UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Sviluppo ed applicazione di reti neurali per segmentazione semantica a supporto della navigazione di rover marziani

Relatore
Prof. Stefano Debei

Correlatori
Prof. Marco Pertile
Ing. Sebastiano Chiodini

Laureando
Luca Torresin

ANNO ACCADEMICO 2019/2020

Sommario

A supporto della possibile navigazione autonoma di rover marziani possono trovare impiego algoritmi di analisi di attraversabilità del terreno. Uno di questi è la segmentazione semantica, una tecnica di classificazione di pixel sulla base della categoria alla quale sono attribuiti. Nella presente tesi, l'implementazione è condotta mediante reti neurali convoluzionali addestrate su dataset classificati manualmente e suddivisi in sottoinsiemi di addestramento, validazione, test. I dataset preparati sono due: uno di immagini contenenti terreno sabbioso e rocce artificiali ed uno di immagini provenienti dal rover *Curiosity*. La tecnica di segmentazione semantica è sperimentata sul primo dataset per fornire una base di confronto con le prestazioni ottenute sul secondo, corrispondente all'applicazione reale. Le reti di addestramento sono DeepLabv3+, di recente ideazione, ed una rete creata per l'occasione. Data l'assenza di ambiguità nei contenuti delle immagini, i risultati ottenuti sono buoni per il primo dataset, meno buoni per il secondo, caratterizzato invece da un'ampia variabilità delle classi. La tecnica di segmentazione si è rivelata robusta e le prestazioni ottenute ne suggeriscono un impiego a supporto di *path planning* e *obstacle avoidance* di rover marziani.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

Uno degli aspetti di grande interesse nel campo della robotica spaziale è la navigazione autonoma di robot mobili, in particolare di rover marziani. Attualmente, la navigazione di rover marziani prevede la gestione di quattro *task* fondamentali: *path planning*, *obstacle avoidance*, *localization*, *terrain traversability analysis*. [1]

Path planning consiste nel definire una traiettoria che il rover può percorrere per raggiungere la destinazione finale partendo da una posizione iniziale. Si distingue tra *global path planning* e *local path planning*. Il primo consiste nella determinazione di una traiettoria sulla base di dati noti a priori, il secondo prevede la gestione degli input provenienti dai sensori per effettuare correzioni istante per istante nel rispetto delle tolleranze imposte dalla traiettoria globale. [1]

Obstacle avoidance concerne l'individuazione ed il superamento o aggiramento di ostacoli che impediscono al rover lo svolgimento della funzione corrente. Compito del sistema di percezione è inferire la presenza di ostacoli, quindi la loro geometria e la loro distanza, sulla base dei dati forniti da stereo camera e/o LiDAR. Le tecnologie proposte sono in genere affidabili in presenza di ostacoli statici ma in difficoltà con ostacoli dinamici e in numero elevato. [1]

Localization consiste nella determinazione della posizione del rover rispetto all'ambiente circostante. Tecniche di *visual odometry* tracciano gli spostamenti del rover rispetto alla posizione di partenza in assenza di GPS. SLAM (**S**imultaneous **L**ocalization **A**nd **M**apping) è invece un insieme di tecniche di mappatura di ambienti sconosciuti, effettuata in contemporanea alla localizzazione. [1]

Terrain traversability analysis è una componente indispensabile della navigazione in terreni di cui non è nota a priori la conformazione e costellati da ostacoli, come appunto quelli marziani. In percorsi su terreno marziano i rover incontrano zone non attraversabili a causa dell'interazione ruote-suolo. Si fa riferimento per lo più a rocce di varia natura e dimensione; altre pericolosità sono insite in superfici in apparenza innocue ma impattanti sulla mobilità del veicolo: terreno sabbioso, ad esempio, può complicare l'attraversamento per effetto dello slittamento delle ruote. [1]

Il rover marziano più recente, Curiosity (2011), dispone di tre modalità di navigazione: *blind-drive*, *hazard avoidance*, *visual odometry*. In modalità *blind-drive*, operatori a terra estraggono informazioni dalle immagini provenienti dalle NAVCAM e tracciano una traiettoria percorribile. Il rover compie l'attraversamento seguendo le istruzioni ricevute e al contempo misura lo spostamento tramite *dead reckoning*. Questa modalità consente rapidi spostamenti ma è limitata in distanza dalla quantità di informazioni visive disponibili. In modalità *hazard avoidance* invece, il rover opera autonomamente fermandosi ad intervalli regolari per aggiornare la traiettoria passo passo con il supporto dei dati forniti dalle fotocamere HAZCAM. Le stereo camere catturano immagini da cui il computer di bordo estrae informazioni inerenti la presenza di ostacoli. *Visual odometry* è infine adottata per verificare se il rover è bloccato da terreno sabbioso od altri ostacoli per effetto dello slittamento delle ruote, non computabile tramite *dead reckoning*. [1]

Curiosity ed i suoi predecessori hanno ottenuto largo successo nella navigazione di terreno marziano ma vi sono comunque ampi margini di miglioramento. In primo luogo, lo spostamento del rover avviene molto lentamente, sia in caso di operazione autonoma che assistita: limitate risorse computazionali di bordo e la dipendenza da operatori di terra impongono al robot stop continui e ne inficiano la produttività. [1] In secondo luogo, le interazioni ruote-suolo pongono notevoli difficoltà nella determinazione della traiettoria in particolare in presenza di slittamento. La preservazione dell'integrità strutturale delle ruote impone inoltre l'applicazione delle informazioni di analisi del terreno alla determinazione del percorso, sia essa effettuata a bordo (online) o a terra (offline).

La generazione di mappe tridimensionali dell'ambiente circostante può soddisfare requisiti di ciascun aspetto precedentemente discusso, dal *path planning* all'analisi di attraversabilità del terreno, in vista di una possibile navigazione interamente automatizzata. La ricostruzione può essere effettuata con tecniche diverse, come ad esempio *meshes*, *elevation maps*, *cost maps*, a seconda delle geometrie dell'ambiente, delle risorse computazionali disponibili (in primis: memoria volatile e potenza di calcolo), dell'impiego. [4] Sviluppi recenti hanno dato luce alla possibilità di creare mappe semantiche dalla fusione dei risultati di segmentazione con i dati di una mappa 3D, effettuando la proiezione delle etichette dal piano immagine alle superfici tridimensionali. [2]

Questa tesi presenta una tecnica a supporto dell'analisi di attraversabilità del terreno basata su immagini catturate dalle fotocamere del rover. In particolare, ci si propone di valutare la possibilità di far uso di reti neurali convoluzionali al fine di ottenere risultati soddisfacenti a fondamento della costruzione di mappe semantiche tridimensionali. Indipendentemente dalla loro applicazione, sia essa online od offline, le mappe semantiche possono indirettamente velocizzare lo spostamento del rover, fornendo al contempo una variegata quantità di informazioni sulla pericolosità del terreno, a supporto di *path planning* e *obstacle avoidance*.

Lo scritto è strutturato come segue. Il capitolo a seguire descrive a grandi linee le caratteristiche peculiari delle reti neurali convoluzionali, presenta alcune architetture convoluzionali di successo, l'architettura encoder-decoder e le reti impiegate. Il capitolo successivo espone la procedura di preparazione dei dataset, dalla definizione delle classi alla suddivisione in sottoinsiemi, passando per la classificazione manuale e le tecniche di *data augmentation*. Segue una descrizione dei due dataset. Gli ultimi due capitoli presentano la fase di addestramento ed i risultati.

CAPITOLO 2

Reti neurali convoluzionali

La segmentazione è una tecnica di visione artificiale di suddivisione di immagini in sezioni caratterizzate da simili proprietà visuali. La segmentazione semantica, nello specifico, consiste nell'associazione di porzioni di immagini ad una categoria. Essa trova impiego in settori dove sono richieste mappe immagine precise, come ad esempio: guida autonoma, ispezione idustriale, immagini satellitari e, per l'appunto, visione robotica. Date le ottime prestazioni, la segmentazione semantica è attualmente condotta tramite le cosiddette **reti neurali convoluzionali**.

Le reti neurali convoluzionali (**C**onvolutional **N**eural **N**etworks o *ConvNets*) sono reti neurali profonde realizzate per operare su input a griglia caratterizzati da forti dipendenze spaziali in regioni locali. Un esempio di input a griglia è un'immagine bidimensionale: essa non è altro che una matrice di valori compresi fra 0 e 255. Pixel adiacenti presenti in un'immagine sono interlacciati l'uno all'altro e nel loro insieme definiscono un pattern, una texture, un contorno, etc. Le CNN associano queste caratteristiche a valori detti "pesi", che saranno simili per regioni locali con pattern simili. La qualità che le contraddistingue è anche l'operazione che dà loro il nome, ossia la convoluzione. Essa non è altro che il prodotto scalare tra matrici, nello specifico tra una struttura a griglia di pesi e una simile struttura proveniente dall'input. Le reti convoluzionali sono *deep network* in cui lo strato convoluzionale appare almeno una volta, benchè la maggioranza ne usi ben più di uno. [6]

L'input di una ConvNet è un'immagine, ossia una matrice di valori di ogni singolo pixel occupante una precisa posizione all'interno dell'immagine. Le immagini in

formato RGB sono descritte da una terna di matrici delle intensità dei colori primari (rosso, verde, blu); le dimensioni di un'immagine non sono quindi limitate ad altezza e larghezza ma ve n'è una terza: *depth*, profondità. Le dimensioni dell'immagine, inclusa la profondità, saranno conferite allo strato di input, il primissimo *layer* di una CNN. Le mappe di attivazione successive, ossia gli input degli strati successivi, hanno anch'esse una struttura a più dimensioni, e saranno in numero congruente con le proprietà indipendenti rilevanti alla classificazione. [6]

2.1 Reti neurali artificiali

Le reti neurali convoluzionali sono parte di una catergoria ben più grande, ossia le reti neurali artificiali. Una rete neurale artificiale (**A**rtificial **N**eural **N**etwork) è un gruppo interconnesso di piccole unità computazionali denominate *neuroni*, il cui il principio di funzionamento trova fondamento nelle reti neurali biologiche. Un neurone j è descritto da una funzione matematica del tipo:

$$y_j = f(x_j)$$

con

$$x_j = \sum(w_{ij} + b_j)$$

dove i indica neuroni precedenti j, x_j e y_j input ed output di j, w_{ij} il peso della connessione tra i due neuroni, b_j il *bias*. La funzione f è in genere non-lineare ed è denominata *funzione di attivazione*. Un esempio di come sono connessi tra loro i neuroni è raffigurato nella figura sottostante: ogni cerchio rappresenta un neurone, definito da f, mentre ciascuna freccia grigia è una connessione di peso w_{ij}. Quello di figura è un tipico *feedforward neural network* ed è contraddistinto da strati, composti da più neuroni, denominati: *strato di input*, *strato nascosto*, *strato di output*.

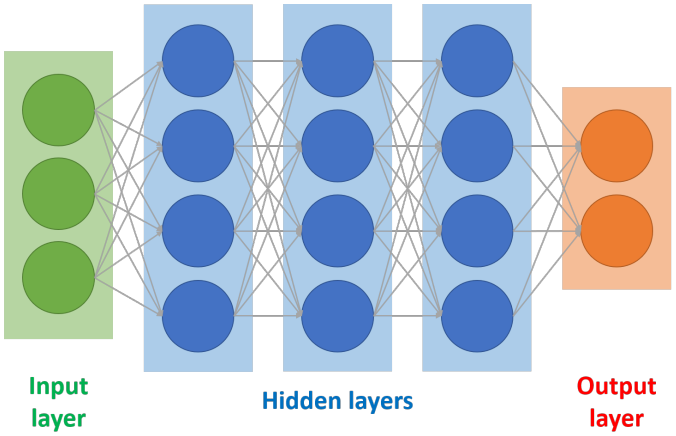


Figura 2.1: Struttura di base di una rete neurale *feedforward*.

L'aspetto caratterizzante delle reti neurali artificiali è la loro capacità di approssimare funzioni continue con un solo singolo strato. In un generico problema di classificazione multiclasse, le classi non sono linearmente separabili ed è necessario formulare una funzione lineare che sia in grado di eseguire la corrispondente separazione lineare. Affrontando il problema con una rete neurale artificiale non è necessario definire alcuna funzione ma solo i cosiddetti *iperparametri* (numero di strati, numero di neuroni, tipo di funzioni di attivazione): il modello impara a classificare gli elementi una volta determinati i pesi tramite l'algoritmo di retropropagazione dell'errore. [5]

Retropropagazione dell'errore Si consideri una semplice rete neurale il cui output è y_{output}. La quantità di cui y_{output} si discosta dal valore desiderato al termine della propagazione in avanti è l'errore E. L'algoritmo di retropropagazione calcola le derivate dell'errore rispetto ai pesi $\frac{\mathrm{d}E}{\mathrm{d}w_{ij}}$ e li aggiorna secondo l'espressione:

$$w_{ij}^{n+1} = w_{ij}^{n} - \alpha \frac{\mathrm{d}E}{\mathrm{d}w_{ij}}$$

dove α è una costante positiva detta *learning rate* e assegnata per via empirica. Molto semplicemente, se l'errore descresce all'aumentare del peso allora il peso aumenta, se invece l'errore aumenta all'aumentare del peso, allora questo decresce. L'algorit-

mo risale l'intera rete neurale calcolando gli errori passo-passo fino al layer di input e procede iterando fino a convergenza. [19]

2.2 Architettura

Le reti neurali convoluzionali operano su strutture a griglia contraddistinte da relazioni spaziali tra pixel, ereditate da uno strato al successivo tramite valori che descrivono piccole regioni locali dello strato precedente. L'insieme di matrici degli strati nascosti (*hidden*), risultato della convoluzione o di altre operazioni, è definita *feature map* o *activation map*, i parametri addestrabili sono tensori denominati *filtri* o *kernel*.

Una CNN è composta da successioni di strati, illustrati in dettaglio nei paragrafi seguenti. Gli strati principali sono:

1. strato convoluzionale;
2. strato di attivazione;
3. strato di *pooling*.

2.2.1 Strato convoluzionale

La convoluzione è l'operazione fondamentale delle CNN. Essa dispone un filtro in ogni possibile posizione dell'immagine, coprendola interamente, e computa il prodotto scalare tra il filtro stesso e la matrice corrispondente del volume di input, avente eguale dimensioni. È possibile visualizzare la convoluzione come una sovrapposizione del *kernel* sull'immagine in input (o strato nascosto). [6] Un filtro è caratterizzato dai seguenti iperparametri (*hyperparameters*):

- altezza F_q
- larghezza F_q
- profondità d
- numero q

Solitamente i filtri hanno forma quadrata e profondità uguale a quella dello strato al quale sono applicati. Per un'immagine in scala di grigio, il primo filtro convoluzionale

$F \times F \times d$ potrà avere dimensione $5 \times 5 \times 1$, oppure $7 \times 7 \times 1$, ma non $5 \times 5 \times 3$ o $5 \times 5 \times 3$, casi in cui è invece possibile applicarlo ad immagini in formato RGB. Il numero di possibili allineamenti tra filtro e immagine definisce altezza e larghezza della successiva *feature map*.

Dallo strato nascosto q_1, il filtro produce uno nuovo strato nascosto q_2

$$L_{q+1} = L_q - F_q + 1$$

$$B_{q+1} = B_q - F_q + 1$$

Ad esempio, per immagini di dimensioni 32×32, un filtro 5×5 genera uno strato nascosto 28×28.

È opportuno fare distinzione tra profondità del filtro e profondità di strato nascosto/mappa di attivazione: la prima, d è la stessa dello strato al quale è applicato, la seconda deriva invece dal numero di filtri applicati. Il numero di filtri è un iperparametro definito sulla base della capacità di distinguere forme sempre più complesse che si vuole conferire alla rete. I filtri sono dunque i componenti a cui saranno associate le caratteristiche dei pattern delle immagini.

I filtri dei primi strati individuano forme primitive, quelli successvi imparano a distinguere forme sempre più grandi e complesse. Una proprietà della convoluzione è l'equivarianza alla traslazione: immagini traslate sono interpretate allo stesso modo e i valori delle mappa di attivazione traslano con i valori di input. Ciò significa che forme particolari generano *feature maps* simili, indipendentemente dalla loro collocazione nell'immagine.

Una delle proprietà della convoluzione è la seguente. Una convoluzione sullo strato q incrementa il campo recettivo di una *feature* dallo strato q allo strato $q + 1$. In altre parole, ogni valore della mappa di attivazione dello strato successivo cattura una regione spaziale più ampia del precedente. *Feature maps* degli strati catturano aspetti caratteristici di regioni via via maggiori e questo è il motivo per cui le CNN possono essere definite "profonde": per studiare l'intera immagine sono necessarie lunghe successioni di "blocchi" di strati. [6] In figura, un esempio dell'operazione di convoluzione.

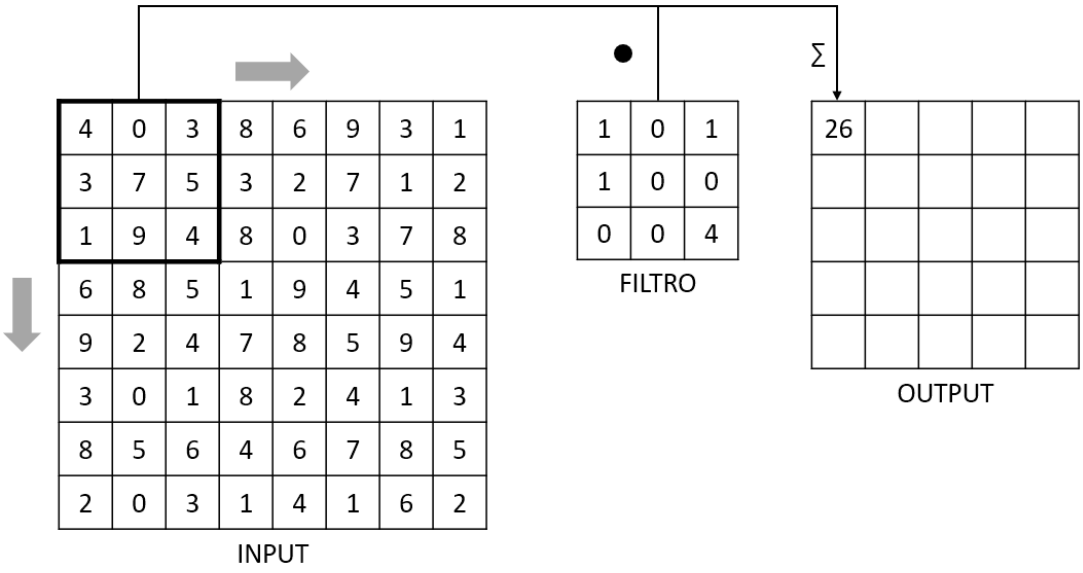


Figura 2.2: Convoluzione: dimensione 3, *stride* 1.

Padding L'operazione di convoluzione comporta una contrazione dello strato q rispetto a $q + 1$ ed una conseguente perdita di informazioni. Il problema può essere arginato utilizzando il cosiddetto *padding*, una tecnica che prevede l'aggiunta di $\frac{F-1}{2}$ pixel ai bordi delle mappe di attivazione per mantenere l'impronta spaziale. Ovviamente, per non alterare l'informazione, ai pixel sono assegnati valori nulli. Il risultato è un incremento delle dimensioni (altezza e larghezza) del volume di input di $F - 1$, esattamente la quantità di cui è ridotto a seguito della convoluzione.

Essendo il prodotto zero, le regioni esterne soggette a padding non contribuiscono al risultato finale del prodotto scalare. Ciò che invece accade è permettere al filtro convoluzionale di scavalcare i bordi dello strato e computare il prodotto scalare solamente per le celle di valori diversi da 0. Questa tipologia di padding è definita "half-padding" in quanto circa metà del filtro oltrepassa i bordi, quando collocato alle estremità. L'half-padding è utilizzato per mantenere il "footprint" spaziale.

Quando il padding non è utilizzato, si parla semplicemente di *valid-padding* e nella pratica non dà buoni risultati per il seguente motivo: mentre con l'half-padding le celle ai bordi contribuiscono all'informazione, nel caso di valid-padding, queste non vedono il passaggio del filtro e sono sotto-rappresentate.

Un'altra forma di padding è il *full-padding*, con il quale si lascia che il filtro esuli completamente dal layer andando ad occupare celle di soli zeri. Così facendo si incrementa l'impronta spaziale dello strato, allo stesso modo in cui il valid-padding la riduce. [6]

Strides Un filtro convoluzionale computa il prodotto scalare in ogni singola posizione dello strato di input ma è altresì possibile limitare la computazione ad un numero inferiore di posizioni, facendo uso dello *stride* S. La convoluzione è allora applicata alle posizioni 1, $S+1$, $2S+1$ etc., lungo entrambe le dimensioni. L'output avrà altezza e larghezza, rispettivamente:

$$L_{q+1} = \frac{L_q - F}{S} + 1$$

$$B_{q+1} = \frac{B_q - F}{S} + 1$$

Ne consegue che lo stride comporta una riduzione delle dimensioni di un fattore di circa $1/S$, e dell'area di S^2. Generalmente si usano valori limitati a 1 o 2 mentre con *stride* maggiori si punta alla riduzione della richiesta di memoria. Tramite lo *stride* è possibile catturare pattern complessi in vaste porzioni di immagine, con risultati simili a quelli prodotti dal *max-pooling*.

In genere le dimensioni delle immagini in input sono ridotte a $L = B$, per evitare complicazioni nella definizione degli iperparametri. Per quanto concerne la convoluzione, il numero di filtri è solitamente potenza di 2 per facilitare la computazione, lo stride di 1 o 2, le dimensioni del filtro di 3 o 5. Filtri piccoli significano reti più profonde e più performanti.

Ogni filtro convoluzionale è infine associato ad un *bias*: dato un filtro p, ed il layer q, bias è indicato con $b(p, q)$. Il bias è un fattore di moltiplicazione della mappa di attivazione e la sua presenza incrementa il numero di parametri di un'unità. Il numero di *features* in ogni strato sarà dunque 1 + $L \times B \times d$. Come tutti gli altri parametri, il valore del bias è definito tramite retropropagazione in fase di addestramento. [6]

2.2.2 Funzione di attivazione ReLU

L'operazione di attivazione non-lineare segue l'operazione di convoluzione. Per ogni strato $L_q \times B_q \times d$, la funzione di attivazione genera uno strato di eguale dimensione $L_q \times B_q \times d$ di valori limitati da soglie: in quanto semplice mappatura uno-a-uno dei valori di attivazione, la funzione ReLU non altera l'importa spaziale dello strato.

L'attivazione avviene tramite funzioni matematiche. Mentre in passato la tangente iperbolica, la funzione sigmoide, *softsign* godevano di ampia diffusione, ora sono limitate a reti non-profonde e ormai rimpiazzate dalla funzione di attivazione ReLU (**Re**ctified **L**inear **U**nit). Il motivo principale è che in reti neurali profonde, il gradiente di queste tre funzioni di attivazione si annulla durante la retropropagazione ed impedisce all'algoritmo di proseguire con l'addestramento. Inoltre, la funzione ReLU è computazionalmente molto più efficiente. ReLU è definita come segue:

$$f_{relu}(x) = max(0, x)$$

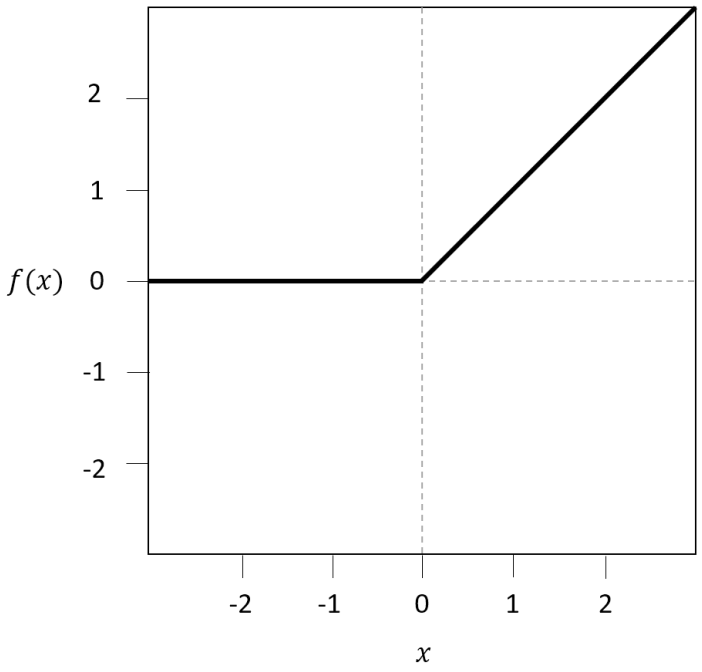


Figura 2.3: Funzione di attivazione ReLU.

ReLU è di fatto una funzione di rimozione di valori negativi rivelatasi efficace ed efficiente: la sua derivata in $\mathbb{R}+$ è sempre 1 e non satura in $\mathbb{R}$; in altre parole, il codominio della funzione è $[0, inf)$. In corrispondenza dell'origine, la funzione non approssima l'identità e genera invece un gradiente elevato impedendone la scomparsa. [5]

2.2.3 Strato di Pooling

Il *max-pooling* estrae il massimo valore contenuto in matrici $P \times P$ di ogni mappa di attivazione e produce un altro *hidden layer* di eguale profondità. Anche in questo caso, come per la convoluzione, si fa uso di *stride*: se lo *stride* è 1 lo strato 2 così generato avrà dimensioni:

$$L_2 = L_1 - P + 1$$

$$B_2 = B_1 - P + 1$$

$$d_2 = d_1.$$

Per *stride* maggiori di 1, come solitamente si usa, altezza e larghezza saranno, rispettivamente:

$$L_2 = \frac{L_1 - P}{S} + 1$$

$$B_2 = \frac{B_1 - P}{S} + 1$$

Rispetto alla convoluzione, il *pooling* è effettuato al livello di ciascuna mappa di attivazione, quindi il loro numero rimane alterato e l'output è uno strato della stessa profondità (ma di altezza e larghezza differenti). Una configurazione tipica è dimensione 2×2 e *stride* 2: così facendo non c'è sovrapposizione tra regioni.

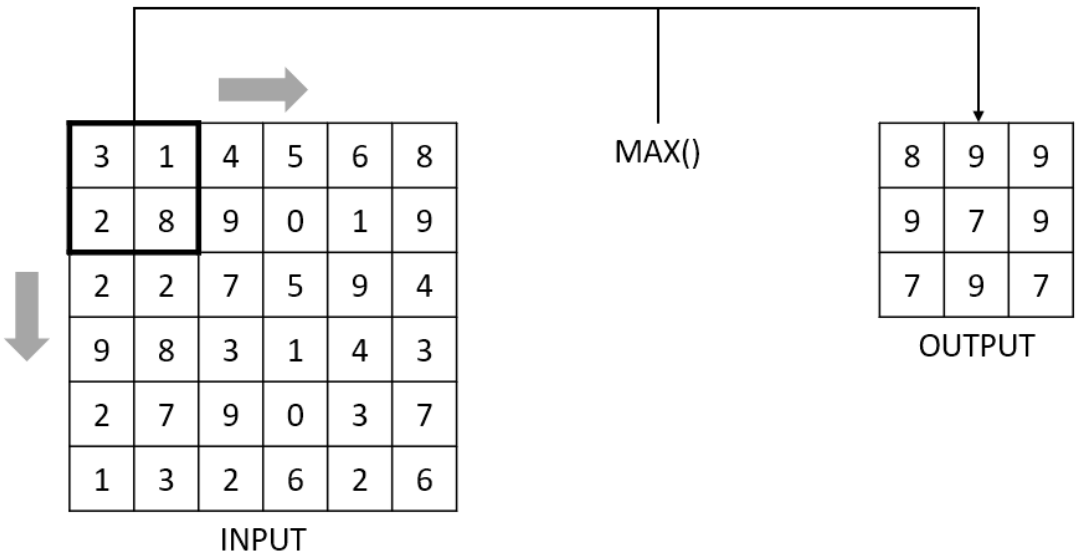


Figura 2.4: *Max-pooling*: dimensione 2, *stride* 2.

Esistono altre varianti del pooling, meno diffuse, come ad esempio *l'average-pooling*, utilizzato da LeNet-5 e denominato *subsampling*, termine con il quale ora ci si riferisce ad una generica operazione di riduzione dell'impronta spaziale.

L'uso dello *stride* nel pooling è importante per tre motivi. Il primo è la riduzione dell'impronta spaziale delle mappe di attivazione, il secondo è un certo grado di invarianza alla traslazione e il terzo è un incremento del campo ricettivo. Si osservi che per ridurre l'impronta spaziale possono essere utilizzati unicamente strati convoluzionali con *stride* maggiori di 1. Nonostante ciò, si preferisce tuttora utilizzare *max-pooling* o qualche altra variante dato il grado di non-linearità e invarianza alla traslazione che introducono. [6]

2.2.4 Strato completamente connesso

Connette ogni *feature* in input con la corrispondente *feature* in output. Presenta la struttura di una rete *feed-forward* tradizionale e aumenta a dismisura il numero di parametri addestrabili per effetto del numero di connessioni. Ad esempio, se due strati completamente connessi hanno 4096 unità ciascuno, il numero di connessioni (quindi di parametri) sarà superiore a 16 milioni. [6]

2.3 Architetture convoluzionali

Di seguito sono presentate alcune architetture convoluzionali di ampio successo e che maggiormente hanno contribuito all'avanzamento della ricerca. Due architetture convoluzionali di prima generazione e molto semplici sono LeNet e AlexNet; seguono VGG e ResNet, due modelli più profondi e complessi. La rete proposta segue le "linee guida" tracciate da queste reti. ResNet sarà invece utilizzato come encoder nell'implementazione di DeepLab qui considerata.

2.3.1 LeNet

LeNet è una rete originariamente intesa per il riconoscimento di cifre scritte a mano. Il modello non è profondo ma ha portato alla luce i principi ora alla base dei *deep network* moderni. La rete si compone di due strati convoluzionali, due di *pooling* e tre *fully connected*. In figura la rappresentazione DAG (**D**irected **A**cyclic **G**raph) della rete, seguita da una rappresentazione grafica di facile intuizione. Nella rappresentazione DAG, C(a,b) è uno strato convoluzionale di profondità (numero di filtri) a e dimensione b×b, P(c,d) è uno strato di *pooling* di *stride* c e dimensione d×d.

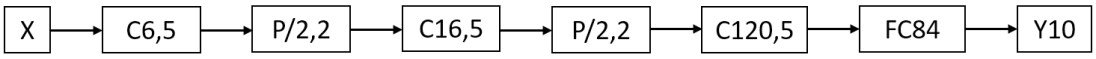


Figura 2.5: Rappresentazione DAG di LeNet. [5]

L'input della rete sono immagini in formato *grayscale* 32×32. Dunque, output della prima convoluzione 5×5 sono 6 *feature maps* di di dimensione 28×28, da $L_2 = L_1 - F + 1$. Al primo strato convoluzionale segue il primo di *pooling*, il quale compie un *subsampling* a 14×14. Seguono la seconda convoluzione 5×5 con 16 filtri ed il secondo pooling (2×2, stride 2) l'output è una mappa di attivazione di dimensione 5×5 e profondità 16. Viene poi un ultimo strato convoluzionale di 120 filtri 5×5. Lo strato finale è detto di classificazione e si compone di 10 neuroni, ciascuno corrispondente ad una singola cifra.

Un aspetto di grande importanza nella visione d'insieme dei modelli convoluzionali è il numero di parametri addestrabili. La quantità precisa è calcolata sommando

tra loro i contributi di ciascuno strato convoluzionale e completamente connesso come indicato. ReLU e Pooling non hanno parametri addestrabili.

- $(F \times F \times d + 1) \times N$, per lo strato convoluzionale
- $n \times N + n$, per lo strato completamente connesso

Dove F sono le dimensioni del filtro convoluzionale, d la sua profondità, N il numero di mappe di attivazione; il termine unitario corrisponde al bias. P ed S sono rispettivamente la dimesnione e lo stride dello strato di pooling, n il numero di neuroni dello strato completamente connesso. LeNet risulta avere 61750 parametri. [5]

2.3.2 AlexNet

AlexNet si distingue da altri modelli convoluzionali per l'impiego dei due blocchi *Slice* e *Concat*. Il primo strato di convoluzione applica 96 filtri di dimensioni 11×11 con stride 4 su immagini 224×224×3. Seguono ReLU e pooling 3×3 con stride 2. Le 96 mappe di attivazione sono suddivise in due gruppi da 48 (blocco *Slice*), passate attraverso due distinti strati convoluzionali 5×5 di profondità 128, quindi nuovamente riconnesse (blocco *Concat*). Segue un secondo blocco di convoluzioni intervallate da *Slice* e *Concat*, quindi due strati *fully connected* di 4096 neuroni ciascuno ed infine lo strato di ouput, anch'esso completamente connesso, di 1000 unità. AlexNet ha in totale 60,965,224 parametri addestrabili. [5]

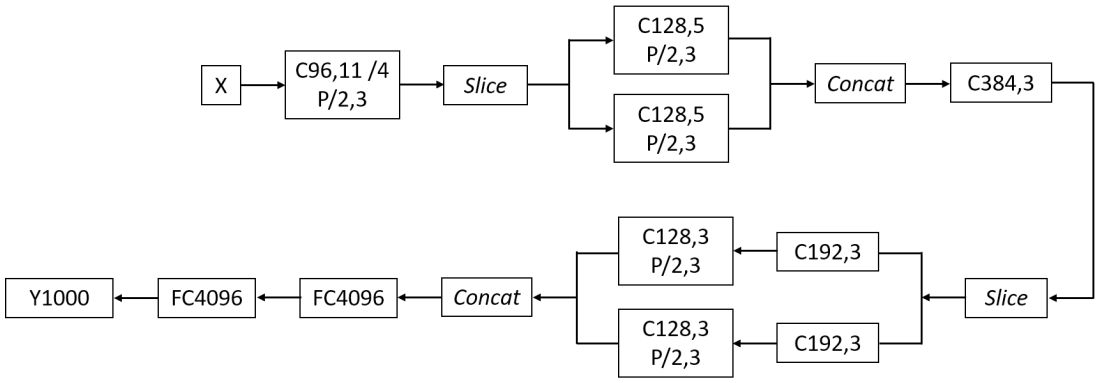


Figura 2.6: Rappresentazione DAG di AlexNet. [5]

2.3.3 VGG

Le architetture VGG sono una categoria di reti neurali convoluzionali di profondità variabile, compresa fra 11 e 19 strati. Le più performanti si sono rivelate essere quelle con 16 o più, cioè VGG-D e VGG-E, denominate VGG16 e VGG19.

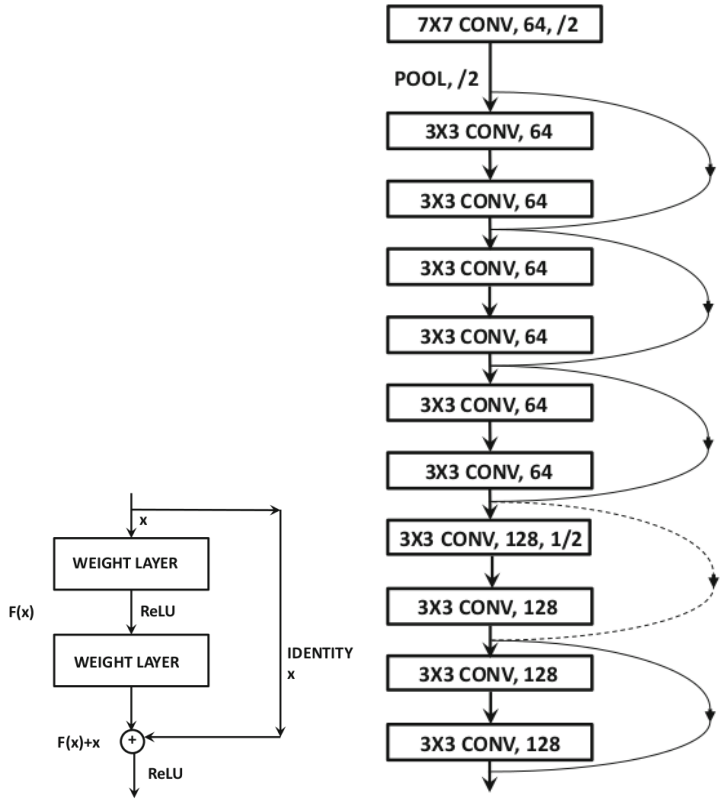
Le reti VGG adottano filtri convoluzionali di dimensioni ridotte, per il motivo seguente. Si supponga di effettuare convoluzione con un filtro 7×7; l'impronta del filtro sull'immagine in input è 7×7. La stessa impronta la si può ottenere con una serie di tre convoluzioni 3×3, ottenendo un numero di parametri inferiore: da 7×7=49 a 3×3×3=27, per descrivere la stessa porzione di input. Al fine di captare aree sempre maggiori è dunque necessario disporre più strati convoluzionali in successione, fino a raggiungere la dimensione desiderata e catturando così geometrie via via più grandi e complesse, con il vantaggio di un numero contenuto di parametri. Questo avviene poichè i parametri crescono più velocemente con il quadrato di F×F che non con la profondità. Sulla base di queste premesse, VGG fa uso di filtri convoluzionali 3×3 con padding 1 e pooling di 2×2, con stride 2. In tal modo la convoluzione mantiene costanti le dimensioni ed il subsampling è affidato allo strato di pooling. [6]

2.3.4 ResNet

ResNet è una rete neurale convoluzionale di 157 strati, quasi un ordine di grandezza in più rispetto a VGG. Come descritto in precedenza, gli strati più alti di una CNN sono contraddistinti da filtri in grado di individuare forme semplici, come contorni e spigoli, mentre gli strati più profondi imparano a riconoscere caratteristiche più complesse, come texture e figure geometriche complesse. Laddove sole forme semplici fossero presenti, l'algoritmo di retropropagazione dovrà comunque percorrere l'intera rete nella sua profondità, con notevole rallentamento della convergenza. Prestazioni simili in inferenza potrebbero essere raggiunte con l'addestramento di una rete molto meno profonda, adatta alle circostanze imposte dall'input.

ResNet risolve il problema mediante le cosiddette *skip connections*, secondo quella che è definita una vista "iterativa", e non più gerarchica, della rete. Le reti feed-forward tradizionali sono caratterizzate da connessioni dirette tra uno strato i ed il successivo

$i + 1$. ResNet, invece, connette tra loro strati i ed $i + r$ (con $r > 1$) sommandone i contributi. Qualora le dimensioni delle mappe di attivazione non coincidessero, sono applicate semplici convoluzioni 1×1 tramite le cosiddette matrici di proiezione, secondo filtri i cui valori saranno definiti in fase di addestramento. Il salto di strati consente al gradiente di fluire rapido ed indisturbato: è possibile immaginare le *skip connection* come autostrade per il gradiente, propagato all'indietro lungo il percorso appropriato attraverso unità di base definite *residual modules*. Ciò conferisce all'algoritmo un buon grado di flessibilità: se le geometrie sono semplici, esso salterà le connessioni, in caso contrario sarà costretto a percorrerle. Si parla di *residual learning*: l'addestramento attraverso percorsi più lunghi è una sorta di *fine-tuning* dell'intera rete ed avviene solamente quando necessario. [6]



(a) *Residual module* con *skip connection* **(b)** ResNet, architettura parziale

Figura 2.7: *Residual module* e primi strati di ResNet. [6]

Modelli pre-addestrati Uno dei punti di forza dei dati sotto forma di immagini è l'omogeneità delle *features*: contorni, spigoli, pattern sono parte integrante di qualunque immagine e sono le caratteristiche che i primi strati convoluzionali, operanti in aree molto piccole delle immagini in input, imparano a riconoscere. Per questo motivo le CNN non sono solitamente addestrate da zero ma si fa uso di reti dette "pre-addestrate", ossia modelli allenati su dataset appositi. Uno dei dataset in questione è ImageNet, contentente più di un milione di immagini suddivise in 1000 classi. In classificazione di oggetti in genere si utilizza una rete pre-addestrata su ImageNet e si sostituiscono gli ultimi strati, inclusi quelli dipendenti dal numero di classi. Ciò permette all'algoritmo di retropropagazione di effettuare l'addestramento solamente sugli strati più profondi, quelli cioè che cattureranno le *features* caratterizzanti. [6] In segmentazione semantica questo approccio non è attuabile, in quanto semplici modelli convoluzionali non sono adatti al problema: *max-pooling* ed in generale il subsampling riducono la risoluzione delle mappe di attivazione impedendo alla rete di fornire risultati *pixel-level*. Si utilizzano allora architetture apposite di tipo encoder-decoder, come appunto DeepLabv3+, le quali però adottano, in qualità di encoder, proprio modelli pre-addestrati.

2.4 L'architettura encoder-decoder

La tipologia di CNN di maggior successo e largo impiego nel campo della segmentazione semantica, e l'unica utilizzata fino ad ora, è l'**autoencoder convoluzionale**. Esso effettua una compressione delle immagini, seguita da una decompressione, facendole passare prima attraverso strati convoluzionali e di *pooling*, poi *de-convoluzionali* e di *un-pooling*. La deconvoluzione è definita anche *transposed convolution*, in quanto il filtro dello strato non è altro che un tensore di convoluzione trasposto e invertito. La struttura di base di un autoencoder è la seguente:

$$\text{Conv.} \rightarrow \text{ReLU} \rightarrow \text{Pooling} \rightarrow \ldots \rightarrow \text{Unpooling} \rightarrow \text{ReLU} \rightarrow \text{Deconv.}$$

Esistono molteplici varianti dell'architettura di base, generalmente di profondità

assai maggiore e spesso antisimmetici. La simmetria facilita però la fase di design della rete: ogni strato convoluzionale e il corrispondente deconvoluzionale avranno la stessa dimensione e la stessa profondità. [6] La particolarità delle reti encoder-decoder risiede nell'uguaglianza tra dimensioni di input e di output, necessaria alla classificazione *pixel-level*. Un esempio di architettura encoder-decoder è SegNet.

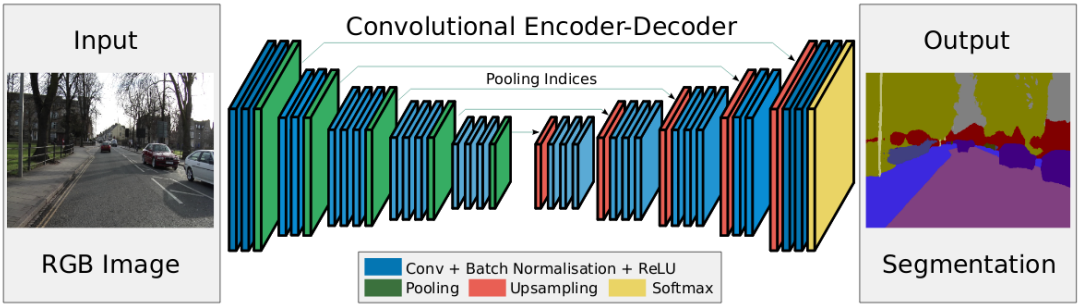


Figura 2.8: Architettura di Segnet. [7]

2.4.1 Architettura proposta

I principi dell'architettura encoder-decoder sono a fondamento dell'architettura qui proposta. Essa trae ispirazione dall'autoencoder convoluzionale Segnet ed è stata sviluppata tramite l'applicazione *Deep Network Designer*.

A tal proposito, è doveroso introdurre alcuni ulteriori strati: oltre a convoluzione, ReLU e *pooling*, le reti di segmentazione semantica fanno uso degli strati seguenti.

Transposed Convolution layer Lo strato deconvoluzionale compie un'operazione molto simile a quella della convoluzione, come visualizzato in figura. Al contrario di quanto prodotto dallo strato da cui prende il nome, la deconvoluzione genera una mappa di attivazione di dimensioni maggiori. In figura è mostrato un esempio di deconvoluzione simile a quello adottato nel modello creato: filtro 3×3, *stride* 2.

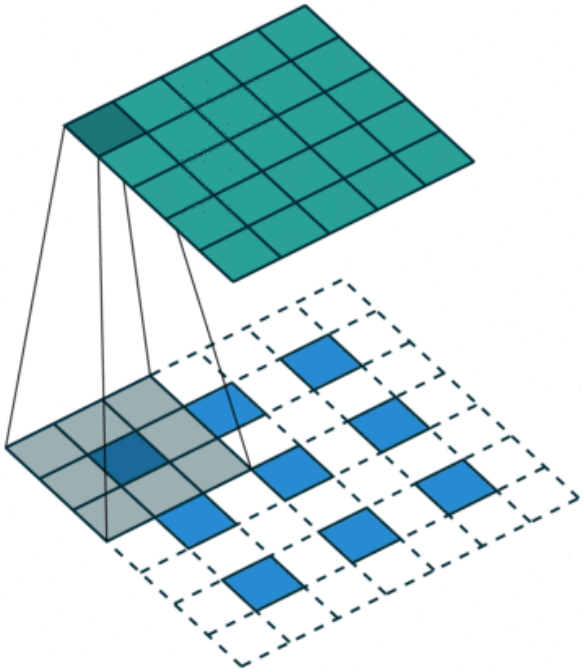


Figura 2.9: *Transposed convolution* con *padding* e *stride*. [18]

Batch Normalization layer Strato di normalizzazione di ciascun canale di input attraverso il *mini-batch*, velocizza l'addestramento e riduce la sensibilità all'inizializzazione. È inserito tra convoluzione e ReLU. Lo strato normalizza le attivazioni dei canali sottraendo la media del *mini-batch* e dividendo per la devizione standard. Per evitare ampie variazioni dei valori delle attivazioni, lo strato implementa due parametri addestrabili, β e γ, i quali operano sull'output uno *shift* ed una variazione di scala. [14]

Softmax layer Applica all'input la funzione *softmax*. *Softmax* associa una probabilità ad ogni classe, riducendo gli output a valori compresi fra 0 e 1. Lo strato softmax precede lo strato di classificazione, sia esso per classificazione pura o segmentazione. [14]

Pixel Classification layer Segue la funzione softmax e dà in output la matrice di classificazione, i cui elementi sono le etichette di ciascun pixel. Qualora il dataset fosse sbilanciato, fenomeno verificatosi in entrambi i casi in esame, è opportuno ag-

giornare lo strato con i parametri relativi ai pesi delle classi. Lo strato di classificazione computa inoltre la *loss function*, utilizzata dall'algoritmo di retropropagazione nell'aggiornamento dei pesi. [14]

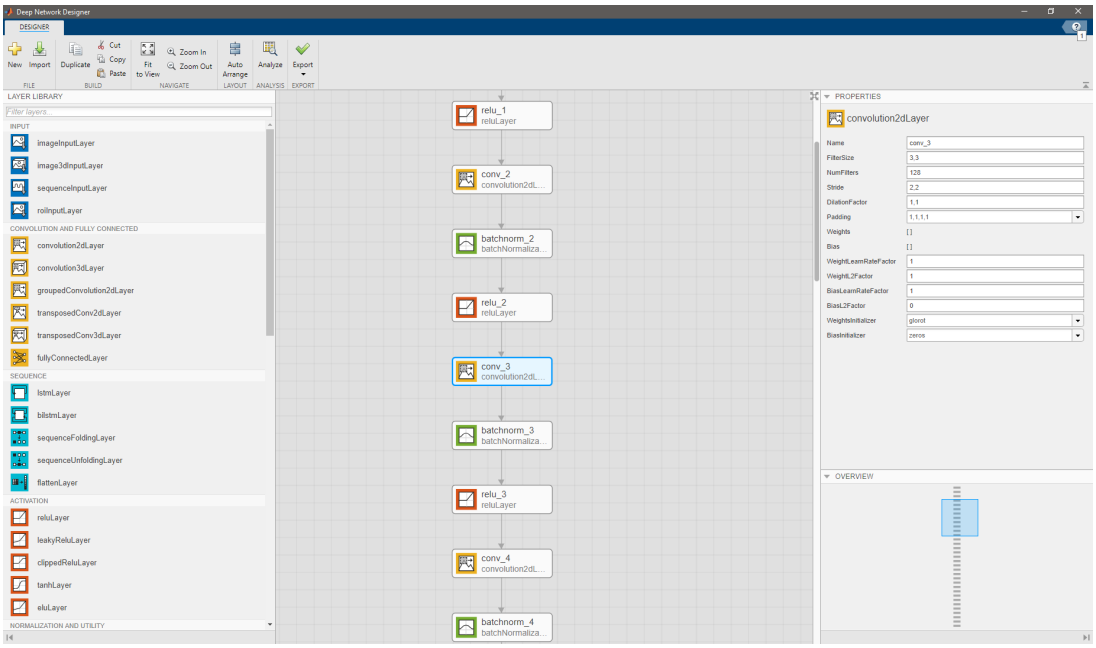


Figura 2.10: Finestra principale dell'applicazione *Deep Neural Network Designer*.

Il modello creato si compone di un totale di 33 strati e 32 connessioni. Il primo strato è lo strato di input, con il quale sono indicate le dimensioni delle immagini, per semplicità di implementazione fissate a 512×512×3. Segue una successione di 5 blocchi:

$$\text{Convoluzione} \rightarrow \text{Batch Normalization} \rightarrow \text{ReLU}$$

costituenti l'encoder. Vi è poi il decoder, i cui blocchi, anch'essi in numero di 5, sono costituiti da:

$$\text{Deconvoluzione} \rightarrow \text{Batch Normalization} \rightarrow \text{ReLU}$$

I filtri convoluzionali e deconvoluzionali hanno tutti dimensione 3×3 con *stride* 2 e, per la sola convoluzione, *padding* (1, 1, 1, 1), corrispondente a due colonne e due

righe di zeri aggiunti in corrsipodenza dei bordi dell'input. Il numero di filtri incrementa di un fattore 2 ad ogni blocco. Eventuali parametri degli strati di convoluzione e deconvoluzione sono lasciati inalterati. Il modello è chiuso dalla coppia *Softmax-Classificazione*.

Si osservi come l'architettura proposta non abbia alcuno strato di Pooling o di Unpooling: downsampling e upsampling sono affidati agli strati di convoluzione e deconvoluzione, rispettivamente.

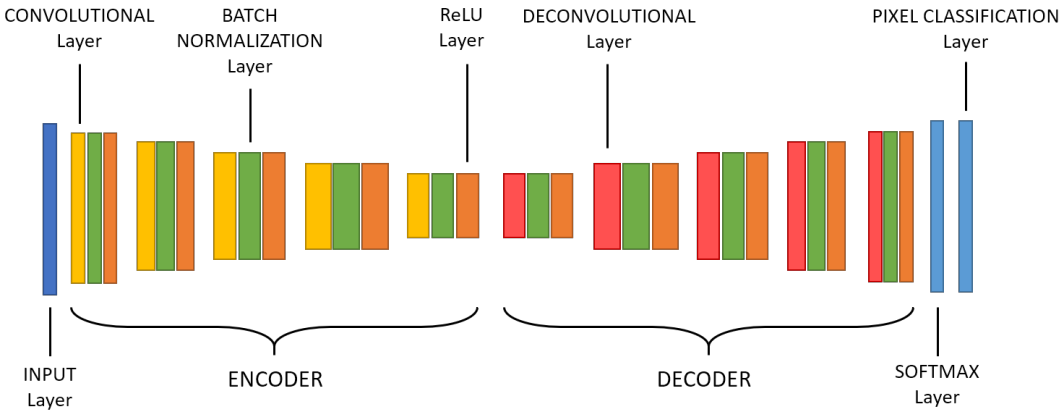


Figura 2.11: Visione d'insieme del modello proposto.

Tabella 2.1: Tabella degli strati della rete proposta.

N.	Strato	Dimensioni filtro	Canali	Attivazioni
1	Input	-	-	512×512×3
2	Convoluzionale	3×3×3	32	256×256×32
3	Batch Normalization	-	32	256×256×32
4	ReLU	-	-	256×256×32
5	Convoluzionale	3×3×32	64	128×128×64
6	Batch Normalization	-	64	128×128×64
7	ReLU	-		128×128×64
8	Convoluzionale	3×3×64	128	64×64×128
9	Batch Normalization	-	128	64×64×128
10	ReLU	-	-	64×64×128
11	Convoluzionale	3×3×128	256	32×32×256
12	Batch Normalization	-	256	32×32×256
13	ReLU	-	-	32×32×256
14	Convoluzionale	3×3×256	512	16×16×512
15	Batch Normalization	-	512	16×16×512
16	ReLU	-	-	16×16×512
17	Deconvoluzionale	3×3×512	512	32×32×512
18	Batch Normalization	-	512	32×32×512
19	ReLU	-	-	32×32×512
20	Deconvoluzionale	3×3×512	256	64×64×256
21	Batch Normalization	-	256	64×64×256
22	ReLU	-	-	64×64×256
23	Deconvoluzionale	3×3×256	128	128×128×128
24	Batch Normalization	-	128	128×128×128
25	ReLU	-	-	128×128×128
26	Deconvoluzionale	3×3×128	64	256×256×64
27	Batch Normalization	-	64	256×256×64
28	ReLU	-	-	256×256×64
29	Deconvoluzionale	3×3×64	3	512×512×3
30	Batch Normalization	-	3	512×512×3
31	ReLU	-	-	512×512×3
32	Softmax	-	3	512×512×3
33	Pixel classification	-	-	-

2.4.2 DeepLabv3+

Una delle quattro architetture presenti in MATLAB è DeepLabv3+ e tra quelle tuttora disponibili è una delle più performanti. È stata scelta come rete di segmentazione, a fianco del modello proposto. DeepLabv3+ è l'iterazione più recente dell'architettura sviluppata da Google ed è caratterizzata da:

- **A**trous **S**patial **P**yramid **P**ooling (ASPP)
- Architettura encoder-decoder

Il primo aspetto riguarda una particolare modalità di combinazione di strati di convoluzione *atrous* e *depthwise*, con la quale il modello cattura e concatena *features* a scale differenti. Al contempo, essa riduce la complessità computazionale mantenendo alte prestazioni.

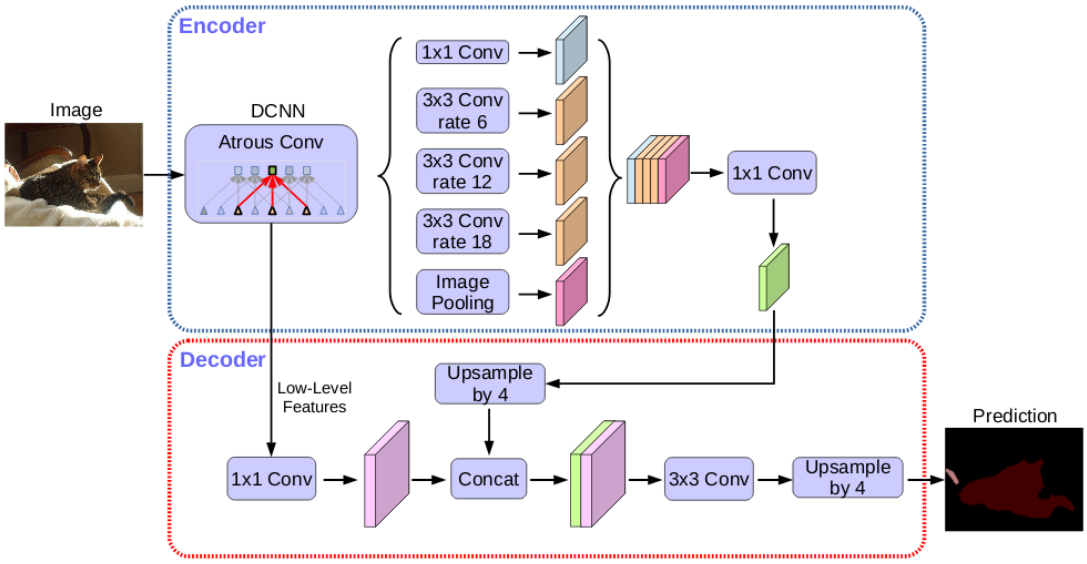


Figura 2.12: Architettura di DeepLabv3+. [10]

Il secondo aspetto concerne il modo con cui l'architettura encoder-decoder è stata interpretata. L'encoder opera un primo *upsampling* di un fattore 4 poi concatenato con le *features* di basso livello, le cui mappe sono sottoposte ad una convoluzione 1×1 per

ridurre il numero di canali. A seguito della concatenazione vi sono alcuni strati convoluzionali 3×3 ed un secondo strato di *upsampling* di *4*. Un decoder così configurato si è rivelato essere più performante di un diretto *upsampling* di *16*. [10]

L'implementazione di DeepLabv3+ in MATLAB può adottare encoder differenti, come ad esempio ResNet, Xception, etc. Qualora si utilizzasse ResNet, come nel caso in esame, il modello caricato si compone di semplici strati convoluzionali anziché di *depth separable convolutions* e ASPP.

2.5 Metriche di valutazione

Di seguito sono presentate le metriche di valutazione delle prestazioni dei modelli. La valutazione del modello su $\mathcal{X}_{test}$ è condotta tramite le cosiddette *classification metric functions*. [5] Le funzioni accettano in input i label di *ground truth* e quelli derivati dall'inferenza e ritornano un punteggio od un set di punteggi. Di seguito sono presentate le funzioni di classificazione più diffuse.

Accuratezza La più semplice funzione metrica per la classificazione è l'accuratezza di classificazione. Essa corrisponde alla frazione di campioni classificati correttamente. In un problema di segmentazione semantica:

- per ogni classe, l'accuratezza è il rapporto tra il numero di pixel correttamente classificati e il numero totale della singola classe, secondo quanto specificato dalla *ground truth*.
- per ogni immagine, il valore di accuratezza è l'accuratezza media di tutte le classi per la singola immagine.
- per l'intero dataset, il valore di accuratezza è l'accuratezza media di tutte le classi in tutte le immagini. [13]

L'accuratezza *globale* è invece la frazione di pixel correttamente classificati, indipendentemente dalla classe.

In generale, l'accuratezza esprime una rozza valutazione della bontà del modello; è indicativa e non affidabile, soprattutto in presenza di dataset sbilanciati. [5]

Matrice di confusione La matrice di confusione è un potente e efficace strumento di valutazione. Per un problema di classificazione con C classi, la matrice di confusione **M** è una matrice CxC dove l'elemento M_{ij} corrisponde al numero di campioni definiti come i ma classificati dalla rete come j. M_{ii} corrisponde invece ai campioni correttamente classificati. La matrice di confusione più semplice è quella di un problema di classificazione binario; supponedo di avere due etichette, -1 e 1, la matrice di confusione è la seguente:

		Previsione	
		1	**-1**
Etichetta	**1**	Vero positivo	Falso negativo
	-1	Falso positivo	Vero negativo

Figura 2.13: Matrice di confusione, due classi.

M_{11} corrisponde ai campioni etichettati come 1 e classificati come 1, questi elementi sono definiti veri-positivi (TP, dall'inglese *true-positive*). L'elemento M_{12} corrisponde ai campioni classificati come 1 ma etichettati come -1, si parla allora di falsi-negativi (FN). Similmente, M_{21} sono i falsi-positivi (FP) ed infine M_{22} i veri-negativi (TN). Per un problema di classificazione a 5 classi la matrice è:

		Previsione				
		A	**B**	**C**	**D**	**E**
Etichetta	**A**		FP			
	B	FN	VP	FN	FN	FN
	C		FP			
	D		FP			
	E		FP			

Figura 2.14: Matrice di confusione, cinque classi.

L'accuratezza è quindi espressa come:

$$accuratezza = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

dalla quale si deduce che un modello è un classificatore ideale se FP = FN = 0. La matrice di confusione è di diretta applicazione anche per problemi di classificazione multi-classe. In questo caso la rete è un classificatore perfetto se tutti gli elementi non-diagonali sono 0. FN ed FP sono allora definiti come:

$$\mathrm{FN_i} = \sum_{j \neq i} M_{ij}$$

$$\mathrm{FP_i} = \sum_{j \neq i} M_{ji}$$

e l'accuratezza

$$\mathrm{FP_i} = \frac{\sum_i M_{ii}}{\sum_i \sum_j M_{ji}}$$

In generale la matrice di confusione permette un'immediata visualizzazione delle prestazioni del modello qualora il numero di classi coivolte fosse limitato. In problemi di classificazione di oggetti, dove le classi possono essere attorno le centinaia, diviene difficile, se non impossibile, estrapolare informazioni utili al fine di una prima, superficiale valutazione. [5]

Precisione e recupero Due espressioni utili a semplificare la lettura della matrice di confusione sono: precisione e recupero, definiti come:

$$precisione = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

$$recupero = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Se FN ed FP sono zero, caso di un classificatore ideale, precisione e recupero saranno unitari. Da un buon modello ci si aspettano dunque valori prossimi all'unità. Nel caso di classificazione multiclasse si ha, per la classe i:

$$precisione = \frac{M_{ii}}{\sum_j M_{ji}}$$

$$recupero = \frac{M_{ii}}{\sum_j M_{ij}}$$

In generale si avranno:

$$precisione = \sum_{i=1}^{C} w_i \times precisione_i$$

$$recupero = \sum_{i=1}^{C} w_i \times recupero_i$$

dove w_i sono i pesi delle classi. [5]

F1 *Score* Precisione e richiamo sono parametri assai utili per valutare un modello ma solitamente si preferisce associare alle reti un unico valore. A tal fine, si può computare la media armonica dei due valori, come segue:

$$\mathrm{F1} = \frac{2}{\frac{1}{precisione} + \frac{1}{recupero}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

F1 è compreso tra 0 ed 1, dove 1 indica un risultato perfetto. Per un problema multi-classe, è la media dei valori F1 di ciascuna classe. In relazione alla segmentazione semantica, questo parametro è indice della vicinanza tra contorno tracciato e contorno segmentato. Nello specifico:

- per ogni classe, l'F1 *score* medio è la media degli F1 della classe su tutte le immagini.
- per ogni immagine, l'F1 *score* medio è la media degli F1 di tutte le classi per la singola immagine.
- per l'intero dataset, l'F1 *score* medio è la media degli F1 di tutte le classi in tutte le immagini. [13]

IoU *Intersection over Union*, noto anche come coefficiente di similarità di Jaccard, è il parametro di valutazione più usato.

- per ogni classe, l'IoU è il rapporto tra pixel correttamente classificati e la somma del numero complessivo di pixel etichettati e di pixel classificati:

$$\mathrm{IoU} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP} + \mathrm{FN})}$$

- per ogni immagine, l'IoU medio (mIoU) è la media dei valori di tutte le classi per la singola immagine.
- per l'intero dataset, mIoU è la media dei valori di tute le classi in tutte le immagini. [13]

Qualora il dataset fosse particolarmente sbilanciato, è possibile calcolare un valore di mIoU pesato secondo il numero di pixel totale di ogni singola classe. Lo IoU, anch'esso variabile da 0 ad 1, non soffre del problema da cui è afflitta l'accuratezza ed è il singolo parametro che meglio descrive la qualità della segmentazione.

CAPITOLO 3

Preparazione del dataset

L'addestramento di una generica rete neurale prevede la creazione di un dataset sulla base della procedura:

1. Definizione delle classi
2. *Labeling*
3. *Data augmentation*
4. Suddivisione del dataset

Ciascuno *step* è illustrato in dettaglio nei paragrafi seguenti.

3.1 Definizione delle classi

La definizione delle classi è una scelta immediata in classificazione di oggetti, ed una più difficile in segmentazione semantica. La segmentazione semantica trova applicazione principalmente in due campi: navigazione autonoma in ambiente urbano e immagini biomediche, entrambi contraddistinti dalla presenza di elementi ben distinguibili l'uno dall'altro e descritti da termini univoci. Nell'ambito della navigazione di rover marziani, la definizione delle classi presenta invece difficoltà maggiori. In primo luogo, è necessario tener presente che un numero contenuto di classi facilita l'inferenza, riducendo le ambiguità; qualora emergessero dubbi sull'aggiunta di un'ulteriore classe è opportuno ponderare attentamente la decisione. Ciò comporta la necessità di

raggruppare tra loro terreni di natura differente che l'occhio umano tenderebbe a differenziare. È inoltre doveroso precisare che la classificazione del terreno non è condotta strettamente in base al tipo di roccia, come potrebbe invece avvenire per *task* scientifici: la coppia antitetica in gioco nella scelta delle classi è costituita da due attributi fondamentali del terreno, riassumibili in a) apparenza visiva e b) pericolosità. La scelta avviene tenendo conto dei due aspetti, senza esibire preferenze: una classe contraddistinta da eccessiva variabilità nella geometria può ostacolare il raggiungimento di buone accuratezze in fase di *training*, mentre una associata a rocce di un'unica tipologia può portare ad incoerenze nella distinzione fra terreno attraversabile e non-attraversabile.

Prima della definizione delle classi è necessario osservare attentamente le immagini a disposizione tenendo traccia dei terreni presenti, quindi si redige una lista di possibli classi e loro descrizione. Scelte le classi e dato avvio al *labeling*, è possibile, anche se sconsigliato, rimuoverne od aggiungerne.

3.2 Labeling

L'addestramento di una rete neurale convoluzionale richiede in input un dataset composto da due distinti insiemi di immagini: del primo fanno parte le immagini da segmentare, del secondo le cosiddette *ground-truth label mask*, ossia matrici W$\times$H di tipo *categorical* i cui elementi sono le etichette associate al pixel corrispondente. Le maschere sono il prodotto dell'operazione di *labeling*, ossia l'assegnazione dell'etichetta ad un insieme di pixel, condotta manualmente su ogni singola immagine del dataset. La classificazione è effettuata tramite applicazioni realizzate appositamente, qui **Image Labeler**, fornita dal toolbox MATLAB *Image Processing*.

Il primo passo consiste nella creazione delle classi e nella scelta della tipologia di etichetta, la quale può essere: linea, rettangolo, pixel label. Come è logico, l'unica categoria di label qui rilevante è *pixel label*, la quale è quindi selezionata per ogni classe. Per ogni immagine sono tracciati i perimetri delle aree con lo strumento grafico che più si addice alla geometria.

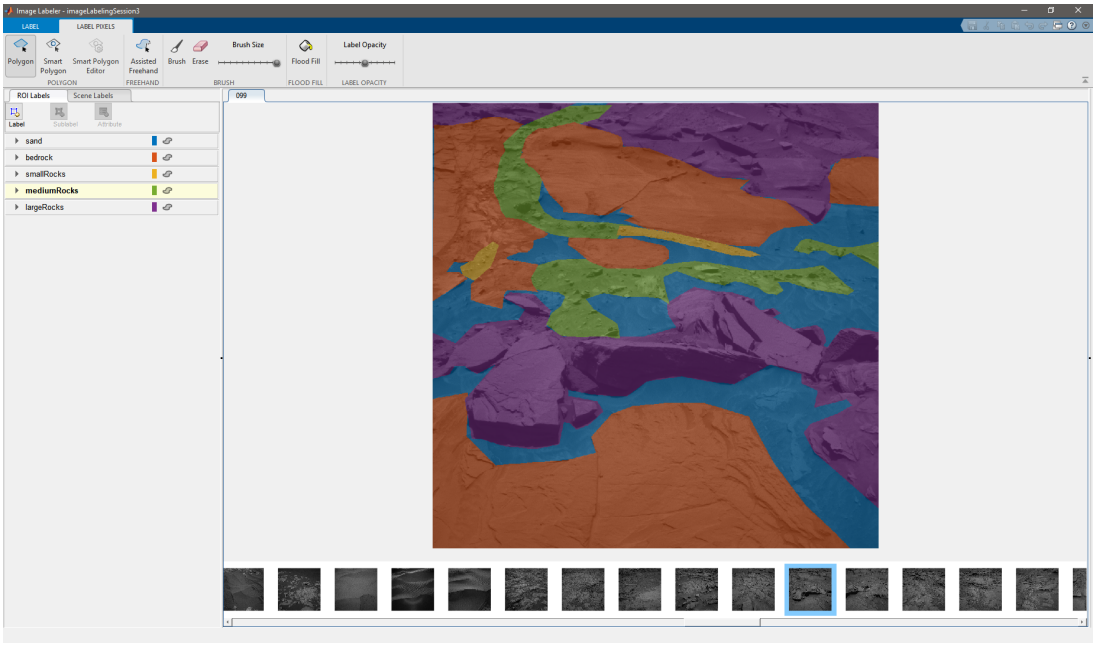


Figura 3.1: Schermata dell'applicazione *Image Labeler*.

Lo strumento *label summary* genera due distinti grafici. Il primo rappresenta la percentuale di area dell'immagine associata a ciascuna classe per ogni campione ed è utile al fine di valutare, in prima istanza, un eventuale sbilanciamento. A tal proposito, si noti come `largeRocks` sia presente in percentuali ridotte ed in percentuale ancora inferiore sull'intera sequenza di immagini. Il secondo offre una visione d'insieme dell'intera segmentazione e si è rivelato adatto all'individuazione di eventuali immagini o porzioni di immagini non ancora etichettate.

Figura 3.2: *Label summary.*

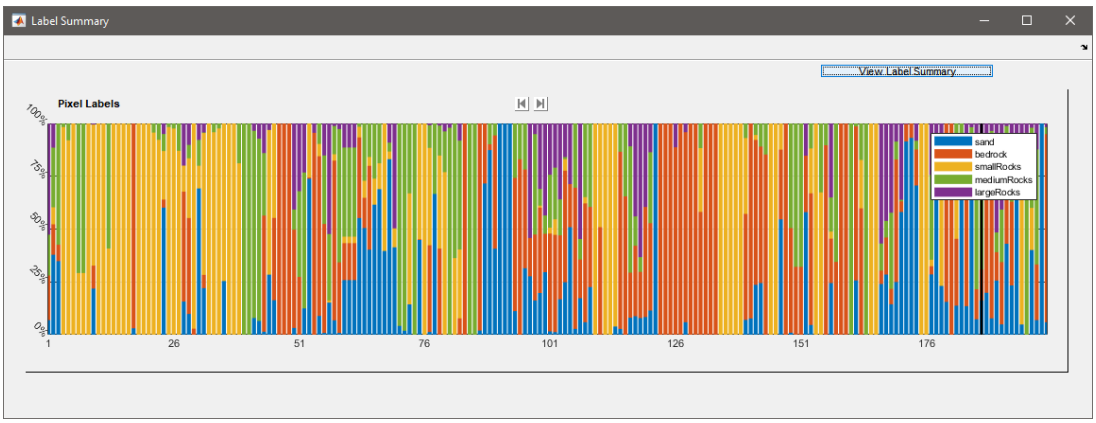


Figura 3.3: *Label summary*. Visione d'insieme.

L'operazione di *labeling* richiede molto tempo ed attenzione e alla lunga diviene esau-

stiva. Si propende quindi a segmentare manualmente una quantità di immagini ritenuta (appena) sufficiente; dataset creati per validazione di reti di segmentazione semantica ne contengono generalmente nell'ordine delle migliaia. Se l'applicazione lo permette, numeri elevati sono ottenibili con tecniche di *data augmentation*.

3.3 Data augmentation

Una buona rete neurale è contraddistinta dalla capacità di generalizzare, la quale è garantita dall'allenamento su un dataset sufficientemente ampio e variegato. Qualora invece l'addestramento sia condotto su un insieme limitato, la rete memorizza le caratteristiche specifiche dei campioni (come la disposizione delle classi nelle immagini, se ve n'è una prevalente) e non è in grado di generalizzare: è il caso dell'*overfitting* (sovradattamento). Viene dunque spontaneo escogitare tecniche per arginarne gli effetti. Esse prendono il nome di *data augmentation* e consistono nella generazione di nuovi campioni a partire dagli originali. L'allenamento su un dataset così generato determina un notevole incremento dell'accuratezza delle reti e della loro resistenza a sostanziali variazioni dell'aspetto dei campioni in input. [5]

L'elaborazione delle immagini è un campo nel quale il *data augmentation* trova discreto successo in quanto trasformazioni quali traslazione, rotazione e riflessione non non alterano le proprietà sostanziali degli oggetti rappresentati. [5] Molte di queste tipologie di trasformazione non richiedono elevata potenza computazionale e possono essere eseguite direttamente in fase di addestramento, eccone alcune.

Smoothing *Smoothing* di immagini tramite filtri medi e/o gaussiani. Lo smoothing emula l'effetto generato su immagini catturate da fotocamere non a fuoco. Leggero incremento della tolleranza del modello ad effetti di *blurring*.

Motion Blur Fotocamere in movimento, in base ad ISO, *shutter speed* e velocità del veicolo possono generare immagini degradate. Non il caso di un rover marziano.

Median Filtering	Filtri di *smoothing*. Per ogni pixel dell'immagine, la media delle intensità dei pixel adiacenti, contenuti in un riquadro di dimensioni prefissate, sostituisce il valore originario. Potrebbe non produrre immagini realistiche.
Sharpening	Produce immagini dai contorni di maggior definizione. L'effetto è ottenuto sottraendo all'immagine originaria la controparte *smoothed*.
Random Crop	Semplice ritaglio dell'immagine. Innumerevoli combinazioni, nel rispetto dei requisiti dimensionali.
Saturation Changes	Nello spazio HSV le immagini sono caratterizzate dalle proprietà *saturazione* e *valore*, le quali possono essere alterate a piacere. L'immagine è poi riconvertita in formato RGB.
Resizing	Ridimensionamento dell'immagine. Applicate per emulare la distanza.
Mirroring	L'immagine è rispecchiata.
Additive Noise	Immagini rumorose: a) generano copie distanti dall'originale, b) insegnano al modello ad effettuare previsioni in presenza di rumore. Immagini rumorose possono essere ottenute sommando alla matrice immagine una matrice random di eguali dimensioni.

Le tecniche di data augmentation sono applicate con attenzione, a seconda del caso in esame. Ad esempio, per un problema di segmentazione di immagini urbane per navigazione autonoma di veicoli, variazioni di contrasto, saturazione, luminosità delle immagini irrobusticono il modello rendendolo resistente a differenti condizioni di illuminazione. Capovolgere orizzontalmente le immagini può invece portare ad una totale alterazione del significato di elementi chiave come ad esempio i cartelli stradali.

[5] Nel caso della presente applicazione, si è voluto limitare le trasformazioni a *cropping*, *mirroring* e *resizing* con il duplice obiettivo di ottenere un numero di immagini sufficientemente elevato onde evitare sovradattamento e fornire uno strato di input di dimensioni contenute per limitare il consumo di memoria.

3.4 Suddivione del dataset

Come intuibile, è inopportuno utilizzare le stesse immagini per addestramento e valutazione: quando il dataset è ristretto e non sufficientemente disomogeneo, la rete memorizza le immagini e l'addestramento non ha significato. In genere, i dataset sono suddivisi in un *training* set, un *validation* set e un *test* set, tali per cui:

$$\mathcal{X} = \mathcal{X}_{allenamento} \cup \mathcal{X}_{validazione} \cup X_{test}$$

e

$$\mathcal{X}_a \cap \mathcal{X}_v = \mathcal{X}_v \cap \mathcal{X}_t = \mathcal{X}_t \cap \mathcal{X}_a = \emptyset$$

dove $\mathcal{X}_a$, $\mathcal{X}_v$, $\mathcal{X}_t$ sono rispettivamente i sottoinsiemi di allenamento, validazione, test. [5]

Se le immagini sono state raccolte in istanze diverse, ad esempio in condizioni di illuminazione differenti, come giorno e notte, è opportuno preparare un numero di set di validazione corrispondente al numero di cosiddette *distribuzioni*. Nel caso in esame ciò non è stato necessario; tuttavia, a supporto di una possibile navigazione in condizioni di scarsa visibilità, diverrebbe indispensabile introdurre le immagini corrispondenti e preparare due distinti set di validazione. [5]

$\mathcal{X}_a$ è stato utilizzato esclusivamente per l'addestramento, durante il quale la prestazione della rete è stata valutata, ad intervalli regolari, su $\mathcal{X}_v$. Infine il test è condotto, una sola volta, sul sottoinsieme $\mathcal{X}_t$; la prestazione del modello in $\mathcal{X}_t$ è indice dell'accuratezza che ci si può aspettare nel mondo reale.

Sulla base della generalizzazione ottenibile sul modello addestrato, segue la scelta del numero effettivo di immagini da assegnare a ciascuna categoria. Qualora il dataset fosse esteso e variegato, si propende per un rapporto di 80-10-10%: certi del buon livel-

lo di generalizzazione a cui provvedono quantità e varietà del set di addestramento, viene meno anche la necessità di un vasto set di validazione. Altre scelte comuni sono 70-15-15% oppure 60-20-20%. Quest'ultima è adatta a dataset ridotti: non potendo supplire alla generalizzazione per via del rischio di sovradattamento, si ampliano i set di validazione e test e si punta ad una robusta valutazione della bontà del modello. [5]

Prima di effettuare test, se il risultato della validazione ottenuto al termine dell'allenamento non è soddisfacente è necessario condurre un'attenta revisione delle scelte (classi, *labeling*, rete, parametri di addestramento). È di fatto una procedura di *trial and error*: effettuati nuovamente addestramento e validazione, se non soddisfatti, li si ripetono fino al raggiungimento di risultati ritenuti accettabili.

3.5 Dataset di addestramento

Le immagini sulle quali sono condotti addestramento, test e validazione, appartengono a due distinti dataset: "terrestre" e "marziano". Con il primo si vuole testare la tecnica di segmentazione semantica nel suo insieme e possibilmente ottenere buoni risultati sul sottoinsieme di test; il secondo corrisponde a quella che più si avvicina all'applicazione reale di reti per segmentazione di terreno marziano.

3.5.1 Dataset terrestre

Con l'intento di condurre una prima, grezza valutazione dell'efficacia della segmentazione semantica tramite reti neurali convoluzionali e al contempo acquisire un certo grado di confidenza con l'intero processo, è stato preparato un dataset di immagini catturate sulla Terra. Esse provengono dal **Katwijk Beach Planetary Rover Dataset** ,[17] una raccolta di sequenze di attraversamento di terreno simil-marziano appositamente realizzato. Il dataset è stato creato per la validazione di algoritmi di *visual odometry*, messi in difficoltà dalla presenza di artefatti prodotti dalla luce solare, ma ben si presta anche alla presente applicazione.

Figura 3.4: ESA testbed rover. [17]

Figura 3.5: Immagini LocCam.

L'insieme qui utilizzato si compone di 50 immagini selezionate casualmente tra le 2250 della *LocCam* di sinistra delle prime due traccie della seconda sequenza.

Le classi scelte per questo dataset sono 3: “sabbia”, “roccia”, “background”. La scelta di sabbia e roccia non richede giustificazioni, è ovvia. La classe background comprende invece tutti gli elementi che non appartengono alle altre due: persone, mare, boscaglia, cielo. Definire una classe per ognuna di queste categorie non porterebbe alcun beneficio ai fini dell’obiettivo proposto e sarebbe anzi d’intralcio.

Cinquanta immagini sono un insieme molto piccolo, anche per applicazioni di segmentazione, e sono quindi applicate alcune tecniche di *data augmentation*. Le immagini hanno risoluzione 1024×768, non particolarmente adatta all’addestramento, e un diretto ridimensionamento a 512×512 porterebbe alla perdita di importanti informazioni spaziali. Sono state allora capovolte orizzontalmente, quindi ritagliate come evidenziato in figura. Sono così ottenute 400 immagini: un numero non molto elevato ma ritenuto sufficiente ad evitare sovradattamento. Per la suddivisione è stato adottato il rapporto generico di 70%-15%-15%: 280 per addestramento, 120, ripartite in due sottoinsiemi, per validazione e test.

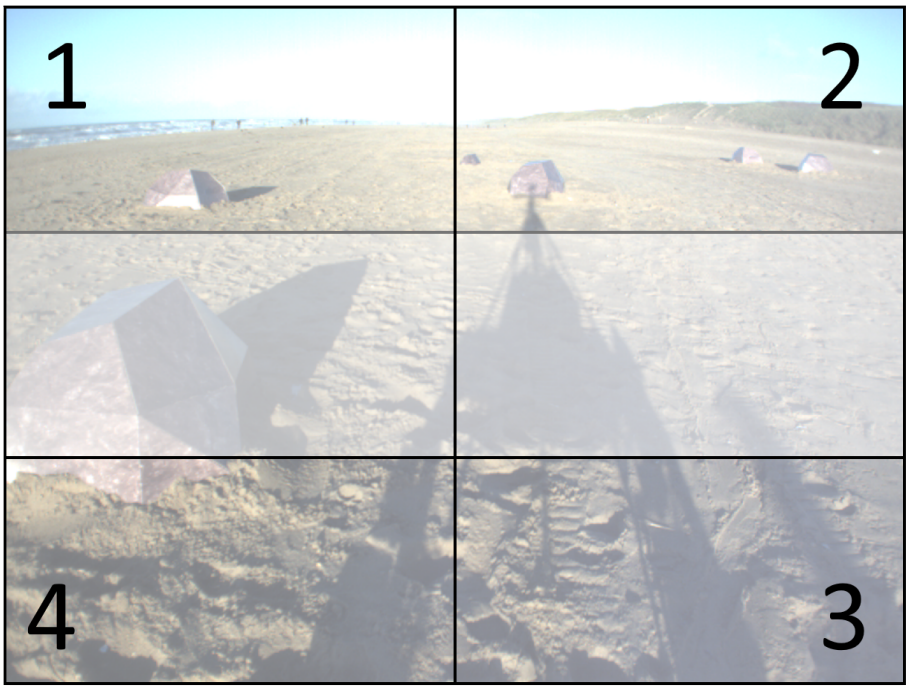


Figura 3.6: Sequenza di cropping delle immagini terrestri.

3.5.2 Dataset marziano

Milioni di immagini della superficie marziana sono accessibili al pubblico dai siti web NASA; i rover di riferimento sono *Spirit*, *Opportunity*, lanciate nel 2003, e *Curiosity*, lanciata nel 2011. A costituire il corpo del dataset qui utilizzato sono, per semplicità, immagini di più recente acquisizione, provenienti dal solo Curiosity. Nei 21 chilometri percorsi, il rover ha attraversato un discreto numero di siti di varia conformazione geologica registrando migliaia di immagini contenenti un'altrettanto ricca varietà di terreni. Quantità e varietà delle immagini permettono la creazione di un dataset ampio e variegato, secondo quanto necessario per un'efficace segmentazione.

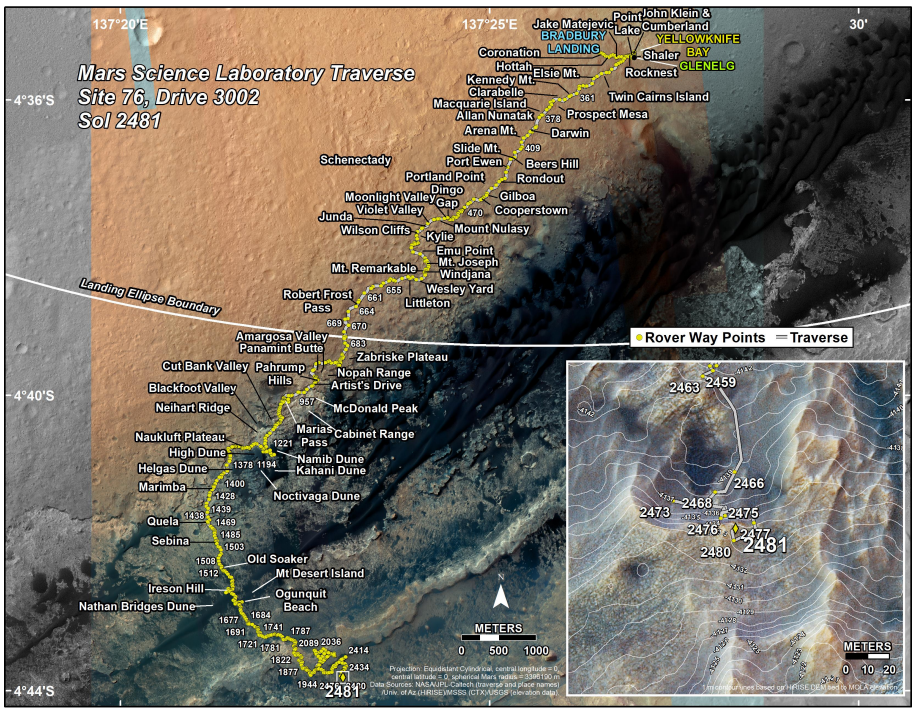


Figura 3.7: Percorso di Curiosity dal SOL 1 al SOL 2481. [15]

Si vuole stabilire da quanti e quali sensori scegliere le immagini, poiché non tutte sono adatte al *task* di segmentazione: alcune sono in bianco e nero, altre a colori, alcune includono elementi artificiali, altre ancora contengono abbondanti porzioni di cielo.

Le macchine fotografiche del rover sono 17 e suddivise in *Engineering*, *Science* e *Descent Imaging*. Del primo set fanno parte: *Hazcam* e *Navcam*. [11]

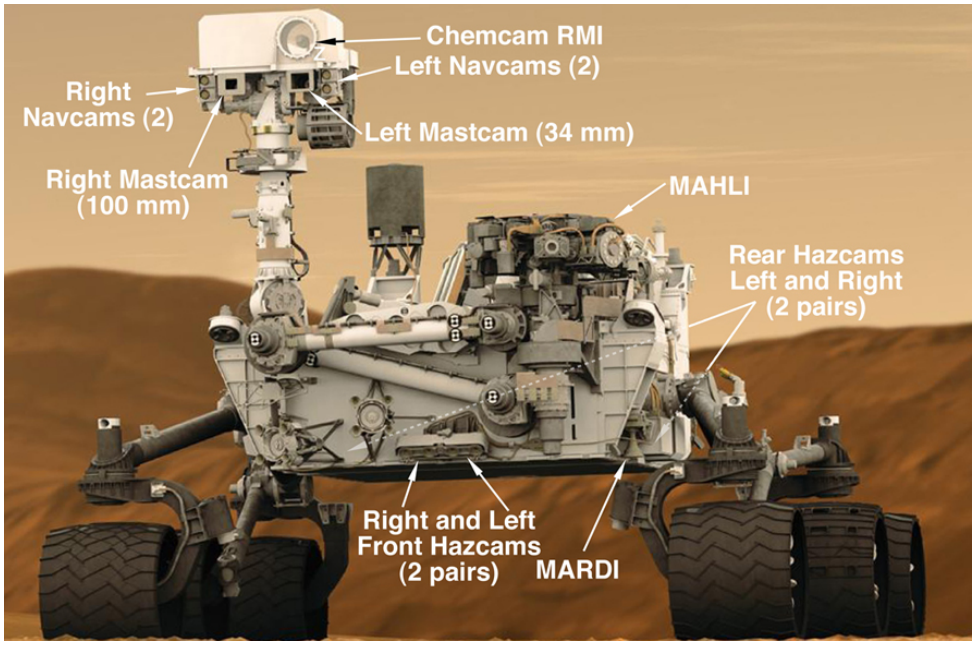


Figura 3.8: Fotocamere di Curiosity. [16]

Hazcam. Hazard Avoidance Cameras: due paia di stereo-camere fisse, putate verso il terreno e montate in fronte e sul retro del rover. Catturano immagini di 1024×1024 pixel, dal visibile, in bianco e nero e con un campo di vista di 120 gradi. Individuano la presenza di ostacoli nelle immediate vicinanze e contribuiscono alla mappatura del terreno. [11]

(a) Vista dell'Hazcam anteriore. **(b)** Vista dell'Hazcam posteriore.

Figura 3.9: Hazcam; immagini di sinistra delle due stereo-coppie anteriore e posteriore.

Navcam. Navigation Cameras. Montata sull'albero (*mast*), la stereo-camera cattura immagini di 1024×1024 pixel con un campo di vista di 45 gradi. Fornisce informazioni complementari ai dati delle Hazcam a supporto del *path-planning*. [11]

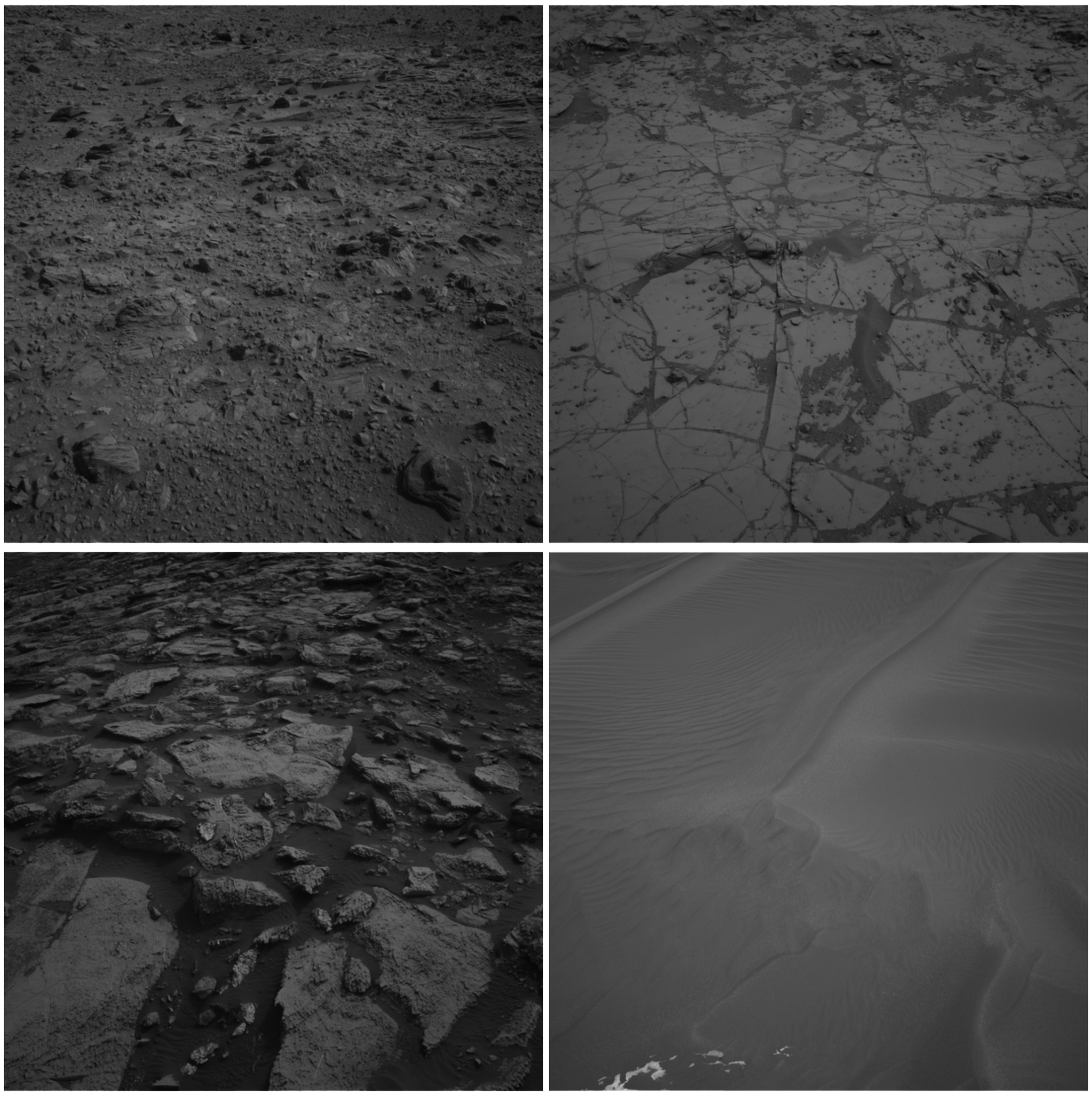


Figura 3.10: Immagini Navcam.

Del secondo set fanno parte Mastcam, Chemcam, MAHLI, del terzo MARDI.

Mastcam. Due macchine fotografiche di specifiche differenti, è stata concepita per svolgere il compito di una macchina fotografica *consumer-grade*. È montata sull'albero,

a circa 2 metri da terra. Acquisisce video ed immagini a colori di 1600×1200 pixel alla risoluzione massima di 150 micron alla distanza di 2 metri. Le immagini sono di supporto allo studio del suolo e delle rocce e possono essere cucite assieme per realizzare panoramiche della superficie marziana. [11]

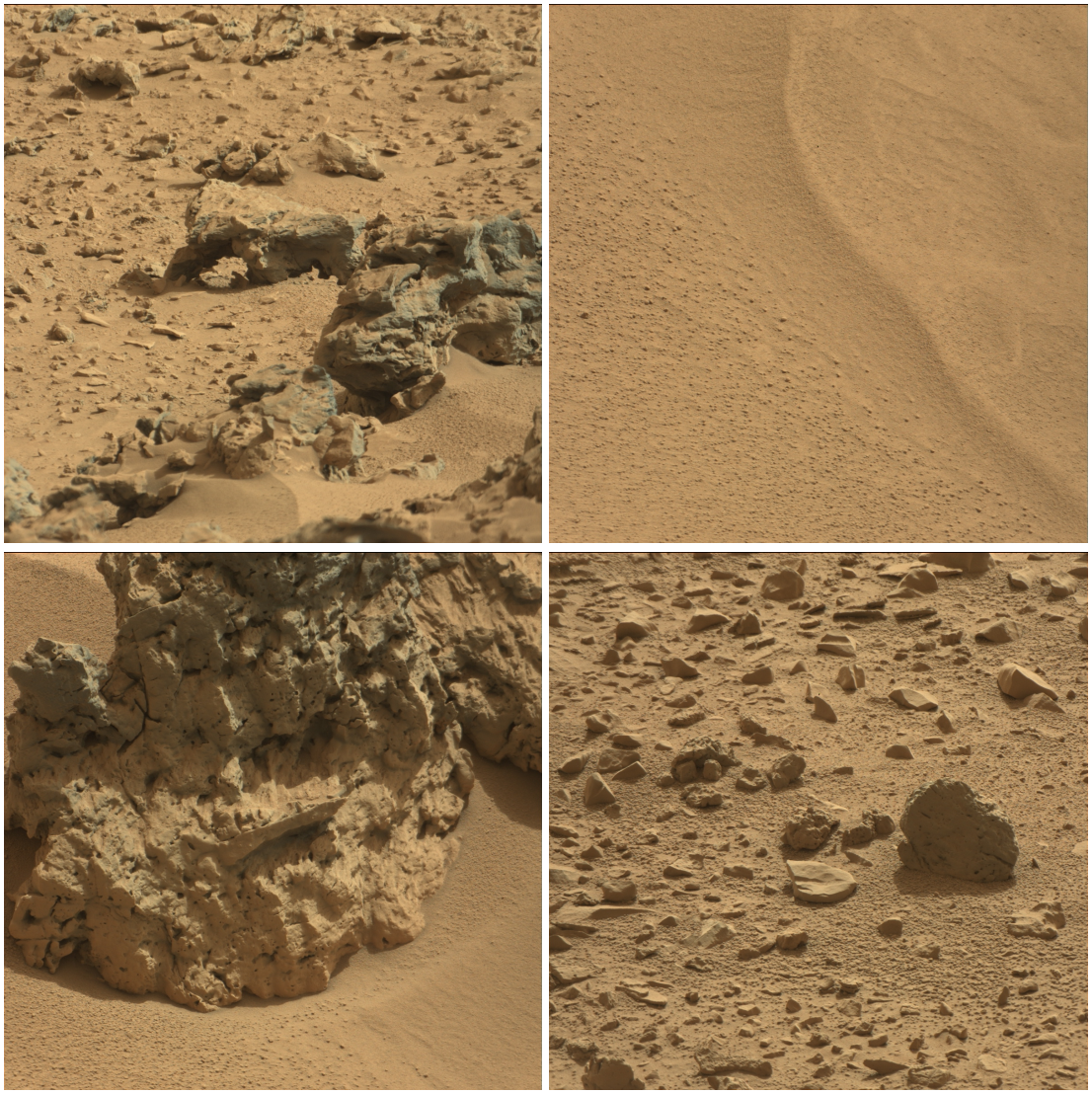


Figura 3.11: Immagini Mastcam.

Le altre fotocamere non forniscono immagini adatte all'applicazione in esame e sono quindi scartate a priori. Mastcam fornisce buone immagini a colori ma caratterizzate

da FOV assai ristretti, a differenza di Navcam, la quale fa dell'ampio campo di vista il suo punto di forza. Il formato in scala di grigio corrisponde ad una perdita di informazioni relative al colore ma non pregiudica la segmentazione: rocce, sabbia, sassi, rimangono ancora perfettamente distinguibili l'una dall'altra. Stabiliti missione e strumento si procede con la selezione delle immagini. Sul portale **PDS Image Atlas**, la NASA mette a disposizione del pubblico milioni di immagini, ciascuna delle quali è accoppiata al *label* corrispondente.

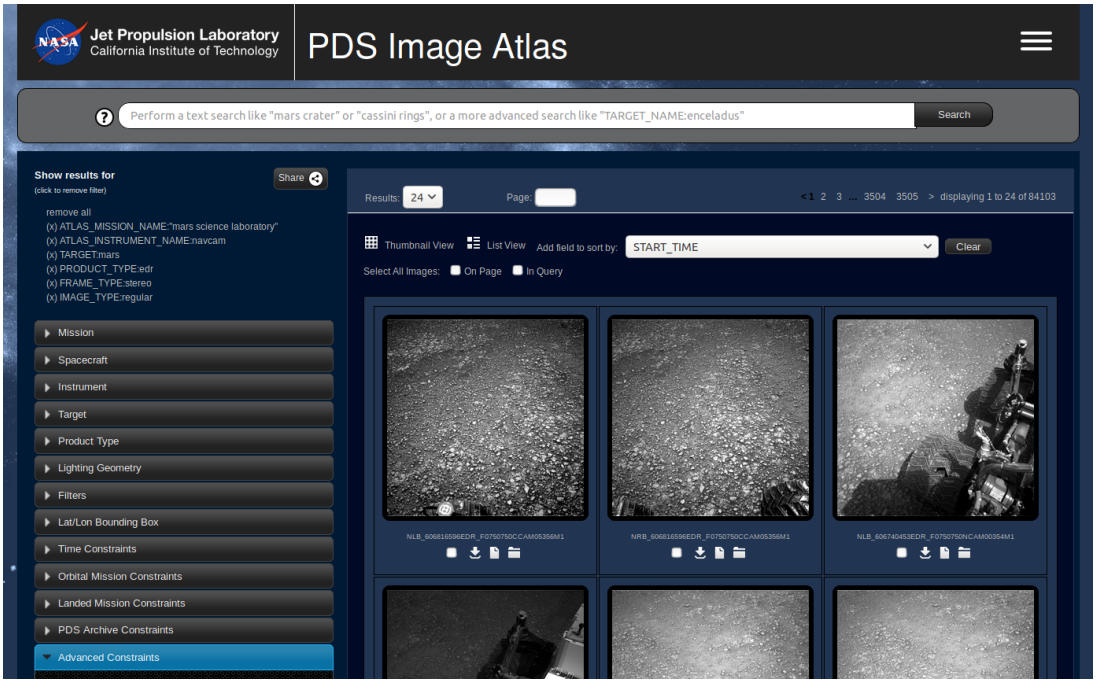


Figura 3.12: Schermata del portale PDS Image Atlas.

L'etichetta è un file in formato .LBL contenente informazioni relative alle proprietà dell'immagine .IMG e allo stato del rover e dei suoi strumenti all'istante in cui la foto è stata catturata. Anche in questo caso non tutte le immagini presenti sono adatte all'applicazione in esame: alcune sono troppo piccole, altre sono ritoccate, altre ancora coprono esclusivamente porzioni di cielo e il dataset risulta comunque fin troppo vasto.

Al fine di scremare la notevole quantità di immagini disponibili, è fatto uso della funzionalità di ricerca avanzata *advanced constraints*: con notevole risparmio di tempo,

viene meno la necessità di scaricare grandi quantità di dati per poi doverne scartare (manualmente) in abbondanza. Stabiliti e applicati i vincoli, se la quantità di immagini che li soddisfa è sufficente si procede al download, in caso contrario si rivedono i parametri e si compie una nuova ricerca. La procedura di selezione dei parametri e dei loro valori è stata condotta per tentativi sulla base dei risultati prodotti. Osservando le immagini agli estremi dei valori è stato possibile fissare con discreta precisione parametri numerici come `RSM_ELEVATION_MEASURED` e `RSM_AZIMUTH_MEASURED`. Ad esempio, se l'immagine presentava abbondanti porzioni di cielo o di rover, il valore associato al parametro determinante (elevazione nel primo caso, azimuth nel secondo) veniva alzato o abbassato a seconda della necessità, via via escludendo dalle immagini l'oggetto indesiderato. In tal modo si è giunti alla coppia di estremi indicata in tabella.

Qualsiasi tipo di informazione proveniente da strumenti di sonde, satelliti, etc., incluse le immagini, è classificato in due macrocategorie: EDR (**E**xperiment **D**ata **R**ecord) ed RDR (**R**educed **D**ata **R**ecord). [11] Nonostante le immagini *raw* navcam abbiano esclusivamente dimensioni pari a 1024×1024 pixel, l'atlante proponeva in aggiunta una serie di thumbnail, troppo piccoli per poter essere segmentati appropriatamente, quindi esclusi fissando `NUMBER_OF_LINES` a 1024. La prima consiste in dati non-processati, la seconda in dati ri-elaborati; le immagini classificate come EDR sono quelle più simili l'una all'altra quanto a dimensioni e contenuti e per questo motivo sono state selezionate a dispetto delle RDR, disomogenee ed inferiori in numero. Per escludere i mosaici sono state selezionate immagini del tipo `regular`, per escludere immagini simili (trattandosi di una stereo-camera) è stata presa in considerazione solamente la `navcam_left_b`. È stato infine necessario rimuovere manualmente qualche decina di immagini contenenti le impronte e/o porzioni di rover o segmenti di cielo, in quanto elementi di scarso interesse ai fini della classificazione.

Tabella 3.1: Parametri e valori per la selezione delle immagini.

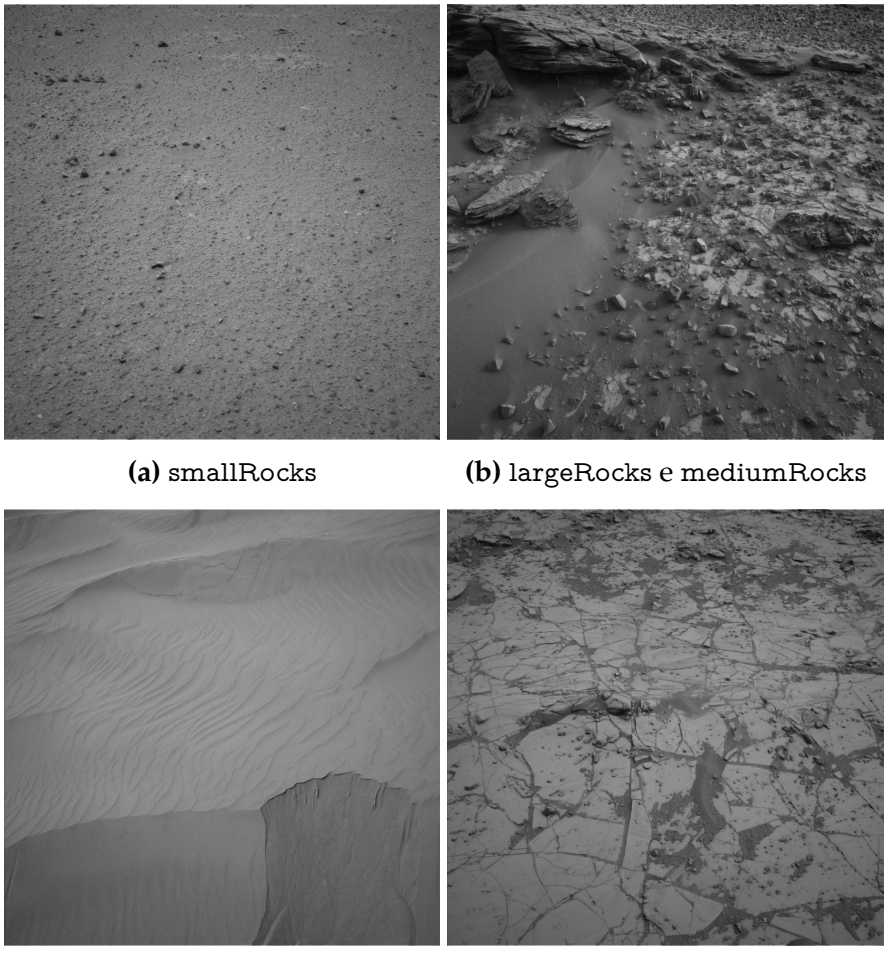
Parametro	Valore
ATLAS_MISSION_NAME	mars science laboratory
ATLAS_INSTRUMENT_NAME	navcam
TARGET	mars
PRODUCT_TYPE	edr
FRAME_TYPE	stereo
IMAGE_TYPE	regular
INSTRUMENT_ID	nav_left_b
NUMBER_OF_LINES	1024
RSM_ELEVATION_MEASURED	0.8 to 1.1
RSM_AZIMUTH_MEASURED	3 to 5

Per poter essere visualizzate, le immagini sono convertite dal formato .IMG al formato .png tramite l'utility **img2png**, fornita dalla NASA. Essa converte *raw data* dei file IMG prelevando le informazioni necessarie dal file di testo .lbl corrispondente. Senza definire il parametro **s** di moltiplicazione dei valori RGB, le immagini risultano troppo scure. Osservando invece come il software di visualizzazione **NASAView** interpreta il formato IMG, è stato assegnato ad **s** il valore 13. Per facilitare la segmentazione manuale è possibile scegliere valori più alti, facendo però attenzione a fornire alle reti immagini con luminosità simile anche in inferenza.

La scelta delle classi, condotta sulla base delle linee-guida del paragrafo 3.1, è ricaduta sulle cinque in tabella 3.2: "sabbia", "roccia liscia", rocce "piccole", "medie", "grandi". La scelta non è stata immediata. mediumRocks è stata introdotta solo in un secondo momento, una volta osservata l'ingente presenza di rocce di medie dimensioni caratterizzate da discreta pericolosità; la classe outcrop, introdotta in un primo istante, è stata poi rimossa in quanto non associabile ad un unico livello di pericolosità.

Tabella 3.2: Classi

Classe	Descrizione
sand	Duna di sabbia, pericolosità bassa
bedrock	Letto di roccia, pericolosità molto bassa
smallRocks	Rocce di piccole dimensioni, pericolosità bassa
mediumRocks	Rocce di medie dimensioni; non necessariamente terreno intransitabile ma segnato dalla presenza da rocce appuntite che possono danneggiare le ruote. Pericolosità media
largeRocks	Rocce grandi; terreno intransitabile, pericolosità alta



(a) smallRocks **(b)** largeRocks e mediumRocks

(c) sand **(d)** bedrock

Figura 3.13: Esempi delle classi

Definite le classi, è effettuato il *labeling*. In un primo momento sono state etichettate 200 immagini navcam 1024×1024 facendo uso dell'applicazione *Image Labeler*. Tramite lo strumento *polygon* è tracciato il contorno di ciascuna porzione di immagine corrispondente alla classe selezionata, a cui l'applicazione poi associa l'intero contenuto del poligono. Talvolta la segmentazione non è semplice ed immediata per via di alcune immagini caratterizzate da insiemi di rocce di varia natura miste a sabbia, le quali rendono molto difficili la distizione tra classi e il tracciamento del poligono. Consci degli effetti di propagazione, durante l'addestramento, dell'errore commesso in fase di classificazione manuale, si è cercato, per quanto possibile, di tracciare una *ground truth* accurata.

Anche qui, essendo il numero di immagini ridotto, si è percorsa la strada del *data augmentation*. A causa di alcune difficoltà incontrate con le funzioni apposite fornite da MATLAB ed è stato compilato un breve script che opera le seguenti trasformazioni.

1) Ciascuna delle 200 immagini è capovolta orizzontalmente come in figura:

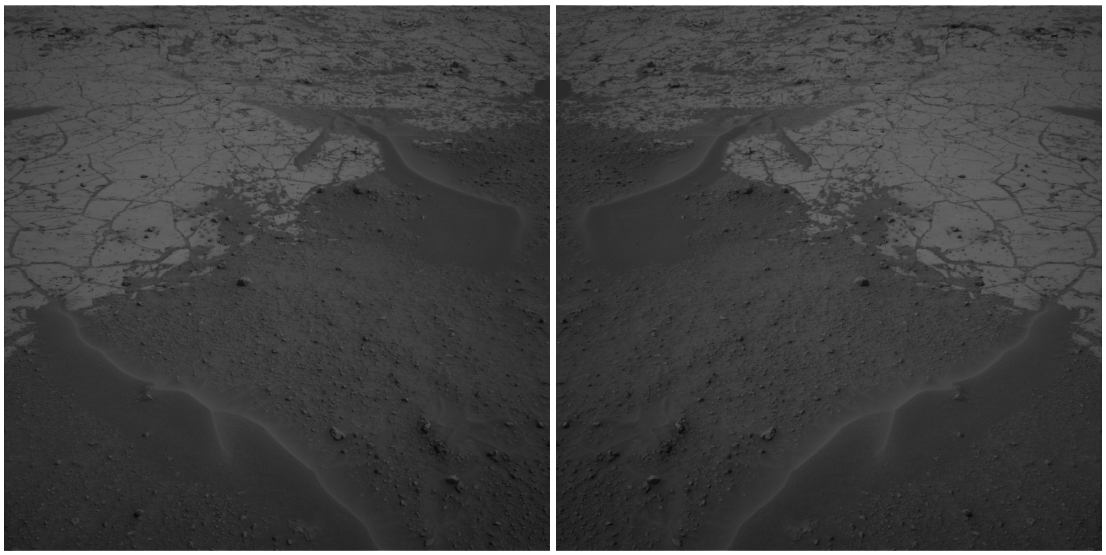


Figura 3.14: Esempio di *mirroring*.

2) Le 400 immagini sono ritagliate come evidenziato. Si ottengono, per ciascuna, 5 immagini di dimensione 512×512, per un totale di 2000.

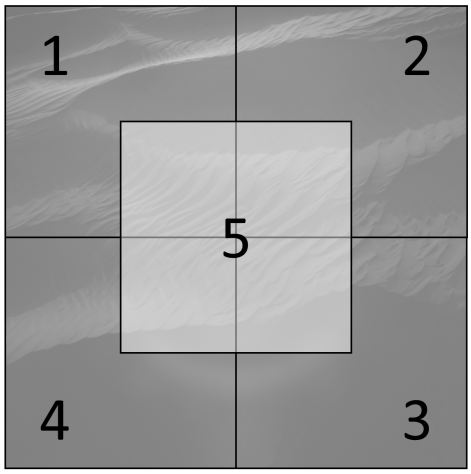


Figura 3.15: Sequenza di cropping di primo tentativo.

3) Le 400 full-size del punto *1* sono ridimensionate a 512×512. Il numero finale di immagini è 2400.

4) Tramite la sintassi seguente:

```
I = cat(3, I, I, I)
```

le immagini sono convertite in formato RGB per poter essere date in inupt alle funzioni di creazione degli strati delle reti, le quali non accettano immagini in scala di grigio ma solo a profondità 3. Ciò comporta un incremento del tempo computazionale senza acquisizione di ulteriori informazioni, ma non è stato possibile fare altrimenti.

Ad esclusione della conversione in formato RGB, procedura di cui sopra è stata applicata anche alle immagini dei *label*. È opportuno infine rinominare i file secondo la denominazione 0001, 0002, 0003, ... in modo che le funzioni `imageDatastore` e `pixelLabelDatastore` interpretino correttamente la sequenza, ordinando le immagini numericamente e non per stringa.

In un secondo momento, visti i primi risultati dell'addestramento sul dataset così configurato, è stato ritenuto opportuno rivedere la segmentazione manuale ed apportare appropriate modifiche.

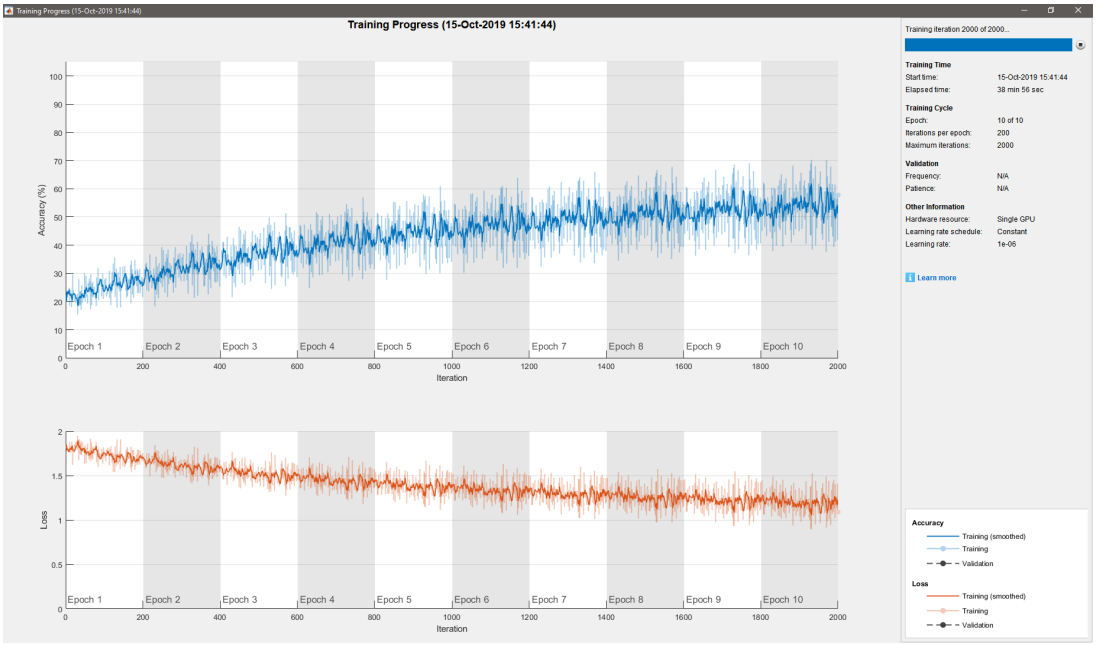


Figura 3.16: Primo tentativo di addestramento.

Senza soffermarsi sui dettagli, in figura è evidenziato come la rete addestrata non raggiunga un livello soddisfacente di accuratezza e si assesti su una bassa percentuale. Sono stati quindi rifiniti alcuni contorni in immagini critiche ed escluse porzioni di terreno di incerta classificazione. In alcuni casi la segmentazione è apparsa completamente errata, a testimonianza dell'alto grado di variabilità delle caratteristiche delle classi e del successivo *labeling*. Infine, per non indurre la rete in confusione con scale differenti, il ritaglio delle immagini originale è stato sostituito dal seguente:

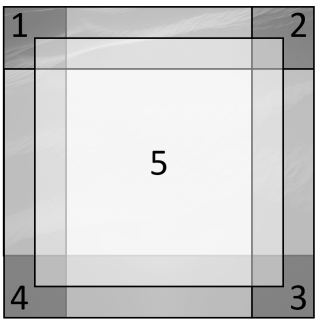


Figura 3.17: Sequenza di cropping di secondo tentativo.

Una sesta immagine (per un totale di 2400) è stata ottenuta da un ulteriore ridimensionamento dell'immagine già ridimensionata. Questa volta però la scala non risulta alterata come avvenuto in precedenza e la rete beneficia dell'irrobustimento garantito da una tecnica simil-*sliding-window*.

La suddivisione è stata condotta con rapporti di 80%-10%-10%: 1920 immagini per addestramento e 240 ripartite in due sottoinsiemi per validazione e test.

CAPITOLO 4

Addestramento

L'addestramento è condotto su due distinte reti, la prima come architettura di esempio, la seconda scelta tra quelle disponibili in MATLAB. Le reti neurali presenti nel *Computer Vision* toolbox sono:

- FCN (**F**ully **C**onvolutional **N**etwork)
- SegNet
- U-net
- DeepLabv3+

FCN (2015), SegNet (2015), U-Net (2015) sono tre architetture convoluzionali per segmentazione semantica di prima generazione, seguite, in ordine cronologico, da DeepLab. DeepLabv3+ è stata sviluppata nel 2018 ed è la rete più performante tra quelle disponbili e per questo motivo è scelta come seconda rete di addestramento. Di seguito sono presentati due aspetti inerenti il consumo delle risorse computazionali e alla tipologia di modelli solitamente utilizzata.

Apparato hardware e software Gli algoritmi di training di reti neurali sono generalmente assai dispensiosi in termini di risorse computazionali. I due fattori determinanti sono: a) la dimensione della rete, ossia il numero di parametri; b) la dimensione del *dataset*. Al fine di velocizzare l'addestramento e l'*inferencing* si adottano soluzioni a livello software, con l'implemetazione di tecniche di compressione dei modelli, e hardware, con l'uso di (multiple) GPU.

Come suggerisce il nome, le ***G**raphic **P**rocessing **U**nits*, comunemente dette *schede grafiche*, sono state ideate appositamente per il *rendering* grafico. Mentre la CPU esegue *branching operations*, ossia istruzioni ripetitive in rapida successione del tipo `if-then`, la GPU gestisce liste di coordinate tridimensionali sulle quali conduce, in parallelo per velocizzarne l'esecuzione, operazioni matriciali. Numero elevato di *cores*, spinta parallelizzazione, memoria ad ampia larghezza di banda permettono alla GPU di svolgere più complesse computazioni. A tal proposito, si ricordi come l'addestramento di reti neurali veda coinvolte moltiplicazioni di vettori, matrici, tensori, le stesse operazioni alle quali le GPU sono orientate. Le potenzialità prestazionali delle GPU in ambito *deep learning* hanno trovato espressione grazie a NVIDIA, la quale ha eliminato la necessità di programmazione a basso livello mettendo a disposizione dell'utente finale la libreria CUDA. La libreria è integrata nei deep learning *frameworks* più diffusi come Caffe, TensorFlow, Theano, Torch, e presente in MATLAB. [6]

Il calcolatore sul quale è condotto l'allenamento è dotato di: scheda grafica NVIDIA RTX 2060 Super (con *tensor cores*), processore Intel Core i7 4770k @ 4.2GHz, SSD. Per semplicità, il software utilizzato è MATLAB R2019b con i toolbox *Deep Learning*, *Computer Vision*, *Image Processing*.

Come espresso in precedenza, l'addestramento delle reti è condotto in ambiente MATLAB. Per l'occasione sono creati tre script.

`dataSetup.m`	Script di definizione del dataset e delle classi e di partizionamento del dataset
`networkSetup.m`	Script di generazione dei *layers* delle reti, definizione delle dimensioni dello strato di input (ossia delle immagini) e del numero di classi.
`networkTraining.m`	Script di definizione delle opzioni di addestramento e di addestramento.

L'addestramento di reti neurali avviene tramite algoritmi di retropropagazione, ciascuno dei quali vede coinvolti differenti parametri. In MATLAB, la funzione `trainingOptions` implementa gli algoritmi e le differenti opzioni di addestramento disponibili,

lasciando all'utente le scelte relative ai valori dei parametri. La prima scelta è quella dell'algoritmo di retropropagazione: quello più diffuso, quindi qui adottato, è l'SGDM (**S**tochastic **G**radient **D**escent with **M**omentum). L'algoritmo base, il solo SGD, aggiorna i parametri della rete con l'obiettivo di minimizzare la funzione di perdita, facendo piccoli passi nella direzione del gradiente negativo, secondo la formula:

$$p_{i+1} = p_i - \alpha \nabla L(p_i)$$

dove p è il vettore parametro, i il numero dell'iterazione, $\alpha > 0$ il *learning rate*, $L(p)$ la funzione di perdita. La discesa del gradiente è definita stocastica in quanto i parametri computati sulla *mini-batch* corrispondono ad una stima rumorosa del valore che assumerebbero se fossero calcolati sull'intero set di addestramento.

L'introduzione del termine *momento* contrasta l'oscillare dell'algoritmo nella discesa verso l'ottimo. La formula è allora aggiorata come segue:

$$p_{i+1} = p_i - \alpha \nabla L(p_i) + \gamma(p - p_{i-1})$$

dove γ è un valore prefissato. [12]

Tra le opzioni relative al *mini-batch*, alla validazione, effettuata in contemporanea all'addestramento, e all'algoritmo di retropropagazione, le seguenti sono di particolare interesse.

Numero di epoche (massimo) Un'iterazione è un passo verso la discesa del gradiente e la minimizzazione della funzione di perdita. Un'epoca è un unico passaggio attraverso l'intero sottoinsieme di addestramento. Con questo parametro è definito il numero massimo di epoche, raggiunto il quale l'addestramento si interrompe.

Dimensione della *mini-batch* Sottoinsieme dell'insieme di addestramento utilizzato per il calcolo del gradiente della funzione di perdita e l'aggiornamento dei pesi. Più avanti, alcuni dettagli sulla scelta dei valori.

Shuffling	Semplice operazione di rimescolamento delle immagini ad ogni epoca.
Sottoinsieme di validazione	Subset di immagini di validazione, come già specificato.
Frequenza di validazione	Frequenza con la quale l'algoritmo opera l'inferenza sul set di validazione, la scelta è arbitraria.
***Learn rate* iniziale**	Come descritto poco sopra, il valore iniziale del parametro associato alla rapdità dell'apprendimento. Se troppo basso, il *training* richiede tempi lunghi, se troppo alto, il *training* può giungere a scarsi risultati o divergere.
Learn rate schedule	Opzione per ridurre il *learning rate* durante l'apprendimento. Il valore `'piecewise'`, qui adottato, fa sì che il software aggiorni il *learning rate*, ogni n epoche, moltiplicandolo per un fattore $k < 1$.
Learn rate period	Numero n di epoche, default: 10.
Learn rate drop factor	Fattore k.
Momento	Fattore γ.

Infine, le opzioni relative all'ambiente hardware ove condurre l'addestramento sono specificate con il parametro `ExecutionEnvironment`. Qualora fosse presente una GPU CUDA adatta, MATLAB ne fa uso, a discapito delle più lente CPU.

Gran parte di questi parametri possono essere stabiliti una sola volta e rimanere invariati. Tre, in particolare, sono invece il risultato di molteplici tentativi di addestramento; essi sono: numero di epoche, dimensione della *mini-batch*, *learn rate* iniziale.

Il numero di epoche in genere è compreso tra 10 e 100 e tende ad avvicinarsi al limite inferiore nel caso in cui la rete arrivi a convergenza molto rapidamente. Stabilito

il valore, al termine dell'addestramento si osserva attentamente la curva descritta dall'accuratezza: se nei tratti finali appare ancora in salita, allora è opportuno aumentare la quota di epoche. Al contrario, se l'accurattezza non registra variazioni sostanziali nel corso di più epoche, è il caso di ridurne la quantità. Il numero massimo qui considerato è *20*.

L'algoritmo SGDM opera su un piccolo sottoinsieme (*mini-batch*) di immagini dell'insieme di addestramento, la cui dimensione è uno dei due fattori di grande impatto sulla memoria volatile occupata, assieme al numero di parametri della rete. Più elevato è il numero di immagini elaborato per iterazione, maggiore è il numero di valori coinvolti e maggiore sarà la memoria occupata. Ciò comporta ovvi limiti associati al consumo di risorse computazionali e, anche qui, la necessità di ripetere più volte l'addestramento. In questo caso è però sufficiente osservare i primissimi istanti dell'addestramento: se la memoria grafica occupata raggiunge il limite superiore consentito, MATLAB avvisa l'utente e l'accuratezza raggiungerà valore nullo. Gli strumenti di controllo delle risorse dei sistemi operativi facilitano l'osservazione dell'andamento della memoria disponibile permettendo una rapida modifica del parametro `miniBatchSize`, qualora fosse ritenuta opportuna. Il numero di immagini inizialmente considerato è 8 per DeepLabv3+ e 16 per la rete di esempio, poi riportato a 8-10.

Learning rate è un parametro chiave dell'addestramento in quanto responsabile della rapidità con la quale si arriva a convergenza. Se il valore è troppo basso o troppo elevato la rete non impara. In figura, due esempi. È opportuno evidenziare come non esista un valore ottimale ma solamente un intervallo indicativo, i cui estremi sono compresi fra 0.1 e 1e-6 per buona parte delle applicazioni. Nella presente applicazione, alterazioni di `initialLearningRate` di un ordine di grandezza sono risultate sufficienti a rendere possibili buone progressioni dell'accuratezza.

Lo script per l'addestramento in MATLAB si presenta nella forma seguente:

```
%% Network setup

opts = trainingOptions('sgdm', ...
    'InitialLearnRate', 1e-2, ...
```

```
    'MaxEpochs', 15, ...
    'MiniBatchSize', 10, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', pximdsVal, ...
    'ValidationFrequency', 20, ...
    'ValidationPatience', 5, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 5, ...
    'LearnRateDropFactor', 0.5, ...
    'Plots', 'training-progress');

net = trainNetwork(pximdsTrain, layers, opts);
```

I parametri scelti ed alcuni valori rilevanti sono riportati in tabella.

Tabella 4.1: Parametri di addestramento.

	Dataset terrestre		**Dataset marziano**	
	Rete proposta	DeepLabv3+	Rete proposta	DeepLabv3+
InitialLearnRate	1e-2	1e-4	1e-3	1e-5
MiniBatchSize	8	8	10	8
MaxEpochs	10	10	20	20
Shuffle	'every-epoch'	'every-epoch'	'every-epoch'	'every-epoch'
Validation				
- **Frequency**	20	20	100	100
- **Patience**	Inf.	Inf.	5	8
LearnRate				
- **Schedule**	'none'	'none'	'piecewise'	'piecewise'
- **DropPeriod**	-	-	5	5
- **DropFactor**	-	-	0.5	0.5

Avviato lo script, MATLAB apre la finestra di monitoraggio dell'addestramento. Essa contiene i grafici di progressione dell'accuratezza e della perdita ed altre utili informazioni. Nelle figure che seguono sono riportate le finestre al termine dell'addestramento delle tre reti.

Tabella 4.2: Tempi di addestramento.

Dataset terrestre		**Dataset marziano**	
Rete proposta	DeepLabv3+	Rete proposta	DeepLabv3+
3m 3s	5m 17s	38m 39s	72m 23s

I tempi di addestramento sono maggiori per DeepLabv3+, minori per la rete proposta, per effetto del numero di parametri addestrabili: $20,460,374$ per la prima, $5,482,761$ per la seconda. Nonostante i tempi di addestramento siano molto lunghi, l'inferenza richiede appena qualche millesimo di secondo per entrambe le reti.

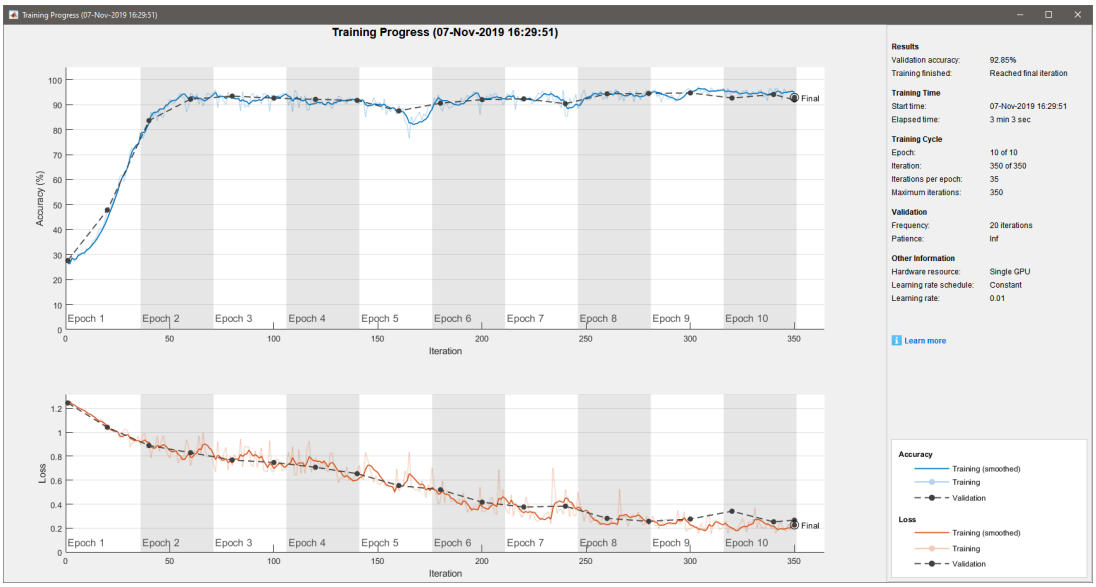


Figura 4.1: Addestramento sul dataset terrestre, architettura proposta.

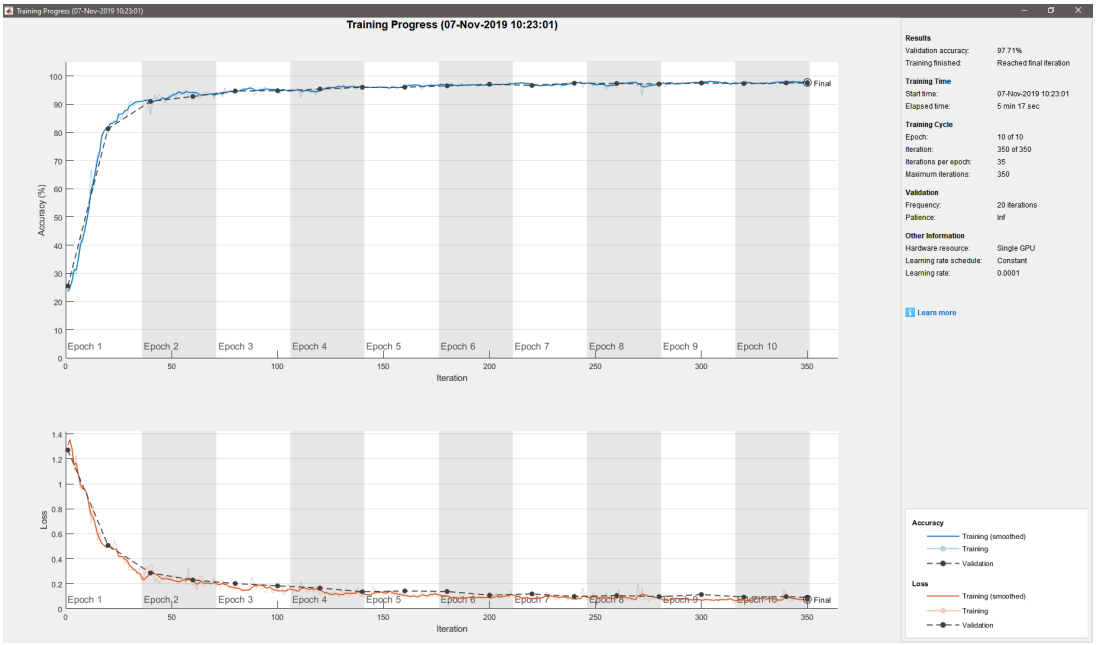


Figura 4.2: Addestramento sul dataset terrestre, DeepLabv3+ con ResNet18.

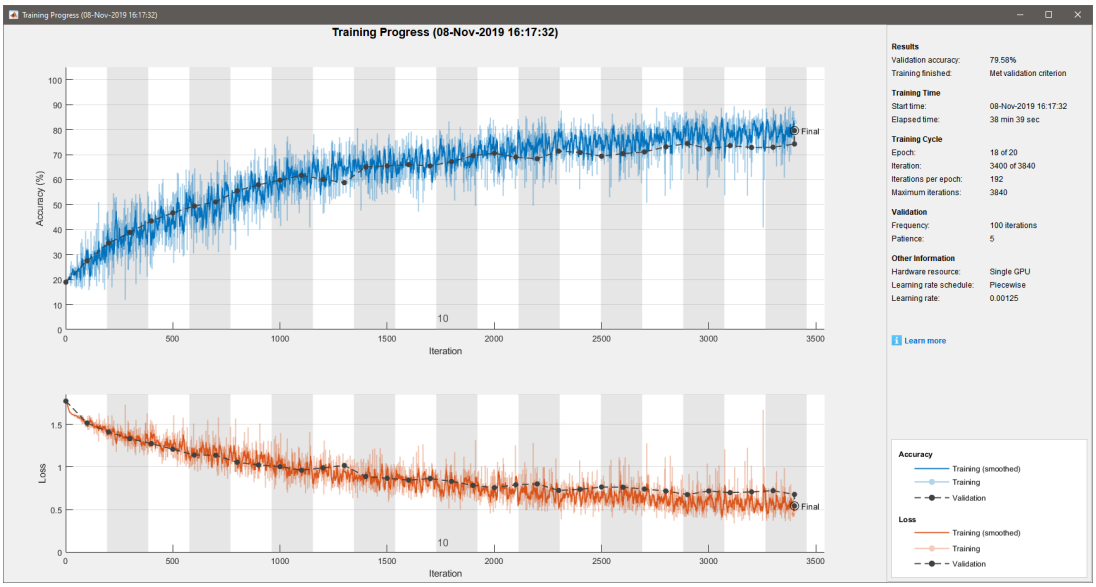


Figura 4.3: Addestramento sul dataset marziano, architettura proposta.

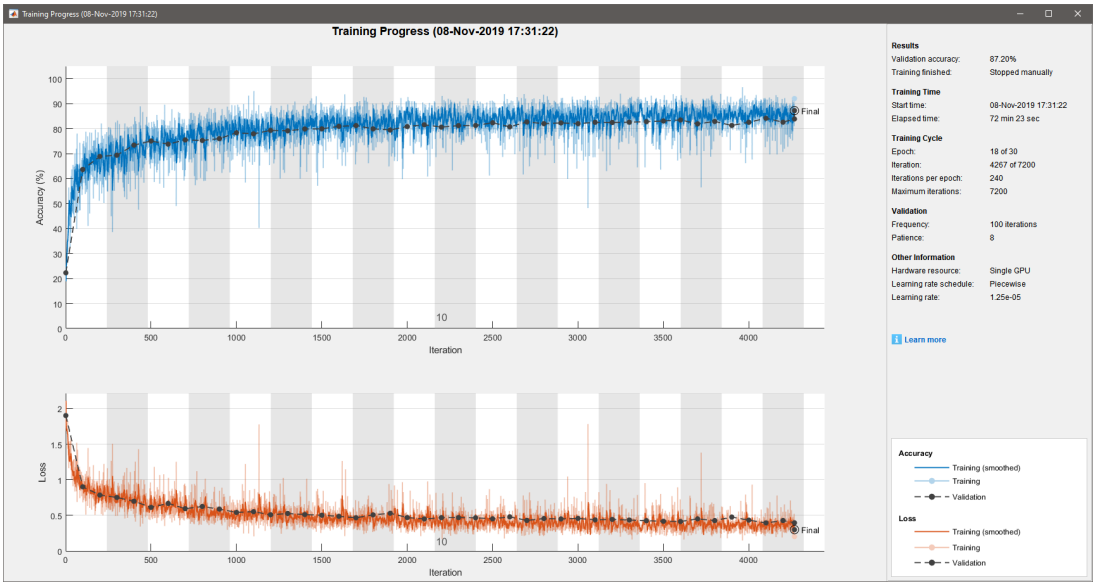


Figura 4.4: Addestramento sul dataset marziano, DeepLabv3+ con ResNet18.

CAPITOLO 5

Risultati

Il test delle reti addestrate è condotto sui rispettivi sottoinsiemi di test, contenenti una piccola percentuale di immagini scelte casualmente tra quelle dell'intero dataset classificato. Effettuata l'inferenza tramite `semanticseg`, `evaluateSemanticSegmentation` computa:

- Accuratezza globale
- Accuratezza media
- IoU medio
- IoU pesato
- BF Score (F1 Score)

L'oggetto `metrics.m` contiene inoltre la matrice di confusione e la sua versione normalizzata, qui riportata assieme ai risultati della segmentazione così calcolati e ad alcune immagini classificate per un confronto visivo diretto.

Tabella 5.1: Dataset terrestre: accuratezza, IoU, F1.

	Accuratezza		**IoU**		**F1**
	Globale	Media	Medio	Pesato	Medio
Rete proposta	0.940	0.936	0.763	0.907	0.522
DeepLabv3+	0.970	0.974	0.874	0.947	0.731

Tabella 5.2: Dataset marziano: accuratezza, IoU, F1.

	Accuratezza		**IoU**		**F1**
	Globale	Media	Medio	Pesato	Medio
Rete proposta	0.811	0.812	0.666	0.688	0.307
DeepLabv3+	0.883	0.884	0.767	0.795	0.465

Sulla base dei valori di mIoU pesati ottenuti sul dataset terrestre, è possibile decretare con certezza il successo della segmentazione per quell'insieme; in particolare, il modello DeepLabv3+ ha raggiunto la punta di 0.947. La rete proposta ha prodotto buoni valori di accuratezza ma un discreto IoU ed uno scarso F1: le immagini segmentate denotano la difficoltà del modello nel distiguere rocce da sabbia a lunghe distanze e nel generare contorni con un buon grado di approssimazione.

Di seguito, le matrici di confusione. Le colonne corrispondono alle *predizioni*, le righe alla *ground truth*.

Tabella 5.3: Dataset terrestre: matrice di confusione. Rete proposta, percentuali.

	`sand`	`rock`	`background`
`sand`	96.5	2.6	0.9
`rock`	0.3	90.6	9.1
`background`	1.1	5.2	93.7

Tabella 5.4: Dataset terrestre: matrice di confusione. DeepLabv3+, percentuali.

	`sand`	`rock`	`background`
`sand`	98.5	0.4	1.1
`rock`	0.2	96.9	2.9
`background`	0.7	2.6	96.7

Le matrici di confusione relative al dataset terrestre non sono di grande utilità in quanto i valori TP sono elevati ed FN e FP assai contenuti, ad eccezione della coppia `rock` - `background`.

Tabella 5.5: Dataset marziano: matrice di confusione. Rete proposta, percentuali.

	sand	bedrock	sRocks	lRocks	mRocks
sand	85.7	3.3	1.7	5.4	3.7
bedrock	2.4	82.7	2.6	3.4	8.8
smallRocks	6.6	2.1	84.8	0.3	6.0
largeRocks	1.8	4.5	0.5	82.5	10.6
mediumRocks	1.9	9.9	11.2	6.3	70.7

Tabella 5.6: Dataset marziano: matrice di confusione. DeepLabv3+, percentuali.

	sand	bedrock	sRocks	lRocks	mRocks
sand	90.7	2.4	1.6	4.1	1.2
bedrock	2.5	88.0	0.9	3.6	4.9
smallRocks	2.5	0.9	93.6	0.3	2.6
largeRocks	2.4	2.7	0.5	90.8	3.6
mediumRocks	2.1	7.2	5.2	6.3	79.1

Da un primo confronto diretto traspare la netta superiorità di prestazioni di DeepLabv3+ rispetto al modello proposto. A prima vista, la differenza potrebbe essere attribuita esclusivamente alla quantità di parametri, circa quattro volte maggiore rispetto alla rete proposta. Benché il numero di parametri addestrabili sia influente, un contributo notevole è dato dalla struttura del modello. In particolare nel tracciamento dei contorni, l'*unpooling* e la concatenazione di *features* operate dal decoder di DeepLabv3+ si rivelano essere superiori alla deconvoluzione.

In entrambi i casi le reti compiono un buon lavoro nell'identificare gran parte dei pixel ma alcune limitazioni appaiono evidenti fin da subito. Gran parte degli errori di classificazione avvengono tra classi simili (in particolare in presenza di `mediumRocks`) e per le quali è difficile stabilire contorni precisi. Anche in fase di classificazione manuale non sempre è stato possibile stabilire con certezza la categoria di un gruppo di pixel e il fenomeno si ripresenta anche in segmentazione con errori rispetto alle etichette, anche quando in realtà potrebbero non esserlo.

Sviluppato da esperti ricercatori, DeepLabv3+ ha superato i risultati ottenuti da reti allo stato dell'arte e rappresenta un punto di riferimento nell'ambito della segmentazione semantica. Il modello proposto, seppur discreto, non sarà in grado di raggiungere prestazioni simili e risultati soddisfacenti per applicazioni di classificazione di terreno marziano.

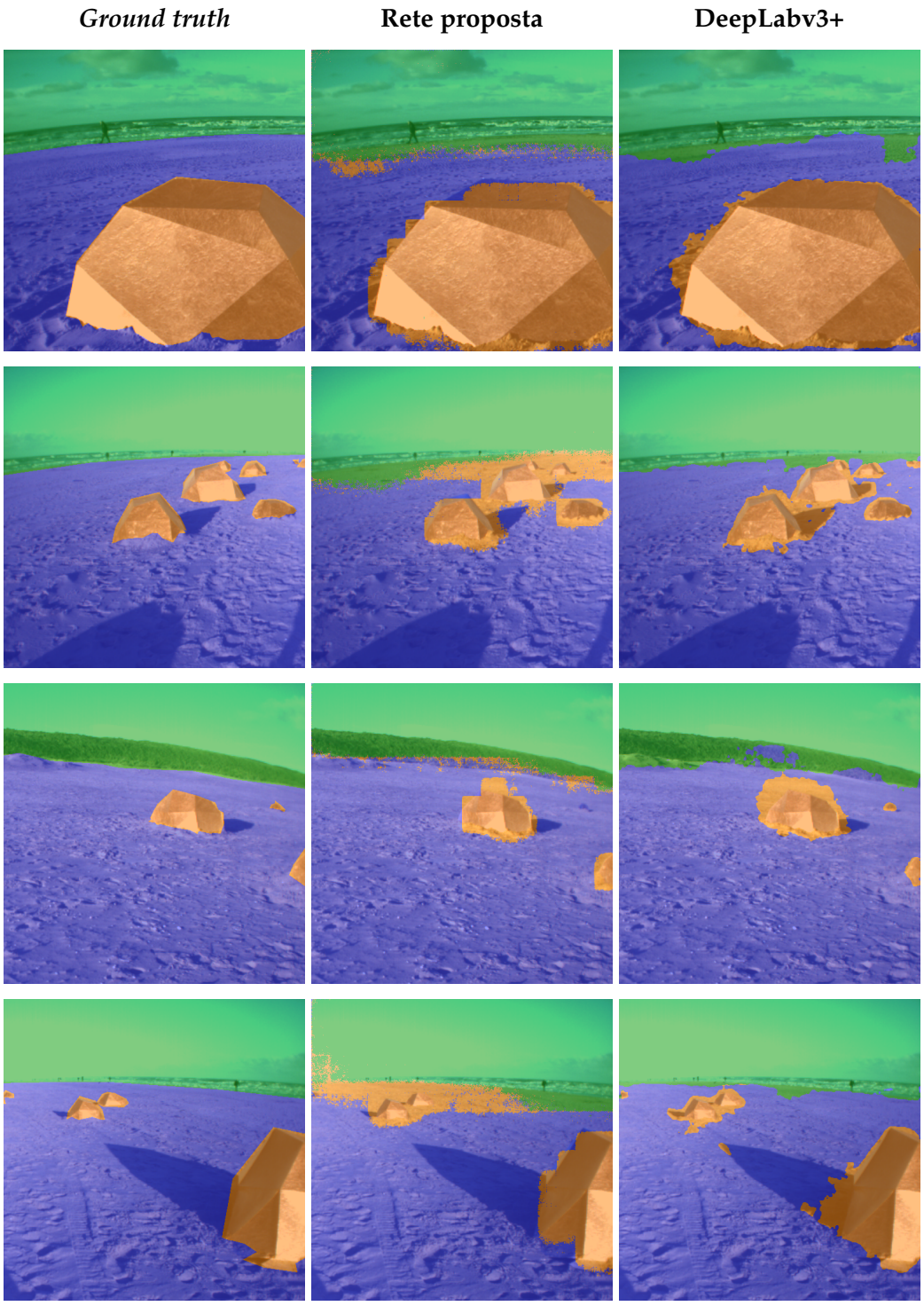
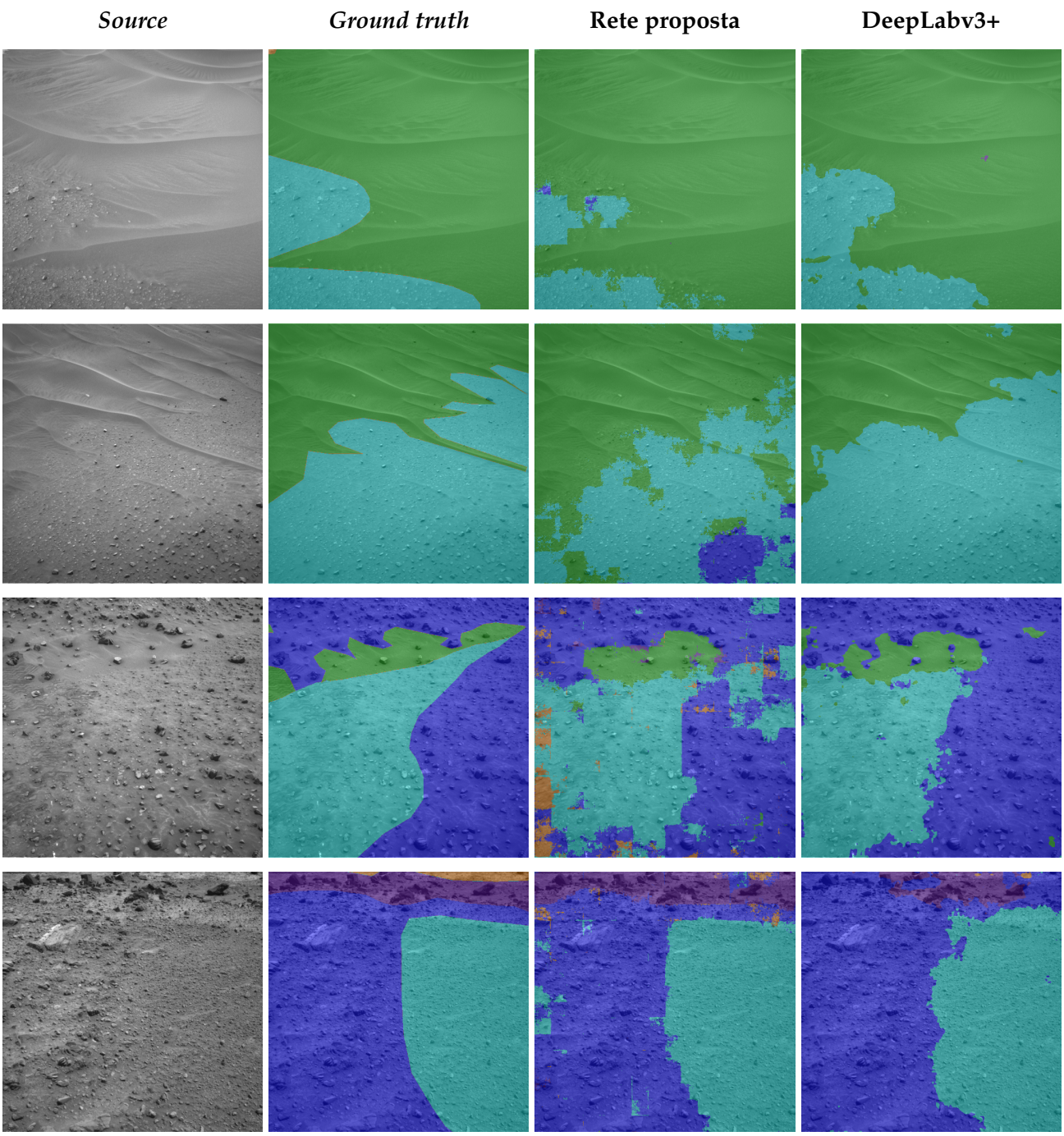
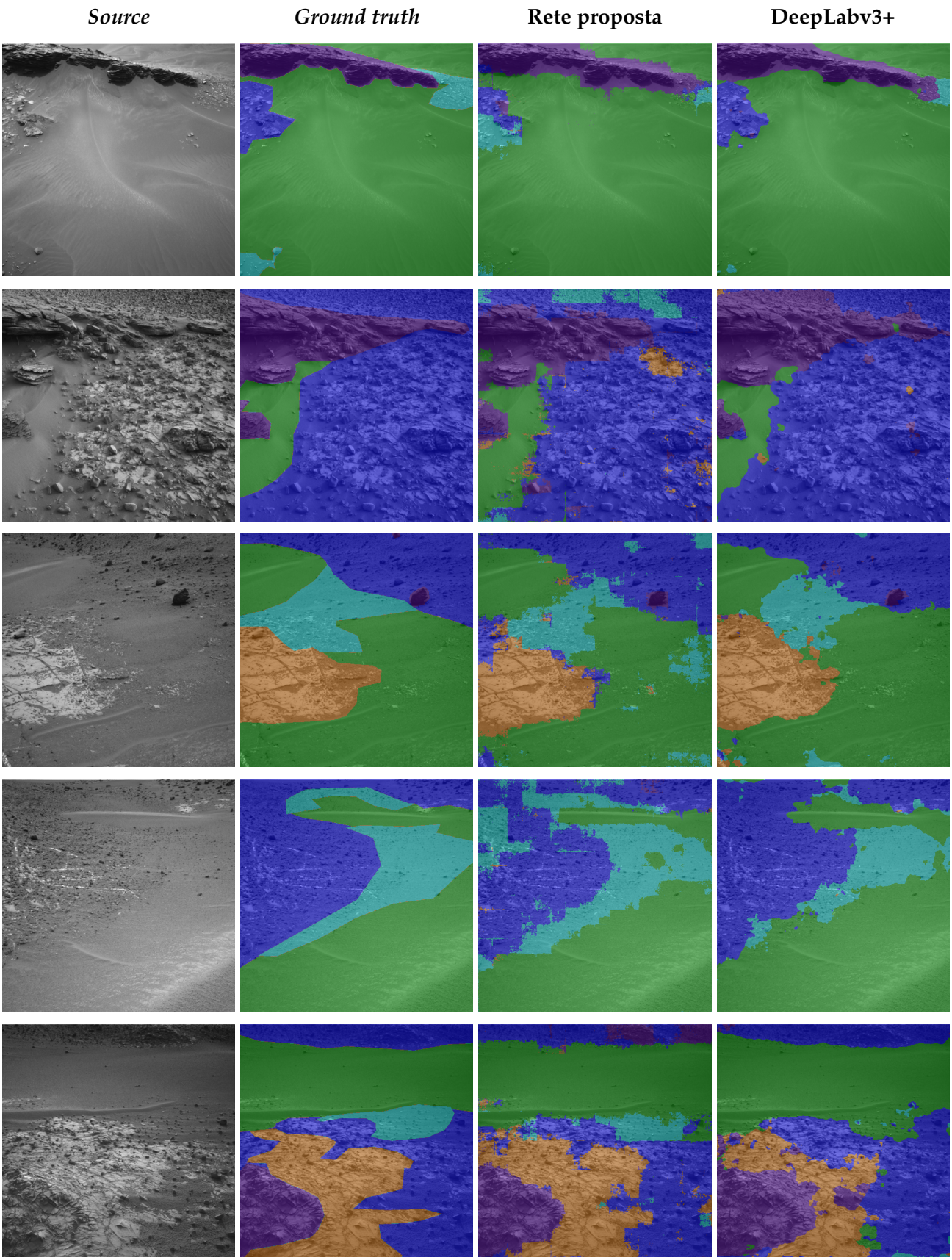


Figura 5.1: Risultati della segmentazione, dataset terrestre.

Tabella 5.7: Dataset marziano: legenda

sand	bedrock	smallRocks	mediumRocks	largeRocks




Source
Ground truth
Rete proposta
DeepLabv3+

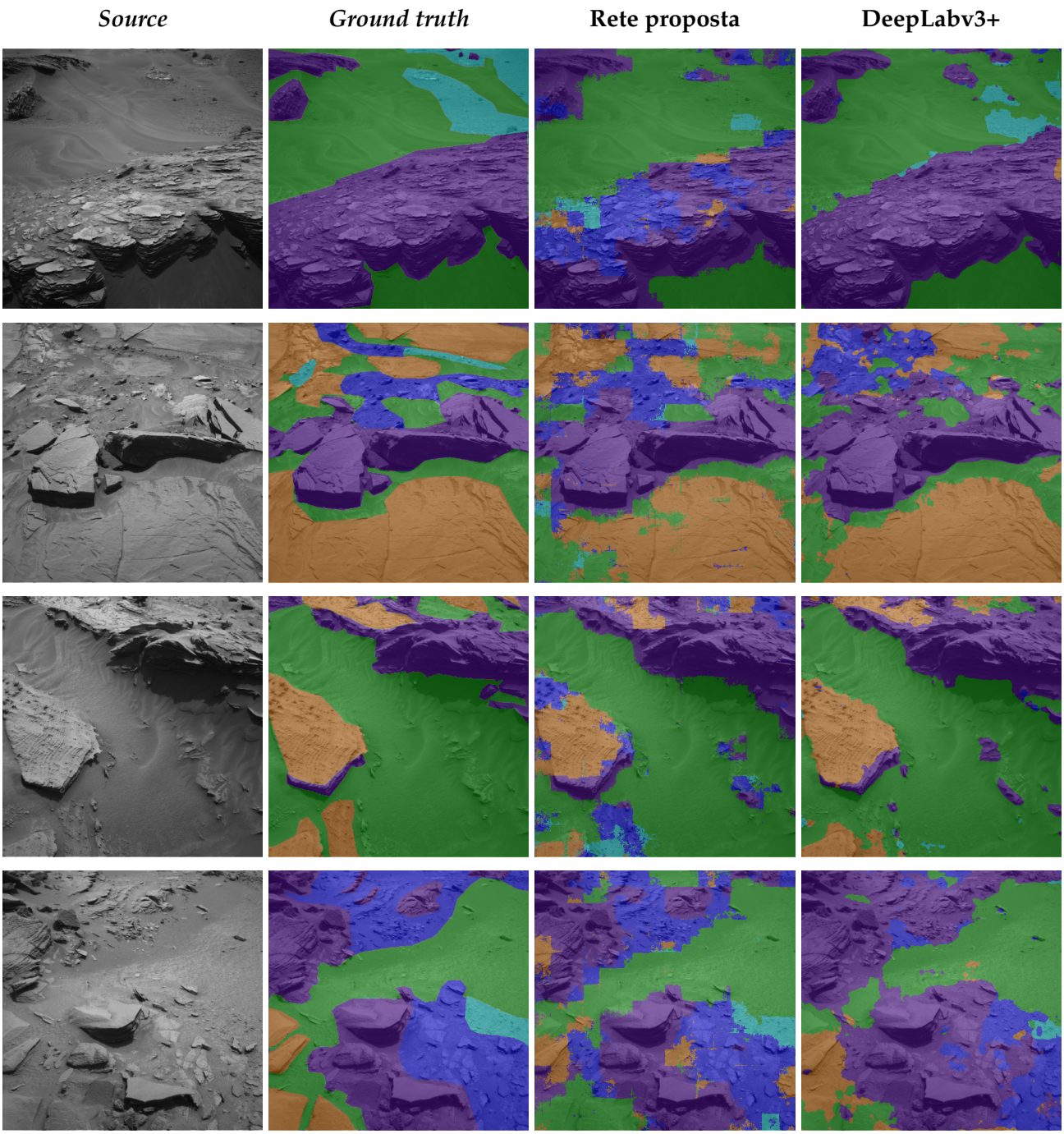


Figura 5.2: Risultati della segmentazione, dataset marziano.

CAPITOLO 6

Conclusioni

È stata presentata una tecnica di segmentazione semantica di immagini del terreno marziano basata su reti neurali convoluzionali e ne sono state valutate le prestazioni. Sono stati presi in considerazione due distinti dataset: un primo, di simulazione del terreno marziano, ed un secondo, di immagini catturate da Curiosity. Le architetture impiegate sono invece: DeepLabv3+, una delle più performanti attualmente disponibili, ed una più semplice rete realizzata per l'occasione. L'implementazione è stata condotta seguendo la procedura delineata: preparazione del dataset, addestramento, test. Definite le classi, è stata effettuata la classificazione manuale di ogni singola porzione di immagine ed il successivo *data augmentation*, per concludere con la suddivisione dei dataset in sottoinsiemi di addestramento, test, validazione. Condotti gli addestramenti per ciascuna delle quattro differenti combinazioni di dataset e rete, sono infine valutate le prestazioni dei modelli.

I risultati ottenuti mettono a risalto le buone accuratezze delle reti, in particolare di DeepLabv3+, in semplici problemi di classificazione dove le classi sono in numero ridotto e non sono affette da ambiguità, come nel caso del dataset terrestre. Il dataset marziano mette invece le reti in seria difficoltà: l'elevata variabilità nell'aspetto visivo delle classi, unita all'addestramento su un dataset non particolarmente esteso, genera una segmentazione robusta ma non sempre affidabile. È però doveroso evidenziare l'effetto delle difficoltà incontrate in fase di segmentazione manuale: non sempre è stato possibile tracciare con sicurezza i contorni, o semplicemente associare i pixel alla classe corretta, a causa della mutevolezza delle classi. Questi errori hanno trovato ri-

percussione nelle segmentazioni con rete, per le quali la scarsa approssimazione dei bordi si è rivelato essere uno dei problemi maggiori. È inoltre evidente come DeepLabv3+ risulti più performante della rete proposta e sia quindi da preferirsi nell'applicazione reale, dove tempi di addestramento, seppur lunghi, non sono di ostacolo. Le prestazioni nettamente superiori sono da attribuirsi in parte al numero di parametri addestrabili, in parte alla struttura della rete ed alla concatenazione di *features* di livello differente attuata dal decoder di DeepLabv3+.

Labeling e addestramento sono due ulteriori aspetti critici. La classificazione manuale richiede il lavoro di uno o più operatori a terra ed una abbondante quantità di immagini, possibilmente catturate in-situ, oltre ad una notevole disponibiltà di tempo. Il dataset dev'essere preparato con attenzione e la rete scelta, o creata, appositamente. Operazioni non eseguibili autonomamente da una CPU, compreso l'addestramento, il quale dev'essere anch'esso effettuato offline data la notevole richiesta di risorse computazionali. Fortunatamente, l'inferenza non necessita di elevata potenza e può essere effettuata da CPU a basse prestazioni in intervalli di tempo di qualche decimo di secondo. Ciò implica la possibilità di affidare l'inferenza al computer di bordo, una volta ricevuta da terra la rete addestrata. A tal proposito, è opportuno evidenziare come dalla segmentazione semantica possano trarre beneficio sia operatori a terra, per una più rapida definizione delle traiettorie, sia robot mobili, per soddisfare requisiti di navigazione autonoma. Associando le distanze derivate dalle stereo-camere ai punti immagine classificati, rover marziani possono essere resi in grado di distinguere ostacoli da aree transitabili, valutare le possibili traiettorie ed agire di conseguenza. Il vantaggio della segmentazione semantica effettuata su immagini NAVCAM sta nella distanza dalla quale è possibile individuare eventuali ostacoli. A differenza dell'attuale modalità di *hazard avoidance*, basata su HAZCAM e limitata a piccole superfici nelle immediate vicinanze di Curiosity, quella qui valutata vedrebbe coinvolte superfici ben più distanti e contribuirebbe alla gestione del *path planning* locale e globale, eventualmente anche nelle circostanze imposte dalla navigazione autonoma.

La segmentazione semantica si è rivelata un potente strumento di classificazione del terreno. Immagini segmentate possono essere integrate nella generazione di mappe semantiche a supporto di *path planning* e *obstacle avoidance*, anche in presenza di modelli

addestrati non particolarmente accurati. Ripetendo l'inferenza su immagini catturate in istanti ravvicinati durante il moto del rover, diviene possibile mediare i risultati di classificazione e correggere la segmentazione di volta in volta. Tecniche di *data fusion* possono essere adottate per ridurre ulteriormente le incertezze.

Ulteriori possibili sviluppi futuri riguardano due aspetti. Il primo concerne la combinazione di dati di *wheel odometry* e segmentazione di terreno per la generazione di accurati modelli di previsione dello slittamento delle ruote, in misura simile a quanto implementato in SPOC. [2] Il secondo è relativo all'applicazione dell'apprendimento non-supervisionato, ossia su dataset non classificati, al caso della classificazione del terreno. Questo aspetto, in particolare, rappresenta l'obiettivo finale per futuri sistemi di navigazione autonoma, qualora fossero supportati da computer di bordo di adeguata potenza.

APPENDICE A

Listato

```
%% Data setup

dataDir = fullfile('c:\','Users','user','Documents','Thesis'
    ,'Data');
imDir = fullfile(dataDir,'Training','Mars');
pxDir = fullfile(dataDir,'Training','Mars','PixelLabelData')
    ;

% Earth dataset
% classNames = [
%     "sand"
%     "rock"
%     "background"
%     ];

% Mars dataset
classNames = [
    "sand"
    "bedrock"
    "smallRocks"
    "largeRocks"
```

```
    "mediumRocks"
    ];

pixelLabelID = [1 2 3 4 5];

imds = imageDatastore(imDir);
pxds = pixelLabelDatastore(pxDir, classNames, pixelLabelID);


% Splitting dataset

Q = numel(imds.Files);
trainRatio = 0.75;
valRatio = 0.15;
testRatio = 0.15;

[trainingIdx, valIdx, testIdx] = dividerand(Q, trainRatio,
   valRatio, testRatio);

trainingImages = imds.Files(trainingIdx);
valImages = imds.Files(valIdx);
testImages = imds.Files(testIdx);

trainingLabels = pxds.Files(trainingIdx);
valLabels = pxds.Files(valIdx);
testLabels = pxds.Files(testIdx);

imdsTrain = imageDatastore(trainingImages);
imdsVal = imageDatastore(valImages);
imdsTest = imageDatastore(testImages);
```

```
pxdsTrain = pixelLabelDatastore(trainingLabels, classNames,
    pixelLabelID);
pxdsVal = pixelLabelDatastore(valLabels, classNames,
    pixelLabelID);
pxdsTest = pixelLabelDatastore(testLabels, classNames,
    pixelLabelID);

pximdsTrain = pixelLabelImageDatastore(imdsTrain, pxdsTrain)
    ;
pximdsVal = pixelLabelImageDatastore(imdsVal, pxdsVal);


%% Network setup

% Setting image size and number of classes
imageSize = [512 512 3];
numClasses = 3;

% Balancing classes
tbl = countEachLabel(pxds);
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;

% Loading layers
layers = deeplabv3plusLayers(imageSize, numClasses, "
    resnet18");
pxLayer = pixelClassificationLayer('Name', 'labels', '
    Classes', tbl.Name, 'ClassWeights', classWeights);
layers = replaceLayer(layers, "classification", pxLayer);

layers = [
```

```
imageInputLayer([512 512 3],"Name","imageinput")
convolution2dLayer([3 3],32,"Name","conv_1","Padding",[1
    1 1 1],"Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_1")
reluLayer("Name","relu_1")
convolution2dLayer([3 3],64,"Name","conv_2","Padding",[1
    1 1 1],"Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_2")
reluLayer("Name","relu_2")
convolution2dLayer([3 3],128,"Name","conv_3","Padding
   ",[1 1 1 1],"Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_3")
reluLayer("Name","relu_3")
convolution2dLayer([3 3],256,"Name","conv_4","Padding
   ",[1 1 1 1],"Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_4")
reluLayer("Name","relu_4")
convolution2dLayer([3 3],512,"Name","conv_5","Padding
   ",[1 1 1 1],"Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_5_2")
reluLayer("Name","relu_5_2")
transposedConv2dLayer([3 3],512,"Name","transposed-
   conv_0","Cropping","same","Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_0")
reluLayer("Name","relu_0")
transposedConv2dLayer([3 3],256,"Name","transposed-
   conv_1","Cropping","same","Stride",[2 2])
batchNormalizationLayer("Name","batchnorm_5_1")
reluLayer("Name","relu_5_1")
transposedConv2dLayer([3 3],256,"Name","transposed-
   conv_2","Cropping","same","Stride",[2 2])
```

```
    batchNormalizationLayer("Name","batchnorm_6")
    reluLayer("Name","relu_6")
    transposedConv2dLayer([3  3],256,"Name","transposed-
        conv_3","Cropping","same","Stride",[2  2])
    batchNormalizationLayer("Name","batchnorm_7")
    reluLayer("Name","relu_7")
    transposedConv2dLayer([3  3],3,"Name","transposed-conv_4
        ","Cropping","same","Stride",[2  2])
    batchNormalizationLayer("Name","batchnorm_8")
    reluLayer("Name","relu_8")
    softmaxLayer("Name","softmax")
    pixelClassificationLayer('Name', 'labels', 'Classes',
        tbl.Name, 'ClassWeights', classWeights)
    ];


%% Network training

opts = trainingOptions('sgdm', ...
    'InitialLearnRate', 1e-2, ...
    'MaxEpochs', 15, ...
    'MiniBatchSize', 10, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', pximdsVal, ...
    'ValidationFrequency', 20, ...
    'ValidationPatience', 5, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 5, ...
    'LearnRateDropFactor', 0.5, ...
    'Plots', 'training-progress');
```

```
net = trainNetwork(pximdsTrain, layers, opts);


%% Network evaluation

evalDir = fullfile('c:\','Users','ukaudio','Documents','
    Thesis','Code','beachEval');
pxdsResults = semanticseg(imdsTest, net, "WriteLocation",
    evalDir, 'MiniBatchSize', 4);
metrics = evaluateSemanticSegmentation(pxdsResults, pxdsTest
    );
```

Bibliografia

[1] Cuebong Wong, Erfu Yang, Xiu-Tian Yan, Dongbing Gu, *Adaptive and Intelligent Navigation of Autonomous Planetary Rovers — A Survey*, 2017.

[2] Brandon Rothrock, Ryan Kennedy, Chris Cunningham, Jeremie Papon, Matt Heverly, Masahiro Ono, *SPOC: Deep Learning-based Terrain Classification for Mars Rover Missions*, AIAA Space Forum, 2016.

[3] Clark F. Olson, Larry H. Matthies, John R. Wright, Ron Li, Kaichang Di, *Visual Terrain Mapping for Mars Exploration*.

[4] Bruno Siciliano, Oussama Khatib, *Handbook of Robotics*, Springer, 2016.

[5] Hamed Habibi Aghdam, Elnaz Jahani Heravi, *Guide to Convolutional Neural Networks*, Springer, 2017.

[6] Charu Aggarwal, *Neural Networks and Deep Learning*, Springer, 2018.

[7] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*, 2017.

[8] Evan Shelhamer, Jonathan Long, Trevor Darrell, *Fully Convolutional Networks for Semantic Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016.

[9] Olaf Ronneberger, Philipp Fischer, Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015.

[10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, 2018.

[11] Mars Science Laboratory Software Interface Specification, *Camera & LIBS Experiment Data Record (EDR) and Reduced Data Record (RDR) Data Products*, 2018.

[12] MATLAB Documentation, *Options for training deep learning neural network*, <mathworks.com/help/deeplearning/ref/trainingoptions.html>, accesso: ottobre 2019.

[13] MATLAB Documentation, *Evaluate semantic segmentation data set against ground truth*, <mathworks.com/help/vision/ref/evaluatesemanticsegmentation.html>, accesso: ottobre 2019.

[14] MATLAB Documentation, *Layers of a convolutional neural network*, <mathworks.com /help/deeplearning/ug/layers-of-a-convolutional-neural-network.html>, accesso: ottobre 2019.

[15] *Where is Curiosity?*, <mars.nasa.gov/msl/mission/whereistherovernow>, accesso: ottobre 2019.

[16] *Seventeen Cameras on Curiosity*, <nasa.gov/mission_pages/msl/ multimedia/malin-4.html>, accesso: ottobre 2019.

[17] *Katwijk Beach Planetary Rover Dataset*, <robotics.estec.esa.int/datasets/ katwijk-beach-11-2015>, accesso: ottobre 2019.

[18] *A technical report on convolution arithmetic in the context of deep learning*, <github.com/vdumoulin/conv_arithmetic>, accesso: novembre 2019.

[19] *Backpropagation algorithm*, <google-developers.appspot.com/machine-learning /crash-course/backprop-scroll>, accesso: novembre 2019.