Visione artificiale

Alessandro Pioggia, Luca Rengo, Federico Brunelli, Leon Baiocchi

21 febbraio 2022

Indice

Capitolo 1

Intro

Idea = cercare di dotare le macchine della vista, si cerca di insegnare ai computer come interpretare le informazioni presenti in immagini e video.

1.1 Che cosa vede un computer?

Un pc vede una matrice di pixel ed ogni pixel è rappresentato da un numero. Questo è il punto di partenza che ci permetterà di capire che cosa c'è nell'immagine.

1.2 Perchè è difficile?

Punti di vista diversi, occlusione (un oggetto ne copre un altro), distorsione, movimento (a volte può tornare utile), variazioni-intra-classe, sfondo complesso (sfondo non omogeneo, pieno di informazioni che non ci interessano).

1.3 Perchè è importante?

Ha molteplici applicazioni, quali

- Sicurezza stradale;
- Salute;
- Prevenzione del crimine;
- Protezione civile;
- Divertimento;
- Domotica.

1.4 Possibili applicazioni - Panoramica generale

- Misurazione, conteggio, qualità : Hanno in comune il fatto di volere misurare la qualità di un qualche oggetto;
  - Misurazione precisa non a contatto : In campo industriale pensiamo ad una catena di montaggio in cui passano prodotti che hanno bisogno di un controllo qualità, che viene svolto da una macchina che sfrutta tecniche di visione artificiale (Controllo delle quantità di liquido in una bottiglietta).
  - Conteggio di veicoli e persone in una piazza, in generale stime dimensionali;
- Elaborazione immagini avanzata
  - Miglioramento immagini;
  - Immagini mediche;
  - Segmentazione di aree agricoli, utili per individuare le immagini aeree e satellitari.
- Riconoscimento
  - Classificazione/individuazione di oggetti : Tag inseriti all'interno dell'immagine, Ricerca per somiglianza di immagini, riconoscere segnali stradali;
  - Riconoscimento persone : punta non solo a classificare, bensì riconoscere univocamente una persona.
- Movimento
  - Video sorveglianza : individuare un ladro, ritrovare uno zaino perso;
  - Navigazione e guida autonoma.

1.5 Strumenti di lavoro

1.5.1 Python

Standard di fatto per lo sviluppo di software scientifico, tra le altre cose anche di visione artificiale. Linguaggio di alto livello, che ha un certo livello di astrazione, in generale con poche linee, se utilizzate in modo corretto possiamo esprimere concetti piuttosto complessi.

1.5.2 OpenCV

OpenCV è la libreria standard per quanto riguarda la visione artificiale, è sviluppata in C++, però è utilizzata anche da altri linguaggi (quali python).

1.5.3 NumPy

La Lazzaro ce lo ha già spiegato.

1.5.4 OpenCV-Python

Una volta richiamato OpenCV da python, tutte le strutture dati (vettori, punti nelle immagini, ecc.) sono array numPy.

1.5.5 Jupyter Notebook

Ambiente web, in cui vengono scritti piccoli pezzi di codice, immagini e altro al fine di creare dei documenti interattivi.

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 2

Python

Python è il Leonardo Di Caprio dei linguaggi di programmazione. Ma perchè?

2.1 Descrizione

- Python è un linguaggio ad alto livello, general purpose, può essere utilizzato per qualunque tipo di applicazione;
- il codice deve essere facile da leggere, è una prerogativa;
- occorrono poche linee di codice per sviluppare anche concetti complessi;
- supporta programmazione : oop, imperativa e funzionale;

2.2 Fun fact

L'autore, Guido Van Rossum, sceglie il nome python perchè amante del gruppo comico Monty Python, attivo negli anni 70-80.

2.3 Zucchero sintattico

2.3.1 Indentazione

```
#Non si puo' cambiare indentazione in un blocco
print("Salve Cesena")
   print("Salve di nuovo!")

#L'indentazione definisce i blocchi di codice
if 42 < 0:
   print("ciao")
print("ciao ma in corsivo")

#andata a capo
istruzione_molto_lunga = 2 + 5 \
                  + 6 + 7

#Piu' istruzioni in una singola riga
print("a"); print("b")
```

2.3.2 Variabili

```
#Le variabili in python non si dichiarano, vengono create automaticamente al
    momento dell'inizializzazione sono case sensitive, iniziano con una lettera
    o underscore e i caratteri ammessi sono le Lettere, i numeri e l'underscore
    (codifica UNICODE)

x = 4
y = "Visione artificiale"
print(x, y)

#Assegnamento dello stesso valore a piu' variabili

x = y = z = 90

#Variabili globali e locali

a = "cool" #Variabile globale

def function():
   a = "bad" #Variabile locale alla funzione

#Se aggiungo global la variabile globale a, definita inizialmente verra'
    modificata
def function():
   global a = "bad"
```

2.3.3 Tipi di dati

Python è tipizzato, solo che, una volta inizializzata una variabile non abbiamo un modo per specializzare un tipo, una volta assegnato un valore viene definito automaticamente (se scrivo tra "", l'oggetto diventerà una stringa). La variabile non ha tipo, è l'oggetto creato che ha un tipo. Il tipo non è legato alla variabile ma all'oggetto!!! I tipi predefiniti di python sono:

- Testo : str
- Numeri : int, float, complex
- Sequenze : list, tuple, range
- Dizionari : dict
- Insiemi : set, frozenset
- booleani : bool
- binari : bytes, bytearray, memoryview

```
x = "ciao" #stringa
x = 20 #int
x = 20.5 #float
x = 1j #numero complesso
x = ["ciao", "come", "stai"] #list
x = ("ciao", "come", "stai") #tuple
x = range(5) #range
x = {"nome" : "Va", "codice" : 17633} #dict
x = {"ciao", "amico"} #set
x = frozenset(x) #frozenset, ovvero un set non modificabile
x = True #bool
x = b"ABCD" #bytes
x = bytearray(5) #bytearray
x = memoryview(bytes(5)) #memoryview
#con type viene stampato il tipo della variabile
type(x)
```

2.3.4 Oggetti, valori, tipi

Qualunque dato in python è un oggetto, ogni oggetto è caratterizzato da : un tipo, un'identità ed un valore. Inoltre:

- il tipo determina le operazioni che l'oggetto supporta;
- l'identità non cambia mai;
- Il tipo determina se un oggetto è mutabile o meno;
- le variabili sono riferimenti ad oggetti.

Python in sè per sè è lento, perchè però ha questo successo? Perchè in un programma sono poche le componenti che devono essere efficienti e sono spesso algoritmi. Gli algoritmi verranno presi da NumPy o altre librerie, che hanno la loro implementazione in C o C++, garantendo massima efficienza.

```
#L'assegnamento crea un oggetto in memoria di tipo Float, a cui la variabile fa
   riferimento
answer = 3.4
print('Type:', type(answer))
print('Identity', id(answer))
print('Value', answer)

#L'assegnamento copia il riferimento dell'oggetto nella nuova variabile
spam = answer
print(spam.equals(answer)) #ritornera' True

 # L'istruzione seguente NON modifica l'oggetto ma crea un nuovo oggetto
    contenente il risultato e ne assegna il riferimento alla variabile: si puo'
    osservare infatti che l'identita' dell'oggetto associato a 'answer' e'
    cambiata, mentre l'identita' di spam e' la stessa
 answer *= 2
 print('Identita nuovo oggetto:', id(answer))
 print('Identita vecchio oggetto :', id(spam))
```

2.3.5 Alcuni oggetti predefiniti

Il loro tipo non fa parte dei tipi di base visti prima, hanno un valore particolare.

- None : None è l'unico oggetto della classe NoneType, è simile a null ma c'è una differenza sostanziale: quando una variabile è a null significa che non ha un riferimento ad un oggetto mentre il None è un oggetto che esiste in memoria (ne esiste solo uno). Ponendo una variabile = None, la variabile punta a quell'oggetto.
- Ellipsis : Sono i puntini di sospensione, è utile in NumPy
- NotImplemented : Metodi numerici e di confronto possono restituire questo valore se non implementano l'operazione per determinati operandi.

```
#Sintassi per oggetto Ellipsis
x = ...
y = Ellipsis
print(x, y)
#Confronto fra stringa e numero, restituisce notImplemented
ret = 'testo'._eq_(42)
```

2.4 Numeri

I numeri possono essere rappresentati attraverso int, float e Complex, possono essere convertiti utilizzando i relativi costruttori (es: float()).

2.5 Stringhe

Tutti i caratteri messi dentro ad una stringa con triplici apici vengono considerati, non succede la stessa cosa con apici doppi o singoli. Non esiste il tipo carattere! E' possibile accedere alle stringhe come se fossero array(liste python).

2.6 Booleani

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
a = 200
b = 33
if b > a:
  print("b > a")
else:
  print("b <= a")
#Valori convertiti a True
x = "VA"
y = 15
print(bool(x), bool(y), bool("abc"), bool(123))print(bool(["rosse", "verde",
   "blu"])) #Valori convertiti a False
print(bool(False), bool(None), bool(0), bool(""))print(bool(()), bool([]),
   bool({}))
```

2.7 Operatori aritmetici, di assegnamento, di identità, di appartenenza e bit a bit

Osservazione sulla divisione : la divisione fra numeri interi avviene attraverso il // (risultato arrotondato all'intero inferiore). Invece usando / ritornerà un float.
Per il resto si trova tutto sulle slide del prof oppure sulla documentazione ufficiale di python.

2.8 Condizioni if..elif..else

```
a = 200
b = 1
if b > a:
   print("b>a")
m = a
   elif b == a:
print("a=b")
m = a
   else:
print("b<a")
m = b
print(m)
# Espressione condizionale
print("b>a") if b>a else print("b<=a")
# Due espressioni condizionali a cascata
print("b>a") if b>a else print("b=a") if b==a else print("b<a")
# Altro esempio
print("b>a" if b>a else "b=a" if b==a else "b<a")
```

2.9 Cicli

```
i = 0
while True:
   i += 1
   if i == 4:
      continue
   print(i)
   if i == 6:
      break

#E' possibile mettere un else fuori da un while, in modo che quando il while
    diventa falso (solo in questo caso) funziona come una sorta di if

i = 0
while i < 6:
   print(i)
   i += 1
else:
   print("Fine del ciclo")
```

```
colori = ["rosso", "verde", "blu"]

for x in colori:
  print(x)

for x in "Visione Artificiale":
  print(x)

for i in range(6):
  print(i)
```

2.10 Liste

```
colori = ['Rosso', 'Verde', 'Blu'] #in alternativa uso il costruttore list()
print(colori)
print(type(colori), len(colori))
print(colori[0]) # Primo elemento
print(colori[-2]) # Penultimo elemento

for c in colori:
  print(c)

colori[1] = 'Grigio' # Modifica di un elemento

colori.append('Verde')
colori.remove('Grigio')

 #fa l'append ma con una lista, di solito si utilizza l'operatore +=, che ci da
    ugualmente la possibilita' di concatenare liste

colori.extend(['Arancione', 'Giallo', 'Viola','Azzurro'])
print(colori)

#metodologia con +=

l1 = [1, 2]

#mi permette di creare la lista [1, 2, 3, 4], l'id della lista rimane invariato,
   perche' la lista e' un oggetto modificabile, quindi non occorre fare puntare
   la lista nuova ad un nuovo oggetto

l1 += [3, 4]

# Esempi di "list comprehension", modo per generare delle liste con la sintassi
   fra quadre

coloriA = [c for c in colori if c[0]=='A'] #for = tali che
quadrati = [x*x for x in range(1,10)] #lista di tutti i quadrati degli x che
   stanno nel range fra 1 e 9
print(coloriA, quadrati)
```

Accesso agli elementi :

- indice fra parentesi quadre
- Possibile usare indice negativo e intervallo di indici
- iterazione con for..in

L'operazione * con le liste ripete la stessa lista n volte.

2.11 Tuple

Le tuple non sono modificabili, possono essere create anche attraverso il costruttore tuple().

```
colori = ('Rosso', 'Verde', 'Blu')
print(colori)
print(type(colori), len(colori))
print(colori[0]) # Primo elemento
print(colori[-2]) # Penultimo elemento

for c in colori:
   print(c)
t0 = (3) # N.B. questa non e' una tupla, e' un int
t0 = () #questa e' invece una tupla
print(type(t0))

t1 = (3,) # Tupla con un solo elemento
t2 = (1,2)
t3 = (4,5,6)
t = t2 + t1 + t3 # Concatenazione di tuple
tm = t2 * 3 # Moltiplicazione di una tupla
print(t, tm)

# Le parentesi si possono omettere
t1 = 3,
t2 = 1,2 # Oppure 1,2,
t3 = 4,5,6 # Oppure 4,5,6,
print(t1,t2,t3)
```

2.12 Slicing

Informazioni:

- $a[i : j]$ restituisce tutti gli elementi di a con indice k tale che $i \leq k < j$;
- $a[:]$ seleziona tutto;
- l'espressione estesa : $a[i : j : s]$ comprende lo step, ovvero dice quanti elementi saltare all'interno del range indicato. Posso avere anche uno step negativo, quindi restituisco una stringa ribaltata.

```
a = "Python!"
print(a[4:6], a[0:2], a[:2]) # on Py Py
print(a[4:len(a)], a[4:100], a[4:]) # on! on! on!
print(a[:-3], a[-3:]) # Pyth on!
print(a[1:-1], a[:]) # ython Python!
print(a[::2]) # Pto!
print(a[1::2]) # yhn
print(a[::-1]) # !nohtyP
print(a[-3::2]) # o!
print(a[-1:-4:-2]) # !o
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(a[-2::-2]) # [8, 6, 4, 2]
#Qui notiamo che e' possibile sfruttare anche l'assegnamento, al posto dei primi
   3 elementi metto gli altri che ho selezionato
a[:3] = a[3:]
print(a) # [4, 5, 6, 7, 8, 9, 4, 5, 6, 7, 8, 9]
a[::2] = [0]*6 #qui sostituisco tutta la lista, a step 2, con 6 zeri
print(a) # [0, 5, 0, 7, 0, 9, 0, 5, 0, 7, 0, 9]
a[3:-3] = []
print(a) # [0, 5, 0, 7, 0, 9]
a[:] = a[::-1]
print(a) # [9, 0, 7, 0, 5, 0]
```

2.13 Insiemi

Collezione non ordinata e non indicizzata, non può contenere duplicati ma possono essere aggiunti o rimossi elementi (non modificati).

```
colori = {'Rosso', 'Verde', 'Blu'}
print(colori)
print(type(colori), len(colori))
print('Rosso' in colori)

for c in colori:
   print(c)

#Differenza fra remove e discard : remove da' errore se l'elemento non esiste,
    discard no, non da' nessun errore
colori.remove('Verde')
colori.discard('Giallo') # nessun errore
colori.add('Azzurro')
colori.update({'Arancione', 'Viola'})
print(colori)

# Esempi di "set comprehension", stessa sintassi usata con le liste ma con le
    graffe
colori_a = {c for c in colori if c[0]=='A'}
quadrati = {x**2 for x in range(1,10)}
print(colori_a, quadrati)
```

2.14 Dizionari

Non capita spesso in questo corso.

```
studenti = {101:"C.Rossi", 103:"M.Bianchi",111:"L.Verdi"}
print(studenti)
print(type(studenti), len(studenti))
if 103 in studenti:
   print(studenti[103])
studenti[112] = "L.Neri"
studenti[115] = "A.Rosa"
for mat in studenti:
   print(mat, studenti[mat])
for mat, nome in studenti.items(): #items() ritorna chiave-valore
   print(mat, nome)
for nome in studenti.values(): #values ritorna solo i valori
   print(nome)
rimosso = studenti.pop(103)
studenti[103] = rimosso
# Esempi di "dictionary comprehension"
studenti_105 = {m: studenti[m] for m in studenti if m<105}
somme = {(k, v): k+v for k in range(4) for v in range(4)}
print(studenti_105, somme)
```

2.15 Funzioni

I parametri sono riferimenti ad oggetti. Per gli oggetti immutabili, l'effetto è sostanzialmente analogo al passaggio per valore. Per quelli mutabili una funzione può modificare il contenuto di un oggetto.

```
def stampa_messaggio(): # Definizione funzione
   print("Addio, e grazie per tutto il pesce!")
stampa_messaggio() # Chiamata della funzione

def calcola(x, y): # Funzione con parametri
   """Questa e' la docstring di calcola"""
   return x * y
print(f"Il risultato e' {calcola(6, 7)}.")

# Funzione che sostituisce elementi di una lista, enumerate restituisce un
    elenco di tuple con indice e valore
def sostituisci(lista, x, y):
   for (indice, valore) in enumerate(lista):
      if valore == x:
         lista[indice] = y

# Versione piu' "pythonic" della stessa funzione
def sostituisci2(lista, x, y):
   lista[:] = [y if v==x else v for v in lista]

l = [1, 2, 3, 1, 2, 3]
sostituisci(l, 2, 0)
sostituisci2(l, 3, -1)
print(l)
```

2.16 Parametri delle funzioni

```
def calcola(x, y, z = 1, k = 0):
   return (x * y) / z + k
print(calcola(2,3), calcola(2,3,3), calcola(2,3,3,-2))
print(calcola(y=1,x=3), calcola(2,3,k=2),calcola(k=1,x=2,y=3,z=1))

def prodotto(x, *altri_fattori):
   p = x
   for f in altri_fattori:
      p *= f
   return p
print(prodotto(2), prodotto(6,7), prodotto(2,2,2,2,2))

#* = insieme di parametri senza nome (in questo caso gli argomenti
    dell'ipotetico corso
#** = dizionario di parametri con nome
def Esame(corso, *argomenti, **studenti):
   print("Corso:", corso)
   print("Argomenti:", end=' ')
```

```
for a in argomenti:
   print(a, end=', ')
   print("\nStudenti:")
for mat in studenti:
   print(mat, studenti[mat])
Esame("Visione Artificiale", " Python", "NumPy", "OpenCV",M101="C.Rossi",
    M103="M.Bianchi", M111="L.Verdi")
```

2.17 Unpacking

Lo useremo a volte nel passaggio dei parametri.

```
def calcola(a, b, c):
   return a + b + c

parametri = [4, 18, 20]
print(calcola(*parametri)) #spacchetta la lista, questo mi permette di passare
    tre valori separati come argomento della funzione

a = ["Python", "NumPy", "OpenCV"]
s = {"M101":"C.Rossi",
   "M103":"M.Bianchi",
   "M111":"L.Verdi"}
Esame("Visione Artificiale", *a, **s)

x, y, z = 2, 3, 4 # (x, y, z) = (2, 3, 4)

a1, a2, a3 = a
print(a1, a2, a3)
a1, *r = a #questo serve per spacchettare una lista della quale non ne conosco
    le dimensioni
print(a1, r)
primo, secondo, *altri, ultimo = range(10)
print(primo, secondo, ultimo, altri)

x, y, _* = a #questo vuol dire che a deve contenere almeno 2 elementi, il primo
    va in x, il secondo va in y e il resto non mi interessa
```

2.18 Funzioni come oggetti e lambda

```
def prodotto(x, y):
   """Restituisce il prodotto di x e y."""
   return x * y

print(type(prodotto)) # <class 'function'>
print(prodotto.__name__) # prodottoprint(prodotto.__doc__) # la docstring

def esegui(f, x, y):
   """Esegue la funzione f su x e y."""
   return f(x, y)

print(esegui(prodotto, 2, 3))

f = lambda x, y: x ** y
print(type(f)) # <class 'function'>
print(f.__name__) # <lambda>
print(f.__doc__) # None
print(esegui(f, 4, 2))
print(esegui(lambda x, y: x // y, 9, 2))
```

2.19 Funzioni predefinite

```
s = "Python"
a = enumerate(s)
print(list(a))
# [(0,'P'), (1,'y'), (2,'t'),
(3,'h'), (4,'o'), (5,'n')]

n = [ord(c) for c in s]
print(list(n), sum(n), min(n), max(n))# [80, 121, 116, 104, 111, 110] 642 80 121

print(sorted(n))
# [80, 104, 110, 111, 116, 121]

z = zip(s, n)
print(list(z))
# [('P',80), ('y',121), ('t',116),
('h',104), ('o',111), ('n',110)]
```

2.20 Moduli python

```
# Modulo fib.py
def fibonacci(n): # serie di Fibonacci fino a n
   result = []
   a, b = 0, 1
   while a < n:
      result.append(a)
      a, b = b, a+b
   return result

# Esempi di importazione
import fib
print(fib.fibonacci(90))

import fib as f
print(f.fibonacci(90))

from fib import fibonacci
print(fibonacci(90))

from fib import fibonacci as f
print(f(90))
```

Capitolo 3

NumPy

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 4

Immagini

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 5

Calibrazione

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 6

Filtri

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 7

Analisi

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 8

Movimento

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```

Capitolo 9

Riconoscimento

```
//Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
   public void paintComponent(Graphics g) {
      g.drawString("Hello, world!", 65, 95);
   }
}
```