UNIVERSITÀ DEGLI STUDI DI CATANIA

Relazione progetto

Training Object Detector
EfficientDetV0 & EfficientDetV4

Gioele Cageggi & Pietro Andrea Vassallo

Anno accademico 2020-2021

Computer Vision - Prof. Sebastiano Battiato

Introduzione

Uno dei principali problemi nella computer vision è l'object detection.

L'object detection è un importante problema che combina le attività di image classification e object localization e che consiste nell'individuare istanze di oggetti all'interno di un'immagine e nel classificarle come appartenenti ad una certa classe (come umani o auto).

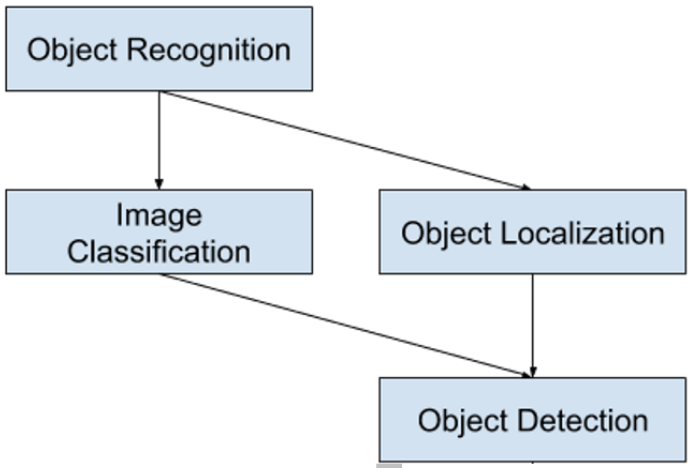
L'obiettivo è quello di sviluppare tecniche e modelli computazionali che forniscano uno degli elementi basilari necessari per le applicazioni di computer vision: comprendere quali oggetti sono presenti nella scena catturati da una camera RGB. Nel contesto di questo progetto gli oggetti del dominio da riconoscere saranno le classi "people" e "helmet".

In questo elaborato verrà analizzato il task di object detection e successivamente le reti, EfficentDetV0 e EfficentDetV4, che verranno utilizzate per effettuare un'operazione di Transfer Learning, tramite la libreria di TensorFlow: Model Maker. Infine verranno confrontati e discussi i risultati ottenuti tramite le metriche COCO su dati di training.

Inoltre, per una maggiore completezza, è stata sviluppata un'applicazione Android capace di acquisire la scena, tramite le API Camera2API, che permette di eseguire inferenze di modelli TensorFlow lite. I risultati delle inferenze, ovvero bounding box e labels degli oggetti riconosciuti, verranno disegnati sullo stream elaborato. L'applicazione darà la possibilità di switchare modello .tflite in real time.

Object Detection

Considerando che il task di Image Classification è responsabile di assegnare una etichetta di classe, o class label, ad una immagine e il task di Object Localization si occupa di identificare una regione d'interesse all'interno di una immagine allora potremmo definire il task di Object Detection come la combinazione dei due task accennati: all'interno di un'immagine, localizzare la presenza di oggetti con dei bounding box e assegnare una label class per ogni oggetto localizzato.



Esistono diverse famiglie di modelli che permettono di risolvere il task di Object Detection come R-CNN, SSD, YOLO in modi piuttosto diversi ma in generale un sistema di Object Detection è possibile identificare quattro componenti:

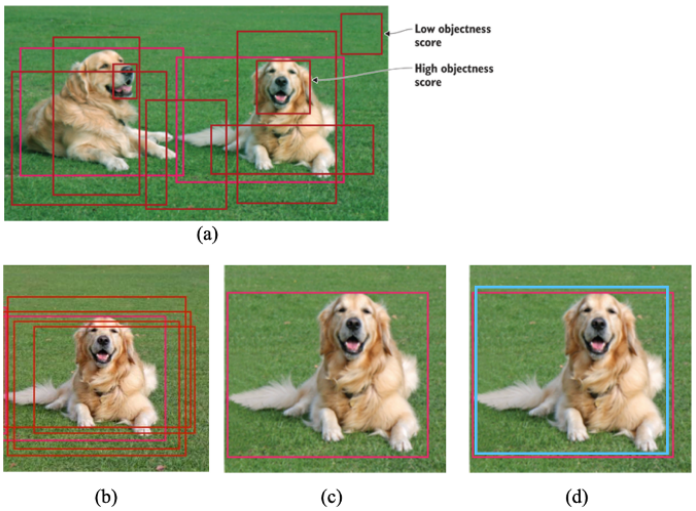
1. *Region Proposal*, un algoritmo o un modello di deep learning è utilizzato per generare regioni di interesse (RoI) da essere ulteriormente processati dal sistema. Queste sono regioni che la rete creda possa contenere un oggetto. L'output è un insieme di bounding box, ognuno dei quali ha associato uno score. I box con

score più alto saranno quelli che andranno avanti nei layer della rete per ulteriori analisi.

2. *Feature Extraction e Predizione*, le feature visuali sono estratte per ognuno dei bounding box. Esse vengono valutate e si determina se e quale oggetto è presente nella proposal, basandosi esclusivamente sulle feature visuali.

3. *Non-Maximum Suppression*, arrivati in questo step il modello, tipicamente, ha identificato multipli bounding box per lo stesso oggetto. Tramite la Non-Maximum Suppression si rimuovono le detection ripetute relative allo stesso oggetto combinando i vari bounding box sovrapposti in un unico bounding box, questo avviene per ogni oggetto.

4. *Evaluation Metrics,* similmente all accuracy, precision e recall nei task di classificazione, anche l'object detection ha i suoi sistemi di misurazione per valutare la performance di detection. Esempi sono mean Average Precision (mAP), la curva Precision-Recall (PR curve) e l'intersezione sull'unione (IoU).



EfficientNet

L'efficacia del ridimensionamento del modello dipende anche in larga misura dalla rete di base. Quindi, per migliorare ulteriormente le prestazioni, google ha sviluppato una rete di base eseguendo una ricerca sull'architettura neurale utilizzando il framework AutoML MNAS, che ottimizza sia l'accuratezza che l'efficienza (FLOPS). L'architettura risultante utilizza la convoluzione del collo di bottiglia invertito mobile (MBConv), simile a MobileNetV2 e MnasNet , ma è leggermente più grande a causa di un maggiore budget FLOP. Quindi si ingrandisce la rete di base per ottenere una famiglia di modelli, chiamata EfficientNets.

L'architettura per la rete di base EfficientNet-B0 è semplice e pulita, rendendo più facile la scalabilità e la generalizzazione.

EfficientDet

In "EfficientDet: Scalable and Efficient Object Detection", introduce una nuova famiglia di rilevatori di oggetti scalabili ed efficienti. Basata sul precedente ridimensionamento delle reti neurali (EfficientNet) e incorporando una nuova rete di funzionalità bidirezionali (BiFPN) e nuove regole di ridimensionamento, EfficientDet raggiunge la massima precisione pur essendo fino a 9 volte più piccola e utilizzando significativamente meno calcolo rispetto ai rivelatori all'avanguardia.

BiFPN

Il problema con l'FPN convenzionale, è che è limitato dal flusso di informazioni unidirezionale (dall'alto verso il basso). Per risolvere questo problema, PANet aggiunge un'ulteriore rete di aggregazione dei percorsi bottom-up. Inoltre, ci sono molti documenti, ad es. NAS-FPN, che ha anche studiato le interconnessioni per acquisire una migliore semantica. In breve, il gioco riguarda le connessioni per connettere funzionalità di basso livello a funzionalità di alto livello e viceversa per catturare una semantica migliore.

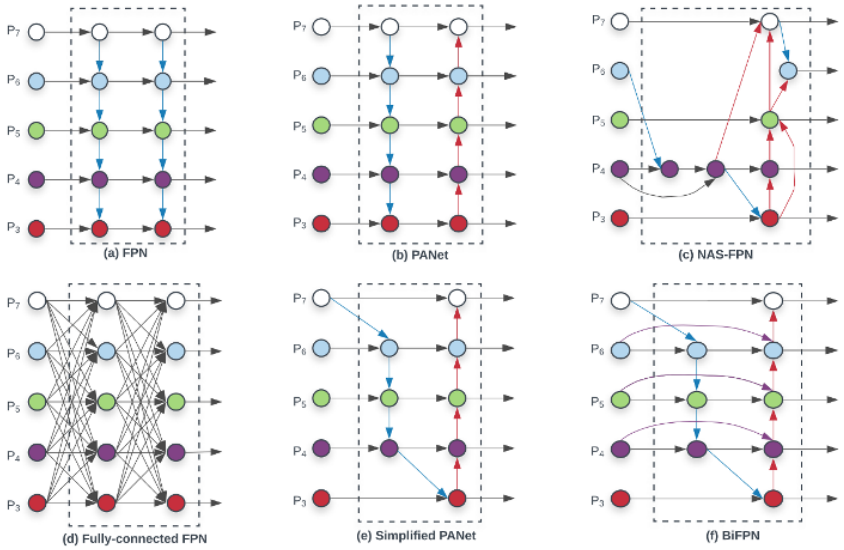


Figura 1 - Design delle Feature networks

Il problema con il NAS è che si basa su Reinforcement Learning che impiega migliaia di ore GPU/TPU per individuare le connessioni migliori. Inoltre, è stato riscontrato che le connessioni incrociate finali trovate dal NAS, come mostrato nella Figura (c), erano irregolari e difficili da interpretare. Anche dopo questo, PANet ha battuto sia FPN che NAS-FPN in termini di precisione, ma ha un costo di calcolo aggiuntivo. Quindi ha senso prendere PANet come base e migliorare le connessioni sia per l'efficienza che per la precisione. Per ottenere connessioni incrociate ottimizzate, gli autori hanno proposto quanto segue: Rimuovere i nodi che hanno solo un bordo di input. Se un nodo ha solo un bordo di input senza fusione di funzionalità, avrà meno contributo alla rete di funzionalità che mira a fondere funzionalità diverse. Questo porta a una PANet semplificata come mostrato nella Figura (e). Aggiungi un edge in più dall'input originale al nodo di output se sono allo stesso livello, al fine di fondere più funzionalità senza aggiungere molti costi, come mostrato nella Figura 1 (f). A differenza di PANet che ha solo un percorso dall'alto verso il basso e uno dal basso verso l'alto, gli autori trattano ogni percorso bidirezionale (dall'alto verso il basso e dal basso verso l'alto) come un unico livello di rete e ripetono lo stesso livello più volte per abilitare più livelli di alto livello fusione di caratteristiche. Il risultato di queste ottimizzazioni è la nuova rete di funzionalità denominata BiFPN, come mostrato nella figura superiore in (f).

EfficientDet

Come suggerisce il nome, utilizzando EfficientNets come rete dorsale insieme a BiFPN, otteniamo una nuova famiglia di detectors chiamata EfficientDet.

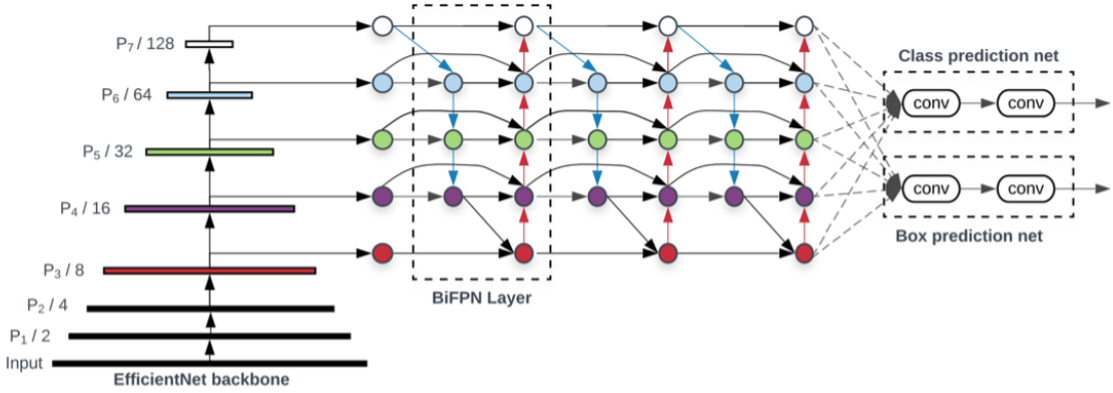


Figura 2 - Architettura EfficientDet

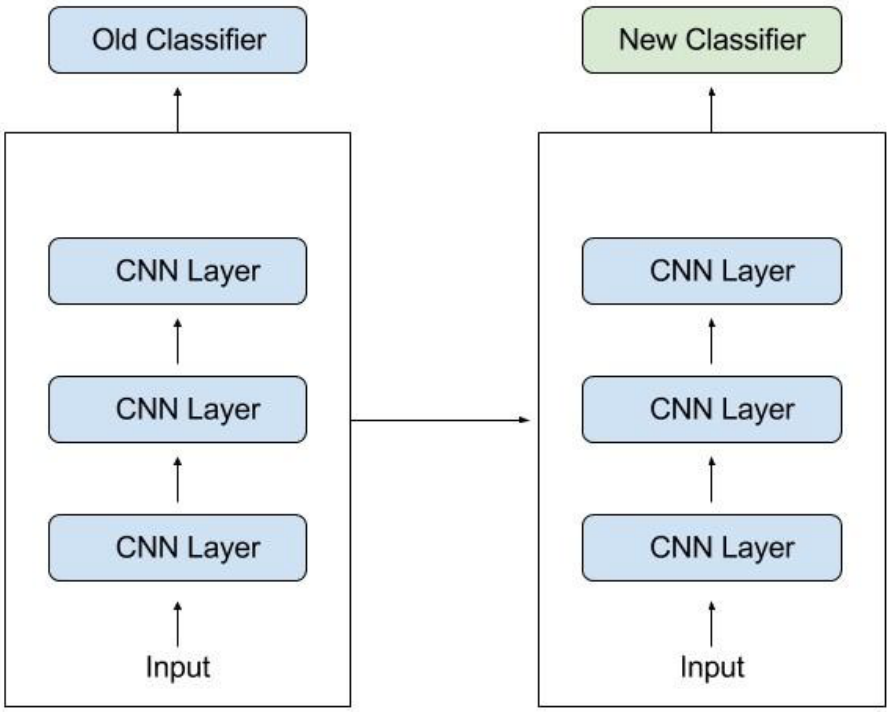
I rilevatori EfficientDet sono rilevatori single shot molto simili a SSD e RetinaNet. Le reti dorsali sono EfficientNet pre addestrate da ImageNet. Il BiFPN proposto funge da feature network, che prende le features di livello 3-7 {P3, P4, P5, P6, P7} dalla rete dorsale e applica ripetutamente la fusione di features bidirezionali top-down e bottom-up. Queste features fuse vengono inviate a una rete di classi e box per produrre rispettivamente previsioni di classi di oggetti e bounding box. I pesi della rete di classe e box sono condivisi tra tutti i livelli di features.

Introduzione al Transfer Learning

Una pratica molto comune è sfruttare una rete già addestrata e utilizzarla per un task successivo. In particolare, vengono presi i pesi dall'addestramento effettuato precedentemente e trasferiti per ri-allenare un modello simile: la rete può partire quindi con dei pesi pre-allenati.

Il transfer learning include anche i processi di feature extraction e di fine-tuning, in questo modo le features generiche della rete diventano piano piano sempre più specifiche per la nuova attività.

- La Feature extraction di un modello pre-allenato è una tecnica che consiste nel rimuovere l'ultimo layer (fully-connected) che è specifico per il task e utilizzare il resto della rete come feature extraction su un nuovo set di dati su cui lavorare.
- Il Fine-tuning è un processo di ottimizzazione dei pesi della rete pre-addestrata.



Dataset

Il dataset utilizzato per la procedura di training dei modelli è stato fornito da Xenia progetti s.r.l, azienda con cui si è collaborato per la realizzazione del progetto.

Il dataset presenta un totale 5252 immagini di dimensione 1920 x 1080 in formato png. Le immagini sono state acquisite da un punto di acquisizione fisso della scena, dunque si possiede un unico punto di vista della scena inquadrata. Si mostra nella figura seguente un sample delle immagini presenti:

Figura 3 - immagine di esempio dataset

Insieme al dataset di immagini, sono state fornite le corrispondenti annotazioni dei bounding box in formato Yolo delle etichette associate alle immagini: “people” e “helmet”, rendendo quindi non necessaria una procedura di etichettatura delle immagini.

Per motivi di compatibilità con gli strumenti utilizzati per il training si è scelto di convertire le immagini in formato jpeg e le annotazioni in formato Pascal Voc.

Quest’ultima conversione in particolare ha richiesto la costruzione di un file csv contenente le colonne: filename, class, width, height, xmin, ymin, xmax, ymax. Di cui riportiamo un esempio:

	filename	class	width	height	xmin	ymin	xmax	ymax
0	2019-11-14 14-00-00~15-00-02_84575.jpg	helmet	1920	1080	423	240	441	260
1	2019-11-14 14-00-00~15-00-02_84575.jpg	people	1920	1080	418	237	457	377
2	2019-11-14 14-00-00~15-00-02_84575.jpg	helmet	1920	1080	312	257	339	280
3	2019-11-14 14-00-00~15-00-02_84575.jpg	people	1920	1080	318	256	371	392
4	2019-11-13 14-00-01~15-00-03_06399.jpg	helmet	1920	1080	764	389	805	428
...	...	...	...	...	...	...	...	...
12881	2019-11-14 12-07-30~13-00-02_09439.jpg	helmet	1920	1080	851	360	885	383
12882	2019-11-13 15-00-03~16-00-02_88863.jpg	people	1920	1080	1198	436	1286	578
12883	2019-11-13 15-00-03~16-00-02_88863.jpg	helmet	1920	1080	1235	443	1285	476
12884	2019-11-13 15-00-03~16-00-02_88863.jpg	people	1920	1080	1215	473	1327	824
12885	2019-11-13 15-00-03~16-00-02_88863.jpg	helmet	1920	1080	1258	482	1320	541

Tabella 2 - file csv annotazioni

Queste annotazioni in formato csv sono infine state convertite in formato Pascal voc.

Il dataset è stato successivamente suddiviso in training, validation e test seguendo una ripartizione 80 – 10 – 10 (80% training, 10% validation, 10% test), ottenendo dunque i seguenti set:

- Training set: 4201 immagini
- Validation set: 525 immagini
- Test set: 525 immagini

Training

Per eseguire la procedura di training ci si è serviti di Google Colab, uno strumento gratuito presente nella suite Google che consente di scrivere codice python direttamente dal proprio browser.

Una piattaforma online che offre un servizio di cloud hosting per notebook Jupyter dove creare ricchi documenti che contengono righe di codice, grafici, Si può creare un documento Google Colab direttamente da Google Drive e, proprio come un qualunque documento in G Suite, può essere condiviso con altri utenti che hanno la possibilità di modificarlo e lasciare commenti direttamente nel notebook.

Il notebook Jupiter verrà poi eseguito su macchine virtuali di server Google. Ciò consente di svincolarsi dalla parte hardware e di concentrarci solamente sul codice Python e sui contenuti che si vuole integrare nel notebook.

Si allegano i link ai due notebook contenenti i training delle due reti:

- EfficientDet_V0: https://drive.google.com/drive/folders/1-LpDEISVmBuJzFc2Z8HOAKAUgxI41Wfc?usp=sharing
- EfficientDet_V4: https://drive.google.com/drive/folders/1hYzs8qLmZP6MriSTzm4MmufG08-dOgUc?usp=sharing

Demo

Per la realizzazione di questo progetto è stata sviluppata anche una applicazione mobile per dispositivi Android in grado di testare i modelli tflite generati nella precedente fase di training.

Tramite l'apposita interfaccia è possibile selezionare il modello che si desidera utilizzare per la detection.

L'applicazione è stata implementata utilizzando la libreria TensorFlow Lite Lib che prevede l'utilizzo di un interprete per i file *.tflite*. Tramite l'interprete è possibile eseguire le inferenze su un crop effettuato sul frame prelevato dalla camera. Inoltre tramite l'interfaccia è possibile definire il numero di thread su cui suddividere le operazioni degli operatori dei modelli tflite.

Il link al repository GitHub per l'applicazione Android è inserito di seguito:

https://github.com/mrjoelc/AndroidObjectDetectionTFLite

Confronto risultati

Come riportato nella tabella seguente delle performance ufficiali delle due reti, si osserva che la rete EfficientDet-LiteV0 risulta essere molto più veloce e leggera, presentando una latenza assai inferiore rispetto alla EfficientDet-LiteV4 e una media di precisione del 25%. Tuttavia, EfficientDet-LiteV4 ha una precision quasi doppia rispetto ad EfficientDet-LiteV0.

Model architecture	**Size(MB)***	**Latency(ms)****	**Average Precision*****
EfficientDet-Lite0	4.4	37	25.69%
EfficientDet-Lite4	19.9	260	41.96%

Valutazione modello TensorFlow VS TensorFlow Lite su Test Set

EfficientDet-LiteV0

```
1 model.evaluate(test_data)
```

```
9/9 [==============================] - 52s 4s/step

{'AP': 0.31709027,
 'AP50': 0.6183029,
 'AP75': 0.3169516,
 'AP_/helmet': 0.07095062,
 'AP_/people': 0.5632299,
 'APl': 0.34202254,
 'APm': 0.18775134,
 'APs': 0.016068738,
 'ARl': 0.4107843,
 'ARm': 0.41395643,
 'ARmax1': 0.24342301,
 'ARmax10': 0.44377357,
 'ARmax100': 0.47029784,
 'ARs': 0.08429054}
```

```
1 model.evaluate_tflite('/content/drive/MyDrive/Dataset/EfficentDetV0/model.tflite', test_data)
```

```
525/525 [==============================] - 1441s 3s/step

{'AP': 0.30599794,
 'AP50': 0.6094584,
 'AP75': 0.3006431,
 'AP_/helmet': 0.06363064,
 'AP_/people': 0.5483652,
 'APl': 0.37255904,
 'APm': 0.17299683,
 'APs': 0.014809396,
 'ARl': 0.41391402,
 'ARm': 0.32654127,
 'ARmax1': 0.24172142,
 'ARmax10': 0.40621063,
 'ARmax100': 0.40978768,
 'ARs': 0.06570946}
```

EfficientDet-LiteV4

```
1 model.evaluate(test_data)
```

```
9/9 [==============================] - 58s 5s/step

{'AP': 0.34340948,
 'AP50': 0.63888747,
 'AP75': 0.35492224,
 'AP_/helmet': 0.08475102,
 'AP_/people': 0.60206795,
 'APl': 0.34963137,
 'APm': 0.27784818,
 'APs': 0.015413367,
 'ARl': 0.52096534,
 'ARm': 0.49967477,
 'ARmax1': 0.26149145,
 'ARmax10': 0.48691297,
 'ARmax100': 0.5191359,
 'ARs': 0.09932432}
```

```
1 model.evaluate_tflite('/content/drive/MyDrive/Dataset/EfficentDetV4/model.tflite', test_data)
```

```
525/525 [==============================] - 25651s 49s/step

{'AP': 0.33514616,
 'AP50': 0.6322582,
 'AP75': 0.3408327,
 'AP_/helmet': 0.077582374,
 'AP_/people': 0.5927099,
 'APl': 0.39870495,
 'APm': 0.263188,
 'APs': 0.014644179,
 'ARl': 0.44894418,
 'ARm': 0.43270645,
 'ARmax1': 0.2572161,
 'ARmax10': 0.4564803,
 'ARmax100': 0.46417493,
 'ARs': 0.07652027}
```

Tuttavia, nel caso reale, si è notato un comportamento differente: infatti, le reti sembrano effettuare detection molto simili tra loro sia in termini di bounding box, label ed accuracy; rimane invariato il fattore latenza che risulta decisamente superiore nel modello V4, come da teoria, dovuto ad una complessità e profondità maggiore della rete rispetto alla sua versione base V0.

Interessante anche come il numero di thread sia significativo al fine di avere una latenza più bassa possibile: infatti, si è notato che nel caso di utilizzo del modello EfficientDet-LiteV0 si abbiano migliori risultati (sempre in termine di latenza, poiché in termini di *precision* la situazione rimane analoga) quando il numero di thread, per eseguire le operazioni, è fissato su quattro. Mentre, EfficientDet-LiteV4 riesce ad effettuare inferenze con tempo di latenza ridotto quando il numero di thread è settato a tre.

Uno sviluppo sperimentale futuro potrebbe senz'altro essere lo studio più approfondito delle reti per analizzare quali sono i dettagli che permettono di avere tempi di latenza differenti al cambio del numero di thread in corso.