



**UNIVERSITÀ DI PISA**

**Dipartimento di Ingegneria dell'Informazione**

**Realizzazione di un prototipo per il riconoscimento  
umano attraverso immagini basato sulla  
piattaforma Intel Movidius**

**A.A 2018-2019**

***Marco Pettorali***

# Sommario

Introduzione .....	2
Architettura del sistema .....	4
Installazione dell'ambiente .....	6
Premesse .....	6
Prerequisiti.....	7
Installazione del Neural Compute SDK .....	8
Installazione di openCV .....	10
Programmi di esempio di Person Detection .....	12
Installazione di ncappzoo .....	14
Installazione di NCS-Pi-Stream .....	15
Installazione di SSD_MobileNet.....	17
Implementazione del servizio di controllo.....	18
Descrizione .....	18
Scelte implementative .....	18
Variabili globali .....	18
Inferenze ed elaborazioni sulle immagini.....	20
Server Web .....	26
Inizializzazione dell'applicazione .....	31
Programma in esecuzione .....	33
Avvio del servizio .....	33
Esecuzione: nessun umano rilevato.....	34
Esecuzione: umano rilevato.....	35
Conclusioni .....	36

# Introduzione

Negli ultimi anni si sta affermando una nuova realtà industriale, comunemente nota con il nome di Industria 4.0. Questo nuovo paradigma prevede l'utilizzo intensivo di tecnologie quali ad esempio, analisi di big data, cloud computing, intelligenza artificiale, sistemi avanzati di automazione, controllo remoto degli apparati industriali.

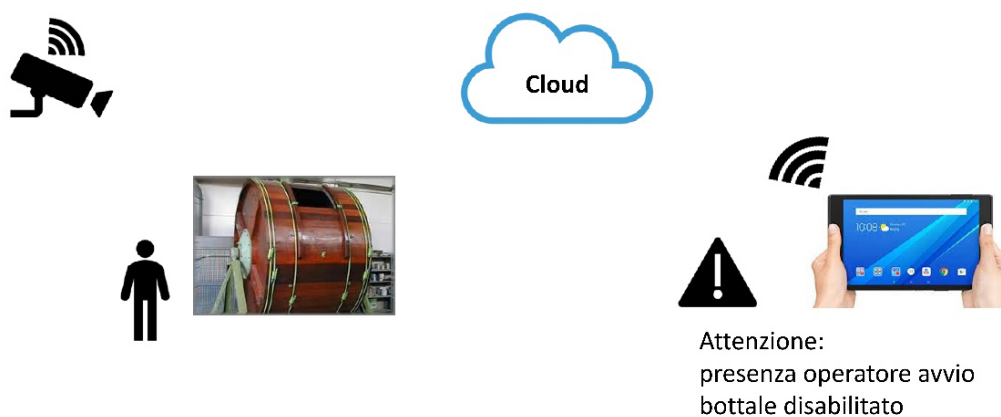
Il concetto di Industria 4.0 risponde dunque alla necessità di avere informazioni e servizi distribuiti per ottimizzare le scelte e analizzare i processi industriali con accuratezza sempre maggiore: per fare ciò, è necessario estendere il concetto di “rete di calcolatori”, fondamento della rete Internet, a “rete di oggetti”, come introdotto dal concetto di “Internet delle cose” (Internet of Things) in cui ogni elemento, ogni componente del processo industriale è visto come nodo di una rete, capace di raccogliere e inviare segnali per essere successivamente monitorati, controllati, elaborati.

Grazie a queste tecnologie, le aziende e le industrie possono avere un maggior controllo sui processi industriali, sia in termini di produttività che di sicurezza, segnando così una profonda spaccatura con il passato sul piano tecnico per la gestione dell'impresa. Il nome stesso ci suggerisce, infatti, come questa possa essere considerata la Quarta Rivoluzione Industriale, figlia della Terza Rivoluzione Industriale, che negli anni '60 ha preso piede con l'introduzione dei calcolatori all'interno delle industrie, e della Internet Revolution, che ha interessato il mondo a partire dagli anni '90 con la rete Internet.

Proprio nell'ambito dell'industria 4.0 si inserisce il lavoro svolto in questa tesi. Si vuole realizzare, infatti, un prototipo per verificare la fattibilità di un sistema di Person Detection per il controllo e il monitoraggio della sala macchine di un impianto conciario del Polo Tecnologico Conciario di Santa Croce sull'Arno (PI). In particolare, si vuole ottenere un sistema per impedire, per motivi di sicurezza, il riavvio delle macchine adibite alla concia delle pelli (“bottali”) se nei dintorni delle stesse venisse rilevata una presenza umana.

Questo lavoro non è incentrato sulle operazioni tra macchinario e sistema di controllo e pertanto, dell'interfaccia tra macchinario e sistema di controllo, verrà soltanto indicato il punto nel codice di controllo in cui inserire le istruzioni per il riavvio del bottale.

Come riassunto dal seguente schema, l'utente può quindi collegarsi al server web messo a disposizione dal servizio tramite un qualsiasi dispositivo connesso ad internet e, tramite l'opportuna interfaccia, dare il comando di riavvio del bottale. Il sistema provvederà a verificare la presenza umana e comunicherà all'utente il risultato dell'analisi.



# Architettura del sistema

Per realizzare il servizio, si utilizzerà un Raspberry Pi 3B+, la scheda Movidius (un neural stick prodotto dalla Intel) e una webcam, per quanto riguarda il comparto hardware; il software consisterà invece dalla rete neurale SSD MobileNet di Google e lo script Python “person-detector-web-server.py”, il software di controllo da me scritto che, tramite alcune chiamate di funzione a Movidius, implementa l’algoritmo di riconoscimento di oggetti e presenza umana. L’applicazione, inoltre, offre un server web per visualizzare i risultati da remoto.

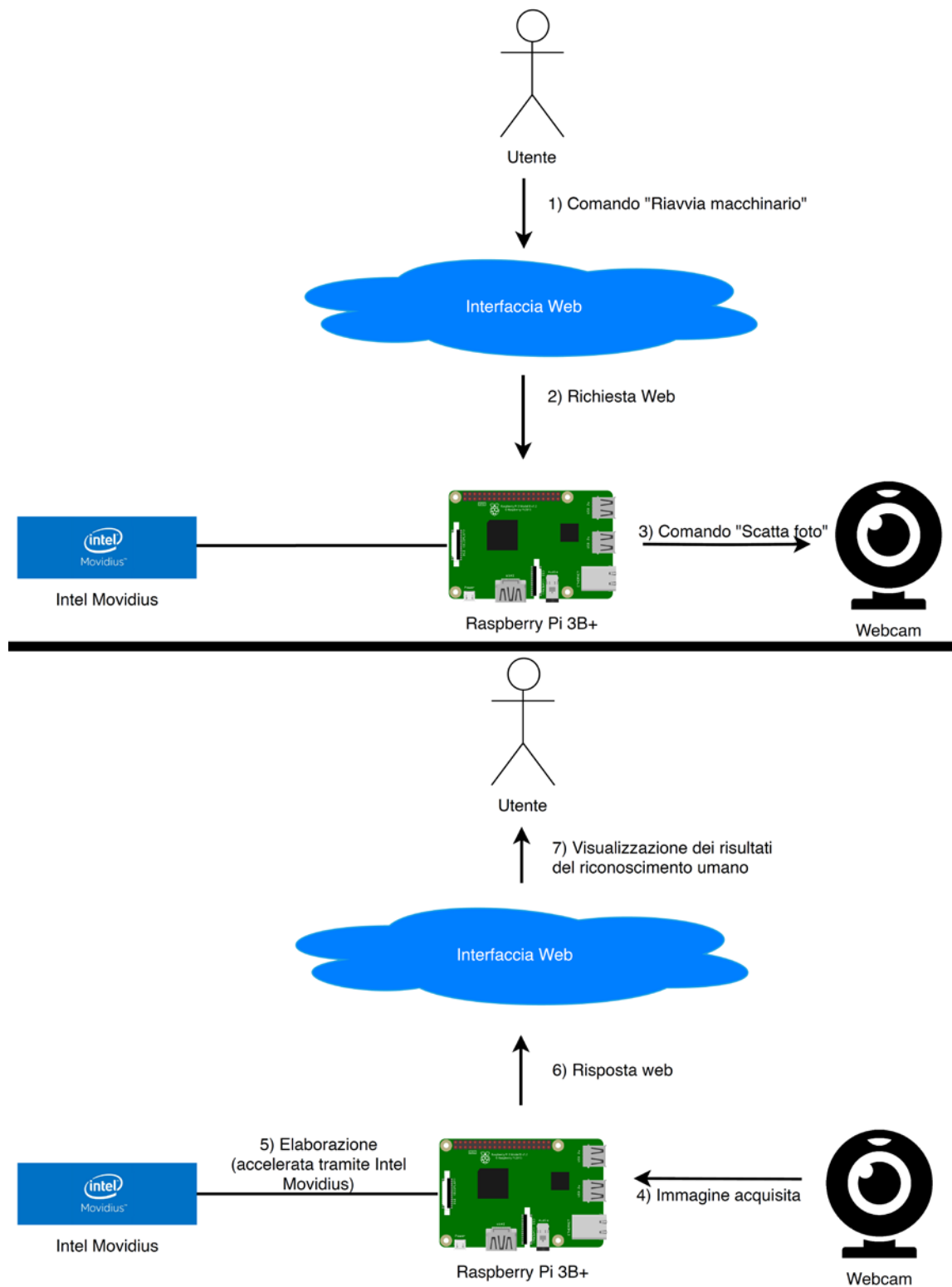
È importante sottolineare che questa trattazione è valida soltanto per la prima versione di Movidius, altresì chiamata NCS1. Lo SDK che verrà ampiamente utilizzato nel software, infatti, non supporta la nuova versione del Neural Compute Stick, NCS2.

Questa è una foto dell’hardware che necessita il sistema:



La webcam (collegata al Raspberry Pi) raccoglie le immagini che riprendono il bottale, per essere successivamente elaborate. Da remoto, connettendosi al server web sul Raspberry Pi, si possono vedere le immagini elaborate. La webcam utilizzata durante le mie prove ha una risoluzione di 1.3 Mega pixel.

Il seguente schema mostra le interazioni tra utente e sistema, dettagliato nelle sue componenti hardware:



# Installazione dell'ambiente

## Premesse

In questa sezione, spiegherò punto per punto il procedimento per installare il materiale necessario al corretto funzionamento di “person-detector-web-server.py”; spiegherò anche come installare ed eseguire ad alcuni programmi di esempio, forniti da Movidius, che implementano semplici servizi di live streaming e object detection.

È consigliabile utilizzare la versione Lite del sistema operativo Raspbian Stretch, in quanto non include software aggiuntivo eccetto quello necessario a questo procedimento. Si supporrà quindi di avere installato sul Raspberry Pi la suddetta versione, scaricabile da questo indirizzo:

<https://www.raspberrypi.org/downloads/raspbian/>

In questo modo avremo un'installazione di base del sistema operativo, che non include applicazioni e pacchetti non utili per la realizzazione di un riconoscitore di presenza umana. Per collegarsi al Raspberry Pi utilizzeremo il protocollo ssh; per configurare Raspberry Pi come server ssh basterà creare un file vuoto, senza estensione, dal nome ‘ssh’ nella scheda SD, insieme agli altri file del sistema operativo: questo file verrà letto da Raspbian, ignorato il contenuto, e successivamente eliminato. Questa operazione andrà svolta soltanto al primo avvio del Raspberry Pi, dopo il quale non sarà più necessaria.

Il software da installare è composto dal Neural Compute SDK e dalle relative API per Python, per quanto riguarda l'interfaccia con Movidius, OpenCV per l'acquisizione e la visualizzazione delle immagini acquisite dalla webcam, e da ncappzoo, una raccolta di programmi di esempio forniti da Movidius che include anche una versione semplificata del framework Caffe, necessario per la creare i file contenenti la rete neurale in un formato adatto a Raspberry Pi.

## Prerequisiti

Avremo inizialmente bisogno di installare alcuni pacchetti e dipendenze sui quali i programmi di interfaccia con Movidius si appoggiano. Dopo aver aggiornato le package lists con il comando `sudo apt-get update`, possiamo installare i tool ‘git’ e ‘pip3’, che saranno estremamente utili nelle fasi successive dell’installazione. Si noti che non sarà necessario installare i pacchetti di python3, perché questi sono già integrati in Raspbian Stretch.

```
sudo apt-get update
```

```
sudo apt-get install git
```

```
sudo apt-get install python3-pip
```



## Installazione del Neural Compute SDK

Prima di proseguire, è consigliabile aumentare la dimensione dello swap file del Raspberry Pi. Per farlo è sufficiente modificare la variabile `CONF_SWAPFILE` del file `/etc/dphys-swapfile`, impostandola al valore 1024; salviamo anche il file originale in una copia di backup per poterlo riusare una volta terminata l'installazione.

```
echo CONF_SWAPSIZE=1024 > dphys-swapfile-enlarged  
sudo mv /etc/dphys-swapfile /etc/dphys-swapfile-original  
sudo mv dphys-swapfile-enlarged /etc/dphys-swapfile  
sudo /etc/init.d/dphys-swapfile restart
```

Adesso proseguiamo con l'installazione di `ncsdk2`, l'SDK di Movidius. Notare l'opzione per cambiare branch dalla repository Github in fase di cloning `-b ncsdk2`: sul branch master della repository, attualmente (febbraio '19) c'è `ncsdk`, ovvero la versione precedente e deprecata dell'SDK. E' già stato annunciato dalla Intel sul file `README.md` della repository, che a breve `ncsdk2` verrà spostato sul master branch, quindi il comando di cloning riportato qua sotto potrebbe non essere più valido.

Dopo l'operazione di cloning, assicurarsi che lo Movidius sia correttamente collegato in una porta USB del Raspberry Pi, e proseguire con il comando `make install`. Oltre ad installare `ncsdk2`, il comando installerà anche la versione 2.08 delle API per Python, chiamate `NCAPI2`.

```
cd ~  
sudo git clone -b ncsdk2 https://github.com/movidius/ncsdk.git  
cd ncsdk  
sudo make install
```

Per controllare che l'installazione di `ncsdk2` sia avvenuta con successo, eseguiamo il programma d'esempio fornito all'interno della repository, senza scollegare Movidius dal Raspberry Pi.

```
cd examples/apps/hello_ncs_py
```

```
make run
```

Se ci sono stati errori durante l'esecuzione, si consiglia di rimuovere la cartella 'ncsdk' in cui il comando git clona la repository, di ripetere il comando di cloning e provare a reinstallare nuovamente ncsdk2.

## Installazione di openCV

La repository su Github contiene anche uno script per l'installazione di openCV, un framework molto utile per la raccolta, l'elaborazione e la visualizzazione delle immagini, e viene utilizzato da molte applicazioni di object detection. Per installarle basta dare il comando `./install-opencv.sh`. L'installazione può durare molto tempo, anche due ore in caso di connessione lenta.

```
cd ~/ncsdk
./install-opencv.sh
```

Dall'installazione di openCV mancano, però, alcune dipendenze fondamentali senza le quali python3 non riconoscerà le librerie di openCV. Questo sembra essere un problema di compatibilità con Raspberry Pi, in quanto molti utenti Raspberry, sui forum ufficiali, lamentano problemi di mancanza di dipendenze di vario genere, problemi non sperimentati dagli altri utenti di openCV.

Infatti, se all'interno della console di python3, si proverà a dare il comando `'import cv2'`, questo puntualmente terminerà con errore, indicando ogni volta quale file sta cercando di caricare e non trova. Interpretando questi messaggi di errore, e svolgendo alcune ricerche sul web, ho individuato ed installato i seguenti pacchetti mancanti:

```
pip3 install opencv-python
sudo apt-get install libgstreamer1.0-0
sudo apt-get install libqtgui4
sudo apt-get install libqt4-test
pip3 install -U numpy==1.15.4
```

Adesso possiamo finalmente testare l'installazione di openCV e delle librerie di python3 ad esso collegate. Aprendo la console di python con il comando `python3` e inserendo il comando `import cv`, non dovremmo vedere adesso nessun errore, e l'importazione termina con successo.

Se, invece, si fosse verificato un errore, è necessario interpretare il messaggio nell'ultima riga dello stack trace per cercare di capire quale è il file mancante.

Successivamente è consigliabile fare una ricerca su Internet per capire in quale pacchetto è contenuto il file mancante. Consiglio vivamente di usare l'elenco ufficiale dei pacchetti stable di Debian, al link <https://packages.debian.org/stable>. In alto a destra, e a sinistra della barra di ricerca, selezionare la voce 'package contents', quindi digitare nella barra di ricerca il nome del file ricercato. Premendo il pulsante 'Search', apparirà una lista di pacchetti nel quale si trova il file in questione. Sceglierne uno, ponendo attenzione sull'architettura del pacchetto: quella di Raspberry Pi 3B+ è 'arm64'. Se la query dovesse terminare con esito nullo, provare nei packages in fase di testing, al link <https://packages.debian.org/testing>.

Se questo metodo dovesse fallire, fare una ricerca tra i forum ufficiali di Movidius (<https://ncsforum.movidius.com/>), openCV (<http://answers.opencv.org/questions/>), e Raspberry Pi (<https://www.raspberrypi.org/forums/>), oppure postare un quesito sui suddetti forum. Provare anche su altri forum non ufficiali (uno su tutti Stack Overflow <http://stackoverflow.com>) in cui contattare altri utenti che hanno affrontato lo stesso problema o problemi simili.

Se in precedenza è stato modificato il file /etc/dphys-swapfile, è consigliabile riportare la variabile CONF\_SWAPFILE reimpostandola al valore di default 100.

```
sudo mv /etc/dphys-swapfile /etc/dphys-swapfile-enlarged
sudo mv /etc/dphys-swapfile-original /etc/dphys-swapfile
sudo /etc/init.d/dphys-swapfile restart
```

## Programmi di esempio di Person Detection

Andremo ad utilizzare due software, indipendenti l'uno dall'altro, entrambi basate su MobileNet SSD, un'implementazione delle reti MobileNets sviluppate da Google, compatibili con il framework Caffe.

Le reti MobileNets (presentate nell'articolo <https://arxiv.org/pdf/1704.04861.pdf>), sono una classe di reti neurali progettate per essere estremamente veloci, precise e a basso consumo, indicate per installazioni embedded quali il Raspberry Pi.

Caffe (<http://caffe.berkeleyvision.org/>) è, invece, un framework sviluppato dalla Berkeley AI Research, che permette la descrizione, l'ottimizzazione, il training e il testing di reti neurali.

Il primo che installeremo è NCS-Pi-Stream, realizzato e caricato su GitHub da HanYangZhao. E' un software di Live Object Detection, la SSD MobileNet che usa è stata allenata su un set di immagini in modo tale che categorizzi gli oggetti individuati nelle 20 categorie proposte per la prima volta nella VOC2007 Challenge, organizzata da PASCAL Visual Object Classes (<http://host.robots.ox.ac.uk/pascal/VOC/>), raccolte in quattro macro-classi: Person (che è quella che ci interessa), Animal, Vehicle, Indoor.

NCS-Pi-Stream sfrutta openCV per l'acquisizione delle immagini dalla webcam e per la successiva visualizzazione di riquadri (box) attorno agli oggetti trovati, sfruttando le informazioni in output fornite da MobileNets e integra un semplice web server che visualizza in diretta le immagini analizzate ed elaborate in formato .mjpg, come sequenza di immagini .jpg.

Il secondo è invece il programma 'SSD\_MobileNet/run.py', incluso nella repository GitHub 'ncappzoo' che contiene tutti gli esempi ufficiali (sviluppati da Movidius Ltd.) per la dimostrazione del Neural Compute Stick. In particolare, questo programma funge da dimostrazione di MobileNet SSD e, a differenza di NCS-Pi-Stream, non offre un'interfaccia web, ma stampa il risultato a video sull'emulatore di terminale, e non scatta un'immagine da una webcam per analizzarla, ma di base sfrutta un'immagine di esempio contenuta nella repository Ncappzoo. Ugualmente a NCS-Pi-Stream, include una rete allenata secondo le categorie del VOC2007.

Basandomi su quest'ultimo esempio, ho realizzato lo script “person-detector-web-server.py”, che integra una semplice interfaccia web e l'acquisizione di frame provenienti direttamente dalla webcam collegata al Raspberry Pi.

## Installazione di ncappzoo

L'installazione di ncappzoo è necessaria perché installa una versione di base del framework Caffe, delle mvnc\_simple\_api (versione semplificata e ridotta di ncsdk1) su cui alcuni esempi si basano, includendo alcuni file necessari per l'esecuzione dei programmi che vogliamo utilizzare.

Innanzitutto, creiamo una cartella in cui posizionare ncappzoo:

```
cd ~  
  
mkdir -p workspace  
  
cd workspace
```

Cloniamo da GitHub la repository di ncappzoo, e creiamo tutte le dipendenze mancanti con il comando `make all`:

```
git clone -b ncsdk2 https://github.com/movidius/ncappzoo.git  
  
cd ncappzoo  
  
make all
```

## Installazione di NCS-Pi-Stream

Adesso possiamo proseguire con l'installazione dell'esempio NCS-Pi-Stream. Dopo essersi posizionati nella cartella 'apps' di ncappzoo, cloniamo da GitHub la repository:

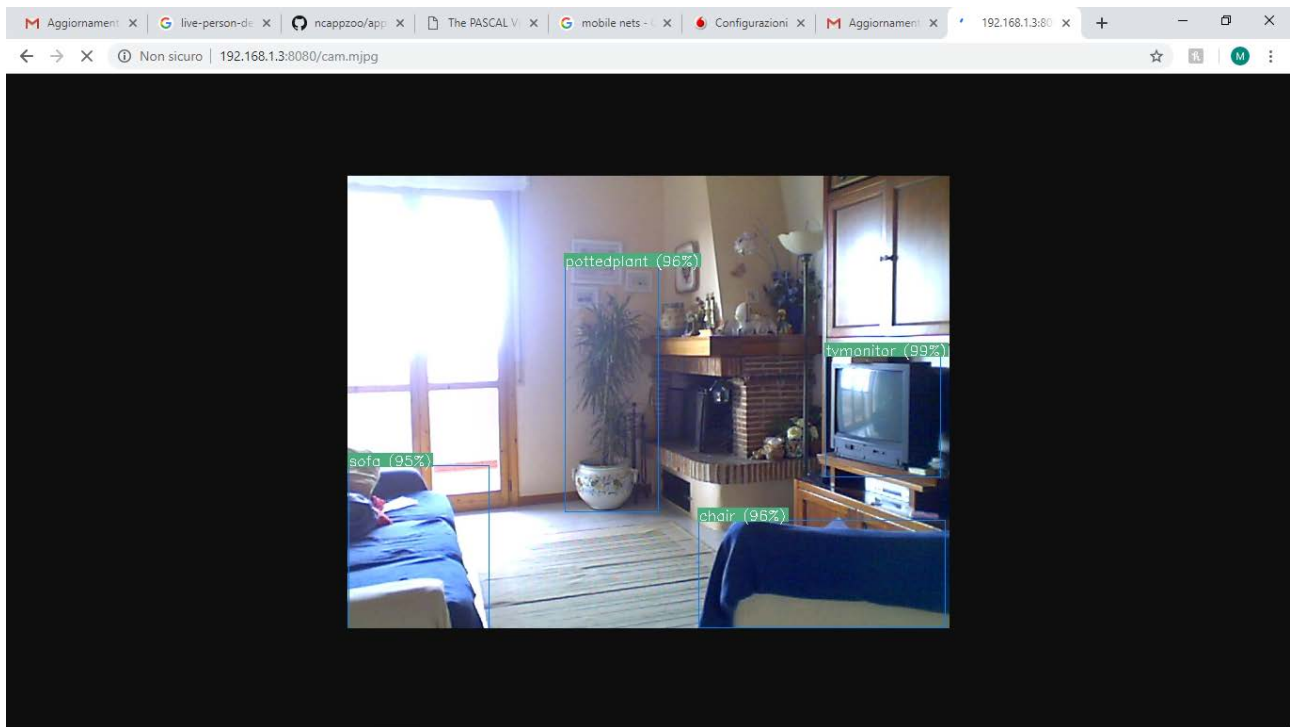
```
cd ~/workspace/ncappzoo/apps/  
git clone https://github.com/HanYangZhao/NCS-Pi-Stream.git  
cd NCS-Pi-Stream/models
```

Poiché il file graph, utilizzato da ncsdk2 per caricare la rete neurale all'interno di Movidius, è contenuto nella repository sembra essere corrotto e non riconosciuto da Movidius, è necessario ricompilarlo tramite il comando 'mvNCCompile' fornito nel ncsdk. Questa operazione è possibile perché l'autore della repository, ha incluso in una cartella i file .prototxt e .caffemodel per la ricompilazione del file graph. In particolare il file .prototxt rappresenta una descrizione della rete neurale MobileNet conforme al framework Caffe, mentre il file .caffemodel contiene i pesi della rete ottenuti dopo le iterazioni di training eseguite con Caffe stesso.

```
mvNCCompile MobileNetSSD_deploy.prototxt -s 12 -w MobileNetSSD_deploy.caffemodel  
  
mv graph ../graph/mobilenetgraph
```



Avviando il programma con il comando `python3 streamer_ncs.py`, e visitando la pagina `http://YOUR_PI_IP:8080/cam.mjpg`, dove al posto di `YOUR_PI_IP` bisogna inserire l'indirizzo IP al quale è collegato il Raspberry Pi, si potranno vedere in diretta le immagini elaborate.



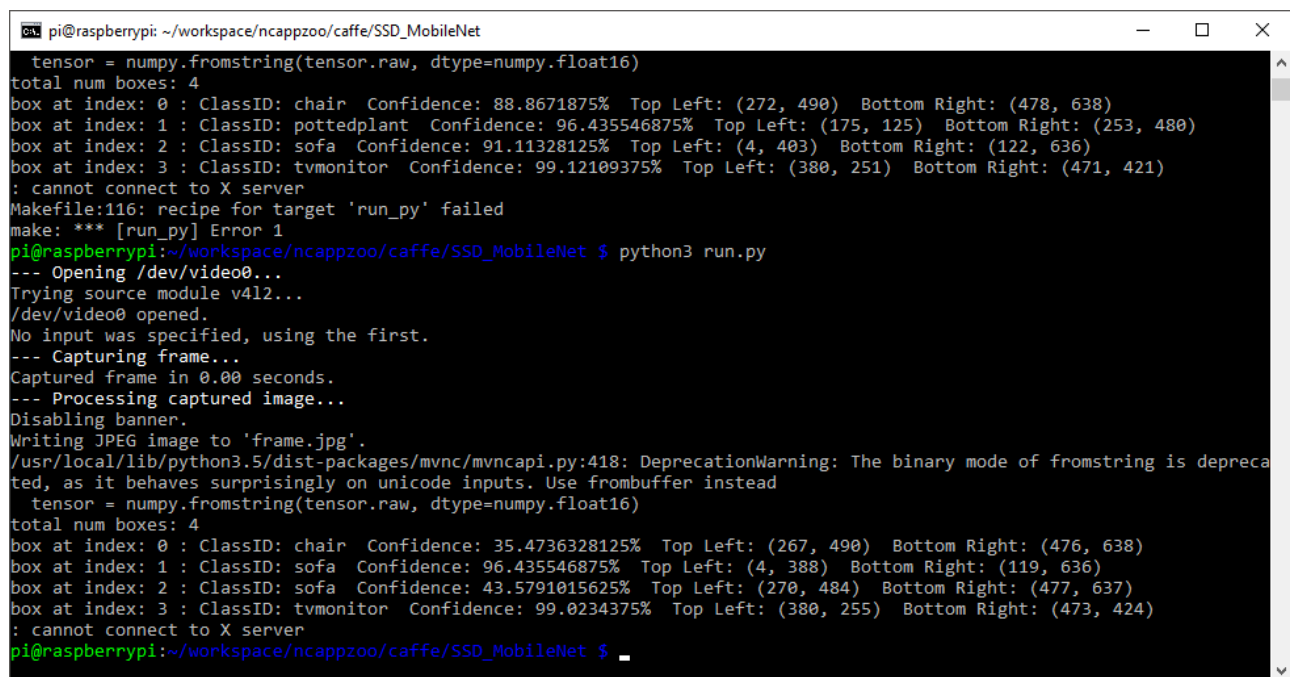
## Installazione di SSD\_MobileNet

I file per l'installazione di SSD\_MobileNet sono già inclusi nella cartella `ncappzoo/caffe/SSD_MobileNet`. Sarà sufficiente collocarsi in questa cartella e dare il comando `make run`, che recupererà da Internet i file mancanti, e successivamente eseguirà il programma.

```
cd ~/workspace/ncappzoo/caffe/SSD_MobileNet
```

```
make run
```

Prima di eseguire il programma, aprire il file `run.py` con un editor di testo per modificare la variabile dove è memorizzato il percorso dell'immagine da analizzare, da sostituire con il percorso della propria immagine da elaborare.



```
pi@raspberrypi: ~/workspace/ncappzoo/caffe/SSD_MobileNet
tensor = numpy.fromstring(tensor.raw, dtype=numpy.float16)
total num boxes: 4
box at index: 0 : ClassID: chair Confidence: 88.8671875% Top Left: (272, 490) Bottom Right: (478, 638)
box at index: 1 : ClassID: pottedplant Confidence: 96.435546875% Top Left: (175, 125) Bottom Right: (253, 480)
box at index: 2 : ClassID: sofa Confidence: 91.11328125% Top Left: (4, 403) Bottom Right: (122, 636)
box at index: 3 : ClassID: tvmonitor Confidence: 99.12109375% Top Left: (380, 251) Bottom Right: (471, 421)
: cannot connect to X server
Makefile:116: recipe for target 'run_py' failed
make: *** [run_py] Error 1
pi@raspberrypi:~/workspace/ncappzoo/caffe/SSD_MobileNet $ python3 run.py
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Disabling banner.
Writing JPEG image to 'frame.jpg'.
/usr/local/lib/python3.5/dist-packages/mvnc/mvncapi.py:418: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
tensor = numpy.fromstring(tensor.raw, dtype=numpy.float16)
total num boxes: 4
box at index: 0 : ClassID: chair Confidence: 35.4736328125% Top Left: (267, 490) Bottom Right: (476, 638)
box at index: 1 : ClassID: sofa Confidence: 96.435546875% Top Left: (4, 388) Bottom Right: (119, 636)
box at index: 2 : ClassID: sofa Confidence: 43.5791015625% Top Left: (270, 484) Bottom Right: (477, 637)
box at index: 3 : ClassID: tvmonitor Confidence: 99.0234375% Top Left: (380, 255) Bottom Right: (473, 424)
: cannot connect to X server
pi@raspberrypi:~/workspace/ncappzoo/caffe/SSD_MobileNet $
```

# Implementazione del servizio di controllo

## Descrizione

Il software “person-detector-web-server.py”, implementa un algoritmo di person detection e integra un server web che mette a disposizione dell’utente i comandi per riavviare da remoto i macchinari. Cliccando sul bottone di riavvio, il sistema scatterà alcune foto da analizzare per cercare un’eventuale presenza umana; se questa viene trovata, il sistema non riavvierà la macchina. In ogni caso, l’interfaccia web mostra all’utente il fotogramma scattato (comprendente gli eventuali box attorno agli oggetti riconosciuti), la lista degli oggetti riconosciuti e la loro posizione nel fotogramma, e l’esito della ricerca di presenza umana.

Il codice, in Python 3, è basato su ncsdk2 per l’interfaccia con la neural stick Movidius e su openCV per l’acquisizione delle immagini e la visualizzazione dei box attorno agli oggetti individuati. La rete neurale utilizzata è MobileNet SSD di Google.

## Scelte implementative

### *Variabili globali*

All’inizio del codice, sono dichiarate alcune variabili globali di utilità.

La variabile PORTA, indica su quale porta verrà aperto il socket per offrire il servizio web.

```
PORTA = 8080
```

MobileNet SSD comunica, per ogni oggetto identificato, la percentuale di attendibilità del riconoscimento. Per rendere pulito l’output del web server, è opportuno non visualizzare oggetti di cui il sistema non assicura la presenza sopra un certo valore (espresso in percentuale): la variabile SOGLIA indica, appunto, la soglia minima per la visualizzazione di un oggetto tra i risultati finali.

```
SOGLIA = 60
```

Poiché il sistema di riconoscimento non è perfetto può capitare che, dato in input alla rete neurale lo stesso frame, il numero, il tipo e anche la percentuale di attendibilità degli oggetti identificati, vari sensibilmente. Per rimediare a questo problema, al momento della richiesta dell'utente, non viene scattata una sola foto, ma un numero massimo di fotografie contenuto nella variabile globale NUM\_FOTO, impostata di default a 20, ed ognuna viene analizzata: appena in una di queste viene riconosciuta la presenza umana, non vengono scattati più frame e viene mostrato all'utente l'esito.

```
NUM_FOTO = 20
```

Abbiamo anche bisogno di definire alcuni oggetti globali per essere recuperati dalle funzioni che gestiscono le richieste al web server, e che quindi avvengono in modo asincrono.

```
CAMERA = None
```

```
IMMAGINE = None
```

```
GRAPH = None
```

```
INPUT_FIFO = None
```

```
OUTPUT_FIFO = None
```

La variabile LABELS è un array contenente tutte le etichette rappresentanti i tipi degli oggetti riconosciuti dalla rete neurale, definiti dallo standard de facto introdotto dalla PASCAL Visual Object Classes Challenge del 2007 (VOC2007). Dato l'environment di utilizzo dell'applicazione, le etichette sono state tradotte in italiano. Il tipo di un oggetto è l'indice della propria etichetta all'interno di questo array.

```
LABELS = ('sfondo', 'aereo', 'bicicletta', 'uccello', 'barca',  
          'bottiglia', 'autobus', 'automobile', 'gatto', 'sedia',  
          'mucca', 'tavolo', 'scrivania', 'cane', 'cavallo',  
          'motocicletta', 'persona', 'pianta',  
          'pecora', 'divano', 'poltrona', 'treno', 'schermo')
```

## *Inferenze ed elaborazioni sulle immagini*

Questa sezione del codice comprende le funzioni per gestire le inferenze sulle immagini e la successiva elaborazione per la visualizzazione dei box attorno agli oggetti riconosciuti.

Prima di passare le immagini a MobileNet SSD, è necessario preprocessarle, riducendole e scalandole: questa operazione di normalizzazione delle immagini è necessaria per incrementare l'affidabilità dei dati di output. Essendo le tecniche di normalizzazione un argomento molto vasto e complesso, mi sono limitato a riportare nel mio programma l'algoritmo di preprocessing dell'esempio "SSD\_MobileNet/run.py", che adatta le immagini in input alle dimensioni richieste dalla rete MobileNet SSD.

```
def preprocessa_immagine(src):  
    LARGHEZZA = 300  
    ALTEZZA = 300  
    img = cv2.resize(src, (LARGHEZZA, ALTEZZA))
```

In questo punto della funzione, l'array rappresentante l'immagine in formato RGB (a valori tra 0 e 255 per ciascun colore), viene riportato come array di valori tra -1 e +1. Si noti che  $0.007843$  è  $1/127.5$ , quindi se  $n \in [0, 255]$ , dopo la prima operazione  $n - 127.5 \in [-127.5, 127.5]$ ; infine  $\frac{n-127.5}{127.5} \in [-1, 1]$ .

```
img = img - 127.5  
img = img * 0.007843  
return img
```

Possiamo dunque definire la funzione che avrà il compito di eseguire l'inferenza sulle immagini passate come argomento. Con "inferenza" si intende il processo di riconoscimento di oggetti a partire da una immagine tramite algoritmi. Si noti che questi algoritmi sono eseguiti tutti internamente allo stick Movidius, che fornisce uno speed-up significativo al sistema di riconoscimento, e vengono semplicemente invocati dal codice usando le NCAPID2 per Python 3 e ncsdk2.

```
def esegui_inferenza(immagine):
```

Innanzitutto è necessario preprocessare l'immagine in input per ridimensionarla.

```
immagine_ridimensionata = preprocessa_immagine(immagine)
```

Successivamente si può procedere con la chiamata a Movidius per passare l'immagine da elaborare. Questa viene passata nel campo `tensor` del metodo `queue_inference_with_fifo_elem`, che scrive il tensore nella coda fifo di input:

```
GRAPH.queue_inference_with_fifo_elem(INPUT_FIFO, OUTPUT_FIFO, immagine_ridimensionata.astype(numpy.float32), None)
```

Infine si recuperano i risultati dell'elaborazione dalla coda fifo di output:

```
output, dati = OUTPUT_FIFO.read_elem()
```

Adesso segue la fase di interpretazione e recupero dell'output della rete neurale. In particolare, l'output di MobileNet SSD è strutturato nel seguente modo:

- Il primo valore indica il numero di oggetti riconosciuti dalla rete nell'immagine
- I seguenti 6 sono inutilizzati
- I successivi sono organizzati a gruppi di 7, e sono campi relativi a ciascun oggetto riconosciuto
  - Il primo, con indice 0, è inutilizzato;
  - Il secondo (indice 1) indica il tipo dell'oggetto riconosciuto;
  - Il valore all'indice 2 indica invece l'attendibilità dell'inferenza (espresso in percentuale);
  - Gli indici 3 e 4 indicano la posizione in percentuale rispettivamente dal bordo sinistro e alto, del vertice in alto a sinistra del box contenente l'oggetto riconosciuto all'interno dell'immagine;
  - Gli indici 5 e 6 indicano la posizione del vertice in basso a destra del box.

Dopo aver recuperato il numero di oggetti riconosciuti con l'istruzione:

```
num_oggetti_riconosciuti = int(output[0])
```

eseguiamo un ciclo per recuperare tutti i campi dei vari oggetti, memorizzandoli in una stringa 'ret\_str' per essere poi passata al web server che la mostrerà nell'interfaccia web.

```
ret_str = ' '  
for i in range(num_oggetti_riconosciuti):  
    base = 7 + i * 7
```

Adesso dobbiamo gestire una potenziale situazione di errore, ovvero quando i campi hanno valore non finito (infinito, NaN, None...), situazione che si verifica molto raramente se qualche dato si corrompe durante l'elaborazione. Durante le mie prove non ho mai incontrato questo errore, questo segmento di codice è identico a quello del file "SSD\_MobileNet/run.py".

```
if (not numpy.isfinite(output[base]) or  
    not numpy.isfinite(output[base + 1]) or  
    not numpy.isfinite(output[base + 2]) or  
    not numpy.isfinite(output[base + 3]) or  
    not numpy.isfinite(output[base + 4]) or  
    not numpy.isfinite(output[base + 5]) or  
    not numpy.isfinite(output[base + 6])):  
    print('il box all\'indice: ' + str(i) + ' ha dati di output non  
definiti')  
    continue
```

Prima di recuperare i campi dell'oggetto in questione, ricaviamone innanzitutto la percentuale di attendibilità, e ignoriamolo se ha attendibilità sotto la soglia desiderata. Questa operazione di filtraggio è fondamentale, perché la rete neurale tende a trovare più oggetti di quanti ce ne siano effettivamente nell'inquadratura.

```
percentuale = output[base + 2]*100  
if(percentuale <= SOGLIA):  
    continue
```

Degli oggetti che superano questo test, recuperiamo anche gli altri campi, partendo dai vertici necessari per disegnare il box attorno all'oggetto, da moltiplicare per la dimensione dell'immagine iniziale per ottenerne le coordinate in pixels. Poiché i valori in output dalla rete possono eccedere i bordi dell'immagine, è necessario prima limitarli alle dimensioni dell'input.

```
xas = max(0, int(output[base + 3] * immagine.shape[0]))
yas = max(0, int(output[base + 4] * immagine.shape[1]))
xbd = min(immagine.shape[0], int(output[base + 5] * immagine.shape[0]))
ybd = min(immagine.shape[1], int(output[base + 6] * immagine.shape[1]))
```

Fatto ciò, possiamo disegnare il bordo del box sull'immagine:

```
overlay_on_image(immagine, output[base:base + 7])
```

Aggiorniamo la stringa che racchiude tutti gli oggetti trovati, includendo anche il nome dell'oggetto identificato ricavato tramite il tipo dell'oggetto e l'array LABELS. Riportiamo anche le altre informazioni riguardante il box dell'oggetto e l'attendibilità dell'inferenza. Formattiamo il testo per renderlo pronto alla visualizzazione sul browser, includendo i tag <br> che indicano il ritorno carrello in html.

```
ret_str = ret_str + (
'Oggetto n.' + str(i) + ' : ' + LABELS[int(output[base + 1])] + '<br>' +
'Attendibilità: ' + str(round(output[base + 2]*100,2)) + '%<br>' +
'Angolo in alto a sx: (' + str(xas) + ', ' + str(yas) + ') ' +
'Angolo in basso a dx: (' + str(xbd) + ', ' + str(ybd) + ')<br><br>'
return ret_str
```

Andiamo anche a definire la funzione che ha il compito di disegnare i box attorno agli oggetti individuati. Questa funzione sfrutta intensivamente chiamate di metodi di openCV; essendo molto semplici da comprendere e usare, mi sono limitato a riportare dal file “SSD\_MobileNet/run.py” questa funzione, non ritenendo di dover apportare alcuna modifica importante.

```
def overlay_on_image(display_image, object_info):
```



Innanzitutto, è necessario recuperare le informazioni passate per argomento alla funzione, come avevamo fatto nella funzione precedente, passando direttamente tutto l'array dei campi estrapolati dalla fase di inferenza.

```
source_image_width = display_image.shape[1]
source_image_height = display_image.shape[0]
base_index = 0
class_id = object_info[base_index + 1]
percentage = int(object_info[base_index + 2] * 100)
box_left = int(object_info[base_index + 3] * source_image_width)
box_top = int(object_info[base_index + 4] * source_image_height)
box_right = int(object_info[base_index + 5] * source_image_width)
box_bottom = int(object_info[base_index + 6] * source_image_height)
```

Definiamo anche il colore e lo spessore del box, e disegniamolo sull'immagine con l'istruzione `cv2.rectangle`:

```
box_color = (255, 128, 0)
box_thickness = 2
cv2.rectangle(display_image, (box_left, box_top), (box_right, box_bottom), box_color, box_thickness)
```

Facciamo lo stesso con l'etichetta del box, preoccupandoci prima del testo da inserire, comprendente il nome corrispondente al tipo dell'oggetto e la percentuale di attendibilità, e poi della posizione dell'etichetta rispetto al box, ossia immediatamente sopra in alto a sinistra, ma in modo tale che questa non esca fuori dall'immagine, tramite il controllo `if (label_top < 1): label_top = 1`.

```
label_text = LABELS[int(class_id)] + " (" + str(percentage) + "%)"
label_background_color = (125, 175, 75)
label_text_color = (255, 255, 255)
label_size = cv2.getTextSize(label_text, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)[0]
```

```
label_left = box_left

label_top = box_top - label_size[1]

if (label_top < 1):

    label_top = 1

label_right = label_left + label_size[0]

label_bottom = label_top + label_size[1]

cv2.rectangle(display_image, (label_left - 1, label_top - 1), (label_right + 1, label_bottom + 1),

                label_background_color, -1)

cv2.putText(display_image, label_text, (label_left, label_bottom),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, label_text_color, 1)
```

## Server Web

Questa sezione del codice, invece, specifica le azioni da compiere quando arriva una richiesta al server web. Le operazioni da fare sono i metodi ridefiniti della classe 'HTTPHandler', classe figlia di 'BaseHTTPRequestHandler' del modulo 'http.server' di Python 3.

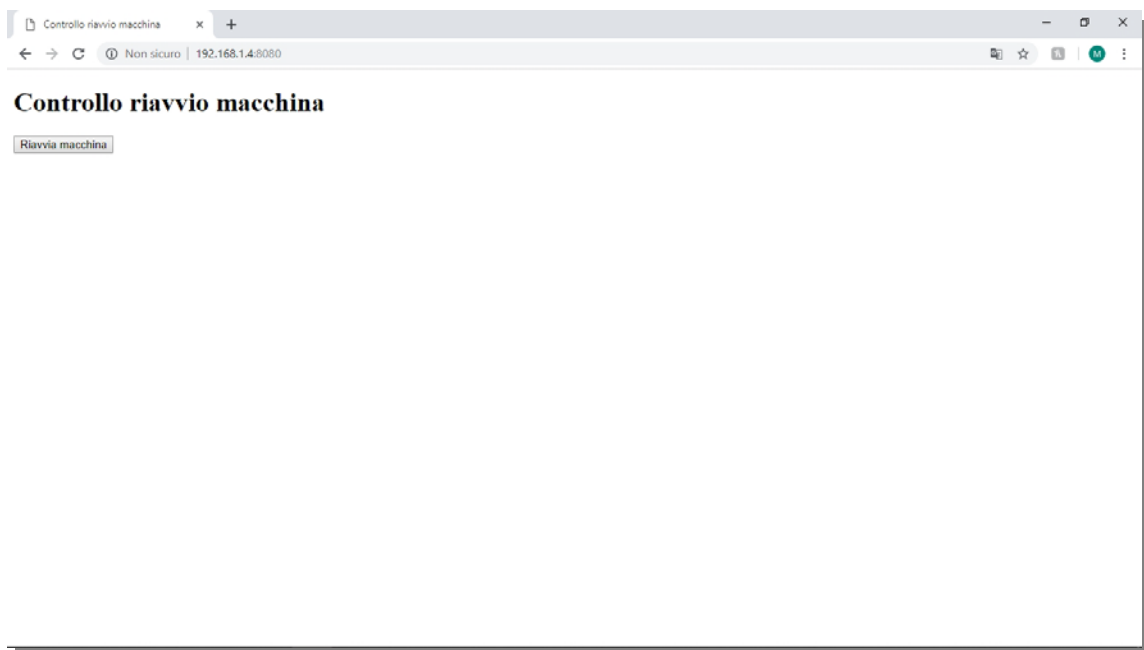
```
class HTTPHandler(BaseHTTPRequestHandler):
```

Quando si riceve una richiesta di tipo HTTP GET al server, viene eseguito il seguente metodo:

```
def do_GET(self):  
    global IMMAGINE
```

Se la pagina non viene specificata, si ridirige la richiesta sulla pagina principale "controllo-macchina.html".

```
if self.path=="/":  
    self.path="/controllo-macchina.html"
```



A questo punto, si verifica quale risorsa è stata chiesta per fornirla all'utente.

Se viene richiesta la pagina principale “controllo-macchina.html”, questa viene aperta da file e inviata all’utente con il metodo `self.wfile.write`, non prima di aver settato correttamente gli header HTTP della risposta per il tipo di file inviato.

```
try:
    if "controllo-macchina.html" in self.path:
        f = open(curdir + sep + self.path)
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(bytes(f.read(), 'utf-8'))
        return
```

Se l’utente, cliccando sul bottone “Riavvia macchina” presente sulla pagina principale, avesse inoltrato una richiesta “riavvio-macchina”, vengono eseguite le seguenti azioni: inizialmente viene eseguito un ciclo di `NUM_FOTO` iterazioni, nelle quali viene scattato un fotogramma dalla webcam, analizzato con i metodi visti precedentemente e ottenuta in output la lista degli oggetti. Se in questa è trovata la parola “persona”, si esce immediatamente dal ciclo, altrimenti si procede fino al numero massimo indicato da `NUM_FOTO`:

```
if "riavvio-macchina" in self.path:
    immagine_scattata = None
    lista_oggetti = None

    for i in range(NUM_FOTO):
        img_ret, immagine_scattata = CAMERA.read()

        print("Fotogramma: " + str(i))

        lista_oggetti = esegui_inferenza(immagine_scattata)

        if(not lista_oggetti == None and "persona" in lista_oggetti):
            break
```

Una volta usciti dal ciclo, si deve visualizzare l'output dell'elaborazione. Imposto i parametri html da visualizzare nella pagina web in caso di persona trovata, che verranno successivamente modificati se nella lista non sono presenti umani.

```
html_colore_esito = "red"

html_esito_persona_presente = "Macchinario non riavviato: persona presente"

if(lista_oggetti == None or not "persona" in lista_oggetti):

html_colore_esito = "green"

    html_esito_persona_presente = "Macchinario riavviato"
```

Si noti che è all'interno del blocco `if` sopra citato che si devono inserire le istruzioni per il riavvio della macchina, ovvero quando non sono stati trovati oggetti nell'inquadratura oppure quando non ci sono persone.

Possiamo strutturare quindi il nostro documento html, che comprende gli elementi dell'interfaccia per la visualizzazione dei risultati

```
f = '''<!DOCTYPE html>

    <html lang="en">

        <head>

            <meta http-equiv="Content-Type" content="text/html; char-
set=UTF-8">

            <title>Riavvio Macchinario</title>

            <meta name="author" content="Marco Pettorali">

        </head>

        <body>

            <h1>Riavvio Macchinario</h1>

            <h1 style = "color: {}">{}</h1>

            <p>{}</p>

            <form method="get" action="riavvio-macchina">

                <button type="submit">Ritenta avvio macchina</button>
```

```

        </form>

    </body>

</html>

''.format(html_colore_esito, html_esito_persona_presente, lista_og-
getti)

```

Per visualizzare l'immagine, è necessario prima convertirla in jpg con il metodo 'cv2.imencode', e salvare i bytes dell'immagine nella variabile globale IMMAGINE, per essere successivamente inviata all'utente appena questi ne fa richiesta ossia appena il browser, incontrando l'oggetto <img>, farà richiesta dell'immagine corrispondente al campo src, ovvero "frame.jpg".

```

retval, jpg = cv2.imencode('.jpg', immagine_scattata)

IMMAGINE = jpg.tobytes()

```

Adesso è possibile inviare la pagina all'utente.

```

self.send_response(200)

self.send_header('Content-type', 'text/html')

self.end_headers()

self.wfile.write(bytes(f, 'utf-8'))

return

```

Quando viene richiesta l'immagine "frame.jpg", il server risponde inviando i bytes dell'immagine contenuta nella variabile IMMAGINE.

```

if "frame.jpg" in self.path:

    self.send_response(200)

    self.send_header('Content-type', 'image/jpeg')

    self.end_headers()

    self.wfile.write(IMMAGINE)

return

```

Se la risorsa richiesta non è tra quelle indicate in precedenza, oppure non è stata trovata, rispondo con un messaggio di errore “404: File Not Found”.

```
else:
    self.send_error(404, 'File Not Found: %s' % self.path)
    return
except IOError:
    self.send_error(404, 'File Not Found: %s' % self.path)
```

## *Inizializzazione dell'applicazione*

L'ultima sezione del codice, è quella eseguita all'avvio del programma, ed ha il compito di inizializzare l'applicazione, di avviare il server e lo stick Movidius.

Innanzitutto, si deve aprire la webcam, tramite `cv2.VideoCapture()`:

```
try:

    CAMERA = cv2.VideoCapture(0)

    print ('Webcam trovata')
```

Successivamente, si controlla se è collegato al Raspberry Pi almeno uno stick Movidius per aprirlo e utilizzarlo durante l'esecuzione. Vengono anche allocate le code FIFO per l'I/O con Movidius.

```
devices = mvnc.enumerate_devices()

if len(devices) == 0:

    print('No devices found')

    quit()

device = mvnc.Device(devices[0])

device.open()

print ('Neural Stick aperto correttamente')
```

In questo punto del codice, viene caricato in Movidius il file 'graph' della rete MobileNet SSD:

```
GRAPH = mvnc.Graph("graph")

with open('graph', mode='rb') as f:

    blob = f.read()

GRAPH.allocate(device, blob)

INPUT_FIFO, OUTPUT_FIFO = GRAPH.allocate_with_fifos(device, blob)

print ('File graph caricato correttamente')
```



Successivamente si può avviare il server web:

```
server = HTTPServer(("", PORTA), HTTPHandler)

print ('Server web avviato sulla porta ' + str(PORTA))

server.serve_forever()
```

Quando si preme la combinazione di tasti CTRL-C, il server viene interrotto, e tutte le risorse vengono chiuse e deallocate.

```
except KeyboardInterrupt:

    print ('Chiudo il servizio')

    server.socket.close()

    INPUT_FIFO.destroy()

    OUTPUT_FIFO.destroy()

    GRAPH.destroy()

    device.close()

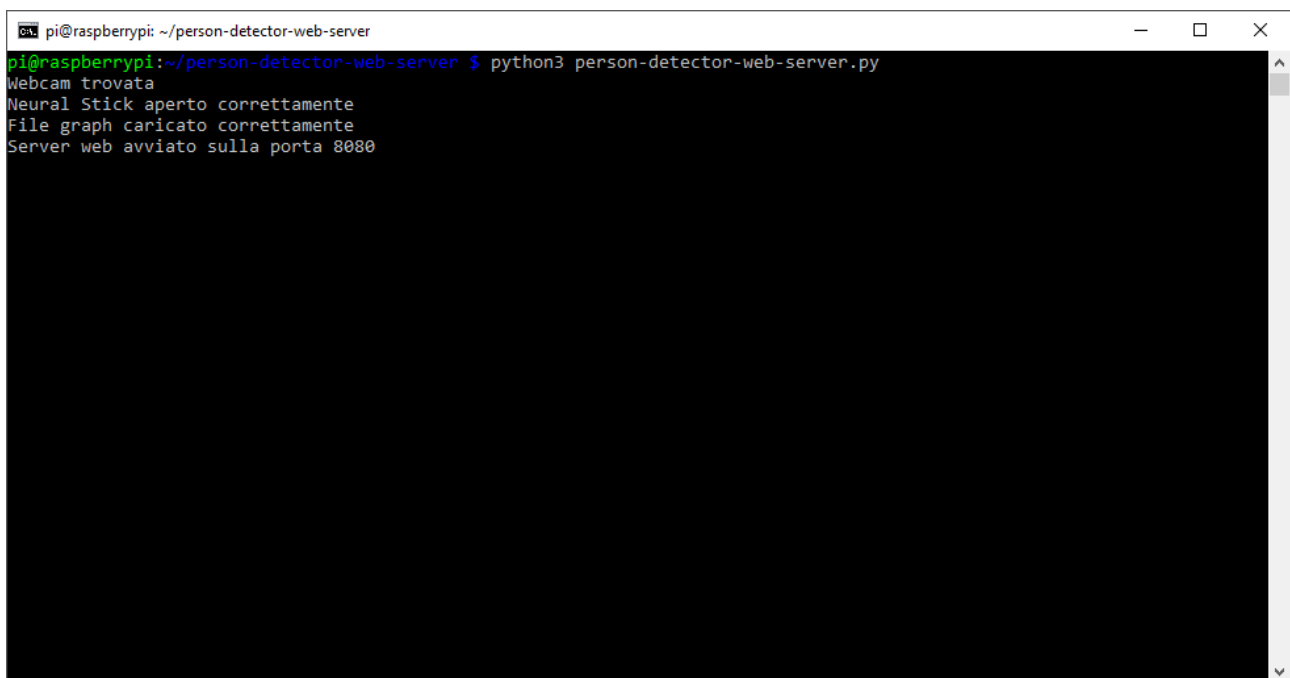
    device.destroy()
```

# Programma in esecuzione

In questa sezione, mostrerò gli screenshot della console del Raspberry Pi e dell'interfaccia web quando il servizio è in esecuzione.

## Avvio del servizio

All'avvio dell'applicazione, se tutte le operazioni sono riuscite correttamente, l'output su console sarà il seguente:



```
pi@raspberrypi: ~/person-detector-web-server
pi@raspberrypi:~/person-detector-web-server $ python3 person-detector-web-server.py
Webcam trovata
Neural Stick aperto correttamente
File graph caricato correttamente
Server web avviato sulla porta 8080
```

## Esecuzione: nessun umano rilevato

Appena si effettua una richiesta al server, in cui nessun umano è stato rilevato:

```
pi@raspberrypi: ~/person-detector-web-server
pi@raspberrypi:~/person-detector-web-server $ python3 person-detector-web-server.py
Webcam trovata
Neural Stick aperto correttamente
File graph caricato correttamente
Server web avviato sulla porta 8080
192.168.1.6 - - [26/Feb/2019 20:11:25] "GET / HTTP/1.1" 200 -
Richiesto avvio macchina
/usr/local/lib/python3.5/dist-packages/mvnc/mvncapi.py:416: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
  tensor = numpy.fromstring(tensor.raw, dtype=numpy.float32)
Elaborati 19 fotogrammi
Nessun umano rilevato
Riavviando la macchina...
192.168.1.6 - - [26/Feb/2019 20:11:31] "GET /riavvio-macchina? HTTP/1.1" 200 -
192.168.1.6 - - [26/Feb/2019 20:11:31] "GET /frame.jpg HTTP/1.1" 200 -
```

Mentre l'interfaccia web presentata all'utente sarà la seguente:



## Esecuzione: umano rilevato

Nel caso di persona rilevata, la console avrà il seguente output:

```
pi@raspberrypi: ~/person-detector-web-server
pi@raspberrypi:~/person-detector-web-server $ python3 person-detector-web-server.py
Webcam trovata
Neural Stick aperto correttamente
File graph caricato correttamente
Server web avviato sulla porta 8080
192.168.1.6 - - [26/Feb/2019 20:13:47] "GET / HTTP/1.1" 200 -
Richiesto avvio macchina
/usr/local/lib/python3.5/dist-packages/mvnc/mvncapi.py:416: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
  tensor = numpy.fromstring(tensor.raw, dtype=numpy.float32)
Elaborati 0 fotogrammi
Umano rilevato!
192.168.1.6 - - [26/Feb/2019 20:14:04] "GET /riavvio-macchina? HTTP/1.1" 200 -
192.168.1.6 - - [26/Feb/2019 20:14:04] "GET /frame.jpg HTTP/1.1" 200 -
```

Mentre l'interfaccia web sarà la seguente:

Riavvio Macchinario

Non sicuro | 192.168.1.4:8080/get-image?

### Riavvio Macchinario

**Macchinario non riavviato: persona presente**

Oggetto n.0 : persona  
Attendibilità: 98.73%  
Angolo in alto a sx: (14, 5) Angolo in basso a dx: (123, 323)

Oggetto n.1 : pianta  
Attendibilità: 61.08%  
Angolo in alto a sx: (194, 0) Angolo in basso a dx: (253, 302)

Oggetto n.2 : divano, poltrona  
Attendibilità: 99.61%  
Angolo in alto a sx: (0, 247) Angolo in basso a dx: (153, 624)

Oggetto n.3 : divano, poltrona  
Attendibilità: 85.35%  
Angolo in alto a sx: (296, 443) Angolo in basso a dx: (480, 638)

Oggetto n.4 : schermo  
Attendibilità: 90.28%  
Angolo in alto a sx: (381, 167) Angolo in basso a dx: (475, 361)

Rilenta avvio macchina

# Conclusioni

Il lavoro svolto in questa tesi dimostra che è possibile realizzare un sistema di person detection utilizzando un Raspberry Pi e la scheda Intel Movidius. Questa implementazione riesce a dare risultati molto affidabili, con percentuali di riconoscimento di presenza umana vicini al 100% nei casi di test considerati durante lo sviluppo. Una validazione del sistema in maniera rigorosa attraverso test estesi in un ambiente di uso reale è lasciato come lavoro futuro.

Restano comunque alcuni casi in cui il sistema fallisce: ad esempio in condizioni di bassa o elevata luminosità, in cui la webcam non riesce a acquisire in modo definito i bordi degli oggetti, quindi la rete neurale non ha modo di produrre un output preciso. Per risolvere questo problema, oltre a cercare il più possibile di illuminare correttamente l'ambiente di lavoro del sistema, può essere quello di usare una webcam ad alta definizione, possibilmente con un regolatore di messa a fuoco.

Un'altra criticità è quando la webcam riprende troppo da vicino un umano: infatti, la rete neurale non è stata allenata per riconoscere i dettagli del corpo umano, e quindi inquadrature comprendenti soltanto il busto o la parte inferiore di un uomo, molto spesso restituiranno un output poco affidabile da parte del sistema. La webcam deve essere posizionata possibilmente in alto, in modo tale che possa essere ripresa tutta la sagoma di una eventuale persona inquadrata.

Si consiglia, per la versione finale dell'applicativo, di affiancare questo sistema automatico ad uno a controllo umano, in modo tale che, se il sistema non rilevasse alcuna presenza umana, prima di riavviare la macchina chieda il consenso ad un operatore.

In generale, nonostante questi casi sopra citati, il sistema è molto efficiente, a basso consumo, user-friendly, ed abbastanza economico da realizzare. Per eventuali sviluppi futuri dell'applicazione, si consiglia di rendere l'applicazione compatibile per l'uso di più webcam, per poter incrociare i dati provenienti da più fonti ottenendo così risultati più affidabili, oppure per controllare zone diverse dell'ambiente di lavoro con un solo dispositivo, ottimizzando i costi del servizio.