UNIVERSITÀ DI PISA

Corso di laurea in Informatica

Utilizzo di Reti Convoluzionali su planimetrie per l'individuazione del posizionamento ottimale dei punti luce

Tutore Accademico:
Prof. Vincenzo Lomonaco

Laureando:
Alessandro Querci
mat. 578615

Tutor Aziendale:
Dott. Davide Moroni
Dott. Ali Reza Omrani

Anno accademico 2022/2023

Indice

1 Introduzione

Il tirocinio è stato effettuato presso il CNR-ISTI di Pisa nel periodo Dicembre 2022 - Febbraio 2023, per un totale di circa 320 ore.
Il dottor Moroni si è occupato principalmente dell'aspetto organizzativo e di supervisione generale, mentre il tutor Omrani è stato il punto di riferimento per l'apprendimento e l'uso delle Reti Neurali. In particolare le conversazioni con il dottor Omrani sono state tutte in inglese, in quanto non di origine italiana, ma non ritengo sia stato un problema per il mio livello di inglese ma al contrario utile per migliorarne il parlato.

Le ore sono state svolte per la maggior parte in presenza, con una piccola quota di ore da remoto per permettermi gestire più agilmente gli spostamenti tra Pisa e residenza durante il fine settimana.
Durante il tirocinio mi è stato fornito un posto nella stessa stanza del dottor Omrani, di modo da poter chiedere chiarimenti in ogni momento, un badge per accedere al complesso del CNR, connessione internet ed accesso al centro di calcolo dell'ISTI. Ho comunque svolto la gran parte del lavoro in autonomia, confrontandomi con i tutor per le decisioni più importanti e per dubbio o problemi riscontrati.

Il tirocinio ha previsto l'utilizzo del linguaggio di programmazione Python con il framework di Tensorflow-Keras, ed altre librerie, per la costruzione ed allenamento di Reti Neurali Convoluzionali. L'utilizzo del centro di calcolo ISTI ha inoltre previsto l'utilizzo di Slurm e Docker per gestire il carico di lavoro delle sessioni di allenamento e test delle reti neurali.

L'obbiettivo del tirocinio è tentare di utilizzare tecnologie di Computer Vision e Machine Learning, in particolare di segmentazione di immagini, per trovare la disposizione ottimale dei dispositivi di illuminazione in un appartamento o ufficio, in modo da coprire la maggior parte dell'area possibile. Questo può contribuire a ridurre il numero di lampade che vengono utilizzate in casa e, di conseguenza, può ridurre il consumo di energia elettrica da parte delle abitazioni. Il tema è particolarmente importante nel periodo corrente sia dal punto di vista ambientale, nel tentativo di ridurre consumi ed emissioni per facilitare la transizione ecologica dell'energia, che economico, dato che i prezzi dell'energia hanno avuto una pesante inflazione per vari fattori geopolitici recenti.

In quanto il progetto è più affine ad una ricerca sperimentale che allo sviluppo di un software commerciale, non avendo veri e propri precedenti o studi su cui basarsi per garantirne il successo, non vi erano aspettative concrete che

l'approccio utilizzato potesse portare ai risultati sperati ma piuttosto che potesse costituire un punto di inizio per il futuro sviluppo o successivi studi su una tecnologia simile.

Inoltre, sia il tempo che i dati disponibili non sarebbero stati sufficienti per realizzare modelli particolarmente complessi e sono state fatte alcune assunzioni e semplificazioni sul dataset per riuscire a rientrare nei tempi previsti dal tirocinio. Il dataset è stato realizzato in collaborazione con Giuseppe Fusco del Centro Ricerche TQV del CNR.

Per semplificazione, questo progetto ha previsto l'utilizzo di un solo tipo di dispositivo illuminate come fonte di luce per semplicità ma ovviamente in una situazione realistica sarebbe molto più utile avere una più ampia scelta in modo da coprire efficacemente aree più piccole senza sprechi, senza dimenticarsi dell'esistenza di dispositivi orientabili, a muro o con supporto a terra che hanno diversi coni di luce. Inoltre in base al suo utilizzo, ogni stanza potrebbe avere un diverso livello di luce ottimale, anche questo dettaglio è stato omesso per non complicare il problema ma potenzialmente i dati di allenamento sarebbero potuti essere annotati con questa informazione aggiuntiva. Anche la presenza di finestre, porte e simili è stata omessa per le ragioni già citate, ma sono elementi importanti che influenzano l'illuminazione significativamente. Dettaglio finale da notare è che anche l'altezza del soffitto delle stanze, che per assunzione abbiamo uniformato a 3 metri, è un dettaglio importante che influenza la disposizione ottimale delle luci.

Nel complesso, la rete risultante è capace di classificare gli elementi della planimetria (esterni/muri/interni) con minimo margine di errore e di posizionare una fonte di luce correttamente nella maggior parte delle stanze di dimensione medio-piccola, ma purtroppo in stanze di dimensione più grande fallisce nel comprendere il concetto di stanza e dispone le luci solo in relazione ai muri più vicini, lasciando il centro scoperto.
Analizzando i risultati, sia sul dataset di training che di validation, il problema sembra essere legato al fatto che la rete va a basarsi sulla presenza di muri per riconoscere il concetto di stanza in cui disporre luci, nonostante la segmentazione di "interno" venga eseguita correttamente. In stanze grandi, il "campo visivo" della rete non è sufficiente per percepire la stanza nel suo complesso e quindi prova a piazzare delle luci vicino ai muri, lasciando scoperta la parte centrale. Inoltre, non sembra avere imparato che le luci devono avere una minima distanza fra loro al fine di ottimizzarne la distribuzione, portando in alcuni casi ad avere luci molto vicine tra loro.

Su opinione personale e dei tutor, uno dei problemi principali risiede nel dataset, in particolare sulle annotazioni. La disposizione delle luci nelle annotazioni fornite dall'esperto non ci sono sembrate particolarmente ottimali e molto spesso eterogenee fra loro. A sua discolpa, c'è da dire che il lavoro di annotazione è stato eseguito in tempi stretti ed in un periodo in cui era abbastanza impegnato, senza l'ausilio di CAD e altri strumenti con cui è solito lavorare. Inoltre la mancanza di dimensioni esatte delle stanze, nonostante i nostri sforzi di uniformare le planimetrie e fare assunzioni per mantenere il lavoro più semplice possibile, potrebbe aver messo in difficoltà. Ho provato a confrontarmi con l'esperto per capire se l'eterogeneità era frutto di errore accidentale, ma a sua opinione non erano errori ma semplicemente diversi approcci per raggiungere una buona disposizione con quell'unico tipo di luce. In assenza di un metodo efficace per stabilire quali disposizioni erano effettivamente corrette e quali soffrivano eccessivamente di errore di bias dell'esperto, oltre che di tempo per far annotare nuovamente le immagini, ho cercato di fare il possibile per vedere i limiti di un architettura allenata su questi dati.

Considerando i vari problemi sui dati di partenza, che sarebbero potuti essere evitabili con un dataset commissionato, ed il fatto che non esistano al momento sistemi simili con cui fare paragoni o da prendere come riferimento, i tutor hanno comunque ritenuto che i risultati ottenuti siano interessanti e possano costituire un punto di partenza per ulteriori ricerche e prototipi sul tema dell'utilizzo di reti neurali per problemi di questo tipo.
Dal parte mia, anche se mi dispiace che la rete non abbia raggiunto uno stato utile all'utilizzo pratico dopo tutto il lavoro svolto, ho imparato molto su questo ramo dell'intelligenza artificiale e ritengo che l'esperienza acquisita con la pratica sul campo mi aiuterà a comprendere meglio i temi della laurea magistrale in Intelligenza Artificiale che intendo intraprendere come proseguimento. Ho iniziato il tirocinio con un conoscenza molto minimale di Python, e librerie annesse, e quasi nulla al proposito delle reti neurali più avanzate, mentre adesso sento di avere molta confidenza su tali strumenti.

2 Background

2.1 Origini delle Reti Neurali Artificiali e termine Intelligenza Artificiale

Il settore dell'IA consiste nell'indagine tecnologica e intellettuale che mira al raggiungimento dei seguenti obbiettivi:

- Costruzione di macchine intelligenti, sia che operino come l'uomo sia che diversamente.
- Formalizzazione della conoscenza e meccanizzazione del ragionamento, in tutti i settori di azione dell'uomo.
- Comprensione mediante modelli computazionali della psicologia e comportamento di uomini, animali e agenti artificiali.
- Rendere il lavoro con il calcolatore altrettanto facile e utile del lavoro con persone capaci, collaborative e possibilmente esperte.

Warren S. McCulloch e Walter Pitts proposero nel 1943 il primo modello di neuroni artificiali, attingendo alla conoscenza della fisiologia e delle funzioni di base dei neuroni, alla logica proposizionale e alla teoria della computabilità di Alan M. Turing. Uno dei risultati più significativi consentì di mostrare come ogni funzione calcolabile potesse essere elaborata da una qualche rete di neuroni connessi.

Successivamente nel 1949 lo psicologo Donald Olding Hebb pubblica Organization of Behavior, dimostrando come una semplice regola di aggiornamento per modificare le forze di connessione fra i neuroni potesse dare luogo a processi di apprendimento. Nel 1956, durante la conferenza di Darthmouth viene ufficialmente coniato il termine "Intelligenza Artificiale". Gli anni successivi al seminario di Dartmouth furono pieni di aspettative, alimentate anche dai successi dovuti ai miglioramenti vertiginosi dei supporti informatici utilizzati. L'enfasi di questo primo periodo si pose quindi sui meccanismi generali di ricerca e su una concezione limitata della nozione di intelligenza, quale per esempio l'abilità di giocare a scacchi o la capacità di risolvere problemi matematici.

Nel 1958 viene proposta da Rosenblatt la prima rete neurale: Perceptron. Le basi dell'apprendimento automatico sono adesso realtà. Perceptron di Rosenblatt possiede uno strato di nodi di input e un nodo di output. I pesi sinaptici (un peso indica la forza di una connessione fra due nodi) sono dinamici, permettendo alla macchina di apprendere, in un modo sommariamente simile, anche se molto più elementare, a quello delle reti neurali biologiche. Questo primo modello è di tipo feedforward: gli impulsi si propagano in un'unica direzione, in avanti.

Il suo campo di applicazione è molto limitato, classificazione binaria di forme e calcolo di semplici funzioni. Si rivela tuttavia incapace di riconoscere stimoli

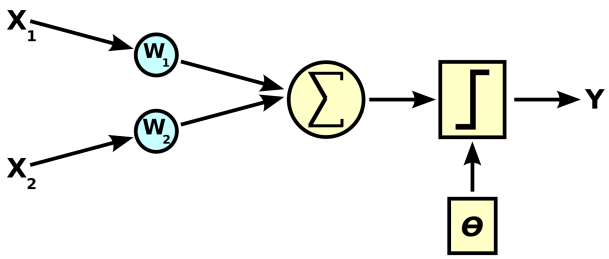


Figura 1: Struttura del Perceptron di Rosenblatt

Fonte: `https://it.wikipedia.org/wiki/Percettrone`

visivi anche molto semplici, portando le reti neurali ad essere messe da parte a favore dei modelli a ragionamento simbolico.

Solo nel 1986 si assiste ad un ritorno delle reti neurali, Rumelhart, McClelland e Hinton espandono il Perceptron multistrato, che introduceva uno strato intermedio e non-linearità all'iniziale Perceptron, combinandolo con algoritmi basati sulla retro-propagazione: modificando i pesi delle connessioni fra nodi in base alla differenza fra risultato ottimale e risultato ottenuto si riesce a migliorare l'apprendimento della rete neurale in stadi successivi. Questo metodo ha permesso di superare i limiti delle reti neurali semplici, che erano incapaci di apprendere funzioni complesse e non lineari, e pose le basi per l'evoluzione in reti neurali profonde.

Negli anni a seguire le reti neurali profonde hanno continuato a evolversi e a migliorare le loro prestazioni, grazie anche all'impiego di nuove tecniche e algoritmi e hardware più potenti. L'espansione di internet ha inoltre permesso di avere molti più dati a disposizione e una maggiore condivisione e collaborazione fra ricercatori, anche appartenenti a diverse discipline.

2.2 Machine Learning e Reti Neurali

Il Machine Learning moderno è un'area che combina l'esigenza di creare macchine in grado di apprendere con i nuovi strumenti adattivi e statistici che di continuo vengono migliorati o inventati. Nasce dalla necessità che abbiamo di analizzare una crescente quantità di dati empirici combinata con la difficoltà che si ha nel "programmare l'intelligenza" a priori.
L'unica scelta possibile è creare macchine in grado di apprendere ed evolversi in modo da rendersi adattive in maniera autonoma e dunque senza avere bisogno di essere programmate per risolvere problemi troppo specifici. A differenza degli

algoritmi tradizionali basati su euristiche, il ML non richiede una programmazione esplicita delle regole per risolvere un determinato problema, ma invece utilizza algoritmi che permettono alle macchine di imparare automaticamente dalle informazioni presenti nei dati.

Questo rende il ML estremamente versatile e adatto a una vasta gamma di applicazioni, dall'analisi dei dati al riconoscimento di immagini, al controllo dei robot, alla previsione del mercato azionario e molto altro ancora. Inoltre, a differenza degli algoritmi tradizionali, che richiedono un continuo aggiornamento da parte degli sviluppatori per rimanere efficaci, il ML è in grado di adattarsi e migliorare costantemente le sue prestazioni attraverso il feedback che riceve dai dati.
In generale si cerca di creare modelli in grado di evolversi con una conoscenza a priori molto limitata ma sufficiente da permettere al modello di imparare e ampliare tale conoscenza.
Diversi approcci all'apprendimento automatico sono stati sviluppati ed in continua evoluzione, come algoritmi basati su Alberi decisionali, Support Vector Machines (SVMs) o K-nearest neighbors (KNN). Quello che tuttavia sta riscuotendo maggiore successo e popolarità negli ultimi anni è l'utilizzo di Reti Neurali.

Le reti neurali moderne sono costituite da una serie di layer o strati, ognuno dei quali svolge una specifica trasformazione dell'input per produrre l'output finale della rete. Nelle architetture più semplici, si distinguono tre tipi di layer principali: i layer di input, i layer nascosti e i layer di output.

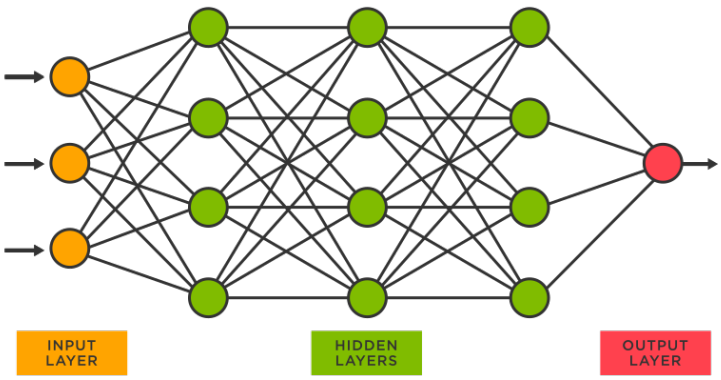


Figura 2: Schema di una rete neurale a 5 strati, con 3 strati densi nascosti

Fonte: `https://medium.com/@srempe/an-introduction-to-neural-networks-8c61b880f358`

Il layer di input riceve l'input grezzo (ad esempio, un'immagine o un testo) e lo trasformano in una rappresentazione numerica che può essere elaborata dalla rete.

Nei layer nascosti avviene la maggior parte dell'elaborazione dei dati. Sono solitamente composti da più layer, ognuno dei quali svolge una trasformazione specifica dell'input. Ad esempio, in una rete neurale convoluzionale, i layer nascosti possono applicare convoluzioni multiple per rilevare caratteristiche sempre più complesse dell'immagine, come bordi degli oggetti o parti del viso. Negli strati nascosti possiamo trovare layer di estrazione di feature come i convoluzionali, layer di normalizzazione come i Batch Normalization o semplicemente layer Fully Connected in cui ogni tutti i neuroni sono connessi tra loro.

Infine, il layer di output trasforma l'output dell'ultimo layer nascosto in una forma adatta per la predizione del compito specifico. Ad esempio, in un problema di classificazione, il layer di output può essere costituito da un insieme di neuroni, ognuno dei quali rappresenta una possibile classe di appartenenza.

Le connessioni tra i vari strati della rete neurale sono un aspetto fondamentale del suo funzionamento. I neuroni di un layer sono connessi a i neuroni del layer successivo attraverso i pesi sinaptici, che determinano l'importanza relativa di ogni connessione.
Nel caso di layer convoluzionali, ogni neurone è connesso solo a una sotto-regione dell'input del layer precedente, attraverso uno o più filtri (o kernel) di dimensione ridotta rispetto all'input. Il filtro è una matrice di pesi sinaptici, che vengono applicati in modo convolutivo all'input per produrre una mappa delle attivazioni che diminuisce la dimensione dell'input ma ne aumenta la dimensionalità in base al numero di filtri utilizzati.

I neuroni di un layer ricevono in input i segnali dai neuroni del layer precedente, moltiplicati per i pesi sinaptici associati a ciascuna connessione. Questi valori pesati vengono quindi sommati e passati attraverso una funzione di attivazione, che determina l'output del neurone. Ad ogni input viene inoltre applicato un leggero rumore, chiamato bias, affetto anch'esso dai pesi sinaptici e che permette alla rete di essere più flessibile durante l'apprendimento.
In questo modo, l'informazione viene elaborata in modo progressivo e sempre più complesso man mano che si procede nei layer della rete. I pesi sinaptici vengono solitamente inizializzati a caso e poi aggiornati durante l'allenamento in base al tipo di apprendimento utilizzato: prendendo ad esempio l'apprendimento

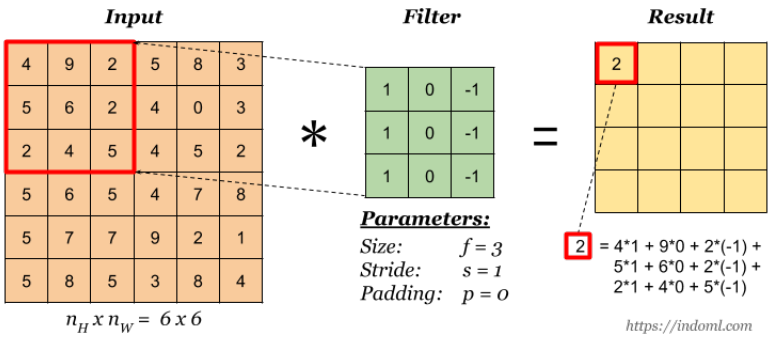


Figura 3: Esempio di convoluzione

Fonte: `https://dvgodoy.github.io/dl-visuals/Convolutions/`

supervisionato, i pesi tra le connessioni sono aggiornate in base agli errori commessi sulle predizioni durante l'allenamento, che vengono comparate con degli esempi corretti forniti.

2.2.1 Apprendimento supervisionato

L'apprendimento supervisionato (supervised learning) è un tipo di apprendimento automatico in cui il modello viene addestrato su un dataset etichettato, ovvero un dataset in cui i dati di input sono accompagnati da corrispondenti etichette (label) di output o valori target, al fine di apprendere una funzione che possa predire il corretto output per nuovi dati di input.
Il dataset viene solitamente suddiviso in training set, validation set e test set.

Il training set è il più grande dei tre e viene utilizzato durante l'allenamento dall'algoritmo per imparare i pattern e le relazioni presenti fra i dati di input e i dati target. Il training set è solitamente diviso a sua volta in batch, ovvero insiemi più piccoli di dati, che vengono processati a turno durante ogni epoch, ovvero un'iterazione su tutto il training set.

Alla fine di ogni epoch, il modello viene testato sul validation set. Questo insieme di dati non viene utilizzato dall'algoritmo per imparare, ma è necessario allo sviluppatore per valutare l'andamento dell'apprendimento e guidare la scelta degli iperparametri. Attraverso i risultati sul validation set è possibile osservare se il modello sta imparando i pattern e le relazioni come desiderato oppure sta semplicemente imparando a memoria i dati di training ed è incapace di analizzare correttamente dati mai visti.

Il test set infine è utilizzato per avere una valutazione finale sulle performance del modello, dopo aver eseguito allenamento sul training set e scelta degli iperparametri basandosi sul validation set.

L'obbiettivo del training con supervised learning è minimizzare l'errore fra i dati forniti dal supervisore e le predizioni ottenute, adattando i parametri interni del modello attraverso l'uso di un algoritmo basato sulla retro-propagazione.
Dopo aver processato una batch di dati l'algoritmo calcola la funzione di Loss, ovvero la funzione che quantifica la differenza fra la predizione dell'algoritmo sul dato e il risultato corretto. Viene chiamata anche funzione di Costo quando è calcolata sull'intero dataset anziché sui singoli campioni.

Due semplici esempi di funzione di Loss sono:

- Mean squared error (MSE) per i problemi di regressione:

$$Loss(\mathbf{y_{true}}, \mathbf{y_{pred}}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y_{true}}_i - \mathbf{y_{pred}}_i)^2$$

  Dove y_{true} è il valore di output atteso, y_{pred} è il risultato dell'algoritmo. Per ogni batch di dati, la differenza viene elevata al quadrato e poi presa la media aritmetica.

- Binary cross-entropy loss per i problemi di classificazione binaria:

$$Loss(\mathbf{y_{true}}, \mathbf{y_{pred}}) = -\frac{1}{N}\sum_{i=1}^{N}[y_{true} \times \log(y_{pred}) + (1 - y_{true})\log(1 - y_{pred})]$$

  Dove y_{true} è la label corretta (0 o 1), y_{pred} è la probabilità stimata che il dato appartenga alla classe 1 e log il logaritmo naturale.

A questo punto viene utilizzato un algoritmo di ottimizzazione chiamato Discesa del Gradiente (Gradient Descent), per permettere al modello di imparare da tali errori.
L'idea alla base della discesa del gradiente è quella di aggiornare iterativamente i parametri di un modello nella direzione di discesa più ripida della funzione di Loss. In altre parole, la discesa del gradiente è un metodo per trovare il minimo di una funzione prendendo passi nella direzione del gradiente negativo.

Un'altra visualizzazione comune utilizza due parametri per la funzione di costo, quindi in uno spazio tridimensionale, e mostra la presenza di minimi locali che

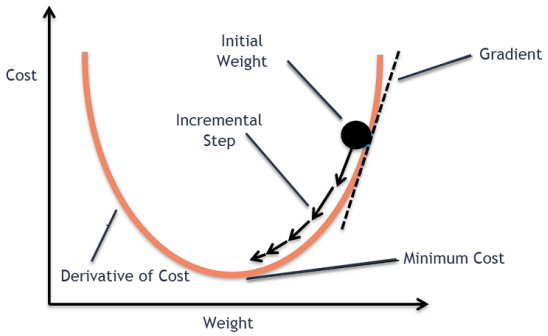


Figura 4: Discesa di gradiente in uno spazio 2D

Fonte: `https://www.researchgate.net/figure/Gradient-Descent-Algorithm-26_fig1_349573260`

possono portare a risultati non ottimali. Tuttavia è necessario puntualizzare che non è possibile rappresentare graficamente la discesa di gradiente rispetto a tutti i parametri di una rete neurale dato nelle reti profonde si raggiunge l'ordine dei milioni di parametri, pertanto si semplifica a questa rappresentazione 3D.

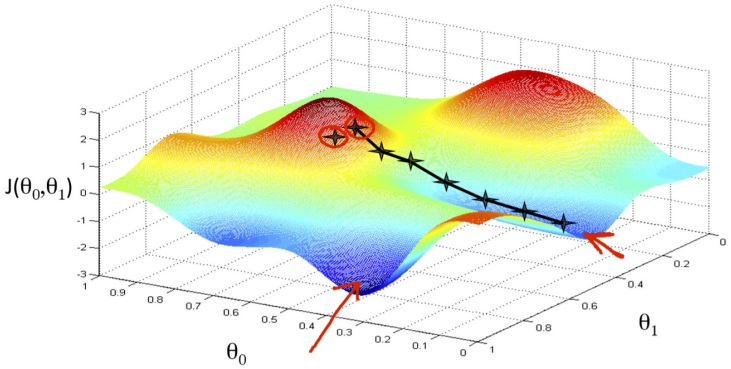


Figura 5: Discesa di gradiente in uno spazio 3D, in evidenza i minimi locali

Fonte: `https://towardsdatascience.com/an-intuitive-explanation-of-gradient-descent-83adf68c9c33`

Un parametro chiamato learning rate (tasso di apprendimento) regola l'estensione del passo nel gradiente eseguito ad ogni iterazione. Se il learning rate è molto basso, l'algoritmo convergerà più lentamente verso il minimo della funzione, ma se è troppo alto si rischia di divergere o oscillare vicino al minimo senza mai raggiungerlo. Per questo motivo alcuni algoritmi e tecniche di

allenamento prevedono un learning rate adattivo o una riduzione graduale per permettere di avvicinarsi velocemente al minimo con un learning rate alto e poi diminuirlo per avere una discesa più controllata.

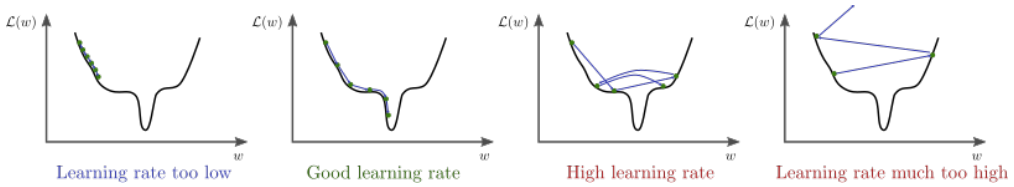


Figura 6: Effetto di diversi learning rate in prossimità di punti di minimo stretti.

Fonte: `https://www.bdhammel.com/learning-rates/`

Vi sono vari tipi di algoritmi per la discesa del gradiente, i più famosi nel campo delle reti neurali sono indubbiamente RMSProp, SGDM e Adam. Tutti e tre sono basati sulla Stocastich Gradient Descent (SDG) che calcola il gradiente ad ogni mini-batch anziché su tutto il dataset ad ogni iterazione.

- RMSProp, Root Mean Square Propagation, adatta il learning rate di ogni parametro basandosi su informazioni storiche sul gradiente. L'idea di base è di mantenere in memoria la media mobile del gradiente al quadrato per ogni parametro ed utilizzarla per regolare il learning rate durante l'aggiornamento del peso relativo al parametro. Più precisamente, ad ogni iterazione RMSProp calcola la media mobile del quadrato del gradiente per ogni parametro e divide il gradiente corrente per la radice di tale media sommata ad una costante epsilon (per evitare divisioni per 0, per ottenere il valore di aggiornamento del peso di ogni parametro.

$$\text{avg} = \text{decay\_rate} \times \text{avg} + (1 - \text{decay\_rate}) \times {g_t}^2$$
$$\theta_{t+1} = \theta_t - \text{learning\_rate} \times \frac{g_t}{\sqrt{\text{avg}} + \epsilon}$$

  Dato il quadrato del gradiente sarà sempre positivo, il parametro decay_rate (tasso di decadimento) permette di mantenere sotto controllo la crescita della media mobile.

- SGDM, Stochastic Gradient Descent with Momentum, si basa invece sul valore di aggiornamento del gradiente all'iterazione precedente e un parametro chiamato momentum. La formula per l'aggiornamento dei pesi diventa :

$$v_t = \text{momentum} \times v_{t-1} + (1 - \text{momentum}) \times g_t$$

$$w_t = w_{t-1} - \text{learning\_rate} \times v_t$$

Dove v_t è l'aggiornamento al passo t e g_t il gradiente.

Il vantaggio del momentum è di permettere alla discesa di proseguire nella stessa direzione dei passi precedenti, utile in casi dove il gradiente cambia direzione spesso o oscilla in quanto lo uniforma ed incoraggia a muoversi uniformemente nella stessa direzione.

- Adam, Adaptive Moment Estimation, fonde elementi da altri algoritmi che lo hanno preceduto. Adam si basa sull'utilizzo del momento primo e secondo del gradiente per regolare il learning rate di ogni parametro. Ad ogni iterazione viene calcolata la media mobile esponenziale (EMA) del gradiente e della sua radice quadrata per ogni parametro. Similarmente a RMSProp e SGDM, la regola di aggiornamento di Adam è:

$$m_t = \beta_1 \times m_t - 1 + (1 - \beta_1) \times g_t$$

$$v_t = \beta_2 \times v_t - 1 + (1 - \beta_2) \times g_t^2$$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

$$\theta_t + 1 = \theta_t - learning\_rate \times \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}$$

Dove β_1 e β_2 sono due iperparametri che agiscono come tassi di decadimento per il primo e secondo momento, mentre ϵ evita la divisione per zero.

2.2.2 Overfitting e regolarizzazione

Underfitting e overfitting sono problemi comuni nel ML che si verificano quando un modello non è in grado di generalizzare bene su nuovi dati. In quanto impossibili da evitare completamente, la necessità di trovare un equilibrio delle capacità di generalizzazione è chiamato "Compromesso Bias-Varianza".

L'errore di Bias deriva da assunzioni errate dell'algoritmo di apprendimento, mentre la varianza è un errore che deriva dall'eccessiva sensibilità a piccole fluttuazioni nella distribuzione dei dati.

L'underfitting si verifica quando un modello è troppo semplice e non può catturare i modelli sottostanti nei dati. Un elevato bias può far sì che un algoritmo manchi le relazioni rilevanti tra le feature estratte e gli esempi forniti, ciò comporta un modello che ha una scarsa performance sia sui dati di training che sui nuovi dati. In altre parole, il modello non è in grado di adattarsi bene ai dati di training e, pertanto, non può neanche generalizzare bene su dati mai visti.

L'overfitting si verifica quando un modello è troppo complesso ed è in grado di adattarsi molto bene ai dati di training, ma ha una scarsa performance sui nuovi dati. Una varianza elevata porta l'algoritmo a modellare eccessivamente il random noise per adattarsi ai dati dati di training. In altre parole, il modello ha memorizzato troppo i dati di training e non è in grado di generalizzare bene su dati mai visti.

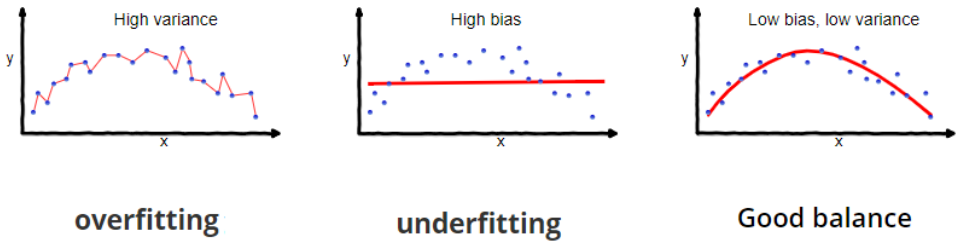


Figura 7: Esempi di overfitting e underfitting in problemi di regressione.

Fonte: `https://www.researchgate.net/figure/overfitting-vs-underfitting_fig7_342624204`

Esistono varie tecniche di regolarizzazione che possono aiutare a mantenere alta la precisione di una rete neurale senza perdere capacità di generalizzazione su nuovi dati, le più importanti sono:

- Regolarizzazione L2, anche detta regolarizzazione di peso, questa tecnica prevede di aggiungere un termine di penalizzazione in modo da limitare la norma L2 dei pesi della connessioni nella rete. Al giorno d'oggi la maggior parte degli algoritmi per la discesa del gradiente implementano nativamente questa regolarizzazione, attraverso il parametro di weight_decay
- Data Augmentation: questa tecnica solitamente prevede di applicare delle trasformazioni ai dati di training durante l'addestramento, in modo da aumentare la varietà di campioni di addestramento e evitare che la rete

veda sempre gli stessi dati senza variazioni. Può anche essere utilizzata per aumentare il numero di dati disponibili, applicando le trasformazioni alle immagini staticamente prima dell'addestramento.

- Dropout: questa tecnica prevede di eliminare casualmente alcuni neuroni della rete durante l'addestramento, in modo che la rete non possa fare eccessivo affidamento su di essi per la classificazione. Ciò costringe la rete a imparare caratteristiche più robuste e generali. Durante la fase di inferenza il dropout viene disabilitato.

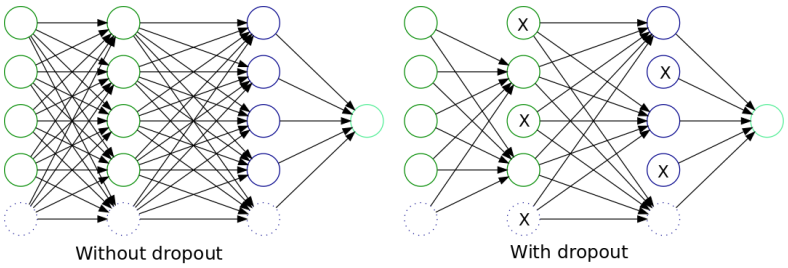


Figura 8: Esempio di utilizzo di dropout.

Fonte: `https://www.oreilly.com/library/view/machine-learning-for/9781786469878/252b7560-e262-49c4-9c8f-5b78d2eec420.xhtml`

- Batch Normalization: questa tecnica prevede di normalizzare le batch di dati durante l'addestramento, in modo da avere una media di zero e una varianza di uno. Solitamente è realizzata attraverso un layer inserito tra un layer convoluzionale e la funzione di attivazione nelle reti convoluzionali. Ciò aiuta a ridurre l'effetto del cambio di scala e velocizza la convergenza dell'addestramento.

- Early stopping: questa tecnica prevede di monitorare la performance della rete durante l'addestramento e di interromperlo quando la performance sul validation set inizia a peggiorare o raggiunge uno stallo, ripristinando eventualmente i pesi dell'ultima epoca prima del degrado. Ciò aiuta anche a risparmiare tempo nel caso in cui la rete finisca velocemente in una situazione di overfitting.

2.3 Computer Vision e Reti Convoluzionali

La computer vision è una disciplina che si occupa dello sviluppo di tecniche e algoritmi per far sì che i computer possano "vedere" e comprendere ciò che appare in immagini e video. La storia della computer vision risale agli anni '60,

tra le prime applicazioni pratiche c'era il riconoscimento dei caratteri, che divenne una tecnologia fondamentale per l'automazione dei servizi postali statunitensi.

Nel corso degli anni '70, gli scienziati della computer vision svilupparono algoritmi per il riconoscimento di oggetti in immagini, e poi per la segmentazione dell'immagine. Mentre nel corso degli anni '80, i ricercatori iniziarono a sperimentare con tecniche di "tracking", per analizzare il movimento degli oggetti in video ad esempio.

In parallelo allo sviluppo di algoritmi "classici", nuove forme di Machine learning vennero sperimentate per trovare soluzioni più semplici e efficaci per eseguire tali compiti. Non necessariamente solo con il deep learning e reti neurali, che hanno avuto un importante svolta nello scorso decennio, ma anche con altre tecniche di apprendimento automatico.

Un esempio è l'utilizzo di istogrammi di gradiente uniti a Support Vector Machines per il riconoscimento facciale, adesso soppiantato in gran parte da reti convoluzionali profonde.

Le reti neurali convoluzionali (CNN) sono state introdotte verso la fine degli anni 80, estendendo l'idea del Neocognitron con algoritmi basati sulla retropropagazione.

Il neocognitron è stato un modello di rete neurale di riconoscimento delle immagini sviluppato da Kunihiko Fukushima nel 1979, ispirato dal lavoro di Hubel e Wiesel sulle cellule semplici e complesse del sistema visivo dei mammiferi.

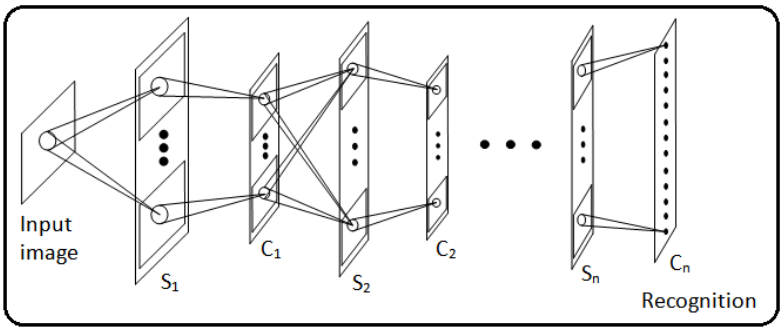


Figura 9: Struttura del Neocognitron. Strati di cellule semplici S e complesse C si alternano

Fonte: `https://www.researchgate.net/figure/The-structure-of-Neocognitron-16_fig1_323354835`

L'idea principale del neocognitron di Fukushima era semplice: catturare modelli complessi (ad esempio, un cane) utilizzando cellule complesse che raccolgono le loro informazioni da altre cellule complesse di livello inferiore o cellule semplici che rilevano modelli più semplici (ad esempio, una coda). Il neocognitron introdusse due dei layer fondamentali delle CNNs: i layer convoluzionali e i layer di pooling. Fukushima inoltre introdusse l'uso della ReLU (Rectified Linear Unit), una funzione di attivazione che è tutt'oggi una delle più utilizzate.

Un altro tassello degno di nota nella storia dei CNN furono i Time Delay Neural Network (TDNN), un tipo di rete neurale artificiale che utilizza delay lines, cioè linee di ritardo, per elaborare le informazioni. Le linee di ritardo permettono alle informazioni di essere trasmesse alle diverse parti della rete neurale con un certo ritardo, in modo tale da poter modellare schemi temporali complessi, come ad esempio riconoscere parole pronunciate lentamente o con accenti diversi.

I parametri utilizzati per i filtri delle convoluzioni erano spesso decisi manualmente basandosi sulla conoscenza che sia aveva del problema, comportando però complicazioni in reti con gran numero di parametri e non permettendo alla rete di adattarsi dinamicamente ai dati di allenamento.

Per ovviare a questo dettaglio, nel 1988 LeCun realizzò una rete convoluzionale per il riconoscimento di codici zip utilizzando la retropropagazione per imparare anche i parametri dei filtri.
L'apprendimento si rivelò essere più efficace rispetto alla scelta manuale dei parametri ed applicabile a una ampia gamma di immagini e problemi. Inoltre introdusse il max-pooling, successivamente raffinato da Yamaguchi nel 1990, che permetteva di avere una migliore generalizzazione utilizzando anche un numero minore di parametri, in quanto la dimensione veniva ridotta e solo le feature più importanti venivano propagate, anziché essere tutte uniformate dall'average pooling.

Il problema principale di questa rete fu che, con le tecnologie del tempo, l'allenamento poteva durare anche giorni anziché ore. Questo fattore scoraggiò l'utilizzo della rete, visti tempi minori di altri metodi esistenti.

LeCun presentò LeNet-5 nel 1998, una rete neurale convoluzionale che usava la retropropagazione dell'errore per riconoscere i numeri manoscritti. Era composta da due layer di convoluzione alternati a layer di max-pooling e due layer fully connected con un layer finale di output.

A causa del grande costo in risorse, tra cui i lunghi tempi necessari per il

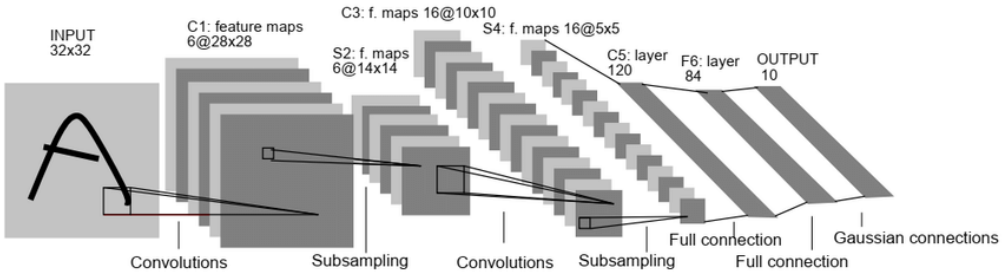


Figura 10: Architettura di LeNet.

Fonte: `LeNetPaper:https://ieeexplore.ieee.org/document/726791`

training, le reti neurali ebbero una svolta solo nel corso degli anni 2000 con l'evoluzione delle GPU e dei primi modelli di rete ottimizzate per il calcolo su di esse. Nvidia nel 2007 sviluppò CUDA per le sue GPU, permettendo di scrivere codice per eseguire in parallelo operazioni su GPU.

Nel 2011 e 2012 furono presentate DanNet e AlexNet, due CNN che grazie all'utilizzo della GPU raggiungevano velocità 60 volte superiori alle reti con CPU. AlexNet inoltre introdusse nei CNN l'utilizzo della ReLU, che si dimostrò più efficace della sigmoide e di tanh, e di layer di dropout dopo i layer FC.

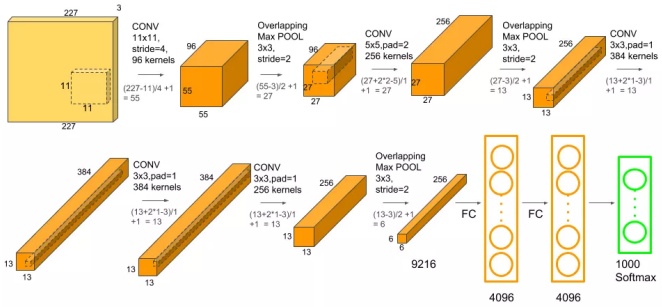


Figura 11: Architettura di AlexNet.

Fonte: `https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/`

Queste due reti erano fondamentalmente un estensione di LeNet, ma incredibilmente più profondi. L'utilizzo di due GPU in parallelo per il training

dimostrò che era possibile realizzare architetture più complesse e precise senza incorrere in tempi eccessivamente lunghi, portando ad un approfondimento sulle reti profonde e lo sviluppo GPU dedicate al Deep Learning negli anni successivi.

Questo fu un punto di svolta per l'applicazione dei CNN, che iniziarono rapidamente ad evolversi in architetture più complesse, precise e veloci, e capaci di andare oltre la semplice classificazione di immagini.

Una delle evoluzioni delle reti convoluzionali consiste nella segmentazione di immagini ed object detection, ovvero di identificare multipli soggetti all'interno di una singola immagine anziché solo quello principale. Vi sono tre tipi principali di segmentazione: semantica, di istanza e panoptica.

- Semantica: classifica ogni pixel dell'immagine, generando una maschera che divide ogni elemento presente in base alla sua classe
- Istanza: identifica ogni elemento in modo simile all'object detection, ma utilizzando una maschera anzichè una bounding box.
- Panoptica: combinazione delle due precedenti

Figura 12: Esempi di Object Detection e Segmentazione

Fonte: `https://datagen.tech/guides/image-annotation/semantic-segmentation/`

In relazione a questo progetto, mi soffermo maggiormente sui principali modelli di segmentazione semantica in circolazione e utilizzati durante il tirocinio. Questi modelli utilizzano un architettura encoder-decoder, dove la prima parte estrae le feature diminuendo la dimensione dell'input e la seconda ricostruisce l'immagine in base alle feature estratte e riporta l'immagine alle dimensioni originali.

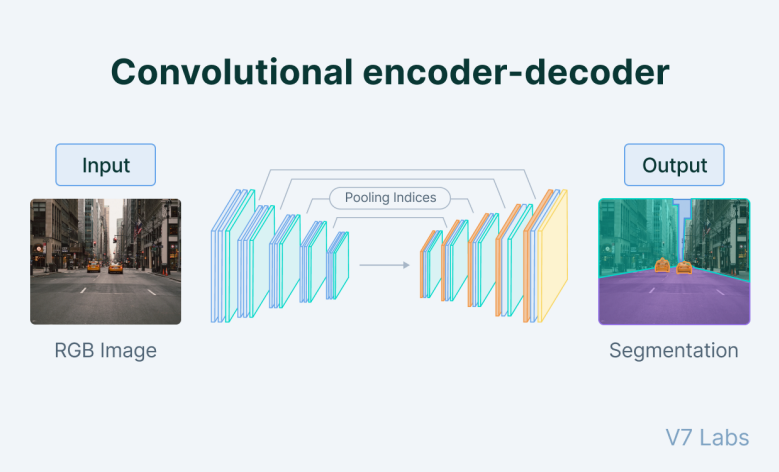


Figura 13: Struttura generale delle architetture encoder-decoder

Fonte: `https://www.v7labs.com/blog/image-segmentation-guide`

- UNet: Il nome del modello deriva dal fatto che sfrutta una architettura simmetrica a forma di "U" per combinare le informazioni di basso livello e di alto livello e migliorare la precisione della segmentazione.

  L'encoder è responsabile dell'estrazione delle features dall'immagine in input e utilizza una serie di strati di convoluzione per ridurre gradualmente la risoluzione dell'immagine e aumentare la dimensionalità delle feature map. L'encoder è strutturato come una successione di blocchi di convoluzione, ognuno dei quali è composto da uno o più strati di convoluzione seguiti da una funzione di attivazione non lineare, come la funzione ReLU.

  Il decoder, d'altra parte, utilizza una struttura "up-sampling" per aumentare gradualmente la risoluzione dell'immagine a partire dalle feature map a bassa risoluzione prodotte dall'encoder. Il decoder è strutturato come una successione di blocchi di up-sampling, ognuno dei quali è composto da uno o più strati di up-sampling seguiti da uno o più strati di convoluzione e una funzione di attivazione non lineare. Inoltre, il decoder utilizza le feature map prodotte dall'encoder prima di ogni pooling tramite una tecnica di concatenazione per fornire informazioni contestuali alle feature map del decoder. In questo modo, il decoder è in grado di utilizzare le informazioni estratte dall'immagine originale attraverso l'encoder per migliorare la segmentazione.

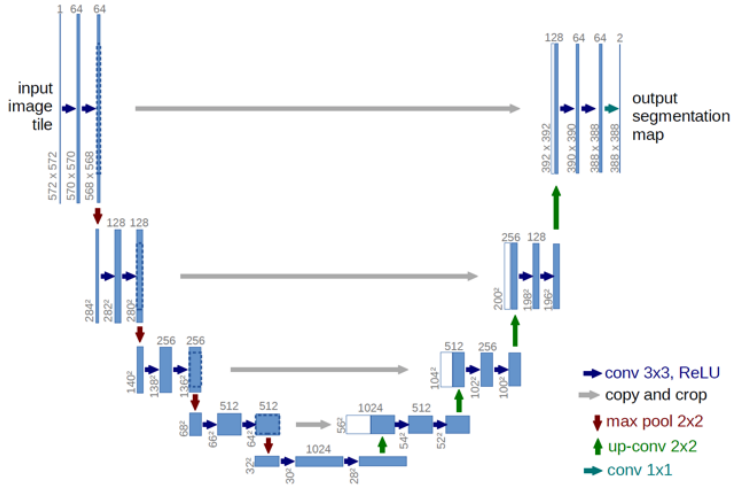


Figura 14: Struttura della UNet Classica

Fonte: `UnetPaper[9]`

L'output finale del modello UNet è una mappa di segmentazione dell'immagine in input, in rappresentazione one-hot oppure softmax, che può essere utilizzata per identificare e separare gli oggetti all'interno dell'immagine. La UNet è stata principalmente utilizzata in campo biomedico, come la segmentazione di organi, tessuti o lesioni all'interno di immagini tomografiche o di risonanza magnetica. Tuttavia, grazie alla sua grande versatilità ha raggiunto buoni risultati anche nella segmentazione di immagini satellitari ed applicazioni per la guida autonoma.

- FPN (Feature Pyramid Network): Una rete che sfrutta una piramide di funzioni per migliorare la rappresentazione delle features e l'accuratezza della segmentazione.

  Questo modello sfrutta la struttura gerarchica delle feature map generate da una rete neurale convoluzionale per creare una piramide di feature map di diverse dimensioni e risoluzioni. Questa piramide viene poi utilizzata per generare una mappa di segmentazione di alta qualità e risoluzione a partire dalle feature map a bassa risoluzione e alta dimensionalità. Similarmente alla UNet, FPN ha una struttura encoder-decoder: l'encoder "backbone" è una rete preaddestrata che si occupa di estrarre le feature dall'immagine di input per generare la piramide.

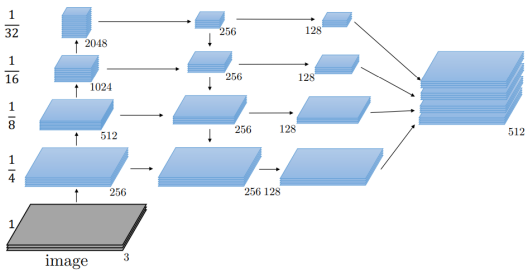


Figura 15: Struttura della Feature Pyramid Network

Fonte: `http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf`

In particolare, la piramide di features è costituita da quattro livelli, dove ogni livello rappresenta una scala diversa. I primi tre livelli sono ottenuti come output di specifici strati della rete preaddestrata, mentre il quarto livello è ottenuto da un'operazione di pooling sui feature map del terzo livello.

Il decoder, chiamato anche "head", utilizza la piramide generata dalla backbone per generare la mappa di segmentazione. Utilizzando una tecnica di up-sampling, aumenta gradualmente la risoluzione delle feature maps a partire dal quarto livello, che rappresenta la scala più bassa, e utilizza layer di concatenazione per combinare le informazioni contestuali di tutte le scale e generare una mappa di segmentazione ad alta risoluzione in output.

- PSPNet (Pyramid Scene Parsing Network): una rete che utilizza una piramide di pooling gerarchica per acquisire informazioni contestuali di diverse scale, similarmente a come il modello FPN utilizza una piramide di feature maps.

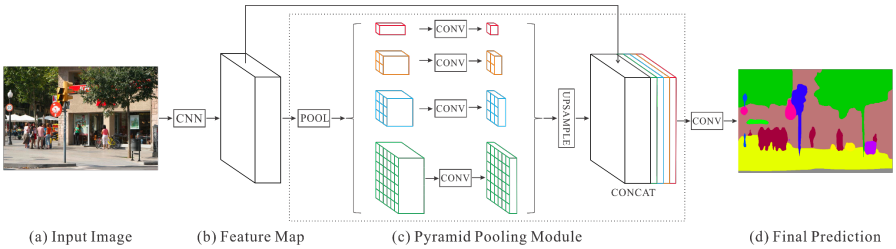


Figura 16: Struttura modello PSP

Fonte: `https://hasty.ai/docs/mp-wiki/model-architectures/pspnet`

La piramide di pooling gerarchica utilizzata dal modello PSPNet consiste in quattro livelli, ognuno dei quali acquisisce informazioni contestuali di una scala diversa. Il primo livello utilizza un pooling globale per acquisire informazioni contestuali di scala globale. I tre livelli successivi utilizzano pooling di dimensioni diverse per acquisire informazioni contestuali di scala locale.

Le feature map ottenute dai quattro livelli di pooling vengono poi concatenate e utilizzate per generare una mappa di segmentazione ad alta risoluzione. Inoltre, il modello PSPNet utilizza una tecnica di "bottleneck" per ridurre il numero di parametri e migliorare l'efficienza computazionale. In pratica, il modello comprime le feature map prima di applicare il pooling gerarchico e le espande dopo il pooling.

- DeepLabV3+ : sviluppato da Google, introduce la Convoluzione Diluita e unisce alcune caratteristiche dei modelli FPN e PSP al fine di espandere il campo visivo della rete neurale e permettere di effettuare segmentazione su immagini ad alta risoluzione in modo più efficiente.

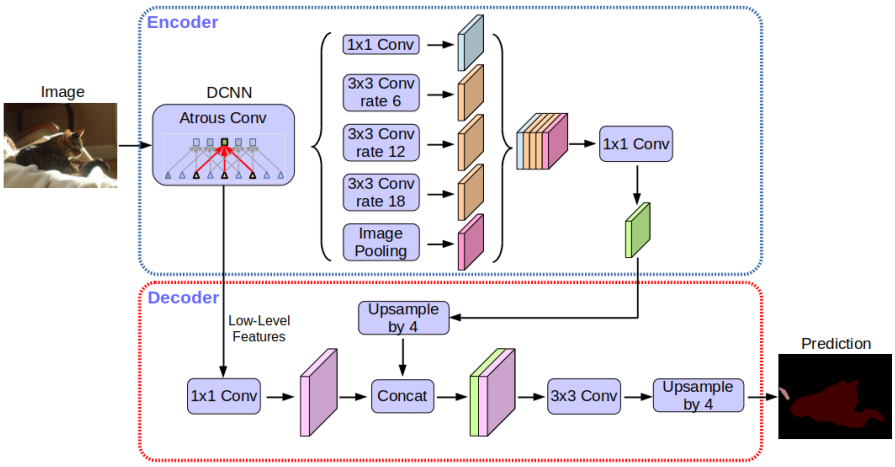


Figura 17: Struttura di DeepLab

Fonte: `https://hasty.ai/docs/mp-wiki/model-architectures/deeplabv3`

In pratica, la convoluzione diluita applica la stessa operazione di convoluzione su una finestra di input più grande rispetto alla convoluzione standard. Ciò significa che ogni valore di output viene calcolato non solo sulla finestra di input adiacente al valore di input corrente, ma anche su una finestra più grande che include spazi vuoti corrispondenti alle dilatazioni.

L'effetto di è quello di aumentare il campo receptivo dell'output senza aumentare il numero di parametri del modello. Ciò può essere utile per modellare pattern a diverse scale e ridurre il rischio di overfitting.

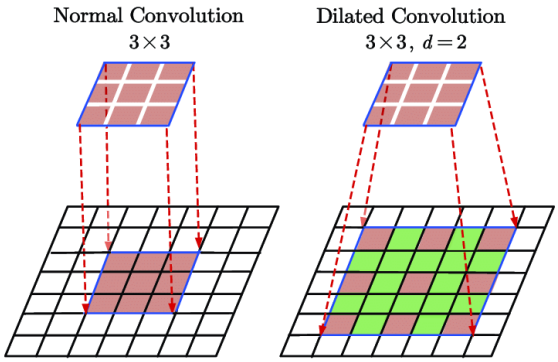


Figura 18: Convoluzione diluita

Fonte: `https://hasty.ai/docs/mp-wiki/model-architectures/deeplabv3`

3 Tecnologie utilizzate

3.1 Python

Python è un linguaggio di programmazione ad alto livello, divenuto molto popolare recentemente grazie alla sua semplicità d'uso e versatilità, soprattutto tra utenti alle prime armi e ricercatori di vari campi scientifici non informatici. Fu creato nel 1991 da Guido van Rossum e da allora si è evoluto fino a raggiungere Python3, uno dei linguaggi di programmazione più usati nei campi di data science, machine learning, e sviluppo web.

Sul lato tecnico Python è un linguaggio ad oggetti con typing forte e dinamico, il che significa che i tipi di variabile vengono determinati durante l'esecuzione anziché essere dichiarati esplicitamente nel codice e non sono permesse operazioni fra variabili con tipo differente senza type casting.

Anziché essere compilato direttamente in codice macchina, viene compilato in bytecode e poi interpretato a tempo di esecuzione. L'implementazione più usata di Python, CPython, non supporta il multithreading e per parallelizzare è necessario ricorrere al multiprocessing che comporta più overhead rispetto all'uso di thread. L'utilizzo dell'interprete per realizzare il controllo dei tipi a runtime infine rende il linguaggio generalmente più lento nelle operazioni matematiche

rispetto a linguaggi compilati; fortunatamente librerie come NumPy e SciPy sono scritte in C e C++ per ovviare a questo problema, permettendo di mitigare tali problemi.

I punti forti di Python sono infatti da ricercare nella sua leggibilità, versatilità e facilità di apprendimento. La disponibilità di librerie che si occupano della maggior parte delle operazioni in backend, può rendere la scrittura del codice molto più veloce e semplice. Nonostante esistano librerie dedicate al machine learning anche su altri linguaggi con cui avevo già confidenza, come Java, abbiamo subito concordato sull'utilizzo di Python per il progetto. I vantaggi sopra riportati mi hanno permesso di ottenere velocemente confidenza con il linguaggio e le varie librerie a disposizione hanno facilitato di molto il lavoro durante il tirocinio.

Due librerie degne di nota utilizzate sono:

- NumPy, Numerical Python, è la più popolare libreria per il calcolo numerico con array multidimensionali e matrici. Come accennato precedentemente, è scritta in C per permettere di ottenere prestazioni maggiori rispetto ad operazioni interamente in Python. NumPy fornisce utili strumenti per la manipolazione di array e matrici, al punto di essere diventato essenziale per librerie di machine learning. Per esempio, fornendo metodi per il reshape (ovvero cambiare le dimensioni di un array per trasformarlo in una matrice o viceversa), stacking e slicing (ovvero unire o dividere array/matrici)

- Pillow, successore della PIL (Python Imaging Library) che non riceve aggiornamenti dal 2011, è una libreria dedicata al fornire strumenti per l'utilizzo e manipolazione delle immagini. Fornisce un'interfaccia semplice e veloce per aprire e salvare file di varie estensioni (es. JPEG, PNG, BMP, GIF), eseguire manipolazioni e correzioni come resizing, cropping e rotazioni o conversioni ad altri formati di colore.

Inoltre per la gestione di pacchetti e dipendenze di Python sono stati utilizzati:

- PIP (Python Package Installer) è il gestore di pacchetti standard per Python. È uno strumento utilizzato per installare, aggiornare e gestire le dipendenze dei pacchetti Python. Consente agli sviluppatori Python di installare e gestire facilmente i pacchetti Python di terze parti, che possono essere utilizzati per aggiungere funzionalità e librerie ai propri progetti. Ad esempio, per installare un pacchetto Python, si può utilizzare il comando "pip install nome_pacchetto". PIP scaricherà automaticamente il pacchetto

richiesto e le sue dipendenze da PyPI (Python Package Index), il repository di pacchetti ufficiale di Python.

- Conda è stato sviluppato dalla Anaconda Inc., ed è stato creato per semplificare l'installazione e la gestione di pacchetti Python e dei loro relativi ambienti virtuali. Similarmente a PIP permette di gestire i pacchetti, ma soprattuto permette la creazione di ambienti virtuali. L'utilizzo di un ambiente virtuale serve per isolare un progetto Python dalle altre installazioni Python presenti sul sistema operativo, in questo modo è possibile lavorare su progetti che richiedono versioni diverse delle stesse dipendenze senza interferire tra loro.

3.2 TensorFlow e Keras

TensorFlow è una libreria software open source creata da Google nel 2015 per il machine learning, che fornisce fornisce API native in linguaggio Python, C/C++, Java, Go, e RUST. L'utilizzo principale di Tensorflow è la gestione del training e inferenza di Reti Neurali Profonde. Il nome deriva infatti da Tensore, ovvero gli array multidimensionali di dati utilizzati dalle reti neurali più complesse.

Keras funge da interfaccia per l'utilizzo di Tensorflow in Python, offrendo una vasta gamma di strumenti utilizzati per la costruzione e il training delle reti neurali, come modelli preallenati, strumenti di data augmentation, funzioni di loss, metriche e algoritmi per la discesa di gradiente. Si occupa anche di gestire la strategia di distribuzione del lavoro su GPU e multiprocessing, permettendo al programmatore di concentrarsi sull'architettura della rete e sulla strategia di training da utilizzare.

Le alternative più famose a Tensorflow sono Theano, non più mantenuto, e PyTorch, rilasciato da Meta nel 2016. La scelta di utilizzare Keras anziché Pytorch è stata presa di comune accordo vista la maggiore esperienza del tutor aziendale su tale framework rispetto a Pytorch e la minore complessità, dato che era la mia prima esperienza di programmazione in Python e non avevo mai usato librerie di machine learning. La comunità scientifica si sta muovendo sempre più verso Pytorch attratti dalle sue maggiori prestazioni, ma le differenze tra i due framework nel complesso non erano sufficienti per decidere di utilizzarlo al posto di Keras.

Più nello specifico, Keras fornisce delle classi Layer e Model per facilitare la costruzione del modello, ad esempio per creare una semplice rete di classificazione di immagini basta scrivere il seguente codice:

```
inputs = keras.Input(shape=(None, None, 3))

# Center-crop images to 150x150 and rescale images to [0, 1]
x=CenterCrop(height=150, width=150)(inputs)
x=Rescaling(scale=1.0 / 255)(x)

# Apply some convolution and pooling layers
x=layers.Conv2D(filters=32, kernel_size=(3,3),activation="relu")(x)
x=layers.MaxPooling2D(pool_size=(3, 3))(x)
x=layers.Conv2D(filters=32, kernel_size=(3,3),activation="relu")(x)
x=layers.MaxPooling2D(pool_size=(3, 3))(x)
x=layers.Conv2D(filters=32, kernel_size=(3,3),activation="relu")(x)

# Apply global average pooling to get flat feature vectors
x=layers.GlobalAveragePooling2D()(x)

# Add a dense classifier on top
num_classes=10
outputs=layers.Dense(num_classes, activation="softmax")(x)
model=keras.Model(inputs=inputs, outputs=outputs)
```

Poi tramite il metodo fit e le callback offerte è possibile iniziare l'allenamento della rete neurale, configurato in base alle necessità:

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-4),
    loss=keras.losses.CategoricalCrossentropy())
model.fit(train_input, train_labels, batch_size=32, epochs=10,
    callbacks = [keras.callbacks.TensorBoard(log_dir)])
```

Molto di aiuto nell'apprendimento della libreria è stata la documentazione, completa di esempi in molti campi di applicazione, sia di computer vision che di linguaggi.

3.3 Tensorboard

Tensorboard è uno strumento di visualizzazione sviluppato da Google per analizzare e monitorare il training di modelli di machine learning e deep learning, parte dell'ecosistema di TensorFlow. Tensorboard consente di visualizzare le metriche di training e di validazione del modello, come l'accuratezza e la perdita, in modo interattivo e in tempo reale. Inoltre, offre la possibilità di visualizzare il grafo computazionale del modello, che mostra come i vari layer del modello si

connettono tra loro. Questa visualizzazione può aiutare a identificare eventuali problemi di prestazione o di architettura del modello.

Tensorboard offre anche una serie di funzionalità più avanzate, come la possibilità di visualizzare la distribuzione dei pesi del modello e l'attivazione di ogni layer, che sono particolarmente utili per identificare problemi di vanishing gradient o di overfitting.

3.4 Albumentations

Figura 19: Esempi di Data Augmentation utilizzando Albumentation

Fonte: `https://albumentations.ai/`

Albumentations è una libreria dedicata al Data Augmentation per immagini, ovvero la manipolazione del dataset al fine di ottenere un set di dati maggiore o più vario. Il Data Augmentation può essere eseguito dinamicamente durante l'allenamento per diminuire il rischio di overfitting, poiché ad ogni iterazione la rete vedrà un campione leggermente diverso, oppure a priori per aumentare la dimensione del proprio dataset.

Nel nostro caso erano disponibili solamente 500 immagini annotate, spesso insufficienti anche per compiti più semplici; pertanto, per compensare il problema abbiamo sfruttato Albumentations per applicare una serie di rotazioni e inversioni orizzontali/verticali, arrivando ad un totale di 4000 immagini diverse.

3.5 Segmentation Models

Segmentation Models è una libreria open source per il deep learning sviluppata da Pavel Yakubovskiy, che fornisce implementazioni di diversi modelli di segmentazione per immagini, come U-Net, FPN e PSPNet, con la possibilità di usare una serie di modelli preallenati come backbone encoder.

La libreria fornisce inoltre delle implementazioni di metriche e loss utili per l'allenamento di modelli di segmentazione di immagini, come IoU (Intersection Over Union) e Dice Coefficient, anche se tali implementazioni non sono compatibili con tutti i tipi di dati.

In sintesi, Segmentation Models è una libreria utile per chiunque voglia creare e addestrare modelli di segmentazione di immagini in modo semplice e intuitivo o avere una veloce prototipazione, senza il bisogno di dover implementare i modelli da zero basandosi sulle loro pubblicazioni o rischiando errori nel codice dovuti a fretta o inesperienza.

3.6 Architettura di calcolo ISTI

Vista la grande necessità di risorse richiesta dalle reti neurali, il mio portatile personale avrebbe avuto grandi difficoltà sia di memoria che di prestazioni del gestire il training. Il CNR a tale scopo mi offerto l'accesso all'architettura di calcolo "AI@Edge" dell'ISTI, composta da vari nodo di calcolo, tra cui alcune gpu NVIDIA A100, e dall'NVIDIA DGX A100, un nodo denso ad alte prestazioni ottimizzato appositamente per l'allenamento di Reti Neurali Profonde ed altri. L'accesso ad entrambe è stato eseguito tramite ssh, con credenziali per il cluster e chiave privata per la DGX, utilizzando poi Slurm per lanciare i processi sul cluster e Docker per la DGX.

SSH (Secure Shell) è un protocollo di rete che permette di accedere in modo sicuro e remoto ad un computer attraverso una connessione crittografata. Il protocollo SSH permette di autenticarsi in modo sicuro utilizzando metodi di autenticazione come le password, le chiavi pubbliche e private, o i certificati digitali.

Utilizzando " ssh nome_utente@indirizzo_server -i key " viene instaurata una connessione ssh con il server remoto, utilizzando il nome e la chiave di autenticazione come credenziali. Per il trasferimento di file, è disponibile invece il protocollo SCP(Secure Copy) per copiare file da un server remoto sulla macchina locale e viceversa. Per interfacciarmi ed eseguire le operazioni in ssh ho inizialmente utilizzato terminale bash di Linux, ma una volta scoperto che

GitBash supportava anche le funzioni ssh ho utilizzato Windows (dual boot) per questioni di comodità personale.

Ho iniziato ad utilizzare la DGX e Docker dopo aver finito di preparare il dataset, mentre era in annotazione potevamo iniziare a progettare le basi della rete di segmentazione. Dato che il corso di Cloud&Green Computing ci aveva già fatto utilizzare Docker sono riuscito a creare i container e usarli per il training della rete senza troppi intoppi. La maggiore velocità della DGX ha permesso di avere un riscontro molto veloce sulla presenza di errori nel codice, impiegando minuti anziché ore per rendere evidente la presenza di anomalie durante le prime epoch di training.

3.6.1 Slurm

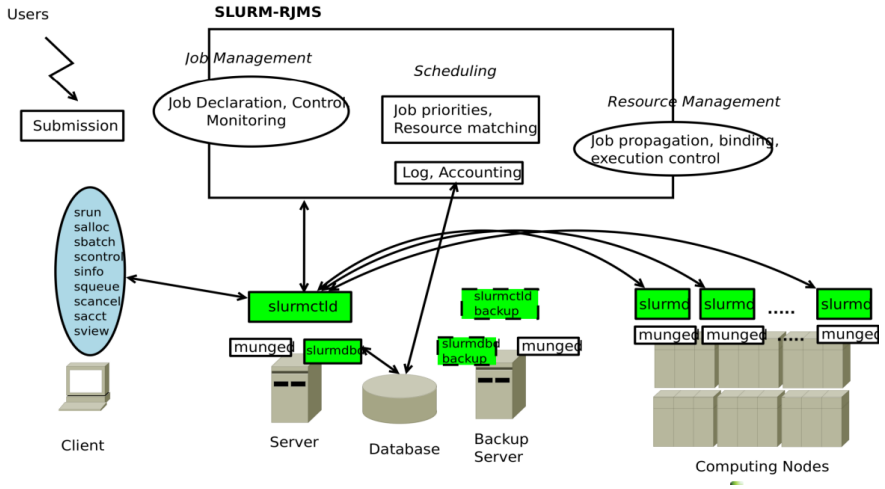


Figura 20: Schema di controllo di Slurm, dalla documentazione ufficiale

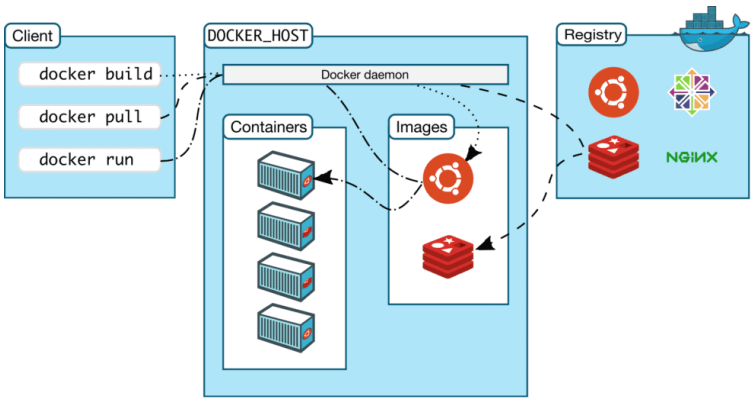
Slurm, Simple Linux Utility for Resource Management, è un sistema di gestione del lavoro open source utilizzato principalmente nei cluster di calcolo ad alta performance. È progettato per controllare e coordinare l'esecuzione dei lavori su una grande quantità di nodi di calcolo. Slurm fornisce una serie di caratteristiche, tra cui l'inoltro di lavoro, la pianificazione delle risorse, la delega delle risorse, il controllo dell'accesso e il monitoraggio delle prestazioni del sistema. Inoltre, è altamente personalizzabile e può essere facilmente configurato per lavorare con una vasta gamma di hardware, middleware e software di applicazione. Slurm viene spesso utilizzato in ambiti di ricerca scientifica, come l'analisi dei dati, la modellazione e la simulazione. Per lanciare un job su Slurm ho utilizzato ad esempio:

```
sbatch --job-name floorplan\_segmentation -n 8 --mem=30G
-t 1-00:00:00 --nodelist=gn04 -p long -o
out\_python.out -e err\_python.err segmentation.sh
```

Questo comando crea un nuovo job chiamato “floorplan_segmentation”, dedica 8 cpu, 30 gigabyte di ram e imposta un timeout di un giorno. Dato che il cluster presenta 4 nodi di calcolo, specifica l’utilizzo del quarto nodo, “long” è necessario perché i job nella coda “short” hanno un timeout massimo di 3 ore. Come ultimi parametri si ha lo stream di output, di errore e lo script del job da eseguire. Durante l’esecuzione di segmentation.sh, i log diretti a STDOUT verranno ridirezionati per la scrittura sul file out_python.out e quelli diretti a STDERR su err_python.err.

3.6.2 Docker

Docker similarmente a Slurm è ampiamente utilizzato per la distribuzione di job su cluster di calcolo, solitamente più utilizzato in cloud computing con Kubernetes.



Fonte: `https://commons.wikimedia.org/wiki/File:ArquiteturaDocker.png`

Figura 21: Schema della struttura di Docker

La caratteristica principale di Docker è il concetto di container e immagine. Un container è un’istanza di un’immagine Docker, che a sua volta è una versione ”immobile” di un’applicazione, delle sue dipendenze e dell’ambiente di esecuzione, come sistema operativo, librerie e variabili di sistema. I container Docker utilizzano una forma più leggera di virtualizzazione diversa dalle Virtual Machines chiamata ”containerizzazione”. In questo caso, il software di

containerizzazione utilizza le funzionalità di isolamento del kernel del sistema operativo host per creare un ambiente isolato e portatile in cui eseguire l'applicazione e le sue dipendenze. Questo significa che i container Docker condividono il kernel del sistema operativo ospite, ma hanno i propri processi, spazio file e rete isolati.

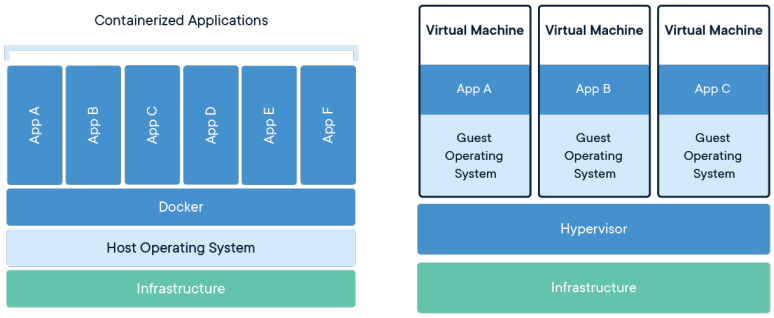


Figura 22: Differenze tra Virtual Machine e Docker

Fonte: `https://www.docker.com/resources/what-container/`

Le immagini Docker sono solitamente distribuite da siti appositi, NVIDIA in particolare distribuisce immagini docker Linux con tutte le dipendenze necessarie per compiti di DeepLearning e compatibili con architetture CUDA (Compute Unified Device Architecture). Tramite un Dockerfile l'utente può poi costruirsi un'immagine personalizzata con le librerie necessarie, ad esempio installando python e poi utilizzando pip.

Tramite pip freeze si può creare un file requirements che contiene il nome e la versione di ogni libreria utilizzata nell'ambiente python utilizzato, e con pip install -r requirements.txt è possibile installare automaticamente tutte le librerie salvate precedentemente.

Infine, tramite Docker si può creare il container basato sull'immagine e lanciarlo in esecuzione con i parametri di utilizzo delle risorse desiderato Ho utilizzato principalmente il cluster nel primo periodo, mentre stavo ancora facendo pratica con esercizi più semplici, in quanto l'accesso alla DGX mi è stato abilitato successivamente.

3.7 Fml-Wright

Fml-Wright è una rete di tipo GAN Image-to-Image (Generative Adversial Network) basata su ByCycleGan, ovvero un tipo di rete che prende in input un immagine, da cui estrarre determinate feature, per generare una nuova immagine secondo specifiche. In particolare, Fml-Wright prende in input la forma di una planimetria, di cui solo la posizione di porte e di finestre è nota, e restituisce una serie di possibili configurazioni verosimili di stanze (soggiorno, cucina, bagni, etc.) in cui la planimetria potrebbe essere divisa. Fml-Wright è un progetto sperimentale creato indipendentemente da Sebastiaan Grasdijk ispirandosi ad Expliquer, espandendo il modello per generare multipli output partendo dalla stessa immagine.

Questa rete generativa è stata presa in considerazione per la creazione di un dataset fittizio per l'allenamento della rete. Tuttavia a causa di alcuni problemi con il suo utilizzo, non è stato possibile utilizzarla direttamente. Inoltre essendo anch'essa un Proof of Concept non più aggiornato dal 2020, le immagini non sarebbero probabilmente state di alta qualità. In compenso parte del dataset utilizzato per l'allenamento di Fml-Wright, ricostruito tramite tool inclusi nella repository, è stato utilizzato (con opportune modifiche e filtri) come punto di partenza per il nostro dataset.

3.8 PyCharm

PyCharm è un ambiente di sviluppo integrato (IDE) per la programmazione in Python, sviluppato dalla società di software JetBrains. L'IDE offre una vasta gamma di funzionalità e strumenti che consentono agli sviluppatori di Python di scrivere codice più velocemente, con maggiore precisione e con meno errori. La licenza d'uso della suite di JetBrains è gratuita per gli studenti e i suoi IDE sono solitamente la mia prima scelta per ogni linguaggio di programmazione che devo utilizzare, se supportato.

PyCharm supporta una vasta gamma di progetti Python, dalle applicazioni console a quelle web, dalle librerie alle applicazioni desktop. L'IDE offre un editor di codice avanzato con funzionalità di completamento automatico del codice, evidenziazione della sintassi, correzione automatica degli errori, debugging e molto altro ancora. Grazie alla vasta gamma di plugin, può anche supportare servizi esterni come ad esempio Docker, permettendo di controllare lo stato di un container e di lanciarlo direttamente dall'IDE.

3.9 Aseprite

Menzione d'onore al software da me utilizzato per la sistemazione dei dataset durante il tirocinio. Aseprite è un software di grafica e animazione, specializzato nella creazione di pixel art e animazioni in stile retrò, sviluppato da David Capello. Aseprite offre un'interfaccia utente intuitiva, che consente agli artisti di creare immagini pixel-per-pixel con facilità. Il software supporta una vasta gamma di strumenti di disegno classici ma anche una vasta gamma di funzionalità più avanzate per la modifica delle immagini, come la scala, la rotazione, la trasformazione e l'alterazione di tonalità. Questo software è a pagamento, ma ne ero in possesso già da prima del tirocinio.

Ho scelto di usare questo software anziché uno qualsiasi degli altri editor di immagini disponibili sia perché ho ormai una certa esperienza con esso e una preferenza per la sua interfaccia, sia perché la maggior parte delle operazioni che ho dovuto eseguire sulle immagini necessitavano di una precisione a livello di pixel, campo in cui Aseprite è specializzato.

4 Lavoro Svolto

4.1 Teoria e preparazione

Durante il mese di dicembre e la prima metà di gennaio mi sono concentrato sull'ottenere le conoscenze teoriche richieste per comprendere sufficientemente le reti neurali profonde e convoluzionali da essere in grado di svolgere il progetto; Ho dovuto inoltre fare pratica con il linguaggio Python e le sue librerie, in particolar modo imparare ad usare Tensorflow e Keras.

Ho svolto la maggior parte dello studio in autonomia, con sporadiche domande al tutor aziendale per piccoli chiarimenti su alcune parti più complesse o consigli su cosa approfondire. Nello stesso periodo stavo anche preparando un esame, che ho superato solo all'appello di gennaio.

Sul lato teorico, le basi ottenute durante il corso di Introduzione all'Intelligenza Artificiale (IIA) sono stati una buona base di partenza per comprendere gran parte dei concetti più avanzati: nonostante la seconda metà del corso tratta principalmente tecniche di Machine Learning per Alberi Decisionali, SVM etc, esso copre anche i concetti alla base dell'apprendimento automatico ed alcuni accenni alle reti neurali più semplici. Mi mancava tuttavia tutta la parte su reti neurali profonde e sulle loro caratteristiche peculiari, che ho dovuto apprendere autonomamente.

Come punti di riferimento per lo studio ho utilizzato "Artificial Intelligence: A Modern Approach" suggerito dal prof. Micheli e i 4 corsi del prof. Andrew Ng, disponibili sul canale YouTube DeepLearningAI, su consiglio del tutore aziendale. Ho trovato tali argomenti molto interessanti anche se non semplici da comprendere ma, tramite l'utilizzo di varie fonti e domande al tutor ove necessario, credo di aver imparato molto.

Ho approfondito come il meccanismo di retro-propagazione funziona nelle reti neurali profonde, specialmente sul lato matematico della discesa di gradiente che era stato solo accennato durante il corso della triennale a causa della sua complessità, in particolare come gli algoritmi di ottimizzazione moderna, come RMSProp e SGD, affrontano tale problema;
Ho studiato le varie tecniche tipicamente utilizzate per contrastare l'overfitting nelle reti neurali, come Dropout, Data Augmentation e BatchNorm, e come la gestione del learning rate e batch size influenzi velocità, memoria necessaria e performance durante l'apprendimento. Ho imparato come funziona la convoluzione e i layer/blocchi convoluzionali per l'analisi di immagini, alla base della computer vision moderna, e alcune tecniche utilizzate per il riconoscimento e localizzazione di oggetti come Yolo.

Sul lato pratico, questa è stata la prima vera esperienza di utilizzo di Python, utilizzato molto brevemente durante i corsi di Cloud e Green Computing e Intelligenza Artificiale ma senza vere e proprie introduzioni al linguaggio e alle sue peculiarità. Grazie alle conoscenze ottenute precedentemente in altri linguaggi (C e Java) non è stato difficile prendere confidenza con la sintassi ed alcune librerie. Più difficile è stata la gestione di tali librerie, in quanto Python e librerie legate al campo del Machine Learning sono in continuo aggiornamento, spesso con forti incompatibilità tra dipendenze e versioni a distanza di pochi anni o addirittura mesi. Pertanto, nonostante Conda si occupi di gestire download e mantenere separate le librerie in ambienti virtuali relegate al progetto, ho dovuto combattere molti errori legati a incompatibilità tra librerie.

Come esercizi pratici ho per prima cosa consultato e provato a replicare gli esempi offerti da Keras sulla Computer Vision, per esempio riconoscimento di immagini di animali tramite dataset pubblici con UNet.
Per aumentare il grado di difficoltà ho chiesto una al tutor un esercizio di segmentazione di immagini. Per la precisione mi sono esercitato su un problema di segmentazione di istanza mentre il progetto finale si basava su segmentazione semantica.

Con segmentazione di istanza si intende che l'algoritmo prende un'intera

immagine in input e restituisce un insieme di bounding box intorno agli oggetti presenti nell'immagine, insieme alle probabilità di classe corrispondenti per ciascuna casella.
Mi sono quindi esercitato su un dataset per il riconoscimento di buche stradali, disponibile su Kaggle, contenente poco meno di 1000 immagini annotate in formato Pascal VOC.

Questo formato di annotazione basato su XML, utilizzato solitamente per object detection e instance segmentation, contiene informazioni sull'immagine e per ogni oggetto in essa contenuto ne annota la classe, i 4 vertici delle bounding box. E molto simile al formato COCO, che però invece utilizza JSON e codifica la bounding box con base e altezza e le coordinate del vertice in alto a sinistra del rettangolo.

Come modelli su cui fare Fine Tuning ho scelto, su consiglio, le reti Yolov3, di experiencor, e MaskRCNN, di matterport, entrambe allenate precedentemente sul dataset COCO.

La rete YOLO, You Only Look Once (ispirata dallo slang You Only Live Once), è una delle reti convoluzionali più utilizzate per il rilevamento di oggetti nelle immagini tramite bounding box. Come suggerisce il nome, l'architettura Yolo esegue contemporaneamente il rilevamento oggetti e la classificazione in un solo passaggio,in contrasto con altri algoritmi di rilevamento oggetti che tipicamente coinvolgono diverse fasi, come ad esempio la region proposal del Faster R-CNN.

Yolo divide l'immagine in una griglia di celle, ed ogni cella prevede un numero fisso di bounding box e le probabilità di classe per tali box, indipendentemente da quanti oggetti sono presenti nella cella. Vengono generate 3 diverse griglie a scale diverse come output, in modo da poter identificare oggetti di diverse dimensioni, le bounding box con confidenza inferiore ad una certa soglia vengono infine scartate.

La rete Mask, o Mask R-CNN, si basa sulla Faster R-CNN. Mask R-CNN è stato sviluppato per migliorare la precisione di Faster R-CNN aggiungendo una diramazione che effettua la segmentazione semantica della bounding box per adattare i bordi all'oggetto rilevato.

Faster R-CNN utilizza una rete per la proposta di regioni dell'immagine, da ogni regione vengono poi estratte le feature ed effettuata la classificazione degli oggetti tramite layer FC. In aggiunta, Mask si dirama prima degli strati di

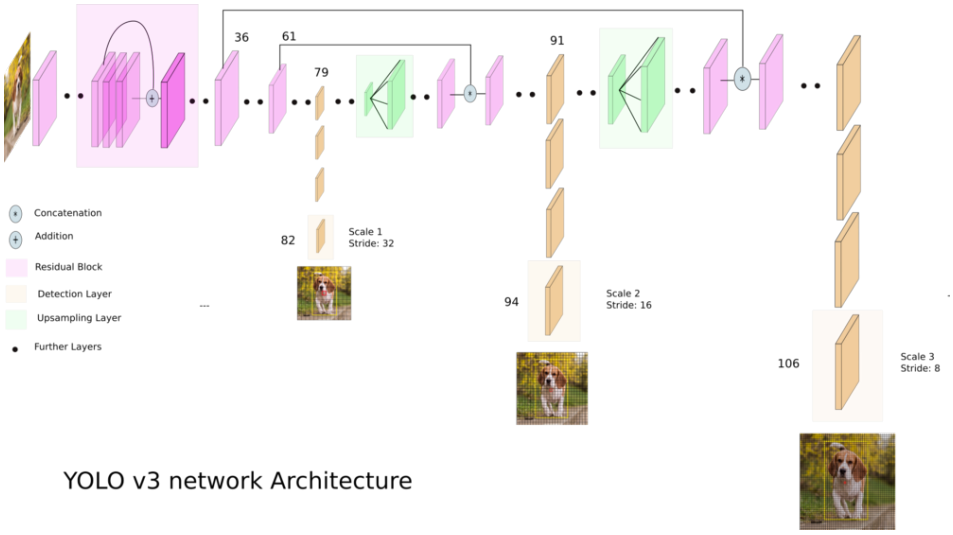


Figura 23: Architettura di Rete di Yolov3

Fonte: `https://www.researchgate.net/figure/YOLOv3-Architecture-YOLO-processes-an-overhead-image-of-a-football-game-and-finds-th fig2_367976420`

classificazione per generare una maschera binaria per ciascuna regione, che indica quali pixel dell'immagine appartengono all'oggetto rilevato e quali no. Di conseguenza YOLO si dimostra molto più veloce della rete Mask, anche se meno dettagliata, in quanto l'immagine deve essere analizzata una sola volta.

La rete Yolo ha necessitato di modifiche minori per aggiornala alla versione recente di Keras o di trovare la giusta versione di alcune librerie, raggiungendo però alla fine un ottima precisione su training e validation sets. Con la rete Mask ho avuto vari problemi legati al suo sviluppo su versioni precedenti di keras e tensorflow e alla lettura del dataset. Dopo varie ore di lavoro sul codice e ricerca di informazioni per renderlo compatibile, il training della rete si è rivelato abbastanza fallimentare in quanto molto più lento e tendente all'overfitting della rete Yolo.

Questo probabilmente perché essendo una rete molto più complessa, con molti più parametri da imparare, soffre molto della dimensione ridotta del dataset. Non escludo inoltre di aver fatto errori nel codice in quanto la mia conoscenza di Python era ancora da principiante e questa esperienza, mi ha fatto approfondire le parti più complesse, similarmente ad una terapia d'urto.

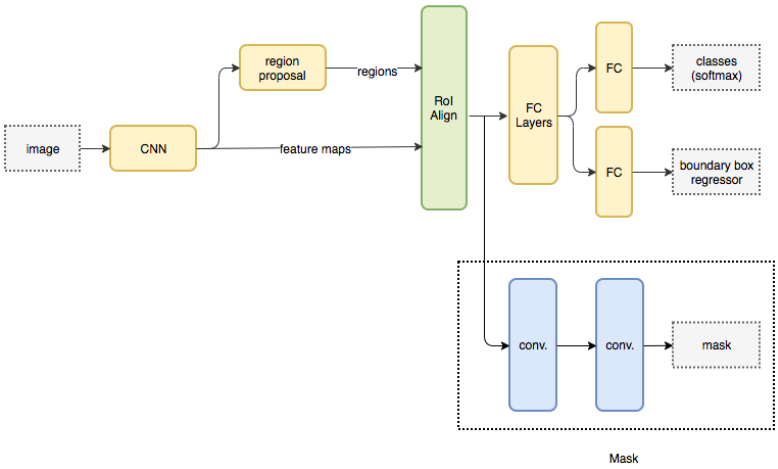


Figura 24: Schema della rete Mask

Fonte: `https://www.researchgate.net/figure/a-Mask-R-CNN-Architecture-He-et-al-2017-b-Baseline-CNN-Architecture-CNN-model_fig2_340598311`

4.2 Dataset

Una delle sfide più grandi del tirocinio è stata la costruzione del dataset per l'allenamento e testing della rete neurale per la segmentazione. Tali dataset sono solitamente sono costituiti da migliaia di immagini e necessitano di essere etichettati correttamente affinché la rete possa imparare efficacemente a risolvere il compito prefissato. Tuttavia dato che non sono stati ancora effettuati studi o esperimenti sul tema, non potevamo partire da un dataset pronto o da espandere.

Il CNR disponeva di alcune planimetrie di edifici ma esse erano numero molto ridotto ed in formato .DWG. Tale formato è utilizzato dai programmi AutoCAD professionali per la creazione di disegni 2D e 3D, contiene immagini vettoriali e vari metadati ma la conversione in immagini .png o .jpeg singole è particolarmente macchinosa. Tali planimetrie erano inoltre piene di dettagli e informazioni superflue e difficili da rimuovere, come la disposizione di mobili o dell'impianto idrico ed elettrico, che molto probabilmente avrebbero reso confusionaria sia l'analisi delle immagini da parte della rete che la creazione di annotazioni da parte dell'esperto a cui ci siamo riferiti. Era quindi necessario trovare un altro dataset di partenza da annotare.

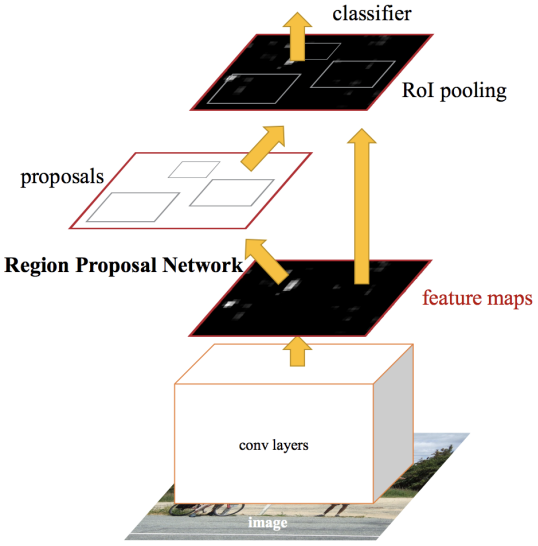


Figura 25: La rete Faster RCNN si basa sull'utilizzo di un Region Proposal Network dopo l'estrazione delle feature

Fonte: `https://hasty.ai/docs/mp-wiki/model-architectures/faster-r-cnn`

4.2.1 Acquisizione immagini

Dato che inaspettatamente era necessario reperire un numero abbastanza elevato di planimetrie in formato png con meno elementi superflui possibili, come alternativa al creare a mano da zero il dataset ho ipotizzato l'utilizzo di un algoritmo di generazione procedurale. Ero a conoscenza dell'esistenza di algoritmi capaci di creare mappe, solitamente di tipo labirintico, tramite l'utilizzo di alcune regole ed un randomizer ma non se fosse possibile trovarne uno open source ed in grado di generare una piantina il più simile possibile ad una abitazione piuttosto che a un dungeon per un gioco da tavolo. La mancanza di esperienza nel campo e la necessità di ottenere le immagini in tempi brevi inoltre giocavano a sfavore.

Su consiglio del tutor accademico ho iniziato a cercare reti generative, anziché algoritmi, in quanto le GAN sono una tecnologia recente ma incredibilmente in espansione, il loro utilizzo inoltre si è già rivelato un metodo efficace, affiancato al data augmentation, per espandere il proprio dataset senza la necessità di ottenere nuovi dati.

Per il nostro caso d'uso abbiamo trovato Fml-Wright, una rete generativa image-to-image che prende in input la forma del piano e la posizione di porte e finestre per generare una planimetria, assegnando anche un colore ad ogni stanza

per classificarla in base alla sua funzione (bagno/cucina/sala/etc.). Inizialmente l'idea era di utilizzarla direttamente per la creazione del dataset, in quanto le immagini generate sarebbero state teoricamente perfette per lo scopo, senza nessun dettaglio superfluo e con una evidente separazione tra esterno ed interni.

Il problema che è sorto nel pratico è che il repository di Fml-Wright non conteneva i pesi del modello, ma era necessario allenarla localmente. A tale scopo l'autore aveva fornito degli script e il link ad un'altra repository con dei file txt da cui estrarre i geodata, da convertire nuovamente in immagini per il training. Tali file erano il risultato di una conversione in formato vettoriale del dataset Lifull Home, che contiene planimetrie di abitazioni di Tokio, di proprietà della società Lifull. La conversione in formato vettoriale ha estratto unicamente le informazioni essenziali dalle planimetrie, ovvero la geometria delle stanze e il ruolo di ognuna, permettendo poi di ricostruirle in versione semplificata ed utilizzarle pubblicamente.

Come successo per la rete Mask, tuttavia, erano presenti delle incompatibilità tra le versioni delle librerie che ne hanno inizialmente reso problematico l'utilizzo. Infine, una volta estratto il dataset di training per la GAN, vi erano ulteriori errori nel far partire l'allenamento, a causa di altri file mancanti che l'autore non aveva incluso nella repository.

Dato che però le immagini estratte dai geodata erano nell'ordine delle decine di migliaia, alcune molte leggermente corrotte dalla conversione ma in maggioranza utilizzabili per il progetto, abbiamo deciso alla fine di utilizzare un set di 500 di queste immagini per procedere con l'annotazione.

4.2.2 Selezione e sistemazione del dataset

Le immagini ottenute dalla ricostruzione di Fml-Wright erano molto varie come dimensione e forma, ma era necessario avere un dataset uniforme per velocizzare (e semplificarne) l'annotazione delle luci.

Per prima cosa ho utilizzato un semplice script Python per dividere le immagini in base alla forma, per filtrare quelle meno quadrate e poi filtrare quelle più piccole. Alle immagini è stato applicato un padding transparente, anziché un rescale che avrebbe potuto distorcere l'immagine, per uniformarle tutte a 800x800.
Questa dimensione è stata presa dato che la maggior parte delle immagini filtrate oscillava tra 700 e 800 e 800 è un multiplo di 32, dettaglio importante per l'utilizzo di alcuni modelli UNet. Revisionando le immagini con l'esperto incaricato delle annotazioni abbiamo notato che la scala pixel:cm non era

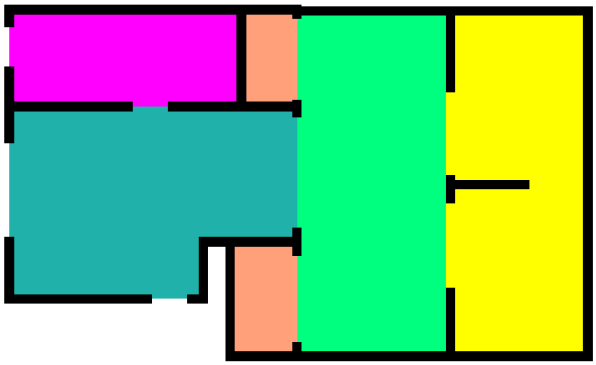


Figura 26: Una delle planimetrie estratte e poi manualmente risistemate

uniforme, oltre al fatto che non si avevano informazioni sulla scala reale delle immagini originali. Tali immagini erano infatti disponibili nel repository insieme ai geodata, ma non era possibile risalire direttamente ad esse dall'immagine generata dalla ricostruzione.

Un altro dettaglio problematico notato in queste immagini è stata la presenza di alcune stanze troppo piccole per la tipologia di luce scelta, più adatta a stanze grandi, che avrebbero potuto creare intoppi durante l'allenamento. Per sistemare questi problemi ho quindi cercato di selezionare le immagini che più si adattavano a questi standard, ma è stato comunque necessario effettuare modifiche alle immagini manualmente.

Da notare che tale ristrutturazione delle stanze ha compromesso l'originale associazione colore-tipologia di stanza presente nelle immagini originali. Questo non ha ovviamente influenzato il progetto dato che tali informazioni sono state definite superflue: adattare la disposizione delle luci basandosi sulla tipologia di stanza avrebbe reso il compito ancora più complesso per la rete, rischiando di non riuscire a portarlo a termine nei tempi stabiliti.

Per riuscire a portare tutto su una scala comune, ho deciso di prendere la dimensione dei muri come punto di riferimento. Nonostante nella realtà ci sia

molta varianza fra lo spessore dei muri, essi erano l'unico elemento affidabile di riferimento tra le immagini. Prendendo lo spessore più comune, ovvero 12-13 pixel, ho dovuto riadattare manualmente (tramite rescaling e padding) una buona percentuale di immagini affinché rispettassero spessore. In questo modo tutte le immagini avrebbero potuto avere un rapporto pixel:cm abbastanza simile da permettere all'esperto di disporre le luci correttamente.

Dato che dalle mie ricerche lo spessore dei muri esterni si aggira tra i 20 e i 30 centimetri, ho proposto di utilizzare un rapporto 1 pixel : 2 centimetri. Dato che sia l'esperto che il tutor aziendale hanno ritenuto valida questa ipotesi, abbiamo proseguito su questa strada. Ovviamente il risultato è un'approssimazione, anche perché nelle immagini non sembra esserci una distinzione di spessore fra muri interni ed esterni, ma permetteva di avere stanze di proporzioni verosimili nella maggior parte dei casi e dato che il fine del progetto era ormai la creazione di un Proof of Concept si è accettato questo compromesso.

Le stanze che seguendo questo rapporto sarebbero state troppo piccole sono state quindi eliminate od espanse, cercando di mantenerle realistiche. La sistemazione delle immagini ha indubbiamente sofferto della mia fretta di avere velocemente un dataset da far annotare, causando la presenza di alcune planimetrie meno conformi di altre all'interno del dataset. Inoltre solo più avanti ho notato che alcune avevano dimensioni rettangolari es. (800x600) a causa del metodo utilizzato per il padding manuale. Questo non ha comunque causato grandi errori nella classificazione delle immagini in quanto il resize effettuato per rimuovere lo sfondo trasparente compensava parzialmente l'errore rendendolo difficile da individuare nella maggior parte dei casi. Gli errori di questo tipo sono stati individuati abbastanza tardi proprio a causa del loro impatto minimo ed hanno richiesto multiple osservazioni sul dataset per essere notati.

4.2.3 Annotazione delle immagini e mappe di segmentazione

Su decisione congiunta del tutor aziendale e dell'esperto, abbiamo deciso di utilizzare lo stesso tipo di fonte luminosa per tutte le stanze, ignorando quindi le informazioni relative alla tipologia di stanza, ed indicarla con un cerchio colorato nella posizione ritenuta ottimale.

Il dispositivo illuminante utilizzato per le annotazioni è un pannello led della serie office della ProLight. La scelta di questo modello è basata su alcuni fattori.

- Ampia diffusione e buon rapporto consumo-luminosità. Per coprire più efficacemente stanze grandi.

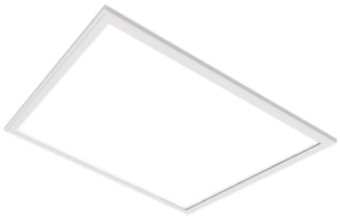


Figura 27: Dispositivo scelto per l'annotazione `https://www.pro-light.it/serie-office/`

- Dato che il caso d'uso iniziale della rete era destinato ad uffici, una luce a pannello era una buona candidata.
- I prodotti di tale marca sono compatibili con DIALux, permettendo una possibile verifica in fase di test delle disposizioni delle luci ottenute dalla rete. La ditta è inoltre di Pontedera e in buoni rapporti con l'esperto, sarebbe stato quindi possibile avere informazioni più dettagliate se necessario.

DIALux è software per la pianificazione illuminotecnica che permette di calcolare un'anteprima del livello di luminosità generata dalle luci in ogni parte della planimetria, ma per utilizzarlo è necessario avere le specifiche tecniche del dispositivo illuminante in un formato specifico offerto dal produttore. Alla fine del progetto non è stato possibile utilizzare questo strumento per ragioni di compatibilità fra i tipi di planimetria su cui lavora la rete ed il formato DWG / CAD necessario per utilizzare DIALux.

Questo tipo di annotazione ha permesso all'esperto di etichettare le immagini abbastanza velocemente e quindi rientrare nei tempi previsti dal tirocinio, cosa che sarebbe stata difficile con un tipo di annotazione più complessa. Contrariamente alle reti di rilevazione di oggetti, che utilizzano annotazioni relative a bounding boxes intorno ad ogni oggetto nell'immagine, le reti per la segmentazione semantica devono classificare ogni pixel dell'immagine e pertanto utilizzano delle cosiddette mappe di segmentazione, un'immagine grayscale speculare all'originale in cui ogni pixel indica la classe di appartenenza.

Come classi si è deciso di utilizzare:

- 0: Esterno

- 1: Muri
- 2: Interni
- 3: Luci

Le immagini annotate dall'esperto non potevano essere utilizzate direttamente come label per il training set, in quanto immagini RGBA. Era quindi necessario convertirle in mappe di segmentazione in scala di grigi, convertendo ogni pixel nell'indice della classe di appartenenza. Grazie alla scelta di utilizzare immagini "pulite" con forte distinzione degli elementi tramite colori, è stato possibile utilizzare uno script Python per eseguire tale conversione velocemente ed in modo automatico. Convertendo l'immagine in grayscale+alpha in una matrice tramite Pillow ed sostituendo ogni pixel.

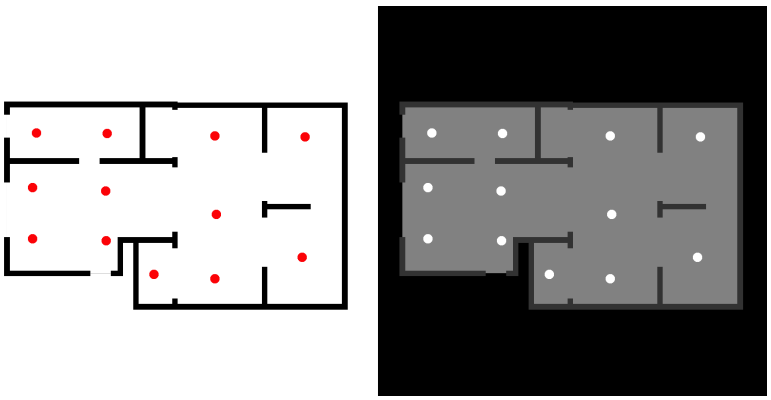


Figura 28: Annotazione ricevuta dall'esperto (sinistra) e mappa di segmentazione risultante con filtro ad alto contrasto (destra)

```
pixdata = image.load()
width, height = image.size
for y in range(height):
    for x in range(width):
        if pixdata[x, y][1] == 0:
            pixdata[x, y] = (1, 255)
        elif pixdata[x, y][0] == 0:
            pixdata[x, y] = (2, 255)
        elif pixdata[x, y][0] == 255:
            pixdata[x, y] = (3, 255)
        else:
            pixdata[x, y] = (4, 255)
```

Ad ogni pixel trasparente è stata assegnata la classe 0, ai pixel neri la classe 1, ai pixel bianchi la classe 2 ed a ogni altro colore la classe 3.

Originariamente avevo previsto l'uso del bianco come indicatore delle luci, mantenendo i colori sugli interni, ma l'esperto ha espresso la preferenza nel cambiare manualmente in bianco gli interni e utilizzare il rosso per le luci. Tuttavia, durante tale processo sono rimaste tracce del colore degli interni in alcune annotazioni, quindi ho dovuto fare ulteriori controlli sul dataset non appena me ne sono accorto per sistemarle. Alcuni di essi sono stati risolti solo durante le fasi finali, dato che erano davvero difficili da notare e influenzavano minimamente i risultati.

4.2.4 Keras Sequence

Per caricare in memoria, ed eventualmente applicare data augmentation a runtime, le immagini ho utilizzato un'estensione della classe Sequence di Keras, un alternativa ai generators più sicura in caso di multiprocessing.

```
class DataGen(Sequence):

def __init__(self, batch_size, img_size, input_img_paths,
  target_img_paths, augment):
  self.batch_size = batch_size
  self.img_size = img_size
  self.input_img_paths = input_img_paths
  self.target_img_paths = target_img_paths
  self.enable_aug = augment


def __getitem__(self, idx):
  """Returns tuple (input, target) correspond to batch #idx."""
   i = idx * self.batch_size
   batch_input_img_paths = self.input_img_paths[i: i + self.batch_size]
   batch_target_img_paths = self.target_img_paths[i: i + self.batch_size]
   x=np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
   for j, path in enumerate(batch_input_img_paths):
     img=load_img(path,target_size=self.img_size, color_mode='rgb')
     x[j] = img
     x[j] = x[j] / 255
   y=np.zeros((self.batch_size,) + self.img_size + (1,), dtype="uint8")
   for j, path in enumerate(batch_target_img_paths):
     img=load_img(path,target_size=self.img_size,color_mode="grayscale")
```

```
        y[j] = np.expand_dims(img, 2)
        # Ground truth labels are 1, 2, 3, 4. Subtract to get 0, 1, 2, 3:
        y[j] -= 1
        if self.enable_aug:
            transformed = augmentation(image=x[j], mask=y[j])
            x[j], y[j] = transformed['image'], transformed['mask']
    return x, y

augmentation = abm.Compose([
    abm.RandomRotate90(p=0.3),
    abm.HorizontalFlip(p=0.3),
    abm.VerticalFlip(p=0.3)
])
```

Durante ogni epoch, per ogni minibatch viene chiamato getitem per caricare in memoria il set di immagini da utilizzare. Sia per l'immagine di input che per la mappa di segmentazione viene inizializzato un array, lungo batch_size, di matrici a 0, di dimensione img_size + numero canali. L'immagine in input viene caricata in modalità rgb e normalizzata tra 0 e 1 dividendo per 255. L'immagine target viene caricata in modalità grayscale e viene sottratto 1 al valore di ogni pixel, le label non partono direttamente 0 dato che solitamente si riserva la label 0 e quindi la classe -1 per elementi di background da ignorare. Infine, se abilitata, vengono eseguite rotazioni/ribaltamenti casuali su ogni immagine e mappa corrispondente tramite Albumentation.

4.3 Funzioni di Loss

In ordine cronologico, sono state utilizzate come funzione di loss per il training del modello: Sparse CategoricalCrossEntropyLoss, Focal Loss, Dice Loss.

- SparseCategoricalCrossEntropy Loss (CCE) La CrossEntropy, (entropia incrociata), è una misura della differenza tra due distribuzioni di probabilità. In generale, si utilizza l'entropia incrociata per confrontare la distribuzione di probabilità stimata da un modello con la distribuzione di probabilità dei dati reali.

  La cross-entropy loss viene calcolata sommando l'entropia incrociata tra la distribuzione di probabilità stimata dal modello e la distribuzione di probabilità "corretta" dei dati di addestramento. In una classificazione a più classi, ad esempio, la distribuzione di probabilità corretta corrisponde a una distribuzione one-hot, dove solo la classe corretta ha probabilità 1 e tutte le altre classi hanno probabilità 0. L'entropia incrociata viene quindi

calcolata come la somma del logaritmo negativo delle probabilità predette per la classe corretta.

$$H(p, q) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(q_{ij})$$

dove: N è il numero di esempi nel set di dati di addestramento; C è il numero di classi; y_{ij} è un indicatore binario che indica se l'etichetta corretta per l'esempio i è la classe j ($y_{ij} = 1$) o no ($y_{ij} = 0$); q_{ij} è la probabilità predetta dal modello per l'esempio i di appartenere alla classe j.

In quanto non si tratta di una classificazione binaria ma con 4 classi, è stata utilizzata l'implementazione Categorical di questa Loss. La necessità di utilizzare la versione "Sparse" deriva invece dal fatto che l'output della rete è l'attivazione softmax su 4 classi, ovvero un array di distribuzioni di probabilità di dimensione 4, mentre la maschera di segmentazione formata da interi che corrispondono alle classi dei pixel. Tale implementazione converte automaticamente i dati nella forma corretta per il calcolo della loss.

Questa loss è stata la prima ad essere utilizzata, in quanto la CCE è la funzione di loss più popolare per problemi che coinvolgono la comparazione di immagini. Il problema principale di questa loss nel nostro caso specifico è che la distribuzione dei dati non è bilanciata, portando gli errori sul posizionamento delle luci (di cui si hanno molti meno esempi) ad avere un impatto minimo sul valore della funzione di loss e di conseguenza rendendo difficile la discesa del gradiente una volta raggiunti ottimi risultati sulla segmentazione delle altre tre classi. Di conseguenza la rete non riusciva ad imparare come piazzare le fonti luminose senza andare in overfitting sui dati di training.

- Focal Loss Per sopperire al problema della distribuzione sbilanciata dei dati, abbiamo quindi deciso, anche su consiglio del tutor, di utilizzare una loss "pesata" compatibile. Una weighted loss applica un fattore moltiplicativo (peso) all'errore derivante da ogni classe, permettendo di aumentare o ridurre l'importanza di tale errore durante l'apprendimento. Tali pesi sono decisi dallo sviluppatore come parametro della Loss in forma di array.

  Quella che più si avvicinava alla funzione già in utilizzo, la CategoricalCrossEntropy, era la Focal Loss implementata in una libreria apposita omonima: Focal Loss.

La Focal Loss ha un iperparametro gamma, che regola quanto i pesi delle classi influiscono sul valore della loss. In particolare, la Focal Loss attribuisce un peso maggiore ai campioni che sono stati classificati in modo errato o che sono stati classificati in modo corretto, ma con una bassa confidenza. Per confidenza si intende la probabilità assegnata dal classificatore che il dato di input appartenga ad una determinata classe.

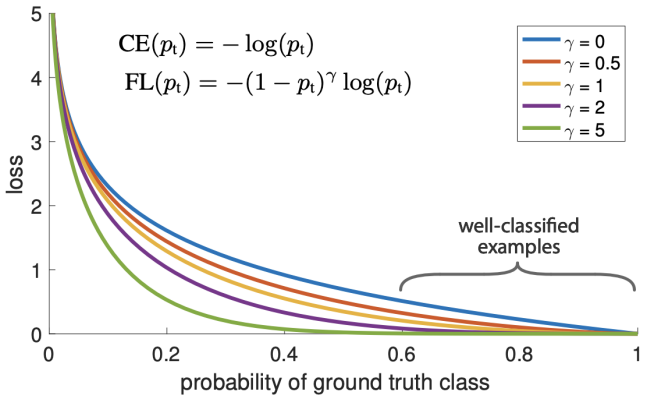


Figura 29: Comportamento della Focal Loss con vari valori di γ e pesi standard

Fonte: `FocalLossPaper[11]`

In questo modo, la Focal Loss incentiva il modello a concentrarsi sui campioni più difficili da classificare, aumentando la sua capacità di discriminazione tra le classi. Matematicamente, la Focal Loss è definita come:

$$FL(p_t) = -(1 - p_t)^{\gamma} \times \log(p_t)$$

dove p_t è la probabilità predetta per la classe corretta, e γ è l'iperparametro che controlla la concentrazione della Focal Loss sui campioni difficili. Quando γ è uguale a 0, la Focal Loss è ridotta alla CrossEntropy Loss.

Dato che lo sfondo era la classe più facile da classificare ma non volevo penalizzare troppo muri e interni, ho inizialmente scelto dei pesi [0.15, 0.35, 0,35, 0.5] per provare a far concentrare la rete maggiormente sulle luci. Successivamente ho intensificato la differenza utilizzando [0.01, 0.2, 0.4, 0.8]

- Dice Loss: Nonostante la Focal Loss era un decisivo passo avanti, piazzando luci con maggiore precisione nelle stanze più piccole, esse erano piazzate spesso in eccesso o molto raramente in base ai pesi dati alle classi.

Altro dettaglio negativo era che il cerchio per indicarle era generato in modo abbastanza impreciso, talvolta generando un blob che univa più luci insieme. Un'alternativa alla Focal Loss, altrettanto famosa per problemi di segmentazione di immagini, è la Dice Loss.

Questa funzione di errore si basa sul Coefficiente di Dice (anche noto come indice Sørensen-Dice) che valuta il rapporto fra Intersezione ed Unione di due campioni di dati. Esso è definito come:

$$Coefficiente DS = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\text{DiceLoss}(y_{pred}, y_{true}) = 1 - \frac{2\sum_i y_{pred,i} \times y_{true,i} + \epsilon}{\sum_i y_{pred,i}^2 + \sum_i y_{true,i}^2 + \epsilon}$$

Dove ϵ è un piccolo fattore di smoothing per evitare la divisione per zero.

Per questa funzione di loss non ho potuto ricorrere a librerie, in quanto non sono riuscito a trovare implementazioni compatibili con la forma dei dati a disposizione che supportasse anche i pesi per le classi. Ho quindi esteso la classe Loss di Keras, convertito i dati da interi ad array one-hot e calcolato il coefficiente di Dice, includendo i pesi delle classi nell'equazione.

```
class WeightedDiceLoss(Loss):
  def __init__(self, weights, smooth=1e-6, name='weighted_dice_loss'):
    super().__init__(name=name)
    self.weights = weights
    self.smooth = smooth

  def call(self, y_true, y_pred):
    y_true = tf.one_hot(tf.cast(tf.squeeze(y_true), tf.int32),
    depth=tf.shape(y_pred)[-1])
    y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
    intersection = K.sum(y_true * y_pred * self.weights, axis=[1, 2])
    union = K.sum((y_true + y_pred) * self.weights, axis=[1, 2])
    dice = K.mean((2. * intersection + self.smooth) /
            (union + self.smooth), axis=0)
    return 1 - dice
```

4.4 Metriche

Durante la prima metà dei test ho utilizzato la metrica basata sull'entropia incrociata, nell'implementazione fornita da Keras

"sparse_categorical_crossentropy" e SparseCategoricalAccuracy. Esattamente come la rispettiva loss, il problema di questa metrica è che calcola la precisione su tutta l'immagine senza dare particolare importanza alla distribuzione delle classi.

In quanto le luci sono solo una piccola parte dell'immagine, una precisione del 98% in realtà poteva benissimo equivalere ad una precisione del 10% od inferiore nella disposizione delle luci. Serviva quindi una metrica capace di indicare specificatamente l'andamento della classificazione della classe luci.

In teoria la metrica utile in queste situazioni è la IoU, Intersection over Union, che tramite il rapporto fra l'intersezione e l'unione della maschera di segmentazione ottenuta con quella target, permette di avere una stima della precisione in problemi di segmentazione.

$$IoU = \frac{TP}{TP + FP + FN}$$

dove TP (Veri Positivi) è il numero di pixel correttamente classificati come appartenenti alla classe di interesse, FP (Falsi Positivi) è il numero di pixel classificati erroneamente come appartenenti alla classe di interesse, e FN (Falsi Negativi) è il numero di pixel appartenenti alla classe di interesse che sono stati erroneamente classificati come non appartenenti alla classe di interesse.

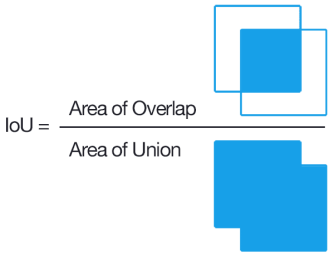


Figura 30: Rappresentazione grafica del rapporto di IoU

Fonte: `https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_visual_equation.png`

Questa metrica è disponibile sia in Keras che nella libreria Segmentation Models, con implementazioni leggermente diverse. Presi in considerazione questa metrica fin dalle prove con la Focal Loss ma a causa di un apparente incompatibilità con entrambe le implementazioni, la misi da parte temporaneamente. Dopo ulteriori ricerche in merito, ho scoperto che il problema era facilmente risolvibile utilizzando "sparse_y_pred=true" come parametro. La metrica si aspetta

normalmente che i dati di output della rete e le annotazioni siano interi ma dato che il layer di classificazione utilizza la funzione softmax, e quindi distribuzioni di probabilità fra 0 e 1, andava incontro ad errori o comunque indicava una precisione nulla.

Una volta risolto il problema di incompatibilità, ho impostato come metriche la meanIoU, per avere una stima complessiva su tutte e 4 le classi, e una IoU specificando la classe 3 come target per avere la precisione relativa al posizionamento delle luci.

4.5 Architettura del modello

4.5.1 Unet Semplice

Viste le scelte di utilizzare come annotazioni delle maschere di segmentazione, anziché bounding boxes come negli esercizi sul rilevatore di buche stradali, come architettura della rete abbiamo puntato su un encoder-decoder ampiamente utilizzata e non troppo complessa, la U-Net. La U-Net è un modello per la segmentazione semantica di immagini, inizialmente utilizzato in ambito biomedico per rilevare tumori e altre anomalie, il cui nome deriva dalla forma ad U data dalla simmetria fra i blocchi di feature extraction (encoder) e i blocchi di ricostruzione dell'immagine (decoder).

Poiché le 5 operazioni di pooling e upsampling della rete sono basate su finestre di dimensioni 2x2, è importante utilizzare immagini di dimensioni multiple di 32 per garantire che le tali operazioni possano essere eseguite senza problemi. Inoltre, l'utilizzo di immagini di dimensioni multiple di 32 consente di evitare problemi di dimensioni di padding inaspettati, come perdita di informazioni, che potrebbero compromettere la performance della rete neurale.

Il primo prototipo di rete quindi si basa sull'esempio proposto da Keras che utilizza un modello Unet per la segmentazione di foto di cani e gatti provenienti dal dataset Oxford-IIIT-Pet. Dato che lo sfondo delle immagini del dataset erano trasparenti, come prima cosa ho adattato la rete a lavorare con immagini in formato RGBA anzichè RGB per evitare che la perdita del canale alfa rendesse lo sfondo nero e quindi difficile da distinguere dai muri.

Inoltre, anziché utilizzare l'ottimizzatore SGD (Stochastic Gradient Descent) ho scelto di utilizzare Adam in quanto generalmente più efficiente nel portare la discesa del gradiente alla convergenza. Come funzione di loss ho scelto la SparseCategoricalCrossEntropy, mentre come metriche per il training inizialmente ho utilizzato "accuracy" e SparseCategoricalAccuracy, rimuovendo

però presto "accuracy" in quanto non sembrava rispecchiare l'effettiva precisione della rete, mentre invece l'altra indicava una corretta segmentazione delle parti fondamentali dell'immagine di partenza.

La disponibilità di immagini a questo punto del tirocinio era ancora abbastanza ristretta, con circa 200 immagini annotate e utilizzando il data augmentation durante il training. Ho diviso in training e validation con un rapporto 80/20 dopo uno shuffle della lista delle immagini, utilizzando un seed per mantenere consistente la divisione tra i vari tentativi.

La rete riusciva a segmentare con successo la maggior parte dell'immagine, ma non riusciva a capire di dover individuare le posizioni delle luci. La parte encoder di questa UNet semplice era probabilmente troppo semplice o "inesperta" nel riconoscimento dei pattern che portavano a scegliere i punti in cui era necessaria una luce; pertanto, abbiamo deciso di provare ad utilizzare una rete pretrained in transfer learning come encoder.

4.5.2 Unet con Resnet50

Dato che Keras fornisce alcuni modelli con pesi standard, ho provato ad utilizzare la ResNet50 preallenata sul dataset imagenet. "ResNet" è l'abbreviazione di "Residual Network", che si riferisce alla tecnica di progettazione della rete che utilizza blocchi residui per superare il problema della scomparsa del gradiente durante l'addestramento di reti neurali profonde. La ResNet50 prende il nome dai suoi 50 layer, tra convoluzionali, pooling e attivazioni. Varianti di questa architettura, come ResNet34 o ResNet100 mantengono la stessa struttura di base ma hanno un numero diverso di layer, indicato nel nome.

La caratteristica principale di una ResNet50 sono i blocchi residui, che sono costituiti da strati di convoluzione e batch normalization, seguiti da una connessione shortcut che salta i layer intermedi. Questa connessione permette al segnale di fluire più facilmente attraverso la rete, riducendo il problema della scomparsa del gradiente e migliorando l'efficienza dell'addestramento.

ResNet50 è stata introdotta da Microsoft Research nel 2015 e grazie alle sue prestazioni elevate e alla sua architettura modulare, ResNet50 è diventata una delle reti neurali convoluzionali più utilizzate per la classificazione di immagini e altre applicazioni di visione artificiale.

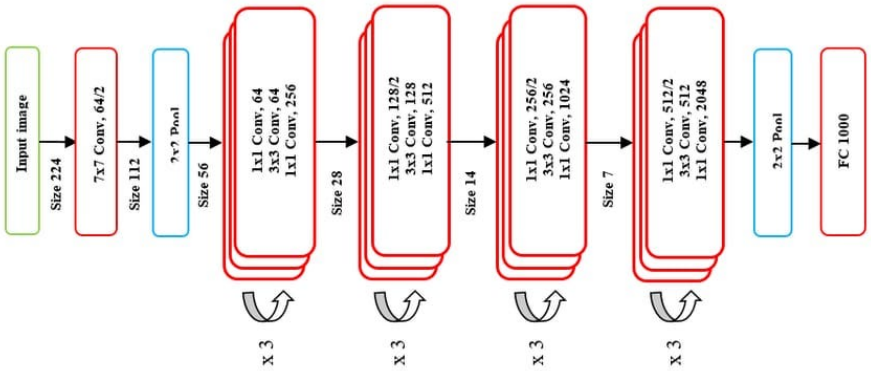


Figura 31: Architettura della ResNet50

Fonte: `https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758`

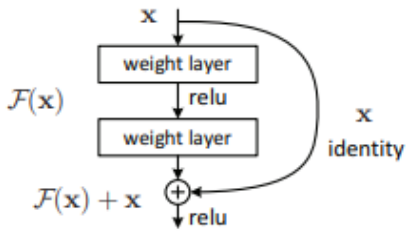


Figure 2. Residual learning: a building block.

Fonte: `ResNetPaper[12]`

Figura 32: Struttura di un residual block. L'input viene sommato alle feature estratte per mantenere intatte le informazioni di alto livello.

Per trasformare la rete in un encoder compatibile con il decoder U-Net ho rimosso gli strati finali fully connected, normalmente utilizzati per la classificazione, con il parametro "includeTop = false" e collegato l'output di cinque degli strati della ResNet al decoder della U-Net. Per scegliere i layer da cui estrarre gli output mi sono basato su altre implementazioni esistenti. Ho dovuto inoltre convertire tutte le immagini da RGBA a RGB, in quanto la rete era stata allenata su immagini a tre canali di colore. Con uno script ho quindi aggiunto uno sfondo bianco a tutte le immagini, sostituendo i pixel trasparenti. Senza questa conversione i pixel trasparenti sarebbero diventati neri, mettendo in difficoltà la rete nel distinguere sfondo e muri.

Questa versione della rete era più lenta nel training ma complessivamente portava la funzione di loss a convergere meglio, raggiungendo risultati migliori

nella segmentazione dell'immagine iniziava a disporre delle luci nelle immagini. Il problema con la ResNet50, tuttavia, era la sua maggiore tendenza verso l'overfitting rispetto all'iniziale encoder, probabilmente per la differenza di complessità (circa 22 milioni di parametri contro 2 milioni) e la presenza di alcuni problemi nel piccolo training set, a cui era spinto a adattarsi per diminuire la loss. L'aggiunta di un layer di dropout prima della convoluzione finale con softmax non migliorò significativamente la situazione ed una batch normalization sarebbe stata superflua in quanto applicata in ogni blocco del decoder.

Nelle immagini di validazione le luci erano posizionate in modo abbastanza casuale e concentrate nelle stanze più piccole mentre nelle immagini di training la precisione rimaneva di poco più alta.

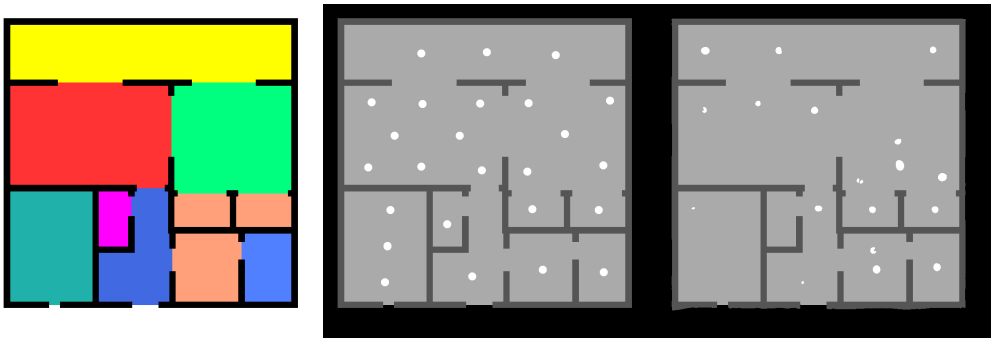


Figura 33: I risultati migliori a questo punto del tirocinio. A sinistra l'input, al centro la mappa target, a destra l'output della rete

4.5.3 Utilizzo di Segmentation Models

L'utilizzo di una backbone preallenata per la rete era comunque un significativo passo avanti; pertanto, abbiamo deciso di provare altre combinazioni. Se il problema era come sospettato nella grande complessità della ResNet50, allora una rete con un minor numero di parametri forse poteva riuscire ad ottenere prestazioni migliori prima di tendere all'overfitting.

Al fine di trovare una combinazione di encoder e decoder efficace per la segmentazione, era però necessario avere a disposizione dei pesi per la rete utilizzata come backbone e conoscerne i layer da utilizzare per i collegamenti residui del decoder. Keras offre vari modelli preallenati sul dataset imagenet, che possono essere caricati velocemente ed utilizzati per fine tuning o, come nel nostro caso, per eseguire feature extraction come parte di un modello più ampio.

Ho provato a adattare la UNet-ResNet50 ad una backbone MobileNetV2, una rete che utilizza un numero di parametri molto inferiore ma senza grandi perdite di precisione.

MobileNetV2 è un tipo di rete neurale convoluzionale sviluppata da Google nel 2018 per l'elaborazione di immagini su dispositivi mobili e incorporata in applicazioni mobili. Questo modello è stato progettato per essere altamente efficiente dal punto di vista computazionale, riducendo al minimo le risorse richieste per l'elaborazione delle immagini, pur mantenendo elevate prestazioni. L'architettura utilizza diversi blocchi di layer convoluzionali chiamati "bottleneck", che hanno l'obiettivo di ridurre il numero di parametri di ogni layer per ottenere una maggiore efficienza computazionale. Utilizza inoltre una tecnica di regolarizzazione chiamata "inverted residual" che ha l'obiettivo di ridurre il numero di operazioni di moltiplicazione, accelerando così l'elaborazione delle immagini.

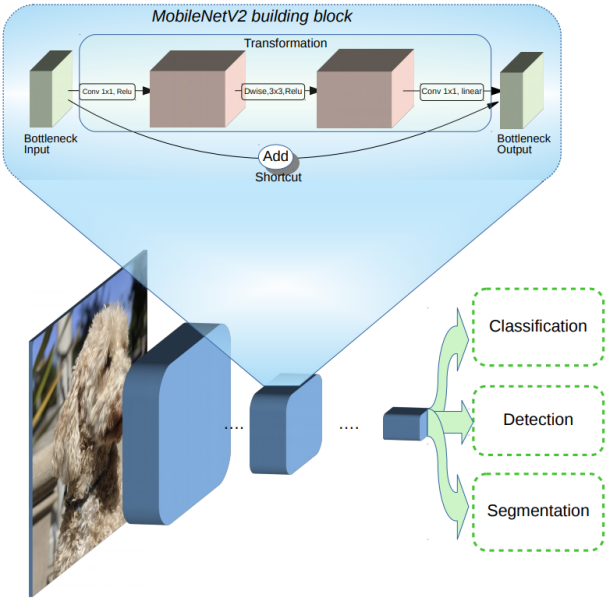


Figura 34: Blocco Convoluzionale della Mobilenet

Fonte: https://www.researchgate.net/figure/MobileNetV2-Architecture-MobileNetV2-model-consists-of-various-layers-that-can-be-se fig2_349532058

L'adattamento si è rivelato più complesso della backbone ResNet, dato che i nomi dei layer differivano tra la nuova versione di MobileNet fornita da Keras e

la versione precedente utilizzata nelle UNet trovate online come esempio. Riuscii comunque a far partire il training su una Mobile-Unet ma a causa della scelta erronea dei layer da usare come skip connection i risultati non erano migliori.

Su suggerimento del tutor aziendale, ho deciso di utilizzare direttamente la libreria SegmentationModels di qubvel per essere sicuro di non sbagliare nella costruzione del decoder e la scelta dei layer per i collegamenti residui. SegmentationModels offre una serie di backbone, preallenate sul dataset Imagenet e con allegati i layer da cui estrarre l'output per il decoder, e quattro diverse architetture decoder: Unet, LinkNet, FPN e PSP.

Ho inizialmente utilizzato la UNet con MobileNet e le versioni meno profonde delle ResNet, ovvero ResNet18 e ResNet34. Ho inoltre aggiunto la callback reduceLROnPlateau per diminuire il learning rate sistematicamente, per evitare che la discesa di gradiente diventasse instabile nei pressi di un minimo locale e facilitarne la convergenza verso il minimo migliore.

La rete continuava ad andare in overfitting relativamente presto poiché i dati erano ancora insufficienti e l'implementazione della UNet fornita dalla libreria era più complessa di quella creata manualmente, ma la MobileNet prometteva buoni risultati grazie al suo minor numero di parametri. Da notare che gli errori in validazione causati da tale overfitting non erano unicamente relativi alle luci ma anche sulla classificazione di interni ed esterni. Tale problema si è però successivamente rivelato essere legato ad alcune immagini di training che non combaciavano correttamente con le annotazioni.

4.5.4 FPN e PSP

Ho quindi provato a cambiare decoder per vedere se il problema poteva essere durante la fase di ricostruzione dell'immagine. Da qui in poi ho inoltre iniziato ad usare la Focal Loss come funzione di loss per compensare la distribuzione non bilanciata dei dati.

A questo punto del tirocinio abbiamo anche ricevuto la seconda batch di immagini annotate, arrivando a 400 delle 500 raccolte. Applicando rotazioni e ribaltamento a queste immagini, ne ho ottenute circa 2000, un numero più ragionevole per l'allenamento anche se non ancora sufficiente per un numero di parametri nell'ordine dei milioni.

Avendo sia il modello FPN che PSP forniti dalla libreria Segmentation Models, ho provato a combinare entrambi con le backbone a disposizione. Dato che

alcune non erano compatibili o avevano un numero di parametri eccessivo, mi sono concentrato su ResNet34 e MobilenetV2.

A causa dei problemi di memoria che il training iniziava a richiedere, ho deciso di diminuire la dimensione delle immagini a 512x512 per poter utilizzare una batch size di almeno 5 durante i test. Inoltre, dato che la funzione di loss iniziava a convergere abbastanza velocemente verso un punto stabile prima di iniziare a divergere lentamente verso un chiaro overfitting, ho aggiunto una callback di early stopping per non sprecare tempo di allenamento e ripristinare i risultati migliori raggiunti.

Le architetture FPN e PSP hanno un numero di parametri inferiore rispetto alla UNet, ed utilizzando una serie di feature map differenti ho ipotizzato che potesse riuscire ad avere campo visivo maggiore ed evitare di focalizzarsi sui muri per la disposizione delle luci.
Inoltre queste due architetture utilizzano dei layer chiamati Spatial Dropout che, similarmente a classici strati di Dropout, scartano casualmente parti delle feature map ottenute per diminuire l'overfitting. Visto il loro maggior impatto sulla precisione, il tasso di dropout è solitamente tra 0.1 e 0.3 anzichè tra 0.5 e 0.7 come per il Dropout classico.

Ho provato sia ad utilizzare la Focal Loss che ad introdurre la Dice Loss e la metrica di IoU, cercando di trovare anche dei pesi migliori per le classi.
Utilizzando la Focal Loss con pesi eccessivamente favorevoli per la classe delle luci si otteneva una disposizione su tutta la planimetria ma spesso eccessiva e che generava dei cluster di luci chiaramente inefficienti.
Attraverso la Dice Loss invece la rete andava a premiare particolarmente le disposizioni delle luci in cui il cerchio era disegnato correttamente ma finiva per disporne di meno.

Nel complesso, l'aumento di dati a disposizione e l'utilizzo delle metriche pesato hanno aiutato a migliorare la precisione della rete sulle stanze medio piccole ma per qualche ragione neanche questi due modelli riuscivano a disporre luci al centro delle stanze grandi.

4.5.5 Test finali sulle varie architetture

Ottenuta l'ultima serie di immagini, raggiungendo le 500, ho applicato il data augmentation per massimizzarne il numero a 4000 e diviso nuovamente il dataset per includere un test set con rapporto 7/2/1. La dimensione delle immagini non è stata ridotta oltre i 512px, ridurre ulteriormente comportava perdere elementi dell'immagine o distorsioni che andavano unicamente a peggiorare i risultati.

Durante la sistemazione di questa ultima parte dei dati ho notato e risolto alcuni piccoli errori nelle immagini che potevano essere stati la causa di alcune anomalie nei risultati finali delle architetture provate.

Ho quindi ripetuto i test su Unet, FPN e PSP utilizzando sia Focal che Dice Loss, con i pesi impostati a [0.01, 0.2, 0.4, 0.8] e mettendo come metriche MeanIoU, IoU sulla classe delle luci ed anche la SparseCategoricalAccuracy per verificare se potesse comunque dare informazioni utili a fianco dell'IoU, ma così non è stato.

Come backbone ho scelto di utilizzare solo Mobilenet e ResNet34, in quanto le altre backbone disponibili erano troppo complesse. ResNet18, ResNet50 e ResNet100 ottenevano risultati molto simili ma leggermente inferiori, quindi ho deciso di utilizzare la versione 34. Ho inoltre deciso di rimuovere il layer di Dropout aggiunto alla UNet in quanto non sembrava influenzare nè la tendenza della rete ad andare in overfitting nè i risultati finali.

In tutti i casi testati, la rete raggiungeva dopo le prime tre epoche un IoU sulle luci intorno al 10% su training e validation set. Nel giro delle successive epoche il training continuava a salire, mentre il validation trovava un picco intorno al 12% per poi iniziare ad oscillare e scendere lentamente o iniziare a danneggiare il punteggio IoU generale. L'utilizzo di learning rate, anche tramite la callback di reduce on plateau, diversi non influenzava questo andamento, se non ritardando l'eventuale divergenza chiaramente causata da overfitting. I risultati migliori li ho ottenuti partendo da learning rate pari a 1e-4.

Infatti per quanto riguarda il training set, ho osservato che senza early stopping si può raggiungere tranquillamente anche sopra il 50% di IoU sulle luci e 90% di MeanIou. In particolare la Focal Loss migliorava molto più velocemente sul training set, raggiungendo tale precisione intorno alla decima epoch, ma con risultati più scarsi in validation rispetto alla Dice.
L'early stopping quindi è andato a salvare i risultati dell'epoch 6 nella maggior parte dei casi con la Dice Loss, con meanIoU intorno al 78%.

Per curiosità ho anche provato ad utilizzare DeepLabv3+ come decoder, prendendo l'implementazione fornita da Keras con una ResNet50 come backbone, ma anche in questo caso nessun cambiamento significativo nei risultati.

5 Conclusioni

5.1 Analisi dei risultati sperimentali

Riporto alcuni dei risultati finali sulle architetture migliori, ovvero utilizzando UNet e FPN come decoder e ResNet34 e MobileNetV2 come encoder, allenate sulle 4000 immagini del dataset, di dimensione 512x512 pixel, divise in train/validation/test sets in rapporto 7/2/1. L'allenamento è stato eseguito con learning rate pari a 1e-4 e utilizzando sia Focal Loss (gamma = 2) che Dice Loss. Come metrica di riferimento per Early Stopping e riduzione automatica del learning rate è stata usata la MeanIoU sul validation set.
Metriche usate: meanIou, IoU su classe luci
Pesi classi utilizzati con Dice Loss: [0.01; 0.1; 0.3; 1.0]
Pesi classi utilizzati con Focal Loss: [0,01; 0.1; 0.4; 0.75]

Encoder	Decoder	Loss	Tempo	IoU	IoU Luci	Val IoU	IoU Luci
ResNet34	Unet	Dice	51s	0.7890	0.1816	0.7755	0.1291
ResNet34	FPN	Dice	81s	0.7900	0.1851	0.7758	0.1306
MobileNet	UNet	Dice	50s	0.7858	0.1695	0.7726	0.1208
MobileNet	FPN	Dice	82s	0.7884	0.1785	0.7745	0.1260
ResNet34	UNet	Focal	48s	0.7914	0.1892	0.7721	0.1148
ResNet34	FPN	Focal	80s	0.7981	0.2162	0.7722	0.1155
MobileNet	UNet	Focal	49s	0.7878	0.1743	0.7646	0.0833
MobileNet	FPN	Focal	80s	0.8022	0.2319	0.7696	0.1050

Tabella 1: Comparazione risultati. Per "Tempo" si intende la durata media di ogni epoch di allenamento.

Le architetture provate non mostrano evidenti differenze nei risultati, raggiungendo in tutti i casi valori di IoU abbastanza simili sulle immagini di validazione più o meno nello stesso numero di epoche (6-7, Focal Loss + FPN ne ha richieste 9) e risultati molto simili, con leggere peculiarità, sulle immagini di test.

I modelli allenati sulla Focal Loss (sotto) hanno migliori risultati sul training set, leggermente minori sul validation, ma più vari sul test set. Riesce a coprire meglio le aree ampie, ma rischia spesso di eccedere con le luci in spazi ristretti. I modelli allenati sulla Dice Loss (sopra) hanno un apprendimento più lento ma anche più stabile, con risultati che non variano particolarmente tra le

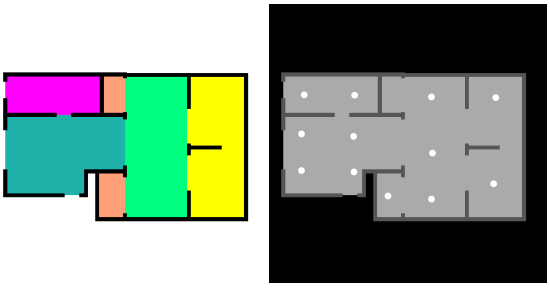
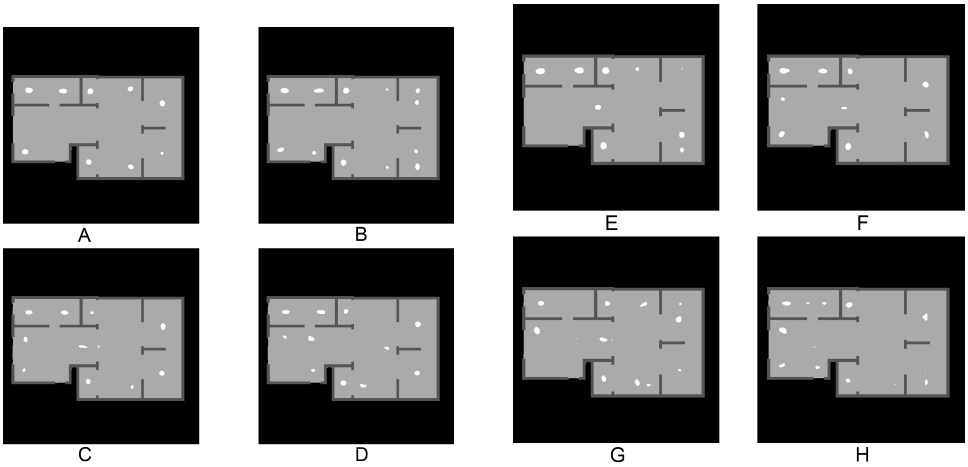


Figura 35: Planimetria in input e segmentazione di esempio



(a) UNet-ResNet34 con Dice (A) e Focal (C); FPN-ResNet34 con Dice (B) e Focal (D)

(b) UNet-MobileNet con Dice (E) e Focal (G); FPN-MobileNet con Dice (F) e Focal (H)

architetture. La ResNet-FPN raggiunge il risultato migliore sul set di validazione, ma ciò non si riflette sui risultati di test.

Analizzare un singolo campione non è ovviamente sufficiente per mostrare come ogni modello preso in considerazione si comporta, in quanto ognuno riesce a segmentare meglio certi tipi di immagini. In tutte le combinazioni prese in analisi tuttavia, l'efficienza della disposizione delle luci è insufficiente nella maggior parte delle immagini, con piccole eccezioni in alcune planimetrie ricche di muri e stanze piccole.

Una costante in tutte le predizioni fatte dalla rete è infatti la sua preferenza nel disporre le luci solo in prossimità dei muri, lasciando scoperte le stanze più grandi soprattutto al centro. Le cause di questo bias potrebbero essere legate al campo visivo della rete oppure dall'eccessiva presenza di stanze piccole e corridoi

nel dataset, facili da riconoscere e altamente legati ai muri circostanti.

Se il peso assegnato alla classe luci è molto sbilanciato, superiore a quello utilizzato in queste prove e specialmente evidente con la Focal Loss, si nota che in tali spazi vengono piazzate un numero eccessivo di luci a dimostrazione del fatto che non riesce a ricondurre quell'area ai muri più in lontananza e alle luci già disposte. La casualità delle luci piazzate in tali spazi è probabilmente legata anche all'eccessiva ed eterogenea, in mia opinione, quantità di luci che sono state disposte dall'esperto nella maggior parte delle stanze più grandi. Mentre le stanze più piccole hanno circa una o due luci, in posizione centrale, e le stanze medie circa 4, disposte in prossimità di muri, le stanze di dimensione molto maggiore hanno un numero molto più alto ed una distribuzione più concentrata e varia.

Pertanto è possibile che la rete non riesca ad imparare come trattare tali aree in quanto c'è troppa varianza nei dati di training ed è costretto ad impararne a memoria le disposizioni per diminuire la loss, portando risultati pessimi su nuovi dati, mentre le spazi più ristretti sono più consistenti e permettono alla rete di imparare a generalizzarli.

Un altro possibile problema evidenziato dal tutor è che la parte encoder era stata preallenata per l'estrazione di feature sulle immagini di ImageNet per la classificazione, mentre la parte del decoder partiva da un inizializzazione casuale dei pesi delle connessioni. L'utilizzo della backbone ha sicuramente velocizzato l'apprendimento della rete durante le prime epoche ma probabilmente ha anche portato a tendere maggiormente verso l'overfitting sul piccolo dataset a disposizione in quanto il decoder non era abbastanza esperto da generalizzare i nostri dati.

Un problema con la metrica IoU che ho notato verso la fine, ripetendo i testi con i vecchi modelli, è che non è sempre affidabile per valutare la qualità del posizionamento delle luci. In quanto la metrica basa la precisione sul rapporto fra l'area della segmentazione ottenuta e quella dell'annotazione fornita, il calo di precisione risultante da uno sbaglio nel replicare il cerchio che indica una luce sarà molto simile a quello del non aver piazzato una luce in tale posizione o ad averla piazzata in un punto vicino, ma che magari era similarmente efficiente. Tale problema è abbastanza evidente nell'immagine data in esempio, in cui il posizionamento ottenuto è in alcuni punti discutibilmente migliore dell'annotazione offerta, anche se insufficiente nel complesso.

Basarsi su tale metrica porta a valutare i risultati solo rispetto alla soluzione fornita e non all'effettiva correttezza del posizionamento, problema è accentuato

dal fatto che le annotazioni utilizzate non forniscono una verità assoluta, ottenuta tramite misurazioni esatte delle stanze, ma si basano sul bias del nostro collaboratore e delle approssimazioni sulle distanze che sono state necessarie in mancanza di un dataset completo di tutte le informazioni utili.

5.2 Lavori Futuri

Vi sono vari miglioramenti che potrebbero essere proposti, fondamentalmente sul dataset.
Un importante miglioramento che studi successivi potrebbero apportare è quindi l'utilizzo di un maggior numero di dati, completi di precise dimensioni delle stanze, ed far eseguire l'annotazione a gruppi di esperti differenti, chiedendo una dimostrazione che quella fornita sia la disposizione ottimale. In tal modo si sarebbe certi che il bias di un singolo non porti ad errori, poi ereditati durante il training dalla rete neurale.

L'annotazione delle luci potrebbe essere inoltre eseguita unicamente tramite il centro del cerchio, possibilmente permettendo una certa tolleranza durante il calcolo dell'errore, per poi disegnare il cerchio completo in fase di visualizzazione della maschera. Questo permetterebbe alla rete di non concentrarsi sulla precisione nella segmentazione del cerchio ma più sulla sua disposizione e renderebbero la lettura dei risultati molto più chiara.

Un altro importante punto che è stato tralasciato in questo progetto, per i vari motivi esposti più volte, ma sui cui è bene espandere è l'utilizzo di più dispositivi di illuminazione, con varia potenza e raggio, per permettere di ottimizzare realmente le stanze più piccole o gli spazi per cui una luce potente sarebbe superflua. A ciò potrebbe essere unita una divisione delle stanze per ruolo, portando diversi pattern di disposizione ad essere incoraggiati o scoraggiati in base ad esso. Tali dettagli sono molto importanti affinché il risultato della rete non sia solo efficiente dal punto di vista energetico ma anche piacevole per l'esperienza delle persone che soggiorneranno in tali stanze.

Per quanto riguarda la funzione di Loss, alternative che potevano essere sperimentate erano le funzioni combinate, come ad esempio la Combo Loss che unisce CrossEntropy e DiceLoss, che utilizzano varie metriche per determinare l'errore, riuscendo a dare diverso peso a diversi tipi di errore, anziché affidarsi ad un unica metrica. Sperimentazione aggiuntiva e più professionale sui pesi delle classi sicuramente potrebbe portare a risultati migliori delle mie prove empiriche.

Una tipologia di metrica alternativa che sarebbe potuta essere utile è la Area under Curve. Questa metrica è similare all'IoU ma con maggior enfasi sulla capacità di classificazione del modello. La curva ROC (Receiver Operating Characteristics) rappresenta il tasso di classificazioni Veri Positivi (anche chiamato Sensitivity) sull'asse y e il tasso di Falsi Positivi (Specificity) sull'asse x, entrambi nell'intervallo [0,1]. Tale metrica ha valore 1 quando il classificatore riesce a non commettere nessun errore, ovvero quando l'area sotto la curva è massima.

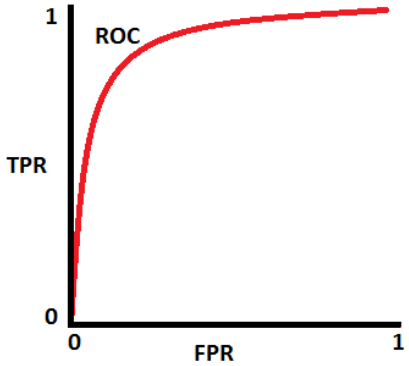


Figura 37: Grafico della ROC Curve

Fonte: `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`

AuC è solitamente usata in contesti di classificazione binaria, ma esistono due varianti per la classificazione a più classi che comparano una classe specifica verso tutte le altre oppure che calcolano una curva AuC in relazione ad ogni coppia di classi del classificatore. Nel nostro caso, AuC sarebbe stata utilizzabile mettendo a confronto la classe luci con tutte le altre classi.

A livello di architettura, l'utilizzo di un encoder più complesso sembra essere una buona idea a patto di migliorare il dataset utilizzato. Gli errori presenti nel dataset inizialmente, per quanto piccoli, mi hanno portato a preferire la MobileNet che riusciva ad ignorarli ma sul dataset completo e revisionato la ResNet34 ha avuto risultati migliori. Probabilmente a causa della dimensione e qualità dei dati prove con ResNet50 e 101 non hanno portato miglioramenti, ma migliori risultati potrebbero essere ottenuti con altre backbone famose, come una delle varianti di EfficientNet.
Una soluzione al problema dell'inizializzazione del decoder sarebbe potuto essere allenare prima l'intera architettura su un dataset di segmentazione molto ampio,

in modo da rendere più robusta la parte decoder, per poi fare fine-tuning con il nostro dataset più contenuto. Questo avrebbe richiesto molto più tempo e risorse, ma potrebbe essere una strategia da valutare per futuri esperimenti.

6 Riferimenti

Riferimenti bibliografici

[1] Repository codice relativo al tirocinio:
`https://github.com/Alexthw46/FloorPlanLightSegmentation`

[2] Russell, S., Norvig, P., & Ciceroni, R. (2019). *Intelligenza artificiale. Un approccio moderno*. Pearson. ISBN: 9788865187455. Google Scholar link:
`https://www.google.it/books/edition/Intelligenza_artificiale_Un_approccio_mo/cNyndn1eI64C`

[3] Wikipedia. *Intelligenza artificiale*.
`https://it.wikipedia.org/wiki/Intelligenza_artificiale`

Computer vision. da `https://en.wikipedia.org/wiki/Computer_vision`

[4] Amigoni, F., Schiaffonati, V., & Somalvico, M. (2008). *Intelligenza Artificiale (Enciclopedia della Scienza e della Tecnica)*. da
`https://www.treccani.it/enciclopedia/intelligenza-artificiale_%28Enciclopedia-della-Scienza-e-della-Tecnica%29`

[5] Mallick, S. (2016). *Histogram of Oriented Gradients (HOG)*. da
`https://learnopencv.com/histogram-of-oriented-gradients/`

[6] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). *AlexNet : ImageNet classification with deep convolutional neural networks*. Communications of the ACM, 60(6), 84-90. DOI: `https://dl.acm.org/doi/abs/10.1145/3065386`

[7] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). *Mask R-CNN*. ArXiv preprint arXiv:1703.06870v3. da `https://arxiv.org/abs/1703.06870v3`

[8] Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. ArXiv preprint arXiv:1804.02767. da `https://arxiv.org/abs/1804.02767`

[9] Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. ArXiv preprint arXiv:1505.04597. da `https://arxiv.org/abs/1505.04597`

[10] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2018). *MobileNets: Efficient Convolutional Neural Networks for Mobile*

Vision Applications. ArXiv preprint arXiv:1801.04381. da https://arxiv.org/abs/1801.04381

[11] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2018). *Focal Loss for Dense Object Detection*. ArXiv preprint arXiv:1708.02002. da https://arxiv.org/abs/1708.02002

[12] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. ArXiv preprint arXiv:1512.03385. da https://arxiv.org/abs/1512.03385

[13] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). *Feature Pyramid Networks for Object Detection*. ArXiv preprint arXiv:1612.03144. da https://arxiv.org/abs/1612.03144

[14] Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2016). *Pyramid Scene Parsing Network*. ArXiv preprint arXiv:1612.01105. da https://arxiv.org/abs/1612.01105

[15] Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2016). *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. ArXiv preprint arXiv:1606.00915. da https://arxiv.org/abs/1606.00915

[16] Jakhar, K. (2019) *Dice and IoU*. da https://karan-jakhar.medium.com/100-days-of-code-day-7-84e4918cb72c

[17] Bhande, P. (2018) *Understanding AUC-ROC Curve*. Immagini da https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[18] Francois Chollet, et al. *Keras*. https://github.com/keras-team/keras. https://keras.io/

[19] Lawrence Livermore National Laboratory. *Slurm Workload Manager*. https://slurm.schedmd.com/

[20] Docker Community. *Docker Docs* https://docs.docker.com/

[21] Hasty *Documentazione aggiuntiva su varie architetture ed immagini*. da https://hasty.ai/docs/mp-wiki/model-architectures/

[22] Corsi Youtube di Andrew Ng: https://www.youtube.com/@Deeplearningai

[23] Grasdijk, S. (2020) *Fml-Wright: Floorplan generation at various stages using GANs*. https://github.com/SebGr/fml-wright

[24] Chaillou, S (2019) *Expliquer* https://stanislaschaillou.com/expliquer/

[25] Iakubovskii, P et al. *Segmentation Models Library* `https://github.com/qubvel/segmentation_models`

[26] Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A.A. (2020). *Albumentations: Fast and Flexible Image Augmentations*. da `https://www.mdpi.com/2078-2489/11/2/125`. DOI: 10.3390/info11020125

[27] Liu, C., Wu, J., Kohli, P., & Furukawa, Y. (2017). *Raster-to-Vector: Revisiting Floorplan Transformation*. GitHub Repository + Paper. `https://github.com/art-programmer/FloorplanTransformation`. `https://art-programmer.github.io/floorplan-transformation/paper.pdf`

[28] Chitholian. (2020). *Annotated Potholes Dataset*. da `https://www.kaggle.com/datasets/chitholian/annotated-potholes-dataset`

[29] AutoDesk *AutoCAD* `https://www.autodesk.it/products/autocad/overview`

[30] DIAL GmbH *DIALux* `https://www.dialux.com/it-IT/dialux`