UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Elaborazione di Segnali Multimediali**

Deep learning per la segmentazione semantica

Anno Accademico 2020/2021

Candidato:

Christian Marescalco

matr. N46005063

Indice

Introduzione

L’intelligenza artificiale è la disciplina che si occupa di realizzare macchine in grado di pensare autonomamente e che possano risolvere problemi con il ragionamento. Alan Turing fu il primo a gettare le basi dell’intelligenza artificiale, tramite il suo omonimo test che consisteva nel riconoscere la presenza di una macchina intelligente.

L’apprendimento automatico gioca un ruolo chiave nell’intelligenza delle macchine ed è proprio tramite questo che l’automa impara a svolgere dei compiti diventando dunque intelligente. Uno degli ambiti più popolari nell’utilizzo dell’apprendimento è quello della Computer Vision, che si occupa di fornire alla macchina una comprensione delle immagini e video digitali, così come accade per gli esseri umani che sono in grado di fornire una descrizione completa degli oggetti e delle loro caratteristiche all’interno delle immagini o video. Per questo motivo la Computer Vision è ritenuta una branca dell’Intelligenza Artificiale, che cerca di replicare l’intelligenza visiva umana. Le applicazioni di questa disciplina sono molte e sono tutte orientate per avere un grosso impatto in un futuro automatizzato e sviluppato tecnologicamente, come ad esempio la guida autonoma di veicoli, l’analisi di immagini biomediche o la realtà aumentata.

I task più frequenti nella Computer Vision spaziano dalla classificazione delle immagini fino al riconoscimento ed al tracking degli oggetti all’interno di un video. La tecnica alla base della comprensione di scene è la segmentazione semantica, ossia il processo di assegnazione ad ogni pixel dell’immagine di una classe. Si effettua prima una

segmentazione a grana grossa, quindi considerando regioni di una stessa classe con la stessa etichetta, per poi effettuare quella che viene chiamata *instance segmentation*, cioè una segmentazione a grana più fine, assegnando etichette diverse ai diversi oggetti di una stessa classe.

Lo scopo di questa tesi è quello di fornire una panoramica sulle tecniche di Deep Learning per la Computer Vision con un focus sulla segmentazione semantica, realizzando poi un caso di studio riguardante la segmentazione semantica di scene urbane per la guida autonoma utilizzando le tecniche precedentemente descritte. Per la risoluzione del caso di studio è stato utilizzato il dataset Mapillary Vistas e un'architettura di rete adattata al problema della segmentazione.

Capitolo 1: Deep Learning per la Computer Vision

1.1 Introduzione al Deep Learning

Il Deep Learning, in italiano apprendimento profondo, è una sottocategoria del Machine Learning che fa uso di *reti neurali profonde*, ossia utilizza modelli di apprendimento su più livelli. Secondo la definizione dell'Osservatorio Artificial Intelligence del Politecnico di Milano, per apprendimento profondo si intende « *un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa* ».

A partire da un insieme di dati rappresentativi di un dominio applicativo, l'algoritmo di Deep Learning (o più in generale di Machine Learning) apprende come effettuare un particolare compito basandosi sull'esperienza accumulata.

Uno degli impieghi principali del Deep Learning è quello della classificazione di dati tramite **l'estrapolazione automatica** delle caratteristiche salienti e discriminative di essi, anche dette **feature**, a partire da un set di dati di addestramento. Come detto precedentemente le reti neurali artificiali presentano delle architetture a strati, in particolare i vari livelli in base alla loro posizione nell'architettura permettono di estrarre feature di diverso livello. Gli strati più in profondità estraggono feature di più alto livello, cioè più vicine alla classificazione, mentre gli strati più superficiali estraggono caratteristiche di basso livello.

La differenza tra il Machine Learning e il Deep Learning giace proprio nell'estrapolazione delle caratteristiche di basso livello: nel primo caso è il programmatore ad effettuare l'estrazione e la scelta delle caratteristiche, mentre nel secondo caso è la rete stessa.

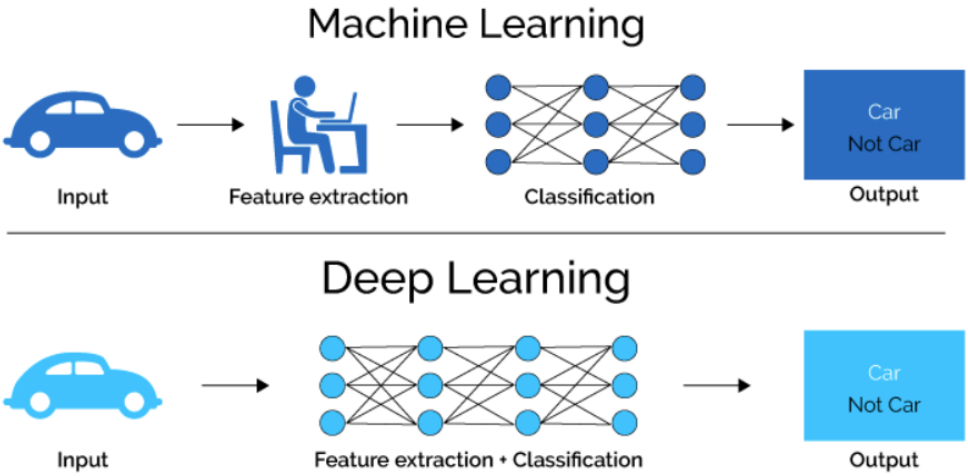

Figura 1.1 Differenza tra Machine Learning e Deep Learning [4]

Per gli algoritmi di Deep Learning sono dunque necessarie moli di dati più grandi rispetto a quelle relative agli algoritmi di Machine Learning, andando però a migliorare le performance, poiché la rete potrebbe estrarre delle caratteristiche migliori rispetto a quelle scelte dall'essere umano.

1.1.1 Le reti neurali artificiali

Ispirandosi al cervello umano ed alle reti neurali biologiche sono state modellate le reti neurali artificiali, alla base di esse vi è il concetto di **neurone artificiale**, il quale rappresenta l'unità computazionale della rete neurale artificiale ed è descritto da una funzione matematica che applica una trasformazione non lineare ai contributi pesati dei vari ingressi. Per il neurone *k*-esimo, la funzione è del tipo:

$$y_k = f_k(\sum_i w_{ki} \cdot x_{ki} + b_k))$$

Dove i indica i vari neuroni precedenti, x_{ki} sono gli ingressi relativi al k-esimo neurone, w_{ki} è il peso della connessione tra i neuroni k ed i, b_k è il bias, il quale permette di migliorare l'apprendimento. La funzione f è detta **di attivazione**, generalmente è non lineare, e permette di attivare il neurone corrispondente in base agli ingressi ricevuti. Tra le più importanti funzioni di attivazione troviamo:

- la funzione gradino, la quale fornisce in uscita un output binario in base ad una

soglia fissata, non viene spesso utilizzata poichè non è differenziabile ed è molto sensibile a variazioni minime;

- la funzione sigmoide o logistica, la quale è differenziabile e dunque permette di ottimizzare i pesi, ma presenta il problema dei *vanishing gradients*, cioè nella fase di apprendimento l'ottimizzazione dei pesi diminuisce fino a stabilizzarsi in valori che non sono necessariamente i migliori;
- la funzione ReLU, supera i problemi delle precedenti ed è la più efficiente computazionalmente;

In figura 1.1.1 è mostrata la rete neurale più semplice, ossia quella composta da un solo neurone ed è riconducibile al modello di *percettrone*.

L'interconnessione di più neuroni permette di creare delle reti in cui vi sono: uno strato di input, degli strati nascosti ed uno strato di output. In figura 1.1.2 è mostrato un esempio di rete neurale *feed-forward*, cioè una rete in cui l'informazione si muove lungo un'unica direzione senza creare dei cicli.

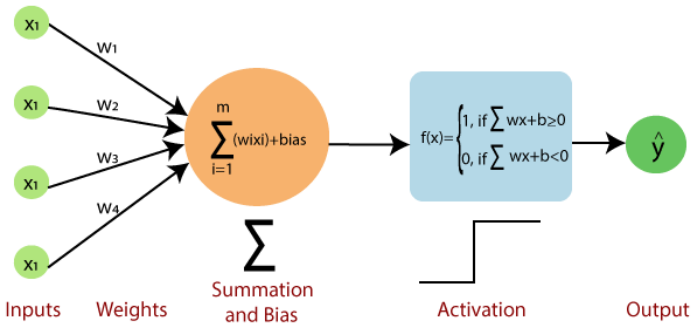

Figura 1.1.1 Esempio di percettrone [5]

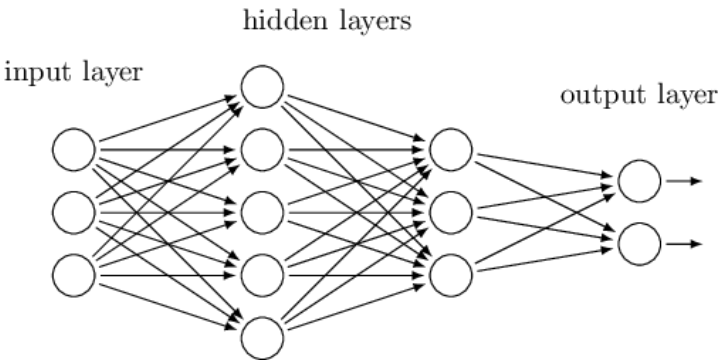

Figura 1.1.2 Esempio di rete neurale [6]

1.1.2 Addestramento di una rete neurale

Dopo aver progettato la rete neurale in base al tipo di problema, questa è pronta per effettuare l'addestramento. Vengono prima inizializzati i pesi (ad esempio in modo casuale), e poi inizia il training.

Esistono due tipi di apprendimento: **supervisionato** e non **supervisionato**. Nel primo caso si parla di classificazione dei dati e l'insieme di dati per l'addestramento è già etichettato. Nel secondo caso, invece, il set di dati non è etichettato e si parla di analisi di pattern. Ci si focalizzerà sul primo, in quanto è quello utilizzato nelle applicazioni che si affronteranno.

Nell'apprendimento supervisionato vengono distinte due fasi principali, le quali si ripetono per l'intero dataset di training, ottenendo una rete con dei pesi regolati rispetto ai dati in input. La prima fase è detta di *forward propagation*, in cui i dati di input attraversano l'intera rete, in modo tale che ogni neurone applichi la trasformazione all'informazione, che riceve dagli strati precedenti, e invii quella ottenuta allo strato successivo. Quando lo strato finale viene raggiunto, si ottiene la predizione per l'input inserito. La rete, successivamente, effettua la comparazione tra gli output ottenuti e gli output desiderati. Gli errori che si ottengono dal confronto vengono stimati tramite la **funzione di loss**, la quale misura quanto le predizioni sono ottime o meno in relazione al risultato effettivo.

Dopo che la funzione di loss è stata calcolata, inizia la fase di *backward propagation*. A partire dal layer di output, l'informazione riguardante l'errore viene propagata a tutti i neuroni nello strato precedente a quello di output, che a loro volta la propagano a tutti i neuroni precedenti e cosi via. Questi vengono aggiornati in base al loro contributo rispetto all'uscita ottenuta. L'aggiornamento dei pesi è un problema di ottimizzazione, cioè bisogna trovare i pesi tali che il valore della funzione di loss sia minimo e dunque la predizione sia il più vicino possibile a quella reale. Un metodo utilizzato spesso è quello della *discesa del gradiente,* in cui vengono calcolate le derivate parziali della funzione di loss rispetto ai pesi della rete, utilizzando la regola della catena. In questo modo se l'errore decresce all'aumentare del peso allora il peso aumenta, se invece l'errore aumenta all'aumentare del peso, allora questo descresce. In entrambi i casi, l'incremento del peso è riscalato da un fattore empirico, detto **learning rate**.

1.1.3 Le reti neurali convoluzionali (CNN)

Le reti neurali convoluzionali (in breve CNN) sono una classe di reti neurali artificiali *feed-forward*. Caratterizzate dall'uso dell'operazione di convoluzione, le CNN vengono utilizzate per l'elaborazione di immagini. Analogamente ad una rete neurale tradizionale, una CNN possiede neuroni con pesi e bias. Tuttavia, tramite l'utilizzo di strati convoluzionali, si riesce a ridurre il numero di parametri da allenare nei singoli strati, utilizzando gli stessi pesi e bias per tutti neuroni di un determinato strato. Questa strategia permette di rendere la rete invariante rispetto alle traslazioni, quindi tutti i neuroni di uno strato rilevano la stessa feature ma in aree dell'immagine diverse.

Uno strato convoluzionale non è altro che un filtro a finestra mobile che scansiona l'intera immagine suddividendola in regioni più piccole, cercando degli specifici pattern caratteristici rappresentati dal *kernel* della convoluzione dello strato. In uscita allo strato si ottiene una feature map, la quale si focalizza solo sulle regioni di interesse per quelle determinate caratteristiche.

Un'architettura completa di CNN prevede più strati convoluzionali in sequenza. Per aumentare l'efficienza computazionale, tra gli strati convoluzionali si introducono strati di *pooling* che riducono la risoluzione della mappa in ingresso, in questo modo l'informazione relativa alle caratteristiche viene compressa. Ottenuta la feature map ridotta, questa viene serializzata in un vettore e data in ingresso ad una serie di strati *fully-connected* come nelle reti neurali classiche, ottenendo in uscita le predizioni della rete.

Le CNN vengono spesso utilizzate nell'analisi di immagini, le applicazioni riguardano soprattutto la risoluzione di problemi per la Computer Vision, dal riconoscimento di immagini alla segmentazione semantica, fino alla comprensione di scene tridimensionali.

Capitolo 2: Segmentazione semantica

2.1 Il problema e la sua importanza

La segmentazione è una tecnica di Computer Vision di suddivisione di immagini in sezioni caratterizzate da simili proprietà visuali. La segmentazione semantica, nello specifico, consiste nell'assegnazione di un'etichetta ad ogni pixel di un'immagine. Essa trova impiego in settori dove è necessario semplificare la rappresentazione dell'immagine in qualcosa più facile da analizzare, come ad esempio: guida autonoma, realtà aumentata, immagini satellitari e visione robotica. Date le ottime prestazioni, la segmentazione semantica è attualmente condotta tramite le reti neurali convoluzionali (CNN).

La segmentazione semantica è principalmente utilizzata per localizzare con precisione i bordi e gli oggetti all'interno di un'immagine, questa rappresenta un task intermedio fondamentale verso la risoluzione di problemi più complessi nella visione computerizzata, come lo *scene understanding* o la ricostruzione 3D di una scena.

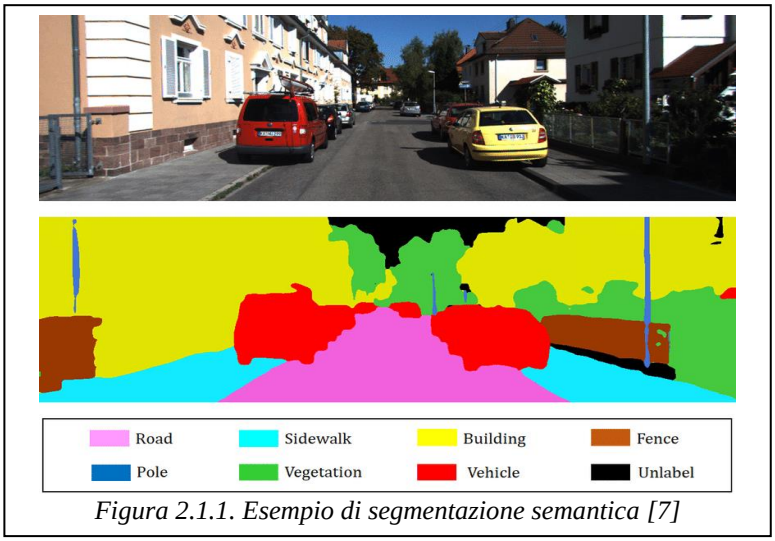

Figura 2.1.1. Esempio di segmentazione semantica [7]

2.1.1 Segmentazione semantica,d'istanza e panoptica

Esistono diverse tipologie di segmentazione in base alla granularità delle etichette:

- **Semantica**, oggetti diversi, ma appartenenti ad una stessa classe di oggetti, vengono classificati con la stessa etichetta, viene anche detta segmentazione *a grana grossa*.
- **D'istanza**, istanze di oggetti vengono classificati con etichette differenti, viene anche detta *segmentazione a grana fine*.
- **Panoptica**, effettua la combinazione delle due tecniche precedenti, in questo modo a tutti i pixel è assegnata una classe e le varie istanze di oggetti presentano etichette univoche, ottenendo una visualizzazione globale sia per categoria che per istanza.

La segmentazione d'istanza risulta essere molto più complessa da realizzare rispetto a quella semantica, infatti nel secondo caso oggetti di una stessa classe che occorrono vicini tra loro verranno etichettati unicamente, mentre nel primo caso è necessario fare attenzione ai bordi dei singoli oggetti [2]. Inoltre le istanze variano sia in grandezza che in numero all'interno delle immagini, ciò rende il compito ancora più difficile da realizzare. Ad esempio, nelle scene di guida autonoma è spesso presente una vista ampia, la quale rende le varie istanze più piccole e dunque più difficili da rilevare.

Si noti che il problema della segmentazione è differente da quello del rilevamento di oggetti (*object detection)*, nella seconda tecnica i bordi degli oggetti non sono rilevati esattamente ma vengono individuate solo dei riquadri che delimitano gli oggetti, dette *bounding box.*

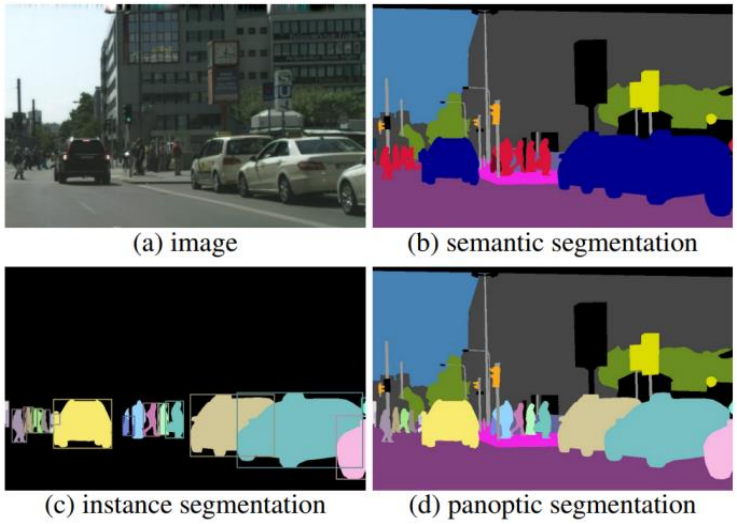

Figura 3. Differenze tra le varie tipologie di segmentazione [2]

2.2 Tecniche per la segmentazione semantica

Tipicamente la segmentazione viene effettuata tramite un classificatore che opera con un approccio a finestra mobile ed utilizzando immagini a grandezza fissata, come mostrato in figura 2.2.1:

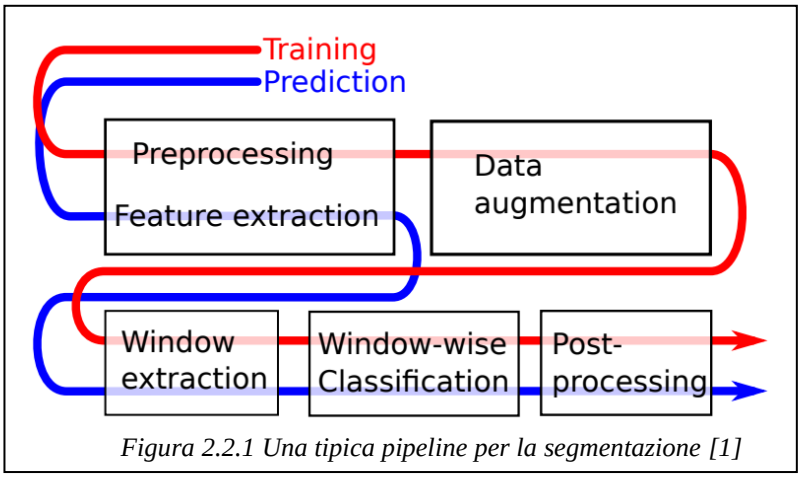

Figura 2.2.1 Una tipica pipeline per la segmentazione [1]

Sia per la fase di training che per la fase di prediction, vengono effettuate delle tecniche di pre-processing sulle immagini, come ad esempio rescaling (per rendere le immagini tutte alla stessa grandezza fissata), e tecniche di estrazione di feature di basso livello, come HOG (*Histogram of oriented Gradients*). Successivamente possono essere applicati metodi di data augmentation per le immagini di training (rotazione, *cropping*, *resizing*). Per ogni immagine, in entrambe le fasi, vengono poi estratte delle finestre che verranno date in ingresso al classificatore, il quale assegna ad esse una classe. Nella fase di post-processing vengono applicate operazioni morfologiche per rifinire la segmentazione semantica, eliminando il rumore e rifinendo i bordi [1].

Tra gli approcci per la segmentazione semantica possiamo effettuare una distinzione tra quelli tradizionali e quelli che si basano sulle reti neurali convoluzionali. Le tecniche tradizionali si basano sul dominio d'interesse del problema e, dunque, sulle varie tipologie d'immagini del dominio. Per queste tecniche è fondamentale la scelta delle feature da estrarre. Le tecniche basate sulle CNN, invece, riescono ad applicare in modo efficiente l'approccio a finestra scorrevole utilizzando una rete allenata come una convoluzione sull'intera immagine, per questo motivo hanno riscontrato molto successo rispetto agli

approcci tradizionali.

I più importanti approcci tradizionali sono:

- *Markov Random Fields(MRF),*sono dei modelli a grafo probabilistici che sfruttano una stima del massimo a posteriori (MAP) per l'ottimizzazione della funzione obiettivo. L'idea di base è di assegnare una variabile aleatoria per ogni *feature* ed una per ogni pixel che verrà etichettato [1]. Le dipendenze condizionali tra le variabili aleatorie vengono rappresentate come lati di un grafo. L'utilizzo del contesto è fondamentale per catalogare correttamente un'entità ed è espresso come probabilità condizionata rispetto ai pixel vicini a quello d'interesse.

  Ad ogni pixel si associa una variabile aleatoria Y che rappresenta la classe del pixel, con dominio {1,.., num. di classi}, ed una variabile aleatoria X che rappresenta il valore del pixel, con dominio {0,…,255} o [0,1]. La probabilità della coppia (X,Y) si esprime come:

  $$P(\mathbf{x},\mathbf{y}) = \frac{1}{Z}e^{-E(\mathbf{x},\mathbf{y})}$$

  Dove *Z* è un termine di normalizzazione, detto *partition function*, della funzione *E*, detta *funzione di energia.* Per poter stabilire la soluzione ottima, ossia quella a probabilità massima, come anticipato si utilizza una stima del massimo a posteriori (MAP) della distribuzione di probabilità *P*. E' possibile dimostrare che il massimo della funzione *P*, ossia la moda, coincide con il minimo della funzione d'energia, dunque è possibile effettuare un *MAP energetico* [8].

- *Conditional Random Fields (CRF),* sono dei MRF in cui invece di apprendere la distribuzione *P*(Y,X), si apprende la distribuzione *P*(Y|X). Il vantaggio di questo approccio è che non è necessario stimare la distribuzione di X e non bisogna fare presupposizioni su di essa [1]. In questo caso la partition function *Z* e la probabilità condizionata *P* sono pari a:

  $$Z(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{x},\mathbf{y}) \qquad P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\prod_{c\in\mathcal{C}}\psi_c(\mathbf{y}_c|\mathbf{x})$$

  Dove i *ψc* sono detti *clique potentials,* mentre l'insieme C contiene tutti i *clique* del modello a grafo.

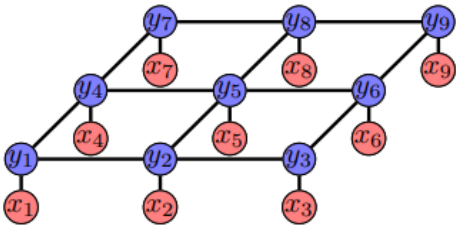

Figura 2.2.2 Esempio di CRF ($\boldsymbol{x_i}$ è il valore del pixel, $\boldsymbol{y_i}$ è l'etichetta) [1]

Tradizionalmente, per compensare alle limitazioni dei CRF e dei MRF, sono stati introdotti approcci più efficienti come quello superpixel-based: che effettua una segmentazione su pixel più grandi (superpixel), ma permette di sfruttare meglio le informazioni del contesto spaziale tramite potenziali di ordine superiore (molto più complessi da realizzare con approcci pixel-based) ed usando opportune funzioni di potenziale a coppie per caratterizzare meglio le dipendenze tra i superpixel lungo 4 direzioni vicine [9].

Un altro approccio introdotto al fine di ottimizzare quello dei CRF è stato proposto da Krahenbuhl e Koltun ed è quello dei Fully-connected Conditional Random Fields (FCRFs), il quale implementa le funzioni di potenziali a coppie tra tutte le coppie di pixel nell'immagine [2].

2.3 Reti neurali per la segmentazione semantica

Il successo delle reti neurali convoluzionali (CNN) nel campo della classificazione d'immagini e del rilevamento di oggetti ha portato a sfruttare il potenziale di tali reti anche in task pixel-based, come la segmentazione semantica. A differenza dei problemi precedenti, per la segmentazione è necessaria una **predizione pixelwise**, ciò rende gli ultimi strati dell'architettura della rete differenti rispetto ai problemi di classificazione.

Uno dei primi approcci che applica la CNN alla segmentazione d'immagini è l'adattamento di un classificatore alla segmentazione ed è mostrato in figura 2.3.1, gli strati densi della rete vengono trasformati in strati convoluzionali, permettendo alla rete di fornire in uscita una mappa di segmentazione, la quale ha una risoluzione inferiore all'immagine d'ingresso.

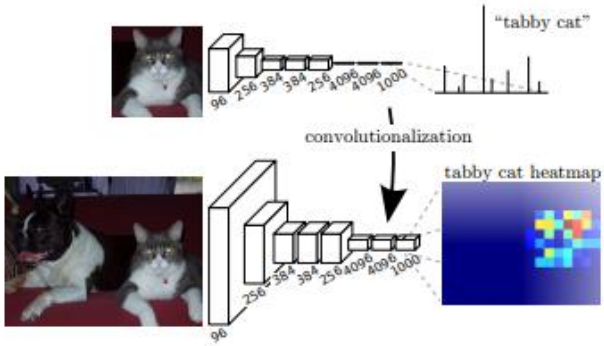

Figura 2.3.1 Esempio di classificatore adattato alla segmentazione [3]

Le reti neurali convoluzionali moderne sfruttano strati di pooling e di sub-sampling che diminuiscono la risoluzione, come mostrato in figura 2.3.1, questo da un lato aumenta l'efficienza computazionale e dall'altro rende più grossolana la mappa di segmentazione. La segmentazione semantica richiede predizioni full-resolution, ossia *predizioni dense*, per questo motivo sono stati introdotti diversi approcci moderni che adattano ed ottimizzano le reti al problema della segmentazione semantica.

Un'architettura di rete comune a questo problema è quella **encoder-decoder**. La prima parte della rete, cioè l'encoder, corrisponde ad una tipica rete neurale convoluzionale privata degli ultimi strati densi, qui viene effettuata la fase di *downsampling* dell'immagine di input, sviluppando delle mappe di feature a bassa risoluzione che discriminano in modo efficiente tra le classi. Nella seconda parte di rete si ha la fase di *upsampling* delle mappe di feature a bassa risoluzione in immagini segmentate ad alta risoluzione. Un esempio di architettura encoder-decoder è mostrato in figura 2.3.2:

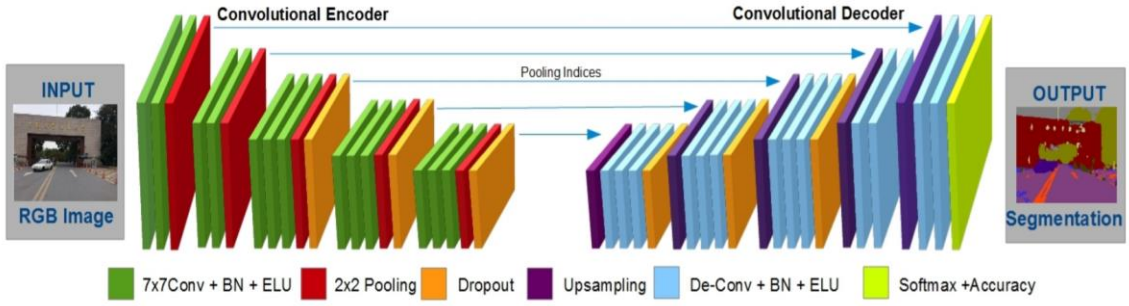

Figura 2.3.2 Esempio di architettura encoder-decoder per la segmentazione semantica [10]

2.3.1 Fully Convolutional Networks (FCNs)

Una Fully Convolutional Network (FCN) è una rete che non contiene nessuno strato denso. A differenza delle CNN tradizionali, una FCN presenta strati convoluzionali 1x1, che sostituiscono quelli densi e trasformano il numero di canali in numero di classi.

J.Long in [3] fu il primo a sviluppare una rete completamente convoluzionale, che viene allenata *end-to-end* per la segmentazione. La rete prende in ingresso un'immagine dalla dimensione arbitraria e produce in uscita una mappa di segmentazione della stessa dimensione.

La progettazione della rete ha inizio con la scelta della rete di backbone a partire da architetture ben note (AlexNet, VGG16, GoogLeNet), andando a rimpiazzare gli strati fully-connected con strati convoluzionali. Per poter creare in uscita un'immagine della stessa dimensione dell'ingresso è necessario effettuare un sovracampionamento, in inglese *upsampling*, tramite interpolazione. La tecnica adottata per le FCN è la **deconvoluzione**, la quale inverte i passi di *forward* e *backward* della convoluzione, per tale motivo risulta essere semplice da realizzare. Inoltre i pesi del filtro di deconvoluzione non devono essere necessariamente fissati ma possono essere appresi, ottenendo un'interpolazione non lineare, per questo motivo la deconvoluzione ha riscontrato molta più popolarità nelle applicazioni rispetto a tecniche non apprensive (*Max Unpooling*, *Nearest Neighbor*).

Per descrivere il funzionamento della deconvoluzione è più semplice sfruttare come esempio un caso monodimensionale in figura 2.3.3. Si nota che ogni valore della finestra di input viene moltiplicato per tutti i pesi della finestra di deconvoluzione, nei punti in cui si ha sovrapposizione tra le proiezioni dei valori si effettua la somma, ottenendo in uscita una finestra di dimensione superiore.

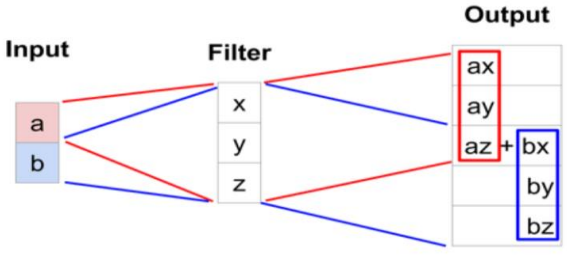

Figura 2.3.3 Caso monodimensionale della deconvoluzione [11]

Per poter realizzare la deconvoluzione basta effettuare l'operazione trasposta alla convoluzione, per questa ragione la deconvoluzione è anche definita **convoluzione trasposta**.

Nel caso bidimensionale per poter definire la convoluzione trasposta è necessario prima:

- distribuire la maschera del filtro, ad esempio 3x3, in modo tale che ogni riga definisca l'operazione di convoluzione (vedi fig. 2.3.4) ed effettuarne la trasposta;
- serializzare la matrice in input, ad esempio 2x2 in 4x1;

Si effettua poi il prodotto matriciale tra la matrice trasposta di convoluzione ed il vettore di input. In uscita si ottiene un vettore 16x1, che tramite opportuno reshape viene trasformato in una matrice 4x4, ottenendo un sovracampionamento. In figura 2.3.4 è mostrato un esempio.

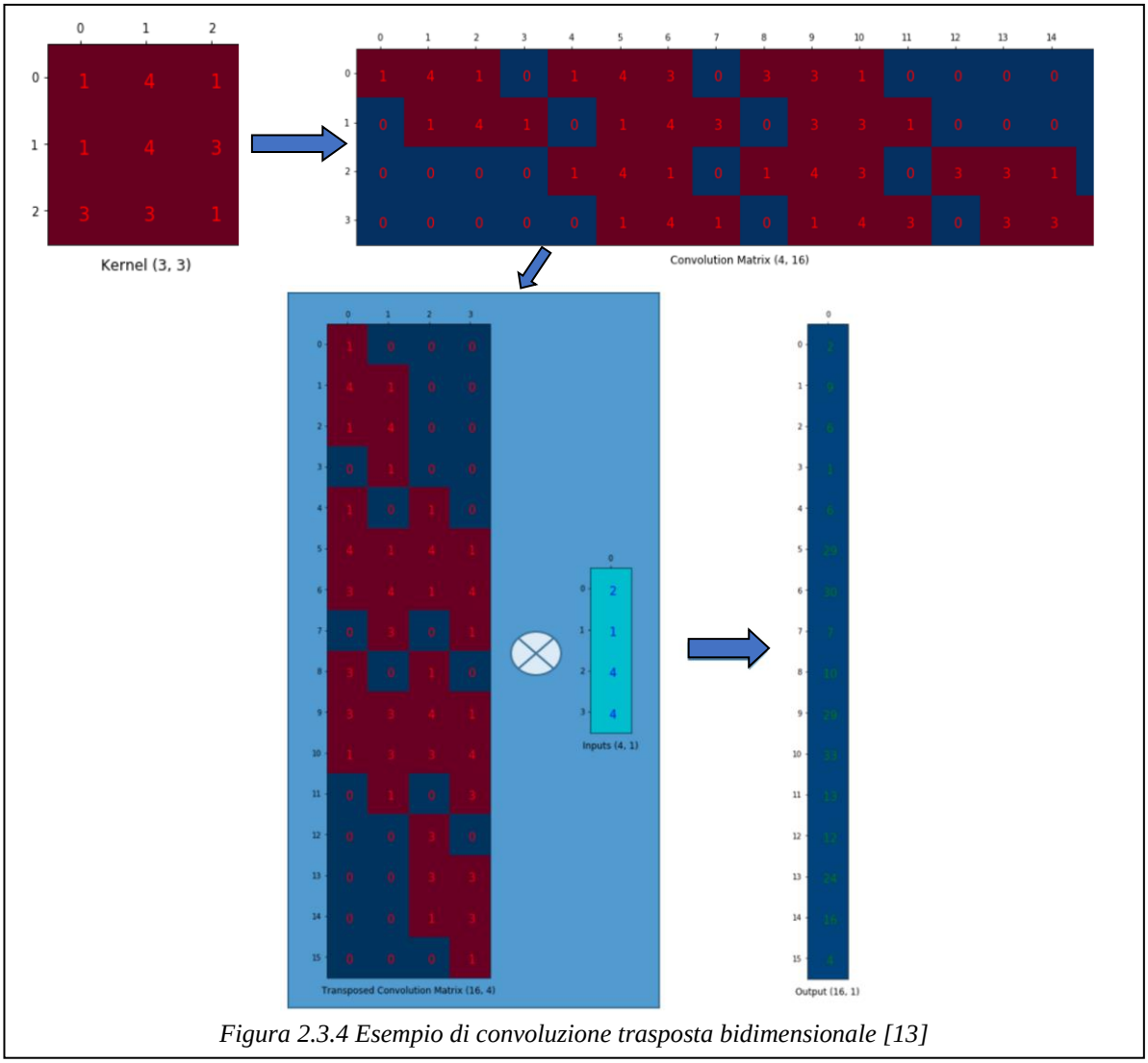

Figura 2.3.4 Esempio di convoluzione trasposta bidimensionale [13]

Per poter ottenere la stessa dimensione dell'immagine in ingresso si può effettuare un upsampling, tramite deconvoluzione, della feature map in uscita all'encoder, così come accade per FCN-32, la quale produce in uscita una predizione tramite fattore di sovracampionamento pari a 32. In quest'ultima architettura, mostrata in figura 2.3.6, la mappa che si ottiene in uscita è grezza, ciò è dovuto al fatto che *le feature più profonde possono essere ottenute solo in profondità della rete, ma negli strati più profondi le informazioni spaziali vengono perse* [13]. La parte del decoder non riesce a produrre delle mappe di segmentazione a grana fine. Long sostiene: « *la segmentazione semantica affronta una tensione intrinseca tra la semantica e la localizzazione: le informazioni globali risolvono il cosa mentre le informazioni locali il dove* ». Come soluzione a tale problema gli autori di [3] propongono di **combinare il cosa ed il dove**, ossia strati in cui la predizione è più fine con strati in cui la predizione è più grezza ma con più informazioni spaziali, tramite l'utilizzo delle *skip connection*, ossia fondendo gli output tramite una somma pixelwise.

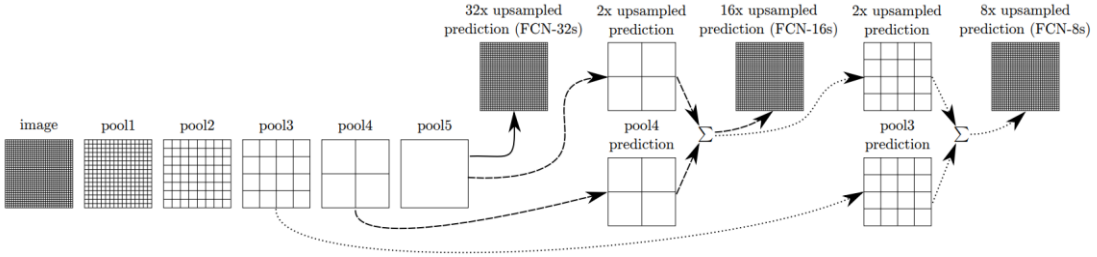

Figura 2.3.6 Architetture FCN-32s, FCN-16s, FCN-8s (solo gli strati di pooling e predizione sono esposti) [3]

Nelle architetture FCN-16s e FCN-8s, mostrate in figura 2.3.6, vengono utilizzate le skip connections, fondendo gli output delle architetture precedenti. Per la FCN-16, l'output del layer pool5 viene sovracampionato con fattore 2 e fuso con l'output del layer pool4, per poi essere sovracampionato con fattore 16, ottenendo la predizione finale. Per la FCN-8, viene effettuato l'upsampling 2x della fusione degli output per la FCN-16 ed al risultato viene fusa la predizione del pool3, dopo un sovracampionamento con fattore 8 si ottiene la predizione della FCN-8.

Effettuando l'apprendimento e la validazione sul dataset PASCAL VOC 2011 [3], le mappe di segmentazione che si ottengono dalle 3 reti sono mostrati in figura 2.3.7. Per le reti che

fanno uso delle skip connection le predizioni sono più accurate. Il miglior risultato lo si ottiene con la FCN-8. In figura 2.3.8, si nota che anche valutando le metriche di pixel accuracy, accuracy media, *mIOU*, *frequency-weighted IOU*, si ottiene che la FCN-8 è la migliore, seguita dalla FCN-16.

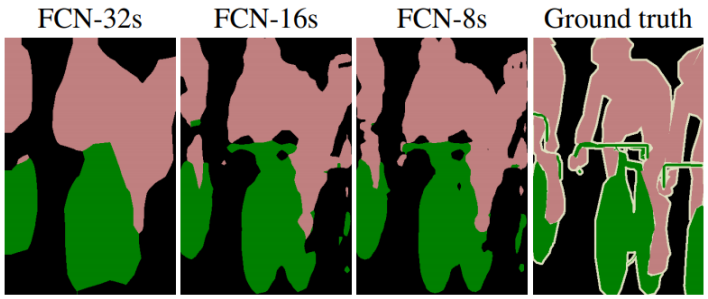

Figura 2.3.7 Comparazione tra le varie reti sul dataset PASCAL-VOC 2011 [3]

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

Figura 2.3.8 Comparazione tramite metriche delle FCN sul dataset PASCAL-VOC 2011 [3]

2.3.2 Architettura U-Net

La rete U-Net fu sviluppata da Ronneberger per la segmentazione di immagini biomediche [14], con l'obiettivo di migliorare ed estendere l'architettura fully-convolutional proposta da Long. Tale rete sfrutta anch'essa l'architettura encoder-decoder con delle opportune modifiche. La parte di encoder è una rete convoluzionale tipica composta da soli strati convoluzionali, ognuno seguito dalla funzione di attivazione ReLU e da un'operazione di max pooling. Per quanto riguarda il decoder, vi è un grande numero di canali per le feature che permettono di propagare l'informazione del contesto ai livelli a risoluzione più alta [15]. Conseguentemente, la parte di upsampling può essere o meno simmetrica a quella di downsampling, generando un'architettura a forma di U.

Dall'architettura in figura 2.3.9, si nota che sono presenti le *skip connection* tipiche delle

FCN, anche in questo caso hanno lo scopo di combinare il '*cosa*' con il '*dove*'. In questa rete, le *skip connection* concatenano l'output delle deconvoluzioni del decoder con le mappe di feature dell'encoder allo stesso livello nell'architettura ad U, ottenendo una migliore precisione spaziale delle feature. Per poter effettuare ogni concatenazione è necessario effettuare un *cropping* della feature map dal lato encoder, in modo che si abbiano le stesse dimensioni della mappa dal lato decoder. Dopo ogni concatenazione, nella parte di upsampling vengono applicate due convoluzioni che consentono di apprendere, a partire dalle informazioni spaziali e caratteristiche dell'immagine, una predizione più precisa.

Infine l'ultimo layer è uno strato convoluzionale che utilizza una o più convoluzioni 1x1, affinchè si possa far corrispondere la mappa di segmentazione ottenuta con il numero delle classi desiderato.

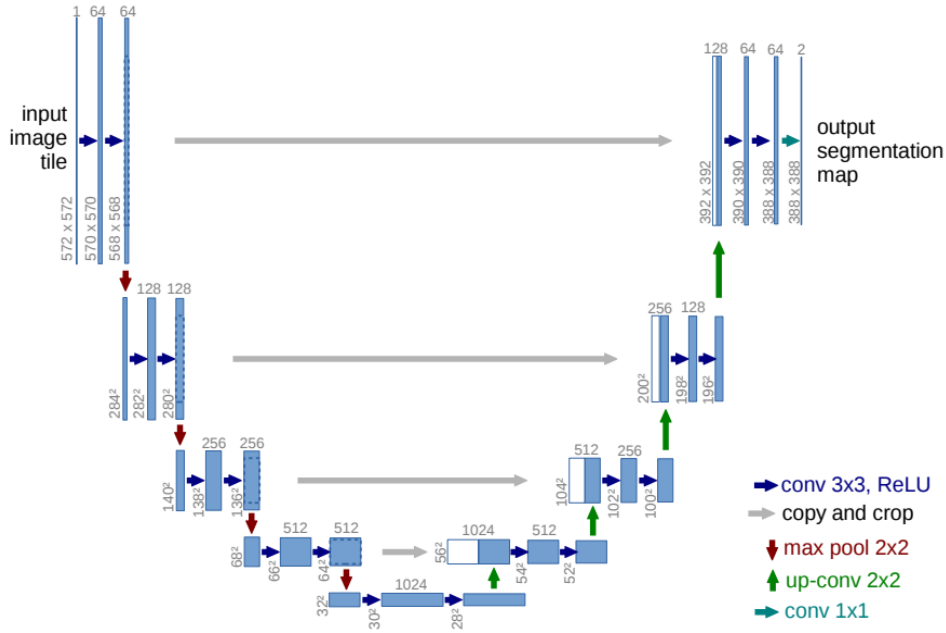

Figura 2.3.9 Architettura U-Net [15]

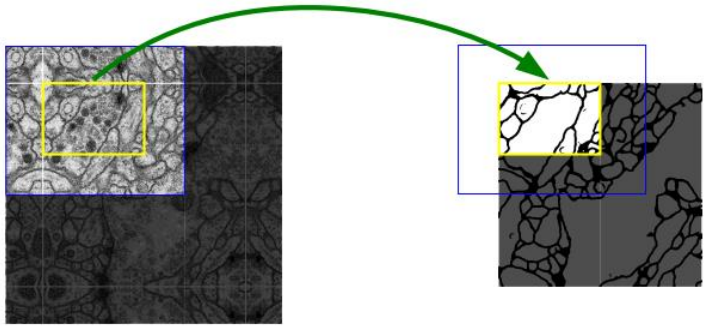

Figura 2.3.10 Esempio di strategia 'overlap-tile' della rete U-Net [14] (In giallo è mostrata la predizione,mentre in blu il contesto necessario alla predizione)

Un’altra novità introdotta nella rete, basata sulle reti fully-convolutional, è l’utilizzo di soli strati convoluzionali evitando quelli densi. Per U-Net viene utilizzata solo la parte valida di ogni convoluzione, cioè la mappa di segmentazione contiene i pixel per i quali è disponibile l’intero contesto nell’immagine di input. Per i pixel nelle regioni marginali, il contesto mancante viene estrapolato effettuando il *mirroring* dell’immagine. Questa strategia permette di applicare la rete anche a grandi immagini, andando a suddividere l’immagine in sottoinsiemi di input più piccoli in modo che siano elaborabili con una memoria GPU limitata. Pertanto, l’intera immagine viene predetta parte per parte. Tale strategia viene definita anche ‘*overlap-tile*’, in figura 2.3.10 è mostrato un esempio.

2.3.3 DeepLab

DeepLab è un modello di deep learning per la segmentazione semantica di immagini, è considerata lo stato dell’arte in questo campo. E’ stato progettato sfruttando le reti neurali convoluzionali profonde (DCNNs), le quali però non sono ideate per il problema della segmentazione semantica, per tale motivo la progettazione dei vari modelli DeepLab mira ad adattare tali architetture alla segmentazione. In tale adattamento sono 3 i problemi principali su cui ruota la progettazione: (1) ridotta risoluzione delle mappe di feature, (2) esistenza di oggetti su diversa scala, e (3) ridotta precisione spaziale dovuta alla rete DCNN che è invariante alle trasformazioni spaziali.

(1) Il primo problema è causato dai layer di downsampling e max-pooling tipici delle reti DCNN, le quali sono progettate per la classificazione d’immagini. Le architetture precedenti, cioè FCN e U-Net, risolvono tale problema aggiungendo un decoder, il quale sfrutta strati di upsampling con l’operazione di deconvoluzione o convoluzione trasposta, tuttavia ciò richiede memoria e tempo aggiuntivi. Le architetture DeepLab, invece, rimpiazzano l’operazione di downsampling degli ultimi strati di max pooling delle DCNN con un’operazione di upsampling tramite filtraggio, ossia con strati multipli convoluzionali caratterizzati da filtri sovracampionati con degli zeri. L’operazione di upsampling, utilizzata in questo caso, viene detta ***atrous convolution***, a questa segue un’operazione di interpolazione bilineare che permette di restaurare la risoluzione delle feature map alle

dimensioni dell'immagine in input. Tale combinazione di *atrous convolution* e interpolazione bilineare offre una valida e più efficiente alternativa alle tecniche precedenti che sfruttano la deconvoluzione. Inoltre, l'approccio di DeepLab converte la rete DCNN, ossia un classificatore d'immagini, in un estrattore di feature dense senza il dispendio di dover apprendere parametri extra nella rete, portando ad una maggiore velocità computazionale.

Per poter descrivere il funzionamento dell'*atrous convolution* è più semplice affrontare il caso unidimensionale. Dato un segnale in input $x[i]$ ed un filtro $w[k]$ di lunghezza K, l'output $y[i]$ è definito come:

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k].$$

Dove r è detto *parametro di rate* e corrisponde al passo con il quale viene campionato il segnale, ad esempio per le convoluzioni standard il parametro r è pari ad 1 [16].

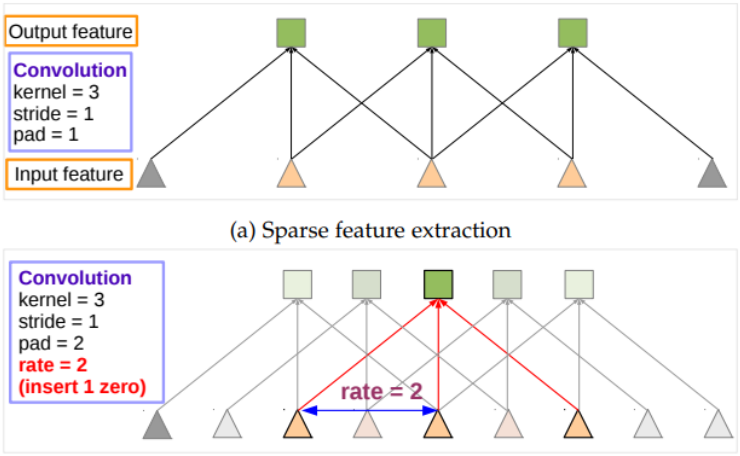

Figura 2.3.11 Esempio di atrous convolution monodimensionale (Pad si intende il padding ai bordi, stride è il passo tra due campioni) [16]

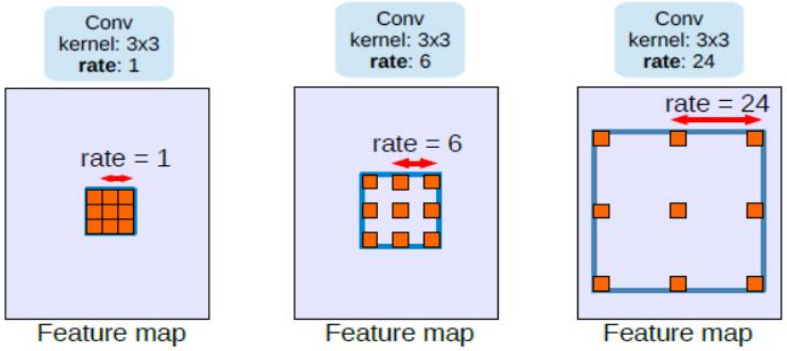

Figura 2.3.12 Esempio di atrous convolution bidimensionale [17]

In figura 2.3.11 è mostrata un'illustrazione per il caso monodimensionale, nel primo caso si utilizza una convoluzione standard ed una mappa di feature a bassa risoluzione, mentre nel secondo si utilizza un'*atrous convolution* con rate *r* pari a 2 ed una mappa di feature ad alta risoluzione. Si nota che con rate pari a 2, il segnale in input viene campionato alternativamente, questo accade perché l'*atrous convolution* è equivalente ad effettuare la convoluzione standard tra valori del segnale in input, equispaziati da *r*-1 zeri, ed i pesi del filtro. E' possibile estendere tale concetto al caso bidimensionale andando a considerare ogni dimensione spaziale. In figura 2.3.12 è mostrato un esempio nel caso bidimensionale.

Data un'immagine, lo schema standard encoder-decoder prevede un'operazione di downsampling che riduce la risoluzione, ad esempio di un fattore di 2, poi viene effettuata la convoluzione con un kernel e successivamente viene effettuato l'upsampling con lo stesso fattore. Andando a valutare la mappa di feature risultante rispetto alle coordinate dell'immagine originale, si ottiene che vengono ottenute risposte solo da ¼ delle posizioni dell'immagine. Invece, è possibile ottenere le predizioni da tutte le posizioni spaziali dell'immagine se si effettua la convoluzione con un filtro sovracampionato con degli zeri, come mostrato in figura 2.3.12, in cui il numero di parametri da apprendere (in rosso) resta costante. Dall'illustrazione in figura 2.3.13, si vede che nel caso dell'*atrous convolution* non si riduce la risoluzione della mappa di feature in ingresso, mentre ciò è necessario nello schema standard a causa dei costi computazionali.

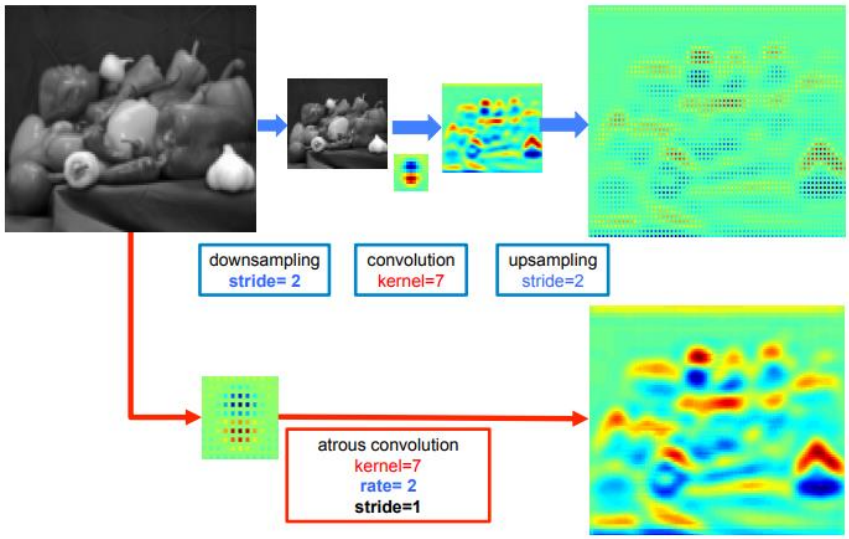

Figura 2.3.13 Schema standard di estrazione delle feature e schema tramite atrous convolution [17]

Un altro vantaggio dell'*atrous convolution* è che permette di allargare arbitrariamente il campo visivo dei filtri per poter incorporare un contesto più ampio. Essa offre, dunque, un meccanismo efficiente per trovare il miglior trade-off tra la localizzazione spaziale (campo visivo piccolo) e l'inclusione del contesto (campo visivo ampio), tutto ciò senza aumentare il numero di parametri. Tale meccanismo ruota attorno il coefficiente di rate *r*, dunque è necessario effettuare il *tuning* di esso in accordo alla grandezza delle mappe di feature.

Applicando una serie di strati con *atrous convolution* (facendo variare opportunamente il parametro di rate lungo la catena), si nota che andando in profondità nella rete si ottiene una mappa di feature in output a dimensione maggiore rispetto ad una sequenza di strati convoluzionali e di pooling standard, evitando di perdere le informazioni spaziali della mappa. In figura 2.3.14 è mostrato il confronto tra le due architetture d'esempio: in uscita alla prima si ottiene un output che è 256 volte più piccolo dell'immagine di input, mentre nella seconda si mantiene un *output_stride* pari a 16 fino alla fine.

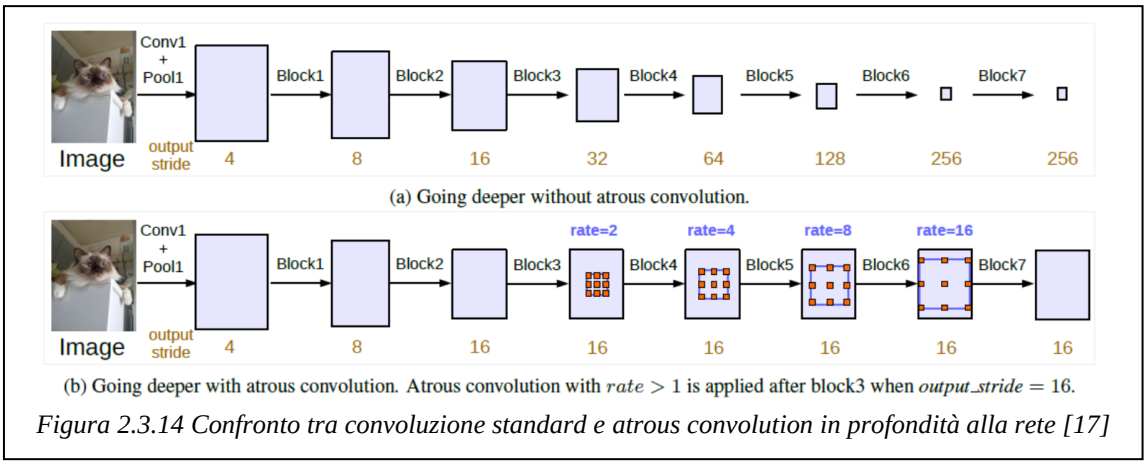

Figura 2.3.14 Confronto tra convoluzione standard e atrous convolution in profondità alla rete [17]

(2) Il secondo problema è causato dalla presenza di oggetti su diversa scala. L'approccio standard per la risoluzione è fornire in ingresso alla rete DCNN versioni riscalate della stessa immagine e poi fondere le predizioni ottenute, nonostante le ottime performance tale approccio ha l'incoveniente computazionale di dover elaborare le risposte a tutti gli strati della DCNN per ogni versione riscalata dell'immagine. La soluzione adottata da DeepLab sfrutta, invece, l'operazione di *Atrous Spatial Pyramid Pooling* (ASPP), l'idea è quella di fornire al modello informazioni multiscala sfruttando lo *Spatial Pyramid Pooling* (SPP) unito alle convoluzioni atroci con diversi coefficienti di rate *r*. A partire dalla feature map

estratta dalla rete DCNN, quattro strati convoluzionali con differenti rate vengono applicati all'immagine per permettere una segmentazione su scale multiple. Per poter incorporare anche informazioni globali dell'immagine può essere applicato un *Global Average Pooling* (GAP) alla feature map in input. Tutte le operazioni vengono effettuate in parallelo ed i risultati ottenuti vengono concatenati ottenendo in uscita un'unica feature map. Successivamente, vi è uno strato convoluzionale 1x1 per ottenere la mappa di uscita. In figura 2.3.15 è mostrata la sequenza di strati.

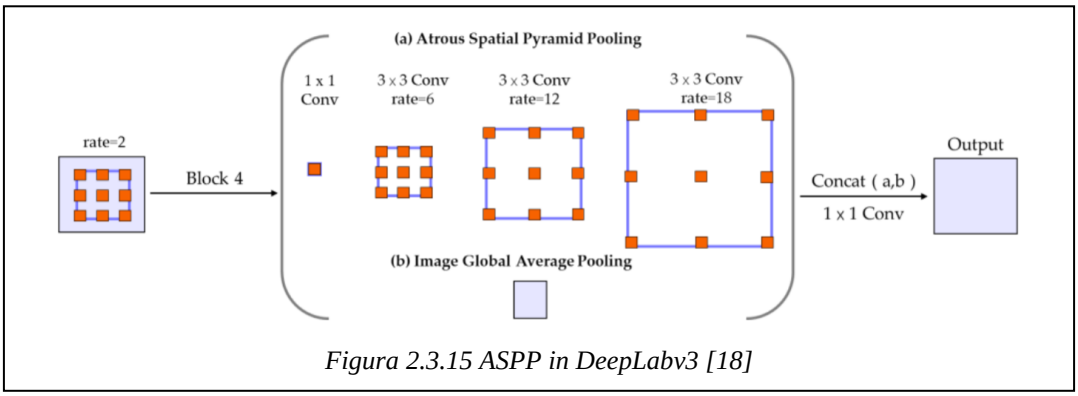

Figura 2.3.15 ASPP in DeepLabv3 [18]

(3) Il terzo problema riguarda la natura del classificatore, il quale è invariante alle trasformazioni spaziali, limitando intrinsecamente la localizzazione spaziale della DCNN. Una possibile soluzione per mitigare il problema è quella di utilizzare le *skip connections*, come accade per FCN e U-Net. L'approccio alternativo, utilizzato in DeepLab, permette di migliorare la capacità del modello nel catturare dettagli fini all'interno delle mappe di segmentazione, ciò avviene impiegando un *fully-connected Conditional Random Field* (CRF).

Tradizionalmente i *conditional random fields* (CRFs), descritti nel paragrafo 2.2, sono stati impiegati con dei classificatori deboli per rendere omogenee mappe di segmentazione rumorose, la funzione principale di questi è quella di eliminare le predizione spurie favorendo l'assegnazione di predizioni uguali per pixel in prossimità tra loro. Lavorando con architetture moderne di DCNN, le mappe di segmentazione ottenute sono differenti da quelle dei classificatori deboli, come mostrato in figura 2.3.16, l'output della DCNN è generalmente già uniforme e *smooth*. Dunque, applicando alla DCNN uno *short-range* CRF può essere controproducente, poiché renderebbe ancor meno dettagliata la mappa in uscita.

Per superare tali limitazioni viene integrato all'interno dell'architettura DeepLab un modello *fully-connected* di CRF, cioè che connette a coppie tutti i pixel della mappa in ingresso. A partire dalla definizione di CRF nel paragrafo 2.2, viene impiegata la funzione di energia:

$$E(\boldsymbol{x}) = \sum_i \theta_i(x_i) + \sum_{ij} \theta_{ij}(x_i, x_j)$$

Dove x_i è l'etichetta assegnata al pixel *i*-esimo. Il primo termine è il potenziale unario e dipende dalla probabilità di assegnazione della label al pixel *i*-esimo secondo la formula:

$$\theta_i(x_i) \;=\; -\log P(x_i)$$

Il secondo termine è la somma dei potenziali di coppia tra tutti i possibili pixel nell'immagine:

$$\theta_{ij}(x_i, x_j) = \mu(x_i, x_j) \cdot [w_1 \cdot exp(-\frac{||p_i - p_j||^2}{2\sigma_\alpha^2} - \frac{||I_i - I_j||^2}{2\sigma_\beta^2}) + w_2 \cdot exp(-\frac{||p_i - p_j||^2}{2\sigma_\gamma^2})]$$

La funzione μ è pari ad 1 se le predizioni tra i due pixel sono differenti, mentre è 0 altrimenti. Per misurare l'affinità tra le etichette si utilizzano due kernel Gaussiani: il primo dipendente sia dalla posizione spaziale p_i dei pixel che dal colore I_i, dunque forza pixel con colore e posizione simili ad avere etichette simili; il secondo dipendente solo dalla posizione spaziale. I parametri σ controllano la scala delle campane gaussiane.

In figura 2.3.16 sono mostrate le mappe risultanti alle varie iterazioni, a man a mano che ci si avvicina alla decima iterazione del CRF, la mappa di segmentazione diventa sempre più nitida.

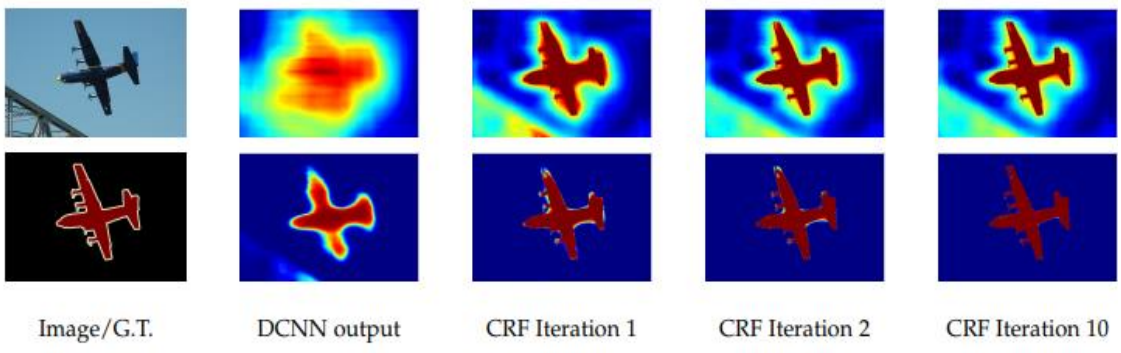

Figura 2.3.16 Mappe di segmentazione prima e dopo la funzione di softmax [16]

La risoluzione dei tre problemi che caratterizzano la segmentazione semantica porta a realizzare il modello completo di DeepLab che include una rete di backbone DCNN, come VGG-16 o ResNet-101, opportunamente modificata usando l'*atrous convolution*, un'interpolazione bilineare che allarga la mappa alle dimensioni originali ed infine un *fully-connected Conditional Random Field* che rifinisce la segmentazione catturando meglio i bordi degli oggetti. L'illustrazione del modello completo è mostrata in figura 2.3.17.

Dai risultati sperimentali sul dataset PASCAL VOC 2012 [16], si ottiene che la rete di backbone migliore da utilizzare è la ResNet-101. Le prestazioni migliorano se si utilizza la rete di backbone pre-addestrata sul dataset COCO e la *data augmentation*. Inoltre, utilizzando le tecniche di *Atrous Spatial Pyramid Pooling* e *fully-connected Conditional Random Field* si ottiene una *meanIOU* del 77,69%.

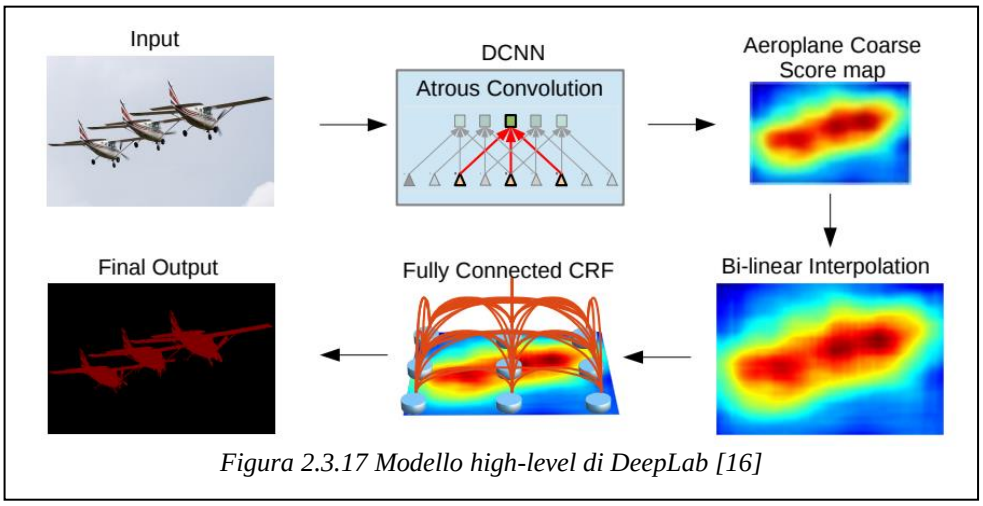

Figura 2.3.17 Modello high-level di DeepLab [16]

2.4 Metriche e funzioni di loss per la segmentazione semantica

Per valutare la qualità della segmentazione esistono diverse misure dei risultati, che differiscono dalle metriche di classificazione a causa della predizione densa. Intuitivamente per valutare l'accuratezza del modello si potrebbe sovrapporre la mappa di segmentazione in uscita con la mappa di *ground truth* dell'immagine. Esistono due metriche che sfruttano questo concetto: il coefficiente di Dice ed il coefficiente di Jaccard.

Per poter definire le metriche per la segmentazione è importante introdurre alcuni concetti per la valutazione di modelli per la classificazione. Uno di questi è la *matrice di confusione*.

La matrice di confusione è un utile strumento per misurare la precisione di un classificatore nella predizione di classi. Date C classi, la matrice di confusione *M* ha dimensioni CxC, dove l'elemento M_{ij} corrisponde al numero di campioni appartenenti alla classe *i* ma classificati dalla rete come *j*. Dunque i campioni correttamente classificati si trovano sulla diagonale principale. In figura 2.4.1 è mostrato un esempio di matrice di confusione per un problema di classificazione binario. A partire dalla matrice di confusione è dunque possibile classificare i campioni secondo quattro differenti categorie: *True Positive* (TP), ossia i campioni correttamente assegnati ad una classe; *True Negative* (TN), ossia i campioni correttamente non assegnati ad una classe; *False Positive* (FP), cioè i campioni classificati incorrettamente dal modello come appartenenti alla classe; *False Negative* (FN), cioè i campioni classificati incorrettamente dal modello come non appartenenti alla classe.

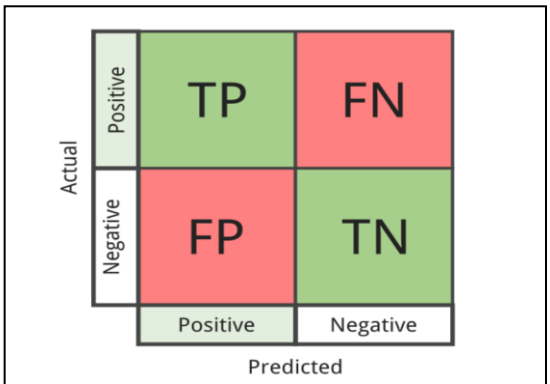

Figura 2.4.1 Esempio di confusion matrix [19]

Per problemi di classificazione multi-classe si aggiungono righe e colonne in base al numero di classi e tutti gli elementi esterni alla diagonale sono degli FP o FN.

E' possibile estendere il concetto di matrice di confusione anche al problema della segmentazione semantica. Data un'immagine W x H e supponendo di avere N classi, allora la mappa predetta in uscita è una matrice di numeri reali di dimensione W x H x N. Scorrendo lungo l'ultima dimensione del matrice per ogni pixel, si ha la distribuzione di probabilità rispetto alle classi. Nella maggioranza dei casi si mantiene solo la predizione migliore, ossia quella a probabilità più alta, assegnando a questa il valore 1 mentre a tutte le altre classi il valore 0. Ottenendo una matrice in uscita che viene detta *one-hot encoded*. A partire da questa si considerano le N matrici binarie di dimensione W x H relative ad ogni classe e si calcola per ognuna di esse la matrice di confusione.

2.4.1 Accuratezza

La metrica più semplice per valutare la segmentazione è l'accuratezza, corrisponde alla percentuale di campioni correttamente classificati. Può essere definita per ogni classe, per ogni immagine o per l'intero dataset:

- nel primo caso si parla di *pixel accuracy* (PA), si valuta la maschera binaria per la singola classe come definita nel paragrafo precedente. Tale metrica è espressa come :

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- nel secondo caso, il valore di accuratezza è la *pixel accuracy* media di tutte le classi per la singola immagine;
- nel terzo caso, il valore di accuratezza è la media delle accuratezze per le immagini del dataset;

La *pixel accuracy* può portare a risultati fuorvianti (polarizzati verso i *true negative*) quando gli oggetti da predire occupano una piccola percentuale dell'immagine mentre la maggior parte è occupata dal background. Tale problema è dovuto ad uno sbilanciamento del dataset: la distribuzione delle classi nelle immagini è squilibrata, cioè alcune sono molto più probabili rispetto ad altre all'interno di un'immagine (ad esempio la classe di background).
E' possibile inoltre definire l'accuratezza *globale* per la singola immagine: è definita come la frazione di pixel correttamente classificati, indipendentemente dalla classe.

2.4.2 Coefficiente di Dice

Il coefficiente di Dice è anche chiamato *Dice similarity coefficient* (DSC), viene spesso utilizzato come funzione di loss durante il training ed è definito come:

$$Dice(A, B) = \frac{2\|A \cap B\|}{\|A\| + \|B\|}$$

dove *A* e *B* sono due insiemi, nel nostro caso due mappe di segmentazione o insiemi di pixel. ||*A*|| è la norma di *A*, ossia la cardinalità dell'insieme (per le immagini è l'area in pixel).

2.4.3 Coefficiente di Jaccard

Il coefficiente di Jaccard è anche conosciuto come ***Intersection over Union*** (IoU) ed è definito come:

$$Jaccard(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|}$$

La metrica di IoU misura semplicemente la percentuale di coincidenza tra la mappa target e la mappa di output, ossia il numero di pixel in comune tra le due mappe diviso per il numero totale di pixel, risulta dunque di facile implementazione sfruttando le operazioni booleane AND e OR pixel per pixel.

E' possibile scrivere il coefficiente DSC in funzione del coefficiente di IoU:

$$DSC(A, B) = \frac{2J(A,B)}{1+J(A,B)}$$

Da notare che se *J* è vicino a 0, allora il *DSC* risulta essere pari al doppio di *J*.

In termini di matrice di confusione, le metriche di Dice e IoU possono essere riformulate come funzioni dei *true positive* (TP), *false positive* (FP), *false negative* (FN):

$$Dice = \frac{2TP}{2TP + FP + FN}, \qquad Jaccard = IoU = \frac{TP}{TP + FP + FN}$$

Entrambe le metriche vengono calcolate per ogni classe separatamente, poi viene effettuata la media tra tutte le misure di ogni classe per ottenere una misura globale. E' molto diffusa nell'ambito delle applicazioni di Computer Vision la media del coefficiente di IoU, anche detta ***mIoU***.

2.4.4 Pixel-wise Cross Entropy

E' la funzione di loss più utilizzata per il problema della segmentazione d'immagini. Questa esamina ogni pixel individualmente, comparando il vettore predizione della classe di appartenenza del pixel con la vera classe target *one-hot encoded*. La *Categorical Cross Entropy* (CE) per il singolo pixel è definita come:

$$CE = -\sum_{i}^{Classes} y_{i,true} \cdot log(y_{i,pred})$$

Dove Classes è il numero di classi, $y_{i,true}$ è il target per la classe i-esima, $y_{i,pred}$ è la probabilità per la classe *i*-esima [20]. Tale operazione viene ripetuta su tutti i pixel e poi si effettua la media su tutta l'immagine.

In presenza di classi sbilanciate nell'immagine, la *Cross Entropy* valuta i pixel individualmente ed effettuando la media su tutti i pixel alcune classi dominanti potrebbero influenzare di più il valore della funzione di loss. Per questo motivo è stata introdotta in [3] la versione pesata della CE, anche detta *Weighted Cross Entropy* (WCE), la quale pesa con un fattore w_i ogni classe ed è data da:

$$WCE = -\sum_{i}^{Classes} w_i \cdot y_{i,true} \cdot log(y_{i,pred})$$

2.4.5 Metriche rigide e metriche tenui

Nelle formulazioni delle metriche precedenti, in particolare quelle di *accuracy*, *DSC* ed *IoU*, si è sempre considerato solo il massimo valore nella distribuzione di probabilità della predizione per il singolo pixel, scartando tutta la restante informazione relativa alla predizione, ma non è detto che la prima scelta del modello sia quella corretta. Tali metriche vengono definite rigide o *hard*.

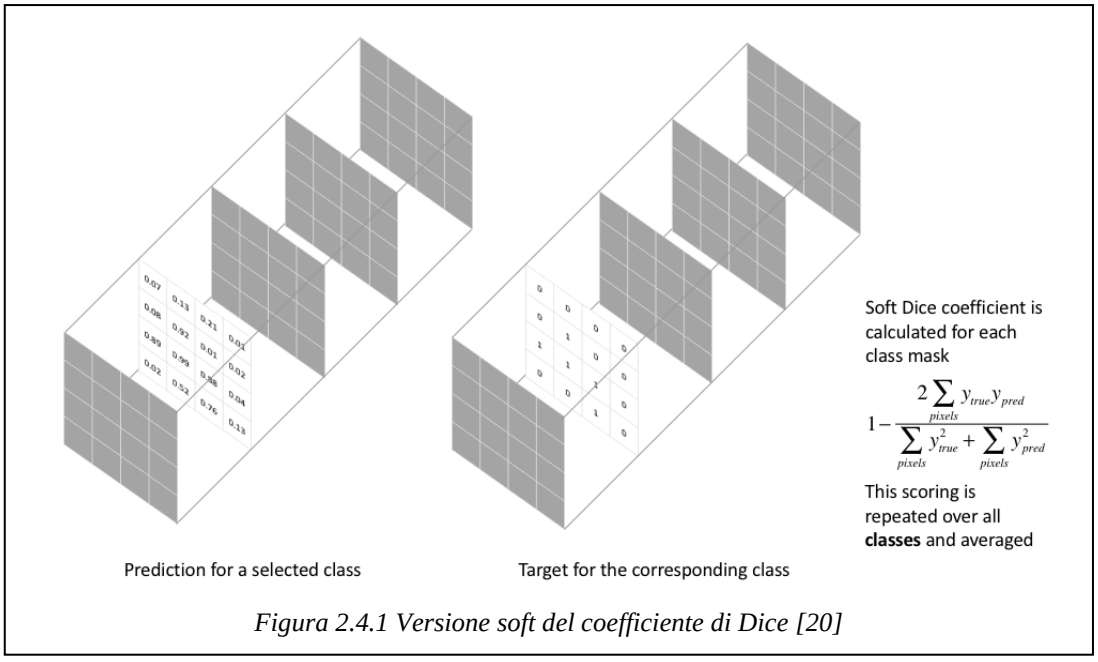

Figura 2.4.1 Versione soft del coefficiente di Dice [20]

Per tenere in considerazione anche le altre scelte è possibile definire versioni tenui, anche dette *soft*, delle metriche precedenti. In questo caso le maschere di predizione per ogni classe non sono binarie ma sono le probabilità che ogni pixel appartiene alla classe.
In figura 2. è mostrato un esempio in cui si definisce la versione *soft* del coefficiente di *Dice* per formulare la funzione di loss detta *soft Dice loss*.

2.5 Applicazioni della segmentazione semantica

La segmentazione d'immagini può essere utile per la risoluzione di diversi problemi reali che vanno dal riconoscimento della grafia alla segmentazione di organi umani fino alla realtà aumentata e molto altro ancora. Gli *use-case* più famosi e diffusi restano però l'*autonomous driving*, l'analisi biomedica ed il *remote sensing*.

2.5.1 Guida autonoma

La guida autonoma, anche detta *autonomous driving*, si riferisce alla capacità di un veicolo di percepire l'ambiente circostante e di muoversi in esso in maniera sicura senza la necessità di input umano. La segmentazione semantica è un ingrediente essenziale nella comprensione dell'ambiente urbano, come ad esempio cartelli stradali, pedoni, altri veicoli ed ostacoli. Uno dei vincoli per l'*autonomous driving* è che l'esecuzione dei task deve avvenire in tempo reale, è necessario quindi ottimizzare le reti neurali utilizzate. In figura 2.1.1 è mostrato un esempio di segmentazione semantica applicata a scene urbane.

2.5.2 Analisi di immagini biomediche

La segmentazione semantica trova casi d'uso anche nel campo biomedico. In particolare può essere utilizzata ad esempio per identificare elementi salienti in scansioni mediche, dunque risulta essere uno strumento potente per la diagnosi di malattie. Un altro possibile caso d'uso è l'identificazione e la delineazione di cellule all'interno di un'immagine, come mostrato in figura 2.5.1.

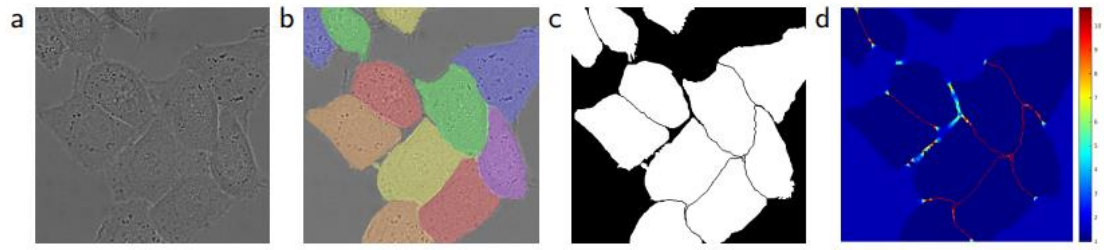

Figura 2.5.1 Esempio di segmentazione di cellule [14]

2.5.3 Telerilevamento

Un'altra applicazione importante della segmentazione è quella del telerilevamento, in inglese *remote sensing*. Casi d'uso tipici riguardano l'analisi di immagini satellitari o aeree al fine di comprendere la morfologia del suolo di regioni di interesse, oppure applicazioni per la creazione di mappe urbane per identificare strade, case ecc. In figura 2.5.2 è mostrato un esempio.

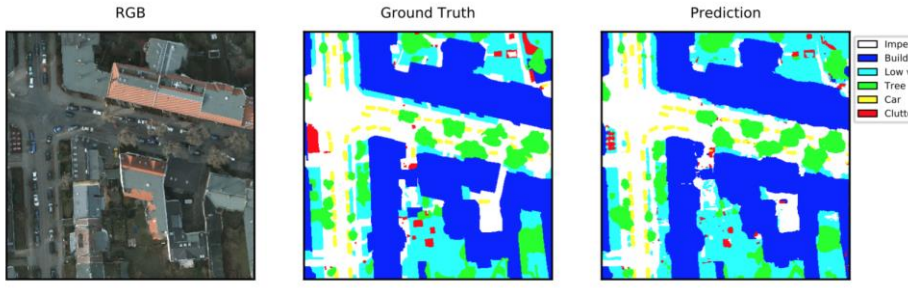

Figura 2.5.2 Esempio di segmentazione satellitare [21]

Capitolo 3: Segmentazione semantica di immagini urbane

3.1 Introduzione al problema

La segmentazione semantica di scene urbane nasce come obiettivo intermedio nel campo della guida autonoma, verso la risoluzione di task di più alto livello come lo *scene understanding* dell'ambiente urbano circostante al veicolo.

A partire da un dataset di immagini di scene urbane, il modello, ossia una rete neurale convoluzionale, apprende in che modo assegnare ogni pixel ad una classe di appartenenza, identificando nel nostro caso la strada, i segnali stradali, veicoli, pedoni ecc.

Le difficoltà del task risiedono nella complessità della scena, nella presenza di tante classi di oggetti e nella presenza di bordi difficili da assegnare. Ciò rende complicato da realizzare il raggiungimento di una buona accuratezza e di buone performance su scene urbane molto eterogenee tra loro.

3.2 Il dataset Mapillary Vistas

Mapillary Vistas è un dataset di immagini per la comprensione semantica delle scene stradali con particolare attenzione verso la guida autonoma. Le immagini provengono da diversi dispositivi fotografici (tablet, smartphone, macchine fotografiche) e sono state scattate da persone con diversa esperienza fotografica.

Il dataset è stato concepito con lo scopo di coprire un'ampia varietà di aree urbane sotto diverse condizioni climatiche in giro per il mondo, infatti rispetto ad altri dataset nello stesso settore risulta essere molto eterogeneo per scene riportate, evitando di polarizzarsi

verso paesi altamente sviluppati. Le immagini oltre ad essere variegate dal punto di vista geografico, lo sono anche dal punto di vista tecnico, infatti sono state scattate con varie lunghezze focali, risoluzioni e sensori. E' possibile tracciare un istogramma geolocalizzato sovrapposto alla mappa globale per valutare la diversità del dataset, in figura 3.2.1 è mostrato l'istogramma ed in figura 3.2.2 esempi di immagini.

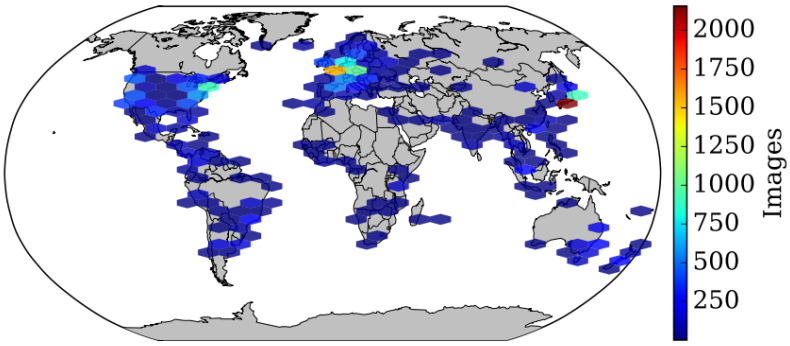

Figura 3.2.1 Istogramma geolocalizzato delle immagini [22]

Figura 3. 2.2 Esempi di immagini del dataset [22]

3.2.1 Struttura e categorie

La struttura del Mapillary Vistas è suddivisa in sottoinsiemi di *training*, *validation* e *test*. Il dataset di training è composto da 18000 immagini, quello di validation da 2000 immagini e le restanti 5000 formano quello di test. Per le immagini di training e validation sono anche presenti le mappe di segmentazione associate ad ogni immagine, di cui ogni annotazione è stata effettuata a mano con un approccio denso a poligoni [22].

Nel dataset vengono distinte 66 categorie d'oggetti, ognuna di esse appartiene ad uno dei 7 gruppi base: *object*, *construction*, *human*, *marking*, *nature*, *void*, ed *animal*. Inoltre sono presenti ulteriori mappe di *instance segmentation* con annotazioni d'istanza per 37 delle 66 classi presenti. Il numero completo di occorrenze di tutte le classi presenti nel dataset con

relative macro-classi è mostrata in figura 3.2.3. Dalla figura si nota che la maggior parte delle istanze è contenuta nella macro-classe *object*, la quale contiene varie classi di oggetti tra cui semafori e cartelli stradali, molto comuni all'interno delle immagini, ed un sottogruppo *vehicle* anch'esso molto ricorrente nelle scene urbane.

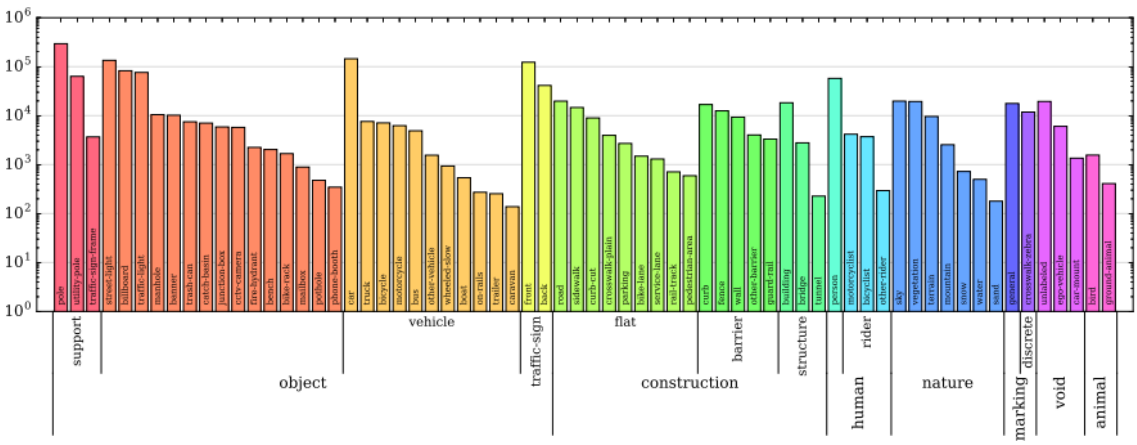

Figura 3.2.3 Occorrenze delle classi nelle immagini [22]

3.2.2 Confronto con gli altri dataset per l'autonomous driving

Nel campo della guida autonoma esistono molti dataset per la comprensione di scene urbane. Uno dei più variegati è proprio Mapillary Vistas, come mostrato in figura 3.2.4, che insieme ad *ApolloScape* rappresenta lo stato dell'arte nel settore.

	Labeled Images for Training	Classes	Multiple Cities	Environment
KITTI	200	34	No	Daylight
Cityscapes	3478	34	Yes	Daylight
Mapillary Vistas	20k	66	Yes	Daylight, rain, snow, fog, haze, dawn, dusk and night
ApolloScape	147k	36	No	Daylight, snow, rain, foggy
BDD100K	8000	19	Yes	Daylight, rain, snow, fog, haze, dawn, dusk and night

Figura 3.2.4 Confronto tra dataset per l'autonomous driving [23]

3.3 Approccio utilizzato

L'approccio utilizzato è quello tipico dei task di Deep Learning e prevede i seguenti passi in figura 3.3.1:

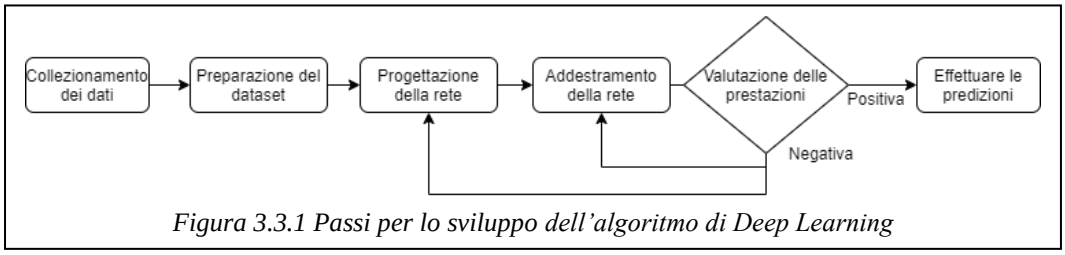

Figura 3.3.1 Passi per lo sviluppo dell'algoritmo di Deep Learning

Nel caso in cui le prestazioni non sono ottimali si decide se ritornare in fase di sviluppo per modificare la rete oppure se modificare i parametri di addestramento della rete.

Per lo sviluppo del codice è stato utilizzato il linguaggio *Python*, utilizzando apposite librerie per il *deep learning* e l'analisi dei dati (TensorFlow, Keras, NumPy, Matplotlib) ed apposite librerie per le immagini e la segmentazione semantica (Skimage, Keras_segmentation).

L'ambiente di lavoro utilizzato per la progettazione e l'apprendimento delle varie reti è stato *Google Colaboratory*, il quale fornisce risorse computazionali come GPU e TPU gratuitamente, tuttavia utilizzabili solo in modo limitato nel tempo. Per tale motivo si è cercato di adattare il problema della segmentazione proposto ai limiti dell'ambiente di lavoro utilizzato. Tale adattamento prevede l'utilizzo di un *subset* dell'intero dataset Mapillary Vistas ed il training dei modelli su un numero ristretto di epoche, in maniera tale da riuscire a sfruttare le risorse offerte evitando disconnessioni intermedie.

3.3.1 Preparazione dei dati

A partire dal dataset Mapillary Vistas, sono state selezionate circa il 40% delle immagini totali. Successivamente, è stato effettuato un pre-processing sia delle immagini che delle mappe di training e validation. In particolare per le mappe di segmentazione, queste sono state precedentemente elaborate per renderle compatibili all'input della rete, infatti le mappe del dataset Mapillary Vistas vengono presentate in versione RGB e necessitano di una conversione in scala di grigi (66 livelli di grigio, ognuno di essi associato ad una classe).

Le immagini di training e validation sono state compresse con algoritmo di compressione JPEG (sfruttando la libreria PIL), utilizzando un fattore di qualità pari a 45. E' stato

necessario effettuare tale compressione per evitare di sovraccaricare l'ambiente di lavoro durante l'addestramento. Inoltre, le immagini e le mappe vengono prima ridimensionate alla risoluzione 416x608, e poi rese a media nulla.

Dopo aver effettuato la preparazione dei dati si creano i generatori delle coppie (immagine, mappa), sia per il subset di training che per quello di validation, da inserire in ingresso alla rete.

3.3.2 Architetture di rete

Sono state utilizzate le tecniche di deep learning descritte nel capitolo precedente, ossia adattando le reti neurali convoluzionali profonde (DCNNs) al problema della segmentazione. In particolare le architetture sono ispirate a [3], per le reti FCN, mentre a [14] per le reti che utilizzano U-Net.

I modelli di rete per la segmentazione necessitano di una DCNN da utilizzare come backbone, si è scelto di utilizzare a tal scopo le reti *ResNet50* e *VGG16*, le quali sono state pre-addestrate con opportuno *fine-tuning* sul dataset *ImageNet*, cioè inizializzando i pesi della rete addestrata su tale dataset. Lo scopo del fine-tuning è quello di permettere alla rete di avere una conoscenza pregressa nel riconoscimento di feature comuni di basso livello.

In figura 3.3.2 è riportata la tabella con tutte le architetture complete utilizzate. Le reti FCN sono caratterizzate da un elevato numero di parametri *trainable*, ciò le rende molto complesse da allenare utilizzando un dataset limitato.

Rete	Parametri trainable	Livelli convoluzionali	Skip connections	U-net	FCN
fcn_32_vgg	152,373,122	16	No	No	Si
fcn_32_resnet50	469,474,370	50	No	No	Si
fcn_8_vgg	135,836,294	16	Si	No	Si
vgg_unet	12,359,874	23	Si	Si	No

Figura 3.3.2 Tabella di comparazione tra le architetture usate

3.3.3 Addestramento

L'addestramento è la fase in cui i parametri della rete si aggiornano e dove, quindi, il modello effettivamente apprende la classificazione pixel-wise delle immagini.

E' importante: scegliere accuratamente la funzione di loss per misurare l'errore delle

predizioni; selezionare l'ottimizzatore che meglio aggiorna i parametri della rete in base alla funzione di loss; regolare i parametri di apprendimento della rete; tenere in considerazione anche le limitazioni temporali e computazionali.

Tra i parametri da regolare in fase di addestramento vi sono: il numero di epoche, il quale indica il numero di volte che il modello lavorerà sull'intero dataset; la finestra di batch, la quale indica il numero di campioni che verrà fornito in ingresso in ogni passo di *backward/forward propagation*; il learning rate, è il parametro che permette all'ottimizzatore, nella fase di *backward propagation*, di aggiornare i pesi della rete, cercando di minimizzare l'errore rispetto ai valori di output desiderati; la finestra di batch di validation, viene utilizzata in modo da non elaborare tutto il dataset di validation in una volta, ma suddividendolo in dei sottoinsiemi con dimensione fissata.

In tabella 3.3.3 sono mostrate le scelte dei parametri dopo varie sperimentazioni sui modelli usati. Si è scelto di utilizzare un batch-size sia per il training che per la validation pari a 2, evitando di sovraccaricare la memoria GPU. Per semplicità si è scelto di usare come loss function la *Categorical Cross Entropy* (CCE), così come definita nel paragrafo 2.4.4.

Parametri	Scelte effettuate
Numero di epoche	5
Batch size	2
Validation batch size	2
Loss function	CCE
Ottimizzatore	Adam
Learning rate	0.001

Tabella 3.3.3 Scelte dei parametri

L'ottimizzatore scelto è Adam (Adaptive Moment Estimation), il quale è considerato il migliore per problemi di segmentazione semantica, ad esso è associato il valore di *learning rate* pari a 0.001.

3.3.4 Metriche di valutazione

La metrica utilizzata in fase di training e validation è stata l'accuracy media, mentre in fase di testing si è misurata anche il *mIoU* e la *class-wise IoU*.

3.4 Risultati ottenuti

3.4.1 Risultati di training

Gli andamenti della funzione di loss e dell'accuracy media al susseguirsi delle epoche permettono di valutare le prestazioni in fase di addestramento del modello.

Per via della presenza di molti parametri allenabili come mostrato nella tabella 33, le reti che utilizzano l'architettura FCN richiedono molto più tempo di addestramento rispetto alle architetture U-Net, ed inoltre presentano risultati più bassi. Per comparare gli addestramenti, oltre alle architetture FCN e U-Net sono stati addestrati anche i modelli *vgg_segnet* e *vgg_pspnet*. Il primo utilizza una classica architettura encoder-decoder [24], mentre la seconda la tecnica di Pyramid Space Pooling (PSP) [25]. Di seguito sono mostrati gli andamenti delle funzioni di loss e dei valori di accuracy per le reti addestrate.

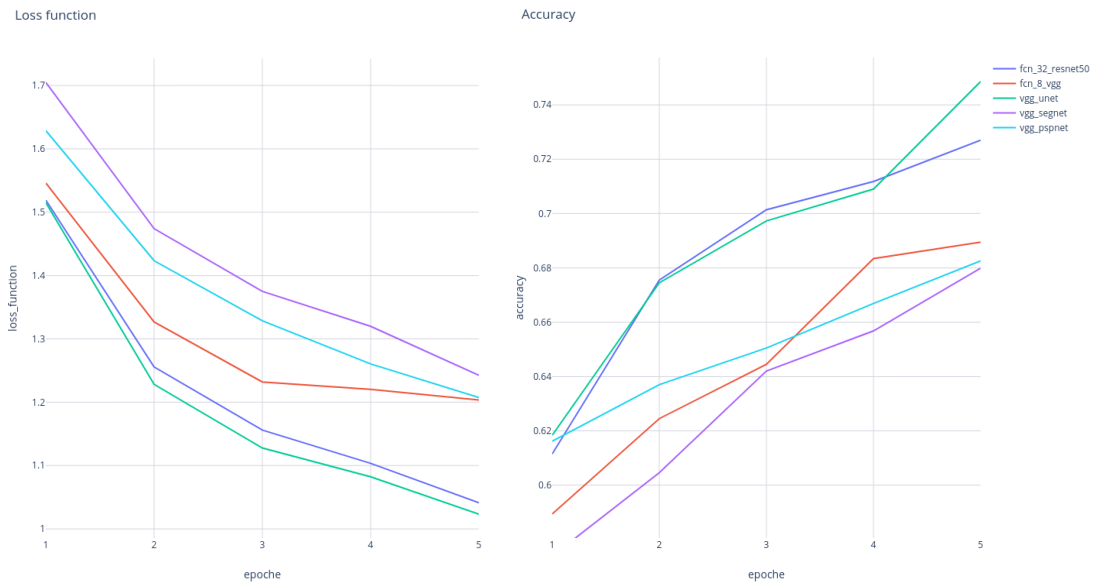

Figura 3.4.1 Andamenti della loss function (a sinistra) e dell'accuracy (a destra)

Come si vede dal grafico della loss function e dell'accuracy (in figura 3.4.1), la migliore architettura risulta essere la rete *vgg_unet* che raggiunge quasi lo 0.75 di accuracy.

In generale i risultati di training non sono pienamente soddisfacenti per via delle poche epoche di addestramento e dell'utilizzo di un subset di dati. Aumentando le epoche di training e utilizzando l'intero dataset si dovrebbero raggiungere risultati migliori.

3.4.2 Risultati sul testing set di Mapillary Vistas

A partire da immagini scelte a campione dal dataset, sono state effettuate le predizioni dei vari modelli. Le singole predizioni ottenute sono in scala di grigi e sono state convertite rispetto alla mappa di colori del dataset. Ottenute le immagini convertite si è poi effettuata la misura delle metriche di IoU rispetto alle annotazioni reali come definite nel paragrafo 2.4.3. Essendo il dataset Mapillary Vistas molto vasto rispetto alle categorie di oggetti classificabili, le classi sono in totale 66 (come mostrato nel paragrafo 3.2.1), risulta difficile effettuare una predizione pixel-wise accurata su tutte le possibili categorie utilizzando un numero ristretto di epoche e di campioni. Le mappe di segmentazione ottenute sono mostrate in figura 3.4.2, le predizioni da sinistra verso destra sono delle reti *fcn_32_resnet50*, *fcn_8_vgg*, *vgg_unet*.

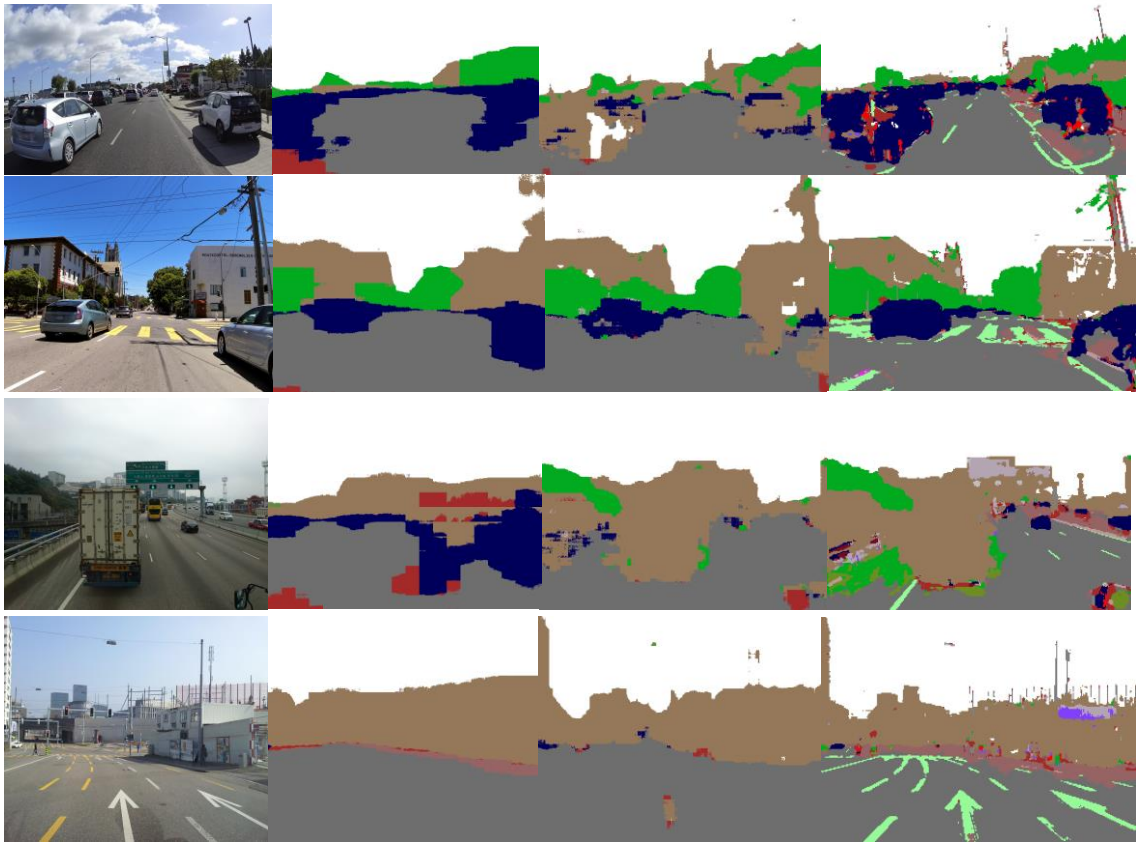

Figura 3.4.2 Risultati su immagini di Mapillary Vistas (in ordine fcn_32_resnet50, fcn_8_vgg, vgg_unet)

Valutando le varie predizioni si vede che, eccetto per *vgg_unet*, le reti presentano delle mappe in uscita molto grezze e dunque i risultati non sono ottimali. I risultati migliori si ottengono, dunque, con la rete *vgg_unet*, la quale riesce a definire discretamente bene i bordi dei veicoli e dei segnali su strada, mentre per la classificazione riesce a predire bene, anche se a volte con del rumore, i gruppi base.

3.4.3 Risultati su scene urbane di Napoli

Di seguito sono riportate le mappe predette delle scene urbane di Napoli, queste risultano essere molto complicate da segmentare per la varietà di oggetti presenti e per la presenza di molti dettagli da valutare. Anche in questo caso i risultati migliori si ottengono con *vgg_unet*. Per le predizioni di *vgg_unet* è presente molto rumore per via dei dettagli, mentre per le altre reti le mappe risultano essere troppo grezze. In figura 3.4.3 sono mostrati degli esempi.

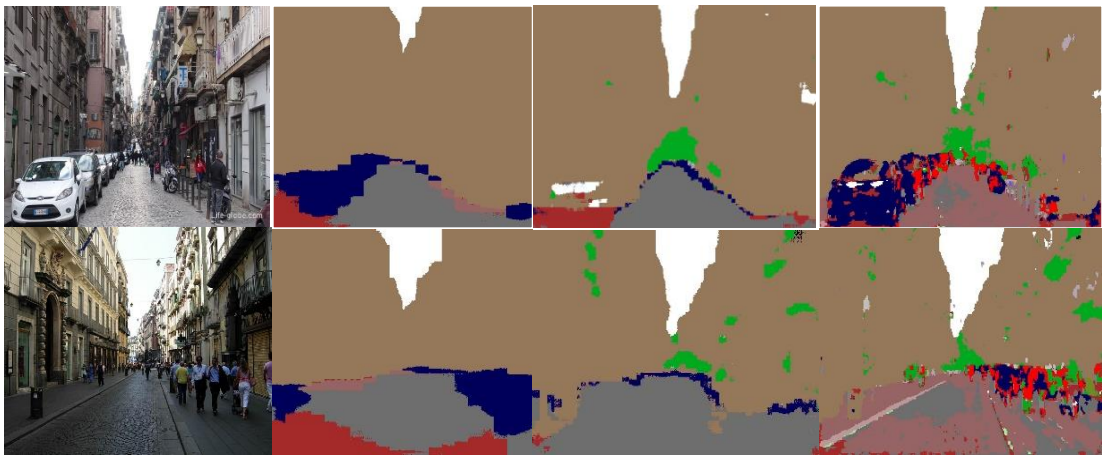

Figura 3.4.3 Risultati sulle scene urbane di Napoli (in ordine fcn_32_resnet50, fcn_8_vgg, vgg_unet)

Conclusioni

Sono state presentate delle tecniche di Deep Learning per la risoluzione del problema della segmentazione semantica. Le architetture di base utilizzate nel caso di studio delle scene urbane per la guida autonoma sono state U-Net e FCN, le quali non sono attualmente lo stato dell'arte per la risoluzione di tale task. I migliori risultati sperimentali che si sono ottenuti con tali architetture sono quelli della rete *vgg_unet*, che nonostante il basso numero di epoche e la ridotta grandezza del dataset di addestramento è riuscita a produrre delle ottime mappe di segmentazione. Un possibile miglioramento potrebbe essere quello di aumentare il numero di epoche e la dimensione del dataset di addestramento, anche utilizzando tecniche di data augmentation per migliorare le predizioni sulle singole classi, tuttavia con la necessità di dover utilizzare una potenza computazionale maggiore.

Un altro dei possibili sviluppi futuri è quello di utilizzare il modello DeepLab, tale architettura è stata sviluppata e resa open-source da Google ed è considerata la migliore per la segmentazione semantica. Utilizzando l'ultima versione, cioè la DeepLabv3+, e l'intero dataset Mapillary Vistas, con un addestramento della rete su un numero non troppo limitato di epoche, si riuscirebbero a raggiungere risultati ottimi nella predizione sulle 66 classi del dataset.

Bibliografia

[1] Martin Thoma, “A Survey of Semantic Segmentation”, arXiv:1602.06541, 2016

[2] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, “Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art”, arXiv:1704.05519, 2017

[3] Jonathan Long, Evan Shelhamer, Trevor Darrell, “Fully Convolutional Networks for Semantic Segmentation”, arXiv:1411.4038, 2014

[4] Lawtomated, lawtomated.com/a-i-technical-machine-vs-deep-learning/, 07/09/2021

[5] Medium, medium.com/nerd-for-tech/flux-prediction-using-single-layer-perceptron-and-multilayer-perceptron-cf82c1341c33, 07/09/2021

[6] Unipr, ce.unipr.it/people/medici/geometry/node107.html, 07/09/2021

[7] Jeong J., Yoon T.S., Park J.B., “Towards a Meaningful 3D Map Using a 3D Lidar and a Camera. Sensors”, doi.org/10.3390/s18082571, 2018

[8] G. Bianco, “Markov Random Field: Teoria e applicabilità nell’elaborazione delle immagini”, di.univr.it/documenti/Tesi/allegato/allegato517081.PDF, 1998

[9] X. Li, H. Sahbi, "Superpixel-based object class segmentation using conditional random fields", IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), doi: 10.1109/ICASSP.2011.5946600, 2011

[10] Yasrab, Robail & Gu, Naijie & Zhang, Xiaoci, “An encoder-decoder based Convolution Neural Network (CNN) for future Advanced Driver Assistance System (ADAS)”, 10.3390/app7040312, 2017

[11] Stanford, cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf, 07/09/2021

[12] Naokishibuy, naokishibuya.medium.com/up-sampling-with-transposed-convolution, 07/09/2021

[13] Towardsdatascience,towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1, 07/09/2021

[14] Olaf Ronneberger, Philipp Fischer, Thomas Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, arXiv:1505.04597, 2015

[15] Wikipedia, en.wikipedia.org/wiki/U-Net, 07/09/2021

[16] Chen, Liang-Chieh & Papandreou, George & Kokkinos, Iasonas & Murphy, Kevin & Yuille, Alan, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”, IEEE Transactions on Pattern Analysis and Machine Intelligence, PP. 10.1109/TPAMI.2017.2699184, 2016

[17] Towardsdatascience,towardsdatascience.com/review-deeplabv3-atrous-convolution-semantic-segmentation-6d818bfd1d74, 07/09/2021

[18] Lin, Yeneng & Xu, Dongyun & Wang, Nan & Shi, Zhou & Chen, Qiuxiao, “Road Extraction from Very-High-Resolution Remote Sensing Images via a Nested SE-Deeplab Model”, Remote Sensing 12 2985, 10.3390/rs12182985, 2020

[19] Towardsdatascience, towardsdatascience.com/visual-guide-to-the-confusion-matrix-bb63730c8eba, 07/09/2021

[20] Jeremyjordan, jeremyjordan.me/semantic-segmentation/, 07/09/2020

[21] Azavea, azavea.com/blog/2017/05/30/deep-learning-on-aerial-imagery, 07/09/2021

[22] G. Neuhold, T. Ollmann, S. R. Bulò & P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes", IEEE International Conference on Computer Vision (ICCV), pp. 5000-5009, doi: 10.1109/ICCV.2017.534, 2017

[23] Autonomous-driving, autonomous-driving.org/2018/07/15/semantic-segmentation-datasets-for-urban-driving-scenes, 07/09/2021

[24] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”, arXiv:1511.00561, 2015

[25] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia, “Pyramid Scene Parsing Network”, arXiv:1612.01105, 2016