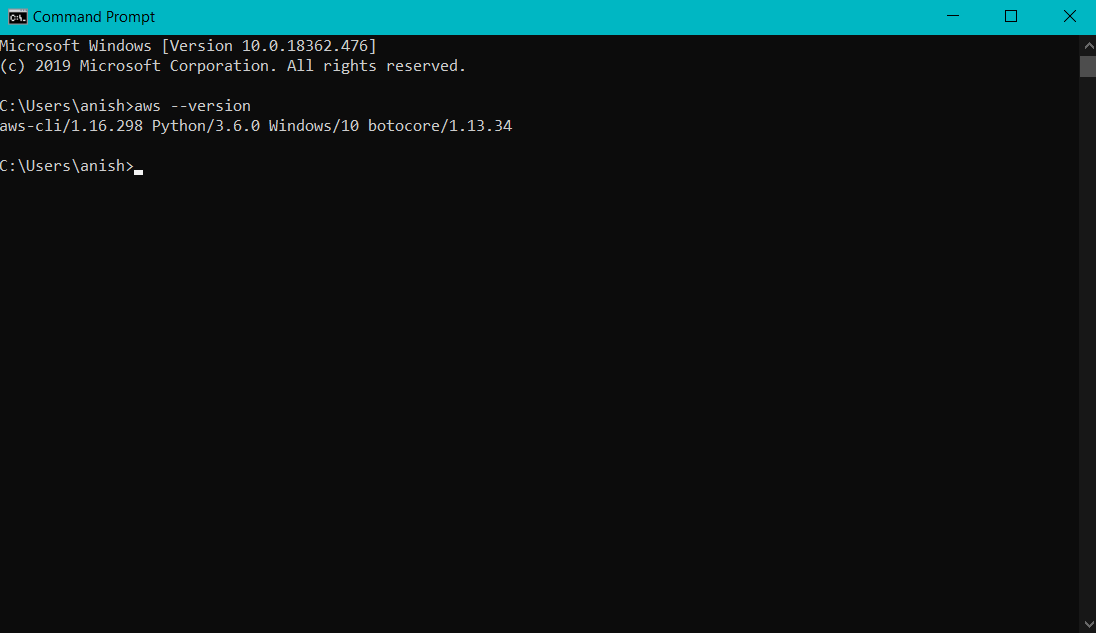
*Figure 3: AWS CL2 installation wizard*



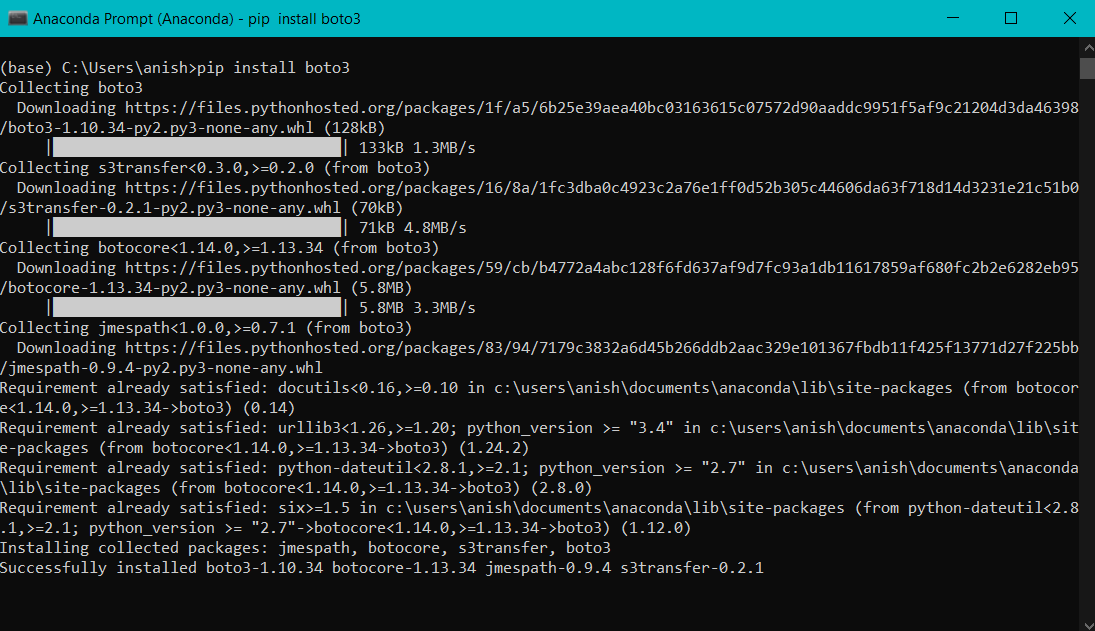
*Figure 4: AWS Command line Interface installation path specification*



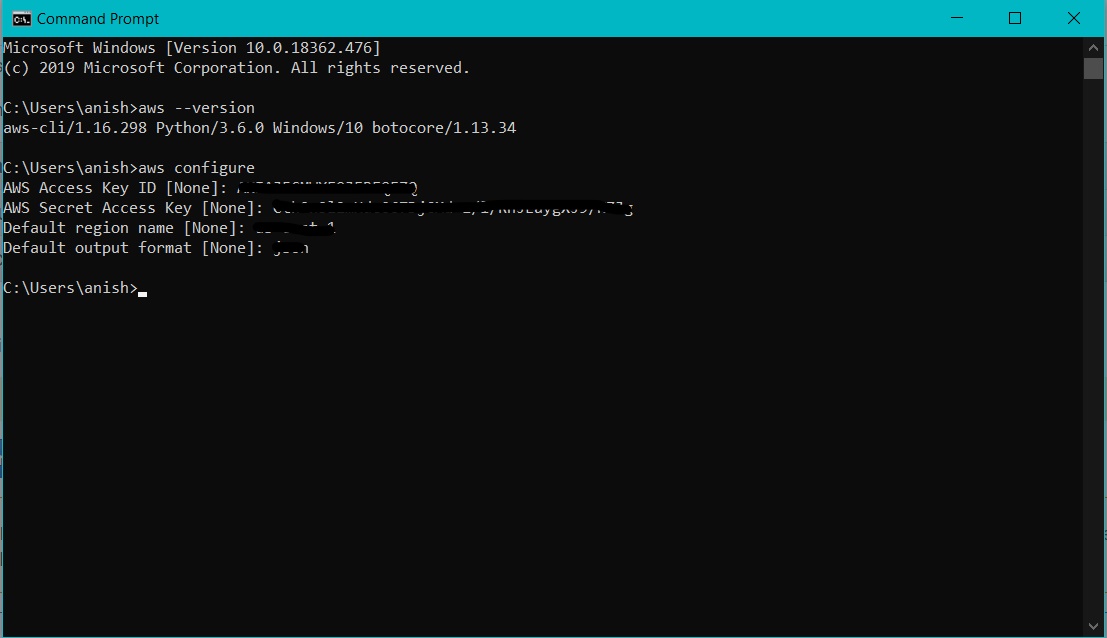
*Figure 5: Installation Status*



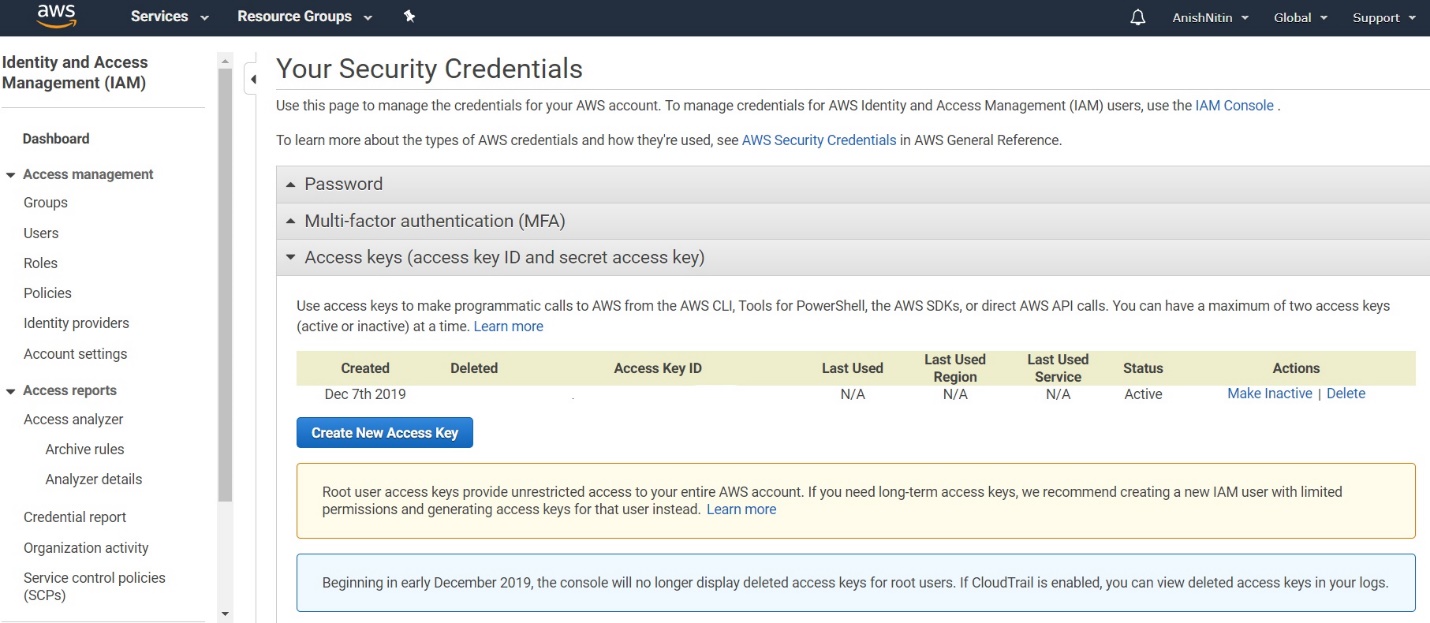
*Figure 6: Command Line Prompt AWS version check*



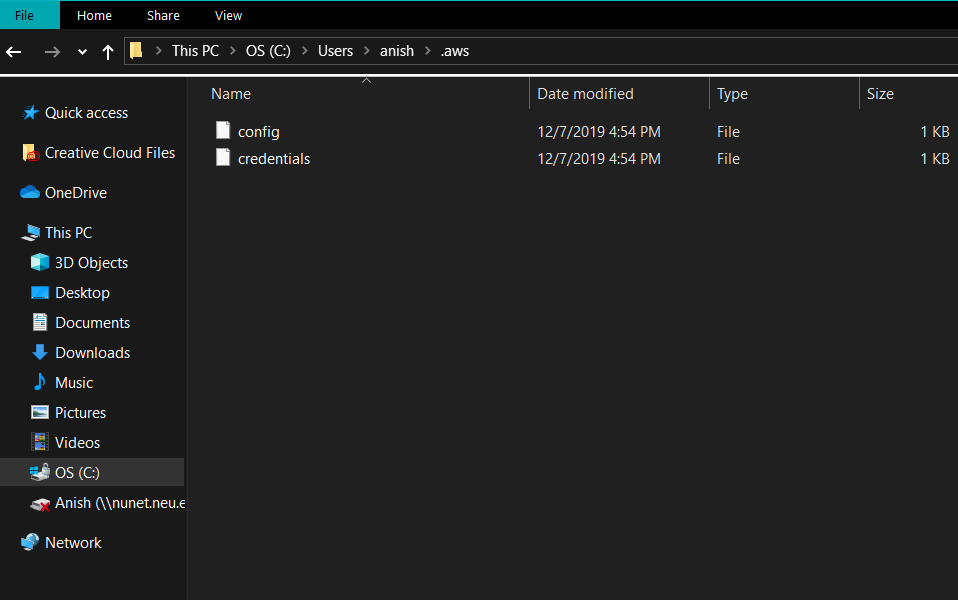
*Figure 7: Boto3 package installation*



*Figure 8: AWS configuration using Access Key ID*



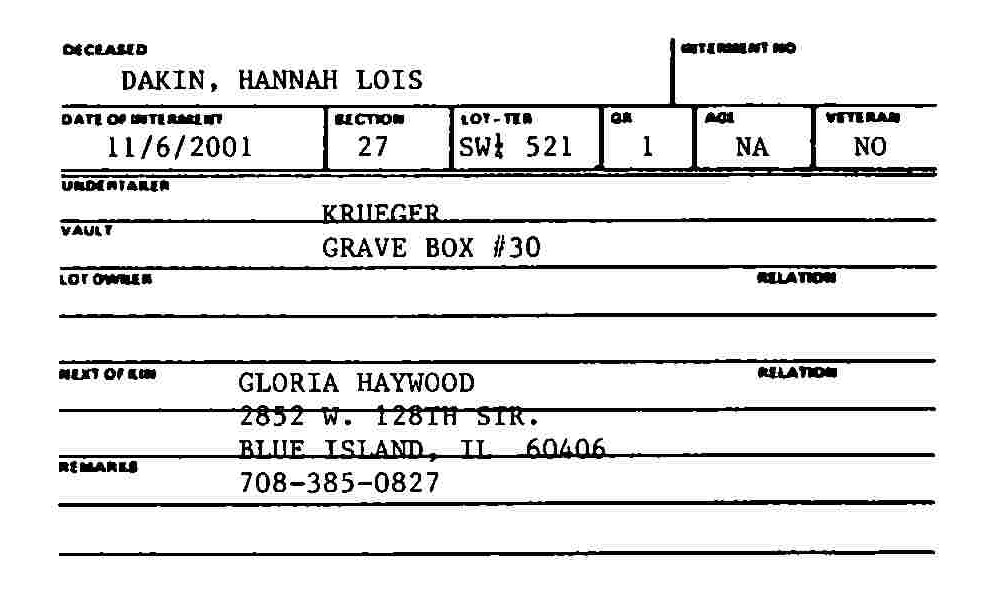
*Figure 9: Access Key ID generation under IAM page in AWS console*



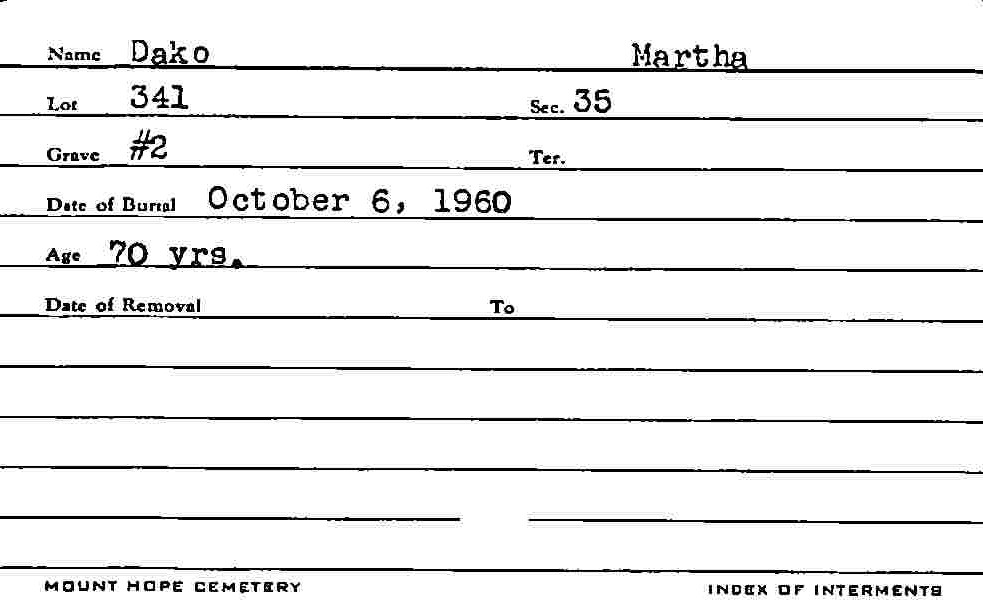
*Figure 10: Config and Credential properly configured in Local Machine*

**Step 2: Setting up necessary libraries for CNN**

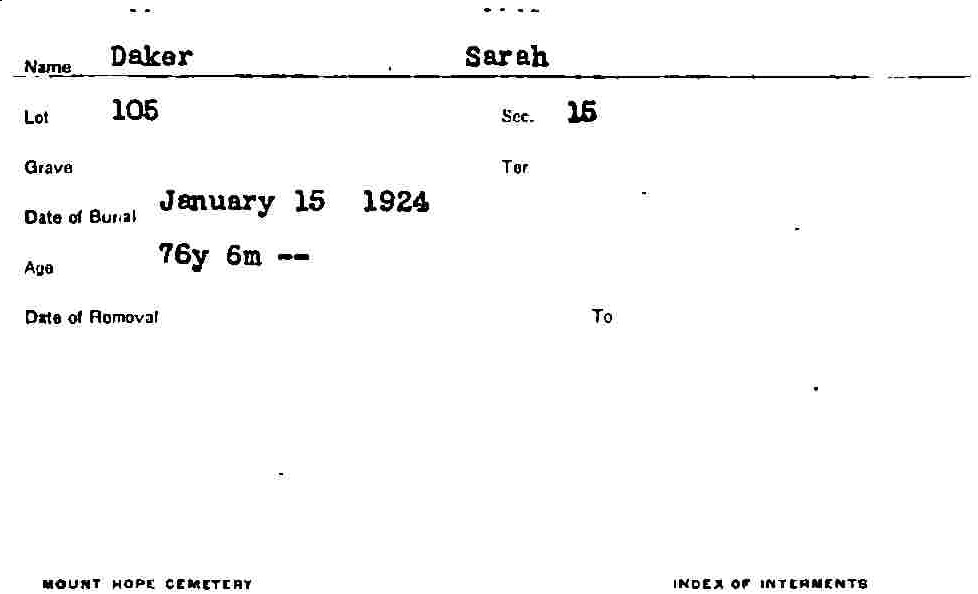
* After setting up AWS Texteract, we now proceed to perform image classification using Convolutional Neural Networks to classify images into Sample 1,2 and 3 respectively based on the image layout.
* Sample 1: Image layout with column sections

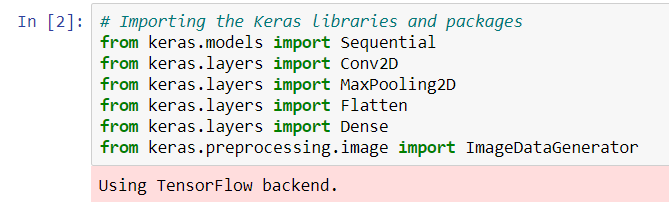


* Sample 2: Image layout with row sections



* Sample 3: Image layout with blank sections



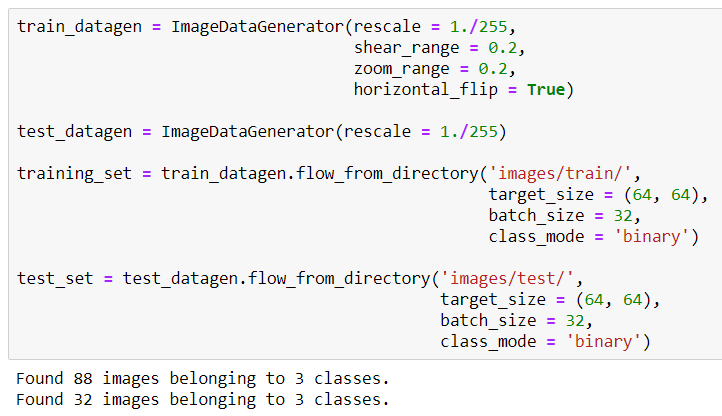


*Figure 11: Library import for CNN*

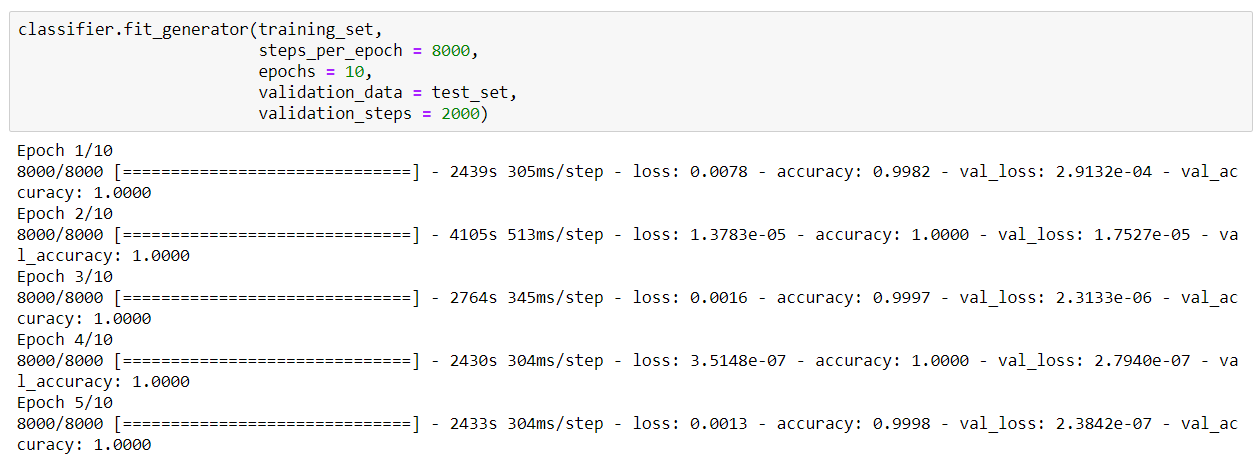
**Step 3: CNN initialization, Image preprocessing for CNN, CNN implementation fit & Accuracy Measurement**



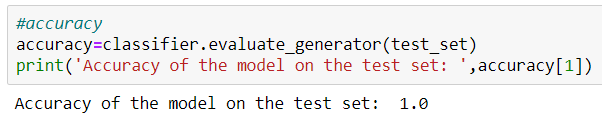
*Figure 12: Step by Step initialization of CNN*



*Figure 13: Image preprocessing under training and test data sets respectively for CNN model.*

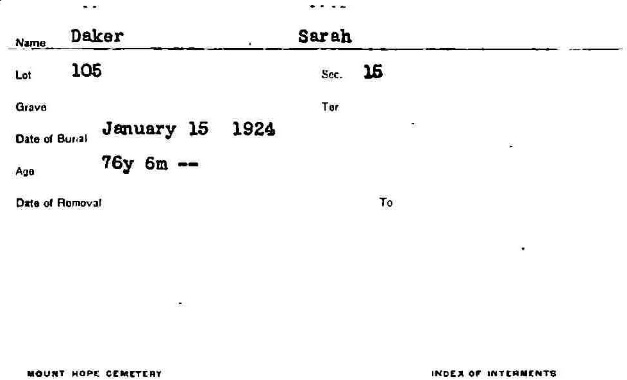


*Figure 14: CNN model fit with 10 epochs*



*Figure 15: Accuracy of classification of images using CNN on test data set.*

**From the above accuracy we can conclude that our CNN classifies our images properly into the 3 sample sets based on the image layout. This is a go sign for our OCR initialization.**



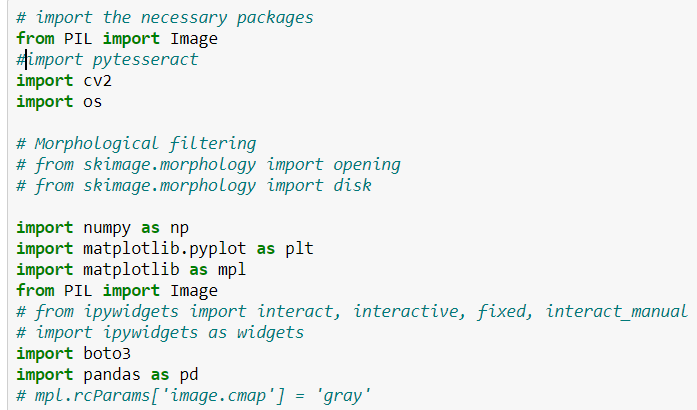


*Figure 16: CNN output testing*

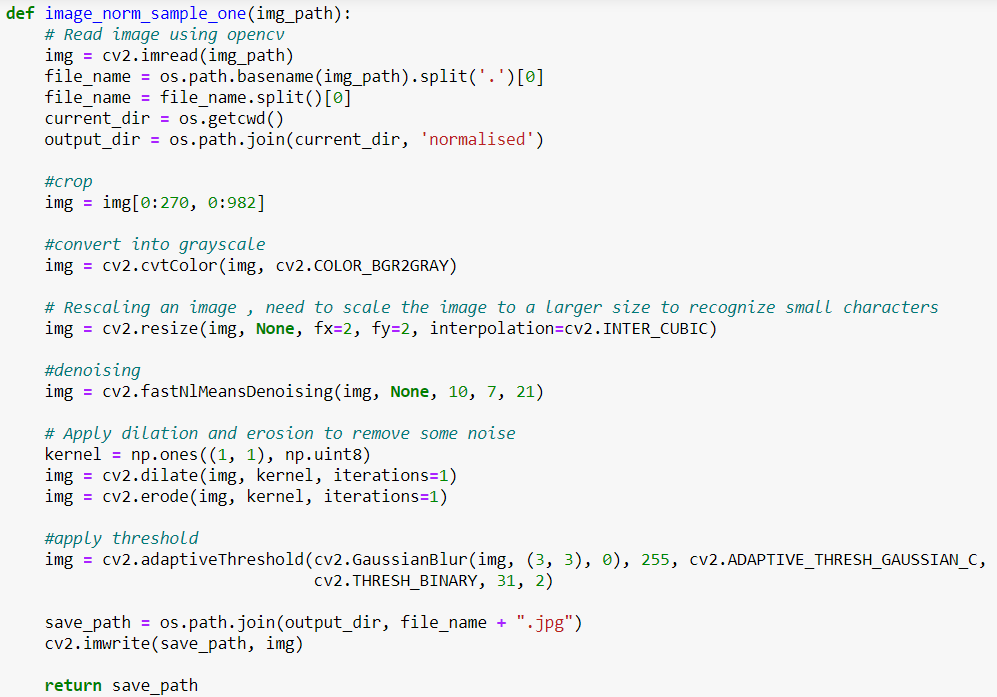
**Here we can see that an input image is correctly classifies as sample 3 by the CNN.**

**Step 4: Function Definitions for Image Preprocessing and AWS Texteract**

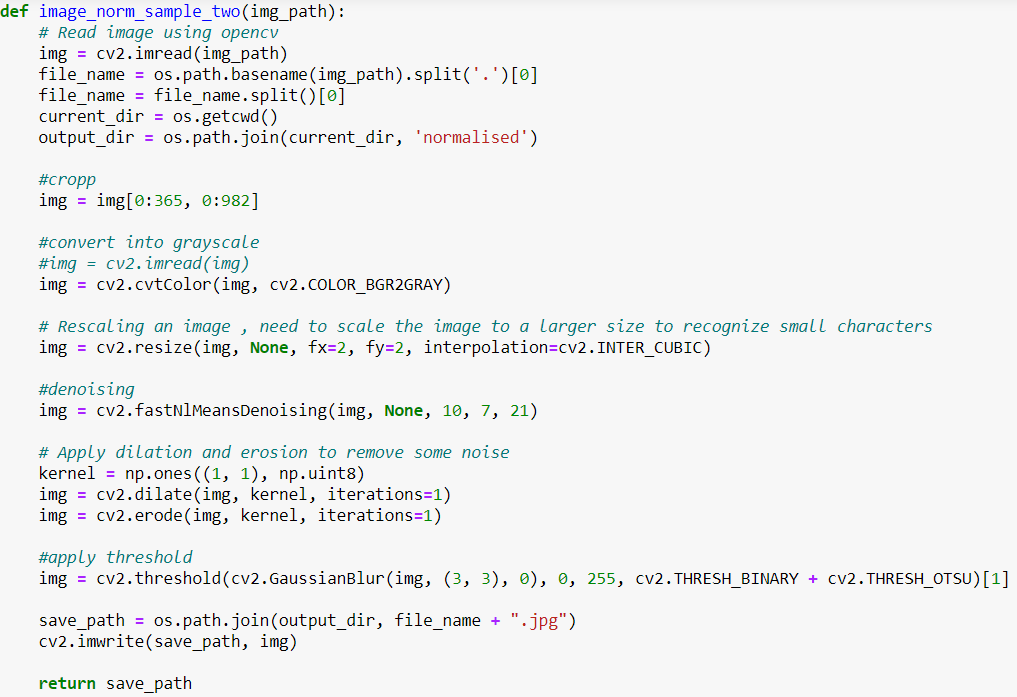
In this step we declare functions to implement certain preprocessing steps to our image such as crop, convert to gray scale, rescale image, denoising, applying dilation & erosion and application of threshold, to improve the quality of our image to enable our machine to perform better optical character recognition.



*Figure 17: Importing necessary libraries for the function implementation*

**

*Figure 18: Preprocessing function for sample 1*

**

*Figure 19: Preprocessing Function for sample 2*

**

*Figure 20: Preprocessing function for sample 3*

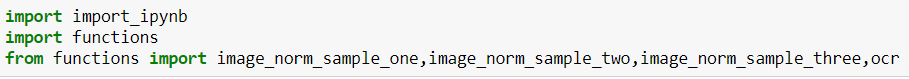
**We can see from the above 3 screenshots that the preprocessing is different for different types of samples such as metric difference in the size of thresholding or method of thresholding etc.**



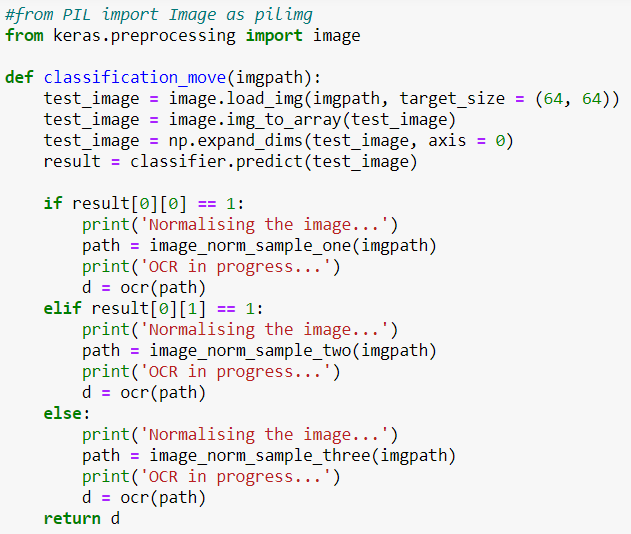
*Figure 21: Function definition for Optical Character Recognition*

**In this function we call the AWS Texteract client that we installed locally as mentioned in Step 1 and perform the optical character recognition. We return d from this function which is the data dictionary of the characters recognized which is then converted into a data frame and then exported as a .csv file in the forthcoming processes.**

**Step 5: Calling functions for image preprocessing based on sample and Implementation of OCR, Output OCR data frame to .csv file as output.**



*Figure 22: import functions from function notebook to the main OCR Jupyter notebook*



*Figure 22: Calling function for Normalizing image and OCR implementation based on image sample type*



*Figure 23: Main function that inputs image using path for Normalizing & OCR and function that outputs OCR components as a data frame to a .csv file in a defined path for user to view output.*

**Note: In order for the entire OCR system to be automated image paths are given to the function which can be implemented for any test image in that specified path.**

**Work Flow:**

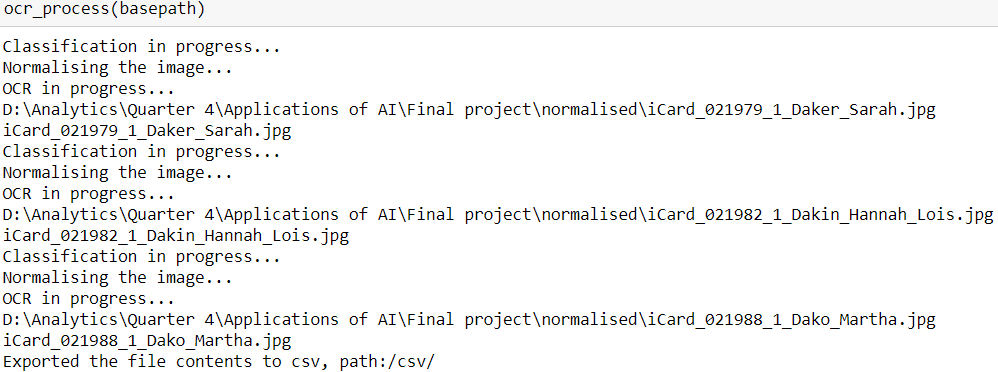
* We have created 2 Jupyter Notebooks

1. **Functions.ipynb :** notebook that contains definition of functions to preprocess images based on the respective samples.
2. **EAI6010\_group3\_final\_project.ipynb :** notebook that contains our CNN model, commands to call functions to implement normalizing and OCR on the images and the main function to input stream images into our system and store output character data frame as .csv file.

* Initially we classify the images into samples 1,2 and 3 based on the layout of the image using CNN. Sample descriptions are mentioned above with image reference.
* We input our image through the main function ocr\_process by keeping them in a folder called ocr\_test and passing its path to the system for streaming the image into our classification and ocr implementation notebook.
* We call the classification\_move function which is the function that will decide the type of the image that is input into the computer.
* 2 functions i.e. one for normalizing the image and one for implementation of ocr are called from the secondary notebook which is nested into this function to perform the necessary operations based on the sample number of the input test image.
* The image pre processing steps that we implement here are as follows:

1. Cropping an image: to remove unwanted parts of the image containing insignificant information
2. Grayscale conversion: to reduce color complexity that the image gives while processing.
3. Rescaling: to improve visibility of smaller characters, the image is rescaled to a larger value.
4. Denoising: reduce noise in image output from scanners/cameras to retain image features such as edges, sharp structures etc.
5. Dilation and erosion: Addition and removal of pixels to boundaries to further reduce noise in image
6. Thresholding: Gaussian blurring is done in this method to remove outlier pixels and it acts as a low pass filter to provide definite boundaries for the image that is considered for OCR.

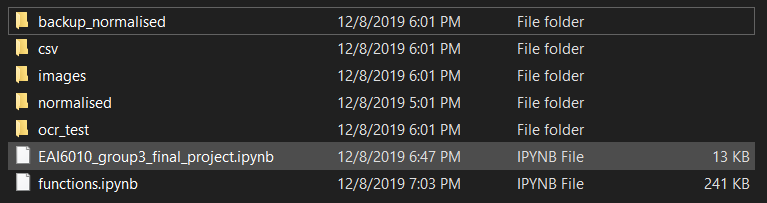
* This function returns a data dictionary which is called back to the main ocr\_process where the recognized characters are converted to a data frame and printed into a csv format for the user to view the results.

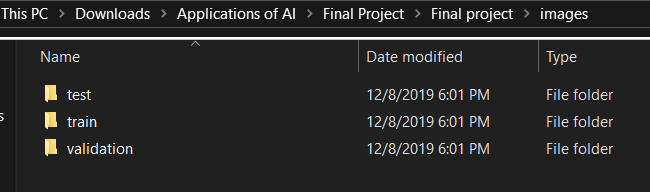


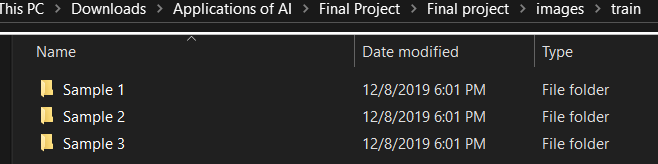
*Figure 24: main function implementation cycle in Jupyter Notebook.*

**Important Note:**

**For the entire project to work flawlessly, the files are to be stored in a format in our local system in the following format.**

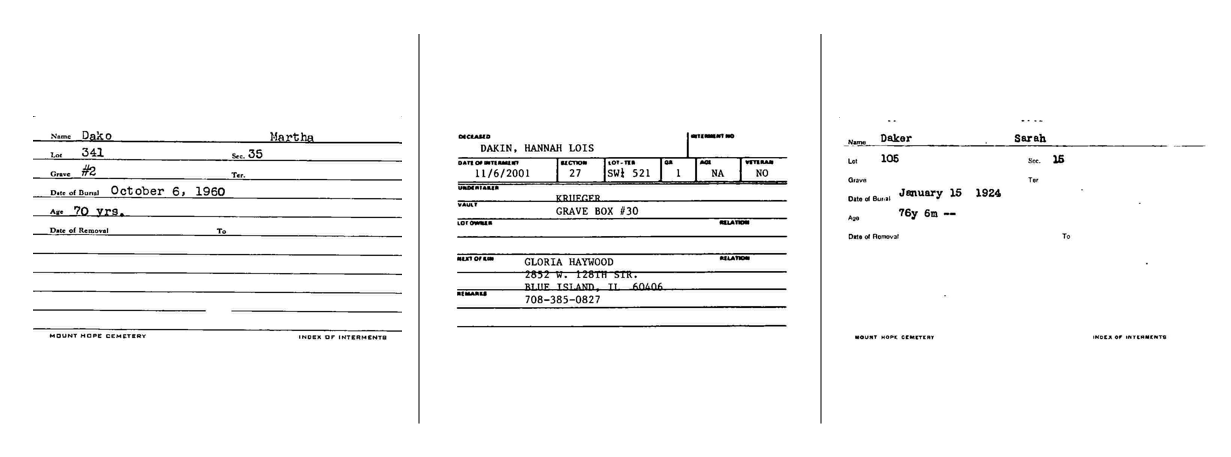




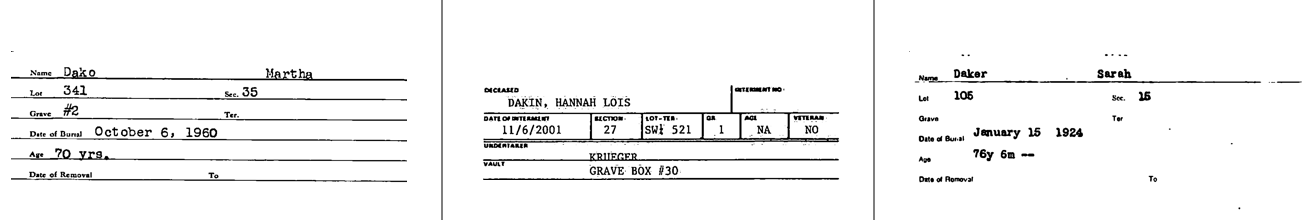


* **Ocr\_test** folder contains the images that we stream into the machine to perform classification and OCR
* **Normalized** folder contains the images that have been normalized after preprocessing
* **Csv** folder contains the output data frame of the features present in the input forms.
* **Images** folder contains 3 subfolders **Train, Test, Validation**
* Each folder contains 3 subfolders **Sample 1, Sample 2 & Sample 3.**
  + Sample 1: Image layout with column sections
  + Sample 2: Image layout with row sections
  + Sample 3: Image layout with blank sections

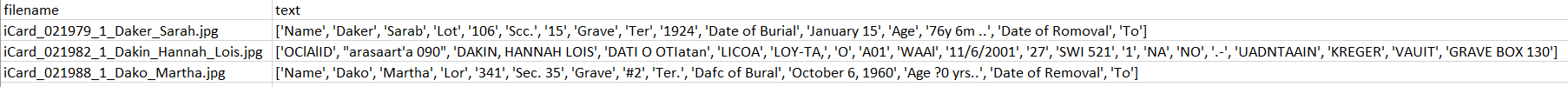
**OUTPUT:**



***Set of Input Images***



***Set of Normalized images***

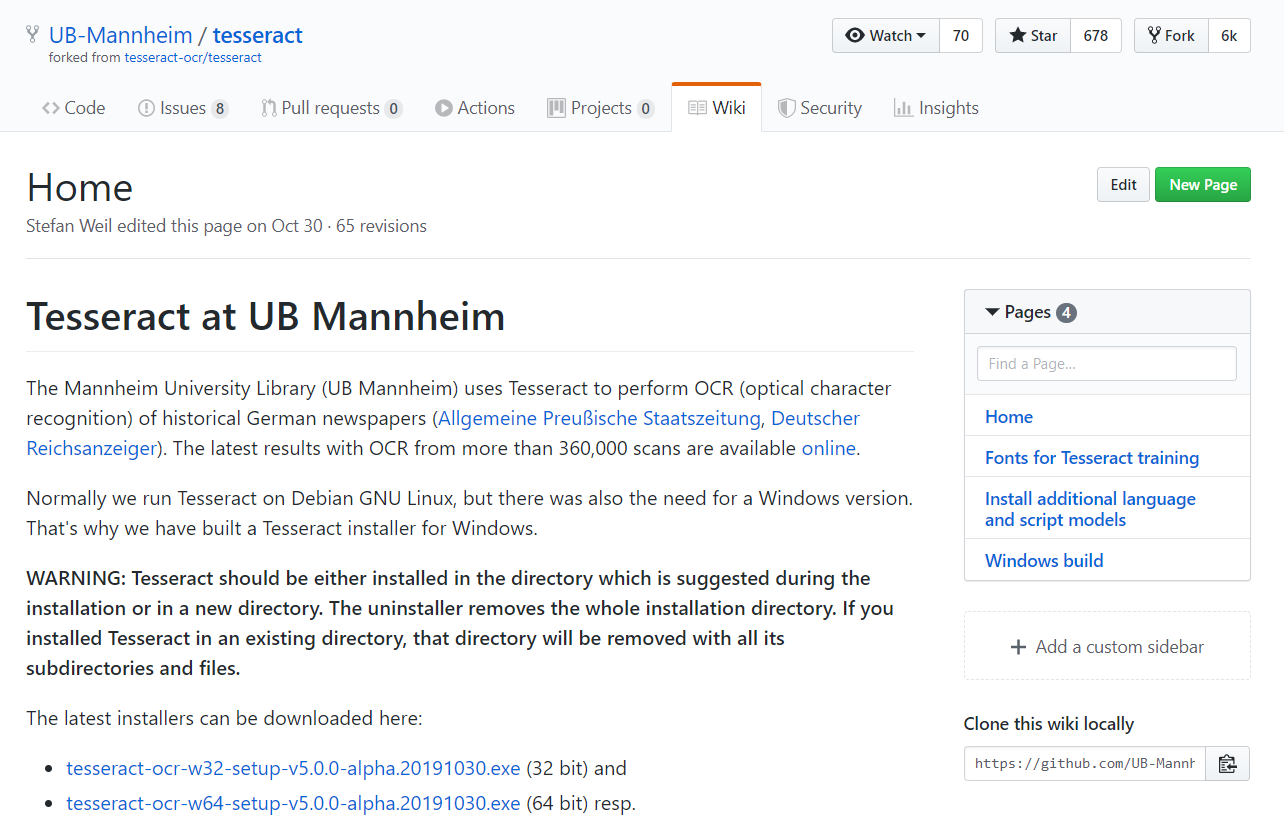


***.csv Output***

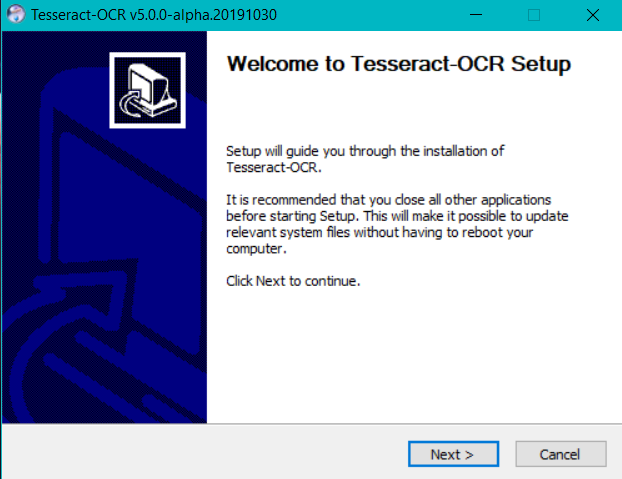
**COMPARISON OF OCR PERFORMANCE BETWEEN AWS TEXTERACT AND PYTESSERACT.**

So far above we have seen the performance of AWS Texteract. Now let us analyze the performance of Pytesseract.

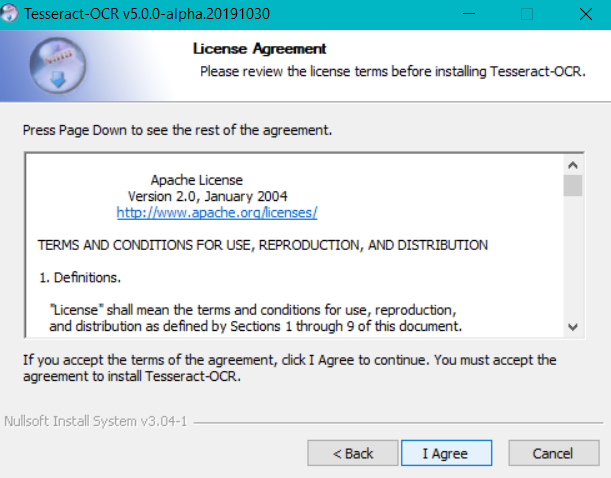
**STEP 1: Installation of Pytesseract**



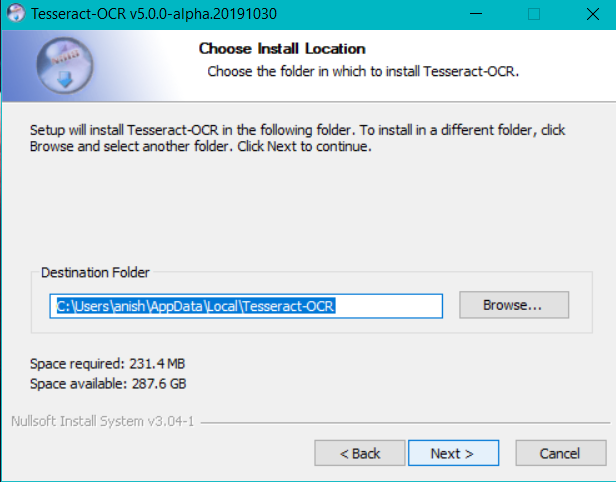
*Figure 25: Git hub repository for exe file download*



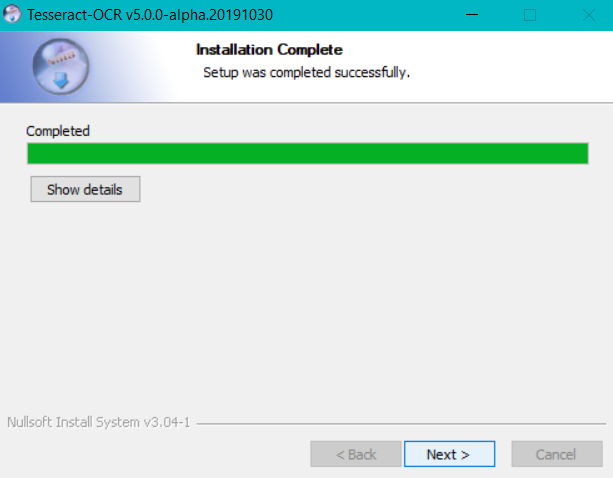
*Figure 26: Pytesseract installation wizard*



*Figure 27: Installation License Agreement*



*Figure 28: Destination path specification*

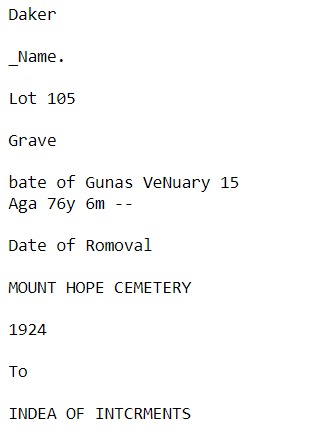


*Figure 29: Pytesseract Installation Complete*

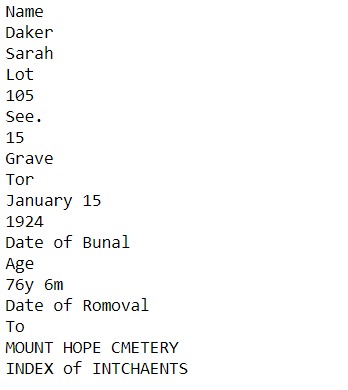
**Step 2: Pytesseract OCR implementation in Jupyter**



*Figure 30: OCR using Pytesseract*



*Figure 31: Output of Pytesseract*



*Figure 32: Output of AWS Texteract*

**From the above 2 outputs we find that AWS Texteract output is much more refined and interpretable than Pytesseract. Furthermore, the character recognition is done pretty well in AWS Texteract compared to Pytesseract. Hence the Amazon tool was used to perform Optical Character Recognition for the given data set.**

**REFERENCE**

* **AWS(n.d),** *Installing the AWS CLI,* Documentation Retrieved from <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html>
* **UB-Mannheim(n.d),** *Tesseract at UB Mannheim,* Documentation Retrieved from <https://github.com/UB-Mannheim/tesseract/wiki>
* **Prince Canuma(Oct 2018),** *Image Pre-Processing,* Retrieved from <https://towardsdatascience.com/image-pre-processing-c1aec0be3edf>
* **TensorScience.com(Dec 2018),** *Optical Character Recognition (OCR) with Python and Tesseract 4,* Retrieved from <https://www.tensorscience.com/ocr/optical-character-recognition-ocr-with-python-and-tesseract-4-an-introduction>
* **Zain Sajjad(Sept 2019),** *Amazon Textract — Going beyond optical character recognition (OCR),* Retrieved from <https://heartbeat.fritz.ai/amazon-textract-going-beyond-optical-character-recognition-ocr-202cf270a0ba>