UNIVERSITÀ
DI TRENTO

Task 1

Team: *Stentinel*

Avesani Manuel | Laiti Francesco

Elaborazione e trasmissione delle immagini
a.a. 2020/2021

Indice

1. Modellazione del progetto
2. Codice e scelte effettuate
3. Problemi & soluzioni
4. Demo

Indice

1. MODELLAZIONE DEL PROGETTO - OBIETTIVO e AMBIENTE

DEFINIZIONE DELLA STRUTTURA DATI BASE

Definizione e implementazione della sequenza di classi che permettono di gestire l'engine, i blocchi e la loro interconnessione.

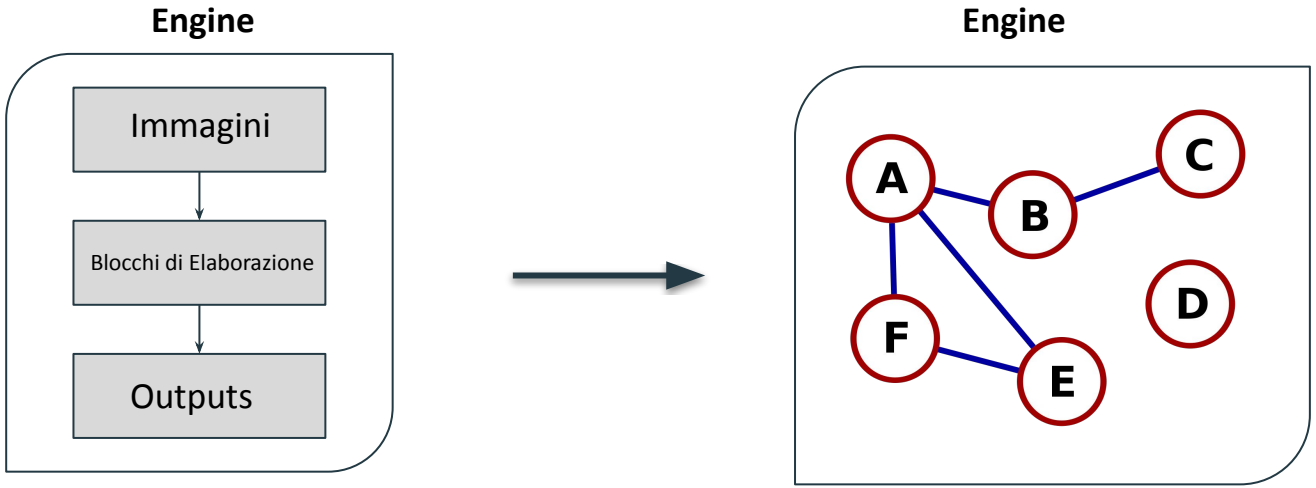
AMBIENTE

Il codice è stato sviluppato originariamente su Google Colab per poter lavorare in modo condiviso allo stesso progetto ed avere un ambiente virtuale con tutte le librerie a disposizione.

Successivamente il codice è stato implementato in PyCharm 2021.1 con ambiente virtuale Python 3.8 per poter testare la webcam.

Nota: la catena di elaborazione lavora solo con immagini a scala di grigi

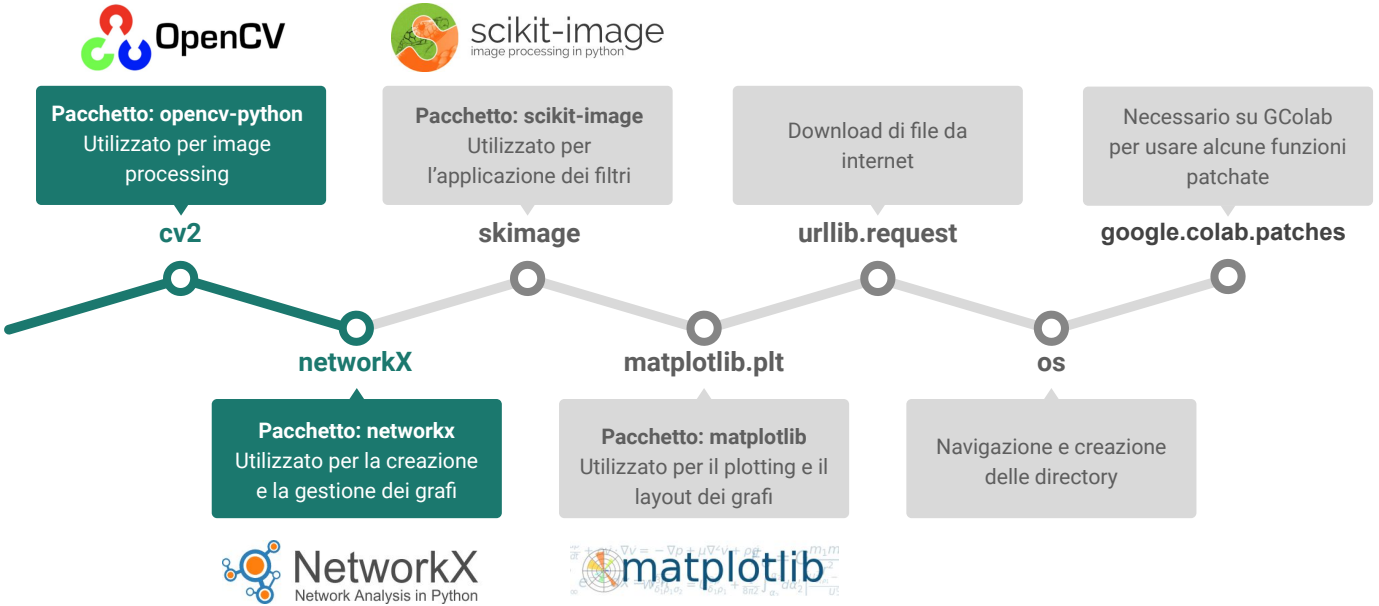
1. MODELLAZIONE DEL PROGETTO - STRUTTURA e GRAFO



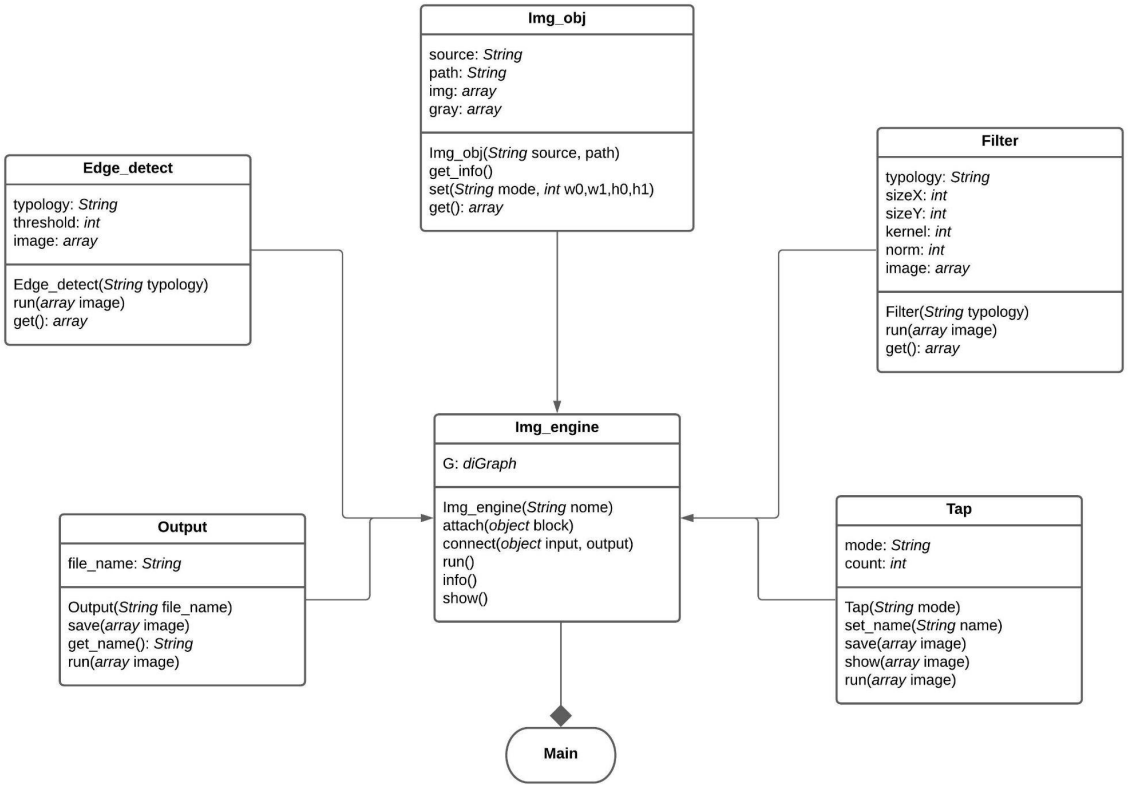
Ogni blocco di elaborazione è stato implementato come **classe**, in modo da poter creare diversi **oggetti** in modo semplice.
Ogni oggetto viene aggiunto all'engine e connesso con altri oggetti per formare un **grafo**, trasformando la rete gerarchica della catena di elaborazione in una concatenazione tra blocchi

Indice

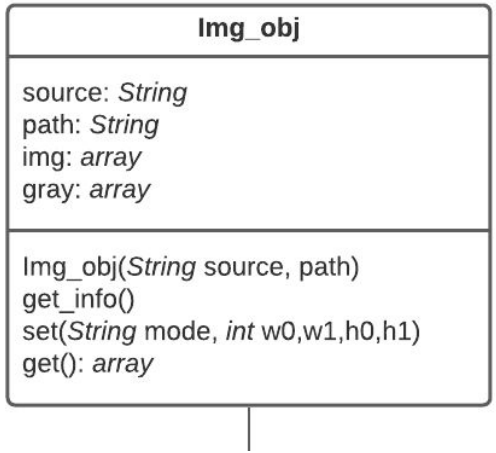
2. CODICE E SCELTE EFFETTUATE - LIBRERIE UTILIZZATE



2. CODICE E SCELTE EFFETTUATE - SCHEMA INIZIALE



2. CODICE E SCELTE EFFETTUATE - CLASSE IMMAGINE



Img_obj:
creazione dell'oggetto immagine ottenuta da una sorgente. Avviene una conversione automatica dell'immagine in input dalla scala a colori alla scala di grigi

Struttura:

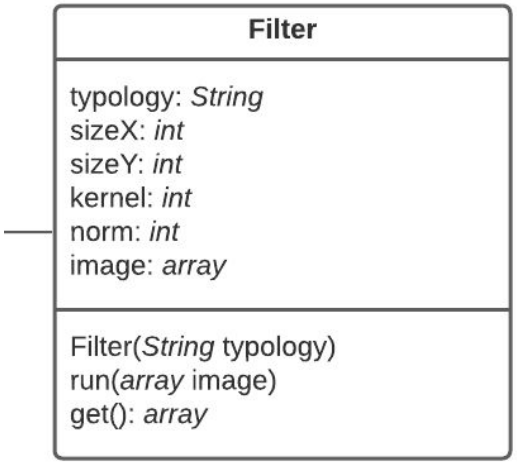
1. __init__(*string* source, *string* path='') -> inizializza l'oggetto Img_obj
2. get_info() -> restituisce le informazioni sull'img
3. set(*string* mode, *int* w0=0,w1=0,h0=0,h1=0) -> modifiche all'img
4. get() -> ritorna l'img dell'istanza corrente

Configurazioni disponibili:

1. *source*
   a. *camera*: cattura l'img dalla camera integrata
   b. *url*: scarica l'immagine da un link
   c. *local*: img già presente sul dispositivo

   } *path* da inserire
2. *mode*
   a. *flip*: ruota l'immagine di 180°
   b. *mirror*: specchia l'immagine
   c. *crop*: taglia l'img nell'intervallo di pixel inseriti dall'utente (lunghezza [w0:w1], altezza [h0:h1])

2. CODICE E SCELTE EFFETTUATE - CLASSE FILTRO



Filter:
creazione e applicazione del filtro impostato. In questo progetto vengono utilizzati dei filtri già presenti nella libreria *skimage*; per questo motivo i parametri per i filtri sono di default =0 se non vengono esplicitamente inseriti

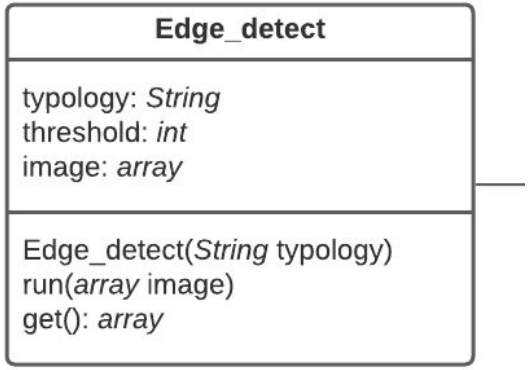
Struttura:

1. __init__(*string* typology) -> inizializza l'oggetto Filter
2. run(*array* image) -> applica il filtro all'immagine *image* passata
3. get() -> restituisce l'img con il filtro applicato

Configurazioni disponibili:

1. *typology*
   a. *median*: applica un filtro a media mobile
   b. *gaussian*: applica un filtro gaussiano

2. CODICE E SCELTE EFFETTUATE - CLASSE EDGE DETECTOR



Edge_detect:
creazione e applicazione del filtro impostato per l'estrazione dei contorni. In questo progetto vengono utilizzati dei filtri già presenti nella libreria *skimage*; per questo motivo il parametro *threshold* è di default =0 se non viene esplicitamente inserito

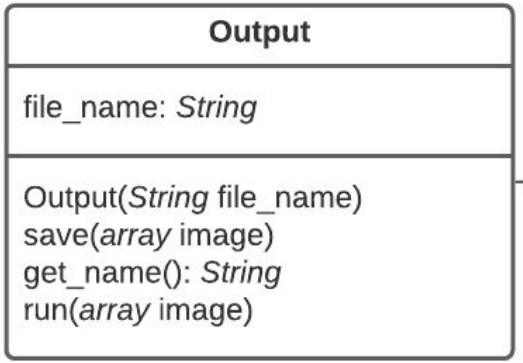
Struttura:

1. __init__(*string* typology) -> inizializza l'oggetto Edge_detect
2. run(*array* image) -> applica il filtro all'immagine *image* passata
3. get() -> restituisce l'img con il filtro applicato

Configurazioni disponibili:

1. *typology*
    a. *sobel*: applica il filtro di Sobel
    b. *prewitt*: applica il filtro di Prewitt
    c. *roberts*: applica il filtro di Roberts

2. CODICE E SCELTE EFFETTUATE - CLASSE OUTPUT

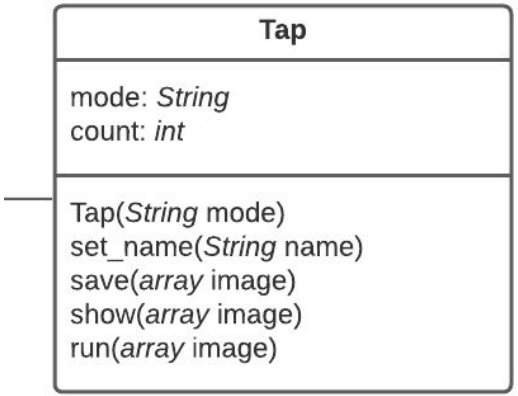


Output:
salvataggio dei risultati nella cartella /Output creata nella directory dove è presente il file sorgente del progetto. Restituisce un messaggio di corretto salvataggio dell'img o di errore

Struttura:

1. __init__(*string* file_name) -> inizializza l'oggetto Output
2. save(*array* image) -> salva l'immagine nella directory con il nome *file_name*
3. get_name() -> restituisce il nome del file in output
4. run(*array* image) -> esegue il salvataggio; richiama la funzione save()

2. CODICE E SCELTE EFFETTUATE - CLASSE TAP



Tap:
visualizzazione o salvataggio dei risultati intermedi nella cartella /Tap creata nella directory dove è presente il file sorgente del progetto. Restituisce un messaggio di corretto salvataggio dell'img o di errore.
La variabile *count* viene utilizzata nel caso ci siano più img da salvare/visualizzare.

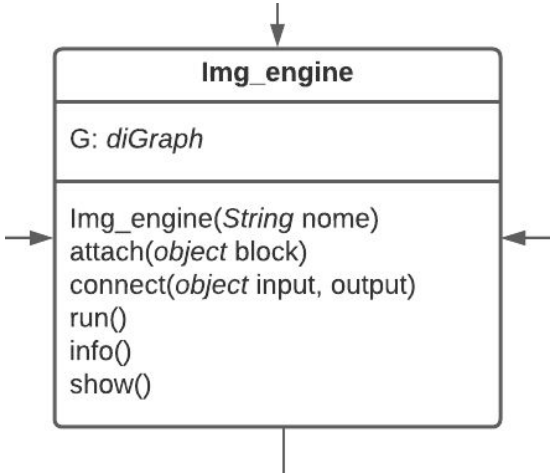
Struttura:

1. __init__(*string* mode) -> inizializza l'oggetto Tap
2. set_name(*string* name) -> imposta il nome dell'img da salvare osservando da quale blocco proviene il file
3. save(*array* image) -> salva l'immagine nella directory con il nome *name*
4. show(*array* image) -> mostra l'img passata
5. run(*array* image) -> esegue il salvataggio o la visualizzazione in funzione del parametro *mode*

Configurazioni disponibili:

1. *mode*
   a. *save*: imposta Tap per il salvataggio
   b. *show*: imposta Tap per la visualizzazione

2. CODICE E SCELTE EFFETTUATE - CLASSE ENGINE



Img_engine:
creazione e gestione delle interconnessioni dei blocchi per giungere al risultato finale. L'engine è stato strutturato come un grafo direzionale ed è stata scelta la libreria *networkx* con soluzione custom per eseguire la catena di elaborazione

Struttura:

1. __init__(*string* nome) -> inizializza l'oggetto Img_engine
2. attach(*object* block) -> creazione del nodo di tipo *block*
3. connect(*object* input, output) -> connessione del nodo *input* con *output*
4. run() -> esecuzione della catena di elaborazione
5. info() -> restituisce informazioni sul grafo ed i nodi presenti nel grafo
6. show() -> mostra una rappresentazione grafica del grafo

2. CODICE E SCELTE EFFETTUATE - UTILIZZO DEL GRAFO

Creazione blocchi	Creazione grafo	Conversione in dizionario	Lettura chiave	Lettura valori
Creazione degli oggetti generati dalle singole classi. Ottengo i blocchi	Aggiunta dei blocchi all'engine e creazione delle connessioni tra i blocchi	Conversione del grafo in dizionario per usare le coppie chiave-valori	Le chiavi del dizionario corrispondono ai blocchi inseriti come input in fase di *connect()*. L'elemento *chiave* è importante per acquisire l'immagine da elaborare, disponibile tramite la funzione *get()* presente in tutti i blocchi tranne per le classi *Output* e *Tap*	I valori della chiave del dizionario corrispondono ai blocchi inseriti come output in fase di *connect()*. L'elemento *valore/i* è importante per elaborare l'immagine passata dal dizionario tramite funzione *run()* disponibile in tutte le classi

2. CODICE E SCELTE EFFETTUATE - RISULTATO FINALE

Grazie alla funzione *show()* disponibile nella classe *Img_engine* è possibile vedere la catena di elaborazione generata dalla classe.

A titolo di esempio, la catena di elaborazione schematizzata in figura 1 è stata implementata nel main ed il risultato del grafo generato dall'engine è visualizzabile nella figura 2.

Nota: il grafico viene generato in modo differente ad ogni esecuzione e non è stato possibile creare un layout di default.

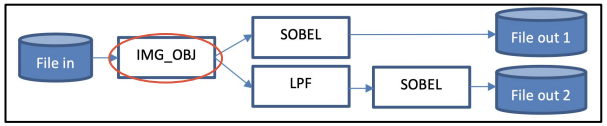


figura 1

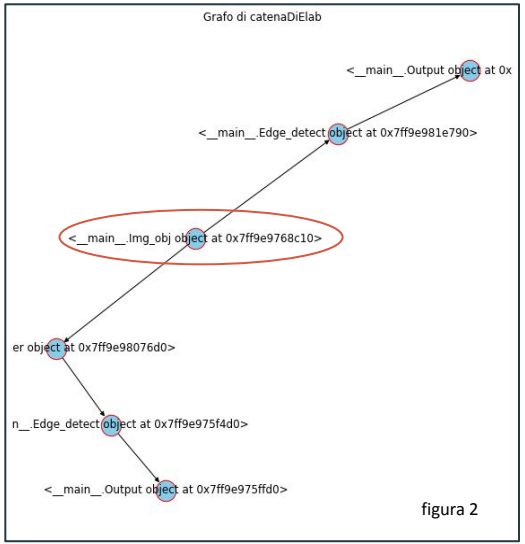


figura 2

Indice

3. PROBLEMI & SOLUZIONI - PROBLEMI RISOLTI

01	**Difficile implementazione dell'engine**	• Utilizzo della libreria *networkX*
02	**Uscita dei filtri compresa tra 0 e 1**	• Normalizzazione tra 0 e 255 dei valori per avere risultati apprezzabili
03	**Assegnazione del nome corretto all'immagine quando viene salvata mediante *Tap.save(image)***	• Uso della funzione *Tap.set_name(name)* in fase di connessione dei blocchi
04	***cv2.imshow(name,image)* non funziona su Google Colab**	• Uso di *cv2_imshow(image)* per correggere il problema
05	**Difficile utilizzo del grafo per il nostro scopo**	• Conversione del grafo in dizionario che permette di usare una chiave ed un valore della lista per eseguire le operazioni

3. PROBLEMI & SOLUZIONI - OSSERVAZIONI

01	**Classe *Tap* e *Output* non possono essere connessi come input**	• Controllo del corretto inserimento in fase di connessione dei blocchi
02	**Classe *Img_obj* non può essere messa come output**	• Controllo del corretto inserimento in fase di connessione dei blocchi
03	**Non tutte le classi utilizzano la funzione *get()***	• Si escludono *Output* e *Tap* da questa funzione perché non restituiscono un img ma sono solo classi di salvataggio o visualizzazione
04	**Creare una funzione standard per tutte le esecuzioni**	• Creazione di *def run()* in ogni classe

Indice

1. Modellazione del progetto
2. Codice e scelte effettuate
3. Problemi & soluzioni
4. **Demo**

DEMO

Google Colab

Source Code