Valutazione delle Prestazioni di una rete di Object Detection sul dataset Oxford-IIIT Pet: Transfer Learning vs Architettura Custom

Alex Ardelean

***Abstract*—L'object detection è cruciale per molte applicazioni di computer vision. Le reti neurali convoluzionali (CNN) si sono dimostrate efficaci per questo task, permettendo di ottenere performance molto superiori rispetto ad un approccio con reti neurali fully connected.**

In questo report si presenta un'architettura per l'object detection che integra un classificatore e un modulo di regressione per individuare singoli oggetti e le loro bounding box. Utilizzando il dataset Oxford-IIIT Pet Dataset, l'architettura viene addestrata con tre diverse backbones: due delle quali preaddestrate (*resnet50* e *vgg19*) e una addestrata da zero (*custom backbone*).

L'addestramento è stato condotto utilizzando una loss function multitask, che combina una loss function per la classificazione e una per la regressione. É stata inoltre implementata una politica di early stopping per evitare l'overfitting durante l'addestramento.

I. Introduction

L'object detection, una delle sfide fondamentali nell'ambito della computer vision, continua a ricevere crescente attenzione grazie agli sviluppi tecnologici e alle nuove applicazioni che essa rende possibili. Consiste nell'identificare e localizzare specifici oggetti all'interno di un'immagine o di un video, svolgendo un ruolo cruciale in molteplici settori quali la sorveglianza video, l'automazione industriale, la guida autonoma e la sorveglianza ambientale, solo per citarne alcuni. L'object detection funge da base per molte altre attività di computer vision, come l'instance segmentation, l'image captioning e l'object tracking.

Il task di object detection è afflitto da diverse sfide che possono rendere complesso il processo di individuazione e classificazione degli oggetti all'interno delle immagini. Una delle principali sfide è rappresentata dalle variazioni di scala, gli oggetti infatti possono comparire in dimensioni diverse all'interno dell'immagine e possono essere parzialmente oscurati da altri elementi nell'immagine. Allo stesso tempo, gli oggetti possono essere soggetti a deformazioni e rotazioni e oggetti della stessa classe possono presentare variazioni significative nel loro aspetto, come diverse pose, colori o texture, il che rende difficile per il modello generalizzare correttamente. Le limitazioni hardware e di risorse costituiscono un'altra barriera nel task di object detection. Alcuni modelli richiedono una quantità considerevole di risorse computazionali e di memoria, rendendo difficile l'esecuzione in tempo reale o su dispositivi con risorse limitate. Un'altra sfida critica è rappresentata dalla necessità di annotare i dati. Ottenere un dataset sufficientemente ampio e accurato con annotazioni precise può essere un compito complesso e dispendioso, che richiede tempo e risorse umane considerevoli.

Una rete neurale fully connected consiste in una serie di strati completamente connessi che collegano ogni neurone di uno strato a tutti i neurone del strato successivo. Il principale vantaggio delle reti completamente connesse è che sono agnostiche alla struttura, ovvero non sono necessarie particolari ipotesi sull'input. Sebbene non fare ipotesi sull'input renda le reti completamente connesse general purpose, tali reti tendono ad avere prestazioni più deboli rispetto alle reti specializzate per un particolare task. Utilizzare una rete fully connected per l'object detection può essere impraticabile a causa del grande numero di parametri coinvolti. Inoltre, le reti completamente connesse potrebbero non catturare efficacemente la struttura spaziale e le dipendenze locali presenti nelle immagini. Di conseguenza, altre architetture come le reti neurali convoluzionali (CNN) sono comunemente utilizzate per compiti di questo tipo grazie alla loro capacità di gestire efficientemente le caratteristiche spaziali delle immagini.

Le reti neurali convoluzionali (CNN) sono una classe di reti neurali progettate principalmente per l'elaborazione di dati strutturati come le immagini. Funzionano attraverso l'uso di filtri convoluzionali che vengono applicati in modo iterativo sull'input per estrarre progressivamente caratteristiche di livello superiore. Un filtro (o kernel) di dimensioni ridotte si sposta sull'immagine di input. Ogni volta che il filtro si sovrappone a una parte dell'immagine, calcola il prodotto scalare tra i pesi del filtro e i valori dei pixel sovrapposti. Questa operazione produce una feature map che evidenzia le caratteristiche rilevanti nell'immagine, come bordi, texture, ecc. Dopo la convoluzione, spesso segue uno strato di pooling. Questo riduce la dimensione spaziale della rappresentazione, riducendo il numero di parametri totali. Il max pooling è uno dei tipi più comuni di pooling, in cui si prende il valore massimo all'interno di una regione definita (ad esempio, una finestra 2x2) e si passa quel valore alla feature map successiva. Dopo diversi strati di convoluzione e pooling, le feature estratte vengono appiattite in un vettore unidimensionale e passate attraverso uno o più strati completamente connessi, come in una rete neurale tradizionale. Questi strati apprendono le relazioni tra le diverse caratteristiche estratte e producono un output finale che può essere interpretato per scopi specifici, come la classificazione e la regressione. Ogni strato, compresi gli strati convoluzionali, è solitamente seguito da una funzione di attivazione non lineare come ReLU per introdurre la non linearità nell'output della rete.

A differenza delle reti neurali completamente connesse, in cui ogni neurone di un layer è connesso a tutti i neuroni del layer successivo, nelle CNN gli strati convoluzionali utilizzano filtri localizzati che si spostano sull'input. Questo porta a una interazione sparsa tra i neuroni, poiché ciascun filtro è responsabile solo di una piccola regione dell'input. Nei layer convoluzionali delle CNN, lo stesso filtro viene utilizzato su diverse parti dell'input. Ciò significa che i pesi del filtro sono condivisi su tutta l'immagine, riducendo significativamente il numero di parametri rispetto a una rete neurale completamente connessa. Questa condivisione dei parametri rende le CNN più efficienti e consente loro di generalizzare meglio su nuovi dati. Le CNN tendono ad apprendere rappresentazioni che sono invarianti rispetto alle traslazioni spaziali dell'input. Ciò significa che se l'input subisce una piccola traslazione, le rappresentazioni apprese dalle CNN subiscono una corrispondente trasformazione. Questo è un vantaggio significativo per le applicazioni di visione artificiale, in quanto semplifica il processo di riconoscimento degli oggetti in diverse posizioni all'interno di un'immagine.

II. Related Work

Negli ultimi vent'anni lo sviluppo del'object detection ha attraversato due fasi storiche principali: il periodo precedente al 2014, caratterizzato dal'object detection tradizionale, e il periodo successivo al 2014, dominato dall'object detection basato sul deep learning.

A. Object detection tradizionale

Prima del 2014, l'object detection tradizionale si basava principalmente sull'utilizzo di feature progettate manualmente e su algoritmi di apprendimento automatico classici, come Support Vector Machines o classificatori basati su alberi decisionali. In questo approccio, gli algoritmi dovevano estrarre manualmente le caratteristiche rilevanti dalle immagini, come bordi, texture o colori, e successivamente utilizzarle per addestrare i modelli di rilevamento degli oggetti.

I più importanti contributi nell'era dell'object detection classico sono stati dati da Viola Jones Detectors (2001), HOG (2005) e DPM (2008).

Il Viola Jones Detectors [1] è stato uno dei primi e più influenti algoritmi per l'object detection con prestazioni significative. È stato introdotto nel 2001 da Paul Viola e Michael Jones ed è diventato una pietra miliare nell'ambito del riconoscimento di oggetti e dei sistemi di visione artificiale. Il Viola Jones Detectors utilizza delle feature chiamate Haar-like features, che sono pattern (rettangoli di pixel scuri e chiari disposti in varie configurazioni e dimensioni) di intensità luminosa sull'immagine. Viene utilizzato l'algoritmo di boosting Adaboost per selezionare in modo intelligente un sottoinsieme di feauter Haar-like rilevanti per il riconoscimento dell'oggetto desiderato. Utilizza un approccio sliding window per esaminare l'intera immagine a diverse scale e posizioni e un insieme di classificatori disposti a cascata per discriminare le regioni che non contengono l'oggetto desiderato. Questa combinazione di tecniche permette un rilevamento efficiente e accurato degli oggetti nelle immagini. L'algoritmo viene addestrato utilizzando un set di dati contenenti esempi etichettati dell'oggetto da rilevare.

Il metodo HOG [2] funziona estraendo le feature locali di un'immagine attraverso la computazione degli Histogram of Oriented Gradients. In sostanza, l'immagine viene divisa in celle, e per ogni cella si calcolano i gradienti. Questi gradienti vengono poi utilizzati per costruire un istogramma, che cattura l'orientamento del gradiente dominante in ogni cella. Queste caratteristiche vengono quindi utilizzate per addestrare un classificatore, spesso un Support Vector Machine, per rilevare gli oggetti di interesse. Il principale vantaggio di HOG rispetto a Viola-Jones è la sua capacità di gestire variazioni di scala e deformazioni dell'oggetto

Il Deformable Part-based Model (DPM) [3] migliora il metodo HOG introducendo la capacità di gestire meglio la deformazione degli oggetti e le variazioni di posizione all'interno dell'immagine. DPM divide l'oggetto in parti più piccole e gestisce ciascuna di queste parti separatamente. Questo consente al modello di essere più robusto rispetto a variazioni di posizione e deformazioni locali delle parti dell'oggetto.

B. Object detection basato sul deep learning

Dopo il 2014, con l'avvento del deep learning, l'object detection è stato rivoluzionato dall'uso di reti neurali profonde, in particolare dalle reti neurali convoluzionali (CNN). Questo approccio ha permesso di superare la necessità di effettuare manualmente l'estrazione delle caratteristiche, poiché le CNN sono in grado di imparare automaticamente le caratteristiche rilevanti direttamente dai dati.

Le RCNN (Region-based Convolutional Neural Networks) [4] sono state uno dei primi approcci basati sul deep learning ad affrontare il task di object detection. Le RCNN utilizzano l'algoritmo Selective Search [5] per identificare le regioni promettenti basandosi su criteri come colore, texture e dimensione. Per ogni regione proposta, vengono estratte le feature attraverso una rete neurale convoluzionale (CNN). Le features estratte vengono quindi utilizzate per classificare ciascuna regione proposta in una delle categorie di oggetti predefinite. Questa classificazione può essere effettuata utilizzando un classificatore lineare. Le regioni che sono state classificate come contenenti oggetti vengono quindi sottoposte a un processo di raffinamento delle bounding boxes. Questo processo mira a regolare le dimensioni e la posizione della bounding box per adattarsi meglio all'oggetto rilevato. Le RCNN hanno permesso un significativo aumento delle prestazioni sul dataset VOC07, con un notevole miglioramento della'mAP dal 33,7% di DPM al 58,5%. Le RCNN presentano il problema di essere troppo lente nella loro esecuzione (14s per immagine). Sono quindi state sviluppate le Fast RCNN.

Le Fast R-CNN [6] sono una versione migliorata delle R-CNN originali, progettata per affrontare le loro limitazioni in termini di velocità ed efficienza. A differenza delle R-CNN originali, che estraevano le caratteristiche per ciascuna regione di interesse separatamente, le Fast R-CNN utilizzano una singola rete neurale convoluzionale per estrarre le features dall'intera immagine. Dopo aver estratto le caratteristiche

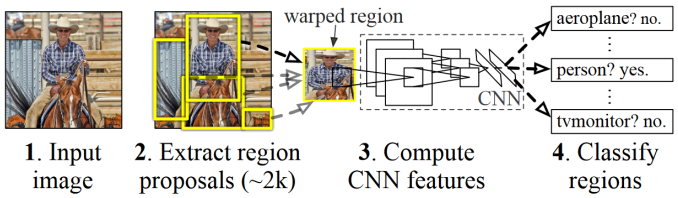

Fig. 1: Architettura RCNN

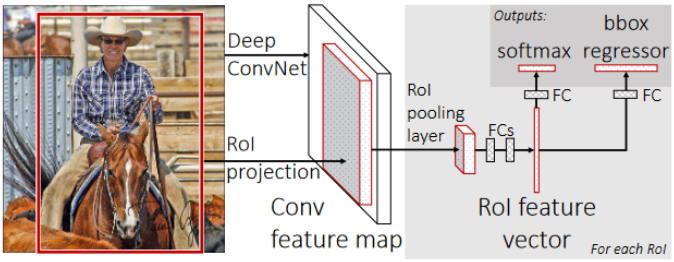

Fig. 2: Architettura Fast-RCNN

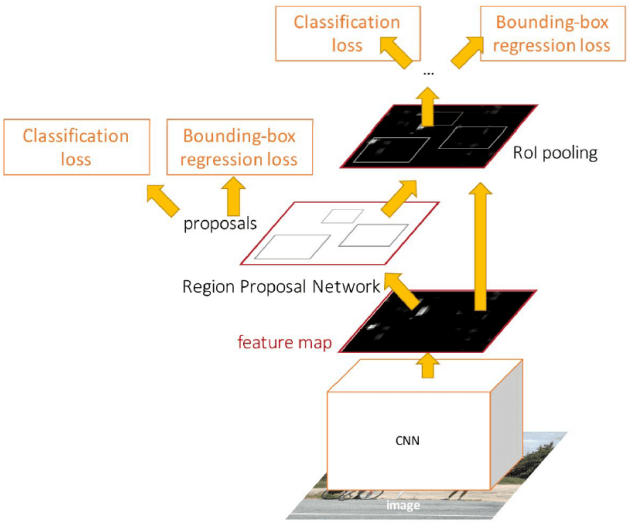

Fig. 3: Architettura Faster-RCNN

dall'immagine, vengono generate le regioni di interesse (RoI) utilizzando un algoritmo di region proposal come Selective Search direttamente sulla feature map in uscita dalla ConvNet. Le RoI sono quindi trasformate in una dimensione fissa tramite una tecnica chiamata RoI pooling. Questo passaggio è fondamentale per consentire la classificazione e il rilevamento degli oggetti su regioni di dimensioni variabili. Le RoI trasformate vengono quindi passate attraverso due bracci distinti della rete neurale: uno per la classificazione delle classi degli oggetti e l'altro per il raffinamento della localizzazione delle bounding box delle RoI. Questo permette di ottenere simultaneamente sia la classe dell'oggetto che la sua posizione precisa nell'immagine. Le Fast R-CNN vengono addestrate end-to-end, il che significa che l'intero modello, compresi i moduli di estrazione delle caratteristiche, la generazione delle RoI e la classificazione/rilevamento, vengono addestrati contemporaneamente durante il processo di apprendimento. Le FastRCNN hanno aumentato il mAP su VOC07 dal 58,5% di RCCN al 70% risultando 300 volte più veloce.

Faster R-CNN [7] è un ulteriore sviluppo delle Fast R-CNN, progettato per migliorare ulteriormente la velocità e l'efficienza nell'object detection. Analogamente a Fast R-CNN, l'immagine viene fornita in input a una rete convoluzionale che produce una feature map. Invece di utilizzare Selective Seach sulla feature map per identificare le proposal region, viene utilizzata una rete dedicata a tale scopo. Le proposal region predette vengono poi scalate utilizzando uno strato di roi pooling che viene quindi utilizzato per classificare l'immagine all'interno della proposal region e per predire le bounding box. La Region Proposal Network (RPN) utilizzata per estrarre le region proposal viene addestrata insieme al resto del modello. Inoltre, la funzione obiettivo di Faster R-CNN include le predizioni sulla classe e bounding box sia nella parte della rete che effettua object detection che sulla RPN. Faster R-CNN ha ottenuto un mAP di 73.2% su VOC07 e può essere eseguito a 17 fps.

Gli approcci visti fin'ora sono tutti del tipo Two-Stage Detectors, nel senso che seguono una procedura a due fasi per effettuare object detection. Nella prima fase, viene generata una serie di region proposal nell'immagine che potrebbero contenere oggetti. Nella seconda fase, queste proposte vengono passate attraverso un classificatore per determinare se contengono effettivamente un oggetto e per raffinare le loro posizioni e dimensioni.

Gli approcci di tipo One-Stage Detectors invece eseguono l'object detection in un solo passaggio. Non richiedono la generazione preliminare delle proposal region, predicono infatti direttamente le classi e le bounding box degli oggetti. Questo li rende solitamente più veloci dei two-stage detectors, ma a volte possono avere prestazioni inferiori, specialmente in presenza di oggetti piccoli.

Uno dei primi approcci One-Stage è stato YOLO (You Only Look Onc) [8]. Utilizza un'unica rete neurale convoluzionale che predice contemporaneamente più bounding box e le probabilità delle classi per tali box. Divide l'immagine di input in una griglia $S \times S$. Se il centro di un oggetto si trova in una cella della griglia, quella cella della griglia è responsabile della rilevazione di quell'oggetto. Ogni cella della griglia predice B bounding box e i relativi punteggi di confidenza per tali box. Questi punteggi di confidenza riflettono quanto il modello sia sicuro che il box contenga un oggetto e anche quanto preciso pensa che sia il box predetto. Ogni bounding box è rappresentato da 6 numeri (pc,bx,by,bh,bw,c), dove pc è la confidenza che un oggetto sia presente nel bounding box; bx, by, bh, bw rappresentano la bounding box stessa; c è un vettore che contiene le probabilità delle classi. I box con punteggi bassi vengono eliminati tramite una soglia. YOLO presenta prestazioni peggiori rispetto a FasterRCNN (mAP pari al 52.7% su VOC07), tuttavia può essere eseguito a 155fps.

III. Proposed Approach

L'architettura proposta (figura 5) effettua classificazione e localizzazione e fa uso di una backbone per la feature extraction a cui si è attaccato:

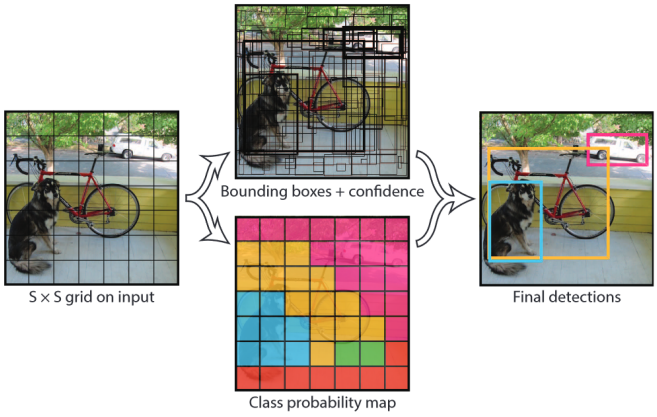

Fig. 4: Approccio YOLO

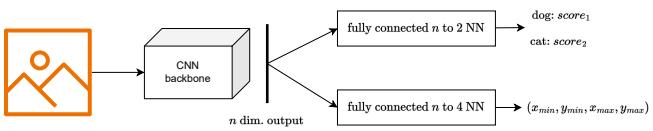

Fig. 5: Architettura proposta

- un classificatore per individuare la classe dell'oggetto;
- un modulo per la regressione per per individuare la bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$;

L'architettura ha quindi come obiettivo non solo quello di riconoscere gli oggetti, ma anche di fornire informazioni spaziali (bounding box) su dove si trovano all'interno dell'immagine. É quindi necessario individuare la bounding box che delimita la zona dell'immagine in cui è presente l'oggetto classificato. Per fare ciò è possibile impostare un problema di regressione che consiste nel determinare i parametri della bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$.

Il dataset deve quindi presentare campioni del tipo $(immagine, classe, x_{min}, y_{min}, x_{max}, y_{max})$. Si è perciò scelto di utilizzare il Oxford-IIIT Pet Dataset [9]. Il dataset contiene oltre 7.000 immagini di 37 razze di cani e 12 razze di gatti. 3685 immagini nel dataset presentano un'annotazione relativa alla bounding box dell'animale. Le informazioni relative alla razza non sono state utilizzate.

A. *Struttura del classificatore e del modulo di regressione*

Per quanto riguarda il classificatore si è utilizzata una rete neurale fully connected che presenta:

- Un input layer lineare seguito da una ReLU con un numero di neuroni pari a 128 connessi ad ognuno degli output della backbone.
- Ci sono due hidden layers, ciascuno con 64 e 32 neuroni rispettivamente. Ogni layer è seguito da una funzione di attivazione ReLU.
- L'ultimo layer è un layer lineare con 4 neuroni, seguito da una funzione di attivazione Sigmoid. Questo è il layer di output che produce la previsione finale. La funzione di attivazione Sigmoid viene utilizzata per produrre output compresi tra 0 e 1.

Per il modulo di regressione si è utilizzata una rete neurale fully connected che presenta:

- Un input layer lineare che mappa l'output della backbone in uno spazio di dimensione 512. Viene applicata la funzione di attivazione ReLU per introdurre non-linearità.
- Un layer di Dropout che aiuta a prevenire l'overfitting durante l'addestramento. Con una probabilità del 60%, i neuroni del layer precedente vengono impostati a zero durante l'addestramento.
- Dopo il primo Dropout, i dati passano attraverso un altro layer lineare che riduce la dimensione dello spazio da 512 a 256. Ancora una volta, viene applicata la funzione di attivazione ReLU per introdurre non-linearità.
- Viene applicato un secondo layer di Dropout, questa volta con una probabilità del 50%.
- Infine, i dati vengono trasformati attraverso un ultimo layer lineare che mappa lo spazio di dimensione 256 al numero di classi nel problema di regressione 2.

B. *Loss function*

Per addestrare la rete si fa uso di due loss function distinte, una per la classificazione e una per la regressione, che vengono poi sommate generando la loss finale. Questo tipo di loss viene chiamata loss multitask. Il gradiente viene propagato sui due branch separatamente per poi essere ricongiunto.

Per quanto riguarda la loss function relativa alla regressione si è utilizzata la loss L_{MSE} implementata nel modulo *torch.nn.MSELoss*. Calcola la media degli errori quadratici tra le previsioni del modello e i valori di target. É utile quando si desidera penalizzare gli errori grandi in modo più significativo.

Mentre per la classificazione si è fatto uso della cross entropy loss L_{CE} implementata nel modulo *torch.nn.CrossEntropyLoss*. Questa funzione di loss misura la discrepanza tra la distribuzione delle probabilità previste dal modello e la distribuzione reale delle etichette di classe nei dati. Questa loss function è risultata particolarmente adatta alla rete dato che l'ultimo layer della rete per la regressione non ha una funzione di attivazione e *torch.nn.CrossEntropyLoss* include già l'applicazione della funzione Softmax per calcolare le probabilità delle classi.

Per combinare le due funzioni di loss, la CrossEntropyLoss e la MSELoss, si è definita una funzione di loss complessiva nel seguente modo:

$$L = L_{CE} + L_{MSE}$$

C. *Backbone*

Si sono utilizzate tre backbone:

- *resnet50*: il termine ResNet deriva da Residual Network, e il numero 50 indica il numero totale di strati convoluzionali nella rete. L'idea chiave di ResNet è l'uso di blocchi residui, che consentono alla rete di imparare ad aggiungere le informazioni residue ai dati in ingresso anziché sovrascriverli completamente. Questo approccio è stato dimostrato essere efficace nel mitigare il problema della scomparsa del gradiente durante la retropropagazione nelle reti neurali molto profonde. In

pratica, un blocco residuo in una rete ResNet è composto da due percorsi: uno è il percorso principale che trasforma l'input in un output desiderato (ad esempio, tramite strati convoluzionali), mentre l'altro è un percorso residuo che aggiunge l'input originale all'output del percorso principale. Questo permette alla rete di apprendere le differenze tra l'input e l'output desiderato anziché imparare direttamente l'intera mappatura. I residual layer, o blocchi residui, vengono impilati uno dopo l'altro per formare l'architettura complessiva di una rete neurale ResNet. Ogni blocco residuo è composto da più strati neurali, di solito strati convoluzionali seguiti da funzioni di attivazione come ReLU, e rappresenta l'unità base della rete.

- *vgg19*: caratterizzata da una struttura profonda, composta da 16 strati convoluzionali seguiti da 3 strati completamente connessi per la classificazione. Ogni strato convoluzionale è seguito da un'attivazione ReLU, che introduce non linearità nel modello. Utilizza convoluzioni multiple con piccoli filtri 3x3 e padding di 1 pixel, seguite da strati di pooling. Questa configurazione di convoluzione riduce progressivamente le dimensioni spaziali delle feature map, mentre aumenta il numero di canali.
- *custom backbone*: costituita da layer del tipo:
  - *Conv2d(n_input, n_output, kernel_size=3, stride=1, padding=1)* che prende in input immagini a *n_input* canali e e produce *n_output* feature map in uscita utilizzando filtri kernel di dimensione 3x3. Il padding è impostato a 1 per mantenere le dimensioni dell'immagine in uscita uguali a quelle dell'immagine in ingresso.
  - *ReLU()*: introduce una non linearità.
  - *MaxPool2d(kernel_size=2, stride=2)*: dimezza la dimensione spaziale delle mappe di attivazione precedenti prendendo il valore massimo all'interno di finestre 2x2 e spostandosi di 2 pixel alla volta.

  Questo ciclo di convoluzione, attivazione e pooling si ripete per un totale di cinque volte:

  1) *Conv2d(3, 16, kernel_size=3, stride=1, padding=1)* seguito da *ReLU()* e *MaxPool2d()*
  2) *Conv2d(16, 32, kernel_size=3, stride=1, padding=1)* seguito da *ReLU()* e *MaxPool2d()*
  3) *Conv2d(32, 64, kernel_size=3, stride=1, padding=1)* seguito da *ReLU()* e *MaxPool2d()*
  4) *Conv2d(64, 128, kernel_size=3, stride=1, padding=1)* seguito da *ReLU()* e *MaxPool2d()*
  5) *Conv2d(128, 128, kernel_size=3, stride=1, padding=1)* seguito da *ReLU()* e *MaxPool2d()*

In tutti i casi è stato rimosso il classificatore finale fully connected e sostituito con i branch relativi alla classificazione e alla regressione.

D. Ottimizzazione

Per la fase di ottimizzazione si è utilizzato il metodo Adam. Esso combina il concetto di momento del gradiente con un tasso di apprendimento adattivo per aggiornare i pesi del modello durante l'addestramento. Questo lo rende efficiente accelerando la convergenza.

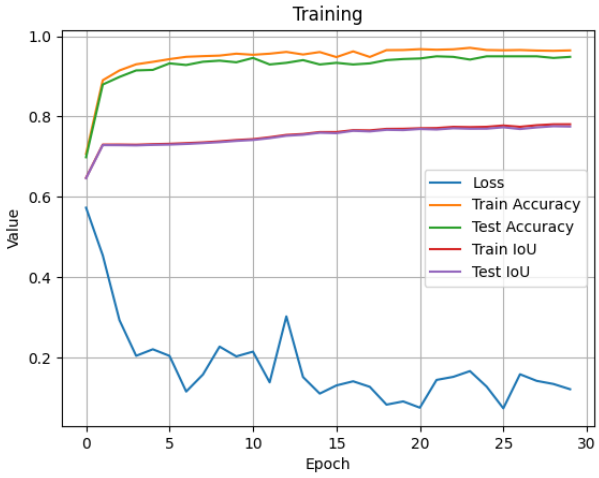

Fig. 6: Addestramento tramite *resnet50*

La politica di early stopping adottata segue le seguenti regole:

- Se l'accuratezza nella classificazione sale per 3 epoche consecutive sul training set e contemporaneamente scende per 3 epoche consecutive sul validation set, l'addestramento del classificatore viene fermato e vengono ripristinati i pesi relativi a 3 epoche prima.
- Se l'IoU nella regressione sale per 3 epoche consecutive sul training set e contemporaneamente scende per 3 epoche consecutive sul validation set, l'addestramento del modulo di regressione viene fermato e vengono ripristinati i pesi relativi a 3 epoche prima.

L'addestramento complessivo della rete termina al raggiungimento del numero massimo di epoche (30) oppure quando sia il classificatore che il modulo di regressione hanno raggiunto la condizione di early stopping.

IV. Experiments

L'esperimento condotto consiste nell'addestramento dell'architettura proposta utilizzando due backbone diverse (*resnet50* e *vgg19*) e il confronto dei risultati ottenuti.

A. Addestramento tramite resnet50

L'andamento dell'addestramento della rete tramite backbone *resnet50* è riassunto nella figura 6.

É possibile notare come dopo poche epoche i valori di accuracy e IoU si sono quasi del tutto stabilizzati. Tuttavia, la condizione di early stopping è stata raggiunta solo dal classificatore all'epoca 23. Il valore di accuracy ottenuto è stato di 95% mentre quello di IoU è stato 77.5%.

B. Addestramento tramite vgg19

L'andamento dell'addestramento della rete tramite backbone *resnet50* è riassunto nella figura 7.

Anche in questo caso dopo poche epoche i valori di accuracy e IoU si sono quasi del tutto stabilizzati. In questo caso, la condizione di early stopping è stata raggiunta solo dal modulo

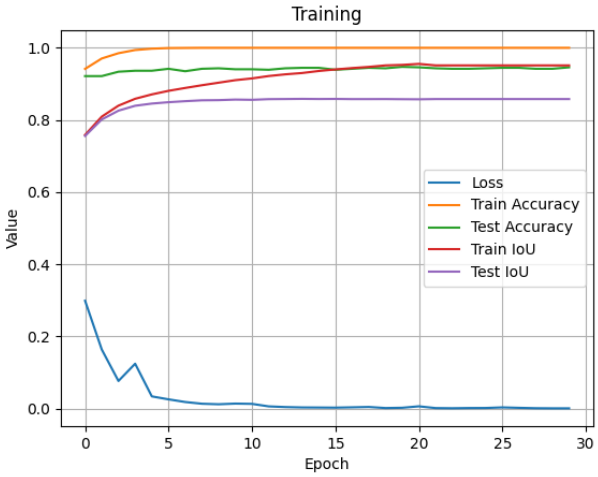

Fig. 7: Addestramento tramite *vgg19*

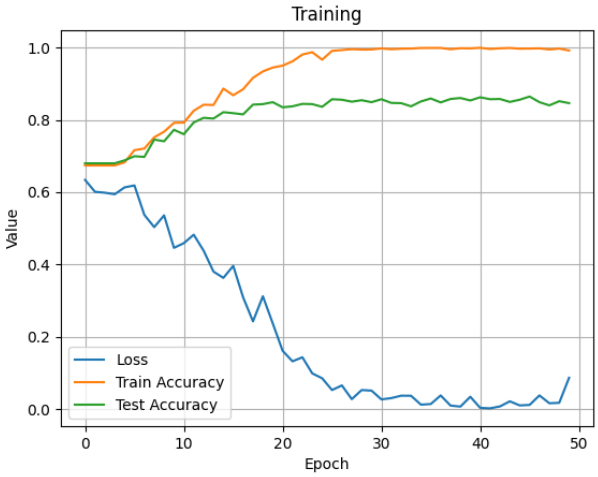

Fig. 8: Addestramento della *custom backbone*

di regressione all'epoca 20. Il valore di accuracy ottenuto è stato di 94.6% mentre quello di IoU è stato 85.8%.

C. Addestramento tramite custom backbone

A differenza degli altri due casi si è prima dovuto addestrare la backbone che effettua feature extraction. Per fare ciò si è deciso di utilizzare un classificatore analogo a quello utilizzato per l'object detection da attaccare alla parte finale della *custom backbone* descritta precedentemente. Dato che il dataset presenta 7383 immagini classificate (e solo 3685 annotate con le bounding box) si è deciso di addestrare la backbone utilizzando solo un classificatore al fine di poter sfruttare l'intero dataset.

I risultati dell'addestramento della backbone sono riassunti in figura 8. Il valore di accuracy ottenuto è stato di 86.4%.

Si è infine addestrato l'object detector utilizzando la *custom backbone* precedentemente addestrata. Il valore di accuracy ottenuto è stato di 92.1% mentre quello di IoU è stato 76%.

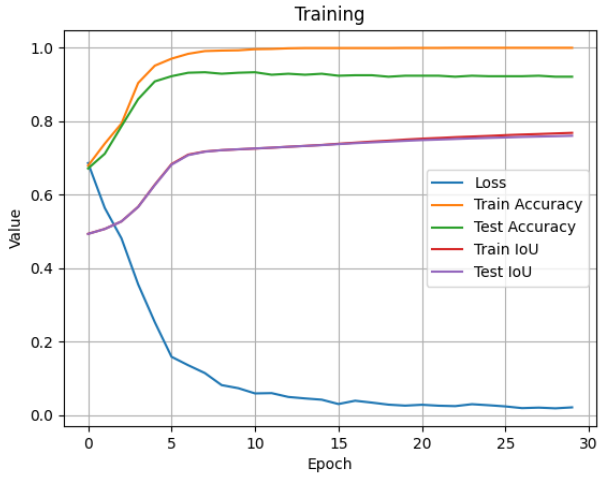

Fig. 9: Addestramento dell'object detector tramite *custom backbone*

Backbone	Accuracy	IoU	Exec. Time	Backbone Training Dataset
resnet50	95%	77.5%	7.32 ms	ImageNet
vgg19	94.6%	85.8%	5.07 ms	ImageNet
custom backbone	92.1%	76.0%	1.27 ms	Oxford-IIIT Pet

TABLE I: Confronto risultati

D. Confronto

Risulta quindi evidente che la rete avente come backbone *vgg19* ha ottenuto prestazioni migliori (vedi tabella I). Questo potrebbe essere dovuto al fatto che *vgg19* presenta 144 milioni di parametri addestrabili mentre *resnet* ne ha solo 23 milioni. In effetti osservando l'andamento dell'IoU nell'addestramento della rete con backbone *vgg19* è possibile notare come la differenza dei valori nel training set e nel validation set siano molto maggiori rispetto all'addestramento effettuato sulla rete con backbone *resnet50*.

I tempi di esecuzione per le predizioni sono paragonabili, si ha infatti una media di 7.32 ms per resnet50, 5.07 ms per vgg19 e 1.27 ms per la custom backbone. Il maggiore tempo di resnet50 potrebbe dipendere dalla sua profondità maggiore, 50 strati convoluzionali nel primo caso e 16 nel secondo. Per ottenere questi risultati si sono eseguite le due reti per 10000 volte sulla stessa immagine e si è misurato il tempo di esecuzione.

V. Conclusion

Gli esperimenti condotti hanno dimostrato l'efficacia dell'approccio proposto nell'affrontare il problema dell'object detection, combinando l'utilizzo di reti neurali convoluzionali (CNN) come backbone per l'estrazione delle feature con un classificatore e un modulo di regressione per la classificazione e la localizzazione degli oggetti.

É inoltre necessario osservare che l'object detector proposto è capace di individuare un solo oggetto nell'immagine e l'architettura implementata non può essere facilmente estesa per l'individuazione di più oggetti. Per tale scopo è infatti necessario utilizzare Two-Stage Detector, come ad esempio Faster

R-CNN, oppure anche un approccio One-Stage Detector, come ad esempio YOLO.

REFERENCES

[1] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, pp. 137–154, 2004.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[3] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*. Ieee, 2008, pp. 1–8.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[5] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[6] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[9] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3498–3505.