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:

Prof. **Claudio Turchetti**

Correlatore:

Prof. **Laura Falaschetti**

Tesi di laurea di

Mirko Ercoli

A.A 2019/2020

Indice

Introduzione

Lo studio condotto è relativo al machine learning e consiste nell'implementazione di una rete neurale per l'identificazione di persone in immagini acquisite tramite termocamera.

Il machine learning (o apprendimento automatico) è una branca dell'intelligenza artificiale che raccoglie un insieme di metodi sviluppati a partire dagli ultimi decenni del XX secolo in varie comunità scientifiche, sotto diversi nomi: statistica computazionale, riconoscimento di pattern, reti neurali artificiali, filtraggio adattivo, teoria dei sistemi dinamici, elaborazione delle immagini, data mining, algoritmi adattivi, ecc; che utilizza metodi statistici per migliorare progressivamente la performance di un algoritmo nell'identificare un pattern nei dati.

Le reti neurali artificiali fanno parte del machine learning e in questo elaborato verranno argomentate le principali reti esistenti come la CNN, R-CNN, FAST R-CNN, FASTER R-CNN e SSD.

Verranno proposti due metodi di implementazione per la rete neurale convoluzionale (CNN) e due applicativi; uno dei quali permette di osservare il funzionamento della rete mettendo in evidenza con diversi grafici la precisione e la perdita durante differenti fasi di allenamento. Il secondo invece, consente di rilevare oggetti e persone all'interno di un'immagine. Infine verranno esposte le linee guida di come una rete neurale può essere implementata in un sistema embedded ST.

L'applicazione finale si potrebbe collocare nell'ambito della guida autonoma in quanto consiste nel riconoscimento di pedoni in visione notturna.

Che cos'è una rete Neurale

Le reti neurali sono **modelli di calcolo matematico-informatici** basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni; tali interconnessioni derivano da neuroni artificiali e processi di calcolo basati sul modello delle scienze cognitive chiamato "connessionismo" (basati su **PDP** – Parallel Distributed Processing, tradotto "elaborazione a parallelismo distribuito" delle informazioni.)

L'inspirazione per le reti neurali deriva dagli studi sui meccanismi di elaborazione dell'informazione nel sistema nervoso biologico, in particolare il cervello umano.

Una rete neurale biologica riceve dati e segnali esterni (nell'uomo e nell'animale vengono percepiti attraverso i sensi grazie a complesse organizzazioni di cellule nervose che hanno "compiti" differenti come la percezione dell'ambiente, il riconoscimento degli stimoli, ecc.); questi vengono elaborati in informazioni attraverso un imponente numero di neuroni (che rappresentano la capacità di calcolo) interconnessi tra loro in una struttura non-lineare e variabile in risposta a quei dati e stimoli esterni stessi. È in quest'ottica che si parla dunque di "modello" matematico-informatico.

Allo stesso modo, le reti neurali artificiali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione: ricevono segnali esterni su uno strato di nodi (che rappresenta l'unità di elaborazione, il processore); ognuno di questi "nodi d'ingresso" è collegato a svariati nodi interni della rete che, tipicamente, sono organizzati a più livelli in modo che ogni singolo nodo possa elaborare i segnali ricevuti trasmettendo ai livelli successivi il risultato delle sue elaborazioni (quindi delle informazioni più evolute).

In linea di massima, le reti neurali sono formate da tre strati (che però possono coinvolgere migliaia di neuroni e decine di migliaia di connessioni):

1) **lo strato degli ingressi** (*I – Input*): è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete;

2) **il cosiddetto strato** (*H – hidden strato nascosto*): è quello che ha in carica il processo di elaborazione vero e proprio (e può anche essere strutturato con più colonne-livelli di neuroni);

3) **lo strato di uscita** (*O – Output*): qui vengono raccolti i risultati dell'elaborazione dello strato H e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

Affinché questo processo risulti performante è necessario "addestrare" le reti neurali, ossia fare in modo che apprendano come comportarsi nel momento in cui andrà risolto un problema ingegneristico, come per esempio il riconoscimento di una persona nell'analisi delle immagini.

Il tema dell'apprendimento è collegato al Machine Learning, inteso come algoritmi che utilizzano metodi matematico-computazionali per apprendere informazioni dall'esperienza (quindi in modo automatico e adattivo).

Questi i principali modelli in uso oggi:

– ***apprendimento supervisionato*** (Supervised Learning): all'algoritmo vengono forniti sia set di dati come input sia le informazioni relative ai risultati desiderati con l'obiettivo che la rete identifichi una regola generale che colleghi i dati in ingresso con quelli in uscita; in altre parole vengono forniti degli esempi di input e di output in modo il sistema che impari il nesso tra loro e ne estrapoli una regola riutilizzabile per altri compiti simili;

– ***apprendimento non supervisionato*** (Unsupervised Learning): al sistema vengono forniti solo set di dati senza alcuna indicazione del risultato desiderato. Lo scopo di questo secondo metodo di apprendimento è "risalire" a schemi e modelli nascosti, ossia identificare negli input una struttura logica senza che questi siano preventivamente etichettati;

– ***apprendimento per rinforzo***: in questo caso, il sistema deve interagire con un ambiente dinamico (che gli consente di avere i dati di input) e raggiungere un obiettivo (al raggiungimento del quale riceve una ricompensa), imparando anche dagli errori (identificati medianti "punizioni"). Il comportamento (e le prestazioni) del sistema è determinato da una routine di apprendimento basata su ricompensa e punizione;

– ***apprendimento semi-supervisionato***: è un modello "ibrido" dove al computer viene fornito un set di dati incompleti per l'allenamento/apprendimento; alcuni di questi input sono "dotati" dei rispettivi esempi di output (come nell'apprendimento supervisionato), altri invece ne sono privi (come nell'apprendimento non supervisionato). L'obiettivo, di fondo, è sempre lo stesso: identificare regole e funzioni per la risoluzione dei problemi, nonché modelli e strutture di dati utili a raggiungere determinati obiettivi.

Le reti neurali offrono un insieme di strumenti molto potenti che permettono di risolvere problemi nell'ambito della classificazione, della regressione e del controllo non-lineare. Un semplice esempio è l'analisi di un'immagine per verificare la presenza di determinati oggetti ed eventualmente di riconoscerli. Ad esempio, potrebbe essere fornito un set di immagini ad infrarosso (fornite da FLIR, azienda leader nel mondo del commercio delle termocamere) e di rilevare in esse la presenza di pedoni. Oltre ad avere un'elevata velocità di elaborazione le reti neurali hanno la capacità di imparare la soluzione da un determinato insieme di esempi.

Tuttavia *i principali svantaggi* nascono:

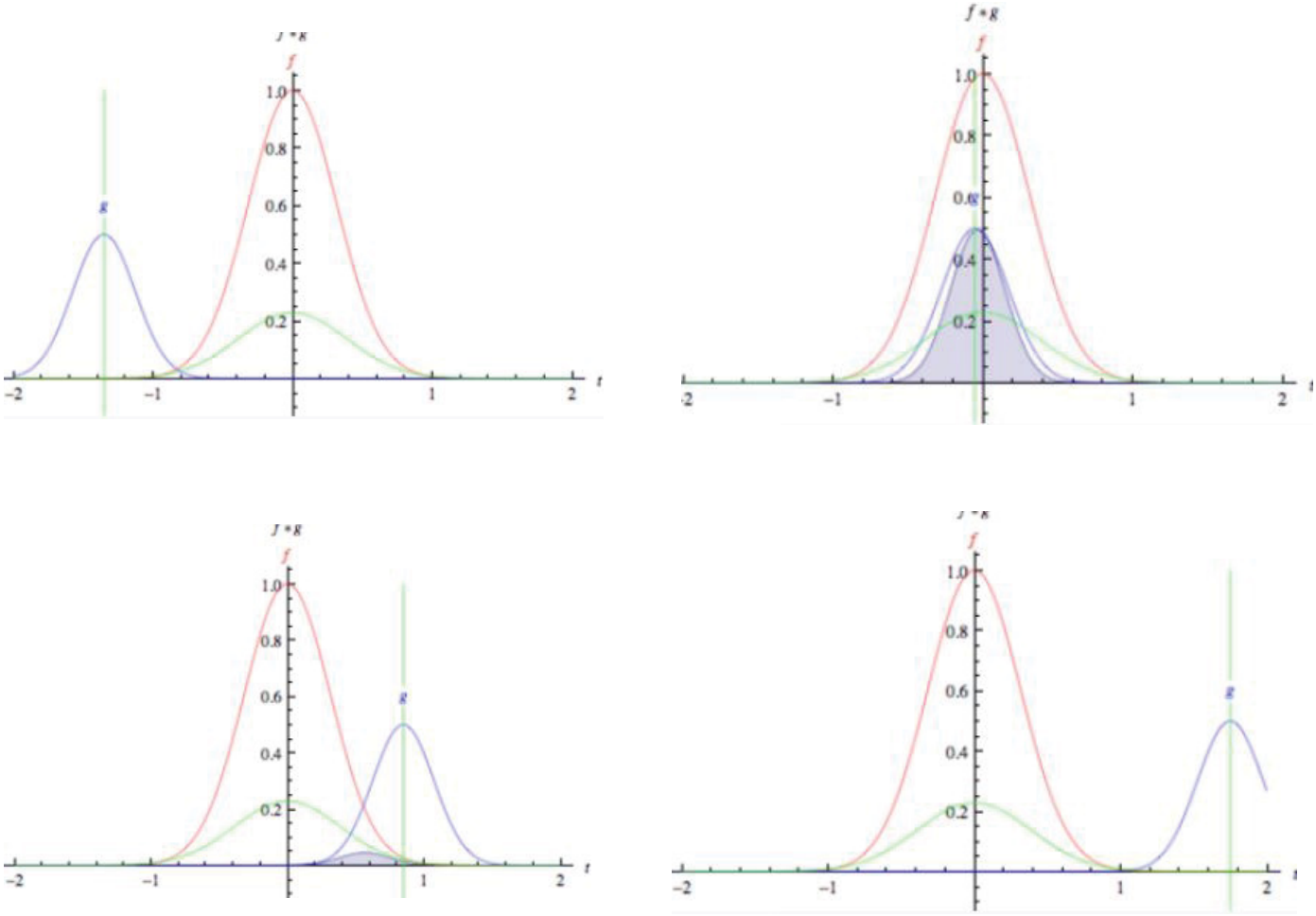
a) *dal bisogno di scegliere un adeguato insieme di esempi*;
b) *da quando la rete deve rispondere ad ingressi sostanzialmente diversi da quelli dell'insieme degli esempi*.

Dalle loro origini ad oggi, sono stati sviluppati numerosi modelli di reti neurali per risolvere problemi di diversa natura, tuttavia si presenteranno principalmente tre modelli di rete convoluzionale CNN, Fast R-CNN, Faster R-CNN e SSD. Essi fanno parte di una classe di modelli di reti generica conosciuta come reti feedforward, ovvero reti in cui l'informazione viaggia in un'unica direzione e non sono presenti cicli.

La rete neurale convoluzionale CNN

Le Reti Neurali Convoluzionali, o ConvNet (CNN) sono uno degli algoritmi di Deep Learning più utilizzati oggi nella computer vision e trovano applicazione in tantissimi campi: dalle automobili autonome ai droni, dalle diagnosi mediche al supporto e trattamento per gli ipovedenti.

Matematicamente parlando, la parola "*convoluzione*" significa "*far scorrere*" una funzione (nel grafico sottostante di colore blu) sopra un'altra (rossa), di fatto "mescolandole" assieme. Il risultato sarà una funzione (verde) che rappresenta il prodotto delle due funzioni.

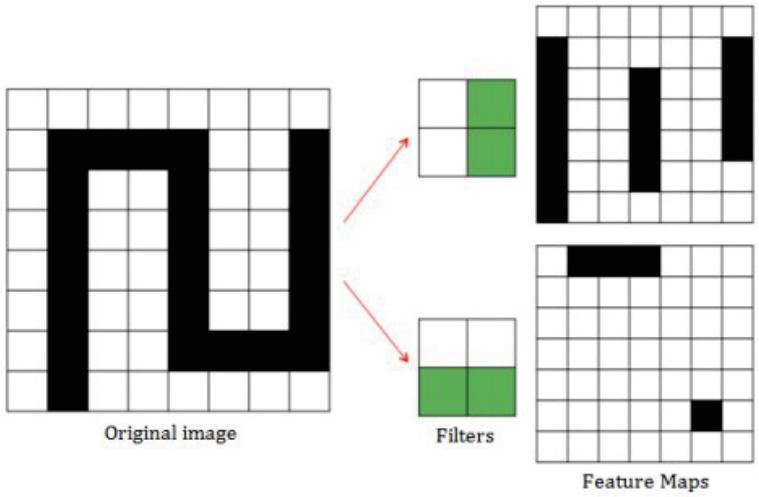


Quindi si può dire che il concetto che si pone alla base di una CNN è che sia possibile analizzare un'immagine, riconoscendo in un primo stadio le "*silhouette*" delle figure. Un esempio potrebbe essere quello di far passare sopra all'immagine tre filtri: uno che ne individui le linee verticali, uno quelle orizzontali ed infine uno quelle diagonali.

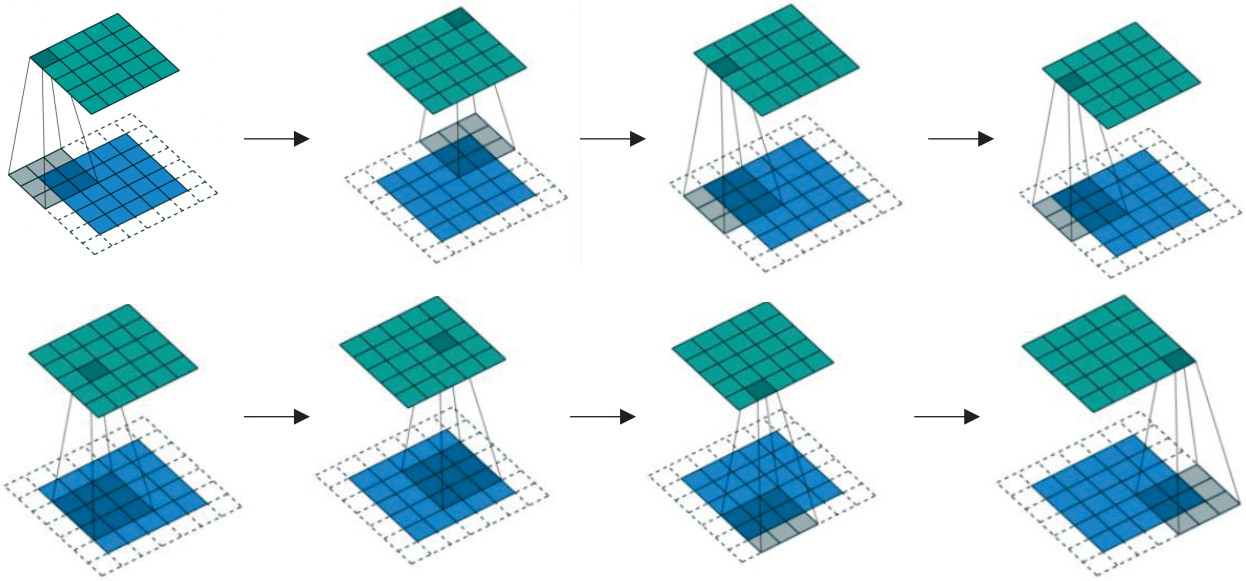
Layer (o strati) successivi potrebbero, per esempio, riconoscere occhi, orecchie, mani, etc. infine, l'ultimo livello potrebbe essere in grado di riconoscere e distinguere animali, persone e automobili.

Le reti convoluzionali funzionano come tutte le reti neurali: un layer di input, uno o più layer nascosti che effettuano calcoli tramite funzioni di attivazione, e un layer di output che fornisce un risultato. La differenza consiste proprio nelle convoluzioni.

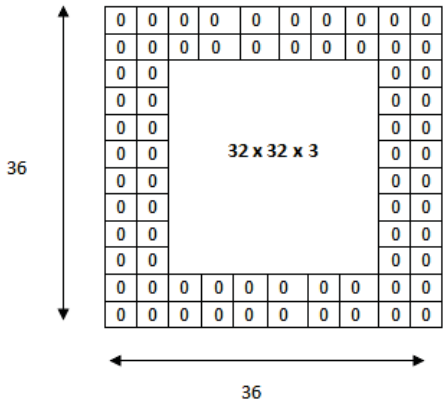
Ogni layer ospita quella che viene chiamata "***feature map***", ovvero la specifica feature che i nodi si preoccupano di cercare. Come nell'esempio in figura il primo layer potrebbe occuparsi di codificare le linee diagonali, quindi si "fa scorrere" un apposito filtro (qui nell'esempio un 2×2) sull'immagine, e moltiplicandolo (prodotto scalare) per l'area sottostante.



Questa moltiplicazione corrisponde alla funzione che si è visto all'inizio, dove l'immagine di input corrisponde alla curva rossa, il filtro alla blu e la feature map alla verde. In sostanza, significa che all'interno del layer convoluzionale ciascun nodo è mappato solo su un sottoinsieme di nodi di input (campo recettivo) e di fatto, moltiplicare il filtro per il campo recettivo di ciascun nodo, equivale concettualmente a "far scorrere" il filtro lungo l'immagine di input (*windowing*).

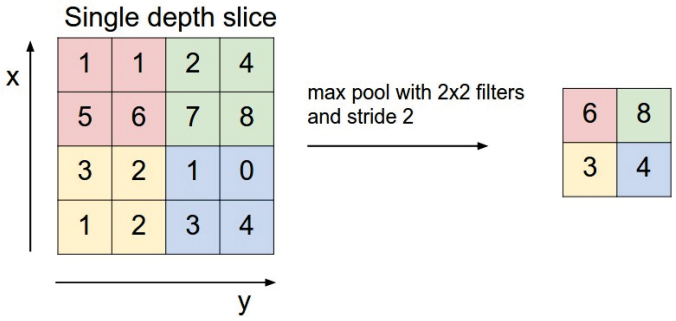


Il risultato sarà un'altra matrice, leggermente più piccola dell'immagine originale, o della stessa grandezza se si usa la tecnica del "**Zero padding"** (come si può osservare nella figura di sinistra è una tecnica che consiste nell'aggiungere (accanto) all'immagine un "bordo" di zeri allo scopo di preservare la dimensione dell'immagine in uscita dal layer, per non perdere informazioni), chiamata appunto feature map.



Ad ogni layer di neuroni si possono applicare eventualmente più filtri, generando quindi più feature map.

Tipicamente ogni layer convoluzionale viene fatto seguire da un livello di **Max-Pooling** (come si può osservare dall'immagine a destra è un metodo per ridurre la dimensione di un'immagine, suddividendola in blocchi e tenendo solo quello col valore più alto. Così facendo si riduce il problema di **overfitting** e si mantengono solo le aree con maggiore attivazione), riducendo via via la dimensione della matrice, ma aumentando il livello di "astrazione". Si passa quindi da filtri elementari, come appunto linee verticali e orizzontali, a filtri via via più sofisticati, in grado ad esempio di riconoscere i fanali, il parabrezza… fino all'ultimo livello dove è in grado di distinguere un'automobile da un camion.

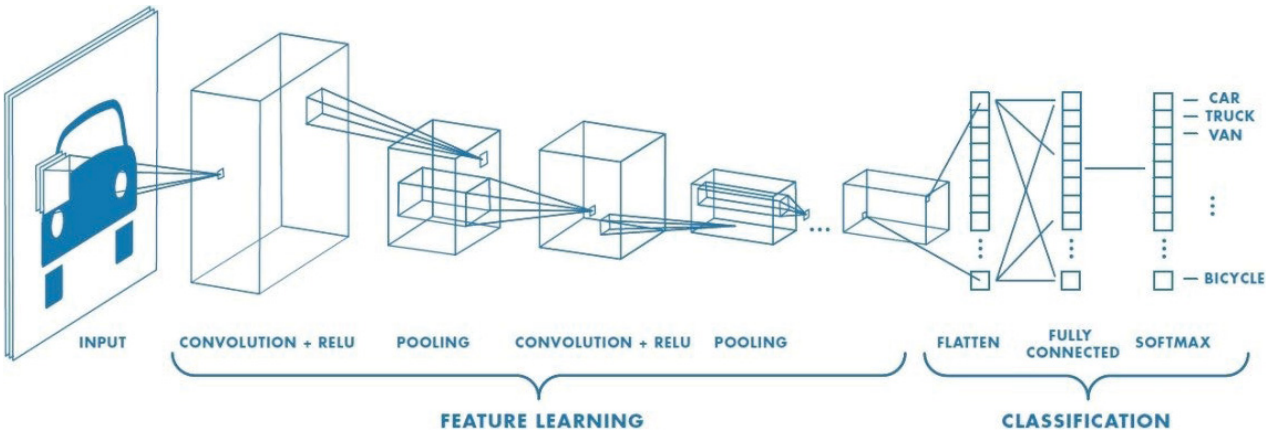


Ricapitolando e precisando si può dire che la rete CNN rispetto ad una semplice rete feed-forward è in grado, pertanto, di trattare informazioni più specifiche ed essere di conseguenza più efficiente.

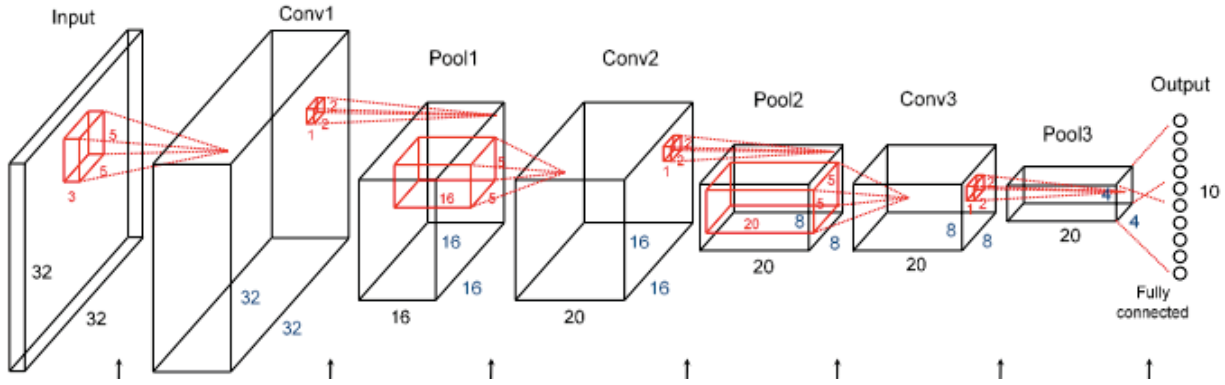
Semplificando pertanto lo schema di funzionamento di una CNN si può dire che la rete è costituita da una catena di blocchi il cui ordine è rappresentato nel modo seguente:

Input->Conv->ReLU->Pool->Conv->ReLU->Pool->ReLU->Conv->ReLU->Pool->FullyConnected

Se si considera, inoltre, che la funzione ReLU è parte integrante del livello Conv si può ridurre la CNN allo schema seguente.



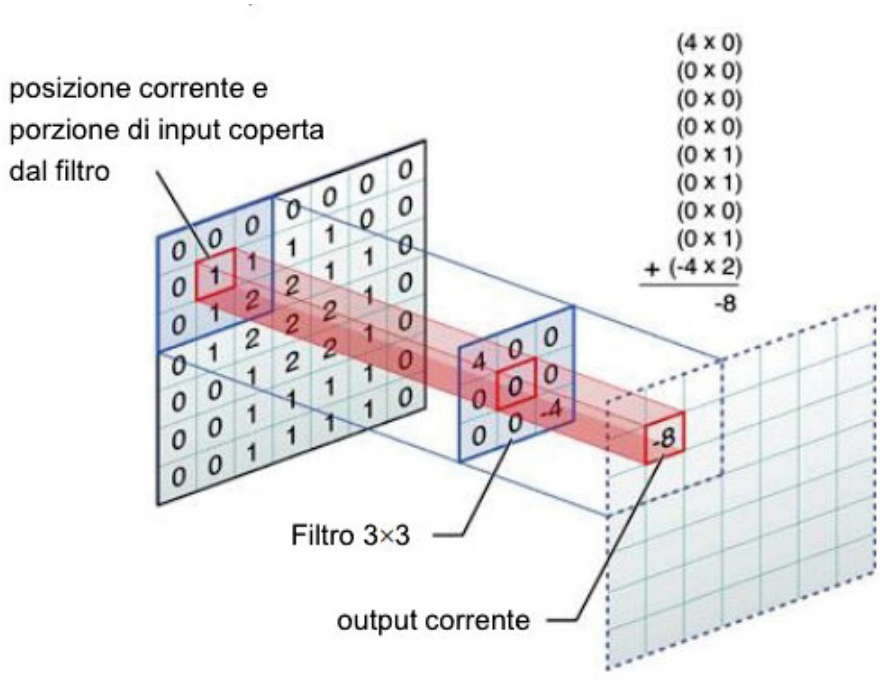
Cerchiamo ora di andare ad identificare il ruolo di ciascuna parte del grafico.



Il ***livello di input*** è costituito da una sequenza di neuroni in grado di ricevere le informazioni dell'immagine da trattare. A questo livello, infatti, verrà passato il vettore di dati che rappresentano i pixel dell'immagine di ingresso. Nel caso venga fornita un'immagine a colori di 32 x 32 pixel il vettore di ingresso dovrà avere una lunghezza di 32 x 32 x 3; in pratica per ogni pixel dell'immagine di dimensione 32 X 32 avremo 3 valori che rappresentano i tre colori dell'immagine in formato RGB (Red, Green e Blue).

Il ***livello convoluzionale*** (***Conv***) è il principale della rete. Il suo obiettivo è quello di individuare schemi, come ad esempio curve, angoli, circonferenze o quadrati raffigurati in un'immagine con elevata precisione. I filtri applicabili sono e possono essere più di uno: maggiore è il loro numero e maggiore è la complessità delle features che si potranno individuare.

Ma come funziona il livello di convoluzione? In pratica un *filtro digitale* (una piccola maschera) è fatta scorrere sulle diverse posizioni dell'immagine in input; per ogni posizione viene generato un valore di output eseguendo il prodotto scalare tra la maschera e la porzione dell'input coperta dal filtro.



Nell’esempio in figura il filtro è rappresentato da una matrice 3×3 pertanto il pennello di scansione prenderà in esame solo una porzione di immagine in ingresso 3×3 e si effettuerà il prodotto con il filtro di convoluzione scelto.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} x \begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix}$$

di seguito il risultato con un esempio di filtro.

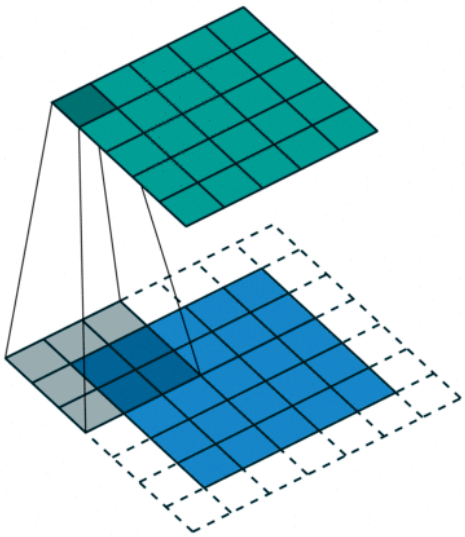
Immagine input

Filtro

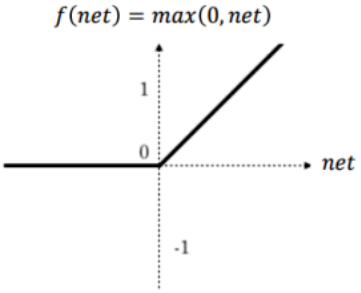
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Immagine output

In definitiva l’immagine verrà scansionata “pezzo per pezzo" ottenendo in uscita una matrice di valori più piccola che rappresenta “l’immagine caratterizzata”



Il ***livello ReLU*** (***Rectified Linear Units***) si pone l'obiettivo di annullare i valori non utili ottenuti nei livelli precedenti ed è posto dopo i livelli convoluzionali. La funzione ReLU è una funzione divenuta ultimamente molto utilizzata, specialmente nei layer intermedi. La ragione risiede nel fatto che si tratta di una funzione molto semplice da calcolare e permette di appiattire a zero la risposta di tutti i valori negativi e di lasciare invariati i valori uguali o superiori a zero.



Il ***livello Pooling*** permette di identificare se la caratteristica di studio è presente nel livello precedente e rende più grezza l'immagine, mantenendo la caratteristica utilizzata dal livello convoluzionale. In altre parole il livello di pooling esegue un'aggregazione delle informazioni, generando feature map di dimensione inferiore (cercando di eliminare "overfitting" ovvero come si può osservare dall'immagine sottostante che la funzione non oscilli troppo attorno ai punti ma sia il più possibile come la figura centrale).

Infine l'ultimo livello è rappresentato dal ***livello FC*** (o **Fully connected**, completamente connesso): che è il livello che esegue di fatto la classificazione delle immagini.

La rete neurale convoluzionale R-CNN

Si può certamente dire che la **R-CNN**, **R**egion-based **C**onvolutional **N**eural **N**etwork, è la rete da cui tutto ha avuto origine. Consiste in tre fasi:

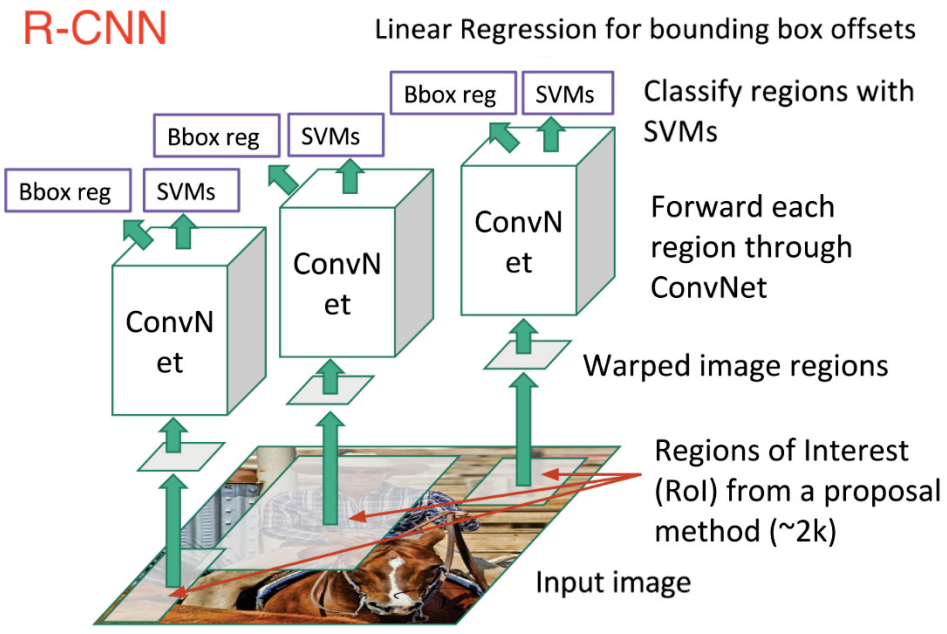
I. **Scansione dell'immagine di input** per la rilevazione di possibili oggetti, utilizzando un algoritmo chiamato *Selective Search*, che estrae approssimativamente 2000 regioni dall'immagine fornita in input (*Region Proposal*).

II. **Esecuzione della rete neurale convoluzionale** (CNN) in cima a ciascuna regione.

III. **Estrapolazione dell'output da ogni CNN**:

   a. *inserimento in una SVM* (*Support Vector Machine*) per classificare la regione;
   b. *applicazione di una regressione lineare* per restringere il riquadro di delimitazione dell'oggetto, se esistente.

Le fasi sono illustrate (dal basso verso l'alto) nella figura sottostante:



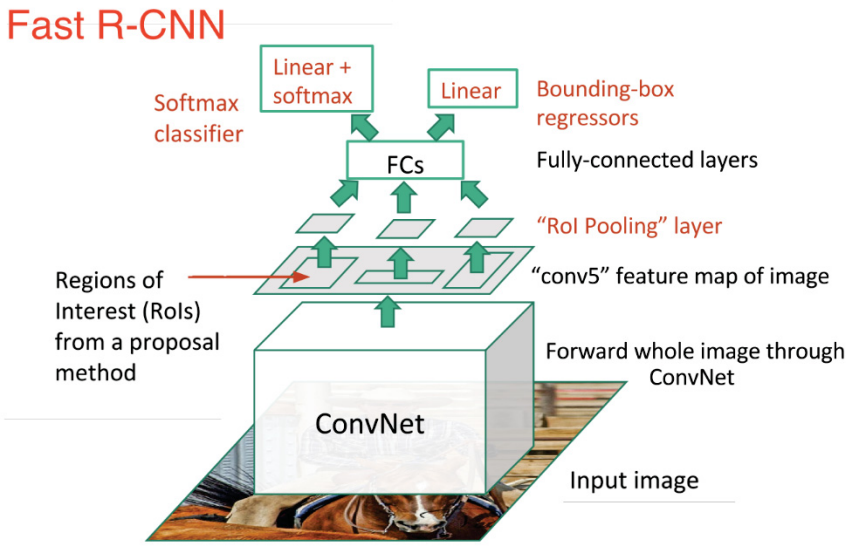
In altre parole, prima si generano le possibili regioni, quindi si estraggono le caratteristiche ed infine si classificano quelle regioni sulla base delle caratteristiche estratte. In sostanza, abbiamo trasformato la rilevazione di oggetti in un problema di classificazione. *La R-CNN risulta molto intuitiva ma, sfortunatamente, anche molto lenta*.

La rete neurale convoluzionale Fast R-CNN

Lo sviluppo immediatamente successivo è la **Fast R-CNN**. Questa ricorda l'originale in molte delle sue caratteristiche ma è migliore in termini di velocità di rilevazione per due aspetti:

1. L'*estrazione delle caratteristiche* viene effettuata utilizzando solo una CNN per l'intera immagine al posto che 2000 CNN sulle 2000 regioni sovrapposte.
2. La *SVM viene sostituita da una funzione Softmax* e, inoltre, la rete neurale viene utilizzata anche per le previsioni anziché soltanto per la creazione di un nuovo modello.

Il modello appare similmente a:



Come si può vedere dall'immagine, stiamo ora generando proposte regionali basate non sull'immagine originale, ma piuttosto sull'ultima mappa delle caratteristiche estrapolate dalla rete. Ciò permette di allenare una sola CNN per l'intera immagine.

In aggiunta, al posto che allenare differenti SVM per classificare ogni oggetto, vi è un singolo strato Softmax che genera direttamente la probabilità per la classe di riferimento. Quindi, *si ha una sola rete neurale da allenare, a differenza di quanto succedeva nel caso precedente dove invece c'era una rete neurale affiancata a molteplici SVM*.

La Fast R-CNN migliora in termini di velocità, ma continua ad avere un grosso difetto: l'algoritmo di Selective Search per proporre le possibili regioni.

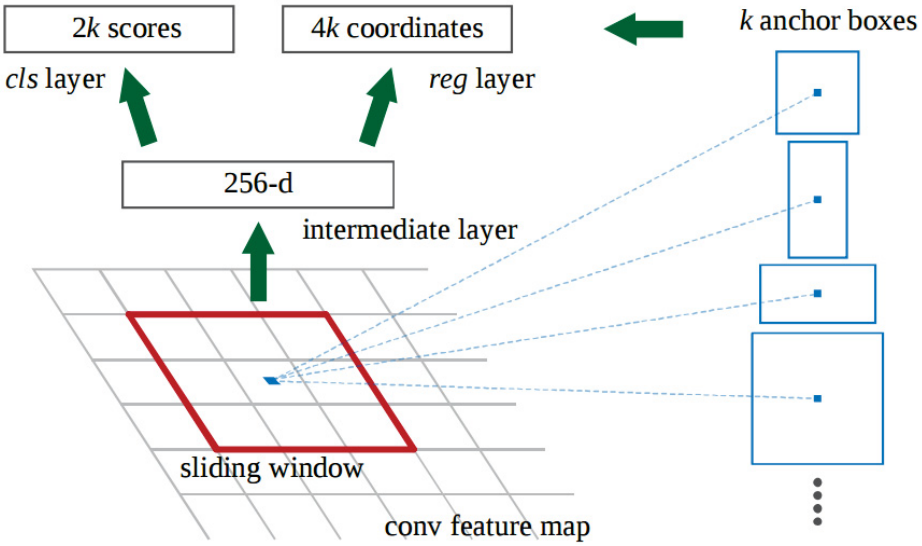
La rete neurale convoluzionale Faster R-CNN

L'intuizione è stata quella di sostituire il lento algoritmo di Selective Search con una rete neurale veloce. Nello specifico essa ha introdotto la **Region Proposal Network** (**RPN**): rete di proposta regionale.

Le principali variazioni introdotte nella rete neurale RPN possono essere così elencate:

- All'ultimo strato di una CNN iniziale, una finestra scorrevole 3×3 si muove attraverso la mappa delle caratteristiche per mapparla poi in una dimensione inferiore (ad esempio *256-d*).
- Per ogni posizione della finestra scorrevole, la RPN genera molteplici regioni possibili basate su vincoli spaziali di dimensioni fisse chiamati ***riquadri di ancoraggio*** (***anchor boxes***).
- Ogni ***proposta regionale*** (RP) consiste in:
  - *un punteggio* (score) per la presenza dell'oggetto in quella determinata regione;
  - *4 coordinate* rappresentanti il riquadro di delimitazione della regione.

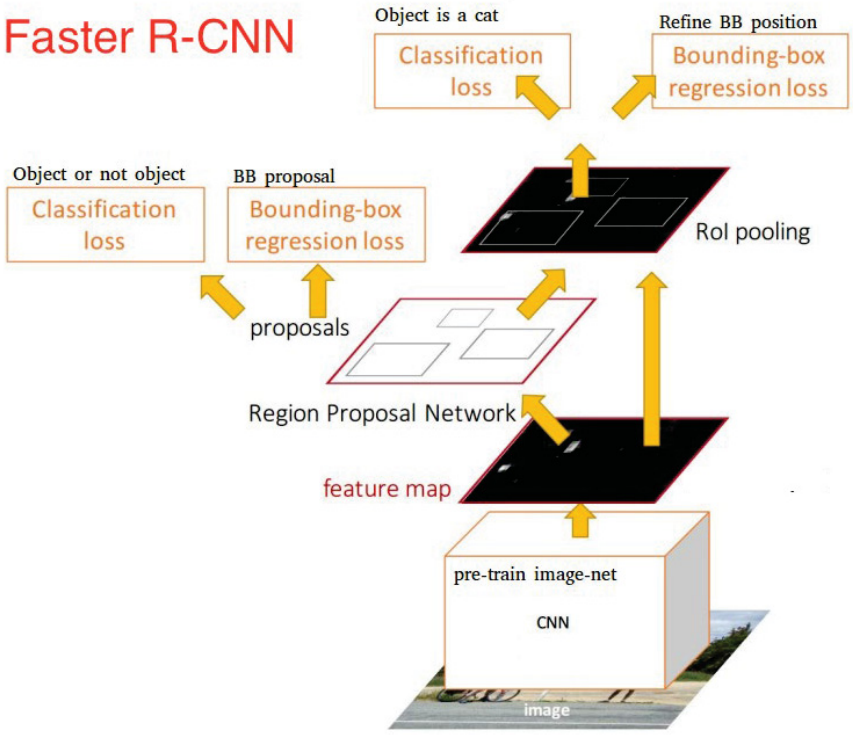
In altre parole, si osserva la regione nell'ultima mappa delle caratteristiche, considerando i differenti k riquadri di ancoraggio attorno ad essa. Per ciascun riquadro viene visualizzata l'eventualità che contenga un oggetto e quali siano le coordinate del riquadro. Nell'immagine è rappresentato come appare dalla posizione di una finestra scorrevole:



Il punteggio 2k rappresenta la probabilità data da Softmax per ciascun riquadro k per la presenza di un oggetto. Da notare è che, nonostante la RPN elabori le coordinate dei riquadri di delimitazione, essa non classifica comunque i possibili oggetti: ha il solo scopo di individuare regioni in cui siano presenti oggetti e quindi comunicare le coordinate dei relativi riquadri. *Se un riquadro di ancoraggio ha un punteggio, relativo alla presenza di un oggetto, superiore a una determinata soglia, allora quel dato riquadro verrà selezionato come possibile regione*.

Avendo ora le nostre possibili regioni, le si inseriscono direttamente nella Fast R-CNN. Si aggiunge: uno strato di Pooling, alcuni strati completamente connessi, uno strato di classificazione Softmax

ed infine un regressore dei riquadri di delimitazione (bounding box regressor). Così facendo si costituisce una rete che in un certo senso non è altro che la somma della Fast-RCNN con l'algoritmo RPN. Il modello può essere rappresentato nel seguente modo:



La *Faster R-CNN raggiunge così miglior velocità e accuratezza*. Nonostante siano stati molteplici i tentativi successivi di aumentare la velocità di riconoscimento degli oggetti, solo pochi modelli sono stati davvero in grado di superare questa rete. In altre parole, la Faster R-CNN non rappresenta certo il metodo più semplice e veloce per la object detection (tecnologia che si occupa del rilevamento di oggetti nelle immagini) ma presenta tutt'ora una delle migliori performance. Quindi, la Faster R-CNN in Tensorflow (libreria software open source per l'apprendimento automatico) con la Inception ResNet è il modello più lento ma anche il più accurato.

La rete neurale convoluzionale SSD

L'ultimo modello che si andrà ad analizzare è la **Single-Shot Detector** (**SDD**) che è negli ultimi tempi è diventata di noto interesse nel campo dell'ingegneria dell'informazione.

I primi due algoritmi delle reti precedenti eseguivano il *Region Proposal* e *Region Classification* in due fasi distinte. Innanzitutto, utilizzano una RPN (region proposal network) per generare le regioni di interesse, successivamente classificano queste regioni attraverso i propri strati interamente connessi o gli strati convoluzionali sensibili alla posizione degli oggetti da classificare. *La SSD unisce questi due processi in un singolo passaggio, prevedendo simultaneamente i riquadri di delimitazione e le classi, nel momento in cui esamina l'immagine*.

Data un'immagine di input ed un insieme di etichette al dataset, la SSD agisce in questo modo:

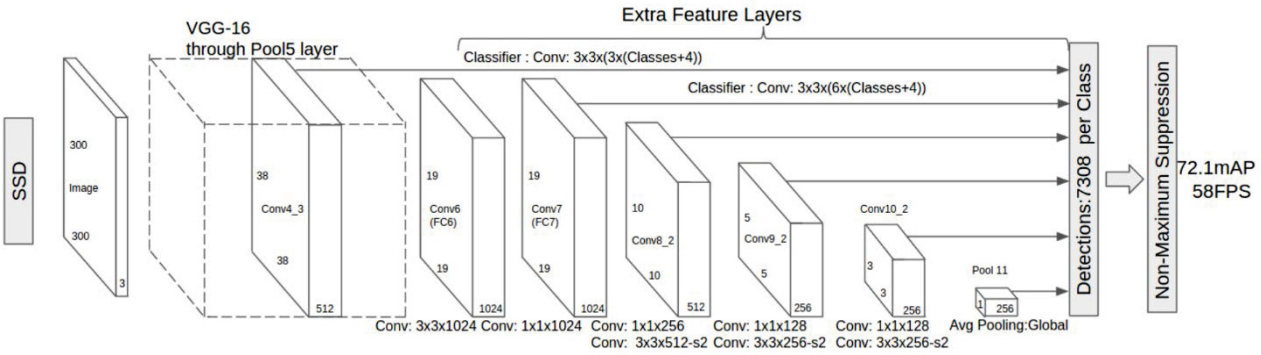
1) *Passa l'immagine attraverso una serie di strati convoluzionali*, producendo mappe delle caratteristiche a scale differenti (per esempio 10×10, poi 6×6 e ancora 3×3 ecc.)
2) Per ogni posizione in ciascuna di queste mappe delle caratteristiche*, usa un filtro convoluzionale 3×3 per valutare un piccolo set di riquadri di delimitazione predefiniti*. Questi ultimi sono l'equivalente dei riquadri di ancoraggio (anchor boxes) della Faster R-CNN.
3) *Per ogni riquadro prevede simultaneamente*:
    - ✓ Il riquadro del bounding box
    - ✓ La probabilità per ogni classe.
4) *Durante l'allenamento fa corrispondere il riquadro effettivo con quello predetto sulla base del metodo* ***Intersection over Union*** *(**IoU**),* ovvero l'Indice di Jaccard. *I riquadri meglio predetti verranno etichettati come "positivi", come anche tutti gli altri riquadri che abbiano un valore IoU >0.5*.

L'allenamento della SSD ha però una difficoltà peculiare rispetto ai precedenti modelli. Nei primi due modelli, la RPN permetteva di prendere in considerazione solo ciò che aveva anche solo la minima possibilità di essere un oggetto. Con la SSD, invece, questo primo filtro viene eliminato: esso classifica e traccia riquadri di delimitazione in ogni singola posizione nell'immagine, usando riquadri di differenti forme e di molteplici scale. Ciò ha come risultato l'avere un numero molto più elevato di riquadri dove, però, la maggior parte di questi è negativo (dal momento che non rileva oggetti).

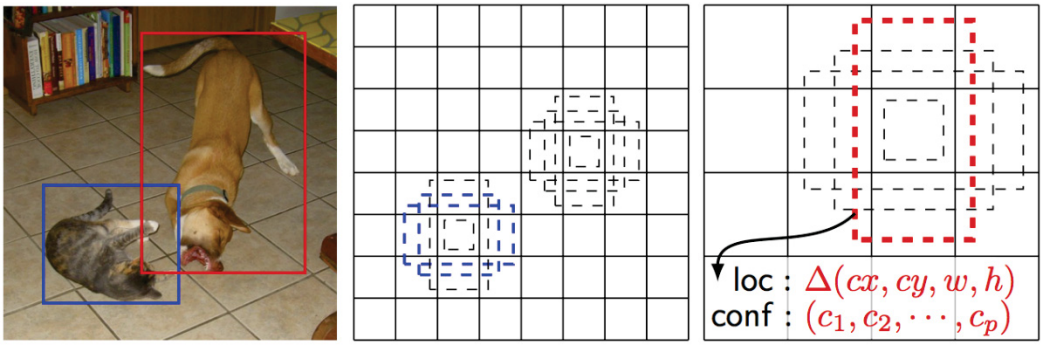
La SSD corregge questo problema in due modi. Per prima cosa utilizza la ***Non-Maximum Suppression*** (**NMS**) per raggruppare diversi riquadri sovrapposti in un unico riquadro. Quindi se ad esempio quattro riquadri di simile forma e grandezza contengono lo stesso oggetto, la NMS *permette di tenere il riquadro migliore, il più accurato, e di scartare i restanti*. In secondo luogo, utilizza una tecnica chiamata ***Hard Negative Mining*** per bilanciare le classi durante l'allenamento. Con questa tecnica, solo un sottogruppo degli esempi negativi con la più elevata loss in fase di

training (i cosiddetti falsi positivi) è utilizzato per ogni iterazione in fase di allenamento. La SSD mantiene un rapporto 3:1 tra negativi e positivi.

La sua architettura appare così:



Come si è detto nelle righe precedenti, in questa rete ci sono degli "*extra feature layers*". Queste mappe delle caratteristiche dalle misure variabili aiutano a individuare oggetti di diverse dimensioni. Eccone un esempio:



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

In mappe delle caratteristiche di dimensioni più piccole (4×4), ogni riquadro ricopre una regione più ampia dell'immagine, permettendo così di individuare oggetti più grandi. La region proposal e la classificazione sono perciò eseguite nello stesso momento: avendo p classi di oggetti, ogni riquadro di delimitazione è associato a un vettore dimensionale (4+p) che produce *4 riquadri di coordinate e p probabilità di classe*. In ultima istanza, viene utilizzata anche qui una funzione Softmax per classificare gli oggetti.

Si può perciò dire che la SSD non è poi così diversa rispetto ai due modelli precedenti. Tuttavia omette il passaggio di individuazione delle regioni, considerando ogni riquadro in ogni posizione dell'immagine insieme alla sua classificazione. Eseguendo tutti questi processi simultaneamente, la SSD è di certo il più veloce tra i tre modelli presi in considerazione.

Implementazione

Per implementare effettivamente delle reti neurali convoluzionali si è utilizzato l'ambiente di sviluppo Google Colab, una piattaforma che ci permette di eseguire il codice direttamente sul Cloud. Per sfruttare le funzionalità di tale ambiente, tutto ciò di cui si ha bisogno è un account Google. Per eseguire il codice, Colab utilizza i cosiddetti Jupyter Notebook. Questi non sono altro che documenti interattivi nei quali si può scrivere (e quindi eseguire) il proprio codice. Più precisamente, tali notebook permettono di suddividere il nostro codice in celle, ognuna delle quali può contenere anche del testo informativo. Tramite un unico documento, è infatti possibile: sia eseguire tutti gli step di un processo di analisi o processing, sia descriverne il comportamento in linguaggio naturale.

Google Colab permette di configurare il notebook da utilizzare permettendo di decidere:

- la versione di Python che si intende utilizzare,
- se abilitare il supporto all'uso della GPU o sfruttare le TensorFlow Power Unit (TPU).

Il vantaggio nell'utilizzo di questa piattaforma è la semplice condivisione con altre persone del codice di sviluppo, mentre lo svantaggio risiede nel fatto di dover lavorare di fatto con una macchina che nel momento in cui si esegue il logout, vengono persi tutti i dati. Così facendo ogni volta che si riavvia la macchina occorre dover reinstallare la versione di Tensorflow, ricompilare le varie librerie e fare una nuova configurazione dell'ambiente.

Nelle prossime pagine verranno fornite due guide: la prima permette dopo aver preparato il dataset, di importare i dati semplicemente in un notebook Colab, già preconfezionato e cambiando alcuni parametri, permette di avere una rete pronta per essere utilizzata.

La seconda invece riporta tutti i vari passaggi che devono essere eseguiti per configurare una rete neurale convoluzionale SSD.

Implementazione attraverso un ambiente Colab pre-impostato

Al fine di implementare una rete neurale attraverso Google Colab i passaggi necessari sono:

1. Installazione
2. Raccolta di dati
3. Dati di etichettatura
4. Generazione di TFRecords per l'addestramento
5. Configurazione del training
6. Modello dell'allenamento
7. Esportazione del grafico di inferenza
8. Test del rivelatore di oggetti

Installazione

Si controlli di avere una versione di Python 3.6 o versioni successive.
Si deve installare TensorFlow 1.x, la versione 1.x è molto importante se si vuole effettuare una verifica attraverso "test_builder.py" per verificare il corretto funzionamento di TensorFlow; quindi per l'installazione di tutte le librerie necessarie alla rete occorre inserire il seguente script:

```
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
! pip install --user Cython
! pip install --user contextlib2
! pip install --user pillow
! pip install --user lxml
! pip install --user jupyter
! pip install --user matplotlib
! pip install --user tf_slim
```

Impostazione dell'ambiente

Occorre clonare il repository dei modelli TensorFlow:

```
git clone https://github.com/tensorflow/models.git
```

Da questo punto in poi, nella tendina a destra verrà creata una directory denominata models.

Dopo aver clonato il repository dei modelli tf, occorre configurare cocodataset attraverso i seguenti comandi:

```
! git clone https://github.com/cocodataset/cocoapi.git
%cd cocoapi/PythonAPI
! make
! cp -r pycocotools /content/models/research/
```

Compilazione di Protobuf

L'API Tensorflow Object Detection utilizza Protobufs per configurare il modello e i parametri di addestramento. Prima di poter utilizzare il framework, è necessario compilare le librerie Protobuf. Questo dovrebbe essere fatto eseguendo il seguente comando dalla directory tensorflow /models/research/:

```
%cd /content/models/research
! protoc object_detection/protos/*.proto --python_out=.
```

Aggiungi librerie a PYTHONPATH

Quando si esegue localmente, le directory tensorflow/models/research/ e slim dovrebbero essere aggiunte a PYTHONPATH. Questo può essere fatto eseguendo quanto segue da tensorflow/models /research/:

```
%cd /content/models/research
import os
os.environ['PYTHONPATH'] = ':/content/models/research:/content/models/res
earch/slim/:'+os.environ['PYTHONPATH']
print(os.environ['PYTHONPATH'])
```

Installazione di librerie per il rilevamento oggetti

Per l'installazione delle librerie bisogna costruire setup.py e poi installarlo attraverso il seguente comando:

```
%cd /content/models/research/slim
! sudo python setup.py build
! sudo python setup.py install
```

Test dell'installazione

Per verificare che Tensorflow sia stato installato correttamente è possibile eseguire i seguenti passaggi:

```
from tensorflow.python.compiler import tensorrt as trt
%cd /content/models/research
! python3 object_detection/builders/model_builder_tf1_test.py
```

Se il risultato è simile al seguente, TensorFlow è stato installato nel modo corretto.

```
----------------------------------------------------------------------
Ran 19 tests in 0.133s

OK (skipped=1)
```

Raccolta di dati

Si può fare attraverso il proprio browser Google Chrome istallando l'estensione Download All Images; a questo punto è sufficiente cercare le immagini desiderate ad esempio "Apple" e poi fare click sul pulsante "Download All Images" che si trova nella parte in alto a destra del browser, così facendo si otterrà un file .zip contenente le immagini ricercate. Un altro modo, potrebbe essere di andare sul campo e fare per proprio conto un numero considerevole di immagini di quello che si vuole far riconoscere con l'object_detection e poi di crearsi un proprio archivio personale. (Un'altra modalità potrebbe essere quella di andare sul sito della FLIR e scaricare un set di immagini già predisposto come si vedrà alla fine della spiegazione delle due guide).

In questa guida si farà l'esempio di implementazione di una rete che vada a riconoscere i diversi tipi di frutta all'interno di un'immagine. Dopo aver trovato un certo numero di immagini (tante più immagini si hanno a disposizione tanto più il modello si allenerà e sarà preciso nel rilevamento) si crea una cartella ad esempio "Apple" dopodiché dall'ambiente di sviluppo di Google Colab ci si sposti nel proprio dispositivo, se è Linux si possono rispettare i seguenti accorgimenti:
Si accede al terminale e si installa *LabelImg*:

```
pip3 install labelImg
```

LabelImg è uno strumento di annotazione grafico delle immagini.
Dopo aver installato labelImg lo si apre digitando:

```
labelImg
```

A questo punto per ogni immagine si deve costruire una label sopra l'oggetto da ricercare ad esempio un rettangolo che evidenzia l'intero oggetto (tanto più precisa sarà tanto più preciso sarà il risultato della rete). Quello che si sta facendo è generare un file XML contenente l'oggetto coordinato con la sua etichetta. Un buon archivio, ad esempio, sarebbe di almeno 100 immagini.
In seguito si importa il repository in Google Colab attraverso il seguente script

```
from google.colab import drive
drive.mount('/gdrive')
```

Una struttura tipica delle cartelle dovrebbe essere così:

```
├── ...
    ├── data
    ├── images
    │   ├── train
    │   └── test
    ├── utils
    │   ├── label_map_util.py
    │   └── visualization_utils.py
    ├── generate_tfrecord.py
    └── object-detection.pbtxt
    └── transform_image_resolution.py
    └── xml_csv.py
    └── webcam_inference.py
```

Generazione di TFRecords per l'addestramento

Ora si copi il 70% delle immagini totali in images/train (queste serviranno per fare l'allenamento) e il restante 30% nella cartella images/test.
Con le immagini etichettate, si deve creare i TFRecords che possono essere utilizzati come dati di input per l'addestramento del rivelatore di oggetti.

```
├── ...
├── data
├── images
    ├── train
                ├── image1.jpg
                ├── image1.xml ..
    ├── test
                ├── image5.jpg
                ├── image5.xml ..
├── generate_tfrecord.py
└── object-detection.pbtxt
└── transform_image_resolution.py
└── xml_csv.py
└── webcam_inference.py
```

Ora si possono trasformare i file XML in train_label.csv e test_label.csv digitando:

```
!python xml_to_csv.py
```

Tale stringa permetterà la creazione di due file nella cartella dei dati. Uno chiamato *test_labels.csv* e un altro *train_labels.csv*.
Prima di poter trasformare i file appena creati in TFRecords, è necessario modificare alcune righe nel file *generate_tfrecords.py*.

Se si ha una sola classe:

```
# TO-DO replace this with label map , replace with your own classes
def class_text_to_int(row_label):
    if row_label == 'apple':
        return 1
    else:
        return 0
```

Se si hanno più classi allora:

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'apple':
        return 1
    elif row_label == 'banana':
        return 2
    elif row_label == 'orange':
        return 3
    else:
        return None
```

Ora si può generare TFRecords digitando:

```
!python3 generate_tfrecord.py --csv_input=data/train_labels.csv  --output_path=train.record --image_dir=images/train
!python3 generate_tfrecord.py --csv_input=data/test_labels.csv  --output_path=test.record --image_dir=images/test
```

Questi due comandi generano un *train.record* e un file *test.record* che possono essere utilizzati per addestrare il nostro rilevatore di oggetti.

Configurazione del training

L'ultima cosa che si deve fare prima dell'allenamento è creare una mappa delle etichette e un file di configurazione del training.

La mappa delle etichette associa un ID a un nome. All'interno del file *object-detect.pbtxt* occorre effettuare le seguenti modifiche:

Se si ha una sola classe:

```
item{
  id:1
  name:"apple"

}
```

Se si utilizzano più classi, seguire questo modello:

```
item {
    id: 1
    name: 'apple'
}
item {
    id: 2
    name: 'banana'
}
item {
    id: 3
    name: 'orange'
}
item {
    id: 4
    name: 'etc'
}
```

Il numero ID di ciascun elemento deve corrispondere all'ID specificato per il file generate_tfrecord.py.

Infine, come si è già detto, occorre creare una configurazione di training per far ciò si utilizzi il modello Google Colab già pre-impostato. Per farlo basta andare al link sottostante:

https://colab.research.google.com/drive/1o7JB0pWanEMn6qnRnEXphu0T4YbuKLL2

Tale modello utilizzerà la rete "SSD_MOBILENET_V2" per la formazione e con la dimensione dell'archivio V2 (ovvero un dataset molto ampio messo a disposizione da Google). Nell'applicativo sarà possibile modificare il numero di passaggi, il modello pre-addestrato da utilizzare, l'archivio delle immagini e la dimensione. Eseguendo le celle sottostanti si inizializzerà la rete. In fondo ci sarà una voce "Upload Record TF" dove sarà necessario caricare il file train.record , test.record e object-detect.pbtxt generati nei passaggi precedenti.

Modello di allenamento

Ora, dopo aver caricato tutti i file precedenti, eseguendo le celle sottostanti nel notebook Colab il modello procederà nella fase di allenamento.

Esportazione del grafico di inferenza

Quando si saranno eseguite tutte le celle, verrà infine scaricato un file denominato frozen_inference_graph.pb e quindi sarà possibile visualizzare i risultati ottenuti.

Implementazione attraverso l'importazione di un modello di rete

Installazione

Si controlli di avere una versione di Python 3.6 o versioni successive.
Si deve installare TensorFlow 1.x, la versione 1.x è molto importante se si vuole effettuare una verifica attraverso "test_builder.py" per verificare il corretto funzionamento di TensorFlow; quindi per l'installazione di tutte le librerie necessarie alla rete occorre inserire il seguente script:

```
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)
! pip install --user Cython
! pip install --user contextlib2
! pip install --user pillow
! pip install --user lxml
! pip install --user jupyter
! pip install --user matplotlib
! pip install --user tf_slim
```

Impostazione dell'ambiente

Occorre clonare il repository dei modelli TensorFlow:

```
git clone https://github.com/tensorflow/models.git
```

Da questo punto in poi, nella tendina a destra verrà creata una directory denominata models.
Dopo aver clonato il repository dei modelli tf, occorre configurare Cocodataset attraverso i seguenti comandi:

```
! git clone https://github.com/cocodataset/cocoapi.git
%cd cocoapi/PythonAPI
! make
! cp -r pycocotools /content/models/research/
```

Compilazione di Protobuf

L'API Tensorflow Object Detection utilizza Protobufs per configurare il modello e i parametri di addestramento. Prima di poter utilizzare il framework, è necessario compilare le librerie Protobuf. Questo dovrebbe essere fatto eseguendo il seguente comando dalla directory tensorflow /models/research/:

```
%cd /content/models/research
! protoc object_detection/protos/*.proto --python_out=.
```

Aggiungi librerie a PYTHONPATH

Quando si esegue localmente, le directory tensorflow/models/research/ e slim dovrebbero essere aggiunte a PYTHONPATH. Questo può essere fatto eseguendo quanto segue da tensorflow/models /research/:

```
%cd /content/models/research
import os
os.environ['PYTHONPATH'] = ':/content/models/research:/content/models/res
earch/slim/:'+os.environ['PYTHONPATH']
print(os.environ['PYTHONPATH'])
```

Installazione di librerie per il rilevamento oggetti

Per l'installazione delle librerie bisogna costruire setup.py e poi installarlo attraverso il seguente comando:

```
%cd /content/models/research/slim
! sudo python setup.py build
! sudo python setup.py install
```

Test dell'installazione

Per verificare che Tensorflow sia stato installato correttamente è possibile eseguire i seguenti passaggi:

```
from tensorflow.python.compiler import tensorrt as trt
%cd /content/models/research
! python3 object_detection/builders/model_builder_tf1_test.py
```

Se il risultato è simile al seguente, TensorFlow è stato installato nel modo corretto.

```
----------------------------------------------------------------------
Ran 19 tests in 0.133s

OK (skipped=1)
```

Raccolta di dati

La raccolta dei dati può essere fatta attraverso il proprio browser Google Chrome istallando l'estensione Download All Images; una volta installata è sufficiente cercare le immagini desiderate, ad esempio "persone" e poi fare click sul pulsante "Download All Images" che si trova nella parte in alto a destra del browser, così facendo si otterrà un file .zip contenente le immagini ricercate. Un altro modo comunque potrebbe essere di andare sul campo e costruire per proprio conto un numero considerevole di immagini di quello che si vuole far riconoscere con l'object_detection inserendo tutto il contenuto in un singolo archivio. (Un'altra modalità potrebbe essere quella di andare sul sito della FLIR e scaricare un set di immagini già predisposto come si vedrà alla fine della spiegazione delle due guide).

Struttura delle cartelle consigliata

Per rendere più semplice seguire questa esercitazione, si crei la seguente struttura di cartelle all'interno della cartella /models appena clonata:

```
models
    ├── annotations
    |   └── xmls
    ├── images
    ├── checkpoints
    ├── tf_record
    ├── research
    ...
```

Queste cartelle verranno utilizzate per archiviare i componenti richiesti per il modello mentre si procede.

Dopo il download, si salvi tutte le immagini in models/images/. Per semplificare i processi successivi, si possono rinominare le immagini come numeri (ad esempio "1.jpg.", "2.jpg",) Eseguendo il seguente script:

```
import os


path = 'models/images/'
counter = 1
for f in os.listdir(path):
    suffix = f.split('.')[-1]
    if suffix == 'jpg' or suffix == 'png':
        new = '{}.{}'.format(str(counter), suffix)
        os.rename(path + f, path + new)
        counter = int(counter) + 1
```

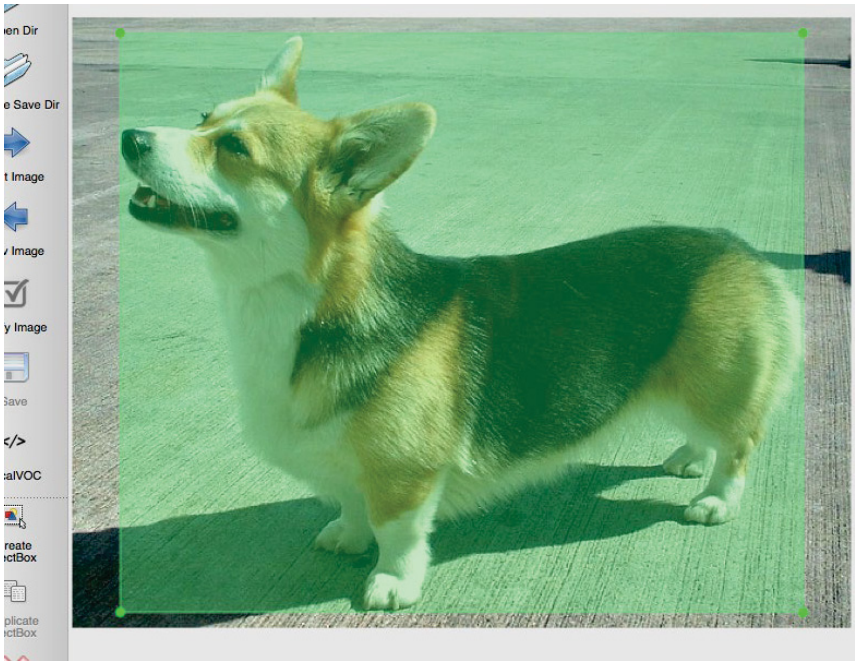
Etichettatura del set di dati

Dopo aver raccolto tutte le immagini necessarie, le si devono etichettare manualmente. Esistono molti pacchetti che servono a questo scopo. labelImg è una scelta popolare.

labelImg fornisce una GUI intuitiva. Inoltre, salva i file delle etichette (.xml) nel popolare formato Pascal VOC.

Volendo mostrare come eseguire passo-passo le procedure a titolo di esempio, in questa guida si cercherà di etichettare e individuare una particolare specie canina, ovvero il “Corgi”.

Ecco come appare un'immagine etichettata in labelImg:



Infine si deve controllare che ogni immagine abbia un file corrispondente .xml e le si salvi nella cartella models/annotations/xmls/.

Creazione della mappa etichette (.pbtxt)

Le classi devono essere elencate nella mappa delle etichette. Poiché si stanno rilevando solo Corgis, la mappa delle etichette dovrebbe contenere solo un elemento come il seguente:

```
item {
    id: 1
    name: 'corgi'
}
```

Si noti che si deve iniziare da 1, in quanto il valore “0” è un ID riservato.

Si salvi questo file come label_map.pbtxt in models/annotations/

Creazione di trainval.txt

Il file trainval.txt è un elenco di nomi di immagini senza estensioni di file. Poiché si è scelto di avere numeri sequenziali per i nomi delle immagini, l'elenco dovrebbe apparire così:

```
1
2
3
...
198
199
200
```

Si salvi questo file come trainval.txt in models/annotations/

Conversione di XML in file CSV (.csv)

È possibile utilizzare il link sottostante per convertire i file XML in CSV:

https://gist.github.com/iKhushPatel/ed1f837656b155d9b94d45b42e00f5e4

In questo modo tutte le immagini che hanno un file XML separato dai relativi controlli di rilevazione del singolo rettangolino verranno unificate in un unico file CSV contenente tutti i file XML e le relative caselle di controllo coordinate che verrà immesso per la creazione di TFrecords.

Creazione dei TFRecord (.record)

TFRecord è un formato di dati importante progettato per Tensorflow.

Prima di poter addestrare il rilevatore di oggetti personalizzati, è necessario convertire i dati nel formato TFRecord.

Poiché è necessario addestrare e convalidare il modello, il set di dati verrà suddiviso in training (train.record) e set di validazione (val.record). Lo scopo è quello di fornire due set: quello di addestramento che è costituito da un insieme di esempi da cui il modello apprende e quello di validazione che è un gruppo di esempi utilizzati durante la formazione per valutare iterativamente l'accuratezza del modello.

Si utilizzi create_tf_record.pyper per convertire il nostro set di dati in train.recorde val.record.

Si può scaricare dal link sottostante e salvarlo in models/research/object_detection/dataset_tools/.

https://gist.github.com/iKhushPatel/5614a36f26cf6459cc49c8248e8b5b48

A questo punto occorre modificare il nome dell'etichetta in `if row_label == 'Label1':` in base alle classificazioni.

Lo script è pre-configurato per eseguire una divisione che associ il 70% delle immagini nella cartella train e il restante 30% alla cartella di validazione. Eseguendo:

```
# From the models directory

$ python research/object_detection/dataset_tools/create_tf_record.py
```

Se lo script viene eseguito correttamente train.recorde val.record dovrebbe apparire nella cartella models/research/. Infine li si devono inserire nella cartella models/tf_record/.

Download del modello pre-addestrato

Ci sono molti modelli di rilevamento oggetti pre-addestrati disponibili nello "zoo dei modelli" nel link sottostante:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Li si devono addestrare utilizzando il nostro set di dati personalizzato, i modelli devono essere ripristinati in Tensorflow utilizzando i loro punti di controllo (file .ckpt), che sono record dei precedenti stati del modello.

Per questa guida, si può scaricare la rete ssd_mobilenet_v2_coco dal link sottostante e si salveranno i file di checkpoint del modello (model.ckpt.meta, model.ckpt.index, model.ckpt.data-00000-of-00001) nella cartella models/checkpoints/.

http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz

Modifica del file Config (.config)

Ognuno dei modelli predefiniti ha un file di configurazione che contiene dettagli sul modello. Per rilevare la classe personalizzata, il file di configurazione deve essere modificato di conseguenza.

I file di configurazione sono inclusi nella cartella models che si è clonata all'inizio. Solitamente si trovano all'interno di:

```
models/research/object_detection/samples/configs
```

Nel caso di esempio, verrà modificato il file di configurazione per ssd_mobilenet_v2_coco. Prima di fare ciò, tuttavia è bene crearne una copia e salvarla nella cartella models.

Gli elementi che devono essere modificati sono i seguenti:

a. Dal momento che si sta solo cercando di rilevare Corgis, si cambi num_classes a 1.
b. fine_tune_checkpoint indica quale file del modello di checkpoint utilizzare in questo caso dunque lo si imposti ad checkpoints/model.ckpt

c. Il modello deve anche sapere dove si trovano i file TFRecord e le mappe delle etichette sia per i set di addestramento che di validazione. Poiché train.record e val.record sono salvati nella cartella tf_record, la configurazione dovrebbe riflettere che:

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "tf_record/train.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
}


eval_input_reader: {
  tf_record_input_reader {
    input_path: "tf_record/val.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Train

A questo punto, la cartella models dovrebbe apparire così:

```
models
    ├── annotations
    |   ├── label_map.pbtxt
    |   ├── trainval.txt
    |   └── xmls
    |       ├── 1.xml
    |       ├── 2.xml
    |       ├── ...
    |
    ├── images
    |   ├── 1.jpg
    |   ├── 2.jpg
    |   ├── ...
    |
    ├── checkpoints
    |   ├── model.ckpt.data-00000-of-00001
    |   ├── model.ckpt.index
    |   └── model.ckpt.meta
    |
    ├── tf_record
    |   ├── train.record
    |   └── val.record
    |
    ├── research
    |   ├── ...
    ...
```

Se si ha questa configurazione si può proseguire ad inserire i successivi comandi ed iniziare la fase di allenamento:

```
# Change into the models directory
$ cd tensorflow/models

# Make directory for storing training progress
$ mkdir train

# Make directory for storing validation results
$ mkdir eval

# Begin training
$ python research/object_detection/train.py \
    --logtostderr \
    --train_dir=train \
    --pipeline_config_path=ssd_mobilenet_v2_coco.config
```

Così inizierà la fase di addestramento ed il tempo varia in base alla potenza di calcolo della macchina.

Validazione

Lo script eval.py verifica l'avanzamento della cartella train e valuta il modello in base al checkpoint più recente.

```
# From the models directory

$ python research/object_detection/eval.py \
    --logtostderr \
    --pipeline_config_path=ssd_mobilenet_v2_coco.config \
    --checkpoint_dir=train \
    --eval_dir=eval
```

Si può monitorare l'avanzamento dei lavori di training e valutazione eseguendo:

```
# From the models directory

$ tensorboard --logdir=./
```

Di solito, è possibile interrompere il processo quando la funzione di perdita si sta assottigliando e non diminuisce più di una quantità significativa. Nel caso in esame si è fermati al passaggio 3258.

Esportazione del modello

Una volta terminato l'addestramento del modello, è possibile esportare il modello da utilizzare per l'inferenza. Se si è seguita la struttura delle cartelle precedentemente utilizzata, si può usare il seguente comando:

```
# From the models directory

$ mkdir fine_tuned_model

$ python research/object_detection/export_inference_graph.py \
--input_type image_tensor \
--pipeline_config_path ssd_mobilenet_v2_coco.config \
--trained_checkpoint_prefix  train/model.ckpt-
<the_highest_checkpoint_number> \
--output_directory fine_tuned_model
```

Classificazione delle immagini

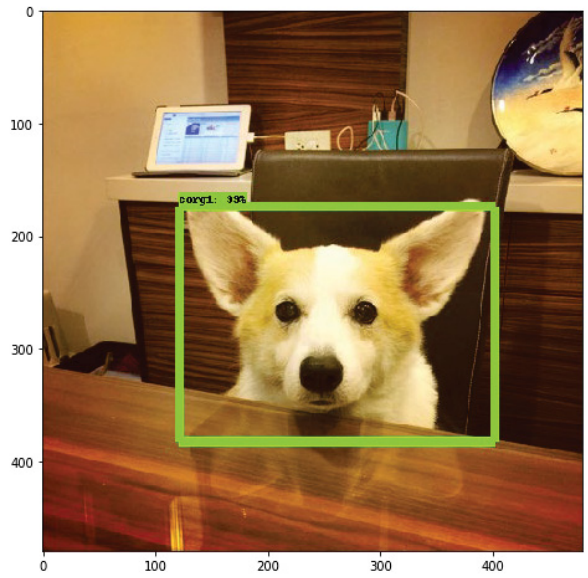
Ora che si ha un modello, lo si può utilizzare per rilevare i Corgi nelle foto. Ai fini della dimostrazione, si potrà vedere la mappatura di un Corgi in un'immagine. Prima di procedere, selezionare un'immagine con cui si desidera testare il modello.

La cartella models è stata creata con un file notebook (.ipynb) che si può utilizzare per ottenere deduzioni con alcune modifiche. Si trova in models/research/object_detection/object_detection_tutorial.ipynb.

Seguire i passaggi seguenti per modificare il notebook:

1. MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
2. PATH_TO_CKPT = 'path/to/your/frozen_inference_graph.pb'
3. PATH_TO_LABELS = 'models/annotations/label_map.pbtxt'
4. NUM_CLASSES = 1
5. Si commenti completamente la cella n. 5 (appena sotto Download Model)
6. Dato che si sta testando solo un'immagine, si può commentare PATH_TO_TEST_IMAGES_DIR e TEST_IMAGE_PATHS nella cella n.9 (appena sotto Detection)
7. Nella cella n. 11 (l'ultima cella), si cancelli il ciclo for, deselezionando il suo contenuto e aggiungendo il percorso all'immagine di prova:
   imagepath = 'path/to/image_you_want_to_test.jpg

Dopo aver seguito i passaggi, si esegua il notebook e si dovrebbe vedere il Corgi nell'immagine di prova evidenziato da un rettangolo di selezione:



Corgi è stato dunque trovato dal rilevatore di oggetti personalizzati.

Implementazione su immagini termiche

Lo scopo della tesi era quello di rilevare persone in immagini ad infrarosso perciò se si desidera creare una rete neurale che vada a rilevare i “pedoni” quello che si deve fare è scaricare il dataset al seguente link:

https://www.flir.com/oem/adas/adas-dataset-form/

Il dataset è costituito nel seguente modo:

Immagini	Più di 14.000 immagini totali con più di 10.000 ottenute da brevi segmenti video e campioni di immagini casuali, oltre a 4K di immagini BONUS da un video di 140 secondi
Frequenza di aggiornamento acquisizione immagine	Registrato a 30Hz. Sequenze di set di dati campionati a 2 frame / sec o 1 frame / secondo. Le annotazioni video sono state eseguite con una registrazione di 30 frame / sec.
Annotazioni totali dell’etichette label	10.228 fotogrammi totali e 9.214 fotogrammi con riquadri di selezione. 1. Persona (28.151) 2. Auto (46.692) 3. Bicicletta (4.457) 4. Cane (240) 5. Altro veicolo (2.228)

Annotazioni totali dell'etichette video	4.224 fotogrammi totali e 4.183 fotogrammi con riquadri di selezione. 1. Persona (21.965) 2. Auto (14.013) 3. Bicicletta (1.205) 4. Cane (0) 5. Altro veicolo (540)
Condizioni di guida	Giorno (60%) e notte (40%) guidando per le strade e le autostrade di Santa Barbara, CA da novembre a maggio, con tempo da sereno a nuvoloso.
Specifiche della fotocamera	IR Tau2 640x512, 13mm f / 1.0 (HFOV 45 °, VFOV 37 °) FLIR BlackFly (BFS-U3-51S5C-C) 1280x1024, Computar 4-8mm f / 1.4-16 megapixel obiettivo (FOV impostato per abbinare Tau2)
Formato file del set di dati	1. Termico - TIFF a 14 bit (senza AGC) 2. JPEG termico a 8 bit (AGC applicato) w/o bounding boxes incorporate nelle immagini 3. JPEG termico a 8 bit (AGC applicata) con bounding boxes incorporate nelle immagini per la visualizzazione degli obiettivi. 4. RGB - JPEG a 8 bit 5. Annotazioni: JSON (formato MSCOCO)
Risultati del campione	i punteggi map sono stati ottenuti per le categorie: Persone (0,794), Bicicletta (0,580) e Auto (0,856). http://cocodataset.org/#detection-eval è stato utilizzato per i criteri di valutazione dell'accuratezza.
Impostazioni di formazione e sviluppo FLIR ADK	Utilizzare FLIR ADK con le impostazioni predefinite per iniziare la raccolta dei dati

In altre parole il dataset contiene immagini termiche in scala di grigi e RGB di scenari contenenti automobili, strade, persone, biciclette, ecc.

Una volta scaricato il seguente set di immagini si può scegliere una delle guide precedentemente fornite e si otterrà la rete desiderata.

Applicazioni

Al fine di fare alcuni esempi pratici sul funzionamento della rete neurale convoluzionale verranno presentati due notebook Colab.

Il primo, (anche se, come si ripete a titolo di esempio, per far comprendere l'importanza di un buon set di dati) è capace di riconoscere cani e gatti senza però dare riscontro (attraverso il riquadro di delimitazione) dell'identificazione del cane o del gatto rilevato, ma soltanto dando il nome dell'immagine e restituendo la classe di appartenenza. L'aspetto più importante dell'applicativo è quello di mettere in evidenza attraverso vari grafici come può essere migliorata l'efficienza di rilevazione.

Il secondo applicativo, invece, è un chiaro esempio di rilevamento degli oggetti nelle immagini a colori o ad infrarosso. La peculiarità del notebook consiste nel far scegliere all'utente l'utilizzo di una tra le due reti precedentemente illustrate quali Faster R-CNN e SSD.

Nelle varie discussioni si è inserito il link del notebook Colab così da poterlo testare personalmente.

Applicativo per la classificazione delle immagini

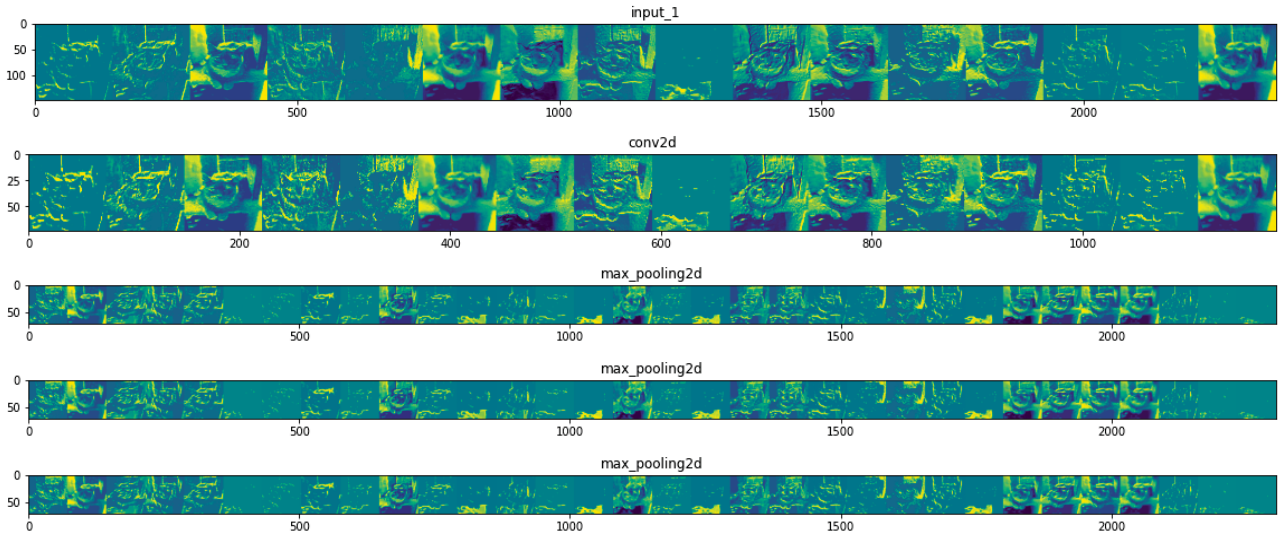
È un notebook Colab che permette di rilevare se nell'immagine è presente la figura di un cane o di un gatto, a differenza delle guide precedenti, nel notebook si utilizzerà la libreria Keras.
Keras è una libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, e supporta come back-end le librerie TensorFlow.
L'applicativo è raggiungibile attraverso il seguente link:

https://colab.research.google.com/drive/1_yNTCPLuhDL7_XaSyCfx4H-9ytEgMna-#scrollTo=hfbAs_uviSE4

In breve il programma è stato progettato nel seguente modo:

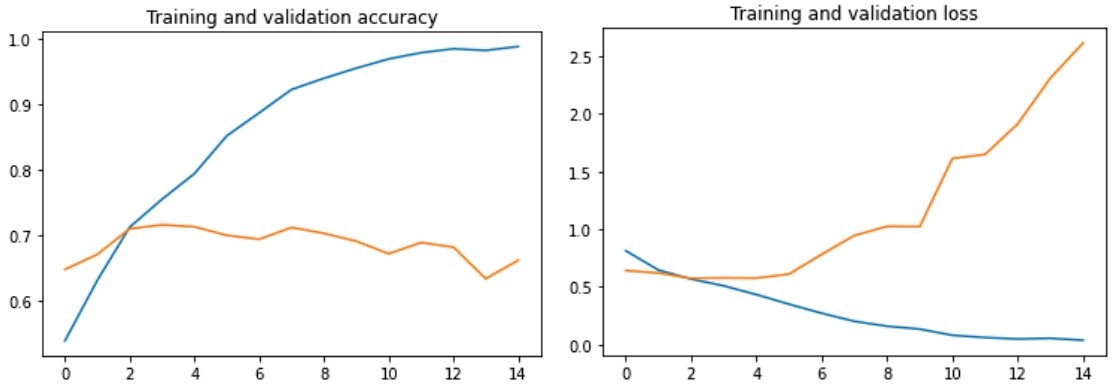
a. Vengono importati i file, installate le librerie necessarie e definite le variabili d'ambiente.
b. In seguito si costruisce da zero un piccolo Convnet da Scratch per ottenere una precisione del 72%. Le immagini che vengono inserite nella Convnet sono immagini a colori 150x150. Per quanto riguarda l'architettura si impilano 3 moduli (convoluzione + ReLu + max pooling). Le convoluzioni operano su finestre 3x3 e i livelli operano su finestre 2x2. La prima convoluzione estrae 16 filtri, la seguente ne estrae 32 filtri e l'ultima ne estrae 64.
c. Prima di lanciare l'allenamento è necessario configurare le immagini che verranno lette nella cartella d'origine, esse vengono convertite in tensorflow_32 ed inserite (con le loro etichette) nella rete. Si avrà un generatore per le immagini di addestramento e uno per quelle di validazione. Questi generatori produrranno gruppi di 20 immagini di dimensioni 150x150 e le loro etichette in formato binario.
d. I dati che entrano nelle reti neurali dovrebbero essere normalizzati in qualche modo per renderli più suscettibili all'elaborazione da parte della rete. (Non è raro utilizzare "pixel grezzi" in una convnet.). In questo caso, si pre-elaborano le immagini normalizzando i valori dei pixel in modo che siano [0, 1] nell'intervallo (originariamente tutti i valori sono [0, 255]). In Keras questo può essere fatto tramite la keras.preprocessing.image.ImageDataGeneratorclass usando il parametro rescale. Questa ImageDataGeneratorclass consente di creare istanze di generatori di batch di immagini aumentate (e relative etichette) tramite .flow(data, labels) o .flow_from_directory. Questi generatori possono essere utilizzati con i metodi del modello KERAS che accettano generatori di dati come ingressi: fit_generator, evaluate_generatore predict_generator.
e. A questo punto può essere lanciato il training che verrà eseguito su tutte le 2000 immagini disponibili, per 15 epoche, e verranno convalidate su tutte le 1.000 immagini di validazione.
f. Al termine dell'allenamento per avere un'idea del tipo di funzionalità apprese dalla convnet, una cosa da fare è visualizzare come un input viene trasformato mentre passa attraverso la convnet. Per farlo si può selezionare un'immagine di cane o gatto casuale dal set di addestramento e generare una figura in cui ogni riga è l'output di un livello e ogni immagine nella riga è un filtro specifico nella mappa delle caratteristiche di output. Questo processo verrà rieseguito per generare rappresentazioni intermedie per una varietà di

immagini di allenamento. Il risultato è:



Come si può notare, si passa dai pixel grezzi delle immagini a rappresentazioni sempre più astratte e compatte. Le rappresentazioni a valle iniziano a mettere in evidenza ciò a cui la rete presta attenzione e mostrano sempre meno le funzioni che vengono "attivate". Queste rappresentazioni presentano progressivamente meno informazioni sui pixel originali dell'immagine ma contemporaneamente forniscono dati più dettagliati sulla classe.

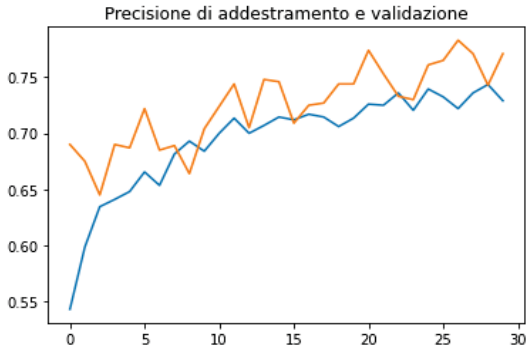
g. Se si traccia l'accuratezza del modello si ottiene che:



Il grafico di sinistra mette in evidenza che la precisione di allenamento (in blu) si avvicina al 100%, mentre quella di validazione (in verde) si blocca al 70%. Per quanto riguarda il grafico di destra si nota che la validazione raggiunge il minimo dopo solo tre epoche. In tutti i passaggi successivi si cercherà di aumentare il livello di precisione fino a raggiungere il 90%.

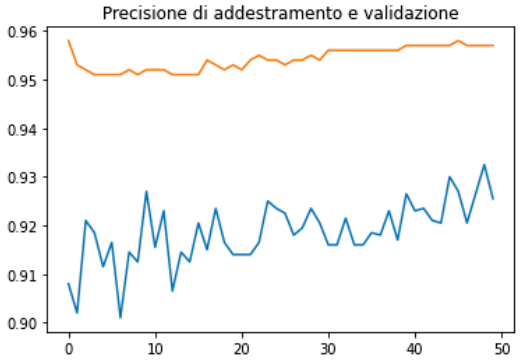
h. La prima modifica che si può effettuare consiste nell'aumentare il numero di esempi di allenamento attraverso una serie di trasformazioni casuali, in modo che al momento dell'allenamento, il modello non vedrà mai esattamente la stessa immagine due volte. Questo aiuta a prevenire un eccesso di adattamento e aiuta il modello a generalizzare meglio. In seguito, si è utilizzata un'altra strategia popolare per combattere il sovra-utilizzo che è quella di utilizzare il Dropout. Con l'aumento dei dati e il dropout in atto, si può riconfigurare il nostro modello di convnet. Questa volta, la rete si è allenata su tutte le

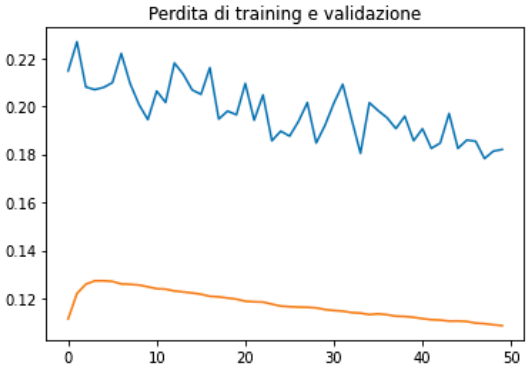
2000 immagini disponibili per 30 epoche, e convalidata su tutte le 1.000 immagini di validazione così da ottenere grafico seguente:



I grafici mettono in evidenza il fatto di non essere in condizione di overfitting e che la rete ha guadagnato cinque punti percentuali in termini di precisione della validazione.

i. Per aumentare ancora di più la precisione una cosa che viene comunemente eseguita in computer vision è prendere un modello addestrato su un set di dati molto grande, eseguirlo su un set di dati più piccolo ed estrarre le rappresentazioni intermedie (caratteristiche) generate dal modello. Queste rappresentazioni sono spesso informative per il proprio compito di visione artificiale, anche se l'attività può essere molto diversa dal problema su cui è stato formato il modello originale. In questo caso, si è utilizzato il modello Inception V3 sviluppato da Google e pre-addestrato su ImageNet: quest'ultimo consiste in un ampio set di dati di immagini web (1.4M immagini e 1000 classi). Per adattare questo modello nella ricerca (in questo caso) di cani e gatti si deve scegliere quale livello intermedio di Inception V3 si vuole usare per l'estrazione delle funzionalità. Una pratica comune è quella di utilizzare l'output dell'ultimo livello prima dell'operazione Flatten, il cosiddetto "collo di bottiglia". Il ragionamento alla base di ciò è che i seguenti livelli completamente connessi (FC) saranno troppo specializzati per l'attività su cui è stata addestrata la rete, e quindi le funzionalità apprese da questi livelli non saranno molto utili per una nuova attività. Utilizzando gli stessi file si addestra il modello usando le funzionalità che si sono estratte. La rete verrà addestrata su tutte le 2000 immagini disponibili per 100 epoche, e verrà convalidata su tutte le 1.000 immagini di validazione. Generando un grafico che metta in evidenza la precisione e la perdita nella fase di addestramento e validazione si ottiene:





Dunque utilizzando l'estrazione delle funzioni, si è creato un modello di classificazione delle immagini in grado di identificare i gatti e i cani con una precisione superiore al 90%.

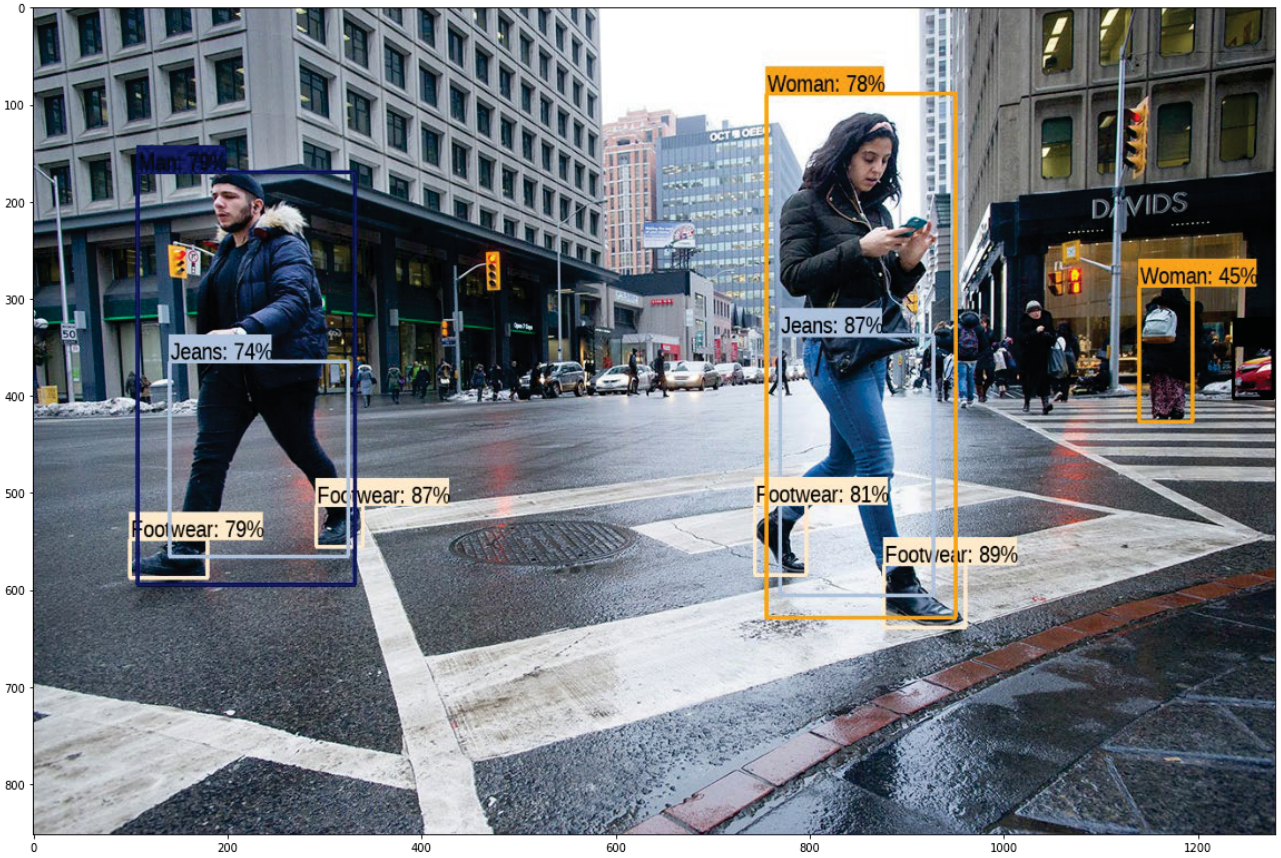
Applicativo per il rilevamento di persone/oggetti nelle immagini

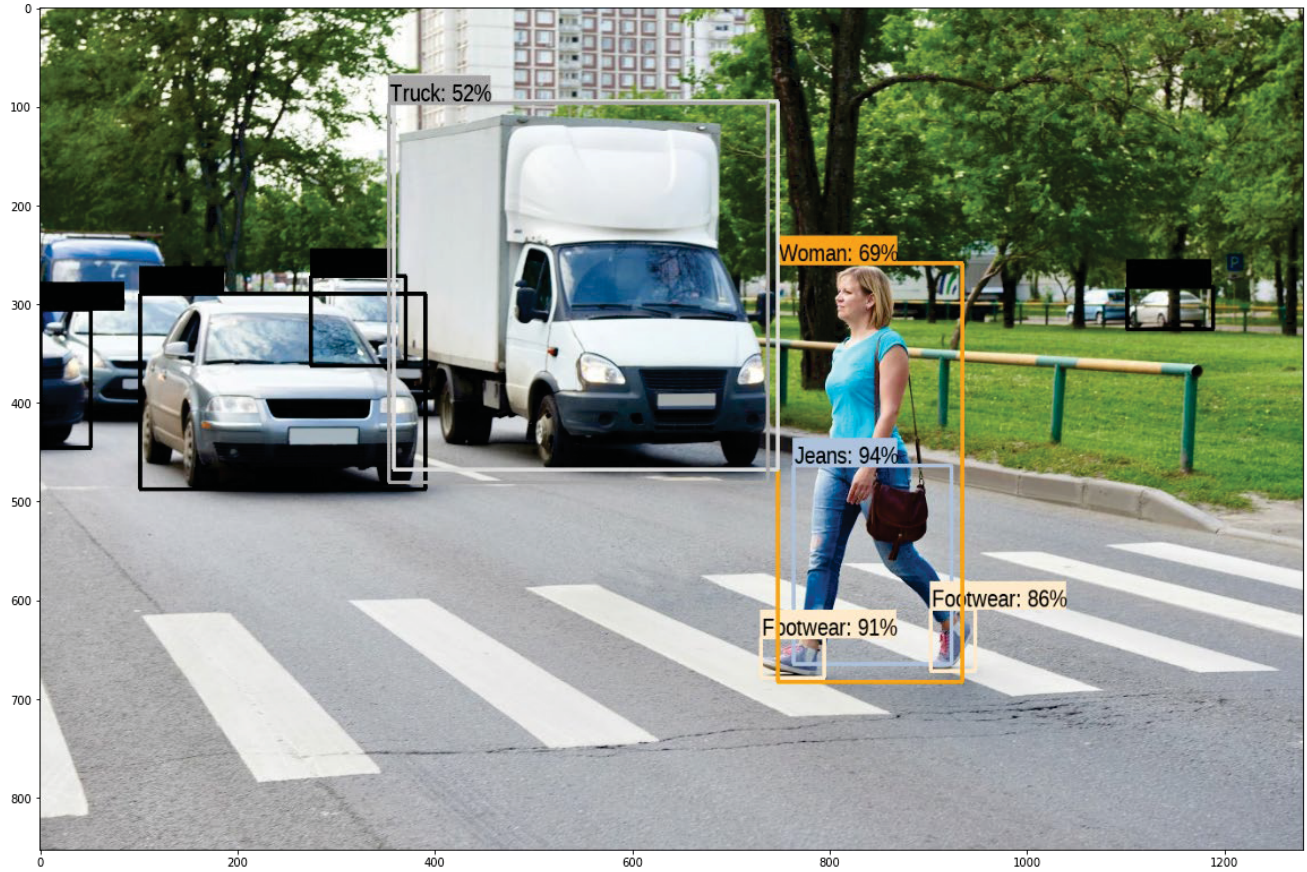
l seguente programma permetterà di rilevare qualsiasi oggetto (rappresentandolo con un rettangolo di diversa dimensione a seconda dell'oggetto rilevato) su qualsiasi immagine data in input attraverso un archivio creato nel proprio drive. Affinché la rete funzioni è necessario fornire un proprio set di immagini (in cui si vogliono rilevare gli oggetti) e che le immagini sia nel formato ".jpg" nella dimensione 1280x816.
Il notebook è raggiungibile attraverso il seguente link:

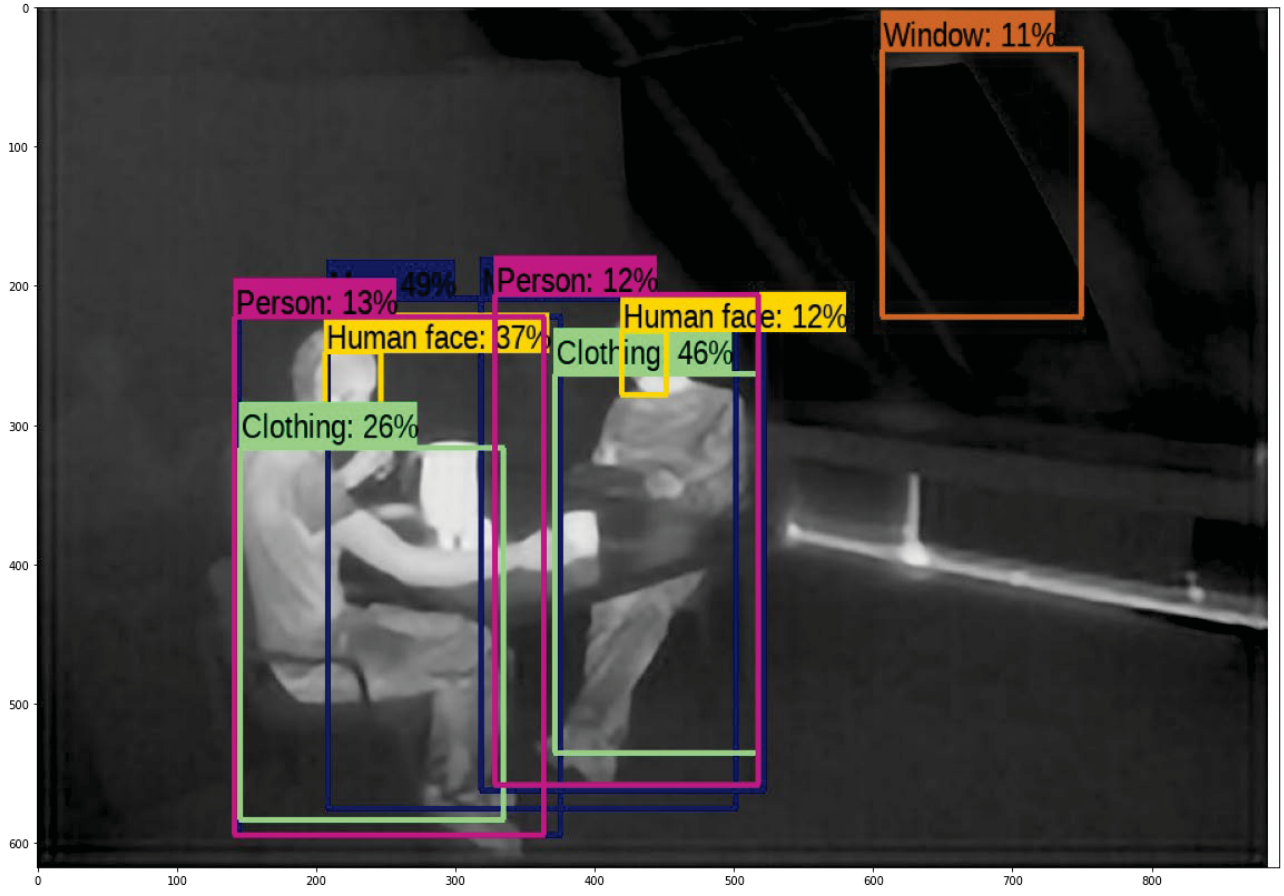
https://colab.research.google.com/drive/1n9mA9-4BDi2WOQwNwU53x7zasKmIs_8h#scrollTo=EmdrkTAyR8mq

Volendo descrivere alcuni passaggi fondamentali del notebook si può dire che:

- Il programma inizia con l'installazione delle librerie necessarie e con delle configurazioni riguardanti le aree dei rettangoli.
- Si crea l'ambiente del contenitore delle immagini.
- Viene fatto scegliere all'utente se si vuole utilizzare una rete FASTER R-CNN o SSD NET (già commentate inizialmente),
- Vengono settati altri parametri di visualizzazione e viene inserito il percorso dove sono situate le immagini in cui la rete deve rilevare gli oggetti.
- La rete va in esecuzione rilevando gli oggetti nelle immagini. Per ogni immagine viene mostrato, attraverso un rettangolo, l'oggetto rilevato e vengono date delle informazioni all'utente riguardanti il numero degli oggetti trovati nell'immagine e il grado di inferenza.
- Al termine dell'esecuzione utilizzando la rete FASTER R-CNN si ottengono le seguenti immagini:


Woman: 78%
Man: 79%
Jeans: 74%
Footwear: 87%
Footwear: 79%
Jeans: 87%
Footwear: 81%
Footwear: 89%
Woman: 45%
DAVIDS


Truck: 52%
Woman: 69%
Jeans: 94%
Footwear: 86%
Footwear: 91%


Window: 11%
Person: 12%
Human face: 12%
Person: 13%
Human face: 37%
Clothing: 46%
Clothing: 26%
0
100
200
300
400
500
600
700
800

Linee guida di implementazione su sistemi embedded ST

L’implementazione su sistemi embedded ST solitamente è realizzata attraverso il pacchetto X-CUBE-AI.

X-CUBE-AI è un pacchetto di espansione STM32Cube che fa parte dell'ecosistema STM32Cube.AI e estende le funzionalità STM32CubeMX con la conversione automatica della rete neurale pre-allenata e l'integrazione della libreria ottimizzata generata nel progetto dell'utente.

Il modo più semplice per utilizzarlo è scaricarlo all'interno dello strumento STM32CubeMX (versione 5.0.1 o successiva) come descritto nel manuale dell'utente X-CUBE-AI per intelligenza artificiale (AI) (UM2526).

Il pacchetto di espansione X-CUBE-AI offre anche diversi mezzi per convalidare i modelli di rete neurale:

- su PC desktop e STM32
- per misurare le prestazioni su dispositivi STM32 privi di codice C inserito dall’utente.

Si differenzia da altri prodotti in quanto permette:

- Generazione di una libreria ottimizzata STM32 da modelli di rete neurale pre-addestrati
- Supporto nativo di vari framework Deep Learning come Keras, TensorFlow ™ Lite, Caffe, ConvNetJs and Lasagne
- Supporto di tutti i framework che possono esportare nel formato standard ONNX come PyTorch ™, Microsoft ® Cognitive Toolkit, MATLAB ® e altri.
- Supporto della quantizzazione a 8 bit delle reti Keras e delle reti quantizzate TensorFlow ™ Lite

- Portabilità attraverso diverse serie di microcontrollori STM32 attraverso l'integrazione STM32Cube
- Termini di licenza gratuiti e intuitivi

Una volta che si è creata la rete neurale convoluzionale bisogna caricare in questo applicativo il formato .ipynb della rete; dopodiché segue una richiesta da parte dell'applicazione che permetterà di scegliere quale chip si intende utilizzare per implementare la rete.

Nel momento in cui si sceglie il sistema embedded, se questo non dovesse riuscire ad eseguire correttamente la rete, solitamente per mancanza di risorse hardware come (dimensioni ram, rom ecc.) il programma permetterà all'utente di scegliere diversi modi per ridurre le risorse utilizzate, dovendo però accettare una certa imprecisione nel risultato.

Ad esempio, se si inserisce una rete che ha bisogno di un grande quantitativo di memoria RAM per rilevare gli oggetti in un chip con poca ram, X-CUBE-AI offrirà delle soluzioni che permetteranno (sempre a titolo di esempio) di andare a ridurre le dimensioni delle immagini in modo tale da alleggerire il carico in memoria, ovviamente ciò comporterà una certa perdita nel risultato finale.

Conclusioni

In questa tesi si sono illustrate le principali reti neurali convoluzionali: CCN, Fast R-CNN, Faster R-CNN e SSD. Nel corso dell'argomentazione è emerso che le reti che negli ultimi tempi sono maggiormente utilizzate risultano essere la Faster R-CNN e l'SSD, entrambe addestrate su un set di immagini molto ampio chiamato Inception Vx (dove la lettera x indica la versione della raccolta).

Dagli applicativi si è constatato che per quanto riguarda la scelta del set delle immagini: tante più immagini si raccolgono, tanto più sarà alta la probabilità che la rete restituisca un risultato desiderato. Nel caso preposto dalla tesi, dunque, un set di immagini come quello offerto dalla FLIR potrebbe risultare ottimo per la creazione di una rete neurale convoluzionale molto accurata. Tuttavia non è importante soltanto la scelta dell'archivio sul quale si addestra la rete ma anche del tipo di convoluzioni che la rete effettua per produrre risultati, sotto questo punto di vista dunque:

- La Faster R-CNN + Inception ResNet non rappresenta certo il metodo più semplice e veloce per la object detection ma il più accurato.
- L'SSD unificando alcuni processi (e quindi perdendo un po' di precisione) è di certo la rete neurale più veloce tra quelle prese in esame.

Quindi per creare una rete neurale che permetta di effettuare una classificazione delle immagini termiche si dovrà scegliere un buon set di immagini e una buona rete.

Si può concludere che per quanto riguarda l'archivio si potrebbe configurare il set della FLIR, in quanto ha un gran numero di immagini che possono essere utilizzate nella fase di training e di validazione. Per quanto riguarda la scelta della rete, questa dipenderà dal fatto se si vuol costruire una rete che sia veloce nel rilevamento, o che sia molto accurata e precisa.
Nel primo caso si utilizzi la rete neurale SSD, nel secondo, invece la rete Faster R-CNN + Inception ResNet. Infine, per implementare la rete su piattaforma embedded, anche se è stata presentata una sola applicazione, quest'ultima è sicuramente la più utilizzata nel mercato in quanto molto intuitiva e veloce dato che permette di ottenere degli ottimi risultati con semplici passaggi.

Concludendo come si è cercato di constatare, le potenzialità delle reti neurali convoluzionali sono molto alte e sono in grado di presentare numerose applicazioni su cui investire in qualsiasi ambito industriale.

Ringraziamenti

Desidero ringraziare innanzitutto il relatore di questa tesi, il professor Claudio Turchetti e la correlatrice Laura Falaschetti, per la disponibilità, l'attenzione e la gentilezza dimostrate durante il tirocinio e la stesura della tesi.

Ringrazio poi tutta la mia famiglia per il sostegno costante e affettuoso di ogni giorno lungo questo cammino universitario.

Un grazie sincero ai miei amici, in particolare a Cristiano per aver sopportato e sostenuto ogni mia preoccupazione e dubbio incontrato lungo il percorso di studi.

Grazie a tutti voi!

Sitografia

Autore Sconosciuto
Reti neurali
https://www.ai4business.it

Joyce Xu
Utilizzo del deep learning per il riconoscimento degli oggetti
https://www.deeplearningitalia.com

Khush Patel
Custom Object Detection using TensorFlow from Scratch
https://towardsdatascience.com

Prabhat Kumar Sahu
Custom Object Detection With Tensorflow Using Google Colab
https://medium.com

Autore Sconosciuto
Applicativo utilizzato per mostrare l'importanza del set di dati in ingresso
https://github.com/

Autore Sconosciuto
Applicativo utilizzato per la rilevazione di persone nelle immagini termiche
https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/object_detection.ipynb

Autore Sconosciuto
X-CUBE-AI
https://www.st.com