

RELAZIONE PROGETTO DI METODI QUANTITATIVI PER L'INFORMATICA

Implementazione di Yolo sul dataset COCO

Pellizzi Elena - 1799226

Data 16/09/2020

ABSTRACT

In questo progetto si utilizza la rete neurale YOLO V2 ⁽¹⁾ con lo scopo di fare object detection, ovvero riconoscimento di oggetti.

In particolare, si farà riferimento a tre classi del dataset COCO ⁽²⁾: laptop, mouse e keyboard.

PRELIMINARIES

La rete neurale YOLO è utilizzata per fare “Object Detection”, ovvero per classificare e localizzare in tempo reale gli oggetti presenti in un’immagine.

YOLO fa uso di reti convolute a passata singola, che analizzano tutte le parti dell’immagine contemporaneamente e la suddividono in diverse regioni: in questo modo la rete neurale è in grado di predire le bounding boxes degli oggetti e determinare, per ciascuno di essi, la probabilità di appartenenza ad una data classe.

In questo caso per ogni immagine è stata usata una griglia di dimensione image_size/32, ovvero 16x16.

Struttura della rete

La rete YOLO è costituita da 24 blocchi convoluzionali (Conv-2D) con Max-Pooling, Batch-Normalization e funzione di attivazione Leaky-ReLu.

- Le **convoluzioni 2D**, in cui ogni livello è rappresentato in 2 dimensioni, fanno uso dell’operatore di convoluzione, che utilizza due matrici:
 - una che rappresenta l’immagine
 - una, detta *filtro* (o *kernel*), alla quale si applica una doppia rotazione, una rispetto all’asse x e una rispetto all’asse y

Dopo aver effettuato la suddetta rotazione, viene effettuata una moltiplicazione “element-wise” tra le due matrici, ottenendo un risultato del seguente tipo:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9)$$

- Il **Max-Pooling** è un’operazione che consiste nel prendere il valore maggiore dell’area di interesse dell’immagine, permettendo così di individuare le sue features più importanti.

- La **Batch-Normalization**⁽³⁾ è un'operazione che permette di risolvere i problemi di *vanishing gradient* ed *exploding gradient*, ovvero situazioni in cui il gradiente tende rispettivamente a 0 (equivalente ad impostare il learning rate pari a 0) o a valori troppo alti (equivalente ad impostare il learning rate ad un valore elevato).

Questa tecnica viene generalmente applicata in ogni hidden layer prima della funzione di attivazione, e consiste nella centratura e normalizzazione di ogni input. Inoltre, calcola media e varianza di ogni input del mini-batch considerato tramite le seguenti formule:

- Media: $\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} \mathbf{x}^{(i)}$ (1)

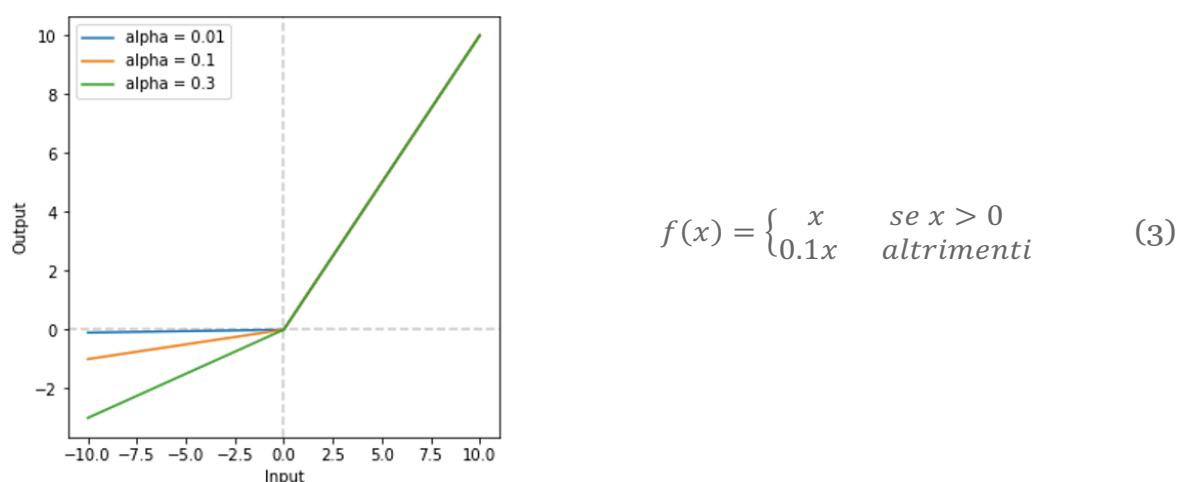
- μ_B è il vettore delle medie di ogni input del mini-batch B
- m_B è il numero di elementi nel mini-batch B

- Varianza: $\sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (\mathbf{x}^{(i)} - \mu_B)^2$ (2)

- μ_B è il vettore delle medie di ogni input del mini-batch B
- σ_B^2 è il vettore delle varianze di ogni input del mini-batch B
- m_B è il numero di elementi nel mini-batch B

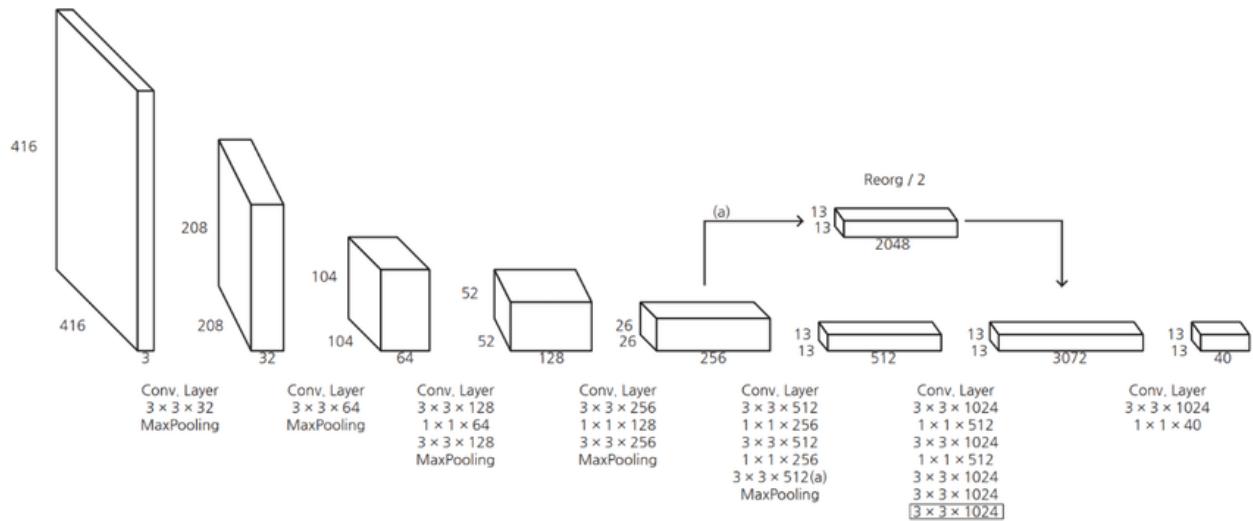
- Le **funzioni di attivazione** permettono di filtrare l'immagine e definire come ogni neurone contribuisca al risultato finale. Esse prendono come input il risultato della somma pesata delle features x_i ($i=1, \dots, n$) moltiplicate per i rispettivi pesi w_i ($i=1, \dots, n$).

In particolare, nella rete YOLO viene usata la funzione Leaky Relu (ovvero una modifica della Rectified Linear Unit, ReLu), che è definita nel seguente modo:



A differenza della ReLu, la Leaky ReLu non spegne i neuroni corrispondenti a $x < 0$, ma li considera con un contributo minimo: così facendo non si avranno neuroni "morti" all'interno della rete, quindi l'output sarà diverso da 0 e, di conseguenza, il gradiente sarà non nullo quando in input alla funzione c'è un valore negativo.

In conclusione, l'architettura di YOLO risulta essere la seguente:



Per risolvere il problema dell'*overfitting*, ovvero per evitare che la rete non sia in grado di generalizzare e quindi di analizzare in modo corretto dati che non sono stati visti durante l'addestramento, nell'ultimo hidden layer viene applicato il **Dropout**.

Si tratta di una tecnica di regolarizzazione della rete basata su un approccio di tipo probabilistico: a ogni neurone è associato un fattore p , detto *dropout rate*, che indica la sua probabilità di non essere attivo durante lo step corrente di addestramento.

Metriche

Un'osservazione importante da fare riguarda la fase di train, in cui viene usata una metrica che prende nome di *Intersection Over Union* (IoU). Essa valuta quanto il modello usato riesce a predire correttamente le bounding boxes, e viene calcolata nel seguente modo ⁽⁴⁾:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

(4)

Definiamo:

- *Area of Overlap* l'intersezione tra le bounding boxes predette e quelle reali
- *Area of Union* l'unione tra le bounding boxes predette e quelle reali

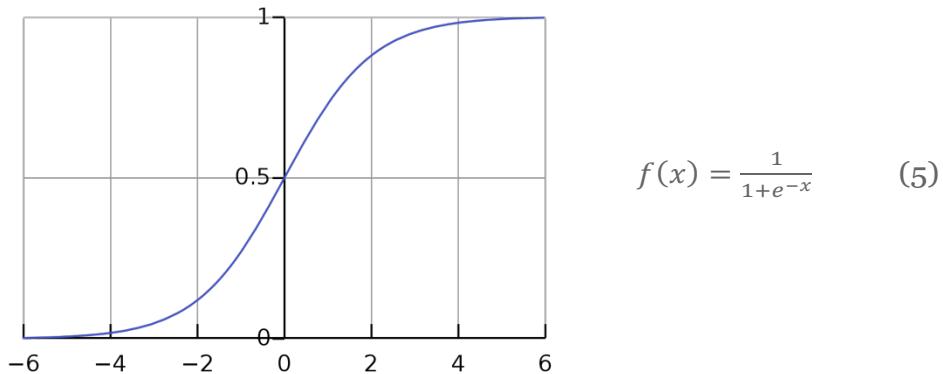
Questo calcolo è utilizzato per vedere lo scostamento tra bounding boxes predette e quelle reali ed il suo risultato sarà un valore tra 0 e 1, dove 1 corrisponde ad avere una sovrapposizione perfetta tra le due aree (ovvero coincidono).

Loss functions

Si tratta di una funzione che misura il grado di accuratezza con cui la rete analizza e descrive il set di dati.

- **Loss sulle coordinate**

Le coordinate x e y delle bounding boxes vengono inizialmente normalizzate tra 0 e 1 tramite la funzione sigmoide, ovvero una funzione del seguente tipo:



La loss complessiva è data dalla somma tra la loss sulle coordinate x e y e la loss sulla larghezza e altezza delle boxes.

- **Loss sulla classe**

Per calcolarla viene usata la “*sparse categorical cross entropy*”.

In generale, si vuole che il modello stimi un’alta probabilità di appartenenza alla classe corretta: tale metrica, a differenza della *categorical cross entropy*, non fa uso di un vettore “one-hot” ma rappresenta le classi tramite numeri interi e lavora in termini probabilistici facendo uso di logaritmi.

- **Loss sulla confidenza**

Si definisce *confidenza* la probabilità che il modello associa a una determinata classe. Questa loss indica quanto le bounding boxes predette si discostano da quelle reali.

- **Loss totale**

È calcolata come la somma delle tre loss precedenti:

```
# total loss  
loss = conf_loss + class_loss + coord_loss
```

(6)

Ottimizzazione

Per l'ottimizzazione del modello si fa uso del gradiente. In questo caso si vuole calcolare l'ottimo della loss function e, per farlo, si utilizza la sua derivata.

Nel caso di reti neurali, i gradienti vengono calcolati durante il passaggio all'indietro, ovvero durante il passo di *backpropagation*: poiché ogni neurone genera un errore che viene propagato in avanti, ai neuroni successivi, in questo modo si può vedere quanto ognuno di essi contribuisca all'errore finale.

Nel codice viene usata una funzione di TensorFlow, denominata GradientTape, per la generazione dei gradienti ed ha il seguente funzionamento:

1. vengono messe su un nastro le variabili indipendenti x_i ($i=1,\dots,n$)
2. viene definita la variabile y , dipendente dalle x_i
3. viene effettuata effettua l'operazione di derivazione

Un altro metodo di ottimizzazione usato è l'*Adam* (Adaptive Momentum).

Si tratta di una tecnica che sviluppa in maniera combinata il *Momentum* e *RMSprop* (Root Mean Square propagation), andando a prendere i vantaggi di entrambi:

- velocizza la ricerca dell'ottimo tramite il Momentum
- riduce la velocità con RMSProp quando si trova vicino all'ottimo

In questo modo si riesce ad arrivare all'ottimo in pochi passi e si riducono di conseguenza i tempi di addestramento.

Adam utilizza tre iperparametri:

- *learning rate* η : parametro che si usa durante l'aggiornamento dei pesi ed è impostato a 10^{-5}
- *momentum decay* β_1 : indica quanto si vuole tenere in considerazione il gradiente del passo precedente ed è impostato a 0.9
- *scaling decay* β_2 : indica quanto influisce il gradiente del passo precedente rispetto al quadrato del passo attuale ed è impostato a 0.999

Inoltre, utilizza le seguenti relazioni:

1. media esponenziale dei gradienti (approccio che utilizza il Momentum):

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t \quad (7)$$

2. Media esponenziale del quadrato dei gradienti (approccio che utilizza RMSprop):

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \quad (8)$$

3. Infine, i pesi vengono aggiornati attraverso la seguente relazione:

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t \quad (9)$$

- ϵ è una costante molto piccola, in questo caso posta pari a 10^{-8} , che serve ad evitare di fare divisioni per zero.

In conclusione, la tecnica Adam riesce ad accrescere il learning rate attraverso il Momentum, e riesce a smorzarlo attraverso RMSprop.
Di conseguenza si ha un learning rate che si adatta alla situazione attuale.

SELECTED DATASET

Il dataset a cui si fa riferimento è COCO (Common Object in Context) 2017, contenente 330 mila immagini, di cui oltre 200 mila etichettate, e 80 classi.

La versione del 2017 si differenzia dalla versione del 2014 per una differente distribuzione delle immagini:

- la versione del 2014 aveva la seguente distribuzione:
 - 82.783 immagini per il train set
 - 40.504 immagini per il validation set
 - 40.775 immagini per il test set
- la versione del 2017 è stata così modificata:
 - 118.287 immagini per il train set
 - 5.000 immagini per il validation set
 - 40.670 immagini per il test set

Come anticipato in precedenza, sono state scelte 3 classi: laptop, mouse e keyboard. Lo scopo del progetto è quello di implementare una rete neurale in grado di riconoscere queste categorie di oggetti, al fine di agevolare e migliorare condizioni lavorative della vita di tutti i giorni.

Si è infatti pensato a realtà comuni come il lavoro in un negozio di riparazioni di computer, in cui si vogliono collocare in parti diverse del negozio le tre categorie di oggetti.

Possiamo per esempio immaginare di mettere su un nastro trasportatore gli oggetti da riparare: l'obiettivo è riconoscere in modo corretto la posizione di ognuno di essi sul nastro e la classe a cui appartiene, così da poterlo prendere e direzionare nella postazione corretta.

Le immagini sono state divise in due set:

- *training set*, contenente 593 immagini
- *validation set*, contenente 45 immagini

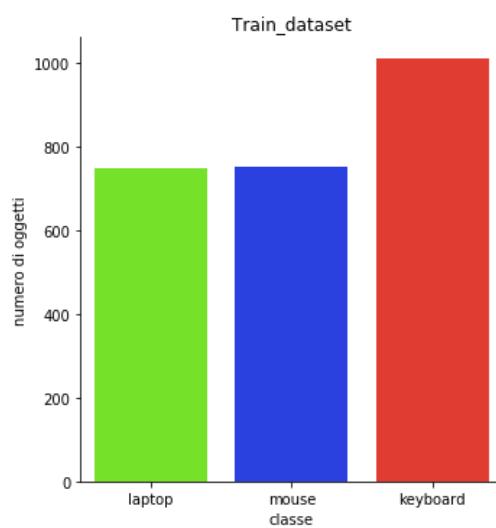
La suddivisione sarà quindi la seguente:

Class Name	Training Set	Validation Set
Laptop	593	45
Mouse	593	45
Keyboard	593	45

In particolare, per i due set possiamo osservare la seguente struttura:

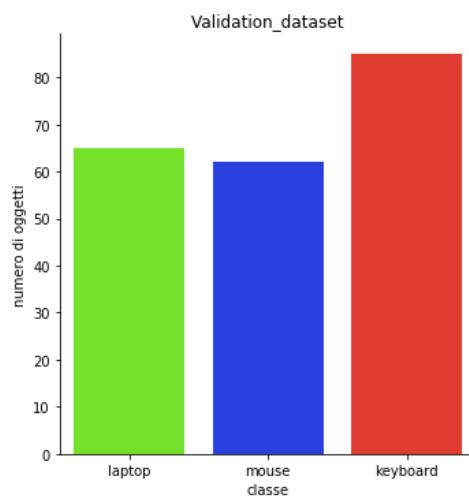
- Train set:

Struttura:		
	classe	numero di oggetti
0	laptop	747
1	mouse	753
2	keyboard	1011



- Validation set:

Struttura:		
	classe	numero di oggetti
0	laptop	65
1	mouse	62
2	keyboard	85



METHOD

Il codice implementato usa la rete neurale YOLO nella versione 2, per la cui struttura si rimanda alla sezione “Preliminaries - Struttura della rete”.

Questa rete risulta essere molto efficiente per il task considerato poiché, fornita in ingresso un’immagine contenente diversi oggetti da classificare e localizzare, riesce a fornire una predizione su di essi in modo estremamente veloce.

Inoltre, la versione utilizzata (YOLO v2) introduce delle *anchor boxes*⁽⁵⁾, che permettono di riconoscere più oggetti all’interno di una singola cella, e le cui dimensioni sono definite a priori:

```
ANCHORS = [0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828]
```

Come per le bounding boxes, anche per trovare le migliori anchor boxes si utilizza la metrica IoU (vedi formula (4)), ovvero per ogni anchor box si calcola quale bounding box ha il valore più alto di IoU.

Se il valore di tale metrica è superiore ad una soglia predefinita IOU_THRESHOLD (nel nostro caso settata al valore 0.45), l’anchor box rileva l’oggetto considerato, altrimenti impara che non ci sono oggetti.

Successivamente viene fatto il calcolo della loss per assicurare il corretto riconoscimento della classe a cui un oggetto appartiene e la sua posizione nell’immagine. In particolare, si è interessati ad ottenere valori bassi per la loss sulle coordinate e quella sulla classe. Si rimanda alla sezione “Preliminaries – Loss functions” per ulteriori chiarimenti.

Poiché il task da soddisfare consiste nel determinare la posizione corretta delle classi di oggetti considerati (ovvero laptop, mouse e tastiere), è necessario avere una loss sufficientemente bassa: dobbiamo quindi impostare il valore di parametri quali *batch_size* ed *epoch_number* affinché l’errore sia sufficientemente piccolo.

IMPLEMENTATION

Preprocessing

In questa prima fase l'algoritmo legge tutto il dataset ed esclude le immagini che non sono di interesse tramite la seguente funzione:

```
def parse_annotation(annotation_dir, img_dir, labels)
```

Dopo aver scelto il subset di immagini su cui lavorare, è stata apportata la seguente modifica: poiché non tutte le immagini erano della stessa dimensione, è stata effettuata un'operazione di ridimensionamento con aggiunta di padding, al fine di preservare le proporzioni originali.

Per fare ciò è stata definita la seguente funzione:

```
def resize_img(img_name):    # img_name = img_dir + elem['file_name']
```

Il suo funzionamento è il seguente:

1. Le immagini vengono ridimensionate tramite la seguente istruzione:

```
im.thumbnail((w, h))      # w = nuova larghezza desiderata  
                          # h = nuova altezza desiderata
```

Tale istruzione mantiene le proporzioni originali dell'immagine, che non viene quindi distorta.

2. Per ottenere le dimensioni desiderate è stato aggiunto del padding tramite la seguente istruzione:

```
constant = cv2.copyMakeBorder(array, pad_top, pad_bottom, pad_left, pad_right, cv2.BORDER_CONSTANT, value=BLACK)
```

Tale funzione fa parte della libreria OpenCV e aggiunge dei bordi neri all'immagine, che viene passata come primo parametro in forma di array numpy.

3. Avendo modificato le dimensioni delle immagini, con aggiunta di padding, è stato necessario modificare anche la posizione delle rispettive bounding boxes:

```
elem['bbox'][0] = (box[0]/cella[1])+cella[3]          # x_min  
elem['bbox'][1] = (box[1]/cella[2])+cella[4]          # y_min  
elem['bbox'][2] = (box[2]/cella[1])+elem['bbox'][0]    # x_max=larghezza+x_min  
elem['bbox'][3] = (box[3]/cella[2])+elem['bbox'][1]    # y_max=altezza+y_max
```

- $cella[1] = w_{old}/w_{new}$
- $cella[2] = h_{old}/h_{new}$
- $cella[3] = padding_x // 2$
- $cella[4] = padding_y // 2$

$Cella[1]$ e $Cella[2]$ sono dei fattori di scala che rappresentano le proporzioni, rispettivamente in larghezza e altezza, tra l'immagine originale e l'immagine ridimensionata.

$Cella[3]$ e $Cella[4]$ indicano quanto padding è stato aggiunto ad ogni immagine dopo essere stata ridimensionata, in modo da ottenere esattamente le dimensioni desiderate.

Data augmentation

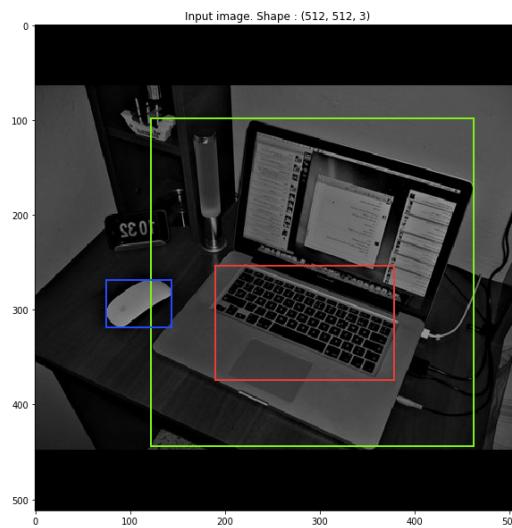
Per aumentare le dimensioni del set usato per l'addestramento sono state applicate tecniche di *data augmentation* quali:

- rotazione rispetto all'asse x
- rotazione rispetto all'asse y
- modifica della luminosità di alcuni pixel.

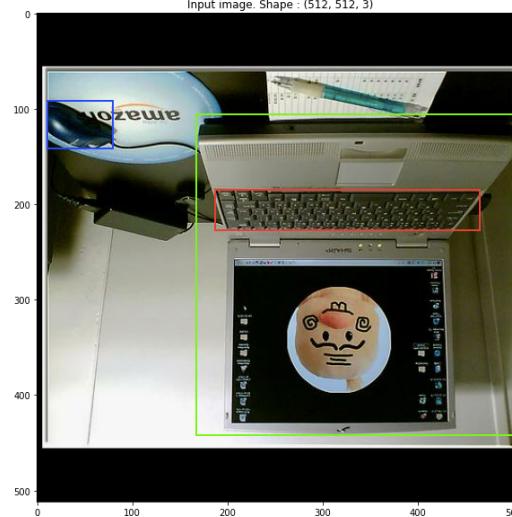
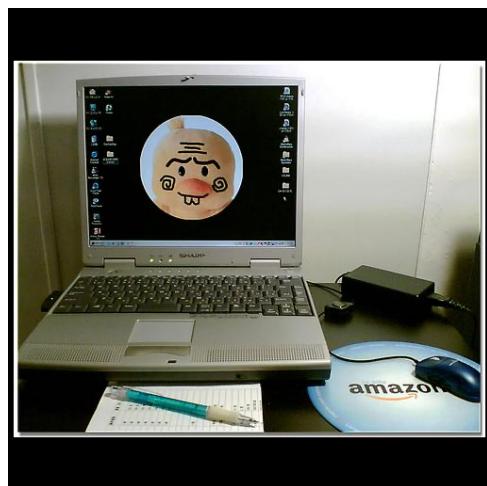
Sono quindi state usate le seguenti funzioni della libreria imgaug di Python:

```
iaa.Fliplr(0.5),          # simmetria rispetto all'asse x
iaa.Flipud(0.5),          # simmetria rispetto all'asse y
iaa.Multiply((0.4, 1.6)),  # change brightness: modifica la luminosità di alcuni pixel
```

Esempio di rotazione rispetto l'asse x con modifica della luminosità dei pixel:



Esempio di rotazione rispetto l'asse y:



EXPERIMENTS

Durante la scrittura del codice sono stati fatti vari esperimenti riguardanti i valori di alcuni *iperparametri* della rete, i quali vengono settati a priori e utilizzati per controllare il processo di apprendimento.

Scelta del numero di epoch e del batch size

Per *epoch* si intende il tempo impiegato dalla rete neurale ad analizzare tutto il dataset. Il *batch size* è la dimensione dei batch, ovvero il numero di campioni analizzati dal modello prima di aggiornare gli altri parametri della rete.

- **Scelta del numero di epoch**

Durante vari esperimenti si è notato che scegliere un numero di epoch troppo basso porta a risultati non ottimali: la perdita di informazioni è troppo elevata e quindi non accettabile.

Considerare invece un numero di epoch troppo grande non comporta un significativo miglioramento del modello: si è notato che le loss, oltre le 250 epoch, tendono sempre ad assestarsi su valori che oscillano tra 0.2 e 0.5 circa.

- **Scelta del batch size**

Tramite esperimenti analoghi a quelli per la risoluzione del numero di epoch, si è notato che scegliere piccole dimensioni risulta essere ottimale per il caso considerato, poiché la rete riesce a fare predizioni migliori sulle classi degli oggetti presenti in ogni immagine.

Considerare invece dimensioni elevate comporta operazioni più onerose, non sempre sostenibili dalle GPU di Google Colaboratory usate in questo progetto. Inoltre, si è notato che la rete trova maggiori difficoltà nel predire le classi considerate.

A fronte di tali osservazioni, sono state effettuate le seguenti scelte:

- Numero di epoch = 200
- Dimensioni dei batch di train = 5
- Dimensioni dei batch di validation = 5

Così facendo si sono ottenuti valori sufficientemente bassi per le loss sulla classe e sulle coordinate, e si è trovato un buon compromesso tra i valori delle loss e le risorse computazionali a disposizione.

Per ulteriori approfondimenti riguardo i risultati ottenuti per ogni esperimento, si rimanda alla sezione successiva, in cui sono stati studiati singolarmente i vari casi.

Scelta della score threshold

Per ottenere dei risultati migliori sono stati effettuati esperimenti per la scelta del valore della *score threshold*, ovvero della soglia usata per predire in modo corretto le bounding boxes: se la predizione porta ad un valore maggiore di tale soglia, allora la bounding box è considerata corretta, altrimenti viene ignorata.

Settare questa soglia a un valore troppo basso porta a predizioni completamente sbagliate, poiché anche le classi predette con probabilità molto basse vengono riconosciute come corrette.

Settarla a un valore troppo alto porta, invece, ad una perdita di informazione notevole poiché solo le classi predette con probabilità elevate vengono classificate come corrette.

È quindi stato scelto un valore intermedio, pari a 0.6.

Anche in questo caso si veda la sezione successiva per maggiori informazioni.

In conclusione, con i parametri scelti, la rete impiega circa 21 minuti per completare l'addestramento, e quindi circa 8 secondi per epoca.

Tale tempo aumenta notevolmente se si scelgono dimensioni più grandi per i batch: scegliendo 200 epoche e dimensioni dei batch pari a 15, il tempo impiegato per l'addestramento è pari a circa 44 minuti.

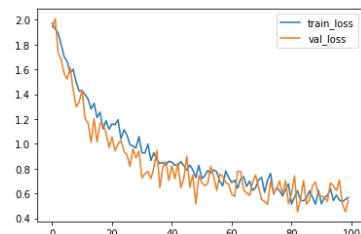
Tutti gli esperimenti e i calcoli dei tempi impiegati sono stati effettuati facendo eseguire l'algoritmo su Google Colaboratory, usando quindi le GPU messe a disposizione da questa piattaforma.

RESULTS

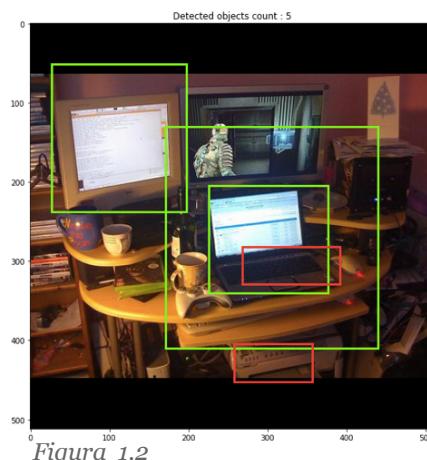
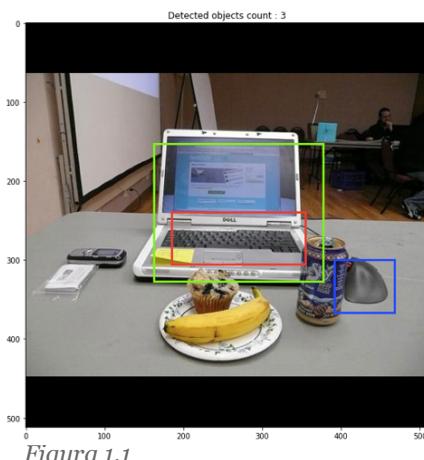
Scelta dei parametri

Per scegliere il valore da assegnare al numero di epoche e batch size sono stati studiati i seguenti casi:

- 100 epoche
 - Batch size = 5
Si ottengono i seguenti risultati:
 - train_loss ≈ 0.5657
 - val_loss ≈ 0.5444
 - confidence_loss ≈ 0.1827
 - class_loss ≈ 0.2017
 - coords_loss ≈ 0.1599



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:



Nella figura 1.1 la rete riesce a riconoscere in modo corretto le classi considerate, con dei piccoli errori sulle coordinate delle rispettive bounding boxes.

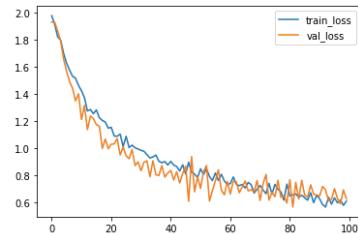
Nella figura 1.2 invece la rete commette diversi errori:

- non riconosce la tastiera sotto al tavolo;
- non riconosce i due mouse a destra delle due tastiere, poiché nascosti dal tavolo e più difficili da individuare;
- riconosce in modo errato due laptop e una tastiera: la rete effettua una generalizzazione tale per cui confonde un generico schermo con un laptop, ed i tasti di una stampante con una tastiera.

○ Batch size = 10

Si ottengono i seguenti risultati:

- train_loss ≈ 0.6096
- val_loss ≈ 0.6264
- confidence_loss ≈ 0.1643
- class_loss ≈ 0.2550
- coords_loss ≈ 0.12071



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

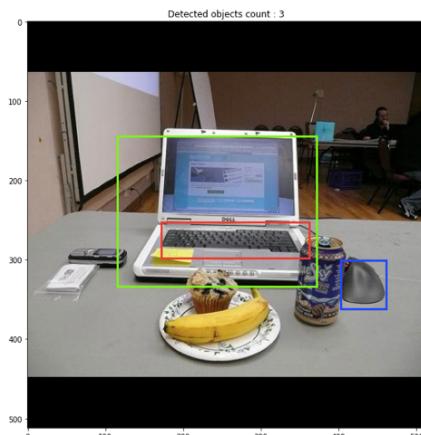


Figura 2.1

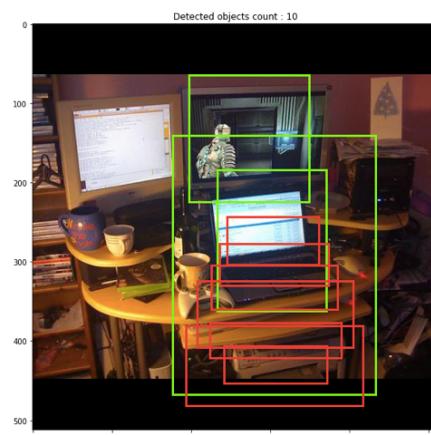


Figura 2.2

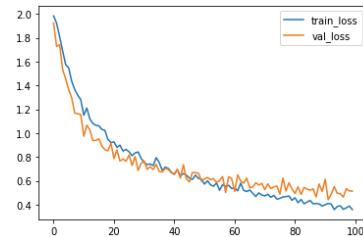
Nella figura 2.1 la rete riconosce correttamente le tre classi di oggetti e, rispetto al caso precedente della figura 1.1, ha un errore minore sulle coordinate delle bounding boxes.

Nella figura 2.2 la rete non riesce ad ottenere buoni risultati poiché riconosce molte superfici rettangolari come fossero tastiere. Anche in questo caso il problema deriva da una generalizzazione eccessiva.

- Batch size = 15

Si ottengono i seguenti risultati:

- train_loss ≈ 0.3584
- val_loss = 0.5128
- confidence_loss ≈ 0.1622
- class_loss ≈ 0.1902
- coords_loss ≈ 0.1604



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

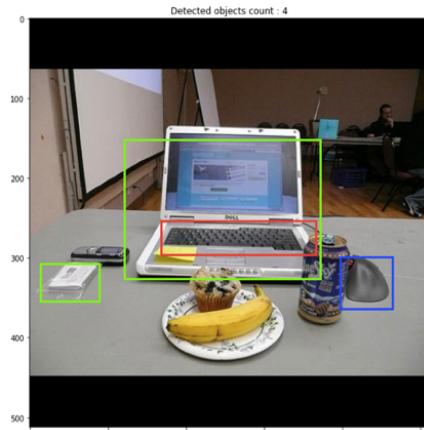


Figura 3.1

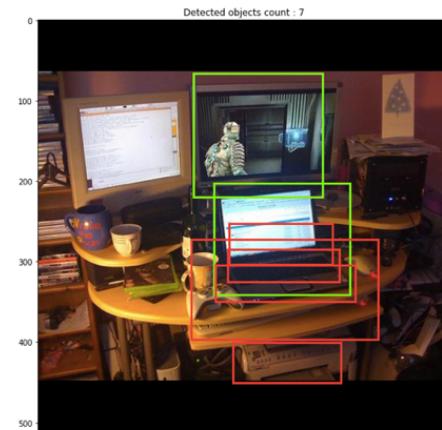


Figura 3.2

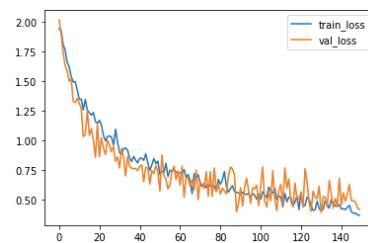
Nella figura 3.1 la rete riconosce in modo errato un laptop, confondendo un oggetto dalla forma rettangolare con un computer portatile chiuso.

Nella figura 3.2 si ha invece un problema simile a quello della figura 2.2.

- 150 epoches
 - Batch size = 5:

Si ottengono i seguenti risultati:

 - train_loss ≈ 0.3688
 - val_loss ≈ 0.4169
 - confidence_loss ≈ 0.1233
 - class_loss ≈ 0.1400
 - coords_loss ≈ 0.1536



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

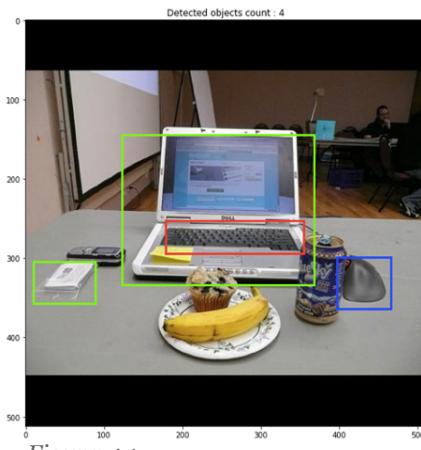


Figura 4.1

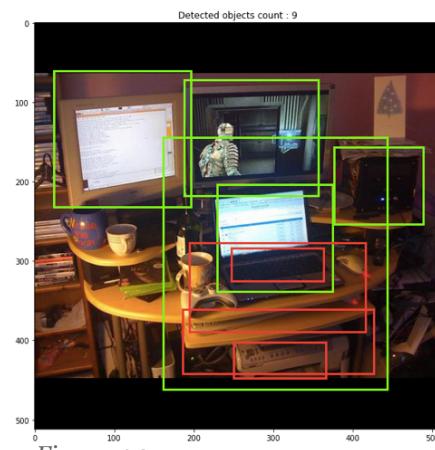


Figura 4.2

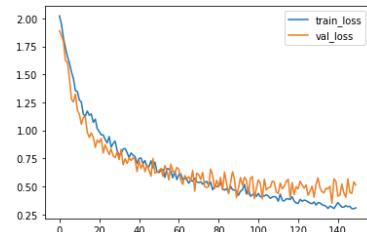
Nella figura 4.1 si ritrova lo stesso problema riscontrato nel caso della figura 3.1.

Nella figura 4.2 ci sono ancora errori dovuti ad una estrema generalizzazione da parte della rete, che non riesce a distinguere correttamente le tre classi considerate.

- Batch size = 10

Si ottengono i seguenti risultati:

- train_loss ≈ 0.3082
- val_loss ≈ 0.5119
- confidence_loss ≈ 0.1369
- class_loss ≈ 0.2365
- coords_loss ≈ 0.1385



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

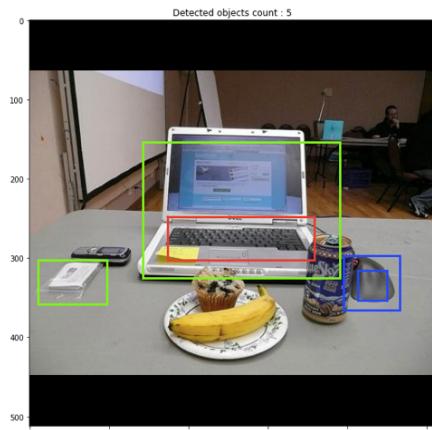


Figura 5.1

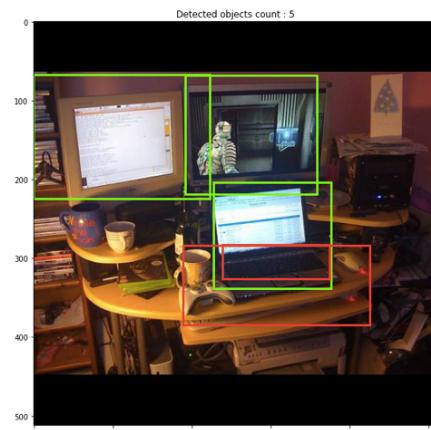


Figura 5.2

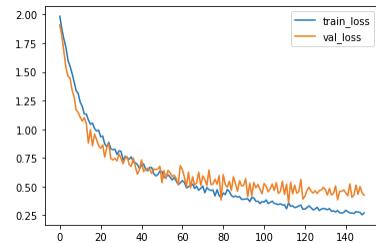
Nella figura 5.1 si nota un comportamento analogo a quello visto per la figura 4.1, ma si trova anche un errore legato al riconoscimento del mouse, che viene individuato due volte.

Nella figura 5.2 si possono invece notare alcuni miglioramenti: la rete non riesce ancora a riconoscere i due mouse, che sono parzialmente nascosti, e considera ancora gli schermi come laptop, ma ha dei miglioramenti nel riconoscere le due tastiere.

- Batch size = 15

Si ottengono i seguenti risultati:

- train_loss ≈ 0.2702
- val_loss ≈ 0.4246
- confidence_loss ≈ 0.1326
- class_loss ≈ 0.1585
- coords_loss ≈ 0.1335



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

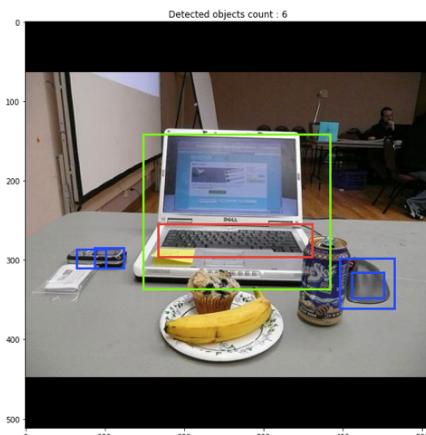


Figura 6.1

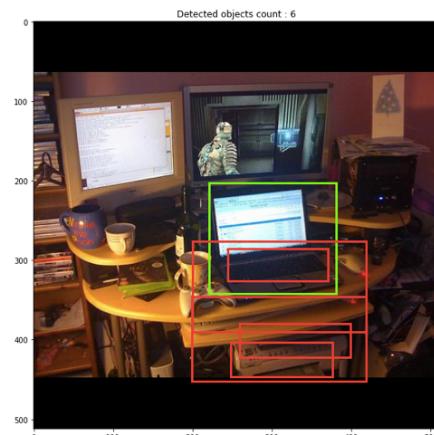


Figura 6.2

Nella figura 6.1 ci sono diversi errori nel riconoscimento dei mouse poiché viene riconosciuto due volte quello nella parte destra della figura e viene scambiato un telefono per un mouse.

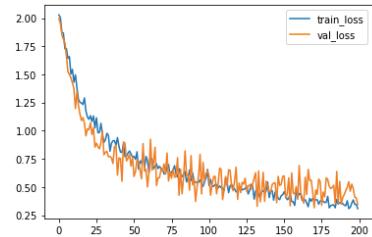
Quest'ultimo errore è dovuto al fatto che il mouse e il telefono hanno circa le stesse dimensioni e gli stessi colori, quindi la generalizzazione effettuata dalla rete non consente di distinguere i due oggetti.

Nella figura 6.2 si può notare ancora una volta come la rete abbia dei problemi nel riconoscere i mouse e le tastiere, ma riconosce in modo corretto il laptop.

Anche in questo caso gli errori riguardanti le tastiere sono dovuti alla generalizzazione da parte della rete.

I mouse non vengono invece riconosciuti poiché sono parzialmente nascosti da altre superfici e presentano una forma leggermente diversa da quelli presenti nelle immagini usate per il train set.

- 200 epoche
 - Batch size = 5
- Si ottengono i seguenti risultati:
- train_loss ≈ 0.3074
 - val_loss ≈ 0.3432
 - confidence_loss ≈ 0.1377
 - class_loss ≈ 0.0705
 - coords_loss ≈ 0.1350



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

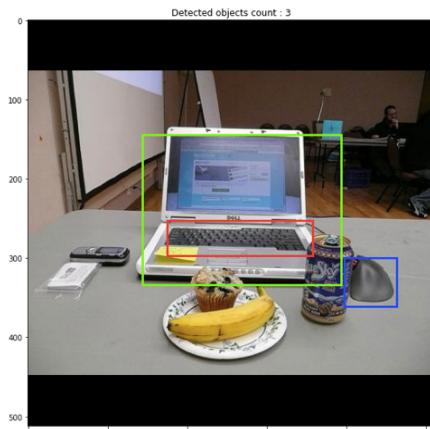


Figura 7.1

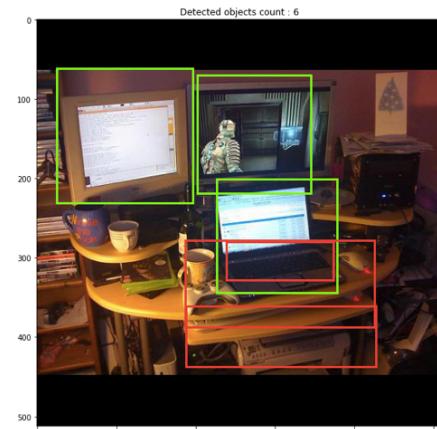


Figura 7.2

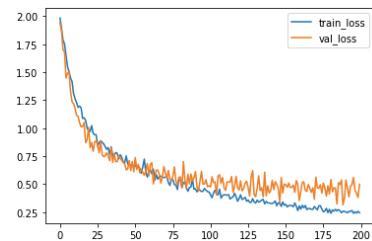
Nella figura 7.1 si può notare come la rete riesca a riconoscere in modo corretto i tre oggetti di interesse presenti in figura, commettendo però piccoli errori riguardo le coordinate delle bounding boxes.

Nella figura 7.2 si ha un caso analogo a quello della figura 5.2, con la differenza che viene anche riconosciuta come tastiera la stampante che si trova nella parte bassa dell'immagine.

- Batch size = 10

Si ottengono i seguenti risultati:

- train_loss ≈ 0.2449
- val_loss ≈ 0.4985
- confidence_loss ≈ 0.1399
- class_loss ≈ 0.2462
- coords_loss ≈ 0.1124



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

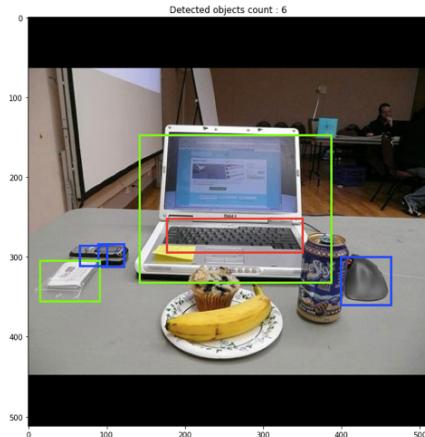


Figura 8.1

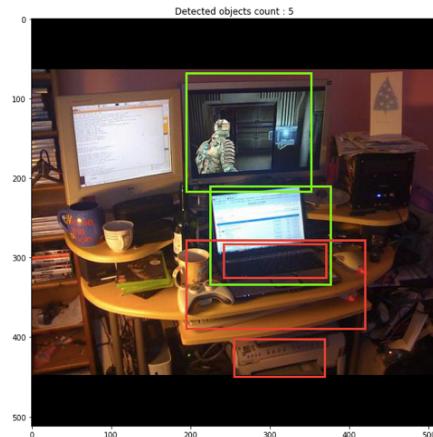


Figura 8.2

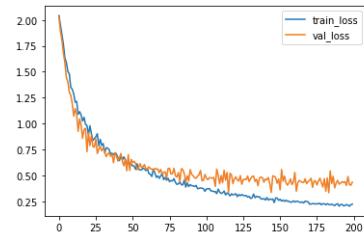
Nella figura 8.1 si possono notare errori sul riconoscimento di laptop e mouse, poiché vengono assegnati a queste due classi anche i due oggetti che si trovano nella parte sinistra dell'immagine.

Nella figura 8.2 la rete riconosce in modo errato uno schermo come fosse un laptop e i tasti della stampante come fossero una tastiera.

- Batch size ≈ 15

Si ottengono i seguenti risultati:

- train_loss ≈ 0.2223
- val_loss ≈ 0.4341
- confidence_loss ≈ 0.1105
- class_loss ≈ 0.1996
- coords_loss ≈ 0.1240



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

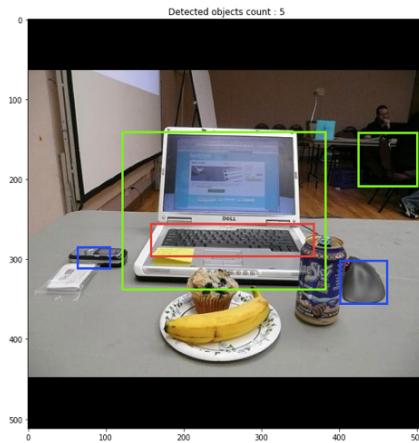


Figura 9.1

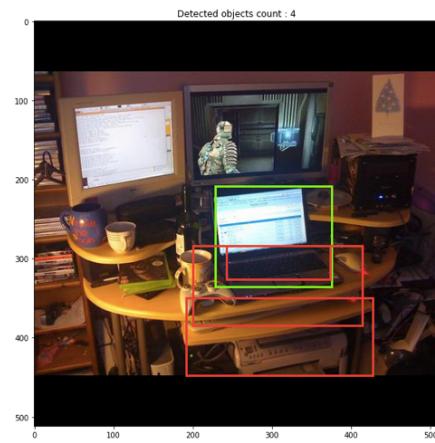


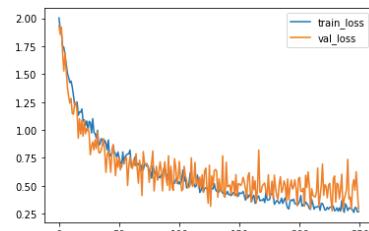
Figura 9.2

Nella figura 9.1 la rete commette ancora una volta un errore sul telefono, classificato come mouse.

In questo caso inoltre classifica una sedia in lontananza come laptop: essendo in penombra, la sedia potrebbe infatti sembrare un computer portatile di colore nero.

Nella figura 9.2 si può notare che la rete fa predizioni simili a quelle che erano state fatte nella figura 6.2, ma si può notare una precisione maggiore nel riconoscere le tastiere.

- 250 epochhe
 - Batch size = 5
- Si ottengono i seguenti risultati:
- train_loss ≈ 0.2699
 - val_loss ≈ 0.2901
 - confidence_loss ≈ 0.1097
 - class_loss ≈ 0.0630
 - coords_loss ≈ 0.1174



Si riportano di seguito due esempi di preazione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

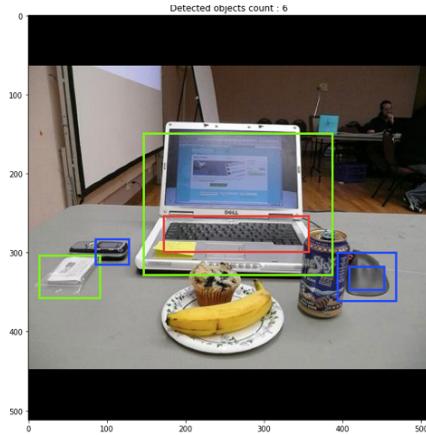


Figura 10.1

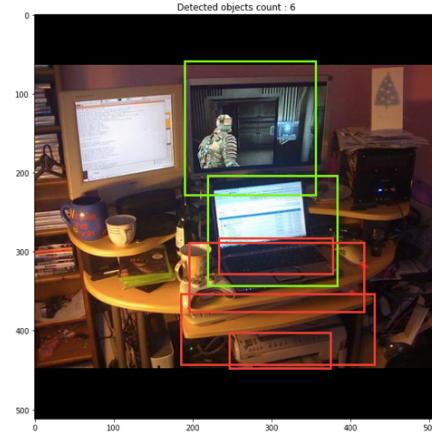


Figura 10.2

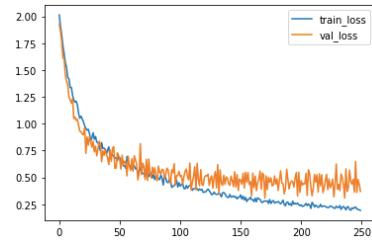
Per la figura 10.1 si rimanda alla figura 5.1: si tratta di casi analoghi, con la sola differenza che in questo caso la rete riconosce anche il telefono come mouse.

Per la figura 10.2 si rimanda invece alle osservazioni fatte per la figura 6.2.

- Batch size = 10

Si ottengono i seguenti risultati:

- train_loss ≈ 0.1941
- val_loss ≈ 0.3709
- confidence_loss ≈ 0.1103
- class_loss ≈ 0.1433
- coords_loss ≈ 0.1173



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

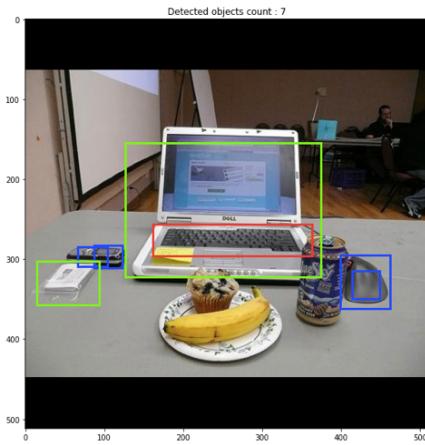


Figura 41.1

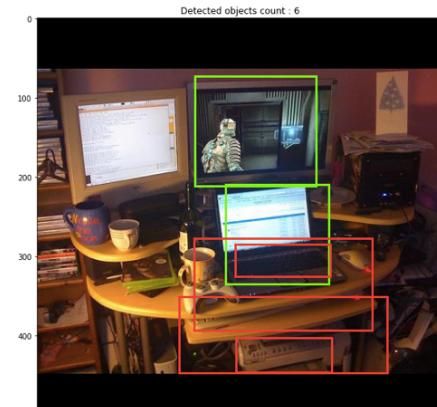


Figura 31.2

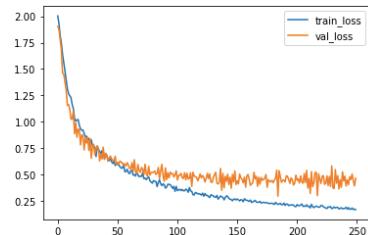
Per la figura 11.1 si rimanda alle osservazioni fatte per la figura 10.1.

Per la figura 11.2 si può invece notare come la rete faccia le stesse predizioni del caso 10.2, con delle piccole modifiche per la posizione delle bounding boxes.

- Batch size = 15

Si ottengono i seguenti risultati:

- train_loss ≈ 0.1653
- val_loss ≈ 0.4632
- confidence_loss ≈ 0.1459
- class_loss ≈ 0.1967
- coords_loss ≈ 0.1206



Si riportano di seguito due esempi di predizione delle bounding boxes da parte della rete neurale, un caso più semplice e un caso più complesso:

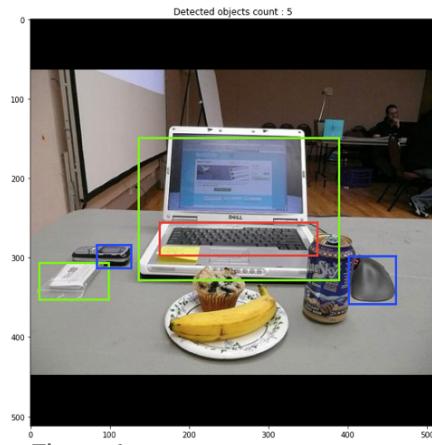


Figura 62.1

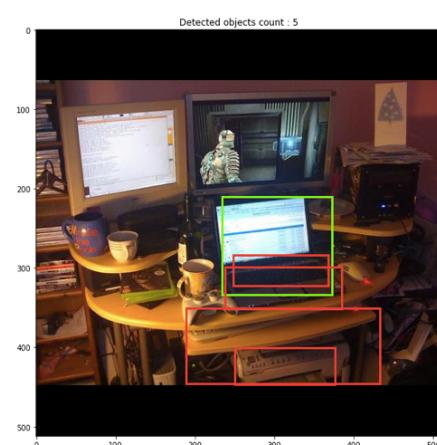


Figura 52.2

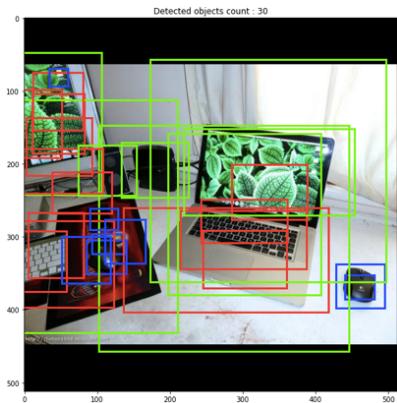
Nella figura 12.1 vengono classificati e localizzati correttamente il laptop, il mouse e la tastiera, ma vengono classificati in modo errato i due oggetti a sinistra nell'immagine.

La figura 12.2 è invece riconducibile al caso analizzato nella figura 8.2.

Per scegliere i valori della score threshold sono stati studiati 4 casi:

- score threshold = 0.3

In questo caso la rete riconosce come corretti anche gli oggetti caratterizzati da una bassa probabilità di appartenere alla classe considerata, ottenendo predizioni sbagliate come nel seguente caso:

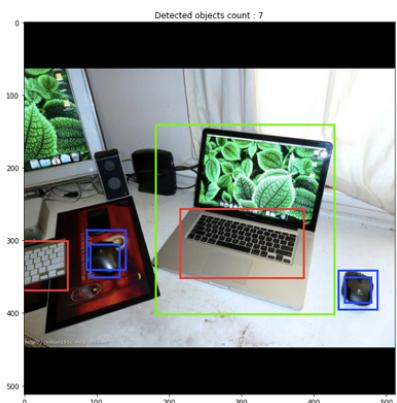


La rete riconosce troppe bounding boxes, portando così ad un errore molto elevato.

- score threshold = 0.5

In questo caso la rete riesce a fare predizioni migliori rispetto al caso precedente, ma non ancora ottimali.

Si riporta di seguito un esempio:



Si può infatti notare come la rete riconosca i mouse in modo doppio, andando quindi ad individuare 7 oggetti di interesse e non 5.

- score threshold = 0.6

In questo caso la rete riesce a fare una predizione ottima, andando ad individuare in modo corretto gli oggetti delle classi di interesse.



- score threshold = 0.7

Aumentando ulteriormente la soglia, si ha una perdita di informazione notevole: solo le classi predette con probabilità elevate vengono classificate come corrette.



In questo esempio infatti si può notare che la rete non riesce a classificare la tastiera e il mouse che si trovano nella parte sinistra dell'immagine.

In conclusione, come detto in precedenza, sono stati scelti i seguenti valori per gli iperparametri considerati:

Iperparametri	Valore
Numero di epoche	200
Train batch size	5
Validation batch size	5
Score threshold	0.6

CONCLUSIONS

Il progetto scelto ha l'obiettivo di semplificare delle operazioni in ambito lavorativo. In particolare, si è considerato il caso di un negozio di riparazione di computer portatili, tastiere e mouse.

Per realizzare questo task è stato scelto un subset del dataset COCO 2017 ed è stato analizzato attraverso la rete neurale YOLO v2.

La rete si è mostrata efficiente nel riconoscere le classi e le coordinate corrette delle bounding boxes, nonostante delle imprecisioni, che possono essere considerate accettabili per l'obiettivo preposto.

A seguito delle scelte sui valori degli iperparametri, si è constatato inoltre che i tempi di addestramento sono notevolmente brevi: la rete infatti impiega solamente una ventina di minuti per analizzare tutte le immagini usate per il train set, e questo potrebbe accelerare i tempi lavorativi.

REFERENCES

- (1) Codice di base della rete YOLO utilizzato:
https://github.com/jmpap/YOLOV2-Tensorflow-2.0/blob/master/Yolo_V2_tf_2.ipynb
- (2) Sito del dataset: <https://cocodataset.org/#home>
- (3) Libro “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition”:
<https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aurélien-Géron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow -Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O'Reilly-Media-2019.pdf>
- (4) Informazioni riguardo l'Intersection Over Union:
<https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>
- (5) Informazioni riguardo le anchor boxes:
<https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9>