UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Elettronica

IMPLEMENTAZIONE DI UNA CNN PER LA CLASSIFICAZIONE DI IMMAGINI TERMICHE SU PIATTAFORMA EMBEDDED ST

Implementation of a CNN for the classification of thermal images on embedded ST platform

Relatore: Chiar.mo
Prof. Claudio Turchetti

Correlatore:
Dott.ssa Laura Falaschetti

Tesi di Laurea di:
Alessio Pavoni

A.A. 2019/ 2020

Indice

Introduzione

Lo studio condotto nell'ambito del *Machine Learning* consiste nella realizzazione e implementazione su piattaforma embedded ST di una rete neurale convoluzionale (CNN) per la localizzazione e la classificazione di persone all'interno di immagini termiche del dataset FLIR.

Le reti neurali artificiali sono dei modelli matematici utilizzati nel campo dell'apprendimento automatico. Alcuni esempi di possibili applicazioni sono il riconoscimento di immagini, il riconoscimento vocale, l'advertising, la cybersecurity e il data mining. Le unità fondamentali delle reti neurali sono i neuroni artificiali. Un neurone prende in ingresso più valori numerici e li elabora prima di fornirli in uscita. Più precisamente, i valori di input vengono moltiplicati per dei pesi e sommati, eventualmente aggiungendo un bias. Il risultato viene poi elaborato da una funzione di attivazione, in modo da renderlo più maneggevole, e infine mandato in uscita al neurone.

In particolare, le reti neurali convoluzionali sono le networks che elaborano le immagini, perciò sono particolarmente utilizzate in diversi ambiti, come la guida autonoma o lo stabilimento di diagnosi mediche da TAC o risonanze magnetiche. Nelle CNN i pesi sono rappresentati dai filtri, delle matrici quadrate che vengono moltiplicate scalarmente (convoluzione) per una porzione dell'immagine di input, producendo così l'output del neurone. Il filtro viene quindi spostato sopra altre sezioni dell'ingresso e il processo viene ripetuto. I risultati ottenuti determinano la cosiddetta *feature map*, l'immagine di uscita. La rete è organizzata in strati (nel caso di due o più strati "nascosti", cioè compresi tra quello di ingresso e quello di uscita, si parla di *Deep Neural Networks*) e quello appena descritto è il layer convoluzionale. In genere, aumentare il numero di strati consente di poter riconoscere caratteristiche sempre più complesse, partendo ad esempio da linee, passando per angoli, fino a riconoscere automobili o persone.

Nello studio sono state sviluppati due tipi di reti neurali convoluzionali, diversi sia nel funzionamento che nelle prestazioni. La prima è una rete per l'*Image Segmentation*, che è in grado di classificare in maniera binaria (persona o non persona) ciascun pixel di un'immagine termica. La network è stata utilizzata per riconoscere persone all'interno delle foto del dataset FLIR, però purtroppo i risultati non sono stati molto soddisfacenti. La seconda è una rete per l'*Object Detection*, che riesce a localizzare e classificare items all'interno di una foto. Quindi, in questo caso la network delimita le sagome degli oggetti sovrapponendo loro dei rettangoli e, contemporaneamente, li distingue tra quattro possibili classi di oggetti. Anche in questo caso sono state utilizzate le immagini termiche del dataset FLIR. In questo secondo approccio le prestazioni della rete sono molto migliori, quindi è stato scelto di implementare questa CNN nella scheda embedded ST. Sfortunatamente, il caricamento del modello della rete non è andato a buon fine a causa di problemi dell'IDE di ST. Quindi, oltre alla proposta di possibili soluzioni, è stato realizzato uno studio di fattibilità riguardo all'occupazione di memoria del modello sulla board.

Nel Capitolo 1 verranno esposti i concetti teorici delle reti CNN. Nel Capitolo 2 verrà presentata una breve panoramica sui principali strumenti utlilizzati nello sviluppo, addestramento, test e implementazione delle reti neurali proposte. Nei Capitoli 3 e 4 verranno illustrati i due approcci all'elaborazione di una rete CNN, dalle basi teoriche fino alla

realizzazione software e i risultati. Nel Capitolo 5 verranno quindi trattati tutti gli aspetti relativi all'implementazione su scheda embedded ST, dalla configurazione del progetto fino ai problemi e gli eventuali metodi risolutivi. Infine, nelle Conclusioni verrà riassunto quanto realizzato e ottenuto nello studio. Nell'Appendice sono indicati tutti gli script che sono stati modificati e utilizzati, mentre nella Sitografia è possibile trovare tutti i link a siti, articoli e tutorial utili per lo svolgimento dello studio e per la comprensione dei concetti teorici di base.

1 Reti Neurali Convoluzionali (CNNs)

Le Reti Neurali Convoluzionali, anche dette *ConvNet* o CNNs (*Convolutional Neural Networks*), sono uno degli algoritmi di *Deep Learning* più utilizzati oggi nella *computer vision*. Essi sfruttano il principio della convoluzione per elaborare le immagini e riconoscere e/o localizzare oggetti al loro interno.

1.1 Prodotto di convoluzione

Prima di vedere come tale concetto venga applicato nell'ambito delle reti neurali, bisogna capire cos'è l'operazione di convoluzione nella sua accezione più generale. Date due funzioni $f(x)$ e $g(x)$, entrambe integrabili nel loro dominio (sottoinsieme di $\mathbb{R}$), si definisce come *prodotto di convoluzione* la funzione ottenuta integrando in $\mathbb{R}$ il prodotto di $f(x)$ e la versione opportunamente traslata di $g(x)$ (o viceversa):

$$h(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(z)g(x-z)dz$$

In altre parole, la convoluzione non è altro che un'operazione di prodotto seguita da una di somma (integrale). Questa *somma di prodotti* viene eseguita per ogni valore di x e l'entità della traslazione di $g(x)$ cambia al variare di x.

1.2 Convoluzione applicata alle immagini

Le immagini sono intese informaticamente come una griglia di valori numerici, ovvero come un vettore a due dimensioni. Nel caso di foto in bianco e nero è presente solo una griglia, mentre nelle foto RGB ne sono presenti tre, una per per il rosso, una per il verde e una per il blu (in questo caso si parla di immagini a tre canali). In sintesi, le immagini vengono rappresentate come entità tensoriali, cioè vettori a due o più dimensioni. Si consideri ora, per semplicità, un'immagine 5x5 in bianco e nero ad un canale, cioè una griglia quadrata di 25 valori come quella in Fig.1(a). Si costruisca in maniera analoga una matrice 2x2, ad esempio quella in Fig.1(b), chiamata *filtro* (i valori del filtro si chiamano *pesi*). A questo punto, si sovrapponga il filtro alla porzione in alto a sinistra della griglia 5x5. Moltiplicando ogni peso del filtro al rispettivo valore della sezione 2x2 di matrice, si ottiene una griglia 2x2 intermedia. Quindi sommarne tutte le componenti e porre il risultato nella casella in alto a sinistra di una nuova griglia 4x4. Tale procedimento è mostrato graficamente in Fig.2. Successivamente, si trasli il filtro di una posizione verso destra e si ripeta la procedura di prima, inserendo il risultato nella matrice 4x4 ma, questa volta, nella casella immediatamente a destra di quella scritta in precedenza. Si dovrebbe ottenere la situazione di Fig.3. Iterando il procedimento per tutte le posizioni rimaste, si ottiene la matrice finale di Fig.4, detta *feature map*. La procedura appena descritta si compone di somme di prodotti e traslazioni, cioè difatti realizza una convoluzione tra l'immagine 5x5 e il filtro 2x2. Il risultato 4x4 cambia al variare dei pesi, e più un valore della *feature map* è alto, più il filtro "combacia" con la relativa porzione di foto. Questo significa che se il filtro venisse creato ad hoc per riconoscere una determinata forma geometrica più o meno complessa, tramite il processo di convoluzione la rete sarebbe

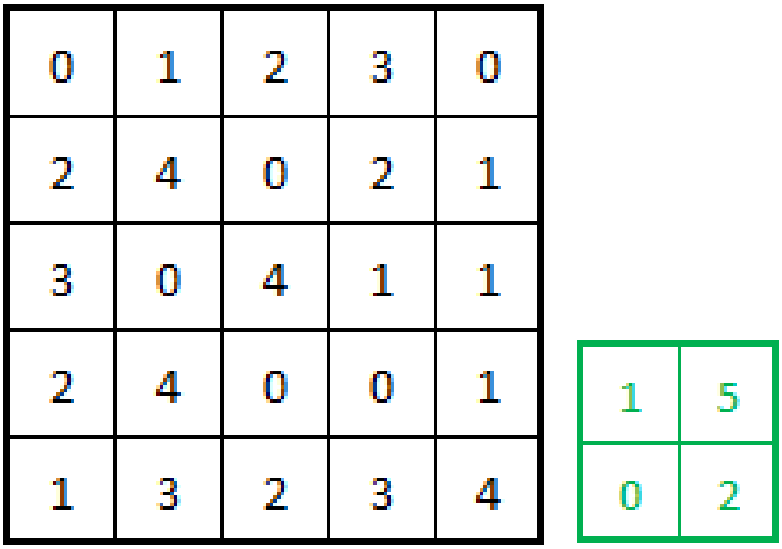


Figura 1: *(a) Immagine B&W 5x5 (b) Filtro 2x2*

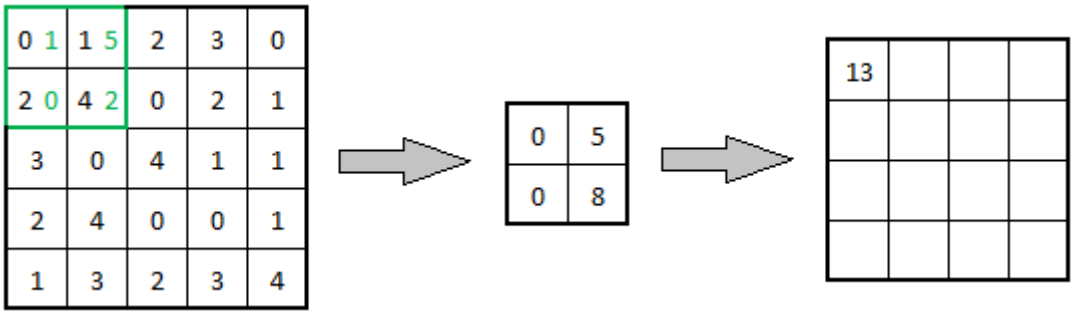


Figura 2: *Applicazione del filtro ad una porzione dell'immagine 5x5*

in grado di dire, in base ai valori della *feature map*, se effettivamente quella forma è presente nell'immagine. Le CNN elaborano le foto in questo modo, cioè applicano loro un filtro in quello che viene denominato *layer convoluzionale* e in questo modo estrapolano informazioni sui pixel che che compongono l'immagine. Oltre a quelli convoluzionali esistono molti tipi di layer neurali e le reti convoluzionali si compongono di diversi strati in cascata. Aggiungendone opportunamente sempre di più la rete è in grado di riconoscere pattern sempre più complessi, anche se non sempre l'aumento dei layer coincide con un miglioramento delle prestazioni della CNN. Affinchè la network sia in grado di distinguere oggetti particolari, come ad esempio persone o automobili, è quindi necessario scegliere opportunamente i pesi dei filtri. Ciò avviene nel cosiddetto processo di addestramento, in cui nel caso di *apprendimento supervisionato* un algoritmo realizza i seguenti passaggi:

- Vengono fornite alla rete una certa quantità di immagini affiancate dalle relative *label*

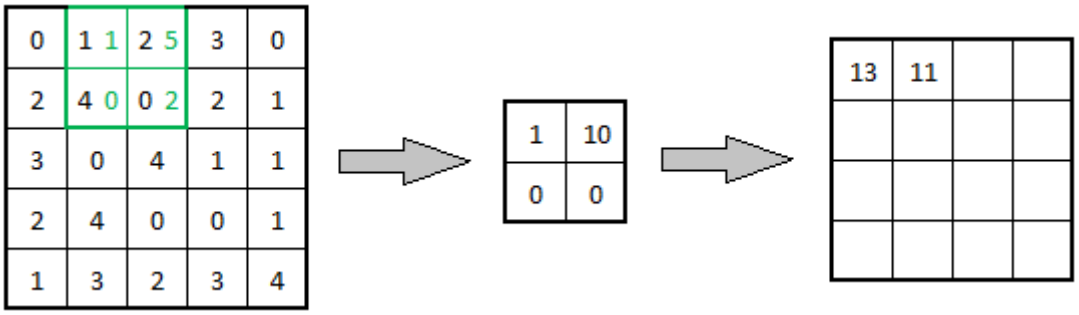


Figura 3: *Applicazione del filtro traslato ad una porzione dell'immagine 5x5*

13	11	21	5
22	12	12	9
11	20	9	8
28	8	6	13

Figura 4: *Risultato dell'elaborazione*

(foto o etichette testuali precedentemente realizzate che contengono informazioni su quali oggetti sono presenti nella foto e/o sulla loro posizione spaziale)

- Tali immagini vengono elaborate dalla CNN, che quindi compie le sue predizioni
- Viene calcolato il gradiente di una funzione perdita (tale funzione indica lo scostamento tra le predizioni della rete e le informazioni vere contenute nelle label) calcolando iterativamente i gradienti di ogni strato, dall'uscita verso l'ingresso
- I pesi dei filtri vengono aggiornati in base al gradiente appena calcolato e il processo si ripete fino a quando la funzione perdita non riesce più a decrescere

L'algoritmo appena descritto è il cosiddetto *backpropagation algorithm* ed è uno dei tanti esistenti. Al termine del training segue la fase di test della rete, in cui vengono esaminate le prestazioni del modello in base a diversi parametri di controllo, detti *metriche*. La CNN è realizzata e addestrata con successo quando essa riesce a processare con abbastanza precisione immagini non utilizzate nel training, cioè completamente sconosciute alla network. In gergo, si dice che il modello neurale è in grado di *generalizzare* a nuovi input ed è proprio questo lo scopo della realizzazione di qualsiasi rete neurale artificiale.

2 Strumenti utilizzati

2.1 Dataset FLIR

Il dataset FLIR [1] è un set gratuito di immagini termiche fornito da FLIR Systems, azienda commerciale specializzata nella progettazione e produzione di termocamere, componenti e sensori di imaging. È composto da tre cartelle:

- Train: Set di immagini di train
- Test: Set di immagini di test
- Video: Set di immagini campionate da un video (cartella non utilizzata)

Ogni cartella è suddivisa a sua volta in:

- Immagini termiche a 8 bit (8862 per il set di train, 1366 per quello di test, tutte con dimensioni 640x512)
- Immagini termiche a 16 bit (non utilizzate)
- Immagini RGB (non utilizzate)
- Immagini termiche a 8 bit già annotate (non utilizzate)
- File di annotazione .json (4 label compatibili con il formato MSCOCO)

2.2 Google Colaboratory

Google Colaboratory, o "Colab" in breve, è un prodotto gratuito di Google Research. Colab permette a chiunque di scrivere ed eseguire codici Python attraverso il browser, ed è particolarmente adatto per *Machine Learning*, *Data Analysis* e educazione. Più precisamente, è un ambiente Jupyter Notebook, cioè un'applicazione web open-source che consente di includere testo, immagini e codici. Tutti i notebooks vengono salvati in Google Drive, nonché possono interagire con i contenuti del drive stesso.

Colab rende possibile l'utilizzo, sotto alcune limitazioni, di moderne GPU (Graphics Processing Unit) e TPU (Tensor Processing Unit), anche da PC non particolarmente performanti. Sono garantiti inoltre circa 13 GB di memoria RAM e circa 108 GB di spazio su disco. A causa dei vantaggi offerti dallo strumento, tutti i codici Python utilizzati sono stati eseguiti in Google Colab e i relativi link sono forniti nella Sitografia.

2.3 STM32CubeIDE e board NUCLEO-F411RE

STM32CubeIDE è una piattaforma di sviluppo per C/C++ della STMicroelectronics con configurazione delle periferiche, generazione e compilazione di codice e funzioni di debug per microcontrollori e microprocessori STM32. In particolare, nell'ambito *Machine Learning* viene utilizzata l'estensione X-CUBE-AI, che consente la conversione di una rete neurale in codice ottimizzato per le schede STM32. La creazione di un progetto e gli

strumenti a disposizione dell'IDE verranno esposti in maniera più dettagliata nel Capitolo 5.

La board ST utilizzata si chiama NUCLEO-F411RE, dotata di microcontrollore STM32F4 con architettura ARM Cortex, memoria Flash da 512 KB e RAM da 128 KB. Ovviamente le dimensioni delle memorie sono molto limitate e questo rappresenterà un problema nel caricamento del modello della CNN.

3 Primo approccio: Rete CNN per l'Image Segmentation

Un primo approccio alla soluzione del problema è stato utilizzare la rete CNN [3] creata per l'*Image Segmentation* di immagini termiche. Questo task consiste nell'etichettare ogni pixel (*pixel-wise classification*). La rete esegue una classificazione binaria: persona o non persona. Per quanto una CNN, per esempio nell'ambito della guida autonoma, debba essere in grado di riconoscere altre classi (ad esempio auto, bici, moto, animali...), una classificazione binaria è già un ottimo inizio ed è quello a cui siamo interessati.

3.1 Schema a blocchi della rete

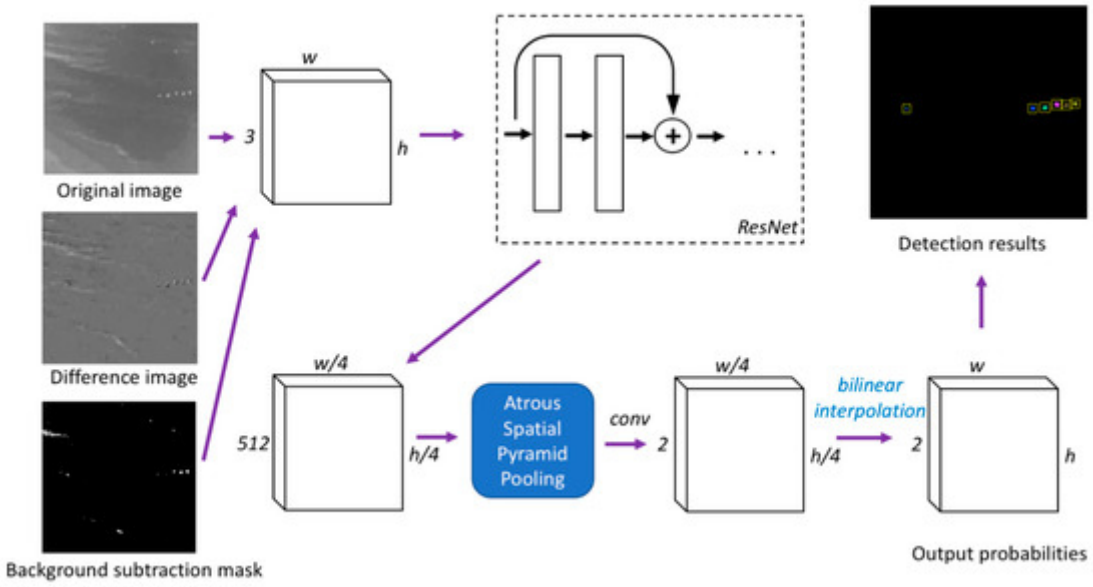


Figura 5: *Schema a blocchi della rete*

Come indicato in Fig.5, la network è basata su due blocchi principali:

- ResNet (*Residual Network*): è una particolare CNN di tipo "residuale". In genere, nell'ambito di *Artificial Intelligence*, si ha l'esigenza di dover aumentare il più possibile i layer delle reti neurali. Così facendo, infatti, si è visto sperimentalmente che la rete è in grado di riconoscere caratteristiche degli oggetti sempre più complesse e di generalizzare il più possibile nuovi input mai visti prima (scongiurando quindi il problema dell'*overfitting*).

  Però, aumentare i layer semplicemente mettendoli in cascata peggiora notevolmente le prestazioni della rete quando il numero di strati è abbastanza alto. La causa è da attribuire al *vanishing gradient problem*: durante la retropropagazione (*back-propagation*) del gradiente (il valore in base al quale si aggiornano tutti i parametri della rete al termine di ogni step di addestramento), esso può diventare estremamente piccolo, di fatto rendendo il processo di aggiornamento dei parametri (training) infinitamente lento.

La ResNet risolve proprio questo problema, rende possibile aumentare il numero di layer senza incorrere in peggioramenti delle prestazioni. Il suo principio di funzionamento nasce da un'osservazione: aggiungendo strati "identità" (uscita uguale all'ingresso) le prestazioni non peggiorano, non si verifica cioè il *vanishing gradient problem*. Per questo motivo, ad ogni stadio l'ingresso viene opportunamente elaborato (tramite blocchi convoluzionali o ReLU) e tali elaborazioni vengono sommate all'ingresso per produrre l'uscita finale, cioè l'input viene "skippato" tramite la cosiddetta *identity shortcut connection* (Fig.6). Da qui l'aggettivo "residuale".

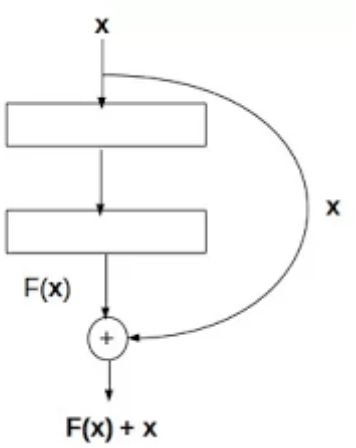


Figura 6: *Identity Shortcut Connection*

- Atrous Spatial Pyramid Pooling (ASPP): L'ASPP è una rete usata per ottenere informazioni multi-scala dell'immagine. Per fare ciò, all'ingresso vengono applicate in parallelo delle "convoluzioni a buchi" (*atrous or dilated convolutions*), ognuna con un diverso *atrous rate* (r).

  La convoluzione a buchi non è altro che una convoluzione con un filtro "sovracampionato", cioè un filtro in cui sono inseriti r-1 zeri, lungo ogni dimensione spaziale, tra due valori consecutivi dello stesso. Così facendo, il campo di visione del filtro aumenta senza modificare l'onere computazionale o il numero di parametri della *feature map* in uscita. Una convoluzione a buchi con r=1 equivale ad una convoluzione classica. La Fig.7 illustra le differenze tra convoluzione a buchi e convoluzione classica. I risultati delle operazioni in parallelo vengono poi concatenati. All'uscita dell'ASSP si ottengono due immagini, una relativa alla classe "persona" e l'altra alla classe "non persona", in cui ogni valore rappresenta la probabilità del rispettivo pixel di appartenere alla classe a cui l'immagine fa riferimento. Infine, attraverso l'applicazione di una soglia pari a 0.99, viene creata la maschera binaria in uscita all'algoritmo. Lo schema generale di una rete ASSP è indicato in Fig.8.

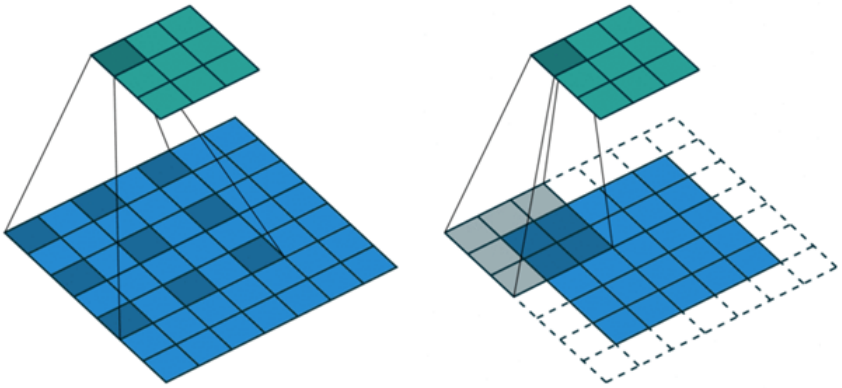


Figura 7: *Da sinistra a destra: Atrous Convolution (r=2), Convoluzione classica*

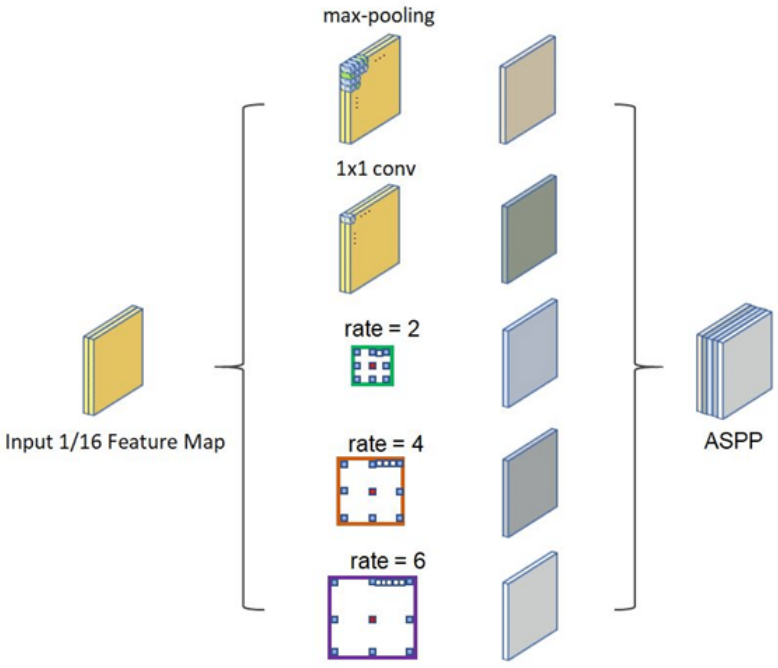


Figura 8: *Schema generale dell'ASSP, nel nostro caso vengono usati dei rates pari a 1, 6, 12 e 18*

3.2 Metriche

Nei risultati verranno forniti dei grafici in cui sono indicati tre parametri di controllo, detti metriche, al variare dell'epoca di addestramento. Essi sono:

- Loss: *Cross-entropy loss*
- Precision: $tp/(tp + fp)$
- Recall: $tp/(tp + fn)$

I termini tp, fp e fn rappresentano rispettivamente il numero dei pixel "true positive", "false positive" e "false negative". Un pixel è tp, fp o fn in base alla relazione tra classe predetta dalla rete e classe vera, come indicato in Fig.9. La loss deve essere la più bassa

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figura 9: *Definizioni degli esiti delle predizioni*

possibile. Precision e recall, che sono compresi tra 0 e 1, sono una misura dei falsi positivi e dei falsi negativi rispettivamente. Più tali metriche sono alte, più il numero dei falsi tende a zero.

3.3 Implementazione software

Innanzitutto, bisogna scaricare tutto il materiale necessario e caricarlo in Google Drive. Creare nel drive una cartella "IR", scaricare i codici Python da [4] e inserire la directory "IR_detection-master" in "IR". Replicare lo schema di cartelle indicato in Fig.10. Scegliere i dataset che si andranno ad utilizzare e inserire foto e annotazioni nella directory opportuna (es. sia label che foto del FLIR vanno nella cartella "FLIR"). Prima di caricarle, però, le immagini devono essere rinominate in maniera sequenziale (1.jpeg, 2.jpeg...etc) e analogamente le label (label1.png, label2.png...etc). Per fare ciò è stato utilizzato il programma *Advance Renamer*, facilmente scaricabile da Internet. Il dataset university fornito dagli autori della rete è reperibile al link [5]. Se c'è la necessità di creare manualmente le label (come nel caso del dataset FLIR) si può usare un apposito software chiamato *PixelAnnotationTool* (Fig.11), disponibile su Internet. Nella stessa cartella, creare un file *params.txt* composto da tre interi numerici separati da uno spazio. Essi rappresentano rispettivamente xOffset, yOffset e cropSize, cioè le coordinate della porzione quadrata di foto, di area cropSize x cropSize, che verrà data in input alla rete. Infine, nella cartella "IR_detection-master", sostituire il codice *train.py* con quello modificato presente nell'Appendice.

Lo script Colab *main_IR_detection.ipynb* è disponibile su OneDrive (link [6]).

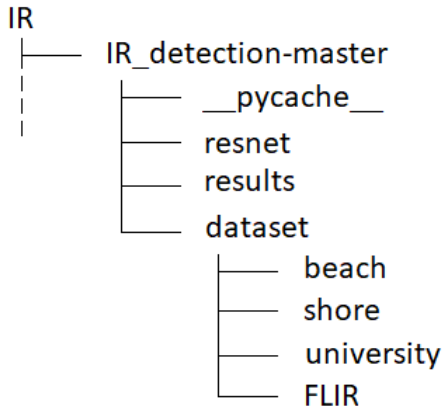


Figura 10: *Schema di cartelle da realizzare in Google Drive*

Figura 11: *PixelAnnotationTool*

1. Per prima cosa bisogna importare le librerie necessarie: Scipy, Pillow, Numpy e TensorFlow. In particolare, la libreria open source TensorFlow (TF) di Google è fondamentale per svolgere funzioni in *Deep Learning*. La versione TF installata consente l'utilizzo della GPU. La prima volta che si esegue l'installazione il runtime dovrà essere riavviato, poi si possono reimportare di nuovo le librerie e continuare col resto del programma. Se al termine dell'esecuzione viene stampata la stringa "True", significa che la GPU è stata correttamente attivata. L'uso della GPU è fondamentale per abbattere i tempi di addestramento della rete, che nel caso di utilizzo della CPU sarebbero molto più lunghi.

2. Successivamente, è necessario "montare" l'account di Google Drive personale. Per farlo, basta eseguire la cella successiva, aprire il link che verrà visualizzato, copiare e incollare il codice alfanumerico. In questo modo, lo script sarà in grado di accedere ai contenuti del drive e di modificarli.

3. Creare i files H5 con lo script *process_record.py*, selezionando opportunamente il dataset in uso. Gli H5 sono files di dati salvati nel formato HDF (*Hierarchical Data Format*). In questo caso, vengono utilizzati per compattare in un unico file tutte le informazioni sulle immagini e le relative label.

4. Addestrare la rete tramite lo script *train.py*. Selezionare il dataset, il numero di epoche di addestramento e il batch size (il batch size è il numero di immagini che vengono propagate attraverso la rete prima dell'aggiornamento dei parametri della stessa). Ad ogni epoca un checkpoint viene salvato nella cartella del dataset in uso. Ogni checkpoint è composto da tre file: *checkpoint_epochX.ckpt.data-00000-of-00001*, *checkpoint_epochX.ckpt.index* e *checkpoint_epochX.ckpt.meta* (X rappresenta il numero dell'epoca). Essi consentono eventualmente di stoppare il processo di training e riprenderlo in un secondo momento, ma in tal caso bisogna commentare e de-commentare opportunamente alcune linee di codice di *train.py* (tali modifiche sono indicate nell'Appendice). In automatico, nella cartella rimangono i cinque checkpoints più recenti, il resto viene mandato al cestino. Ciò potrebbe provocare problemi di spazio nel drive, perciò bisogna ricordarsi di svuotare il cestino di tanto in tanto, anche durante l'esecuzione della cella. In ogni caso, nella stessa cartella al termine dell'addestramento viene salvato il modello della rete con i tre file *model.ckpt.data-00000-of-00001*, *model.ckpt.index* e *model.ckpt.meta*. Durante il training, ad ogni epoca vengono stampati i valori di loss, precision e recall ricavati dalle immagini di train.

5. Eseguendo quindi l'ultima cella si attua il test della rete. Bisogna prima selezionare il dataset in uso, quindi eseguendo lo script vengono visualizzate le immagini di test e i loro risultati. Poi sono stampati diversi parametri, soffermarsi sul secondo e sul quarto che sono precision e recall rispettivamente.

3.4 Risultati

La rete è stata addestrata con batch size, numero di immagini e numero di epoche variabili. Inoltre, sono necessarie immagini di input "quadrate", perciò, modificando opportunamente le coordinate in *params.txt*, sono state selezionate le porzioni centrali delle immagini. Il dataset utilizzato viene automaticamente diviso in 80% train e 20% test.

Per prima cosa è stato eseguito l'addestramento tramite il dataset "university", fornito dagli autori della network. È composto da 120 immagini termiche scattate in notturna e in presenza di illuminazione pubblica da una termocamera di sicurezza, posta in posizione rialzata in un cortile universitario. Il training è avvenuto con un batch size pari a 5 e per 150 epoche (Fig.12). C'è da specificare tutti i grafici mostrati sono relativi ai tre parametri durante l'addestramento, cioè ottenuti utilizzando le immagini di train. Come si può vedere, la perdita è quasi nulla e il recall è massimo. La precisione raggiunge un

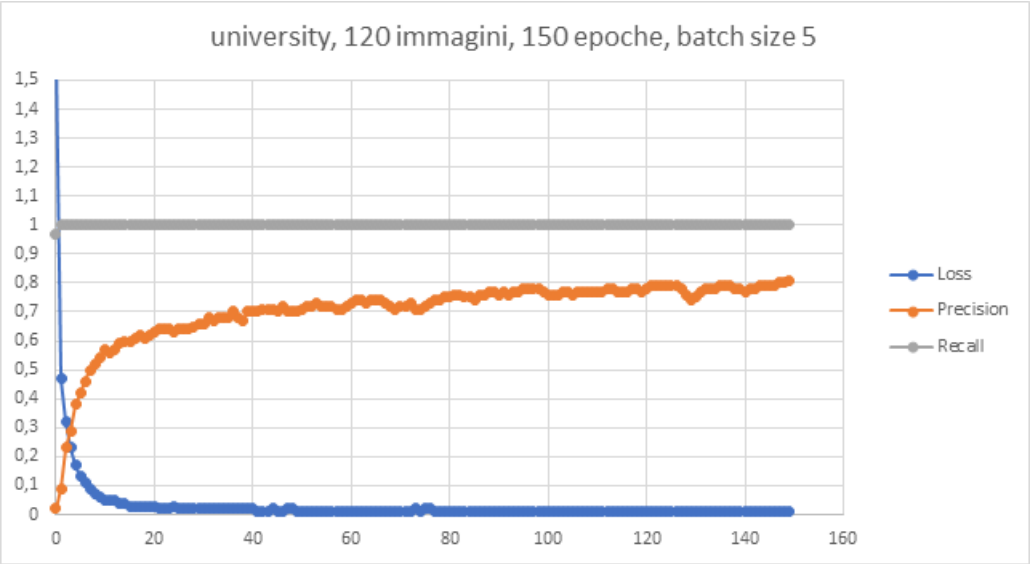
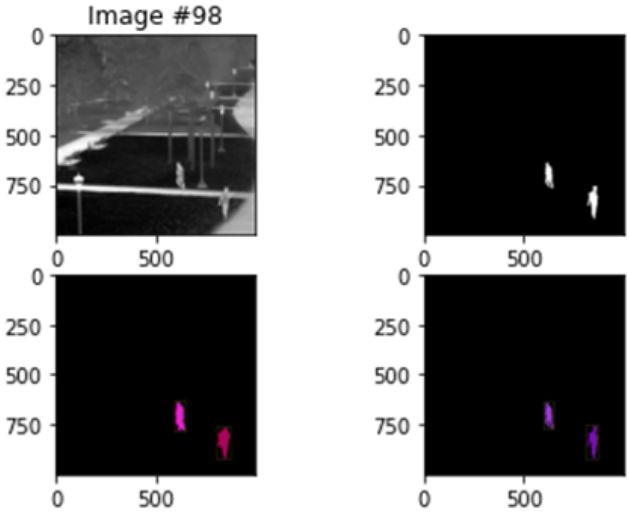


Figura 12: *Grafico delle metriche (university, 120 immagini, 150 epoche, batch size 5) NB: il numero di immagini nei titoli dei grafici sono le immagini totali, solo l'80% di esse viene utilizzato per l'addestramento*

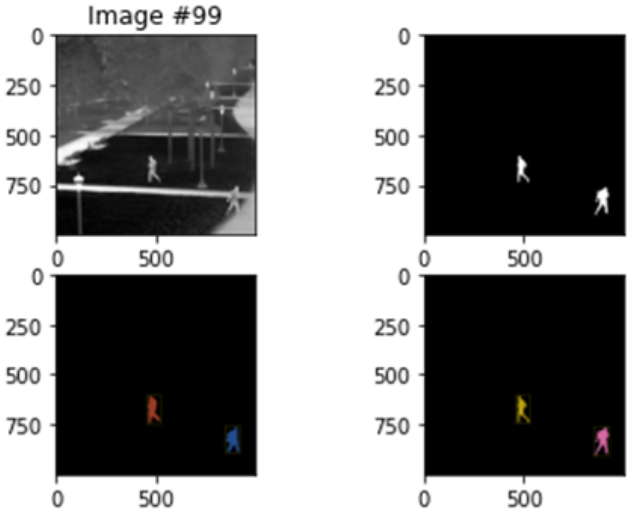
massimo dell'80% che, considerando il basso numero di immagini ed epoche, è sicuramente un buon risultato. Inoltre, si nota come tutti e tre i valori seguono andamenti pressochè monotoni.

Le foto di Fig.13 mostrano i risultati dell'*Image Segmentation* su alcune immagini di test (in queste foto, così come in quelle di seguito, ignorare il riquadro (d) in basso a destra). La rete funziona abbastanza bene su questo dataset. Infatti, con le immagini di test si ottengono una precision del 72.7% e un recall del 90.7%.

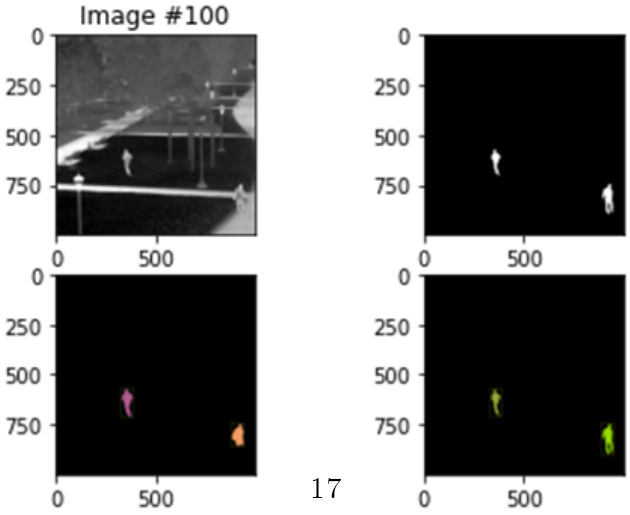
In seguito, la rete è stata addestrata col dataset FLIR, inizialmente con 120 immagini, 150 epoche e batch size pari a 5 (Fig.14) così da poter fare un confronto alla pari con lo scorso grafico. Le immagini di annotazione *pixel-wise* non sono fornite dagli autori di questo dataset, perciò sono state realizzate "a mano" attraverso *PixelAnnotationTool*. Come si può notare la situazione è diversa. La perdita è di poco inferiore a 0.1 e la precisione raggiunge un massimo del 50%, anche se il recall è pressochè sempre massimo. Inoltre, attorno alla 75esima epoca si denota una "ricaduta", cioè un picco in cui le prestazioni si degradano, salvo poi tornare a crescere. Per migliorare i parametri la rete è stata più volte riaddestrata con condizioni sempre differenti. Aumentando semplicemente il numero di immagini (Fig.15) i risultati non migliorano, anzi sono presenti ancora più picchi "di peggioramento" e la precisione supera di poco il 40%. Incrementando sia il numero di immagini che il numero di epoche (Fig.16) la situazione rimane quasi invariata, anzi si conferma l'aumento delle ricadute all'aumentare delle foto. Questa volta, però, la precisione massima raggiunge il 50%. Aumentando ulteriormente il numero di epoche (Fig.17) si nota che la precisione massima a malapena riesce a superare il 50%, perciò i risultati non sono migliorati. Anche aumentando il batch size a 10 (Fig.18) la precisione massima non migliora. In compenso, si nota una diminuzione della frequenza delle ricadute.



(a)



(b)



(c)

Figura 13: *(a) Immagine originale (b) immagine annotazione (c) immagine risultato (d) annotazione con bounding box*

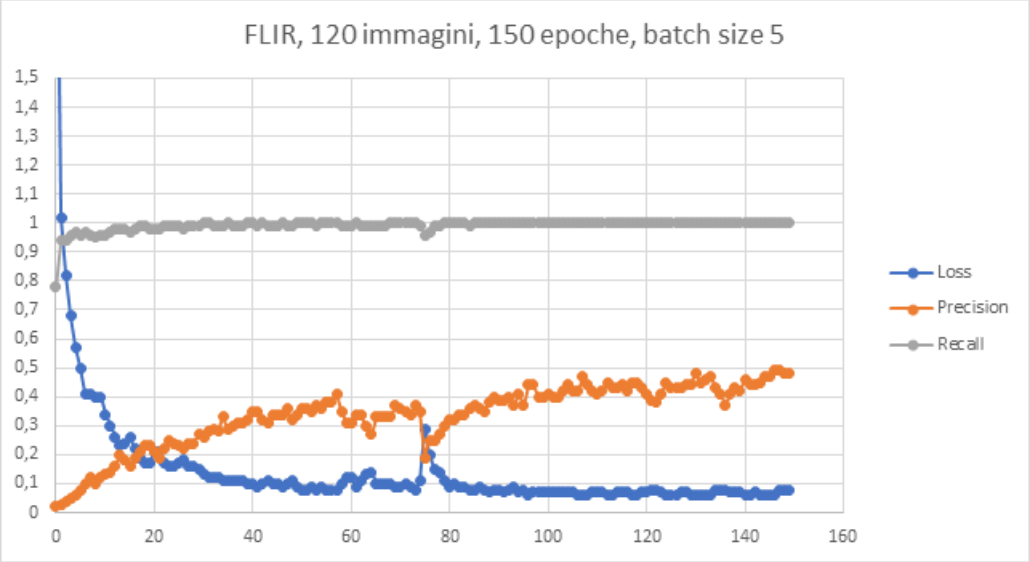


Figura 14: *Grafico delle metriche (FLIR, 120 immagini, 150 epoche, batch size 5)*

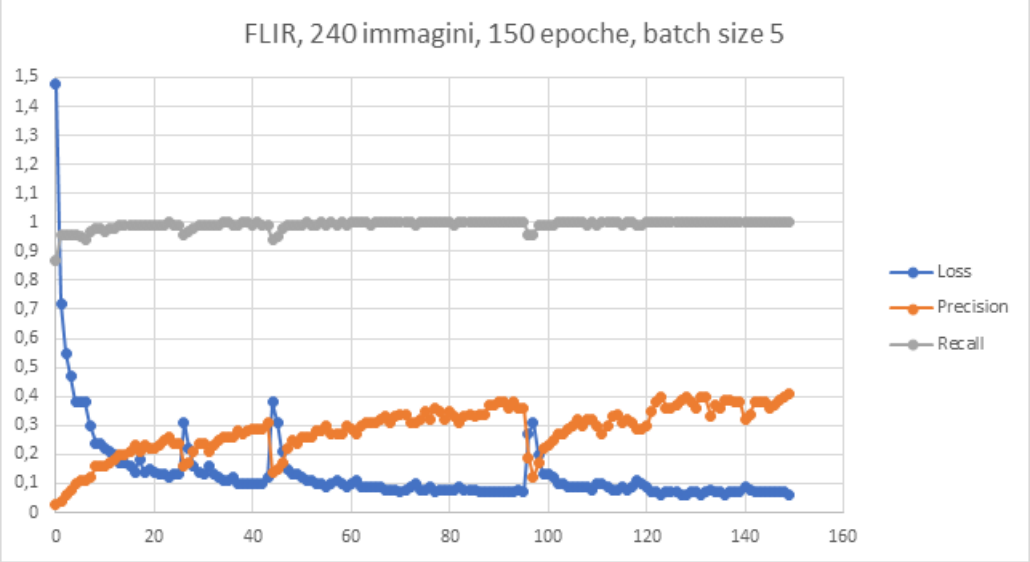


Figura 15: *Grafico delle metriche (FLIR, 240 immagini, 150 epoche, batch size 5)*

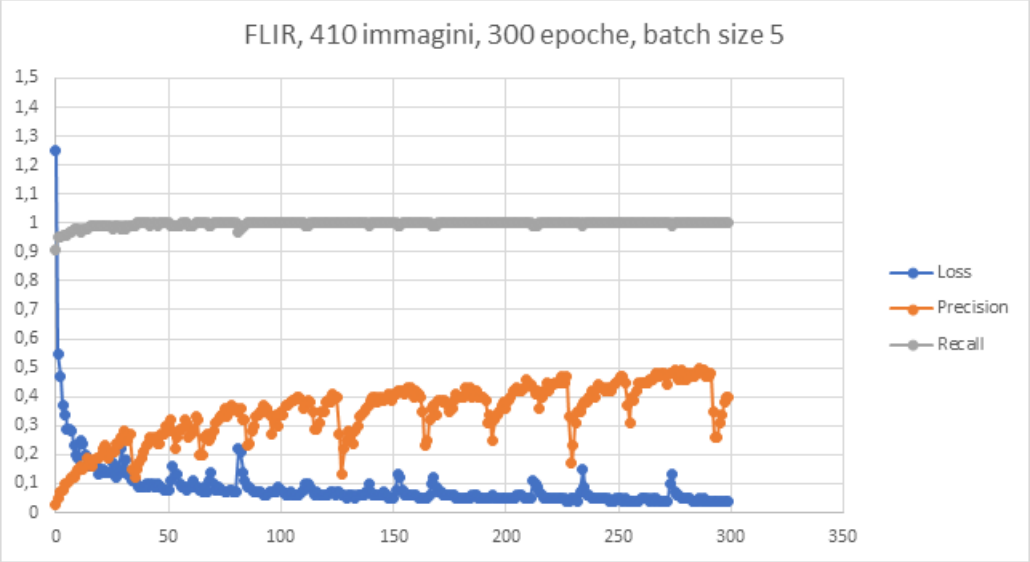


Figura 16: *Grafico delle metriche (FLIR, 410 immagini, 300 epoche, batch size 5)*

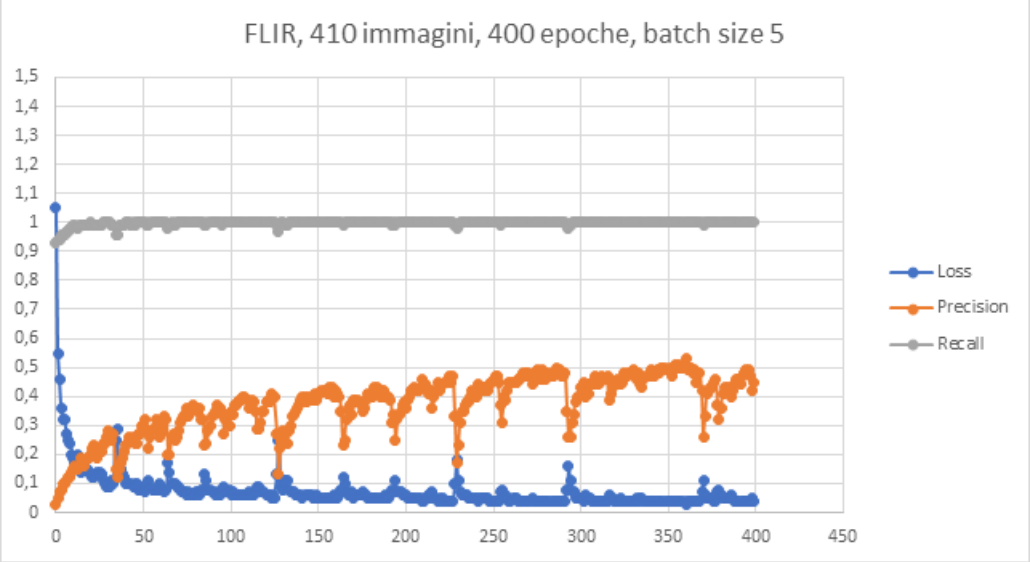


Figura 17: *Grafico delle metriche (FLIR, 410 immagini, 400 epoche, batch size 5)*

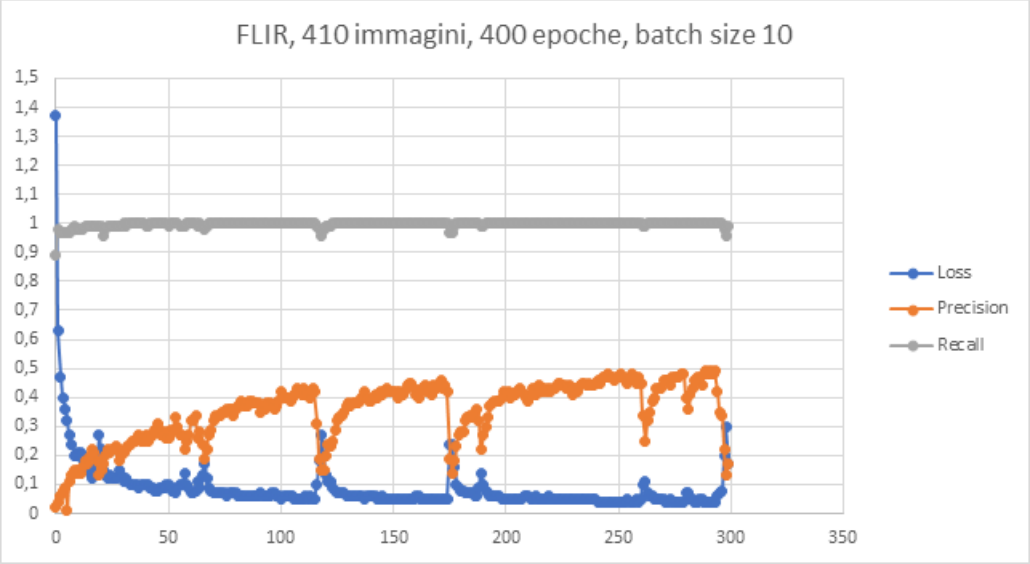
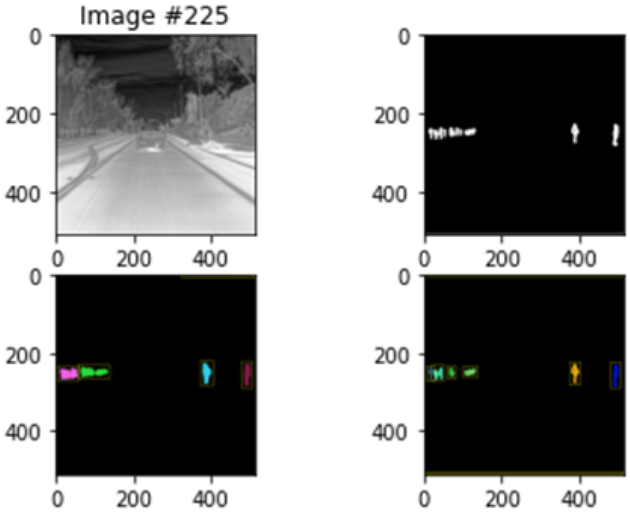


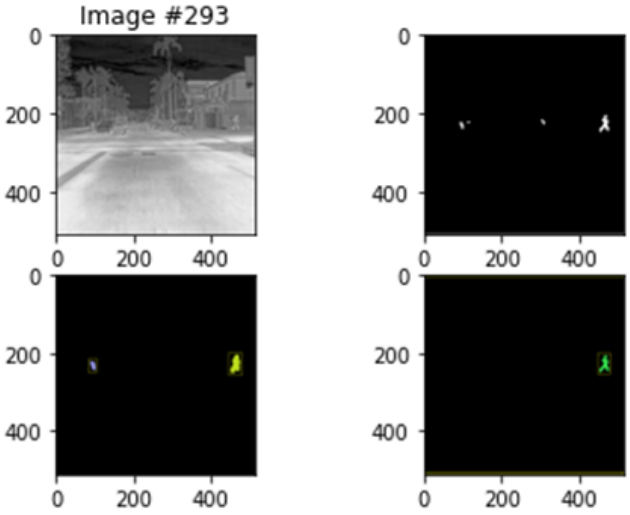
Figura 18: *Grafico delle metriche (FLIR, 410 immagini, 400 epoche, batch size 10)*

Uno dei problemi potrebbe quindi risiedere nel basso valore di batch size. Purtroppo, però, per un motivo sconosciuto lo script non permette di aumentare il batch size al di sopra di 13, quindi non è stato possibile effettuare ulteriori prove.

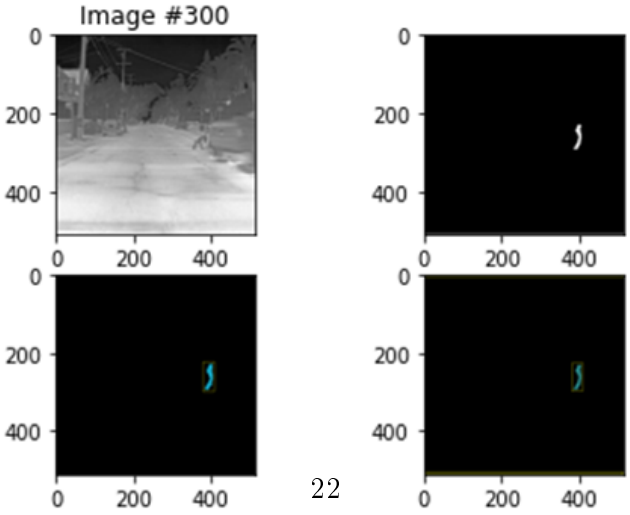
Svolgendo il test della rete con le immagini di train (le prime 330 foto del dataset, cioè circa l'80% di 410) si nota che la rete ottiene delle buone performance, come si può vedere dalla Fig.19. Eseguendo la prova sulle sole immagini di test (dalla foto 331 in poi, cioè circa il 20% di 410) ci si aspetta un peggioramento delle prestazioni, in quanto, dal momento che non vengono utilizzate per il train, queste foto sono "sconosciute" alla rete. Infatti, si ottiene un recall del 17.4%, cioè molto inferiore al 100% di prima. Stranamente, però, la precision passa dal precedente 50% al 66%, valore comunque piuttosto basso. In Fig.20 i risultati della rete sul dataset di test FLIR. Si evidenziano diversi falsi positivi, come nel caso della terza immagine di test. Questo, unitamente al fatto che in genere le sagome delle persone sono composte da pochi pixel e quindi il numero di "true positives" è basso, giustifica lo scarso valore di precision. Inoltre, si notano anche moltissimi falsi negativi, come nel caso della prima immagine di test. Ciò conferma il valore estremamente basso di recall. Quindi, nonostante la precision sia leggermente aumentata, il forte ribasso del valore di recall evidenzia l'inadeguatezza della rete nella segmentazione delle immagini di test. È il cosiddetto problema dell'*overfitting*, cioè l'incapacità della rete di generalizzare a nuovi input diversi da quelli utilizzati in fase di addestramento. Questo potrebbe essere dovuto sia al basso numero di immagini del dataset, sia alla poca rappresentatività delle immagini di train (foto poco generali e molto simili tra loro). In questo caso, però, è possibile scartare tali ipotesi, dal momento che anche il dataset university è composto da pochissime foto e il dataset FLIR di train è stato costruito con fotografie quanto più differenziate possibile. Le vere cause del problema sono da ricercare nelle differenze tra i due dataset. Infatti, quello fornito dagli autori della network presenta uno sfondo statico, essendo la termocamera fissata ad un palo, e in condizioni di media illuminazione. Invece, Il dataset FLIR ha un background mobile, dal momento che le immagini sono state scattate da un'auto in movimento, e in condizioni di forte illuminazione, visto che le riprese sono avvenute sia di giorno che di notte. Inoltre, un altro fattore determinante potrebbe essere la non completa accuratezza delle label, dato che esse sono state realizzate manualmente.



(a)

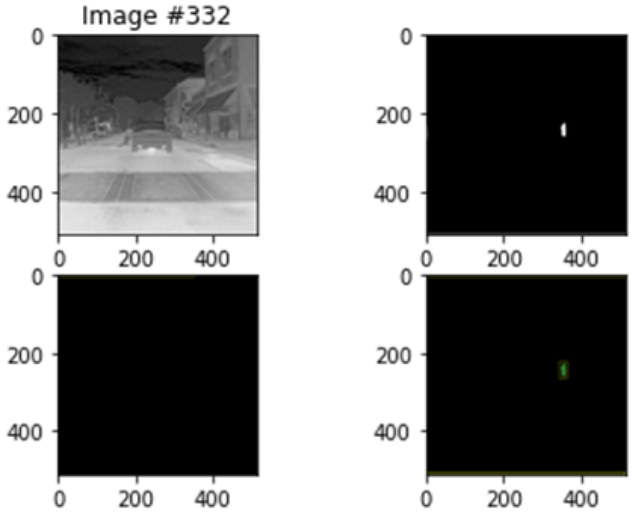


(b)

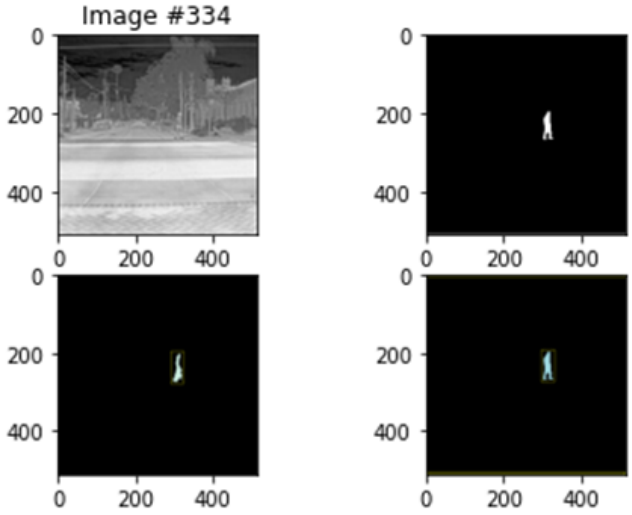


(c)

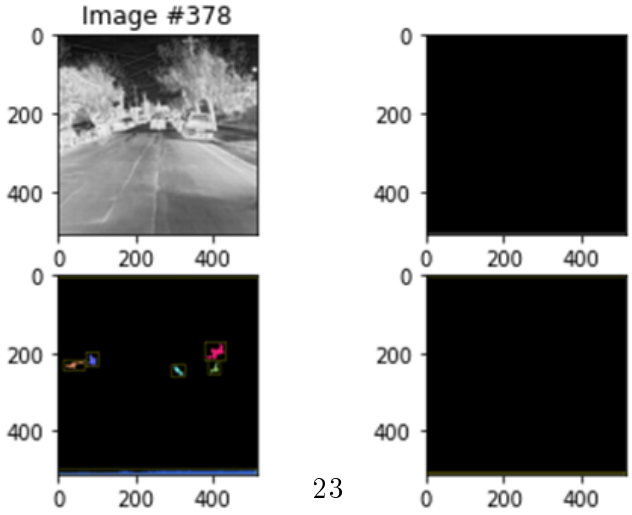
Figura 19: *Test della rete sulle immagini di train: (a) Immagine originale (b) immagine annotazione (c) immagine risultato (d) annotazione con bounding box*



(a)



(b)



(c)

Figura 20: *Test della rete sulle immagini di test: (a) Immagine originale (b) immagine annotazione (c) immagine risultato (d) annotazione con bounding box*

4 Secondo approccio: Rete CNN per l'Object Detection

Il secondo approccio è stato quello di addestrare una CNN per l'*Object Detection*. Una rete in grado di eseguire questo task deve essere capace di eseguire sia una localizzazione (detection), cioè individuare la posizione di tutti i diversi oggetti presenti in un'immagine, sia una classificazione, cioè riconoscere la natura di tali oggetti.

Per fare ciò, seguendo il tutorial [7] è stato eseguito un re-train di una rete preesistente, la SSD-MobileNet-v2.

4.1 SSD-MobileNet-v2

La SSD-MobileNet-v2 è composta da due reti, la MobileNet-v2 e la SSD, le cui caratteristiche sono brevemente descritte di seguito:

- MobileNet-v2: è una CNN per la classificazione di immagini. Il suo ultimo strato è il *fully connected layer*.
- SSD (*Single Shot Detector*): è una CNN per la detection degli oggetti presenti nelle immagini.

MobileNet-v2 e SSD da sole non sono quindi in grado di realizzare un'Object Detection, dato che entrambe, prese singolarmente, possono eseguire solo metà del task. Per questo motivo le due reti vengono fuse: si sostituisce il *fully connected layer* della MobileNet-v2 con una SSD, creando quindi la SSD-MobileNet-v2, capace di eseguire l'intero compito di localizzazione e classificazione.

In Internet si trova la SSD-MobileNet-v2 già addestrata con il famoso COCO dataset, un'enorme raccolta di immagini con le relative annotazioni in cui si possono distinguere fino ad 80 categorie di oggetti diversi. Quindi, volendo utilizzare il set di immagini termiche FLIR, bisogna riaddestrare la rete con il nostro dataset (metodo *Transfer Learning*). Usare una rete già addestrata rispetto ad una costruita ad hoc è sicuramente molto più semplice. D'altra parte, però, c'è da tenere in considerazione due aspetti: affinchè possa essere riaddestrata, la rete è pensata per essere più generica possibile (non è quindi ottimizzata per l'utilizzo) e, inoltre, essa viene addestrata su due dataset. Per questi motivi la rete potrebbe risultare molto pesante dal punto di vista dell'occupazione di memoria, quindi non essere adatta ad un utilizzo in una scheda embedded.

4.2 Metriche

Quando si parla di *Object Detection* esistono diverse tipologie di metriche. Le principali sono:

- IoU (*Intersection over Union*): misura della sovrapposizione di due *bounding boxes* (Fig.21). Si usa per determinare quanto i bordi dei presunti oggetti localizzati (anche detti ROI, *Regions Of Interest*) intersecano i bordi degli oggetti "reali" (quelli indicati nelle label).

$$IoU = \frac{\text{area di intersezione}}{\text{area di unione}}$$

Per stabilire se la detection è stata corretta o meno, si applica un valore di soglia. Se l'IoU è superiore alla soglia, allora la localizzazione è considerata positiva (true positive), altrimenti negativa (false positive).

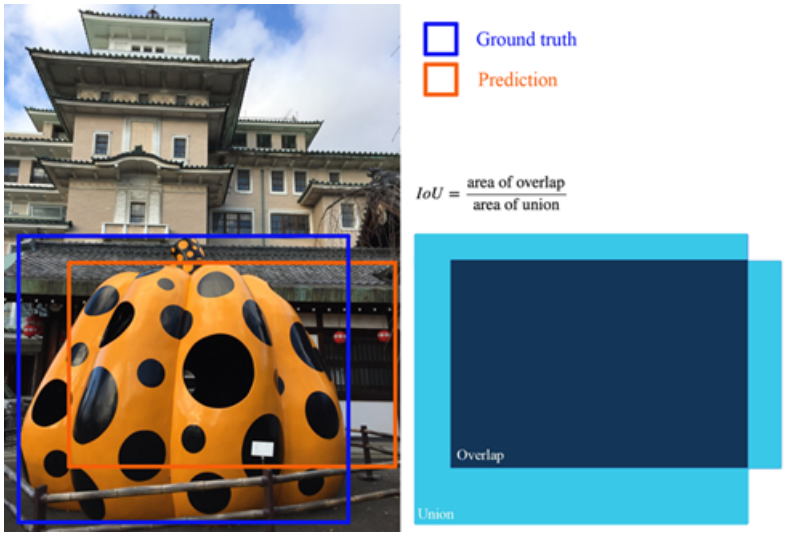


Figura 21: *IoU*

- Precision e Recall: sono gli stessi parametri già visti per l'*Image Segmentation*, con la differenza che in questo caso possono essere applicati anche al task di detection. Infatti, i "true positive" (analogamente i "true negative", "false positive" e "false negative") possono rappresentare sia il numero di classificazioni corrette, sia il numero di localizzazioni corrette. Utilizzando questa volta precision e recall nell'ambito della detection, si supponga di trovarsi nella situazione di Fig.22: I rettangoli

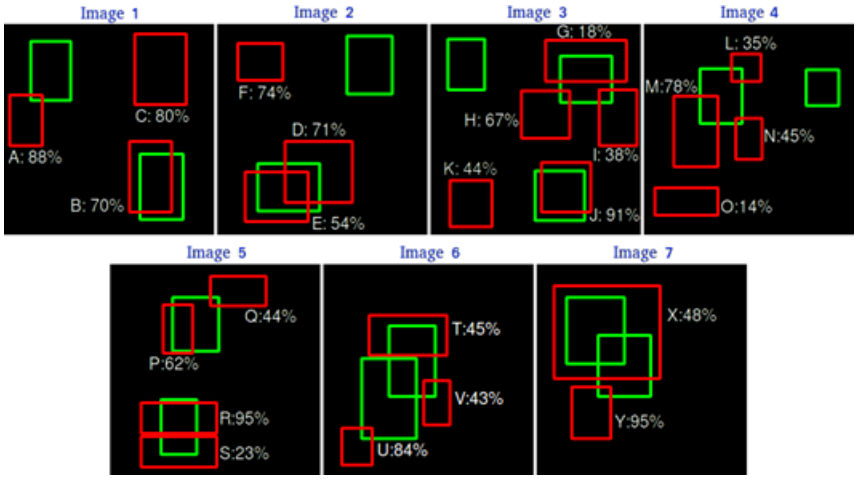


Figura 22: *In relazione ad una classe, i box verdi sono le annotazioni e quelli rossi, ognuno col proprio nome e livello di sicurezza, sono le detection*

verdi sono i *bounding boxes* di annotazione (dove realmente si trovano gli oggetti), mentre quelli rossi sono le detection della rete, ognuna con il proprio nome e il proprio livello percentuale di sicurezza. Tutti i rettangoli sono relativi ad una sola classe. A questo punto, fissato uno specifico valore di soglia dell'IoU, si costruisca una tabella analoga a quella di Fig.23. Ogni riga corrisponde ad un *bounding box*

Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
Image 4	O	14%	0	1	7	17	0.2916	0.4666

Figura 23: *Tabella per il calcolo di precision e recall*

localizzato dalla rete e le righe sono ordinate in base al livello di sicurezza in ordine decrescente. Ogni box è considerato un true positive (TP) o false positive (FP) a seconda di quanto interseca un rettangolo verde, e scorrendo la tabella dall'alto verso il basso si costruiscono le colonne "Acc TP" e "Acc FP", che altro non sono che un conteggio dei TP e FP rispettivamente. Quindi, si calcolano precision e recall utilizzando i conteggi di TP e FP. Possiamo perciò ottenere dalle relative colonne la curva precision-recall, che nel caso in esempio è come in Fig.24.

- AP (*Average Precision*) e mAP (*mean Average Precision*): si definisce l'AP di una classe, dato il valore di soglia dell'IoU, come l'area sottesa alla curva precision-recall

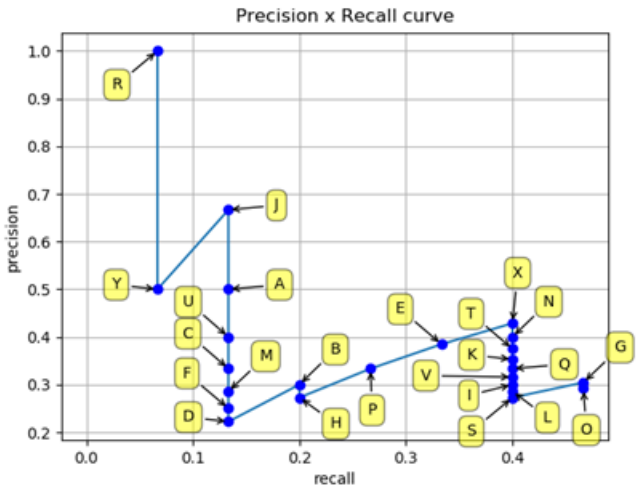


Figura 24: *Curva Precision-Recall*

ottenuta in precedenza:

$$AP = \int_0^1 p(r)dr$$

Per diminuire la sensibilità dell'AP alle piccole variazioni di precision e recall, si utilizza una versione modificata della curva (Fig.25):

$$p_{interp}(r) = \max_{\tilde{r}>r} p(\tilde{r})$$

Esistono diversi modi per calcolare l'area sottesa alla precisione interpolata. Nel

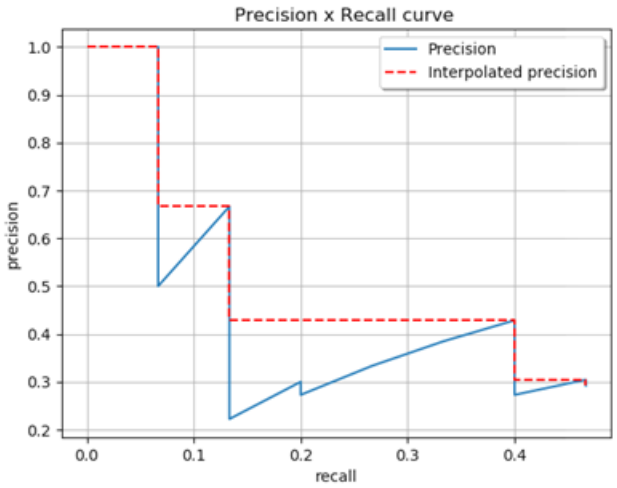


Figura 25: *Precisione interpolata (rosso)*

nostro caso, per il COCO dataset, si campiona la curva in 101 punti (da recall 0 a

1 con passo 0.01) e si fa la media aritmetica:

$$AP = \frac{1}{101} \sum_{r \in \{0, 0.1, \ldots 1\}} p_{interp}(r)$$

Mediando l'AP su tutte le classi e su più valori di soglia dell'IoU si ottiene la mAP. Nel caso del COCO dataset, esistono diverse mAP (si noti dalla documentazione ufficiale [10] che a livello di terminologia la mAP viene chiamata AP). La principale, *AP@IoU=.50:.05:.95*, si ottiene facendo la media delle AP su tutte le classi e su tutti i valori di soglia che vanno da 0.5 a 0.95 con passo 0.05.

- AR (*Average Recall*) e mAR (*mean Average Recall*): Data una classe, l'AR è il doppio dell'area sottesa alla curva del recall al variare del valore di soglia dell'IoU tra 0.5 e 1:

$$AR = 2 \int_{0.5}^{1} r(IoU) dIoU$$

Analogamente alla mAP, il mAR è la media aritmetica dell'AR sul numero di classi. Nel COCO dataset ci sono diversi tipi di mAR (anche in questo caso il mAR viene chiamato AR), che si differenziano tra loro in base a quali *bounding boxes* vengono considerati (vengono scelti in base alla grandezza o al numero massimo di rettangoli presenti in un'immagine).

4.3 Implementazione software

Per iniziare, bisogna creare nel drive una cartella "models". Poi, seguendo lo script Colab *Training an Object Detection Model.ipynb* (link [6]), è necessario effettuare i seguenti passaggi:

1. Installare la libreria TensorFlow versione 1.x e verificare che la GPU sia attiva. Se non viene generato un errore e viene stampato "True", significa che la GPU è stata abilitata correttamente.

2. Successivamente, montare l'account di Google Drive personale. Per farlo, basta eseguire la cella successiva, aprire il link che verrà visualizzato, copiare e incollare il codice alfanumerico.

3. Clonare nella cartella "models" del drive la repository TensorFlow API. Proprio perché il salvataggio avviene nel drive, è necessario eseguire questo passaggio solo una volta. A questo punto bisogna replicare lo schema di cartelle di Fig.26 in Google Drive. In seguito, inserire il dataset in uso nelle cartelle "train" (80% delle immagini) e "test" (20% delle immagini). Se si dispone di label in formato .xml, caricare opportunamente anche loro nelle cartelle "train" e "test". Se invece si usano annotazioni in formato .json con un file per il train e uno per il test (ad esempio utilizzando il dataset FLIR completo), inserire i due file nella cartella "data". Successivamente, caricare la label map *object-detection.pbtxt* nella cartella "training". Tale file deve essere costruito in maniera opportuna, come indicato in Appendice. Nella cartella

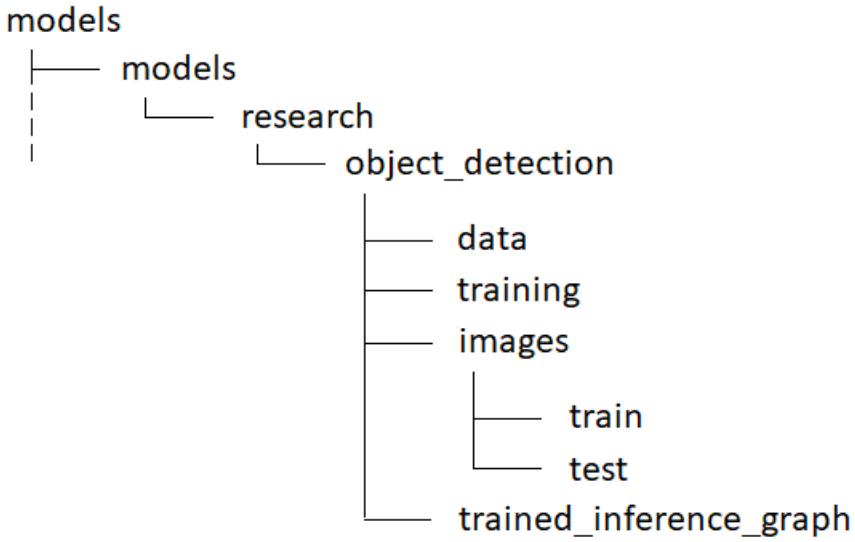


Figura 26: *Schema di cartelle da realizzare in Google Drive*

"object_detection" inserire i file *json_to_csv.py* e *tflite_convert.py*, disponibili in Appendice. Scaricare da [11] *ssd_mobilenet_v2_coco.config* e modificare:

(a) num_classes = numero di classi della label map

(b) type = nome del pre-trained model (nel nostro caso 'ssd_mobilenet_v2')

(c) batch_size = batch size desiderato

(d) fine_tune_checkpoint = path del file *model.ckpt*

(e) input_path for training = path delle immagini di train

(f) input_path for evaluation = path delle immagini di test

(g) label_map_path = path della label map

Dopodichè, caricarlo in "object_detection" e "training".

4. Installare i tools e le dependencies necessarie, poi eseguire il test di TensorFlow 1.x per verificare che tutto sia stato configurato correttamente.

5. Se si usano le annotazioni .xml, eseguire la cella (a) per trasformarle in file .csv . Invece, se si usano label .json convertirle in .csv tramite lo script *json_to_csv.py*, cella (b).

6. Dai file .csv generare i TFRecords tramite il codice *generate_tfrecord.py* (vedere l'Appendice per le modifiche da fare nel caso di utilizzo di una o più classi). Il formato TFRecord è un formato semplice e standard per memorizzare una sequenza di records binari. Selezionare opportunamente i nomi delle classi della label map e i percorsi dei .csv, delle immagini e della cartella in cui salvare i TFRecords.

7. Scaricare il modello pre-addestrato della rete ed estrarlo nel drive (basta farlo una volta sola).

8. Eseguire il codice *model_main.py*. In questo modo verrà eseguito l'addestramento e, dopo un certo numero di epoche, anche il test. Verranno stampate le varie metriche e, quando esse non riusciranno più a crescere, l'algoritmo sarà arrivato a convergenza e il training potrà essere fermato. Al termine dell'addestramento, esportare nel drive il grafico di inferenza della rete. In questo modo, nella cartella "trained_inference_graph" verranno salvati i modelli .pb della network.

9. Attivare TensorBoard, l'interfaccia grafica per controllare le prestazioni della rete. Nella sezione "scalars" sono presenti i grafici di tutte le metriche al variare degli steps, mentre in "images" si possono visualizzare i risultati sulle immagini di test, anch'essi al passare degli steps.

10. Convertire il modello .pb della rete in .tflite, necessario per il caricamento nella board embedded ST. Scaricare il file detect.tflite dal drive, verrà successivamente caricato nella scheda.

4.4 Risultati

Inizialmente è stato utilizzato un piccolo dataset scaricato da Internet, composto da immagini di mele, banane e arance. Dopo aver verificato il corretto funzionamento del programma, è stato utilizzato il dataset FLIR. Inizialmente, attraverso il software *LabelImg* (Fig.27), sono state realizzate "a mano" 900 (su 10228 foto totali) label .xml (file di testo che specificano il nome e le dimensioni dell'immagine, nome delle classi degli oggetti presenti e le coordinate dei loro *bounding boxes*). Solo le persone sono state etichettate. Il re-train della rete è andato a buon fine e con risultati accettabili, come mostrato in

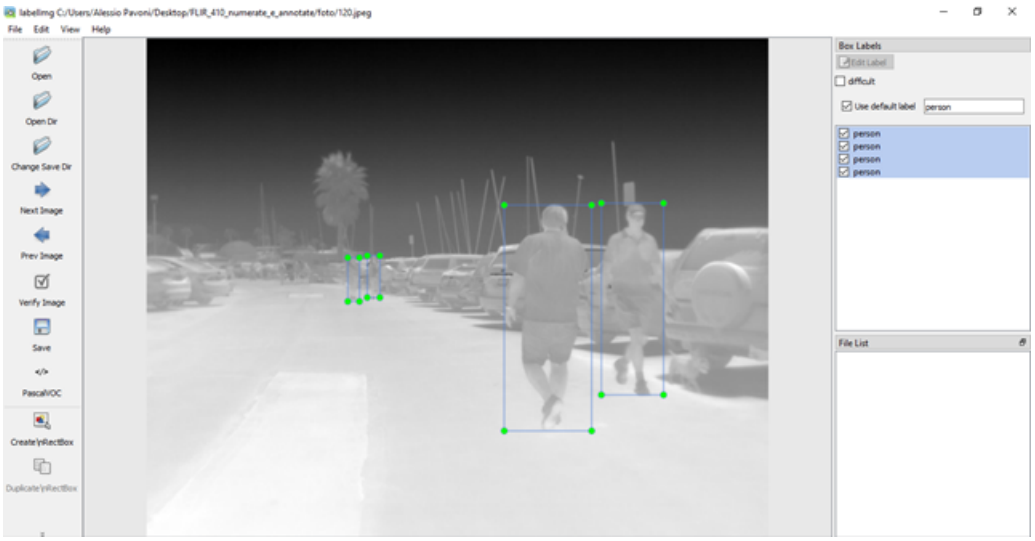


Figura 27: *LabelImg*

Fig.28, ma bisognava trovare un modo per utilizzare l'intero dataset e non solo parte di esso.

Figura 28: *Applicazione della SSD-Mobilenet-v2 su di un'immagine di test (FLIR parziale). Solo la classe "persona" viene riconosciuta poichè è l'unica ad essere stata etichettata*

Quindi, è stato realizzato lo script *json_ to_ csv.py* capace di convertire le annotazioni .json fornite dagli autori del dataset in file .csv (*Comma-Separated Values*), così è stato possibile usare il dataset completo. In questo caso, le immagini sono etichettate sulla base di quattro classi: "person", "bicycle" (bici o motociclette), "car" e "dog".

La rete è stata addestrata per 147128 steps, dove uno step corrisponde alla propagazione di un pacchetto di immagini (*batch*) attraverso la rete. La loss non è riuscita a scendere sotto il valore 3. I risultati in termini di mAP e mAR sono rispettivamente in Fig.29 e Fig.30. Per quanto riguarda la mAP, la *AP@IoU=.50:.05:.95* (primo grafico) non ha superato 0.1. Questo significa che i *bounding boxes* delle localizzazioni non sono molto precisi rispetto a quelli veri. In compenso, però, si nota una mAP sui grandi oggetti ("large") che sfiora 0.45. Ciò vuol dire che la rete individua più precisamente gli items che occupano più spazio nell'immagine, infatti è noto che la SSD presenta maggiori difficoltà nella localizzazione di piccoli oggetti.

Per il mAR si conferma il fatto che la rete riconosce meglio gli items più grandi, come si può vedere dal parametro *AR@100(large)* che supera il 60%. Inoltre, si nota che la rete opera meglio quando ci sono più oggetti presenti nell'immagine. Infatti, mentre *AR@1* (mAR tra le foto che contengono un solo oggetto) è molto basso, *AR@10* e *AR@100*

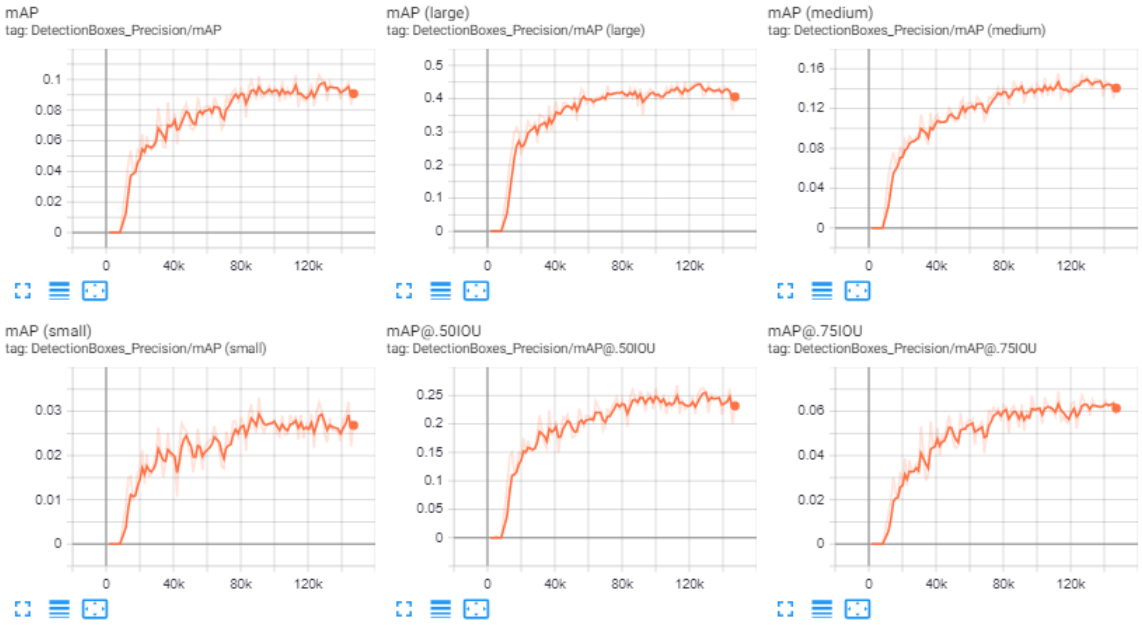


Figura 29: mAP

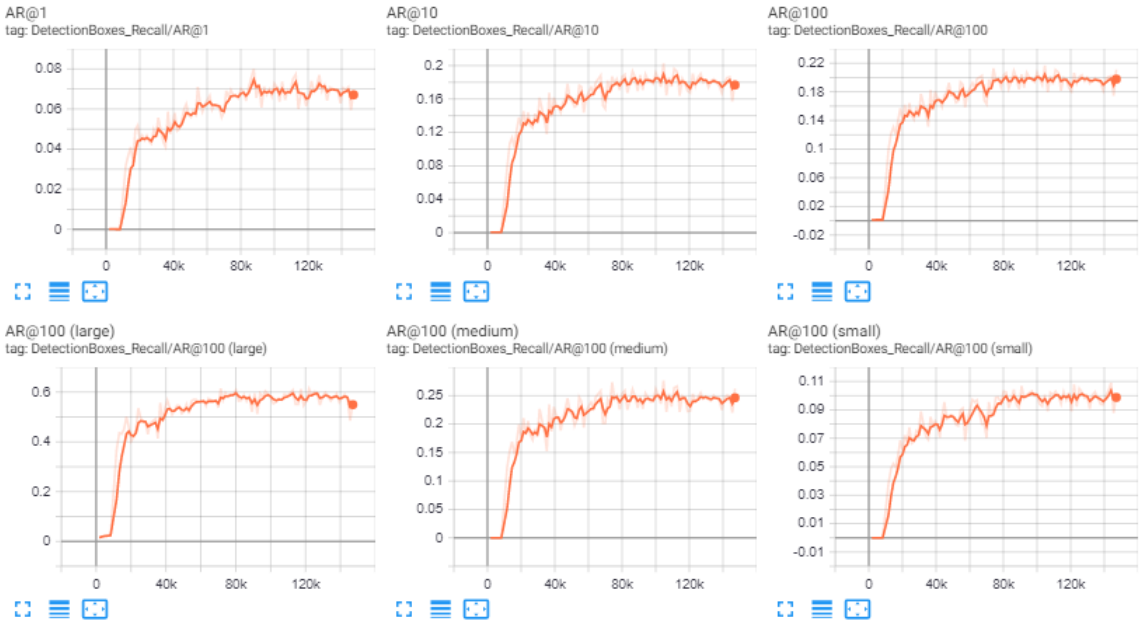


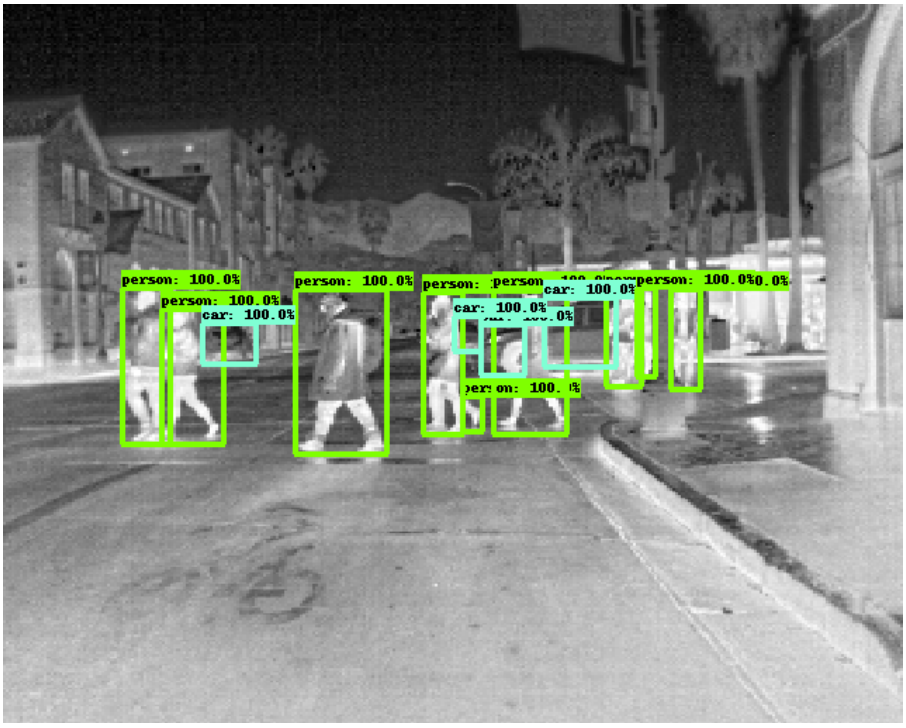
Figura 30: mAR

(mAR tra le foto che contengono fino a dieci o cento oggetti rispettivamente) raggiungono valori di 0.18 e 0.2 rispettivamente.

I risultati su alcune immagini di test sono mostrati in Fig.31, Fig.32 e Fig.33 ((a) è l'output della rete, (b) è l'immagine con le annotazioni corrette). Come si può vedere,

anche utilizzando il dataset FLIR completo le performance della rete sono abbastanza buone. Rispetto al caso del set parziale di immagini FLIR (Fig.28), si nota un calo dei livelli percentuali di sicurezza degli oggetti riconosciuti. Ciò è dovuto al fatto che, mentre prima il dataset prevedeva solo l'esistenza di una classe, ora ne esistono quattro e per questo il task di *Object Detection* risulta più complesso.

(a)



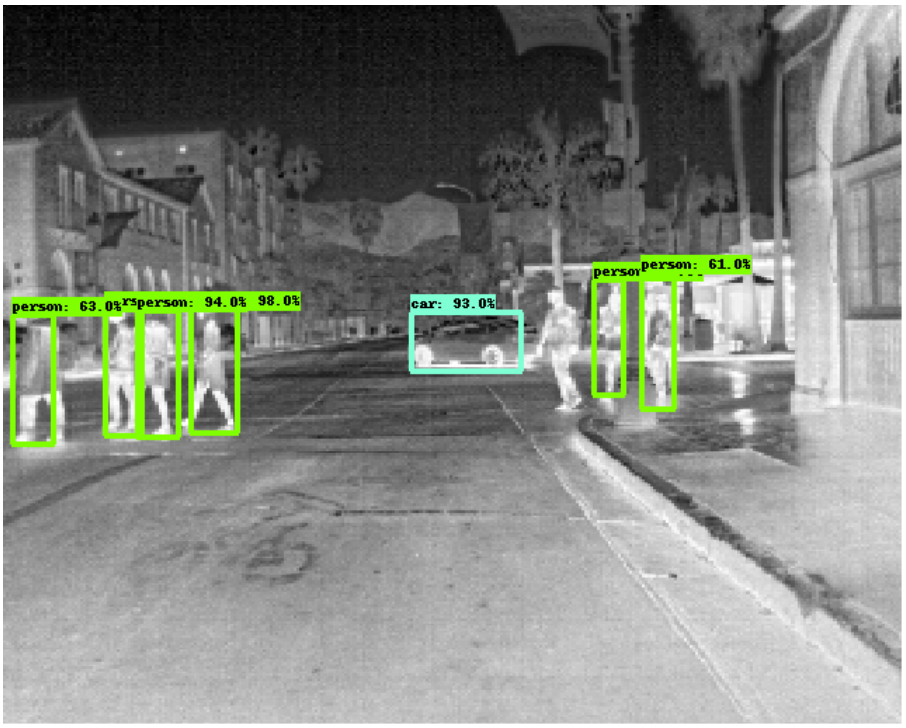
(b)

Figura 31: *Applicazione della SSD-Mobilenet-v2 su di un'immagine di test (FLIR completo): (a) output della rete (b) annotazioni ground-truth*

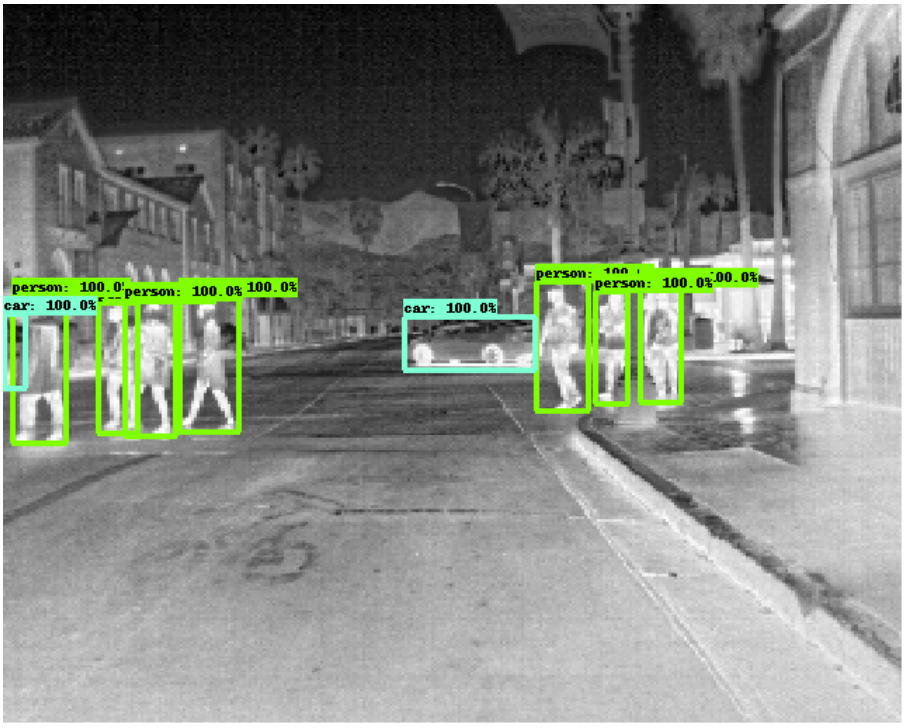
(a)

(b)

Figura 32: *Applicazione della SSD-Mobilenet-v2 su di un'immagine di test (FLIR completo): (a) output della rete (b) annotazioni ground-truth*



(a)



(b)

Figura 33: *Applicazione della SSD-Mobilenet-v2 su di un'immagine di test (FLIR completo): (a) output della rete (b) annotazioni ground-truth*

5 Implementazione su scheda embedded ST

Dopo i buoni risultati conseguiti in seguito all'addestramento e al test, la rete SSD-MobileNet-v2 è pronta per essere caricata sulla scheda embedded ST. A tale scopo, è stata scelta la board NUCLEO-F411RE, già descritta nel Capitolo 1. Vista la ridotta capienza di memoria a disposizione nella scheda, è necessaria una compressione del modello della rete neurale. Proprio in quest'ottica, infatti, al punto 10 il modello .pb della network era stato convertito in formato .tflite, molto più leggero e quindi adatto all'implementazione embedded.

5.1 STM32CubeIDE

L'ambiente di sviluppo della NUCLEO-F411RE si chiama STM32CubeIDE ed è scaricabile da Internet. Una volta aperto l'IDE, aprire un nuovo progetto cliccando File -> New -> STM32 Project. Apparirà una finestra in cui bisognerà selezionare la board e poi cliccare su "Next". Dopo aver scelto il nome del progetto, cliccare su "Finish". Successivamente, bisogna installare l'estensione per l'intelligenza artificiale X-CUBE-AI. Per fare ciò, andare su Software Packs -> Manage Software Packs -> STMicroelectronics -> X-Cube-AI, spuntare la versione più recente e cliccare "Install Now". Adesso cliccare su Software Packs -> Select Components -> STMicroelectronics.X-CUBE-AI. Sotto "Artificial Intelligence X-CUBE-AI" spuntare Core e sotto "Artificial Intelligence Applications" selezionare Validation nella riga "Applications", infine cliccare "OK". A questo punto, sotto Pinout & Configuration -> Categories sarà comparsa la voce "Software Packs". Selezionarla e cliccare "STMicroelectronics.X-CUBE-AI", quindi si aprirà a destra una nuova sezione. Cliccare "Add Network", scegliere il nome della rete, selezionare TFLite e caricare col tasto "Browse" il modello .tflite . In basso ci sono tre pulsanti, ognuno con una specifica funzione:

- Analyze: il programma verifica se la scheda è in grado di ospitare il modello della rete e la compatibilità delle librerie.
- Validate on desktop: vengono confrontate le accuratezze del modello generato in linguaggio C e del modello originale. Se la differenza è abbastanza piccola significa che il codice C è valido.
- Validate on target: vengono confrontate le accuratezze del modello in C caricato nella scheda e del modello originale. Se la differenza è abbastanza piccola significa che il codice C è valido.

Selezionando Analyze, nel caso in cui la rete sia troppo grande, verrà visualizzato l'errore di Fig.34. In questo caso, infatti, il file .tflite pesa circa 18 MB, decisamente troppi per la board. Per questo motivo è possibile comprimere ulteriormente la rete di un fattore 4 o 8, riducendo ovviamente l'accuratezza del modello. Così facendo la rete può essere caricata nella scheda, come mostrato in Fig.35 e Fig.36. Se l'analisi va a buon fine, andare su "Project Manager", impostare il Minimum Heap Size a 0x2000 e premere "GENERATE CODE" per generare il modello in linguaggio C (*porting*). Quindi si può procedere con le validazioni.

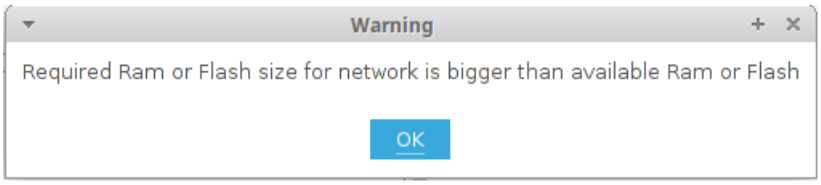


Figura 34: *La rete è troppo pesante per essere caricata sulla scheda*

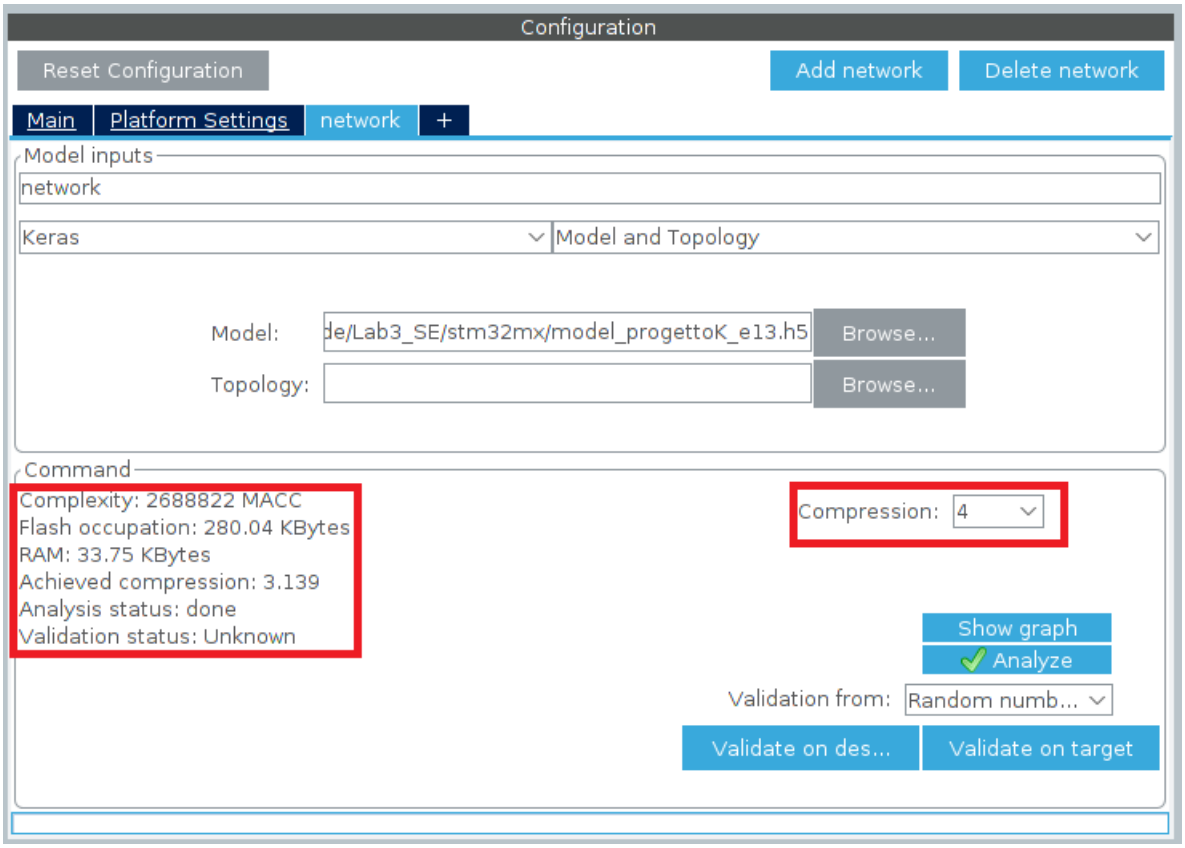


Figura 35: *Fattore 4 di compressione. Le dimensioni del modello sono minori delle capienze massime di RAM e Flash della scheda*

5.2 Limiti del tool ST e possibili soluzioni

In ogni caso, dopo l'analisi del modello, nella finestra pop-up verrà visualizzato l'errore di Fig.37. Ciò è dovuto al fatto che il tool della ST, nell'ultima release, riesce a tradurre la MobileNet-v2, ma non riesce a tradurre il layer "detection" della SSD. Il problema non era previsto e non ha permesso il caricamento della rete sulla board. Le possibili soluzioni all'inconveniente sono due:

- Dato che il problema risiede nell'API "detection" della SSD, si può provare a separare MobileNet-v2 e SSD. La prima può essere convertita con successo con

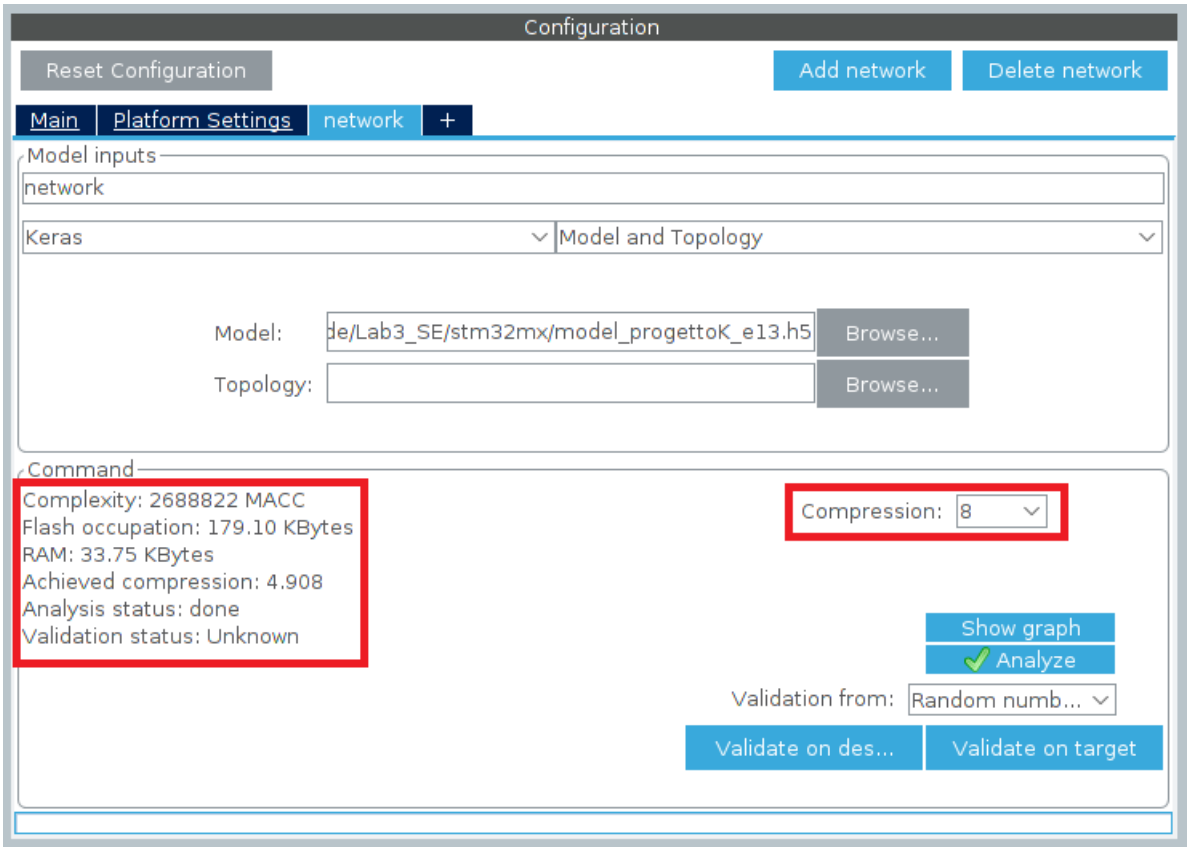


Figura 36: *Fattore 8 di compressione. Le dimensioni del modello sono minori delle capienze massime di RAM e Flash della scheda*

STM32CubeIDE, mentre la seconda deve essere tradotta manualmente in linguaggio C. Il processo è molto laborioso, ma al momento è l'unica possibile soluzione.

- Aspettare le prossime release del software. Il team dell'ST sta infatti lavorando alla traduzione del layer "detection", quindi prossimamente sarà possibile convertire la rete per intero. Purtroppo, però, ad oggi non ci sono ancora notizie al riguardo.

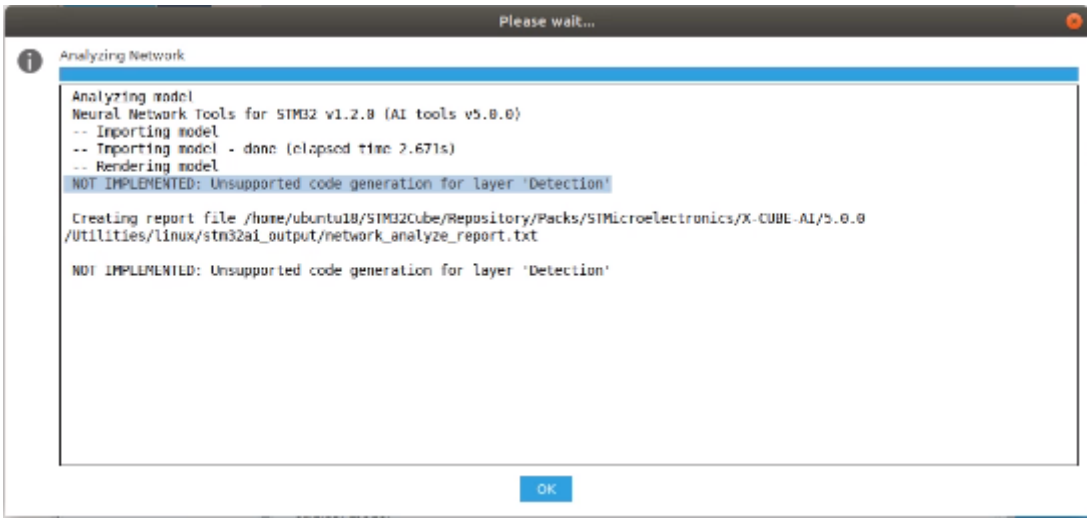


Figura 37: *Il porting non può essere effettuato, il software non riesce a tradurre il layer "detection"*

Conclusioni

In questa tesi sono state addestrate e testate due reti CNN, entrambe con l'obiettivo di riconoscere e localizzare persone all'interno delle immagini termiche del dataset FLIR. Sono stati utilizzati due approcci differenti, l'*Image Segmentation* e l'*Object Detection*. Il primo task consiste nella classificazione *pixel-wise* delle foto, mentre il secondo nella classificazione e localizzazione degli oggetti presenti nelle immagini.

La prima rete per l'*Image Segmentation* è stata addestrata per classificare in maniera binaria (persona o non persona) ogni pixel. Sono stati utilizzati due dataset, il FLIR (label realizzate manualmente) e "university" (fornito dagli autori della network). Essendo la rete creata per il set di immagini university, utilizzando tale dataset sono state ottenute buone performance. Nel caso del FLIR, invece, le prestazioni sono molto inferiori. Ciò è dovuto maggiormente alle differenze tra le immagini, visto che quelle di "university" sono state scattate da una posizione fissa in condizioni di media illuminazione, mentre quelle del FLIR sono state ottenute in diurna e in notturna da un'auto in movimento. In particolare, si notano diversi falsi positivi e molti falsi negativi, confermati da una bassa precision e da un pessimo recall.

La seconda rete per l'*Object Detection*, la SSD-MobileNet-v2, è stata inizialmente addestrata per classificare e localizzare solo le persone, utilizzando parzialmente il dataset FLIR (label realizzate manualmente). Questa volta i risultati sono molto migliori, con livelli di confidenza vicini al 100%. In seguito, tramite la realizzazione dello script Python *json_ to_ csv.py*, è stato possibile utilizzare l'intero set di immagini. Quindi la rete è stata addestrata per riconoscere quattro diverse classi di oggetti: persone, bicicli, auto e cani. I risultati sono comunque positivi, anche se leggermente peggiori per via della presenza di un numero maggiore di classi. In particolare, si notano migliori performance nel caso di foto con items grandi e numerosi.

Infine, è stato utilizzato il software STM32CubeIDE per caricare la seconda rete nella scheda embedded NUCLEO-F411RE. Nonostante il modello fosse già stato compresso nella conversione in formato .tflite, a causa delle ridotte dimensioni di memoria della board è stata necessaria un'ulteriore compressione. Purtroppo, per via di problemi del tool ST, non è stato possibile caricare la rete sulla scheda. In particolare il software non è in grado di tradurre in linguaggio C il layer "detection" della SSD. Oltre che attendere una nuova release dell'IDE che risolva l'inconveniente, l'unica soluzione possibile al momento è quella di scomporre la MobileNet-v2 dalla SSD e convertirle singolarmente in C, nel primo caso utilizzando il software e nel secondo caso attraverso una traduzione manuale.

Concludendo, si è dimostrato che le potenzialità delle reti neurali convoluzionali si adattano anche al riconoscimento di immagini termiche stradali. Questo risultato è molto utile soprattutto nell'ambito della guida autonoma. In tal caso, a causa dei limiti di occupazione di memoria, particolare attenzione va riposta al tipo di scheda embedded che si intende utilizzare. Specificamente alle board dell'STMicroelectronics, non è ancora possibile installare reti di detection in maniera semplice e ciò di fatto complica l'utilizzo di tali schede in questi settori.

Appendice

train.py (se si usano i checkpoint de-commentare le righe 129 e 132 e commentare la 131. Viceversa se non si usano.)

```
1  import numpy
2  import os
3  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
4  import tensorflow as tf
5  slim = tf.contrib.slim
6  import h5py
7  import network
8  import argparse
9
10 parser = argparse.ArgumentParser()
11 envarg = parser.add_argument_group('Training params')
12 envarg.add_argument("--num_epochs", type=int, default=150, help="number of
       training epochs")
13 envarg.add_argument("--batch_norm_epsilon", type=float, default=1e-5, help="batch
       norm epsilon argument for batch normalization")
14 envarg.add_argument('--batch_norm_decay', type=float, default=0.9997, help='batch
       norm decay argument for batch normalization.')
15 envarg.add_argument("--number_of_classes", type=int, default=2, help="Number of
       classes to be predicted.")
16 envarg.add_argument("--l2_regularizer", type=float, default=0.0001, help="l2
       regularizer parameter.")
17 envarg.add_argument('--starting_learning_rate', type=float, default=1e-3, help="
       initial learning rate.")
18 envarg.add_argument("--multi_grid", type=list, default=[1,2,4], help="Spatial
       Pyramid Pooling rates")
19 envarg.add_argument("--output_stride", type=int, default=4, help="Spatial Pyramid
       Pooling rates")
20 envarg.add_argument("--gpu_id", type=int, default=0, help="Id of the GPU to be
       used")
21 envarg.add_argument("--crop_size", type=int, default=513, help="Image Cropsize.")
22 envarg.add_argument("--resnet_model", default="resnet_v2_0", choices=["resnet_v2_0
       ", "resnet_v2_50", "resnet_v2_101", "resnet_v2_152", "resnet_v2_200"], help="
       Resnet model to use as feature extractor. Choose one of: resnet_v2_50 or
       resnet_v2_101")
23 envarg.add_argument("--current_best_val_loss", type=int, default=99999, help="Best
        validation loss value.")
24 envarg.add_argument("--accumulated_validation_miou", type=int, default=0, help="
       Accumulated validation intersection over union.")
25 trainarg = parser.add_argument_group('Training')
26 trainarg.add_argument("--batch_size", type=int, default=5, help="Batch size for
       network train.")
27 trainarg.add_argument("--dataset", type=str, default="beach", help="Dataset
       directory name")
28 trainarg.add_argument("--use_history", help="use historical data input", action="
       store_true")
29 trainarg.add_argument("--use_original", help="use original network architecture",
       action="store_true")
30 args = parser.parse_args()
31
32 if args.use_history:
33         f = h5py.File('dataset/%s/data_history.h5'%args.dataset,'r')
34 else:
35         f = h5py.File('dataset/%s/data.h5'%args.dataset,'r')
36 if args.use_original:
37         args.resnet_model = "resnet_v2_50"
38         args.output_stride = 16
39 train_img = f['train_img'][:].astype(numpy.float32)
40 train_labels = f['train_labels'][:].astype(numpy.int32)
```

```
test_img = f['test_img'][:].astype(numpy.float32)
test_labels = f['test_labels'][:].astype(numpy.int32)
f.close()

print('train',train_img.shape)
print('test',test_img.shape)
imscale = train_img.shape[1]
imchannels = train_img.shape[-1]

class MyNet:
        def __init__(self):
                self.is_training_pl = tf.placeholder(tf.bool, shape=[])
                self.input_pl = tf.placeholder(tf.float32, shape=[args.batch_size,imscale,imscale,imchannels])
                self.label_pl = tf.placeholder(tf.int32, shape=[args.batch_size,imscale,imscale])
                logits_tf = tf.cond(self.is_training_pl, true_fn= lambda: network.deeplab_v3(self.input_pl, args, is_training=True, reuse=False),
                    false_fn=lambda: network.deeplab_v3(self.input_pl, args, is_training=False, reuse=True))

                val_tp = tf.reduce_sum(tf.cast(tf.math.logical_and(tf.math.equal(tf.argmax(logits_tf, axis=-1), 1), tf.math.equal(self.label_pl, 1)),tf.int32))
                val_fp = tf.reduce_sum(tf.cast(tf.math.logical_and(tf.math.equal(tf.argmax(logits_tf, axis=-1), 1), tf.math.equal(self.label_pl, 0)),tf.int32))
                val_fn = tf.reduce_sum(tf.cast(tf.math.logical_and(tf.math.equal(tf.argmax(logits_tf, axis=-1), 0), tf.math.equal(self.label_pl, 1)),tf.int32))
                self.val_precision = tf.cast(val_tp,tf.float32) / tf.cast(val_tp + val_fp + 1, tf.float32)
                self.val_recall = tf.cast(val_tp,tf.float32) / tf.cast(val_tp + val_fn + 1, tf.float32)

                logits_reshaped = tf.reshape(logits_tf, (args.batch_size*imscale*imscale,2))
                labels_reshaped = tf.reshape(self.label_pl, [args.batch_size*imscale*imscale])
                pos_mask = tf.where(tf.cast(labels_reshaped, tf.bool))
                neg_mask = tf.where(tf.cast(1 - labels_reshaped, tf.bool))
                self.pos_loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=tf.gather_nd(logits_reshaped, pos_mask), labels=tf.gather_nd(labels_reshaped, pos_mask)))
                self.neg_loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=tf.gather_nd(logits_reshaped, neg_mask), labels=tf.gather_nd(labels_reshaped, neg_mask)))
                if args.use_original:
                        self.loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits_reshaped, labels=labels_reshaped))
                else:
                        self.loss = self.pos_loss + self.neg_loss
                batch = tf.Variable(0)
                optimizer = tf.train.AdamOptimizer(args.starting_learning_rate)
                self.train_op = optimizer.minimize(self.loss, global_step=batch)

def augment(input_images, input_labels): #random shift
        shift_width = 10
        for i in range(len(input_images)):
                dx = numpy.random.randint(-shift_width, shift_width)
                dy = numpy.random.randint(-shift_width, shift_width)
                #initialize to mean pixel value
                shifted_image = numpy.ones((imscale, imscale, imchannels), dtype=numpy.float32) * 72
                shifted_label = numpy.zeros((imscale, imscale), dtype=numpy.int32)
```

```
                x1_src = max(dx, 0)
                x2_src = min(dx+imscale, imscale)
                y1_src = max(dy, 0)
                y2_src = min(dy+imscale, imscale)
                x1_dst = max(-dx, 0)
                x2_dst = min(-dx+imscale, imscale)
                y1_dst = max(-dy, 0)
                y2_dst = min(-dy+imscale, imscale)
                shifted_image[y1_dst:y2_dst, x1_dst:x2_dst, :] = input_images[i, y1_src:y2_src, x1_src:x2_src, :]
                input_images[i,:,:,:] = shifted_image
                shifted_label[y1_dst:y2_dst, x1_dst:x2_dst] = input_labels[i, y1_src:y2_src, x1_src:x2_src]
                input_labels[i,:,:] = shifted_label

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.allow_soft_placement = True
config.log_device_placement = False
sess = tf.Session(config=config)
net = MyNet()
#variables_to_restore = slim.get_variables_to_restore(exclude=[args.resnet_model + "/logits", "optimizer_vars",
#                                                               "DeepLab_v3/ASPP_layer", "DeepLab_v3/logits"])
#restorer = tf.train.Saver(variables_to_restore)
#restorer.restore(sess, "./resnet/checkpoints/" + args.resnet_model + ".ckpt")
#print("Model checkpoints for " + args.resnet_model + " restored!")

################################################
# https://medium.com/@prajwal.prashanth22/google-colab-drive-as-persistent-storage-for-long-training-runs-cb82bc1d5b71
# salvataggio checkpoints
#from keras.callbacks import *
#filepath = "/content/drive/My Drive/MyCNN/epochs:{epoch:03d}-val_acc:{val_acc:.3f}.hdf5"
#checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
#callbacks_list = [checkpoint]
################################################
saver = tf.train.Saver()
if args.use_original:
        MODEL_PATH = 'dataset/%s/original_model.ckpt' % args.dataset
elif args.use_history:
        print('use_history')
        MODEL_PATH = 'dataset/%s/history/model.ckpt' % args.dataset
else:
        MODEL_PATH = 'dataset/%s/model.ckpt' % args.dataset

# riprende i checkpoint precedentemente salvati. ATTIVARE RIGA SE SI USANO I CHECKPOINTS
#saver.restore(sess,tf.train.latest_checkpoint('/content/drive/My Drive/IR/IR_detection-master/dataset/FLIR'))

init = tf.global_variables_initializer() # DISATTIVARE RIGA SE SI USANO I CHECKPOINTS
#init = tf.local_variables_initializer() # ATTIVARE RIGA SE SI USANO I CHECKPOINTS
sess.run(init, {})
for epoch in range(args.num_epochs):
        idx = numpy.arange(len(train_labels))
        numpy.random.shuffle(idx)
        input_points = numpy.zeros((args.batch_size, imscale, imscale, 3))

```

```
        loss_arr = []
        pl_arr = []
        nl_arr = []
        prc_arr = []
        rcl_arr = []
        num_batches = int(len(train_labels) / args.batch_size)
        for batch_id in range(num_batches):
#               print('batch %d/%d'%(batch_id,num_batches))
                start_idx = batch_id * args.batch_size
                end_idx = (batch_id + 1) * args.batch_size
                input_images = train_img[idx[start_idx:end_idx], :, :, :]
                input_labels = train_labels[idx[start_idx:end_idx], :, :]
                augment(input_images, input_labels)
                _, ls, pl, nl, prc, rcl = sess.run([net.train_op, net.loss, net.pos_loss, net.neg_loss, net.val_precision, net.val_recall], {net.input_pl:input_images, net.label_pl:input_labels, net.is_training_pl:True})
                loss_arr.append(ls)
                pl_arr.append(pl)
                nl_arr.append(nl)
                prc_arr.append(prc)
                rcl_arr.append(rcl)
        print("Epoch %d loss %.2f(%.2f+%.2f) prc %.2f rcl %.2f"%(epoch,numpy.mean(loss_arr),numpy.mean(pl_arr),numpy.mean(nl_arr),numpy.mean(prc_arr),numpy.mean(rcl_arr)))
        saver.save(sess, '/content/drive/My Drive/IR/IR_detection-master/dataset/%s/checkpoint_epoch%d.ckpt'%(args.dataset,epoch))
        #tf.compat.v2.keras.model.fit_generator(datagen.flow(x_train, y_train, batch_size=64), epochs=epochs, verbose=1, validation_data=(x_test, y_test), callbacks=callbacks_list)
#       if epoch % 10 ==9:
        if False:
                loss_arr = []
                pl_arr = []
                nl_arr = []
                prc_arr = []
                rcl_arr = []
                num_batches = int(len(test_labels) / args.batch_size)
                for batch_id in range(num_batches):
#                       print('batch %d/%d'%(batch_id,num_batches))
                        start_idx = batch_id * args.batch_size
                        end_idx = (batch_id + 1) * args.batch_size
                        input_images = test_img[start_idx:end_idx, :, :, :]
                        input_labels = test_labels[start_idx:end_idx, :, :]
                        ls, pl, nl, prc, rcl = sess.run([net.loss, net.pos_loss, net.neg_loss, net.val_precision, net.val_recall], {net.input_pl:input_images, net.label_pl:input_labels, net.is_training_pl:False})
                        loss_arr.append(ls)
                        pl_arr.append(pl)
                        nl_arr.append(nl)
                        prc_arr.append(prc)
                        rcl_arr.append(rcl)
                print("Validation %d loss %.2f(%.2f+%.2f) prc %.2f rcl %.2f"%(epoch,numpy.mean(loss_arr),numpy.mean(pl_arr),numpy.mean(nl_arr),numpy.mean(prc_arr),numpy.mean(rcl_arr)))

saver.save(sess, MODEL_PATH)
```

json_to_csv.py

```
import argparse

parser = argparse.ArgumentParser()
arg = parser.add_argument_group('params')
arg.add_argument("--json_path", type=str, default="/content/drive/My Drive/models/models/research/object_detection/data/thermal_annotations.json", help="Path to JSON label file")
args = parser.parse_args()

def convert_coco_json_to_csv(filename):
    import pandas as pd
    import json

    # COCO2017/annotations/instances_val2017.json
    s = json.load(open(filename, 'r'))
    out_file = filename[:-5] + '.csv'
    out = open(out_file, 'w')
    out.write('filename,width,height,class,xmin,ymin,xmax,ymax\n')


    for ann in s['annotations']:
        image_id = ann['image_id']
        #all_ids_ann.append(image_id)
        xmin = ann['bbox'][0]
        xmax = ann['bbox'][0] + ann['bbox'][2]
        ymin = ann['bbox'][1]
        ymax = ann['bbox'][1] + ann['bbox'][3]
        label = ann['category_id']
        for name in s['images']:
          if name['id'] == image_id:
            image_name = name['file_name']
            image_name = image_name.replace("thermal_8_bit/","")
            #all_names_ann.append(image_name)
            width = name['width']
            height = name['height']
        for lab in s['categories']:
          if lab['id'] == label:
            class_name = lab['name']
        out.write('{},{},{},{},{},{},{},{}\n'.format(image_name, width, height, class_name, xmin, ymin, xmax, ymax))

    out.close()

    # Sort file by image id
    s1 = pd.read_csv(out_file)
    s1.sort_values('filename', inplace=True)
    s1.to_csv(out_file, index=False)

convert_coco_json_to_csv(args.json_path)
```

generate_ tfrecord.py (Se si usano più di una classe de-commentare le righe da 38 a 45 e da 25 a 28 e commentare 23, 35 e 36. Viceversa se si usa una sola classe. Scrivere il corretto formato delle immagini nella riga 59.)

```
"""
Usage:
# Create train data:
python generate_tfrecord.py --label=<LABEL> --csv_input=<
    PATH_TO_ANNOTATIONS_FOLDER>/train_labels.csv  --output_path=<
    PATH_TO_ANNOTATIONS_FOLDER>/train.record
# Create test data:
python generate_tfrecord.py --label=<LABEL> --csv_input=<
    PATH_TO_ANNOTATIONS_FOLDER>/test_labels.csv  --output_path=<
    PATH_TO_ANNOTATIONS_FOLDER>/test.record
"""
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import
import os
import io
import pandas as pd
import tensorflow as tf
import sys
sys.path.append("/content/drive/My Drive/models/models/research")
from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict
flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
#flags.DEFINE_string('label', '', 'Name of class label')
# if your image has more labels input them as
flags.DEFINE_string('label0', '', 'Name of class[0] label')
flags.DEFINE_string('label1', '', 'Name of class[1] label')
flags.DEFINE_string('label2', '', 'Name of class[2] label')
flags.DEFINE_string('label3', '', 'Name of class[3] label')
# and so on.
flags.DEFINE_string('img_path', '', 'Path to images')
FLAGS = flags.FLAGS
# TO-DO replace this with label map
# for multiple labels add more else if statements
def class_text_to_int(row_label):
    #if row_label == FLAGS.label:  # 'person':
    #    return 1
    # comment upper if statement and uncomment these statements for multiple
    labelling
    if row_label == FLAGS.label0: #person
      return 1
    elif row_label == FLAGS.label1: #bicycle
      return 2
    elif row_label == FLAGS.label2: #car
      return 3
    elif row_label == FLAGS.label3: #dog
      return 4
    else:
        None
def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys
    (), gb.groups)]
def create_tf_example(group, path):
```

```
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as
    fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size
    filename = group.filename.encode('utf8')
    image_format = b'jpeg'
    # check if the image format is matching with your images.
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []
    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))
    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example
def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(os.getcwd(), FLAGS.img_path)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))
if __name__ == '__main__':
    tf.compat.v1.app.run()
```

tflite_convert.py

```
# Copyright 2018 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
"""Python command line interface for running TOCO."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import argparse
import os
import sys
from tensorflow.contrib.lite.python import lite
from tensorflow.contrib.lite.toco import toco_flags_pb2 as _toco_flags_pb2
from tensorflow.contrib.lite.toco import types_pb2 as _types_pb2
from tensorflow.python.platform import app
def _parse_array(values):
  if values:
    return values.split(",")
def _parse_int_array(values):
  if values:
    return [int(val) for val in values.split(",")]
def _parse_set(values):
  if values:
    return set(values.split(","))
def _get_toco_converter(flags):
  """Makes a TocoConverter object based on the flags provided.
  Args:
    flags: argparse.Namespace object containing TFLite flags.
  Returns:
    TocoConverter object.
  """
  # Parse input and output arrays.
  input_arrays = _parse_array(flags.input_arrays)
  input_shapes = None
  if flags.input_shapes:
    input_shapes_list = [
        _parse_int_array(shape) for shape in flags.input_shapes.split(":")
    ]
    input_shapes = dict(zip(input_arrays, input_shapes_list))
  output_arrays = _parse_array(flags.output_arrays)
  converter_kwargs = {
      "input_arrays": input_arrays,
      "input_shapes": input_shapes,
      "output_arrays": output_arrays
  }
  # Create TocoConverter.
  if flags.graph_def_file:
    converter_fn = lite.TocoConverter.from_frozen_graph
    converter_kwargs["graph_def_file"] = flags.graph_def_file
  elif flags.saved_model_dir:
    converter_fn = lite.TocoConverter.from_saved_model
    converter_kwargs["saved_model_dir"] = flags.saved_model_dir
```

```
63      converter_kwargs["tag_set"] = _parse_set(flags.saved_model_tag_set)
64      converter_kwargs["signature_key"] = flags.saved_model_signature_key
65    return converter_fn(**converter_kwargs)
66  def _convert_model(flags):
67    """Calls function to convert the TensorFlow model into a TFLite model.
68    Args:
69      flags: argparse.Namespace object.
70    Raises:
71      ValueError: Invalid flags.
72    """
73    # Create converter.
74    converter = _get_toco_converter(flags)
75    if flags.inference_type:
76      converter.inference_type = _types_pb2.IODataType.Value(flags.inference_type)
77    if flags.inference_input_type:
78      converter.inference_input_type = _types_pb2.IODataType.Value(
79          flags.inference_input_type)
80    if flags.output_format:
81      converter.output_format = _toco_flags_pb2.FileFormat.Value(
82          flags.output_format)
83    if flags.mean_values and flags.std_dev_values:
84      input_arrays = converter.get_input_arrays()
85      std_dev_values = _parse_int_array(flags.std_dev_values)
86      mean_values = _parse_int_array(flags.mean_values)
87      quant_stats = zip(mean_values, std_dev_values)
88      if ((not flags.input_arrays and len(input_arrays) > 1) or
89          (len(input_arrays) != len(quant_stats))):
90        raise ValueError("Mismatching --input_arrays, --std_dev_values, and "
91                         "--mean_values. The flags must have the same number of "
92                         "items. The current input arrays are '{0}'. "
93                         "--input_arrays must be present when specifying "
94                         "--std_dev_values and --mean_values with multiple input "
95                         "tensors in order to map between names and "
96                         "values.".format(",".join(input_arrays)))
97      converter.quantized_input_stats = dict(zip(input_arrays, quant_stats))
98    if (flags.default_ranges_min is not None) and (flags.default_ranges_max is
99                                                   not None):
100     converter.default_ranges_stats = (flags.default_ranges_min,
101                                       flags.default_ranges_max)
102   if flags.drop_control_dependency:
103     converter.drop_control_dependency = flags.drop_control_dependency
104   if flags.reorder_across_fake_quant:
105     converter.reorder_across_fake_quant = flags.reorder_across_fake_quant
106   if flags.change_concat_input_ranges:
107     converter.change_concat_input_ranges = flags.change_concat_input_ranges
108   if flags.allow_custom_ops:
109     converter.allow_custom_ops = flags.allow_custom_ops
110   if flags.quantize_weights:
111     converter.quantize_weights = flags.quantize_weights
112   if flags.dump_graphviz_dir:
113     converter.dump_graphviz_dir = flags.dump_graphviz_dir
114   if flags.dump_graphviz_video:
115     converter.dump_graphviz_vode = flags.dump_graphviz_video
116   # Convert model.
117   output_data = converter.convert()
118   with open(flags.output_file, "wb") as f:
119     f.write(output_data)
120 def _check_flags(flags, unparsed):
121   """Checks the parsed and unparsed flags to ensure they are valid.
122   Raises an error if previously support unparsed flags are found. Raises an
123   error for parsed flags that don't meet the required conditions.
124   Args:
125     flags: argparse.Namespace object containing TFLite flags.
```

```
126       unparsed: List of unparsed flags.
127     Raises:
128       ValueError: Invalid flags.
129     """
130     # Check unparsed flags for common mistakes based on previous TOCO.
131     def _get_message_unparsed(flag, orig_flag, new_flag):
132       if flag.startswith(orig_flag):
133         return "\n  Use {0} instead of {1}".format(new_flag, orig_flag)
134       return ""
135     if unparsed:
136       output = ""
137       for flag in unparsed:
138         output += _get_message_unparsed(flag, "--input_file", "--graph_def_file")
139         output += _get_message_unparsed(flag, "--savedmodel_directory",
140                                         "--saved_model_dir")
141         output += _get_message_unparsed(flag, "--std_value", "--std_dev_values")
142         output += _get_message_unparsed(flag, "--batch_size", "--input_shapes")
143         output += _get_message_unparsed(flag, "--dump_graphviz",
144                                         "--dump_graphviz_dir")
145       if output:
146         raise ValueError(output)
147     # Check that flags are valid.
148     if flags.graph_def_file and (not flags.input_arrays or
149                                  not flags.output_arrays):
150       raise ValueError("--input_arrays and --output_arrays are required with "
151                        "--graph_def_file")
152     if flags.input_shapes:
153       if not flags.input_arrays:
154         raise ValueError("--input_shapes must be used with --input_arrays")
155       if flags.input_shapes.count(":") != flags.input_arrays.count(","):
156         raise ValueError("--input_shapes and --input_arrays must have the same "
157                          "number of items")
158     if flags.std_dev_values or flags.mean_values:
159       if bool(flags.std_dev_values) != bool(flags.mean_values):
160         raise ValueError("--std_dev_values and --mean_values must be used "
161                          "together")
162       if flags.std_dev_values.count(",") != flags.mean_values.count(","):
163         raise ValueError("--std_dev_values, --mean_values must have the same "
164                          "number of items")
165     if (flags.default_ranges_min is None) != (flags.default_ranges_max is None):
166       raise ValueError("--default_ranges_min and --default_ranges_max must be "
167                        "used together")
168   def run_main(_):
169     """Main in toco_convert.py."""
170     parser = argparse.ArgumentParser(
171         description=("Command line tool to run TensorFlow Lite Optimizing "
172                      "Converter (TOCO)."))
173     # Output file flag.
174     parser.add_argument(
175         "--output_file",
176         type=str,
177         help="Full filepath of the output file.",
178         required=True)
179     # Input file flags.
180     input_file_group = parser.add_mutually_exclusive_group(required=True)
181     input_file_group.add_argument(
182         "--graph_def_file",
183         type=str,
184         help="Full filepath of file containing TensorFlow GraphDef.")
185     input_file_group.add_argument(
186         "--saved_model_dir",
187         type=str,
188         help="Full filepath of directory containing the SavedModel.")
```

```
    # Model format flags.
    parser.add_argument(
        "--output_format",
        type=str.upper,
        choices=["TFLITE", "GRAPHVIZ_DOT"],
        help="Output file format.")
    parser.add_argument(
        "--inference_type",
        type=str.upper,
        choices=["FLOAT", "QUANTIZED_UINT8"],
        help="Target data type of arrays in the output file.")
    parser.add_argument(
        "--inference_input_type",
        type=str.upper,
        choices=["FLOAT", "QUANTIZED_UINT8"],
        help=("Target data type of input arrays. Allows for a different type for "
              "input arrays in the case of quantization."))
    # Input and output arrays flags.
    parser.add_argument(
        "--input_arrays",
        type=str,
        help="Names of the output arrays, comma-separated.")
    parser.add_argument(
        "--input_shapes",
        type=str,
        help="Shapes corresponding to --input_arrays, colon-separated.")
    parser.add_argument(
        "--output_arrays",
        type=str,
        help="Names of the output arrays, comma-separated.")
    # SavedModel related flags.
    parser.add_argument(
        "--saved_model_tag_set",
        type=str,
        help=("Comma-separated set of tags identifying the MetaGraphDef within "
              "the SavedModel to analyze. All tags must be present. "
              "(default \"serve\")"))
    parser.add_argument(
        "--saved_model_signature_key",
        type=str,
        help=("Key identifying the SignatureDef containing inputs and outputs. "
              "(default DEFAULT_SERVING_SIGNATURE_DEF_KEY)"))
    # Quantization flags.
    parser.add_argument(
        "--std_dev_values",
        type=str,
        help=("Standard deviation of training data for each input tensor, "
              "comma-separated. Used for quantization. (default None)"))
    parser.add_argument(
        "--mean_values",
        type=str,
        help=("Mean of training data for each input tensor, comma-separated. "
              "Used for quantization. (default None)"))
    parser.add_argument(
        "--default_ranges_min",
        type=int,
        help=("Default value for min bound of min/max range values used for all "
              "arrays without a specified range, Intended for experimenting with "
              "quantization via \"dummy quantization\". (default None)"))
    parser.add_argument(
        "--default_ranges_max",
        type=int,
        help=("Default value for max bound of min/max range values used for all "
```

```
            "arrays without a specified range, Intended for experimenting with "
            "quantization via \"dummy quantization\". (default None)"))
  parser.add_argument(
      "--quantize_weights",
      type=bool,
      help=("Store float weights as quantized weights followed by dequantize "
            "operations. Inference is still done in FLOAT, but reduces model "
            "size (at the cost of accuracy and latency)."))
  # Graph manipulation flags.
  parser.add_argument(
      "--drop_control_dependency",
      action="store_true",
      help=("Boolean indicating whether to drop control dependencies silently. "
            "This is due to TensorFlow not supporting control dependencies. "
            "(default True)"))
  parser.add_argument(
      "--reorder_across_fake_quant",
      action="store_true",
      help=("Boolean indicating whether to reorder FakeQuant nodes in "
            "unexpected locations. Used when the location of the FakeQuant "
            "nodes is preventing graph transformations necessary to convert "
            "the graph. Results in a graph that differs from the quantized "
            "training graph, potentially causing differing arithmetic "
            "behavior. (default False)"))
  parser.add_argument(
      "--change_concat_input_ranges",
      action="store_true",
      help=("Boolean to change behavior of min/max ranges for inputs and "
            "outputs of the concat operator for quantized models. Changes the "
            "ranges of concat operator overlap when true. (default False)"))
  parser.add_argument(
      "--allow_custom_ops",
      action="store_true",
      help=("Boolean indicating whether to allow custom operations. When false "
            "any unknown operation is an error. When true, custom ops are "
            "created for any op that is unknown. The developer will need to "
            "provide these to the TensorFlow Lite runtime with a custom "
            "resolver. (default False)"))
  # Logging flags.
  parser.add_argument(
      "--dump_graphviz_dir",
      type=str,
      help=("Full filepath of folder to dump the graphs at various stages of "
            "processing GraphViz .dot files. Preferred over --output_format="
            "GRAPHVIZ_DOT in order to keep the requirements of the output "
            "file."))
  parser.add_argument(
      "--dump_graphviz_video",
      action="store_true",
      help=("Boolean indicating whether to dump the graph after every graph "
            "transformation"))
  tflite_flags, unparsed = parser.parse_known_args(args=sys.argv[1:])
  try:
    _check_flags(tflite_flags, unparsed)
  except ValueError as e:
    parser.print_usage()
    file_name = os.path.basename(sys.argv[0])
    sys.stderr.write("{0}: error: {1}\n".format(file_name, str(e)))
    sys.exit(1)
  _convert_model(tflite_flags)
def main():
  app.run(main=run_main, argv=sys.argv[:1])
if __name__ == "__main__":
```

```
315    main()
```

object-detection.pbtxt (I nomi delle classi sono quelli presenti nelle annotazioni .json o .xml)

```
item {
  id: 1
  name: 'person'
}
item {
  id: 2
  name: 'bicycle'
}
item {
  id: 3
  name: 'car'
}
item {
  id: 4
  name: 'dog'
}
```

Sitografia

[1]
FLIR Systems
Dataset termico FLIR GRATUITO per l'addestramento di algoritmi
https://www.flir.it/oem/adas/adas-dataset-form/

[2]
Khust Patel
Custom Object Detection using TensorFlow from Scratch
https://towardsdatascience.com/custom-object-detection-using-tensorflow-from-scratch-e61da2e10087

[3]
J. Park, J. Chen, Y. K. Cho, D. Y. Kang, B. J. Son
CNN-Based Person Detection Using Infrared Images for Night-time Intrusion Warning Systems
https://www.mdpi.com/1424-8220/20/1/34/htm

[4]
J. Park, J. Chen, Y. K. Cho, D. Y. Kang, B. J. Son
Codici Python per IR_detection
https://github.com/jingdao/IR_detection

[5]
J. Park, J. Chen, Y. K. Cho, D. Y. Kang, B. J. Son
Dataset "university"
https://www.dropbox.com/s/3n4y112uhsonx8a/university_data.zip?dl=1

[6] Autore sconosciuto
main_IR_detection.ipynb e Training an Object Detection Model.ipynb
https://univpm-my.sharepoint.com/:f:/g/personal/p001116_staff_univpm_it/Erp5h6tEw_9Hh606MnT7mUYBGNlHUv53FFpnI7RJ6gyNfg?e=mLAabn

[7]
Nathaniel O Solomon
Training an Object Detection Model with TensorFlow API using Google COLAB
https://medium.com/analytics-vidhya/training-an-object-detection-model-with-tensorflow-api-using-google-colab-4f9a688d5e8b

[8]
Rafael Padilla
Metrics for object detection
https://github.com/rafaelpadilla/Object-Detection-Metrics

[9]
Jonathan Hui
mAP (mean Average Precision) for Object Detection
https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

[10]
COCO dataset
Detection Evaluation
https://cocodataset.org/#detection-eval

[11]
Autore Sconosciuto
ssd_mobilenet_v2_coco.config
https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_mobilenet_v2_coco.config

A conclusione di questo elaborato, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Un ringraziamento speciale al professor Turchetti e alla dottoressa Falaschetti, i miei relatori, per la loro immensa pazienza, per i loro indispensabili consigli, per le conoscenze trasmesse durante tutto il percorso di tirocinio.

Ringrazio i miei genitori, fonte di sostegno e di coraggio, che mi hanno trasmesso la passione per lo studio e la voglia di raggiungere questo traguardo più di qualsiasi altra cosa. Senza mia madre e mio padre, non avrei avuto la possibilità di studiare e arrivare fin qui. Ringrazio anche i nonni e tutta la mia famiglia per la vicinanza dimostrata in questi anni.

Desidero ringraziare anche Marco, Luca e tutti i miei amici per l'incredibile sostegno durante questa esperienza. In particolare vorrei menzionare i ragazzi del gruppo Aspiranti Ingegneri, grazie per tutti i momenti di conforto e di divertimento.

Infine, vorrei dedicare questo piccolo traguardo ad Alessandra.