Università degli Studi di Padova

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Funzioni obiettivo nell'allenamento delle reti neurali

Relatore:
Prof. Nanni Loris

Laureando:
Pietrobon Andrea
1143259

Anno Accademico 2021/2022

Data di Laurea: 22 Settembre 2022

For Mum and Dad. Thanks for Everything.

Abstract

Nell'allenamento di reti neurali profonde è fondamentale lo sviluppo e la progettazione di funzioni obiettivo. Più nello specifico, nel campo della segmentazione semantica, sono indicate diverse tipologie di metriche di valutazione. La differenza tra metriche di valutazione e funzioni obbiettivo però potrebbe condurre alla degradazione delle prestazioni della rete neurale. Il fulcro di questa tesi è di applicare un metodo di AutoML, AutoLoss-Zero, per la rilevazione di polipi del colon. Le fasi di training e di test sono state eseguite sul dataset Kvasir-SEG. Il metodo AutoLoss-Zero per la ricerca e creazione di funzioni obiettivo si basa su uno spazio di ricerca basilare, esclusivamente composto da operatori matematici. L'obbiettivo del presente metodo è quello di ridurre e possibilmente eliminare il gap tra metriche di valutazione e funzioni obbiettivo, senza fare affidamento sull'essere umano. Nella tesi lo abbiamo implementato in Matlab e successivamente lo abbiamo applicato ad un progetto che ha come scopo la rilevazione di polipi del colon.

Indice

Introduzione

L'AutoML è una disciplina in continuo sviluppo e sempre più rilevante come branca dell'intelligenza artificiale. Negli ultimi anni abbiamo assistito a entusiasmanti progressi in questo settore. Il suo obbiettivo principale è quello di eseguire automaticamente molti dei task lunghi e ripetitivi che che si presentano nello sviluppo di un modello. Il fine di questa tesi è quello di portare alla luce i grandi vantaggi di un metodo di Automated Machine Learning, ovvero AutoLoss-Zero. Esso ha come obbiettivo quello di automatizzare la progettazione di funzioni di perdita per compiti generici. Nell'algoritmo di ricerca, per migliorarne l'efficienza, viene proposto l'utilizzo del Loss-Rejection Protocol e Gradient Equivalence Strategy che filtra i candidati alla funzione di perdita poco promettenti e ne evitare valutazioni duplicate.
La tesi è strutturata in 4 capitoli così suddivisi.
Nel primo capitolo viene data un'introduzione alle reti neurali e alla segmentazione semantica.
Il secondo capitolo invece, verterà sulle funzioni obbiettivo (loss functions) e ne descriverá alcune tipologie, come la Mean Squared Error e la Binary Cross-Entropy.
Nel terzo capito ci addentreremo nel cuore di questa tesi, andando a spiegare il funzionamento di AutoLoss-Zero e la sua applicazione per compiti generici. Spiegando inoltre le varie fasi del suo Algoritmo di ricerca.
L'ultimo capitolo tratterà la fase sperimentale di questa tesi, i procedimenti e i test svolti per confrontare i risultati della rete neurale addestrata con una loss function ricavata da AutoLoss-Zero, con quelle di una rete neurale addestrata con altre loss function più comuni e già note.
Concluderemo commentando i risultati ottenuti e faremo delle brevi considerazioni su AutoLoss-Zero.

Capitolo 1

Reti Neurali Artificiali

In informatica le Artificial Neural Network (ANN) o più semplicemente reti neurali, prendono ispirazione dalle reti neurali biologiche del cervello umano e dal modo in cui esse processano le informazioni. Nelle reti neurali artificiali i neuroni mirano ad apprendere autonomamente modificando i propri pesi. Mappando i valori di input ai rispettivi valori di output.

1.1 Neuroni

La componente fondamentale delle reti neurali artificiali e o biologiche è il neurone. Un neurone artificiale è formato da N ingressi distinti che nei neuroni biologici si possono identificare come dendriti, ossia fibre minori che attraverso le sinapsi raccolgono segnali di input dai neuroni vicini e li propagano verso il soma. A suo modo invece la componente di output è assimilabile all'assone, una fibra presente nei neuroni umani, che si allontana dal soma per portare l'output verso i neuroni vicini.

In aggiunta nei neuroni artificiali sono presenti i pesi per ogni singolo ingresso k dell'insieme di N input e un peso aggiuntivo denominato bias.

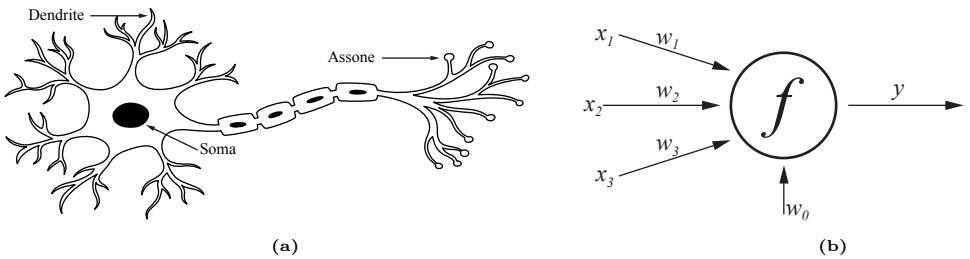

Figura 1.1: Da sinistra la rappresentazione di un neurone biologico e la sua controparte artificiale.

Il lavoro del neurone è quello di applicare una funzione, chiamata funzione di attivazione, alla somma pesata dei vari N input al fine di produrre il valore di output.

L'idea alla base è che il valore di output venga appreso autonomamente e sia in grado di gestire, in base al valore del proprio modulo e segno, l'influenza di ogni neurone sui successivi.

$$net_i = (\sum_{j=1}^{N} w_{ij} \cdot in_j) + w_{i0} \tag{1.1}$$

$$out_i = f(net_i) \tag{1.2}$$

Nei neuroni biologici $f()$ è una funzione temporizzata, quindi quando il valore di net_i supera un predeterminato valore di soglia il neurone manda un impulso per poi tornare nella condizione di riposo.

Le reti neurali più frequentemente utilizzate operano con livelli continui e $f()$ è una funzione con le seguenti proprietà:

- E' una funzione non lineare
- E' continua e differenziabile (per la retropropagazione dell'errore)

Tra le funzioni di attivazione più comuni troviamo la sigmoide, tanh, ReLU e Leaky ReLU. Ma le più utilizzate utilizzate sono:

- Standard Logic Function

$$f(net) = \frac{1}{1 + e^{-net}} \tag{1.3}$$

- Hyperbolic Tangent

$$f(net) = \frac{2a}{1 + e^{-2 \cdot b \cdot net}} - a \tag{1.4}$$

1.2 Deep Learning e Reti Neurali Artificiali

Il Deep Learning, (in italiano apprendimento profondo) è una branca del Machine Learning (ML) che a sua volta è una sottocategoria dell'intelligenza Artificiale (AI). Esso si basa sull'apprendimento dei dati "in profondità" su più livelli.

Questa metodologia di "auto apprendimento" si fa sempre più spazio in svariati settori, come ad esempio la scrittura di testi, il riconoscimento di oggetti in immagini e video (una semplice dimostrazione è il riconoscimento facciale), previsioni meteo o economiche, della borsa e molto altro. Per far ciò, i sistemi di deep learning utilizzano varie tipologie di reti neurali, composte da numerosi neuroni artificiali organizzati in livelli. Generalmente la struttura della rete comprende un livello di input, uno o più livelli intermedi (hidden layers) e un livello di output.

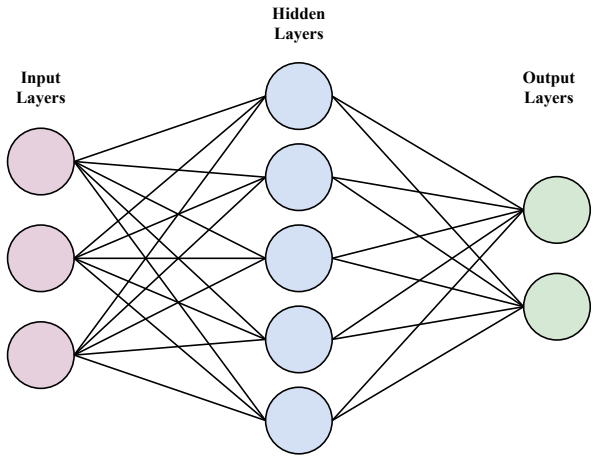

Figura 1.2: Layer di una CNN di Base

Allo stato attuale ci sono diverse tipologie di reti neurali artificiali.
Le più comunemente utilizzate, alle quali daremo una breve introduzione nelle prossime sezioni sono:

- Feed-Forward Neural network (FNN)
  - Perceptron
  - (MLP) Multilayer Perceptron
  - (CNN) Convolution Neural Network
- Recurrent Neural Network (RNN)

1.3 Tipologie di Reti Neurali

1.3.1 Feed-Forward Neural network

La Feed-Forward è formata da una stratificazione di neuroni basata su più livelli. In essa la propagazione delle informazioni in input avviene in una sola direzione. Come precedentemente introdotto, essa è composta fondamentalmente da 3 livelli: Input, Hidden e Output.

Le reti di tipo feedforward sono in grado di apprendere qualsiasi funzione non lineare. Pertanto, queste reti sono comunemente conosciute come Universal Function Approximators.
Uno dei motivi principali alla base dell'approssimazione universale è la funzione di attivazione. Le funzioni di attivazione introducono proprietà non lineari nella rete. Questo aiuta la rete ad apprendere qualsiasi relazione complessa tra input e output.

Percettrone

Il percettrone può essere considerato come il più semplice modello di rete neurale feed-forward in quanto gli input alimentano direttamente l'unità di output attraverso connessioni pesate. Il percettrone utilizza una funzione di attivazione lineare a scalino. Un singolo percettrone o una rete di percettroni a 2 soli livelli (input e output), può essere addestrato con una semplice regola detta

$$\textbf{Delta Rule} \qquad \Delta w_{ij} = \alpha(t_j - y_j) \cdot x_i \tag{1.5}$$

Una rete a 2 livelli di percettroni lineari a soglia è in grado di apprendere solo mapping lineari e pertanto il numero di funzioni approssimabili è piuttosto limitato.

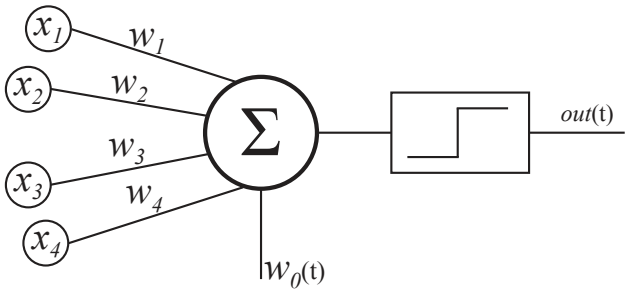

Figura 1.3: Struttura del percettrone

Multilayer Perceptron (MLP)

Questa tipologia di rete Feed-Forward è caratterizzata come le altre da più strati detti layers (Con i tre principali denominati: input, hidden e output). La sua peculiarità sta nel fatto che ogni neurone del livello precedente è collegato a tutti i neuroni del livello successivo.
Ad esempio, immaginiamo una rete formata da 3 livelli. Un livello chiamato Input, un livello chiamato Hidden e come ultimo livello abbiamo Output. Ogni neurone del livello di Input è connesso con ognuno dei neuroni del livello Hidden e a sua volta il livello Hidden ha ogni neurone connesso con tutti i suoi successori al livello Output.

Come precedentemente indicato queste tipologie di reti sono state oggetto del teorema di approssimazione universale, esso afferma che le MLP con un singolo livello intermedio possono approssimare una qualsiasi funzione continua su un insieme compatto di $\mathbb{R}$.

Convolution Neural Network (CNN)

Le reti neurali convoluzionali (CNN) sono di gran moda nella comunità del deep learning. Esse vengono utilizzate in diverse applicazioni e sono particolarmente diffuse nei progetti di elaborazione di immagini e video.
Gli elementi costitutivi delle CNN sono filtri, noti anche come kernel, i quali vengono utilizzati per estrarre le funzionalità rilevanti dall'input utilizzando l'operazione di convoluzione.

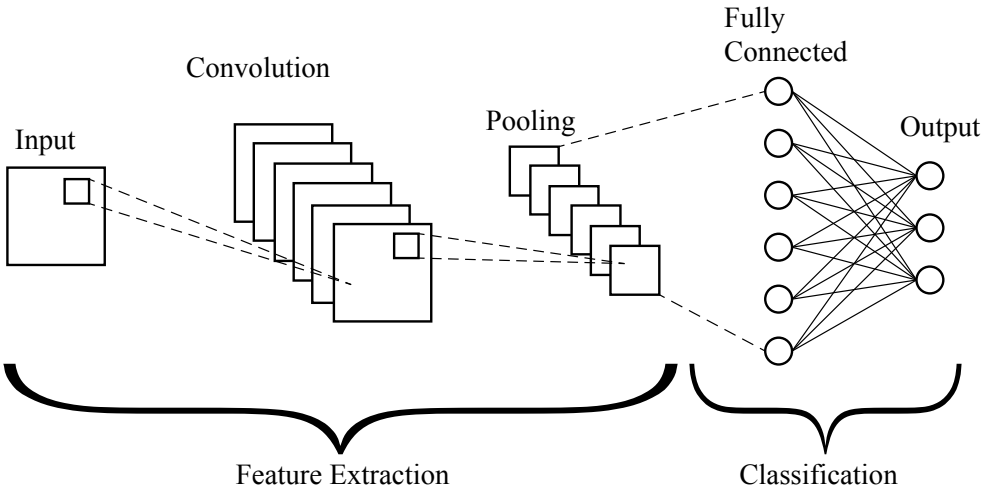

Figura 1.4: Rete CNN con i rispettivi filtri

Sebbene le reti neurali convoluzionali siano state introdotte per risolvere i problemi relativi ai dati delle immagini, hanno prestazioni impressionanti anche sugli input sequenziali.

- La CNN apprende i filtri automaticamente senza menzionarli esplicitamente. Questi filtri aiutano a estrarre le caratteristiche giuste e rilevanti dai dati di input.
- CNN acquisisce le caratteristiche spaziali da un'immagine (la disposizione dei pixel e alla relazione tra loro in un'immagine). Aiutando così ad identificare l'oggetto, la sua posizione e la sua relazione con altri oggetti in un'immagine.

1.3.2 Recurrent Neural Network

Le reti neurali ricorrenti (o RNN, Recurrent Neural Network) hanno la caratteristica di avere connessioni che potrebbero formare dei loop e di conseguenza degli output potrebbero diventare a loro volta degli input per lo stesso neurone che li ha generati. La particolarità di questa rete è dunque quella di poter propagare dati sia in avanti come le feedforward ma anche indietro. I cicli complicano notevolmente la rete ma allo stesso tempo permettono ad essa di gestire diverse tipologie di input, come ad esempio la generazione di testi più articolati.

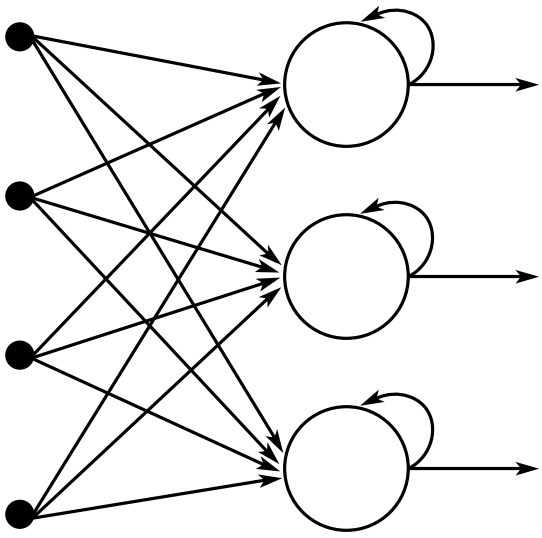

Figura 1.5: Rete RNN con loop di connessione

1.4 Semantic Segmentation

L'intelligenza artificiale ha compiti diversi per il trattamento delle immagini. In questa sezione introdurremo brevemente due delle metodologie più utilizzate nell'ambito del riconoscimento degli oggetti.

- Rilevamento di oggetti
- Segmentazione dell'immagine

In entrambe le attività, l'obbiettivo è quello di determinare le posizioni dei vari elementi presenti all'interno di un'immagine. Un classico esempio è il caso in cui abbiamo una sequenza di immagini di animali e su ciascuna immagine vogliamo identificare le esatte posizioni di tutti gli animali di una determinata specie.

1.4.1 Rilevamento di Oggetti

Con il termine rilevamento oggetti, andiamo ad indicare generalmente i riquadri di delimitazione. Essi infatti sono banalmente dei rettangoli attorno a ciascun oggetto o soggetto che vogliamo identificare. Generalmente i riquadri di delimitazione sono definiti dalla posizione dell'angolo in alto a sinistra e da una larghezza e un'altezza.

1.4.2 Introduzione alla Segmentazione Semantica

L'obiettivo della segmentazione semantica (dall'inglese semantic segmentation) invece, è quello di associare un'etichetta o una categoria a ogni pixel presente nell' immagine con una classe corrispondente di ciò che viene rappresentato. Si può notare una certa somiglianza con l'algoritmo di rilevamento di oggetti precedentemente introdotto, tuttavia la caratteristica che contraddistingue le reti di segmentazione semantica è quella che esse possono rilevare oggetti di forma irregolare o che si estendono in più aree. Dunque la precisione della segmentazione semantica è complessivamente superiore.

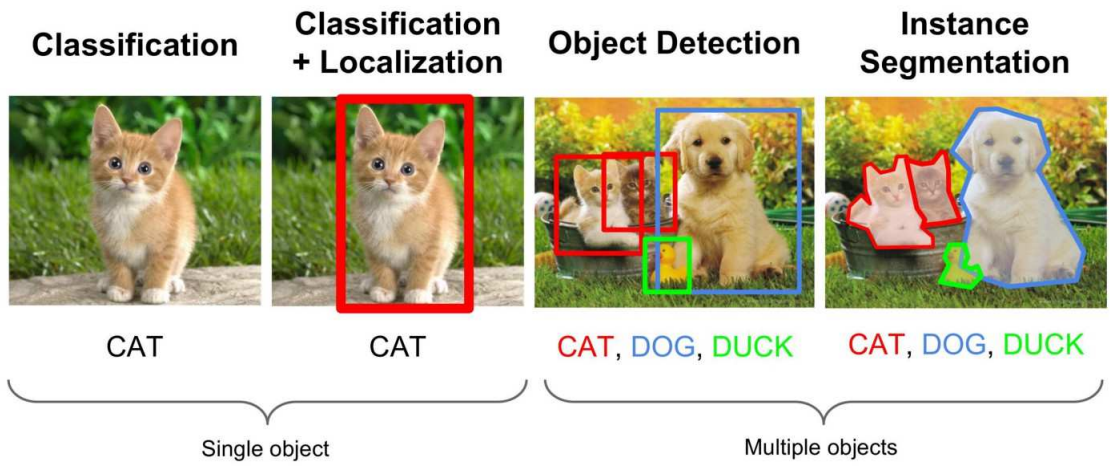

Figura 1.6: Differenti tipologie di classificazione di oggetti

1.4.3 Funzionamento di Base

I modelli di segmentazione sono utili per una varietà di attività, tra questi i più conosciuti sono sicuramente:

- Sistemi di guida autonoma
  Conferiscono alle auto la percezione necessaria per comprendere l'ambiente che le circonda e poter muovere il veicolo in maniera del tutto indipendente.
- Diagnostica per esami medici
  Riducendo notevolmente il tempo necessario per eseguire i test diagnostici.

Le architetture più comuni per la segmentazione semantica sono basate su CNN, suddivise in una parte encoder, e una parte decoder, questo tipo di architettura è chiamata SegNet. La parte encoder della rete ha il compito di estrarre le caratteristiche importanti dall'immagine di input. Come in una normale CNN, queste operazioni sono chiamate Down-Sampling.

La parte decoder della rete è fondamentalmente il riflesso dell'encoder, ma al posto dei livelli convoluzionali (CONV), ci sono strati up-convoluzionali (UP-CONV), oltre a ciò in questa fase abbiamo l'operazione di Up-Sampling. L'architettura del decoder può essere molto diversa nelle varie reti, ma solitamente gli strati UP-CONV sono sempre presenti. Essi si basano sull'operazione di convoluzione trasposta, che prende ogni singolo valore proveniente dal volume in ingresso e lo moltiplica per una matrice di pesi simile ai filtri. Dopo questa operazione i valori vengono disposti spazialmente, tenendo conto dei parametri di padding e striding come in Figura 1.7.

Banalmente il nostro obiettivo è quello di acquisire un'immagine a colori RGB (altezza × larghezza × 3) o un'immagine in scala di grigi (altezza×larghezza×1) e genera una mappa di segmentazione in cui ogni pixel contiene un'etichetta di classe rappresentata come un numero intero (altezza×larghezza×1).

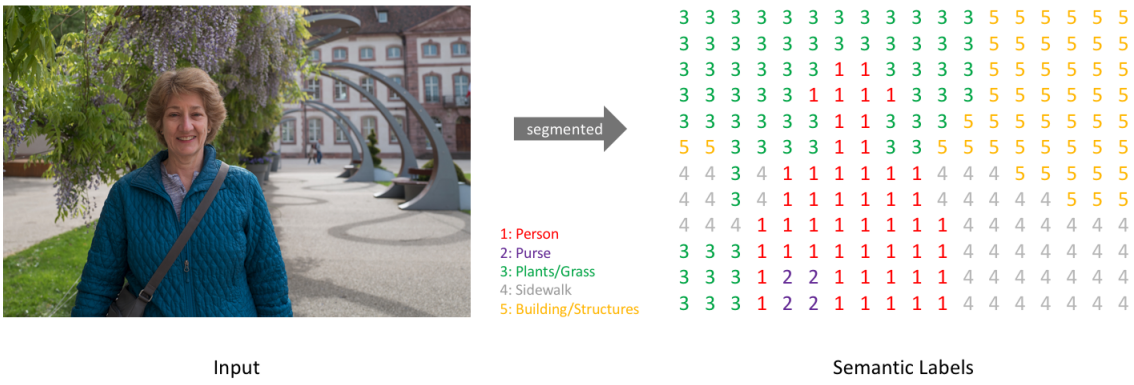

Figura 1.7: Mappatura dei pixel di un'immagine

Creiamo il nostro target codificando le etichette delle classi, realizzando così, un canale di output per ciascuna delle possibili classi.

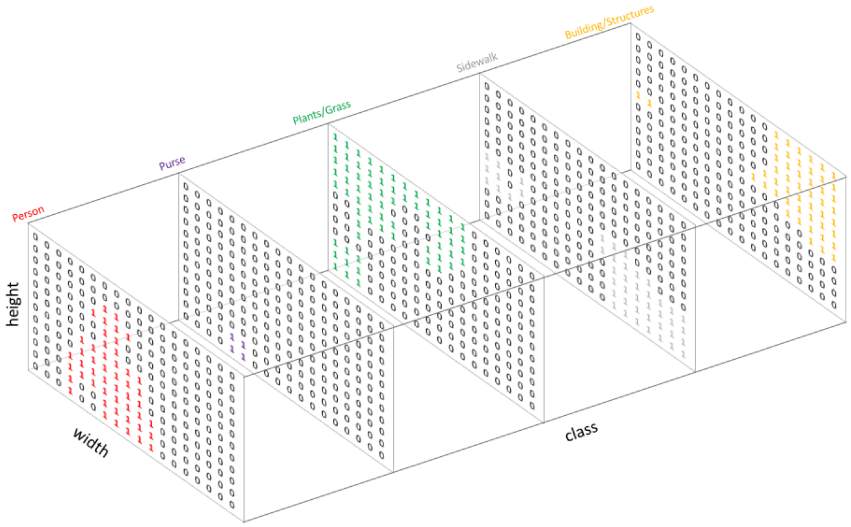

Figura 1.8: Classi degli oggetti che compongono l'immagine

Una previsione può essere compressa in una mappa di segmentazione (come mostrato nella prima immagine) prendendo l'argMax di ciascun vettore di pixel in profondità. Possiamo facilmente ispezionare un obiettivo sovrapponendolo all'osservazione.

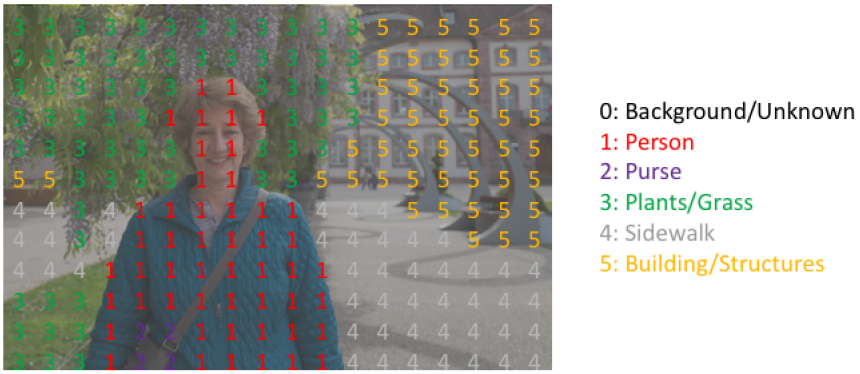

Figura 1.9: Sovrapposizione dell'immagine e della relativa maschera

Quando sovrapponiamo un singolo canale del nostro obiettivo (o previsione), ci riferiamo a questo come a una maschera che illumina le regioni di un'immagine in cui è presente una classe specifica.

Capitolo 2

Funzioni Obbiettivo

Nel presente capitolo andremo a dare una breve introduzione alle funzioni obbiettivo, Inizialmente parleremo della loro utilità nelle reti neurali, del loro funzionamento e di alcuni utilizzi. Successivamente andremo ad introdurre alcune tra le funzioni obbiettivo più utilizzate. La Mean Squared Error e la Binary Cross-Entropy, concludendo con alcuni cenni ad ulteriori funzioni.

2.1 Significato di Funzione Obbiettivo - Loss Functions

Gli algoritmi di apprendimento supervisionato necessitano di uno strumento atto a misurare la qualità delle predizioni in funzione ai parametri del modello. Le funzione obiettivo (dall'inglese loss function) anche chiamate funzioni di perdita, sono una parte importante nelle reti neurali.
Esse vengono utilizzate in fase di training per misurare l'incoerenza tra il valore atteso y' e il valore effettivo y, tramite l'aggiornamento dei pesi. In questo modo l'accuratezza del modello aumenta insieme alla diminuzione del valore della funzione di loss (che non può mai essere negativo, esso infatti può tendere a un valore nullo senza mai oltrepassare la soglia).

É dunque possibile affermare che le loss function in generale hanno un ruolo fondamentale nella definizione del modello e dei rispettivi parametri. Esse infatti riducono tutti gli aspetti positivi e negativi di un sistema complesso in un unico valore scalare, che consente di classificare e comparare la soluzione considerata.

2.2 Tipologie di Funzioni Obbiettivo

Esistono differenti tipologie di loss function che tentano di misurare la differenza tra valori reali e predetti. In base alla loss scelta si cerca di creare una sorta di parametro di regolarizzazione che incentiva determinate scelte penalizzando alcuni errori, di conseguenza modificando ciò che il modello apprende. La funzione obbiettivo potrá dunque avere differenti risultati. Essa va quindi scelta accuratamente in base al tipo di training che si desidera svolgere.

Il goal è quello di selezionare una loss che permetta di minimizzare l'errore della rete artificiale, ed allo stesso tempo punti a massimizzare la selezione corretta dei valori.

In generale, come già anticipato, sono presenti diverse tipologie di loss functions (Dice Loss, Tversky Loss, Cross-Entropy Loss, Top-K Loss e molte altre) che mirano ad ottimizzare i più disparati problemi. Quelle che andremo ad enunciare in questa tesi sono la Mean Squared Error e la Binary Cross-Entropy.

2.2.1 Mean Squared Error

L'errore quadratico medio (Mean Squared Error) misura la media dei quadrati degli errori, ovvero la differenza quadratica media tra i valori stimati e il valore effettivo. MSE è una funzione di rischio, corrispondente al valore atteso della perdita di errore al quadrato.

Essendo la MSE altamente sensibile ai valori anomali, essa viene largamente utilizzata nei problemi di regressione. In quanto essa tiene in considerazione maggiormente i valori di input distribuiti attorno a un valore medio e penalizzando maggiormente i valori anomali.
La formula è la seguente:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_j)^2 \tag{2.1}$$

Ove y corrisponde al valore desiderato ed $\tilde{y}$ corrisponde alla previsione fatta dalla rete neurale.

Esempio: vuoi prevedere i prezzi delle case futuri. Il prezzo è un valore continuo e quindi vogliamo fare una regressione. MSE può dunque essere utilizzato come funzione di perdita.

Come anticipato, la MSE e molto utile nei problemi di regressione, ossia quando si ritiene che l'obbiettivo, condizionato dall'input, sia normalmente distribuito e si desideri che gli errori "grandi" siano significativamente più penalizzati di quelli "piccoli".

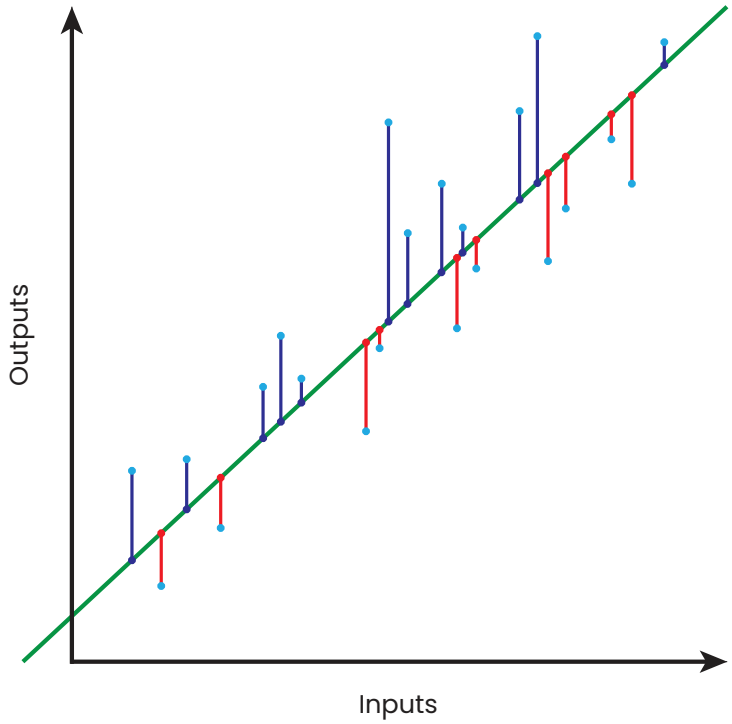

Figura 2.1: Esempio di grafico di Mean Squared Error

Il fatto che questa particolare loss sia strettamente positiva (e non zero) è dovuto alla casualità o al fatto che non si tenga conto di informazioni che potrebbero produrre una stima più accurata. Nell'apprendimento automatico, MSE può fare riferimento alla perdita media su un set di dati osservato, come una stima della perdita media sulla distribuzione effettiva della popolazione.

Poiché deriva dal quadrato della distanza euclidea, è sempre un valore positivo che diminuisce man mano che l'errore si avvicina a zero.

2.2.2 Binary Cross-Entropy

Ricordiamo inanzitutto che l'entropia è il valore atteso dell'informazione. Una distribuzione in cui gli eventi sono equiprobabili porta ad un'entropia elevata. Di conseguenza una distribuzione sbilanciata si tramuta in un'entropia bassa. Questo perchè nella seconda dominano gli eventi più probabili.
L'entropia H(X) può essere calcolata per una variabile aleatoria discreta X come segue:

$$H(x) = -\sum_{a \in A_x} p_x(a) \cdot \log(p_x(a)) \tag{2.2}$$

La Binary Cross-Entropy é una tipologia di Cross-Entropy la quale, misura la dimensione di codifica media necessaria a riconoscere un elemento dell'insieme C (di dimensione due), fra due distribuzioni di probabilità.
Essa è definita come:

$$H(P, Q) = -\sum_{i=1}^{C=2} P(i) \log Q(i) = -P(i_1) \log Q(i_1) - P(i_2) \log Q(i_2) \tag{2.3}$$

Ove con i_1 e i_2 indichiamo gli eventi dell'insieme C.

Comparando la vera distribuzione di probabilità, estrapolata dai dati già classificati in un contesto supervisionato, con le previsioni del modello, la Binary Cross-Entropy diminuisce man mano che le previsioni diventano sempre più accurate. Diventando zero se la previsione è corretta.

Grazie alle sue proprietà la Binary Cross-Entropy viene utilizzata come funzione di costo, per problemi di classificazione binaria e multiclasse.

2.2.3 Cenni ad Ulteriori Funzioni Obbiettivo

Esistono moltissime altre tipologie di loss Functions oltre a quelle appena citate, tra le quali troviamo:

TverskyLoss

L'indice di Tversky, utilizzato per il calcolo della Tversky Loss Function, a differenza del Dice Index (che vedremo in seguito), aggiunge un peso a ai Falsi Positivi (FP) e ai Falsi Negativi (FN) aggiungendo dunque un coefficiente moltiplicativo β.
La definizione formale del Tversky Index è dunque:

$$TI(Y,T) = \frac{YT}{YT + \beta(1-Y)T + (1-\beta)Y(1-T)} \tag{2.4}$$

Dal quale si può in seguto definire la Tversky Loss Function che corrisponde a:

$$L_{Tversky}(Y,T) = 1 - \frac{YT + \epsilon}{YT + \beta(1-Y)T + (1-\beta)Y(1-T) + \epsilon} \tag{2.5}$$

NOTA: É stato aggiunto $+\,\epsilon$ al numeratore e al denominatore per gestire il caso limite nel quale $Y = \overline{T} = 0$.

DiceLoss

La Dice Loss è una funzione che si basa il coefficiente Sørensen-Dice, metrica usata per valutare la somiglianza tra due campioni e le performance dei modelli di image segmentation. Il coefficiente Sørensen-Dice dunque, date in input due immagini, determina quanto simili sono, fornendo in output un valore nell' intervallo [0, 1].
La definizione formale del coefficiente Sørensen-Dice è dunque:

$$Dice\,Coefficient = D(Y,T) = \frac{2\,|Y \cap T|}{|Y| + |T|} \tag{2.6}$$

La Dice Loss Function sarà così definita:

$$L_{Dice}(Y,T) = 1 - D(Y,T) = 1 - \frac{2Y\overline{T} + \epsilon}{Y + \overline{T} + \epsilon} \tag{2.7}$$

NOTA: É stato aggiunto $+\,\epsilon$ al numeratore e al denominatore per gestire il caso limite nel quale y $= \bar{t} = 0$.
per $\beta = \frac{1}{2}$, Dice Loss risulta analoga a Tversky Loss.

PixelLoss

Come suggerisce il nome, questo tipo di funzione di perdita calcola la perdita da pixel a pixel della previsione e delle immagini di destinazione. La funzione di perdita misura le differenze tra i valori dei pixel di output in un'immagine.
La gran parte delle funzioni di perdita come la MSE (Mean Squared Error) o la MAE (Mean Absolute Error), possono essere applicate tra ogni coppia di pixel della previsione e le variabili di destinazione.

In questo caso specifico sfrutta la perdita MSE a livello di pixel calcola quanto sono diverse le immagini generate dalle immagini reali.
La formula per calcolare la perdita MSE a livello di pixel è la seguente:

$$l^{SR}_{MSE} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I^{HR}_{x,y} - G_{\theta_G}(I^{LR})x, y)^2 \tag{2.8}$$

Qui $G_{\theta_G}(I^{LR})$ rappresenta un'immagine ad alta risoluzione generata dalla rete generata. Metre I^{HR} rappresenta un'immagine ad alta risoluzione campionata.

Capitolo 3

AutoLoss-Zero: Ricerca della Loss Function

Gli ultimi anni hanno assistito a entusiasmanti progressi nell'AutoML ossia un processo il cui obbiettivo é quello di eseguire automaticamente molti dei task lunghi e ripetitivi che che si presentano nello sviluppo di un modello. In questo capitolo discuteremo di AutoLoss-Zero, il quale propone di automatizzare il processo di progettazione di funzioni di perdita per compiti generici tramite il suo algoritmo di ricerca.

3.1 Cenni di AutoML

L'AutoML ovvero Automated Machine Learning, é un processo il cui scopo é quello di automatizzare molte delle attività lunghe e ripetitive che sono necessarie per lo sviluppo di un modello.
Lo sviluppo di modelli di apprendimento automatico tradizionali richiedono molte risorse, una notevole conoscenza del dominio e tempo per produrre e confrontare un discreto numero di modelli. Questo metodo di apprendimento consente anche ai meno esperti in questo campo di utilizzare tecniche e modelli di machine learning.

Ulteriori vantaggi del processo di applicazione end-to-end dell'apprendimento automatico mirano a produrre in maniera più rapida soluzioni semplici e modelli più "raffinati" rispetto a quelli progettati manualmente.

Tecniche comuni utilizzate in AutoML includono:

- L'ottimizzazione degli iperparametri

  Una corretta selezione di questi valori permette un accurato addestramento del modello e dunque dell'accuratezza finale della rete neurale.
  Alcuni iperparametri degni di nota sono: Il learning rate, le funzioni di attivazione, il numero di neuroni e il numero di epoche.

- Il meta-apprendimento

  Esso sfrutta gli algoritmi di apprendimento automatico applicati ai metadati sugli esperimenti di machine learning. L'obiettivo principale è utilizzare tali metadati per migliorare le prestazioni degli algoritmi di apprendimento esistenti.

- Rcerca dell'architettura neurale

  Viene applicato il metodo di ricerca dell'architettura neurale a problemi basati sulla discesa del gradiente, sull'apprendimento per rinforzo e sugli algoritmi evolutivi.

3.2 Introduzione ad AutoLoss-Zero

Negli ultimi anni ci sono stati numerosi progressi nel campo dell'Auto Machine Learning. Sfortunatamente però, la progettazione automatica delle funzioni di perdita per compiti generici rimane un settore ancora poco esplorato. In varie attività, come ad esempio la segmentazione semantica ed il rilevamento oggetti, le funzioni di perdita rappresentano parti essenziali nella formazione della rete.

Dato che tali funzioni sono spesso approssimazioni, esiste un disallineamento tra il valore della funzione di perdita e la metrica di valutazione, che normalmente si traduce in soluzioni non ottimali. Per ridurne gli effetti di questo problema o ove possibile, risolverlo completamente, ci sono due principali metodologie. La

prima è la progettazione di varianti differenziabili delle metriche di valutazione, mentre la seconda consiste nella progettazione manuale delle funzioni di perdita sostitutive basate sull'espressione matematica delle metriche di valutazione.
Nonostante queste funzioni di perdita artigianali mostrino significativi miglioramenti rispetto alle metriche target, fanno affidamento su analisi di scenari specifici, le quali limitano la loro estensibilità. Se tale euristica specifica del compito possa essere applicata a compiti generici non è verificato.

AutoLoss-Zero ha quindi come obbiettivo quello di automatizzare la progettazione di funzioni di perdita per compiti generici. Esso formula funzioni di perdita come grafici computazionali composti esclusivamente da operatori matematici primitivi. I grafici di calcolo sono costruiti in modo casuale da zero e si evolvono in base alle loro prestazioni sulle metriche di valutazione.

Nell'algoritmo di ricerca, per migliorarne l'efficienza, viene proposto l'utilizzo del Loss-Rejection Protocol e Gradient Equivalence Strategy, che filtra i candidati alla funzione di perdita poco promettenti e ne evita valutazioni duplicate.

3.3 Struttura dell'Algoritmo di Ricerca

Dato un compito (ad es. segmentazione semantica) e una metrica di valutazione corrispondente, AutoLoss-Zero mira a realizzare da zero ed in maniera automatica una funzione di perdita per l'addestramento di una rete neurale. Viene proposto uno spazio di ricerca generale, in cui ogni funzione di perdita è rappresentata come un grafico computazionale. Il grafico prende le previsioni della rete e le verità di base come input,trasformandole in un valore di perdita finale. Con un'esperienza umana minima, solo le operazioni matematiche primitive vengono utilizzate come nodi computazionali intermedi per accogliere l'elevata diversità tra i vari compiti e le varie metriche.

Per consentire l'evoluzione, (tramite un algoritmo evolutivo sfruttato per cercare la funzione di perdita con compito e metrica dati), vengono definite efficaci operazioni di inizializzazione casuale e di mutazione.

AutoLoss-Zero ricerca le loss functions per attivitá generiche dall'inizializzazione casuale con una competenza umana minima. Il metodo proposto non ha una

progettazione specializzata per attività o metriche specifiche, dunque esso è ampiamente applicabile alle attività generiche.

La struttura dell'algoritmo di AutoLoss-Zero è formata essenzialmente da 5 livelli:

- Inizializzazione delle funzioni di perdita e ricerca della popolazione
- Mutazione
- Loss-Rejection Protocol
- Gradient-Equivalence-Check Strategy
- Proxy Task

La figura 3.1 illustra la pipeline di ricerca di AutoLoss-Zero.

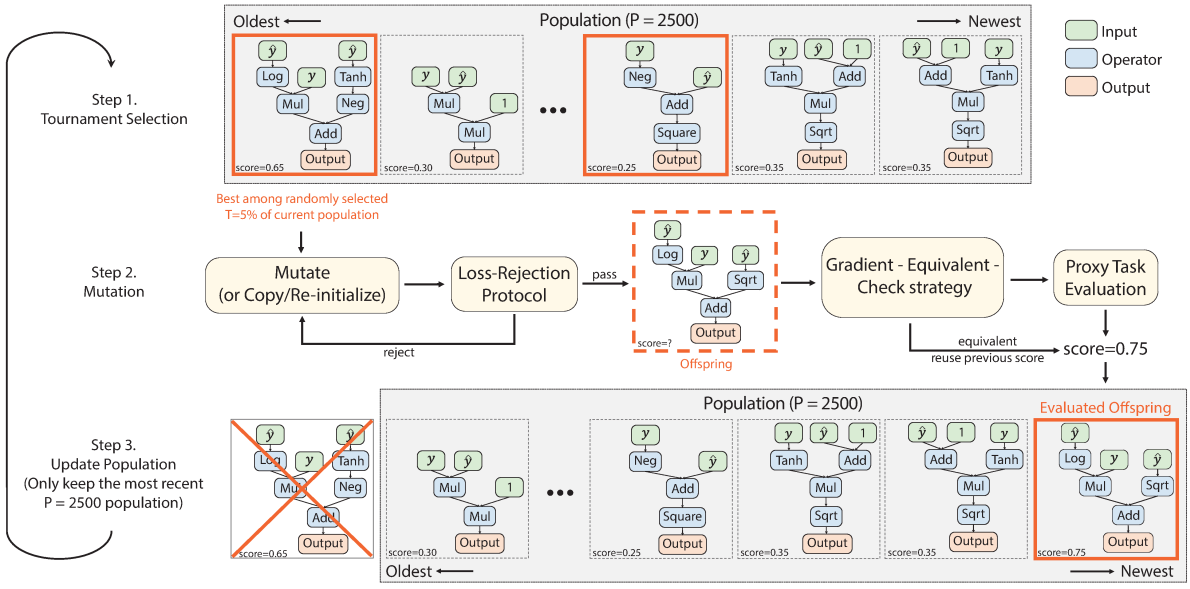

Figura 3.1: Rappresentazione grafica dell'algoritmo di ricerca AutoLoss-Zero

Inizializzazione delle funzioni di perdita e ricerca della popolazione

All'inizializzazione, il grafico computazionale di ciascuna funzione di perdita K ($K = 20$ per impostazione predefinita) viene generato casualmente per formare la popolazione iniziale.

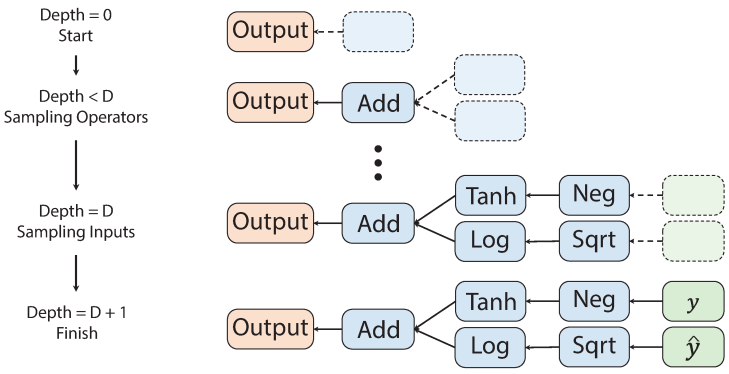

Figura 3.2: Rappresentazione grafica dell'algoritmo di ricerca AutoLoss-Zero

Le loss function vengono quindi rappresentata da un albero, che come nodo radice ha l'output della loss. Ogni nodo campiona casualmente uno o due operatori ta quelli indicati in Figura 3.3 e li aggiunge al grafico come suo/i nodo/i figlio/i.

Element-wise Operator	**Expression**	**Arity**
Add	$x + y$	2
Mul	$x \times y$	2
Neg	$-x$	1
Abs	$\|x\|$	1
Inv	$1/(x + \epsilon)$	1
Log	$\text{sign}(x) \cdot \log(\|x\| + \epsilon)$	1
Exp	e^x	1
Tanh	$\tanh(x)$	1
Square	x^2	1
Sqrt	$\text{sign}(x) \cdot \sqrt{\|x\| + \epsilon}$	1
†**Aggregation Operator**	**Expression**	**Arity**
Mean_{nhw}	$\frac{1}{NHW} \sum_{nhw} x_{nchw}$	1
Mean_{c}	$\frac{1}{C} \sum_{c} x_{nchw}$	1
$\text{Max-Pooling}_{3\times3}$	$\text{Max-Pooling}_{3\times3}(x)$	1
$\text{Min-Pooling}_{3\times3}$	$\text{Min-Pooling}_{3\times3}(x)$	1

Figura 3.3: Tabella delle operazioni primitive

Quando un nodo computazionale raggiunge la profondità target D ($D = 3$ per impostazione predefinita), seleziona casualmente i tensori di input dall'insieme $\{y, \hat{y}, 1\}$, (ove 1 ha lo scopo di migliorare la flessibilitá dello spazio di ricerca) i quali sarebbero i nodi foglia del grafo computazionale. Ogni grafo computazionale generato casualmente ha una profondità di $D + 1$, con D nodi computazionali su ciascun percorso dalla radice a un nodo foglia.

l valore finale della funzione di perdita si ricava dunque dall'output:

$$L(\hat{y}, y) = \frac{1}{NHW} \sum_{nchw} o_{nchw} \tag{3.1}$$

Mutazione

Ispirato da [8], il processo di mutazione è formato da una sequenza di operazioni opportunamente riprogettate per l'opportuno spazio di ricerca.
Le operazioni che lo contraddistinguono sono dunque così definite:

Insertion

Un operatore campionato casualmente viene inserito tra un nodo non radice (selezionato in modo casuale) e il suo genitore come indicato in Figura 3.4.

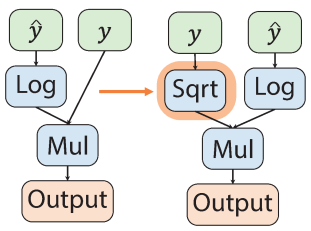

Figura 3.4: Inserimento di un operatore non campionato

Deletion

Un nodo intermedio viene selezionato e rimosso casualmente. Uno dei suoi nodi figlio viene scelto (in modo casuale) per diventare il nuovo figlio del suo genitore.

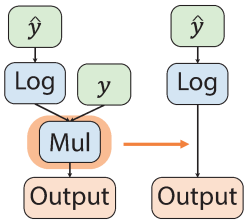

Figura 3.5: Rimozione casuale di un nodo

Replacement

Un operatore viene campionato casualmente per sostituire un nodo non radice (selezionato in modo casuale). Se il nodo non radice ha più figli dell'operatore, un sottoinsieme casuale dei nodi figli con lo stesso numero viene mantenuto come figli dell'operatore.

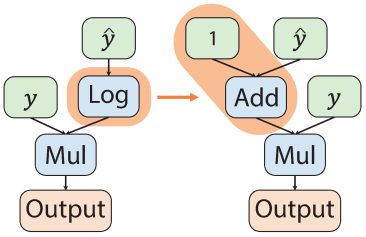

Figura 3.6: Sostituzione casuale di un noto

Loss-Rejection Protocol

Il Loss-Rejection Protocol qui descritto, filtra le funzioni di perdita poco promettenti prima di addestrare le reti. A differenza però di altri protocolli di ricerca esso è generalmente applicabile a vari compiti e metriche.

Questo protocollo si basa sul fatto che la riduzione al minimo delle corrette funzioni di perdita dovrebbe corrispondere alla massimizzazione della metrica di valutazione.

Dati B campioni casuali ($B = 5$ di default) dal training set e una rete inizializzata casualmente, registriamo le previsioni di rete e i corrispondenti target di training. Per stimare in modo efficiente la correlazione tra la data metrica di valutazione ξ e una funzione di perdita candidata L, si calcola un punteggio di correlazione:

$$g(L, \xi) = \frac{1}{B} \sum_{b=1}^{B} \xi(\hat{y}_b^*(L), y_b) - \xi(\hat{y}_b, y_b) \tag{3.2}$$

dove $\hat{y}_b^*(L)$ è la previsione della rete ottimizzata con la funzione di perdita L.

Una grande $g(L;\xi)$ indica che minimizzare la funzione di perdita L corrisponde a massimizzare la data metrica di valutazione ξ. Altrimenti, se $g(L; \xi)$ è minore di una soglia η, la funzione di perdita L è considerata poco promettente.

Per accelerare il processo di rifiuto, l'ottimizzazione della funzione di perdita viene applicata direttamente alla previsione di rete $\hat{y}_b$, invece che ai parametri di rete. Poiché il calcolo della rete viene omesso, il processo di rifiuto è molto efficiente.

Gradient-Equivalence-Check Strategy

Per evitare di rivalutare più volte le medesime funzioni di perdita, è stata sviluppata la strategia di verifica dell'equivalenza del gradiente (Gradient-Equivalence-Check Strategy). Per ogni singola loss, calcoliamo le sue norme di gradiente rispetto alle previsioni di rete utilizzate nel protocollo di rifiuto della perdita. Se due funzioni di perdita hanno le medesime norme di gradiente per tutti i campioni B, allora sono considerate equivalenti. Dunque il punteggio della metrica di valutazione verrebbe riutilizzato.

Proxy Task

La valutazione delle funzioni di perdita è l'operazione più costosa a livello di tempistiche nel processo di ricerca. Per accelerare il processo di ricerca, si sfruttano attività proxy leggere per l'addestramento di rete. Vengono quindi adottate meno iterazioni di addestramento, modelli più piccoli e immagini sottocampionate. Incrementando inoltre l'efficienza interrompendo l'addestramento della rete con valori di perdita non validi (valori NaN e Inf).

Capitolo 4

Fase Sperimentale

In quest'ultimo capitolo introdurremo i metodi e i dataset utilizzati nell'implementazione di AutoLoss-Zero e l'applicazione del suddetti metodi di ricerca di una loss function alla rilevazione di polpi del colon tramite segmentazione semantica. Per concluder andremo ad analizzare i risultati sperimentali ottenuti.

4.1 Metodologie e Dataset Utilizzati

Il dataset utilizzato per questa ricerca è denominato Kvasir-SEG [4]. Questo dataset è formato da 1000 immagini di polpi e 1000 maschere (1 per ogni immagine). Le suddette immagini sono state acquisite da un sistema di imaging elettromagnetico ad alta risoluzione e una base di conoscenza (ground-truth) composta da riquadri di delimitazione e maschere di segmentazione.

Inizialmente abbiamo suddiviso questo dataset in 3 dataset di dimensione più piccola. Il dataset principale, formato da 780 immagini e 780 maschere è quello sfruttato per la fase di training, quindi dove effettivamente è stato addestrato il nostro algoritmo. Il secondo composto da 120 immagini e 120 maschere è stato fondamentale per effettuare la fase di validation. L'ultimo dataset comprendeva le 100 immagini e le 100 maschere rimanenti. Esso è stato utilizzato solo ed esclusivamente per la fase finale di testing.

Come largamente anticipato abbiamo implementato AutoLoss-Zero come metodo per ricavare una funzione di perdita efficace da implementare in questo progetto di segmentazione semantica.

I nostri test si sono basati su alcune varianti di modelli di segmentazione semantica. Tra questi troviamo:

- DeepLabv3 [6]
- DeepLabv3+ [10]
- ResNet-18 [12]
- ResNet-50 [2]

Sfortunatamente, a causa degli elevati tempi per l'addestramento della rete, abbiamo utilizzato solo inizialmente ResNet-50, per poi passare all'utilizzo di ResNet-18. Il quale garantisce tempi di addestramento molto inferiori, a discapito di risultati leggermente meno accurati.

Un ulteriore algoritmo che abbiamo utilizzato per aggiornare i vari pesi e i vari parametri della rete neurale è lo Stochastic Gradient Descent.

Nella fase di training abbiamo settato alcuni degli iperparametri più rilevanti come segue:

- **Learnig Rate:** 0,01
- **Max Epoch:** 5
- **Loss:** autoloss-zero
- **Rete:** resnet18
- **Model Name:** res18
- **Mini Batch Size:** 5

Per quanto riguarda l'allenamento finalizzato alla valutazione (Proxy Task) abbiamo aumentato il learning rate a 0,02 e diminuito il numero di epoche così da ridurre i tempi computazionali.

Dunque in questo caso gli iperparametri sono cambiati come segue:

- **Learnig Rate:** 0,02
- **Max Epoch:** 2
- **Loss:** autoloss-zero
- **Rete:** resnet18
- **Model Name:** res18
- **Mini Batch Size:** 5

Un' ultima fase prima di effettuare i vari test è stata quella di eliminare i bottle necks presenti nel codice, i quali allungavano inutilmente le fasi di training. Tra questi troviamo un problema che riportava costantemente il parametro Max Epoch a 500. Esse rendeva inefficace ogni tentativo di ridurre le tempistiche delle varie vasi (anche andando ad agire direttamente sul valore delle Epoch).
A questo punto è stato possibile inziare l' effettiva fase di test.

4.2 Risultati Sperimentali

I primi test sono stati effettuati utilizzando esclusivamente i dataset di training e di test. Tralasciando dunque la fase di validazione. Questo purtroppo non ha portato ai risultati sperati, ogni prova effettuata portava i risultati a una non convergenza.
Successivamente ulteriori test sono stati effettuati aggiungendo la validazione, così da bloccare e terminare anticipatamente eventuali loss che non portavano a risultati esaustivi. Sfortunatamente anche in questo caso non è stato possibile arrivare ai risultati sperati. Ulteriori test sono stati effettuati per verificare la presenza di errori nel codice tramite l'utilizzo di una funzione stabile come la DiceLoss e diminuendo il numero di MaxEpoch a 2-3. In questo caso abbiano riscontrato una convergenza e di conseguenza il problema che si era presentato non era dato da problemi nella stesura del codice. Questo risultato ci indicava dunque che il problema era circoscritto alla generazione delle loss, e che probabilmente il metodo richiedeva un elevato spazio di ricerca per trovare delle loss sensate che portassero ad una convergenza.

Vista dunque l'eccessiva potenza di calcolo richiesta dall'algoritmo di ricerca, per ottenere risultati non abbiamo potuto procedere oltre. Infatti, le valutazioni delle loss function che l'algoritmo generava, aumentano notevolmente la sua complessitá computazionale. Non possiamo dunque affermare con certezza che l'applicazione di AutoLoss-Zero per la generazione di funzioni obbiettivo per la segmentazione semantica porti importanti miglioramenti nelle prestazioni.

Ad avvalorare la nostra tesi e che quindi AutoLoss-Zero possa portare significativi miglioramenti a progetti di segmentazione presentiamo i risultati ottenuti da [1] nella medesima applicazione.
In Figura 4.1 vengono confrontati i risultati ottenuti da AutoLoss-Zero con la cross-entropy loss e altre funzioni progettate a mano per metriche specifiche e con funzioni create con Auto Seg-Loss [5]. Ricordiamo però che quest'ultimo non pùo essere esteso alla gestione di metriche generiche.
I risultati ottenuti ci dicono che le funzioni di AutoLoss-Zero superano in prestazioni quelle progettate manualmente e sono superiori (anche se di poco) alle funzioni di Auto Seg-Loss.
Per ottenere dei risultati positivi sono stati utilizzati DeepLabv3+ [10] e ResNet-101 [12] come modelli di segmentazione semantica. Come dataset invece è stato utilizzato PASCAL VOC [11].

Loss Function		mIoU	FWIoU	gAcc	mAcc	BIoU	BF1
Cross Entropy		78.7	91.3	**95.2**	87.3	70.6	65.3
WCE		69.6	85.6	91.1	**92.6**	61.8	37.6
DiceLoss		77.8	91.3	95.1	87.5	69.9	64.4
Lovàsz		79.7	**91.8**	**95.4**	88.6	72.5	66.7
DPCE		79.8	**91.8**	**95.5**	87.8	71.9	66.5
SSIM		79.3	**91.7**	**95.4**	87.9	71.5	66.4
mIoU	ASL	**81.0**	**92.1**	**95.7**	88.2	73.4	68.9
	Ours	**80.7**	**92.1**	**95.7**	89.1	74.1	66.0
FWIoU	ASL	80.0	**91.6**	**95.4**	89.2	75.1	65.7
	Ours	78.7	**91.7**	**95.2**	87.7	72.9	64.6
gAcc	ASL	79.7	**91.8**	**95.5**	89.0	74.1	64.4
	Ours	79.4	**91.7**	**95.3**	88.7	73.6	64.8
mAcc	ASL	69.8	85.9	91.3	**92.7**	72.9	35.6
	Ours	75.3	89.2	93.7	**92.6**	73.7	44.1
BIoU	ASL	49.0	69.9	62.6	81.3	**79.2**	39.0
	Ours	39.8	66.6	79.7	47.8	77.6	45.5
BF1	ASL	1.9	1.0	2.7	6.5	7.4	74.8
	Ours	6.0	54.6	73.8	7.3	9.4	**79.8**

Figura 4.1: Risultati ottenuti da AutoLoss-Zero

Al fine di corroborare maggiormente l'idea che AutoLoss-Zero porti ad effettivi miglioramenti prestazionali anche al nostro progetto di rilevazione dei polpi in Figura 4.2 vengono mostrati ulteriori risultati ottenuti da [1], utilizzando differenti tipologie di dataset e reti neurali, i quali mostrano dunque le potenzialità e l'elevata versatilità di generalizzare di AutoLoss-Zero.

Dataset		**Cityscapes**		**VOC**			
Network		**R101-DLv3+**		**R50-DLv3+**		**R101-PSP**	
Loss Function		**mIoU**	**BF1**	**mIoU**	**BF1**	**mIoU**	**BF1**
Cross Entropy		80.0	62.2	76.2	61.8	77.9	64.7
mIoU	ASL	**80.7**	66.5	**78.4**	66.9	**78.9**	65.7
	Ours	**80.4**	63.8	**78.0**	62.8	**78.5**	64.9
BF1	ASL	6.7	**78.0**	1.4	**70.8**	1.6	**71.8**
	Ours	9.3	**77.7**	7.2	**78.3**	5.1	**71.3**

Figura 4.2: Ulteriori risultati ottenuti da AutoLoss-Zero

Conclusioni

In questa tesi è stato descritto e sperimentato un metodo di AutoML (Automated Machine Learning) applicandolo ad un progetto di segmentazione semantica per la rilevazione di polipi del colon.
Dopo una prima panoramica generale sulle reti neurali e sulle loss finctions, si è mostrato più nel dettaglio il metodo utilizzato, ovvero AutoLoss-Zero. Il quale, mira ad automatizzare la progettazione di funzioni di perdita per compiti generici, migliorandone significativamente le prestazioni. L'elevata potenza di calcolo richiesta ci ha reso impossibile portare a termine tutti i test necessari per poter confermare l'efficacia di AutoLoss-Zero. Per questo motivo non possiamo affermare con certezza che la sua applicazione migliori le prestazioni del progetto. Nonostante ciò, abbiamo visto alcuni dei risultati avuti da [1] su differenti tipologie di dataset e differenti reti neurali. Dimostrando quindi la sua capacità di adattamento ad utilizzi generici.
Confidiamo in sviluppi futuri molto interessanti riguardo alla progettazione automatica delle funzioni di perdita per compiti generici, e al campo dell'Auto Machine Learning.

Bibliografia

[1] Hao Li, Tianwen Fu, Jifeng Dai, Hongsheng Li, Gao Huang, and Xizhou Zhu. *Autoloss-zero: Searching loss functions from scratch for generic tasks.* ArXiv, abs/2103.14026, 2021.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep residual learning for image recognition.* CoRR, abs/1512.03385, 2015.

[3] Alessandra Lumini, Loris Nanni, and Gianluca Maguolo. *Deep ensembles based on stochastic activation selection for polyp segmentation.* CoRR, abs/2104.00850, 2021.

[4] Debesh Jha, P. H. Smedsrud, M. Riegler, P. Halvorsen, T. Lange, Dag Johansen, and Haavard D. Johansen. *Kvasir-seg: A segmented polyp dataset.* ArXiv, abs/1911.07069, 2020.

[5] Hao Li, Chenxin Tao, Xizhou Zhu, Xiaogang Wang, Gao Huang, and Jifeng Dai. *Auto seg-loss: Searching metric surrogates for semantic segmentation.* CoRR, abs/2010.07930, 2020.

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. *Rethinking atrous convolution for semantic image segmentation.* CoRR, abs/1706.05587, 2017.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-net: Convolutional networks for biomedical image segmentation.* CoRR, abs/1505.04597, 2015.

[8] Esteban Real, Chen Liang, David So, and Quoc Le. *Automl-zero: Evolving machine learning algorithms from scratch.* In ICML, 2020.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.*
TPAMI, 2017.

[10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. *Encoder-decoder with atrous separable convolution for semantic image segmentation.*
CoRR, abs/1802.02611, 2018.

[11] M. Everingham, L. Gool, Christopher K. I. Williams, J. Winn, and Andrew Zisserman. *The pascal visual object classes (voc) challenge.*
International Journal of Computer Vision, 88:303–338, 2009.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.*Deep Residual Learning for Image Recognition*
arXiv, abs/1512.03385v1, 2015

[13] Joos Korstanje *What is the difference between Object Detection and Image Segmentation?* `https://towardsdatascience.com/what-is-the-difference-between-object-detection-and-image-\segmentation-ee746a935cc1`
Jul 29, 2020

[14] Jeremy Jordan *An overview of semantic image segmentation.* `https://www.jeremyjordan.me/semantic-segmentation/`
May 21, 2018

[15] Knowledge Center *Mean squared error.* `https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/loss-functions/mean-squared-error`

[16] Wikipedia *Mean squared error* `https://en.wikipedia.org/wiki/Mean_squared_error`
Aug 22, 2022

[17] Wikipedia *Meta learning* `https://en.wikipedia.org/wiki/Meta_learning_(computer_science)`
May 20, 2022

[18] Wikipedia *Automated machine learning* https://en.wikipedia.org/wiki/Automated_machine_learning
Sep 2, 2022

E in fine. Le ultime parole di questa tesi le dedico a me.
Grazie per esserti rialzato ad ogni fallimento.
Grazie per non aver mai mollato.
Grazie per tutte le pacche sulle spalle.
Anche quando sembrava fosse arrivato il momento di dire basta, tu non ti sei mai arreso.
Sono sicuro che questo sarà il primo di una lunga serie di traguardi che ti sei prefissato. Persevera in ogni tua singola idea, in ogni tua singola ambizione, in ogni tuo singolo obbiettivo e vedrai, che alla fine, la vista ti ripagherà di tutto.

Memento audere semper.