Documentazione OCR CLI

[1 Introduzione 3](#_Toc72402034)

[1.1 Informazioni sul progetto 3](#_Toc72402035)

[1.2 Abstract 3](#_Toc72402036)

[1.3 Scopo 3](#_Toc72402037)

[2 Analisi 4](#_Toc72402038)

[2.1 Analisi del dominio 4](#_Toc72402039)

[2.2 Analisi e specifica dei requisiti 4](#_Toc72402040)

[2.3 Use case 7](#_Toc72402041)

[2.4 Pianificazione 8](#_Toc72402042)

[2.5 Analisi dei mezzi 9](#_Toc72402043)

[2.5.1 Software 9](#_Toc72402044)

[2.5.2 Hardware 10](#_Toc72402045)

[3 Progettazione 11](#_Toc72402046)

[3.1 Design dell’architettura del sistema 11](#_Toc72402047)

[3.2 Design dei dati e database 13](#_Toc72402048)

[3.3 Design delle interfacce 13](#_Toc72402049)

[4 Implementazione 14](#_Toc72402050)

[4.1 Funzionamento generale 14](#_Toc72402051)

[4.2 Dipendenze e requisiti 14](#_Toc72402052)

[4.2.1 Script per installare le dipendenze 14](#_Toc72402053)

[4.3 Virtual environment 14](#_Toc72402054)

[4.4 Struttura 15](#_Toc72402055)

[4.5 Sviluppo 15](#_Toc72402056)

[4.5.1 Flow 15](#_Toc72402057)

[4.5.2 Logging 15](#_Toc72402058)

[4.5.3 Parametri 16](#_Toc72402059)

[4.5.4 Scan immagine 17](#_Toc72402060)

[4.5.5 Gestione output 17](#_Toc72402061)

[4.5.6 Statistiche d’esecuzione 18](#_Toc72402062)

[5 Test 19](#_Toc72402063)

[5.1 Protocollo di test 19](#_Toc72402064)

[5.1.1 Test funzionali 19](#_Toc72402065)

[5.1.2 Test non funzionali 22](#_Toc72402066)

[5.2 Risultati test 23](#_Toc72402067)

[5.3 Mancanze/limitazioni conosciute 23](#_Toc72402068)

[6 Consuntivo 24](#_Toc72402069)

[7 Conclusioni 26](#_Toc72402070)

[7.1 Sviluppi futuri 26](#_Toc72402071)

[7.2 Considerazioni personali 26](#_Toc72402072)

[7.2.1 In generale 26](#_Toc72402073)

[7.2.2 Considerazioni personali Viktorija 26](#_Toc72402074)

[7.2.3 Considerazioni personali Thaisa 26](#_Toc72402075)

[8 Sitografia 27](#_Toc72402076)

[9 Allegati 28](#_Toc72402077)

[Figura 1 – Schema casi d’uso 7](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402078)

[Figura 2 - Gantt preventivo 8](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402079)

[Figura 3 - Parte iniziale del flowchart 11](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402080)

[Figura 4 - Flowchart Source 11](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402081)

[Figura 5 - Flowchart OCR 12](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402082)

[Figura 6 - Flowchart Output 12](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402083)

[Figura 7 - Flowchart Stats 13](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402084)

[Figura 8 - Gantt consuntivo 24](file:///E:\TerzoAnno\306%20+%20Progetti\OCR%20miscoservice\Documenti\Documentazione_ocr_cli.docx#_Toc72402085)

# Introduzione

## Informazioni sul progetto

|  |  |
| --- | --- |
| **Responsabile progetto** | Geo Petrini |
| **Autori** | Thaisa De Torre, Viktorija Tilevska |
| **Inizio** | 14.01.2021 |
| **Consegna** | 20.05.2021 |

## Abstract

*Optical Character Recognition (OCR) is the process of extracting text from an image. The main purpose of an OCR is to make editable documents from image files. The OCR Command Line Interface (CLI) application can extract the text from an image and return it as a text file. Although it requires basic command line knowledge it is easy to use. The only thing the user is required to do is pass the image’s path and execute the command and almost immediately the application will return a text file containing the text found in the image. In addition, it is possible to specify the language of the extracted text, the destination path, where the output file will be saved, and the name of the output file. If those are not specified, the program takes the default values.*

## Scopo

Lo scopo del progetto è di creare un OCR Command Line Interface. L’OCR CLI è uno strumento a linea di comando in grado di scannerizzare il testo dalle immagini passate come argomenti e ritornarlo in un file di testo. L’applicativo accetta dei parametri opzionali per configurare la lingua (italiano o inglese) e se mostrare le statistiche di esecuzione, è possibile anche specificare il percorso di destinazione e il nome del file di output. L’OCR accetta solamente immagini in formato PNG oppure JPEG/JPG ma può accettare più immagini come input che raggruppa poi in un solo file di output.

# Analisi

## Analisi del dominio

Questo applicativo è orientato verso persone che hanno bisogno di avere il contenuto di un’immagine in testo facile da modificare. Visto che l’applicativo funziona da linea di comando, gli utenti devono avere una conoscenza minima di come funziona la linea di comando.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Acquisizione di immagini in formato PNG o JPG/JPEG |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’OCR deve accettare solo immagini RNG e JPG/JPEG |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Rilevamento del testo dall’immagine con un algoritmo OCR |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L'OCR deve rilevare il testo correttamente dall'immagine |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Output del testo come TXT |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Ci deve essere un file TXT che contiene il testo rilevato |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Dev’essere bilingue |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Il contenuto può essere interpretato in 2 lingue: Italiano e Inglese |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Upload file multipli con output singolo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’elaborato di più file di input viene ritornato come output singolo.  file.png 🡪 file1.txt file.png file.jpg 🡪 file.txt  Anche con mask: \*.\* 🡪 file.txt |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Statistiche scansioni |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Per ogni esecuzione dev’essere prodotta una statistica con: la quantità di parole rilevate e il tempo di elaborazione |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Visualizzazione dati statistici tramite parametro opzionale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | I dati statistici vengono visualizzati solamente se abilitati tramite parametro |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Guida utilizzo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Visualizzare una guida di utilizzo tramite parametro (--help). La guida viene visualizzata automaticamente in caso di parametri assenti |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Utilità (applicazione) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il lavoro deve poter essere utilizzato in produzione nell’azienda |

|  |  |
| --- | --- |
| **ID: REQ-010** | |
| **Nome** | Gestione degli errori |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | eventuali errori sono identificati e gestiti tramite i mezzi adeguati |

|  |  |
| --- | --- |
| **ID: REQ-011** | |
| **Nome** | Registro eventi/Logging |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L'elaborazione dei dati corretta o errata viene registrata in un file di registro. Al fine di permettere un’interpretazione corretta del registro, i dati pertinenti devono essere memorizzati nel formato appropriato e raggruppati in modo utile. |

|  |  |
| --- | --- |
| **ID: REQ-012** | |
| **Nome** | Tutte le dipendenze del software devono essere incluse |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Eventuali librerie esterne devono essere incluse e facilmente installabili |

|  |  |
| --- | --- |
| **ID: REQ-013** | |
| **Nome** | Gestione delle eccezioni (batch) |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il programma deve essere in grado di funzionare senza supervisioni e di reagire alle potenziali situazioni secondo dei processi predefiniti. In caso d’errore, il programma effettua le azioni definite conformemente alla specifica, scrive i dati nel file di log in modo da permettere un’analisi della situazione che ha portato all’interruzione. |

|  |  |
| --- | --- |
| **ID: REQ-014** | |
| **Nome** | Attendibilità dei dati inseriti dall’utente |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | I campi di immissione sono contrassegnati in modo chiaro e vengono rivisti. Plausibilità: In caso di errore di digitazione, l'utente viene aiutato da indicazioni concrete e il campo corrispondente viene attivato/ evidenziato. |

|  |  |
| --- | --- |
| **ID: REQ-015** | |
| **Nome** | Organizzazione del programma |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il programma è stato strutturato in maniera intelligente senza procedure e funzioni e sotto-procedure/funzioni ridondanti. |

|  |  |
| --- | --- |
| **ID: REQ-016** | |
| **Nome** | Utilizzo di diagrammi di flusso |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | I diagrammi di flusso rappresentano lo svolgimento completo del programma. È rappresentato in modo chiaro, facilmente leggibile e contiene anche dei commenti pertinenti. |

## Macintosh HD:Users:Guest:Desktop:Screen Shot 2021-05-18 at 11.20.26.pngUse case

Figura 1 – Schema casi d’uso

Il funzionamento di questo programma è semplice. L’utente inserisce un’immagine nell’OCR, l’immagine viene scannerizzata e all’utente gli viene restituito un file con il testo scannerizzato.

## Pianificazione

Figura 2 - Gantt preventivo

Il modello di sviluppo che abbiamo deciso di utilizzare è il modello a cascata (waterfall). C'è lo sviluppo a fasi sequenziali. Come si può vedere dal diagramma di Gantt, la fine di una fase e l'inizio dell'altra fase.

Abbiamo scelto questo modello perché secondo noi era il modello più efficacie per il progetto. Volevamo lavorare in un modo semplice dove dedichiamo tempo a una fase alla volta e dopo averla finita facciamo il test di solo quella fase e così via. In questo modo pensavamo di poter svolgere il lavoro in un modo organizzato senza avere tanti errori e senza dover tornare indietro e fare tutti i test di nuovo.

Per fare questo progetto avevamo a disposizione circa 5 mesi (dal 14 gennaio 2021 fino al 20 maggio 2021). Il lavoro l'abbiamo diviso nella fase di analisi, pianificazione, implementazione, test, e alla fine il tempo per la documentazione e il diario.

Abbiamo pensato di dedicare 1 lezione alle analisi, 1 lezione alla pianificazione, il resto delle lezioni all’implementazione e i test e alla fine un po’ di tempo ogni lezione per scrivere il diario e la documentazione.

Nella fase di analisi c'è l'attività di comprensione dei requisiti. Questa attività consiste nel analizzare il quaderno dei compiti e costruire i requisiti necessari per lo svolgimento del progetto.

La seguente fase è la fase della pianificazione. In questa fase pensavamo di fare il diagramma di Gantt preventivo, i diagrammi di flusso e di pianificare l'architettura del codice.

La prossima fase è quella dell'implementazione composta da tante attività necessarie per lo svolgimento del progetto.

Dopo l'implementazione c'è la fase dei test dove vengono fatti tutti i test. Nel nostro caso la fase d'implementazione e la fase di test vengono svolti contemporaneamente, ovvero, dopo un'attività d'implementazione viene eseguito il suo rispettivo test.

## Analisi dei mezzi

### Software

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome del software** | **Versione** | **Utilizzo** **nel** **progetto** | **Collegamento** |
| Oracle VM VirtualBox | 6.1.16 | Software usato per creare le macchine virtuali | https://www.virtualbox.org/ |
| GitHub | 2.17.1 | Piattaforma che abbiamo utilizzato per gestire il progetto e sincronizzare il lavoro | https://github.com/ |
| GitHub Desktop | 2.6.3 | Interfaccia grafica che semplifica la gestione dei repository di GitHub | https://desktop.github.com/ |
| VisualStudio Code [extension Python] | 1.52.1 | Editor di testo utilizzato per sviluppare il programma in Python | https://code.visualstudio.com/ |
| Python | 3.9.0 | Linguaggio di programmazione | https://www.python.org/ |
| Pytesseract | 0.3.7 | È uno strumento OCR per Python | https://pypi.org/project/pytesseract/ |
| Tesseract OCR | 4.1.1 | Lo strumento OCR usato nel progetto | https://tesseract-ocr.github.io/ |
| Pillow | 8.1.0 | Libreria di Python per l’apertura, la manipolazione e il salvataggio immagini. | https://pillow.readthedocs.io/en/stable/ |
| Draw.io | 14.4.9 | Software per creare dei diagrammi di flusso. | https://app.diagrams.net/ |
| Microsoft Project Professional 2016 | 2016 | Software per creare il diagramma di Gantt. | https://www.microsoft.com/en-us/microsoft-365/project/project-management-software |
| PowerPoint 2016 | 2016 | Software per creare la presentazione. | https://www.microsoft.com/en-us/microsoft-365/powerpoint?ms.officeurl=powerpoint&rtc=1 |

### Hardware

Questo progetto è stato realizzato sui computer forniti dalla scuola.

# Progettazione

## Design dell’architettura del sistema

Il programma è composto da quattro classi diverse.

Una classe sarà la classe principale che conterrà il metodo main. Un'altra classe si occuperà della lettura delle immagini e della scrittura del file di testo. La terza classe servirà a fare le statistiche e l'ultima, a creare e scrivere i file di log.

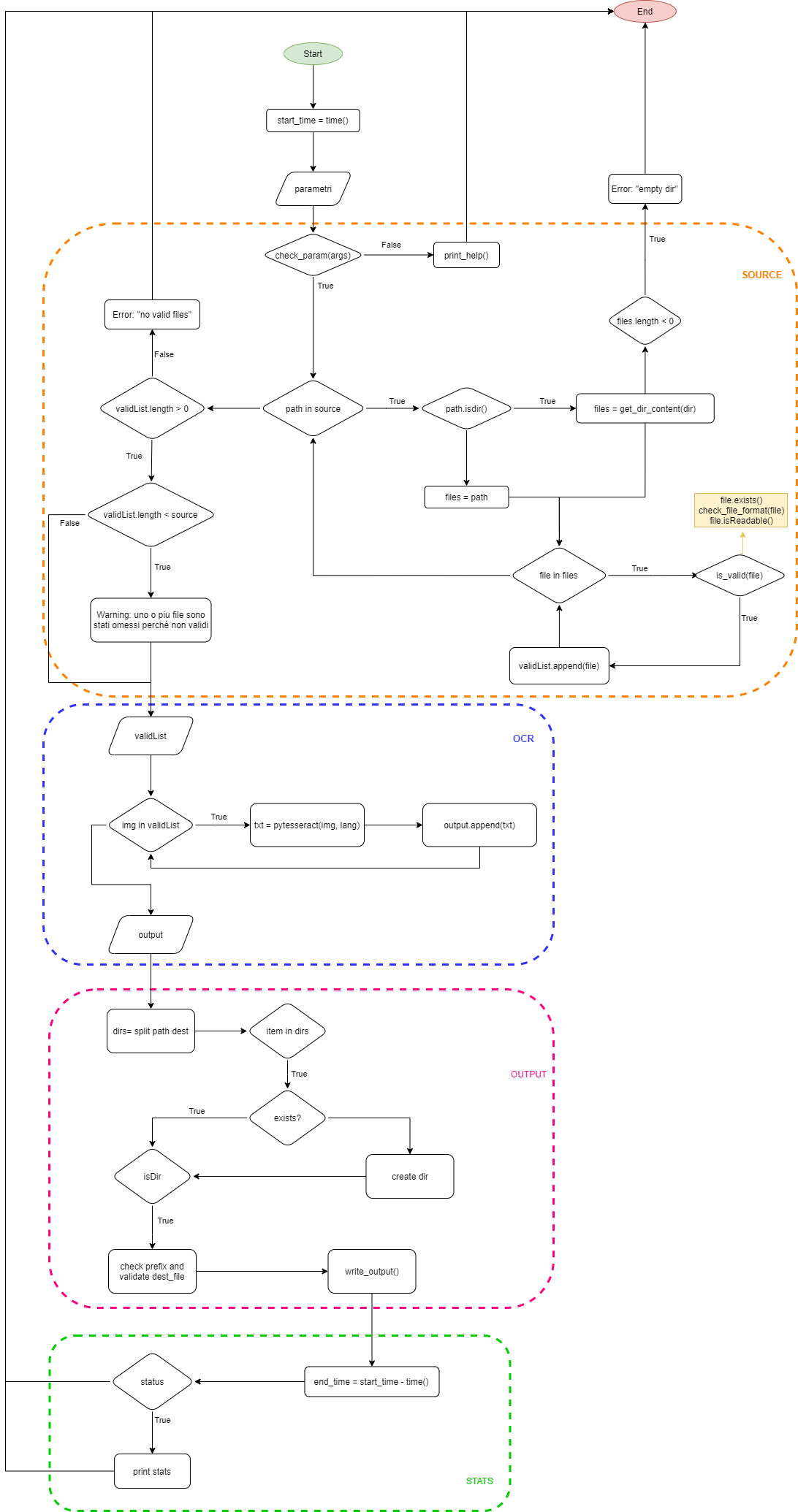
* + 1. **Diagramma di flusso** 

Figura 3 - Parte iniziale del flowchart

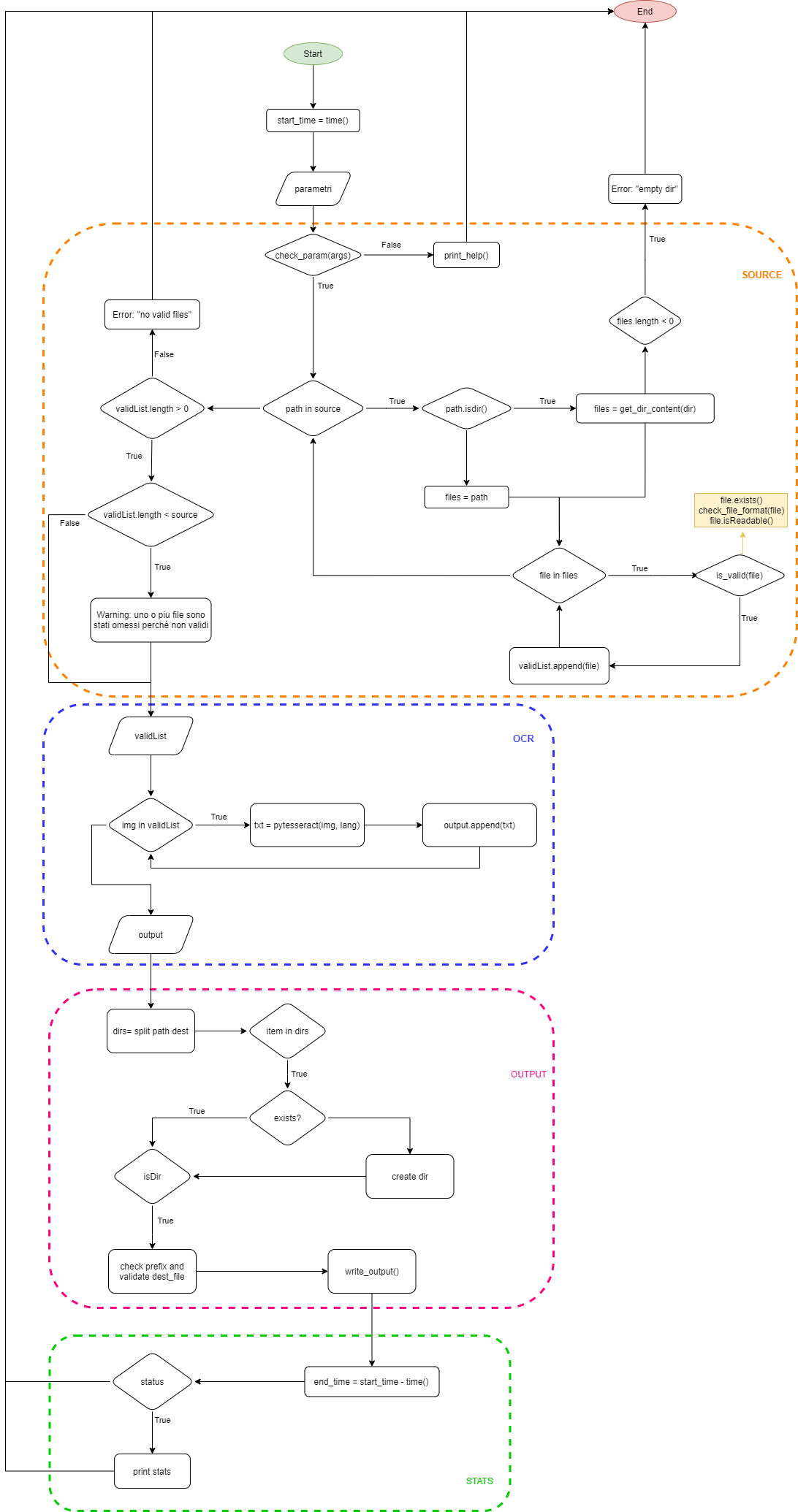
All'avvio del programma, viene salvato il tempo di esecuzione in una variabile. La stessa variabile ci servirà per calcolare il tempo che il programma ci ha impiegato a scannerizzare tutte le foto. Dopo di questo vengono controllati i parametri inseriti dall'utente.

Figura 4 - Flowchart Source

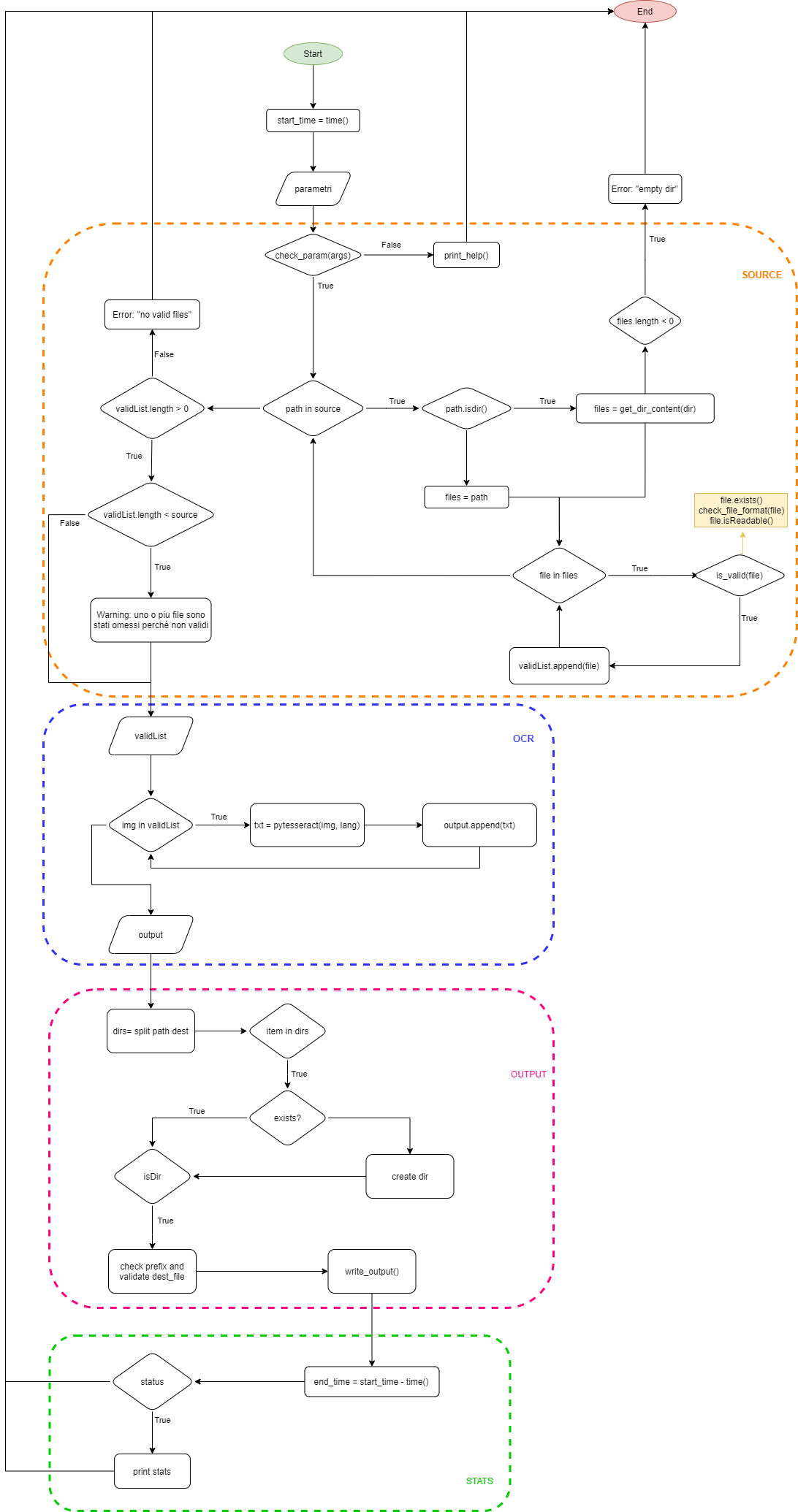
Se i parametri non sono accettabili, viene stampata la guida d'utilizzo e il programma termina, mentre se i parametri sono accettabili si controlla il percorso dell’immagine inserita come argomento. Viene controllato se il percorso è una directory. Se si tratta di una directory, tutti i file nella cartella vengono estratti e aggiunti alla lista di file. Se non è una directory viene controllato se è un file e se è sì, lo stesso viene aggiunto alla lista dei file. Da questo punto si controlla la validità di tutti i file nella lista, ovvero, viene controllato se i file esistono, sono accessibili in lettura e sono nel formato accettato. I file validi vengono inseriti in un'altra lista mentre tutti gli altri file vengono ignorati. L'utente viene avvisato che ci sono stati dei file ignorati.

Figura 5 - Flowchart OCR

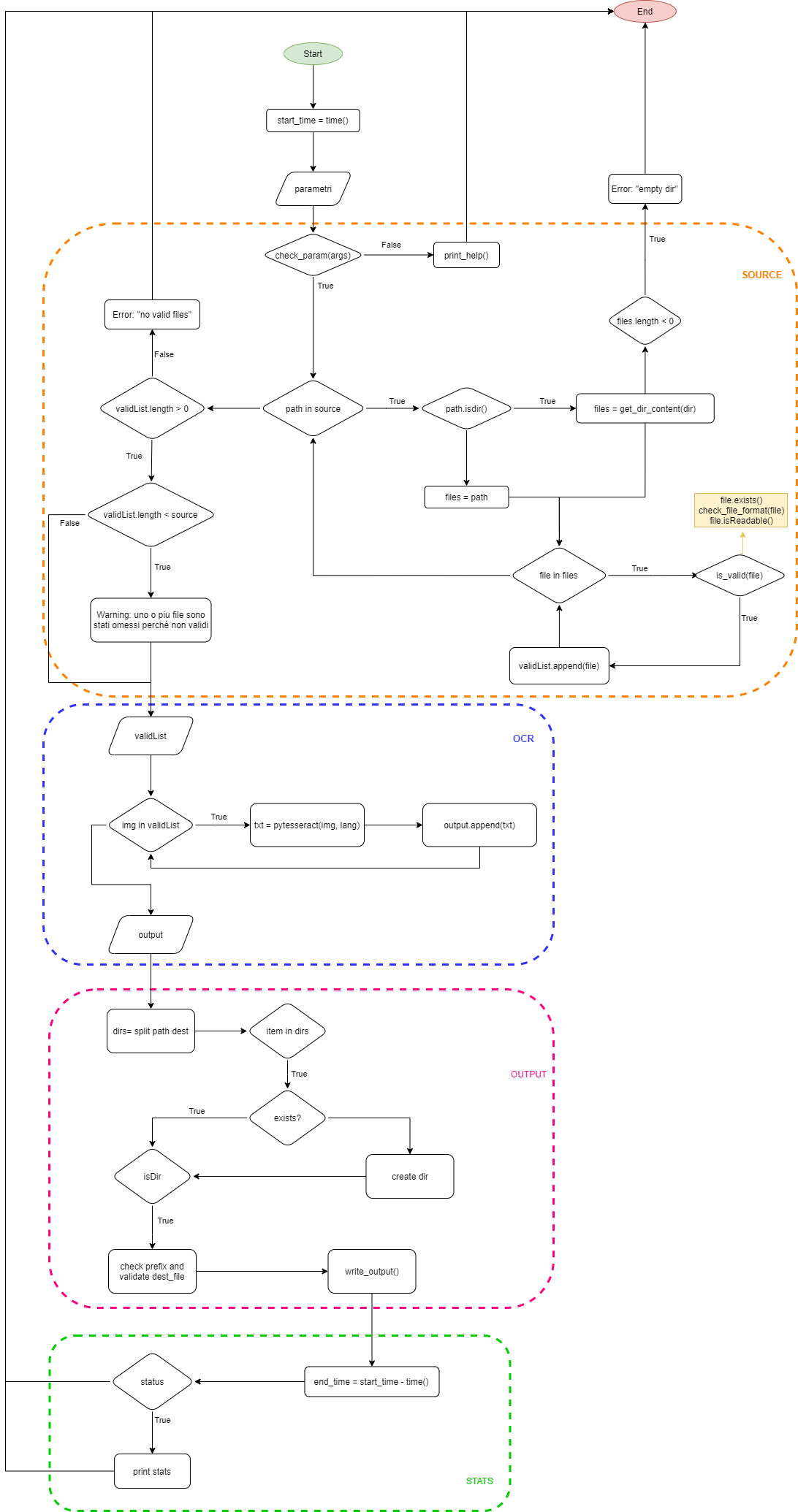
Ogni immagine della lista di file validi la si passa in un metodo dell’OCR che rileva il testo dall’immagine. Il testo rilevato lo si appende a una lista così che alla fine avremmo un file di output che contiene tutti i testi rilevati dalle immagini.

Figura 6 - Flowchart Output

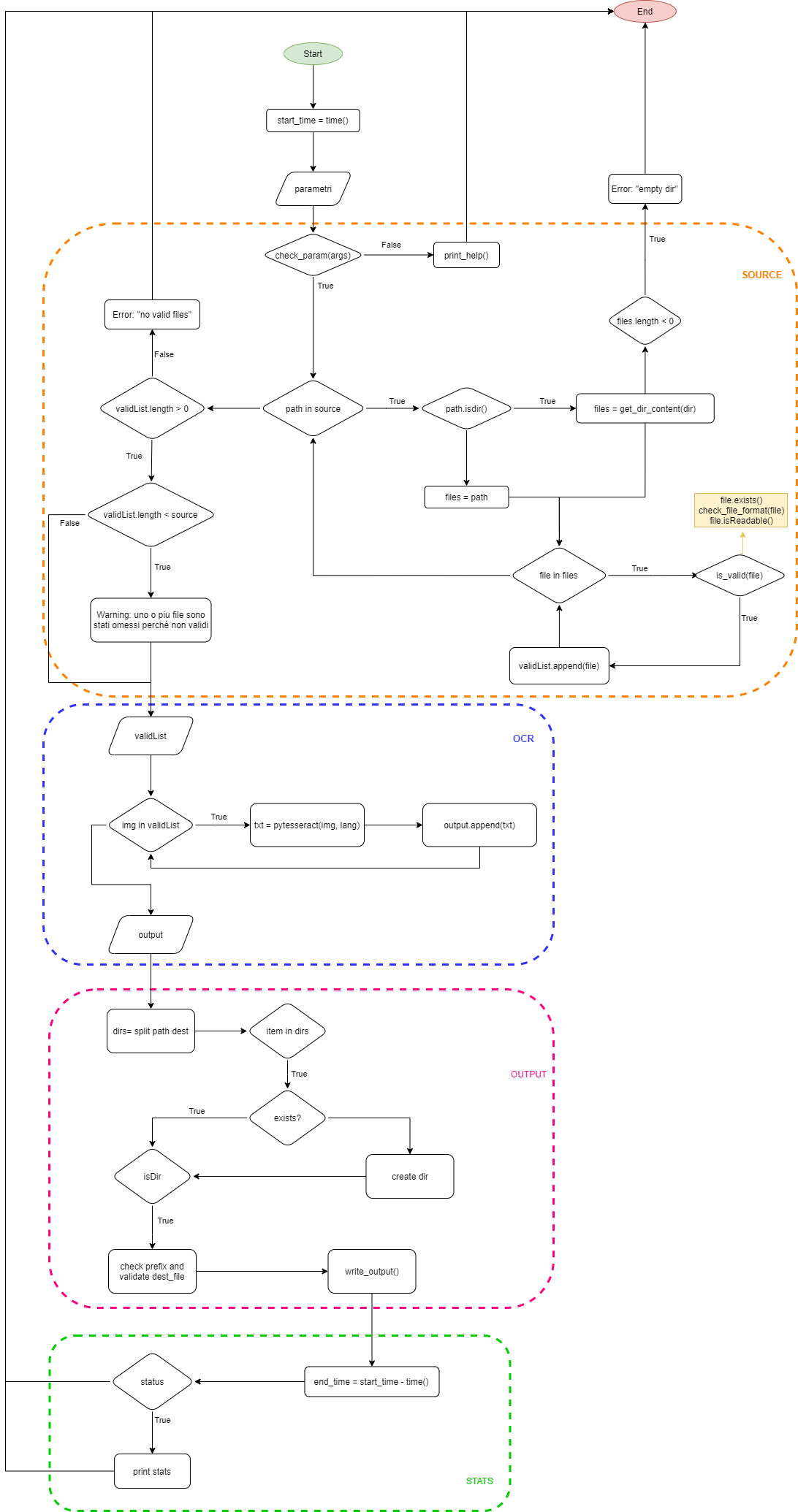
Si prende il percorso di destinazione specificato dall’utente (o quello di default) e lo si splitta in modo di poter controllare se ci sono delle cartelle. Se ce ne sono si controlla se le cartelle esistono e se non esistono le si crea. Poi si controlla se il prefisso, ovvero, il nome del file è valido e si crea il file di output. Alla fine tutti i testi nella lista di output vengono inseriti nel file di output.

Figura 7 - Flowchart Stats

Dopo che abbiamo il file di output calcoliamo il tempo impiegato dal programma. Se all’avvio del programma l'utente ha messo il parametro stats, vengono stampate anche le statistiche del file di testo altrimenti il programma termina.

## Design dei dati e database

Non abbiamo utilizzato dei database per questo progetto.

## Design delle interfacce

Questo progetto non ha un'interfaccia grafica e funziona da linea di comando. L'utente chiama il programma ocr.py inserendo gli argomenti e riceve un file che contiene tutte le scannerizzazioni di tutte le immagini che ha inserito.

Il primo argomento, il percorso del file da scannerizzare, è obbligatorio e se non viene inserito il programma termina e viene stampata la guida d'utilizzo.

Se l'utente non specifica gli altri argomenti (la lingua, la destinazione e il nome del file) il programma prende i valori di default.

Se il valore dell’argomento stats è true al termine del programma vengono stampati anche le statistiche del file di output, altrimenti non viene stampato niente.

# Implementazione

## Funzionamento generale

L'OCR CLI è un applicativo da linea di comando, riceve come input delle immagini dalle quali ricava il testo scannerizzandole e mettendolo in un file di testo. L'OCR ha vari parametri, può ricevere solo immagini png o jpg/jpeg, rileva del testo in inglese e in italiano. Si può passare anche più immagini di fila ad ogni esecuzione, specificando il nome per ogni immagine oppure usando una mask.

L'OCR per funzionare ha bisogno come minimo della sorgente (la o le immagini da cui prendere il testo), se non viene specificato nessun percorso di destinazione il file di output verrà messo in una cartella default 'scans' con il nome default 'scan.txt'. L'applicativo gestisce anche eventuali doppioni aggiungendo un numero per non avere più file con lo stesso nome.

Ovviamente l'OCR mostra l'aiuto per l'utente con il parametro -h o --help.

## Dipendenze e requisiti

Per funzionare questo progetto ha bisogno dei moduli di Python pytesseract, Pillow e del Tesseract-OCR di Google. I due moduli sono facilmente installabili tramite pip (solitamente viene installato automaticamente assieme a Python).

> py -m pip install pytesseract

> py -m pip install Pillow

Chiaramente prima di tutto questo bisogna avere installato **Python**. La versione minima richiesta in questo caso è la **3.6.0** perché è stato usato un tipo di formattazione di stringhe disponibile da quella versione in poi, però pytesseract può essere usato dalla versione 2.5 in poi di Python.

### Script per installare le dipendenze

Nel prodotto finale è stato messo anche uno script in batch che esegue i comandi di pip necessari per l’installazione delle dipendenze. Per evitare di fare un comando per ogni modulo da installare nello script è stato usato il comando con il parametro che prende un file di testo con dentro scritti i moduli.

py -m pip –r requirements.txt

Chiaramente nel file requirements.txt dovranno essere scritti i moduli richiesti.

Nello script viene anche fatto un piccolo controllo con %errorlevel%. Se non ci sono errori stampa un messaggio di riuscita, altrimenti stampa l’errore.

In caso di **proxy** questo dev’essere settato in un file pip.ini e/o nelle variabili d’ambiente del computer.

## Virtual environment

L’intero progetto è stato sviluppato nel **virtual environment** di Python. In questo modo abbiamo potuto installare tutte le dipendenze necessarie nel venv e non nella cartella root di Python.

Per creare un venv su **windows** basta usare il parametro -m venv e specificare il percorso della cartella **<dir>** in cui sarà creato l’ambiente virtuale:

> py -m venv <dir>

Ora avremo le cartelle **Lib**, **Scripts** e il file pyvenv.cfg.

Per lavorare nell’ambiente bisogna attivare il venv, entrare quindi nella cartella **/Scripts** ed eseguire lo script activate.bat.

## Struttura

OCR

|--- ocr.py

|--- reader.py

|--- stats.py

|--- log\_handler.py

|--- installModules.bat

|--- requirements.txt

|--- Tesseract-OCR

|--- log

|--- scans

ocr.py è il file nel quale vengono gestite tutte le operazioni principali: riceve gli argomenti, scannerizza le immagini passate e mette il testo ricavato nel file di testo di output.

log\_handler è il file in cui vengono definiti e gestiti i log per i file (**app\_debug.log** per il livello **debug**; **app.log** per il livello **info**) e il log su **cli** per l’utente.

I file reader.py e stats.py sono i moduli che contengono i metodi per la scannerizzazione dell’immagine, la gestione dell’output e per le statistiche dell’esecuzione.

Lo script installModules.bat serve ad installare con un click le dipendenze (contenute nel file **requirements.txt**) che servono all'OCR per funzionare.

Dopo la prima esecuzione verranno create le cartelle **log** e **scans** (scans no se l’utente specifica una cartella differente) perché sono le cartelle default per salvarci rispettivamente i file di log e i file di testo quando non viene specificato il percorso dell'output.

## Sviluppo

### Flow

La prima cosa che succede all'esecuzione del file è il controllo della versione di Python. Se la versione minima è la 3.6.0 allora continua l'esecuzione. Se poi non ci sono i moduli necessari ferma l'esecuzione mandando un messaggio di errore.

L'utente esegue l’OCR passando i parametri, ocr.py gestisce i parametri e fa partire lo scan del/dei file ottenendo l'output. Dopodiché controlla la validità della destinazione, se tutto funziona crea il file di output. Infine controlla se bisogna stampare le statistiche e se la risposta è sì le stampa.

### Logging

Per il logging abbiamo utilizzato il modulo logger di python.

Nel file log\_handler.py c’è il metodo get\_configure\_logger() nel quale andiamo a definire tre handler differenti: handler, handler\_d e handler\_cli.

handler e handler\_d scrivono nei file di log che sono all’interno della cartella **‘./log’** rispettivamente. Questi due handler hanno una formattazione completa e hanno settati rispettivamente i livelli **info** e **debug**.

handler\_cli stampa i messaggi di log a terminale pensati per l’utente, quindi con una formattazione meno esaustiva e con il livello **warning**.

Attenzione! Se non viene specificata nella configurazione dell’handler viene preso il livello di **default** che è **warning**.

Per poter utilizzare il logger, appena formattato, dobbiamo richiamare il metodo get\_configure\_logger() che ritorna un nuovo oggetto logger formattato. Dato che abbiamo 3 script principali per poter utilizzare lo stesso logger formattato in tutti e tre si dovrebbe richiamare il metodo ogni volta. Per evitare questa cosa, nel metodo è stato preso un oggetto **logger basic** a cui sono state applicate le modifiche.

Questo logger già pronto e formattato viene **ritornato** dal metodo. In questo modo ci basta richiamare il metodo una volta nel main() dell’ocr.py per avere il logger formattato che viene ereditato automaticamente dai suoi moduli importati.

Nel metodo get\_configure\_logger():

logger = logging.getLogger('')

e alla fine return logger.

In ocr.py importiamo log\_handler e poi nel main():

log\_handler.get\_configure\_logger()

### Parametri

L’OCR per funzionare necessita come minimo del percorso della/delle immagine/i da scannerizzare. Accetta anche vari parametri opzionali per configurare la scansione. Ecco la guida di utilizzo che viene stampata in caso di errore o per il parametro -h o –-help:

ocr [-h] source [-dest] [-lang] [-prefix] [--stats]

-h o --help: mostra guida aiuto

source: la sorgente delle immagini. Accetta più valori.

-d o -dest: il percorso di destinazione, dove verrà creato il file con il testo scannerizzato. Cartella default è ‘./scans’.

-l o -lang: ci sono 2 possibilità, ita o eng. Di default è eng.

-p o -prefix: il nome del file di output. Se ci sono doppioni nella cartella di destinazione aggiunge un id alla fine del nome. Default è ‘scan.txt’.

--stats: non bisogna specificarne il valore. Se c'è stampa le statistiche dello scan a fine esecuzione quali il numero di parole scannerizzate e il tempo totale di esecuzione.

Per la gestione dei parametri è stato utilizzato il modulo argparse.

Nel file ocr.py subito dopo il logger vengono creati i parametri utilizzando argparse.ArgumentParser(…). Aggiungiamo tutti i parametri descritti sopra inserendo i valori di default, se ce ne sono, e il messaggio di aiuto.

Per rendere un argomento opzionale basta mettere ‘-‘ o ‘--' davanti al nome dell’argomento, altrimenti lo viene preso come obbligatorio e ferma l’esecuzione mandando un messaggio di errore in mancanza del parametro.

Dato che il parametro source accetta più valori, tra le opzioni dell’argomento va aggiunto nargs=’+’. Questo significa che l’argomento source accetta da **(min)** **1** **valore a n valori**.

parser = argparse.ArgumentParser(usage="ocr [-h] source [-dest] [-lang] [-prefix] [--stats]")

parser.add\_argument('source', type=str, nargs='+', help=message.')

Per inserire il valore default all’argomento basta utilizzare la proprietà default. Ad esempio per il prefix nella creazione ci sarà scritto default = “scan.txt”.

Successivamente salvo i parametri in un dizionario args:

args = parser.parse\_args()

Prima di procedere con l’OCR controllo prima che i parametri non siano vuoti, se lo sono mando l’errore ed esco dall’applicazione.

### Scan immagine

Le scansioni delle immagini e il rilevamento del testo sono svolte principalmente nel metodo scan(source, lang) del modulo reader nel quale vengono passati, appunto, la lista di **immagini** da scannerizzare e la **lingua** del testo.

In questo metodo viene controllata la validità delle immagini richiamando il metodo validate\_source(source) che ritorna una lista di file validi.

#### Controllo validità sorgente

Per il controllo di validità si controlla, prima di tutto, che la lista non sia **vuota**. Vengono interpretati eventuali file **mask** in modo da avere la lista completa di immagini singole e per ogni file di questa lista completa viene controllato che sia del formato giusto (png o jpeg/jpg) e che ci siano i permessi giusti.

Per la gestione dei file mask è stato usato il modulo glob che li interpreta.

Per ogni elemento di source vengono eseguite queste righe:

f = glob.glob(img) # cerca corrispondenze di img nel percorso e le ritorna come f

source.extend(f) # aggiunge il file f alla lista source

source.pop(source.index(img)) # viene tolta dalla lista source img (il file appena interpretato)

#### Rilevamento del testo dall’immagine

Dopo aver filtrato la lista completa, mantenendo solamente i file validi da scannerizzare, si torna nel

metodo scan(…). Tramite il metodo image\_to\_string(…) di pytesseract viene rilevato il testo dell’immagine e inserito in un dizionario dove viene associato file e testo.

files\_text[f]['txt'] = img\_to\_text(f, lang) # dove f è il file corrente e files\_text è il dizionario

Dentro il metodo img\_to\_text(…) viene richiamato il metodo di pytesseract di cui parlavamo per rilevare il testo dall’immagine.

pytesseract.image\_to\_string(Image.open(img), lang)

Alla fine del metodo scan(…) viene ritornato il dizionario con associato file e testo.

### Gestione output

La parte della gestione dell’output viene gestita nel metodo principale output(output, dest, prefix) del modulo reader che accetta il dizionario ritornato da scan(…), la cartella di **destinazione** e il **prefisso**.

Come detto nella sezione [parametri](#_Parametri), essendo dest e prefix parametri opzionali se l’utente non inserisce valori essi prendo il valore default. In questo caso dest default è ‘**./scans**’ mentre prefix default è ‘**scan.txt**’.

#### Gestione destinazione (dest)

Come prima cosa nel metodo output viene controllata la cartella di **destinazione** e sono fatti controlli di esistenza e di permessi. Il percorso di destinazione viene splittato e per ogni elemento viene controllata l’esistenza della cartella. Se non esiste viene creata, e il ciclo si ripete finché non si ha tutto il percorso valido.

for item in dirs:

p = path.join(p, item)

if not path.isdir(p):

create\_dir(p)

#### Gestione prefisso (prefix)

Una volta che si ha la cartella di destinazione si controlla il prefix, si confronta il nome del file di destinazione teorico con il contenuto della cartella di destinazione. Se c’è una corrispondenza allora viene validata la destinazione aggiungendo un id incrementale al nome del file e rifacendo il controllo finché non si ha il nome del file univoco.

Ecco un piccolo metacodice sui passaggi:

if exists(prefix)

get\_dir\_content(dest)

p = f"{prefix}\_{id}.txt"

for file in dir\_content:

if file == p:

id++

p = f"{prefix}\_{id}.txt"

dest\_file = path.join(dest, p)

Se ad esempio all’inizio la cartella scans è vuota, alla prima esecuzione verrà creato il file scan.txt; alla seconda esecuzione scan.txt non andrà bene perché c’è una corrispondenza nella cartella quindi viene aggiungo l’id ottenendo il file scan\_1.txt e così via.

#### Scrittura file output

Una volta ottenuto il percorso valido del file di destinazione non resta che prendere il testo rilevato e scriverlo nel file.

Viene quindi passato il dizionario **output** al metodo merge\_output(output) che prende il testo di tutti gli elementi del dizionario mettendoli in una stringa unica e aggiungendo un separatore tra i testi dei file.

for key, value in output.items():

text += f"\n-----{key}-----\n\n" + value["txt"]

Come ultima cosa non resta che prendere il testo intero e **scriverlo nel file di output** con il metodo write\_output(text, path). Se il file non esiste viene creato.

Viene anche specificato l’enconding in utf-8 in questo modo siamo sicuri che il file di testo interpreterà correttamente i vari caratteri speciali.

with open(path, "w", encoding="utf-8") as f:

f.write(text)

Utilizzando la parola chiave with non dobbiamo neanche gestire eventuali errori di input/output o chiudere il file dopo aver finito di lavorarci perché gestisce tutto lui.

### Statistiche d’esecuzione

Come ultimo passaggio, prima di terminare l’esecuzione viene controllato se il parametro --stats è stato settato dall’utente, se è a True richiama il metodo get\_stats(dest\_file, time) del modulo stats.

All’interno di questo metodo vengono contati i **caratteri rilevati** dall’OCR (incluse righe vuote) assieme al **tempo di esecuzione** che viene calcolato facendo una semplice differenza tra il tempo iniziale e il tempo finale.

Il metodo count\_words(…)va a leggere le righe del file di output contando i caratteri. Vengono contate anche le righe vuote dato che l’OCR deve rilevarle e mettere lo spazio.

Alla fine le statistiche vengono **formattate e stampate a terminale**.

# Test

## Protocollo di test

### Test funzionali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 | **Nome:** | Acquisizione di immagini in formato PNG o JPG/JPEG | |
| **Descrizione:** | Verificare che l'OCR prenda solo PNG e JPG/JPEG | | | |
| **Prerequisiti:** | Avere installato l’OCR e avere un’immagine di un testo in formato PNG, JPG/JPEG e un altro formato | | | |
| **Procedura:** | 1. Utilizzare l’OCR con un’immagine JPEG, JPG, PNG e un altro formato   > ocr.py img/engtxtjpeg.jpeg  > ocr.py img/engtxtpng.png  > ocr.py img/itatxtjpg.jpg  > ocr.py img/error.tiff   1. Guardare il file di log per capire quali sono i formati accettati | | | |
| **Risultati attesi:** | L’OCR dovrebbe accettare solo le immagini PNG e JPG/JPEG e per i file con un formato diverso stampa il messaggio d’errore. | | | |
| **Dato** | | | **Risultato** |
| Con PNG | | | OK |
| Con JPG | | | OK |
|  | Con JPEG | | | OK |
|  | Altro formato | | | FAILED |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 REQ-003 | **Nome:** | Rilevamento del testo dall’immagine con un algoritmo OCR | |
| **Descrizione:** | L'OCR deve rilevare il testo correttamente dall'immagine | | | |
| **Prerequisiti:** | Avere installato l’OCR e avere un’immagine di un testo in formato JPG/JPEG o PNG. | | | |
| **Procedura:** | 1. Utilizzare l’OCR con un’immagine che rispetta le specifiche che si possono trovare nella documentazione 2. Guardare l’immagine e il file di output e vedere se il testo è uguale | | | |
| **Risultati attesi:** | Il testo rilevato dall’immagine deve coincidere con il testo dell’immagine | | | |
| **Dato** | | | **Risultato** |
| Con PNG o JPG/JPEG | | | Output testo dall’immagine su file TXT |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-004 | **Nome:** | Rilevamento del testo in inglese, italiano e qualsiasi altra lingua | |
| **Descrizione:** | L'OCR deve rilevare il testo correttamente dall'immagine | | | |
| **Prerequisiti:** | Aver installato l’OCR e avere un’immagine di un testo in formato JPG/JPEG o PNG aggiungendo l’argomento –lang | | | |
| **Procedura:** | 1. Utilizzare l’OCR con un’immagine aggiungendo il parametro –l o –lang   > ocr.py img/itatxtjpg.jpg -l eng  > ocr.py img/itatxtjpg.jpg -l ita  > ocr.py img/itatxtjpg.jpg -l fra   1. Guardare il file di log per capire se la lingua inserita è accettata | | | |
| **Risultati attesi:** | L’OCR dovrebbe accettare solo le lingue Italiano e Inglese e per le altre lingue stampa un messaggio d’errore. | | | |
| **Dato** | | | **Risultato** |
| Con PNG/JPG/JPEG in inglese | | | OK |
| Con PNG/JPG/JPEG in italiano | | | OK |
| Altra lingua | | | FAILED |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-005 | **Nome:** | Inserire multipli file di input e ricevere un file di output singolo | |
| **Descrizione:** | Fare il rilevamento di più immagini e scrivere l’output in un singolo file TXT | | | |
| **Prerequisiti:** | Avere un’immagine in formato PNG o JPG/JPEG | | | |
| **Procedura:** | 1. Utilizzare l’OCR con più immagini   > ocr.py img/engtxtjpeg.jpeg img/engtxtpng.png img/itatxtjpg.jpg  > ocr.py img/\*  > ocr.py \*.\*   1. Controllare se nel file di output ci sono i contenuti di tutte le immagini | | | |
| **Risultati attesi:** | Si ottiene un singolo file di output | | | |
| **Dato** | | | **Risultato** |
| Input: img1.png img2.png, img3.jpeg | | | Un file di output .TXT che contiene il contenuto di tutte le immagini inserite |
| Input: img.jpg img/ img1/ img.png | | |
|  | Input: \*.\* | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-006  REQ-007 | **Nome:** | Statistiche scansioni | |
| **Descrizione:** | Controllare che dopo l’esecuzione dell’OCR vengano calcolate le statistiche e controllare che i dati statistici vengano visualizzati tramite il parametro opzionale | | | |
| **Prerequisiti:** | Avere un’immagine in formato PNG o JPG/JPEG | | | |
| **Procedura:** | 1. Utilizzare l’OCR   > ocr.py img/engtxtpng.png –stats | | | |
| **Risultati attesi:** | Alla fine dell’esecuzione a terminale dev’essere stampata la statistica del file.  Le statistiche devono coincidere con il file di output dell’immagine | | | |
| **Dato** | | | **Risultato** |
|  | Con il parametro –stats | | | Stampa le statistiche al terminale |
|  | Senza parametri | | | Esegue l’OCR normalmente senza mostrare le statistiche |
|  | Input: img.txt | | | Le statistiche stampate al terminale coincidono con le statistiche del file di output dell’immagine |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-008 | **Nome:** | Guida utilizzo | |
| **Descrizione:** | Con il parametro –h/--help viene stampata a terminale la guida di utilizzo | | | |
| **Prerequisiti:** | Utilizzare il programma ocr.py | | | |
| **Procedura:** | 1. Utilizzare il comando OCR con il parametro –h o -help   > ocr.py –help | | | |
| **Risultati attesi:** | Viene stampata a terminale la guida di utilizzo | | | |
| **Dato** | | | **Risultato** |
| Con il parametro –h / --h / --help | | | La guida di utilizzo |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-012 | **Nome:** | Dipendenze SW incluse |
| **Descrizione:** | Eventuali librerie esterne devono essere incluse e facilmente installabili | | |
| **Prerequisiti:** | Con l’eseguibile devono essere installate e incluse tutte le dipendenze | | |
| **Procedura:** | 1. Eseguire il .exe dell’OCR 2. Controllare che tutte le dipendenze siano installate | | |
| **Risultati attesi:** | Tutte le dipendenze devono essere incluse e installate con l’exe | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-013 | **Nome:** | Attendibilità dei dati inseriti dall’utente | |
| **Descrizione:** | I parametri inseriti dall’utente sono verificati e corretti | | | |
| **Prerequisiti:** | Avere il programma funzionante | | | |
| **Procedura:** | 1. Utilizzare il comando OCR con tutti i parametri   > ocr.py img/engtxtpng.png –lang eng  > ocr.py img/engtxtpng.png –dest C:\Scans\  > ocr.py img/engtxtpng.png -prefix scan  > ocr.py img/engtxtpng.png –stats | | | |
| **Risultati attesi:** | L’OCR dovrebbe accettare solo i parametri dest, lang, prefix e stats. Per qualsiasi altro parametro (tranne –help/-h) dà errore. | | | |
| **Dato** | | | **Risultato** |
| -dest / -d | | | OK |
| -lang / -l | | | OK |
| -prefix / -p | | | OK |
| -stats | | | OK |
| Qualsiasi altro parametro | | | FAILED |

### Test non funzionali

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-09  REQ-009 | **Nome:** | Utilità (applicazione) |
| **Descrizione:** | Controllo del funzionamento del programma | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-010 | **Nome:** | Gestione degli errori |
| **Descrizione:** | Verifica se tutti gli errori sono gestiti correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-011 | **Nome:** | Registro eventi/Logging |
| **Descrizione:** | Verifica se i dati corretti o errati registrati in un file di log;  Verifica se i dati sono salvati nel formato appropriato e raggruppati in modo utile | | |
| **Prerequisiti:** | Utilizzare l’OCR | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-012  REQ-013 | **Nome:** | Gestione delle eccezioni (batch) |
| **Descrizione:** | Verifica se tutte le eccezioni sono gestite correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-013  REQ-015 | **Nome:** | Organizzazione del programma |
| **Descrizione:** | Se il programma è strutturato in metodi ed è commentato | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-014  REQ-016 | **Nome:** | Utilizzo di diagrammi di flusso |
| **Descrizione:** | Se ci sono i diagrammi di flusso | | |

## Risultati test

**Legenda**: test riuscito test fallito

|  |  |  |
| --- | --- | --- |
| ID Test Case | Nome | Risultato |
| TC-001 | Acquisizione di immagini in formato PNG o JPG/JPEG |  |
| TC-002 | Rilevamento del testo dall’immagine con un algoritmo OCR |  |
| TC-003 | Rilevamento del testo in inglese, italiano e qualsiasi altra lingua |  |
| TC-004 | Inserire multipli file di input e ricevere un file di output singolo |  |
| TC-005 | Statistiche scansioni |  |
| TC-006 | Guida utilizzo |  |
| TC-007 | Dipendenze SW incluse |  |
| TC-008 | Attendibilità dei dati inseriti dall’utente |  |
| TC-009 | Utilità (applicazione) |  |
| TC-010 | Gestione degli errori |  |
| TC-011 | Registro eventi/Logging |  |
| TC-012 | Gestione delle eccezioni (batch) |  |
| TC-013 | Organizzazione del programma |  |
| TC-014 | Utilizzo di diagrammi di flusso |  |

## Mancanze/limitazioni conosciute

Il progetto è stato sviluppato rispettando tutti i requisiti presenti nel quaderno dei compiti. È completo e quindi non ci sono delle mancanze. L’unica limitazione erano le nostre conoscenze limitate del linguaggio di programmazione Python. Nel tempo che abbiamo lavorato su questo progetto siamo riuscite a capire il linguaggio abbastanza bene da poter sviluppare l’applicativo.

# Consuntivo

Figura 8 - Gantt consuntivo

Questo e il diagramma di Gantt consuntivo, ovvero è la differenza tra il Gantt preventivo e il Gantt consuntivo. Le righe nere rappresentano il Gantt preventivo mentre quelle blu rappresentano il Gantt consuntivo.

Ci sono delle differenze rispetto alla pianificazione iniziale. Le fasi di analisi e pianificazione non cambiano molto, ma possiamo notare che le fasi di implementazione e test sono cambiate rispetto alla pianificazione. Non abbiamo seguito l’ordine d’esecuzione stabilito nella pianificazione. Nel diagramma di Gantt preventivo abbiamo messo che avremmo fatto il test dell’attività subito dopo averla finita mentre nel diagramma di Gantt consuntivo si vede che prima abbiamo fatto l’implementazione e poi tutti i test funzionali e non funzionali. Inoltre, abbiamo calcolato male alcuni tempi perché non abbiamo pensato agli imprevisti e problemi che si sono presentati durante la fase dell’implementazione.

# Conclusioni

## Sviluppi futuri

Ci sono delle modifiche che si possono fare per migliorare l’applicativo.

In futuro l‘applicazione la si potrebbe trasformare in un applicativo web aggiungendo un’interfaccia grafica. In questo modo sarebbe più facile da usare e gestire. Si potrebbe aggiungere l’interfaccia di registrazione e login in modo da poter gestire i clienti che accedono al applicativo web e i tipi di utilizzo.

## Considerazioni personali

### In generale

Troviamo che questo metodo di lavorale è stato molto utile in quanto abbiamo imparato a suddividerci il lavoro e gestirci meglio.

### Considerazioni personali Viktorija

Questo modo di lavorare mi è piaciuto parecchio. Ho avuto la possibilità di lavorare in un team e grazie a questo ho imparato tante cose dall’altro membro del team. Per esempio ho imparato a organizzarmi meglio e a gestire meglio i tempi. Grazie a questo progetto ho imparato a lavorare in un team e a suddividere il carico di lavoro.

### Considerazioni personali Thaisa

Devo dire che questo progetto mi è piaciuto molto, ho imparato le basi di Python e ho avuto un’ottima esperienza di lavoro in team. Questo è stato il secondo progetto e questa volta la progettazione trovo sia andata molto meglio dello scorso progetto, ci siamo organizzate molto meglio a parte i problemi esterni all’inizio.

# Sitografia

* <https://pypi.org/project/pytesseract/>, *pytesseract,* 14-02-2021.
* <https://github.com/tesseract-ocr/tesseract>, *Tesseract OCR,* 14-02-2021.
* <https://www.w3schools.com/python/>, *Python w3schools*, 28-02-2021.
* <https://realpython.com/>, *Python tutorial realpython*, 28-02-2021.
* <https://docs.python.org/3/library/argparse.html>, *argparse module*, 04-02-2021.
* <https://stackoverflow.com/questions/5458048/how-can-i-make-a-python-script-standalone-executable-to-run-without-any-dependen>, *making a python standalone executable*, 04-02-2021.
* <https://docs.python.org/3/library/logging.html>, *Python* *logging*, 04-02-2021.
* <https://pillow.readthedocs.io/en/stable/index.html>, *Pillow module*, 04-02-2021.
* <https://pip.pypa.io/en/stable/reference/pip/>, *Python pip, 04-02-2021.*
* <https://pep8.org/>, *pep8*, 04-02-2021.
* <https://realpython.com/python-cli-testing/>, *Python cli testing,* 04-03-2021.
* <https://ss64.com/nt/errorlevel.html>, *errorlevels,* 04-03-2021.
* <https://python101.pythonlibrary.org/chapter20_sys.html>, *sys module,* 04-03-2021.
* <http://steve-jansen.github.io/guides/windows-batch-scripting/>, *batch tutorial,* 04-03-2021.
* <https://www.atlassian.com/git/tutorials/setting-up-a-repository>, *saving changes on gitignore*, 04-03-2021.
* <https://docs.python.org/3/library/venv.html>, *Python venv,* 04-03-2021.
* <https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html>, *Quality Tesseract OCR*, 15-04-2021.
* <https://groups.google.com/g/tesseract-ocr/c/Wdh_JJwnw94/m/24JHDYQbBQAJ>, *altezza minima lettere*, 15-04-2021.
* <https://tesseract-ocr.github.io/tessdoc/FAQ.html>, *FAQ Tesseract OCR*, 15-04-2021.
* <https://github.com/tesseract-ocr/tesseract/issues/82>, *risoluzione minima Tesseract OCR*, 15-04-2021.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Quaderno dei compiti
* Codice sorgente presente su GitHub: <https://github.com/geo-petrini/ocr_cli>