UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

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

Elaborato finale in **Elaborazione dei Segnali Multimediali**

Interpretabilità delle reti neurali profonde, tecnica CAM

Anno Accademico 2019/20

Candidato:

Michele Maresca

matr. N46003789

Alla mia famiglia.

Indice

Abstract

Per poter correttamente parlare dell'interpretabilità delle reti neurali credo sia necessario spiegare prima cosa sono e vederne gli ambiti di utilizzo, questo è stato fatto nel primo capitolo dove si è approfondito la loro struttura, partendo dai neuroni, esaminando le varie architetture e soffermandoci su quella di nostro interesse.
Nel secondo capitolo ho illustrato il loro funzionamento, partendo dai concetti base dell'apprendimento, ed esaminando alcuni algoritmi.
Nel terzo, partendo dall'esigenza di reti più performanti, ho introdotto le reti neurali profonde che meglio si adattano alle moderne tecniche dell'intelligenza artificiale. Ho poi illustrato il problema che queste reti hanno fortemente evidenziato con il loro funzionamento in stile black-box. Questo mi ha permesso di introdurre il nocciolo di questo elaborato che è la interpretabilità delle reti neurali, vale a dire, scoprire cosa si nascondesse all'interno di queste scatole nere, ovvero, interpretare come gli algoritmi classificano ed individuano correttamente i problemi in esame.
Nel quarto capitolo ho illustrato la CAM, una ottima tecnica di interpretabilità introdotta da
Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva e Antonio Torralba in [Zhou et al., 2016], essa ha permesso di interpretare in maniera corretta le decisioni prese dalle reti neurali permettendo a sistemi basati su di esse, di essere più robusti e più diffusi nelle applicazioni che hanno un forte impatto sugli individui in termini finanziari, di sicurezza, o di salute. In seguito, ho effettuato un'implementazione della tecnica CAM in Python, verificando praticamente il risultato ottenuto.
Nell'ultimo capitolo, infine, ho esaminato attraverso degli esperimenti i vantaggi introdotti da questa tecnica alle reti neurali nella classificazione e nella localizzazione delle immagini.

Introduzione

Le maggiori capacità dei moderni sistemi di elaborazione, la grande quantità di dati facilmente disponibili hanno fatto si che l'intelligenza artificiale si sia potuta sviluppare in maniera esponenziale. Grande beneficio ne ha tratto il cosiddetto "Machine Learning" e di conseguenza il "Deep Learning". È utile introdurre, in modo chiaro, questi concetti base sui quali spesso si fa confusione.
Il Machine Learning insegna al computer come fare determinate azioni in modo "umano", imparando dall'esperienza.
In pratica, al programma vengono forniti solo dei set di dati che vengono elaborati attraverso algoritmi, sviluppando una propria logica per svolgere la funzione, l'attività, il compito richiesto (ad esempio imparano a riconoscere un'immagine).
Una prima distinzione la si può fare tra apprendimento supervisionato e non supervisionato.
Nel primo caso diamo al computer dei dati e le informazioni relative ai risultati desiderati, cosi da forzare la macchina a trovare la relativa correlazione tra le due cose, imparando cosi la regola che le consentirà di risolvere successivamente problemi simili.
Nel secondo caso, diamo solo un set di dati senza specificare il risultato desiderato.
Si forza cosi la macchina a scoprire schemi e modelli nascosti, partendo solo dall'input.
Vi è inoltre, il caso dell'apprendimento con rinforzo basato sui feedback.
Il Deep Learning è un insieme ristretto del Machine Learning.

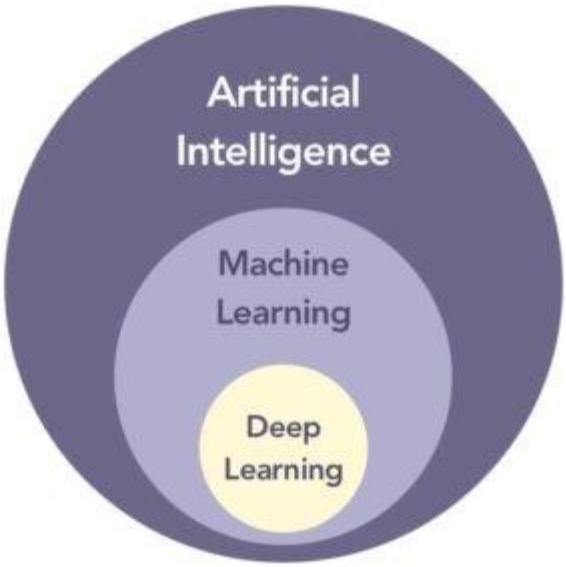


Figura: schema Artificial Intelligence-Machine Learning-Deep Learning

La differenza principale è che nel Machine Learning c'è una fase di estrazione delle features, ovvero le caratteristiche significative dell'immagine, che è "hand-crafted", cioè fatta a mano, e successivamente c'è una fase di classificazione.
Nel caso del Deep Learning, sia la fase di estrazione delle features che la fase di classificazione viene fatta attraverso una rete neurale, in pratica noi chiediamo alla macchina di dirci quali sono le features migliori ai fini della classificazione. Sia l'estrazione delle features che la classificazione sono ottenute dall'addestramento e, in qualche modo, l'algoritmo le apprende dai dati di training.
C'è una differenza enorme di prestazioni tra il Deep Learning e il Machine Learning quando aumenta la quantità di dati a disposizione. In particolare, all'aumentare dei dati il Deep Learning ha prestazioni superiori rispetto alla maggioranza di algoritmi di Machine Learning. Se abbiamo pochi dati le prestazioni sono confrontabili.

Capitolo 1: Reti Neurali

Prima di approfondire il Deep Learning, è utile spiegare nel dettaglio le reti neurali e i neuroni.
Le reti neurali artificiali nascono con lo scopo di riprodurre attività tipiche del cervello umano. Esse rappresentano un nuovo paradigma di programmazione che si oppone alla programmazione tradizionale, dove vengono utilizzate delle istruzioni estremamente precise, il risultato è deterministico, ed è quindi possibile spiegare precisamente cosa succede dall'inizio alla fine dell'esecuzione. Come vedremo in seguito ciò non è automatico.
Una rete neurale è un insieme di unità collegate tra loro, chiamate neuroni.
Le sue proprietà sono determinate dalla topologia e dalle proprietà dei neuroni.

1.1 Neuroni artificiali

Un neurone artificiale di una rete neurale è costituito in genere da molti ingressi ed un'unica uscita. Al singolo ingresso è associato un peso, ovvero, un numero reale che esprime l'importanza del rispettivo input per l'output. L'uscita del neurone è rappresentata da una funzione della somma pesata degli ingressi.
Il primo modello di neurone è quello chiamato di McCulloch e Pitts, esso può essere approssimato come in Figura 1.1.

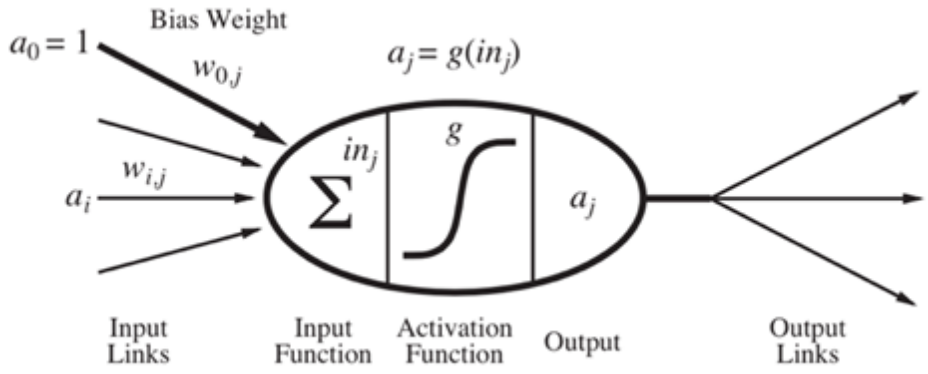


Figura 1.1: modello neurone artificiale di McCulloch e Pitts

È costituito da una cella che riceve in ingresso degli input, i quali possono arrivare dal mondo esterno o da altre celle, e vengono fatti passare per la funzione di attivazione g, il risultato della funzione viene prodotto in uscita dalla cella. L'uscita a_j è data dalla seguente:

$$a_j \leftarrow g(in_j) = g\left(\sum_i \mathrm{w}_{i,j} a_i\right) \qquad (1.1)$$

Nella quale in_j rappresenta la somma degli ingressi, detti anche attivazioni a_i, alla cella, moltiplicati ciascuno per il peso dell'arco corrispondente $\mathrm{w}_{i,j}$.
Nei primi modelli la funzione di attivazione g era approssimata con un gradino (step function), cioè fino a che il valore era sotto una determinata soglia, l'uscita era nulla, nel caso in cui il valore fosse stato maggiore della soglia, veniva prodotto in uscita +1, e l'unità prendeva il nome di Perceptron.

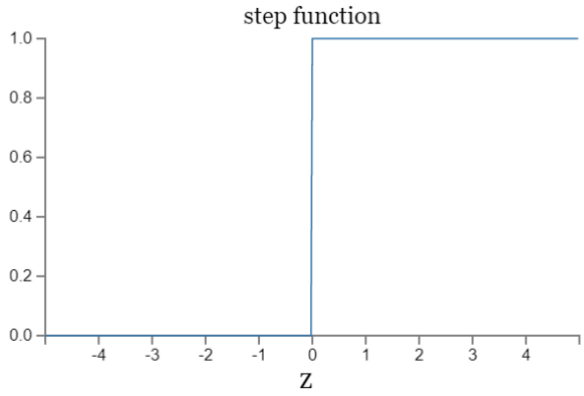


Figura 1.2: grafico funzione gradino

Analiticamente esplicitata dalla seguente espressione:

$$y = \begin{cases} 0 \; se \sum_i w_i x_i \leq threshold \\ 1 \; se \sum_i w_i x_i > threshold \end{cases} \quad (1.2)$$

In particolare, la (1.2) può essere riscritta semplificando il significato di soglia (threshold), spostandola dall'altra parte della disuguaglianza e sostituendola con quella che è nota come "bias" ($b = -threshold$), inoltre, possiamo riscrivere la sommatoria $\sum_i w_i x_i$ come prodotto $w \cdot x$, dove w e x sono vettori i cui componenti sono rispettivamente i pesi e gli input.
Utilizzando la nuova notazione introdotta possiamo riscrivere la (1.2) nel seguente modo:

$$y = \begin{cases} 0 \; se \; w \cdot x + b \leq 0 \\ 1 \; se \; w \cdot x + b > 0 \end{cases} \quad (1.3)$$

Col tempo si è passato alla funzione sigmoidea, esplicitata nella seguente formula:

$$y = \frac{1}{1 + e^{-z}} \quad (1.4)$$

In essa $z \equiv w \cdot x + b$.

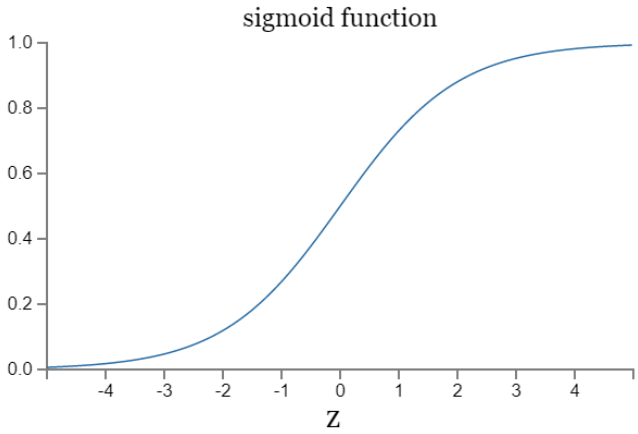


Figura 1.3: grafico funzione sigmoidea

In modo più esplicito, l'output di un neurone sigmoideo con input x_1, x_2, ..., x_n e pesi w_1, w_2, ..., w_n e bias b è:

$$y = \frac{1}{1 + \exp\left(-\sum_i w_i x_i - b\right)} \qquad (1.5)$$

In altre parole, la funzione sigmoidea è una versione più smussata della funzione gradino. La grande differenza tra percettroni e neuroni sigmoidi è che questi ultimi possono avere, come output, qualsiasi numero reale compreso tra 0 e 1.
Altre funzioni di attivazioni possono essere: ReLU (Rectified Linear Unit) e TanH (Tangente Iperbolica).

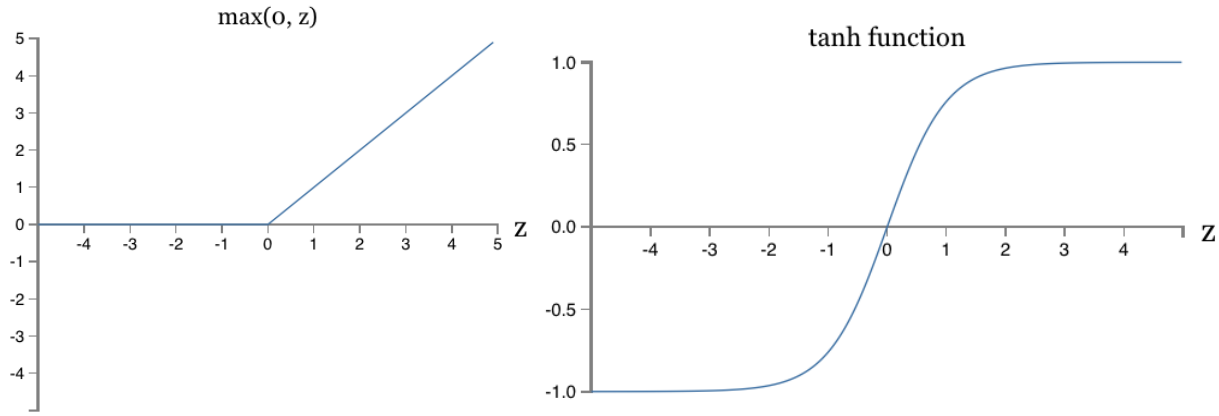


Figura 1.4: (1) grafico funzione ReLU, (2) grafico funzione tangente iperbolica

1.2 Architettura reti neurali

La rete neurale, dunque, è un insieme di neuroni strutturato come in Figura 1.5.

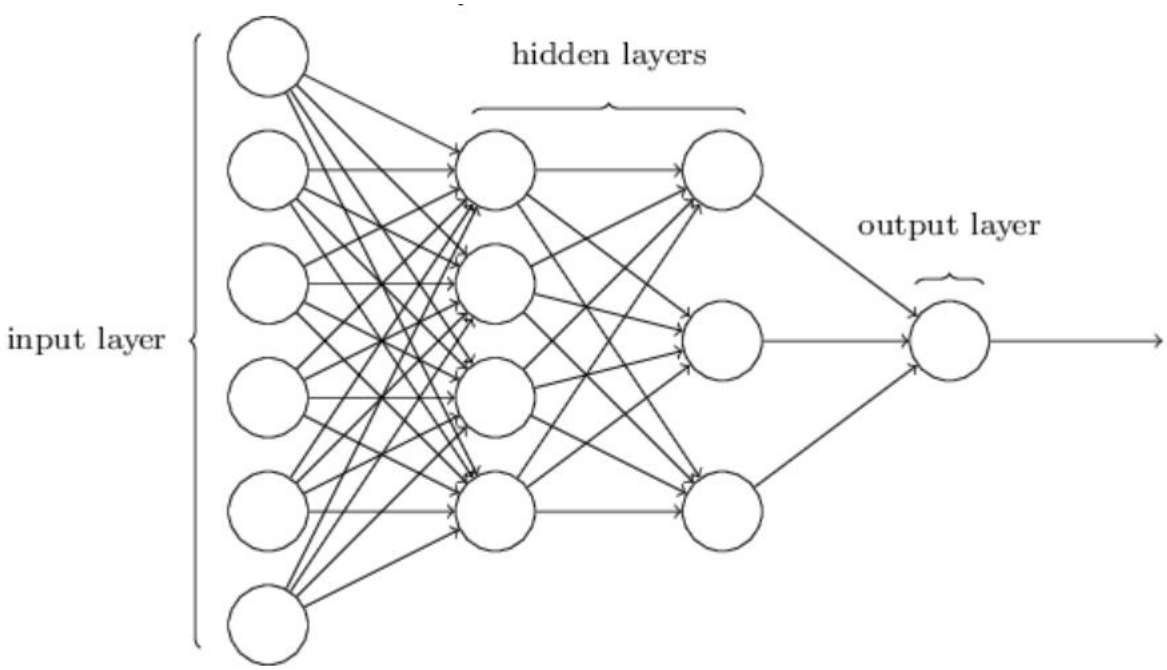


Figura 1.5: esempio di architettura di una rete neurale

Lo strato più a sinistra è chiamato strato di input, i relativi neuroni sono detti di input, quello più a destra è detto di output come il relativo neurone. Lo strato intermedio è chiamato "strato nascosto" (hidden layer).
Una tale rete, detta stratificata o Multilayer Perceptron (MLP), ha i neuroni di uno strato connessi con quelli dello strato successivo, ma non hanno connessioni sullo stesso strato. Nello strato d'ingresso non avviene nessuna elaborazione.
Una rete il cui segnale di ingresso viaggia sempre in avanti, si chiama Feedforward.
Ci sono altri modelli di reti neurali artificiali in cui sono possibili loop di feedback, esse sono reti neurali ricorrenti (RNN, Recurrent Neural Network) sulla quali non ci soffermiamo.

1.3 Convolutional Neural Network (CNN)

Nel campo del riconoscimento delle immagini le reti neurali più utilizzate sono le Reti Neurali Convoluzionali, CNN o Convolutional Neural Network. Queste sfruttano l'operazione matematica detta convoluzione, che lega i vari strati di neuroni di cui sono formate.
Tale convoluzione dipende da due funzioni con argomenti reali: una funzione pesata detta Kernel, che rappresenta un filtro, e una funzione x di input. Dall'applicazione di queste verrà fuori una funzione risultante:

$$s(t) = (x * w)(t) \qquad (1.6)$$

Trattandosi di dati digitali il tempo risulta essere discretizzato per cui la formula (1.6) diventa

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \qquad (1.7)$$

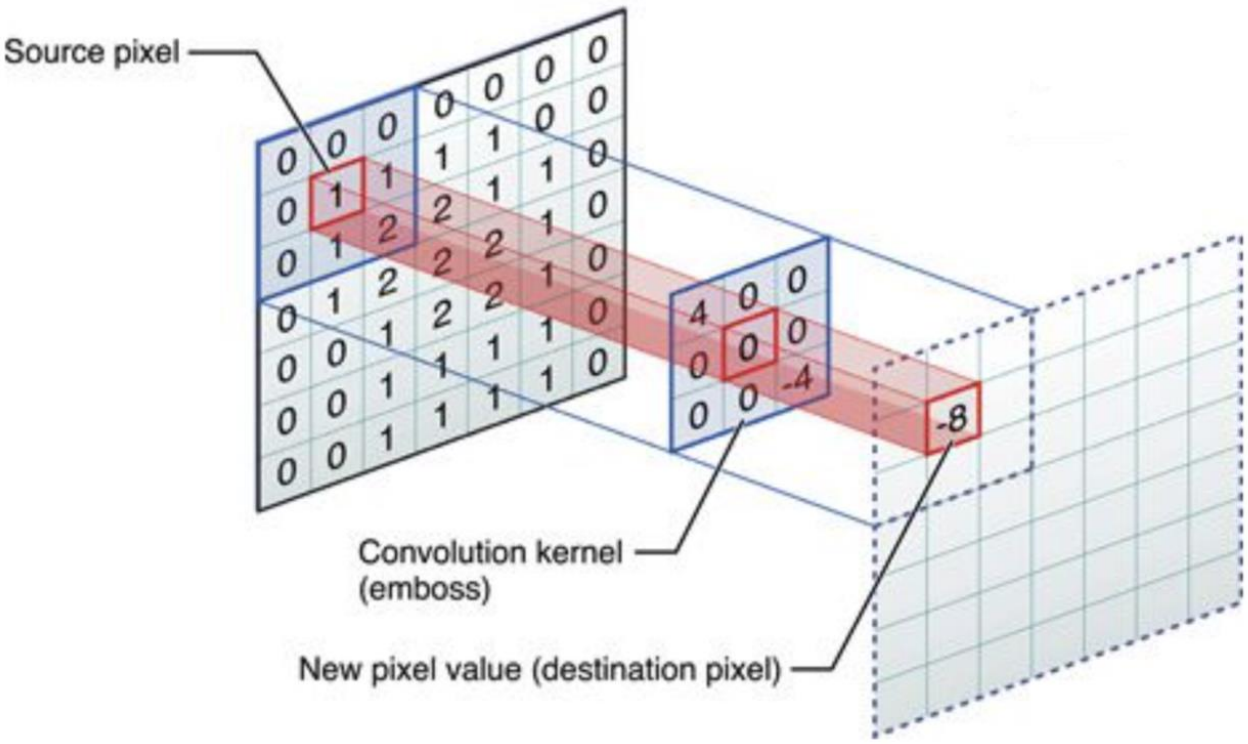


Figura 1.6: schema convoluzione

L'input è un array multidimensionale di dati, mentre il Kernel è un array multidimensionale di parametri.
Nella formula (1.7) la sommatoria infinita deve essere vista come una sommatoria su un numero finito di elementi di un array, su cui applicare la convoluzione.
Le CNN mirano a risolvere il problema delle MLP che non sfruttano la correlazione tra i pixel di un'immagine.
Invece di appiattire l'immagine e fare una semplice moltiplicazione matrice-matrice, impiegano uno o più strati convoluzionali che eseguono ciascuno una convoluzione 2D sull'immagine di input. Un singolo livello di convoluzione è costituito da uno o più filtri che svolgono ciascuno il ruolo di un rilevatore di caratteristiche. Durante l'addestramento, una CNN apprende parametri appropriati per questi filtri.
Le CNN sono progettate per riconoscere dei pattern visivi in modo diretto e non richiedono molto pre-processing o comunque ne richiedono una quantità molto limitata; si ispirano al modello della corteccia visiva animale: i singoli neuroni in questa parte del cervello rispondono solamente a stimoli relativi ad una zona ristretta del campo di osservazione detto campo recettivo.
In Figura 1.7 sono presenti le features visualizzate da una rete neurale convoluzionale, ai primi livelli si estraggono i bordi poi man mano features che sono semanticamente più complesse.

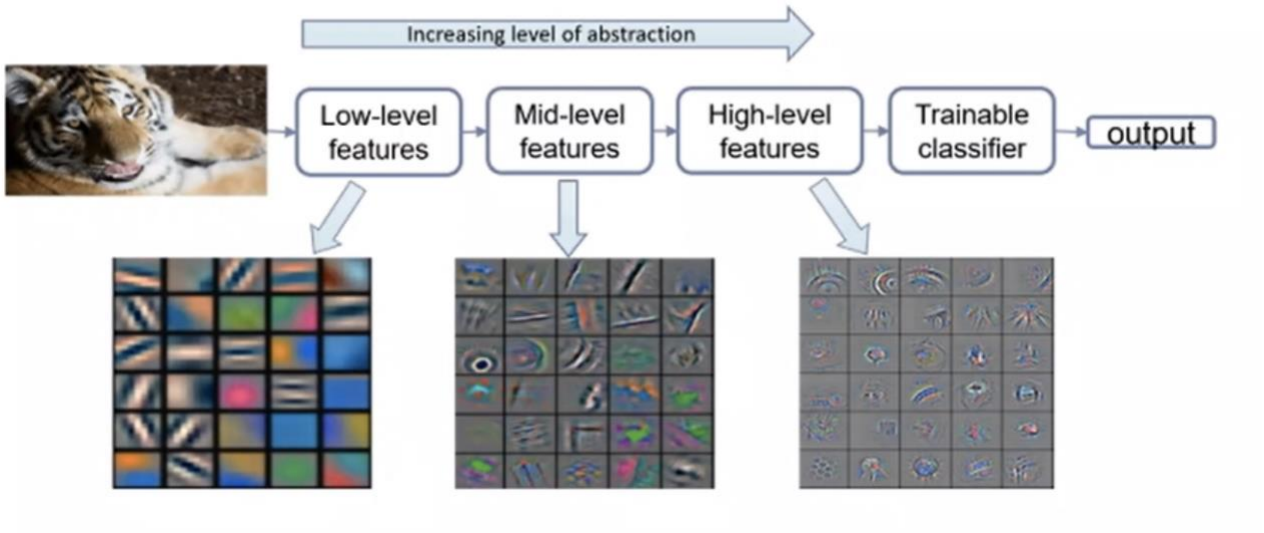


Figura 1.7: esempio di features estratte da diversi livelli di una rete convoluzionale (M. Zeiler, R. Fergus, "Visualizing and understanding convolutional networks", ECCV 2014)

1.4 Struttura delle reti neurali convoluzionali

Le CNN hanno una struttura di base composta da un primo livello chiamato "input layer", questo riceve l'immagine e la ridimensiona prima di passarla ai livelli successivi.
Il secondo livello chiamato "convolutional layer" estrae le features. Questo è il livello principale della rete e può essere presente più volte all'interno della sua architettura. Maggiore è il loro numero, maggiore è la complessità delle caratteristiche che riescono ad individuare. L'output del livello convoluzionale prende il nome di Feature Map, e cambia a seconda del Kernel utilizzato.
I set di features ottenuti, sono solitamente passati allo strato chiamato "pooling layer" che prende ogni mappa delle features e ne genera una versione condensata.
Ad esempio, ogni unità nel layer di pooling può riassumere un'area di 2×2 neuroni nel livello precedente. Una procedura comune per il pooling è nota come Max-pooling. In esso, un'unità di pooling emette semplicemente la massima attivazione dell'area 2×2 di input, come illustrato in Figura 1.8.

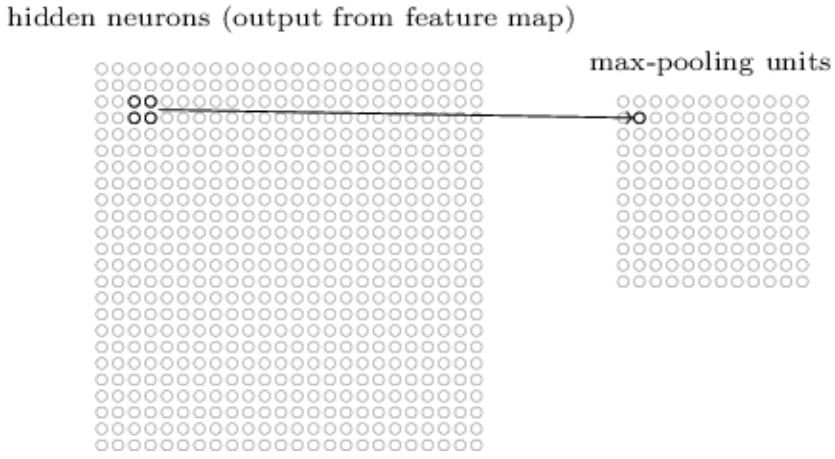


Figura 1.8: rappresentazione funzionamento max-pooling

Il Max-pooling non è l'unica tecnica utilizzata. Altri approcci comuni sono: L2-pooling, nel quale invece di prendere la massima attivazione di una regione 2×2 dei neuroni, prendiamo la radice quadrata della somma dei quadrati delle attivazioni nella regione 2×2 e il Global Average Pooling, il quale riduce ogni Feature Map in un singolo numero che riassume i valori ottenuti, utilizzando una media dei valori della Feature Map del layer precedente.
Il livello di pooling, inoltre, rende l'input delle features più piccolo e gestibile, riducendo il numero di parametri sulla rete e di conseguenza ottimizzando la computazione; rende inoltre la rete invariante alle piccole trasformazioni quali distorsione o traslazione rispetto all'immagine iniziale.
Un altro livello aggiuntivo viene chiamato "Rectified Linear Unit Layer (RELU)", esso sostituisce ogni numero negativo del pooling layer con il valore zero permettendo alla CNN di restare matematicamente stabile. Viene spesso utilizzato per il training perché il calcolo della derivata della funzione di attivazione RELU è veloce ed il valore che questa assume è zero solo se l'ingresso è minore di zero.
Inoltre, può essere presente il "fully connected layer" che implica che ogni neurone del layer precedente è collegato ai neuroni del layer successivo. Questo livello si occupa di prendere le immagini filtrate ad alto livello e tradurre in categorie ciò che ha analizzato, usando le features per classificare le immagini. Solitamente si tratta di un Multilayer Perceptron che usa una funzione di attivazione softmax, analizzata in seguito, questa fa in modo che la somma delle uscite di questo layer sia 1, considerando quindi questi valori come delle probabilità.
La classe relativa all'attivazione con probabilità più alta rappresenta la predizione della rete. La disposizione e la quantità di questi livelli non è rigidamente sequenziale, ma possono essere usate diverse varianti allo scopo di ottenere prestazioni migliori e/o soluzioni che richiedono meno computazione.

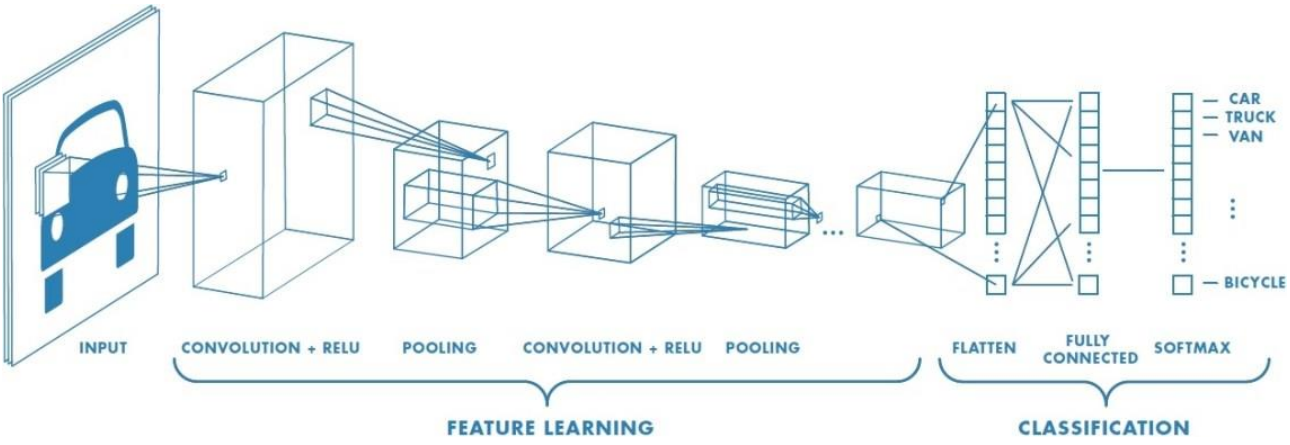


Figura 1.9: tipico schema di CNN

Capitolo 2: Addestramento di una rete neurale

L'addestramento è fondamentale per una rete neurale. In questa fase, come accennato precedentemente, vengono stimati i pesi, con delle tecniche matematiche di ottimizzazione come la gradient descent, e l'Error-Back Propagation.
La prima si basa su due fasi cicliche: propagazione e aggiornamento dei pesi. Nella fase di propagazione (forward step) gli esempi di input, prelevati dal training set, attraversano l'intera rete fino all'uscita, producendo un risultato. Questo viene confrontato, attraverso la "loss-function", con la classificazione corretta relativa all'esempio, la differenza risultante è chiamata errore di predizione. Quest'ultimo viene utilizzato per calcolare il gradiente, il quale viene poi propagato all'indietro nella rete, attraverso l'algoritmo di discesa del gradiente, permettendo di aggiornare i pesi di ciascun neurone. L'addestramento viene svolto mediante cicli con due possibili modalità, quella stocastica e quella di gruppo: nella prima, ogni passo di propagazione in avanti è seguito immediatamente da un passo di aggiornamento, mentre nella seconda viene effettuata la propagazione per tutti gli esempi del training set e dopo viene fatto l'aggiornamento. Questo secondo approccio porta direttamente al risultato finale ma risulta essere complicato o, comunque, computazionalmente oneroso.
Un buon compromesso lo si trova grazie all'uso di mini-batch cioè dei piccoli insiemi casuali di dati sui cui andare ad allenare la rete.

2.1 Apprendimento con gradient descent

Si consideri x input di training e $y=y(x)$ l'output desiderato. L'algoritmo di apprendimento con gradient descent nasce con lo scopo di trovare pesi e bias in modo che l'output dalla rete si avvicini a $y(x)$ per tutti gli input di allenamento x. La funzione di costo considerata è:

$$C(w,b) \equiv \frac{1}{2n}\sum_{x} \| y(x) - a \|^2 \qquad (2.1)$$

In essa w indica l'insieme di tutti i pesi della rete, b l'insieme di tutte le bias, n il numero totale di ingressi di training, a il vettore di output, la somma è fatta su tutti gli input di training della rete. L'output a dipende da x, w e b, ma per mantenere semplice la notazione non è stata indicata esplicitamente questa dipendenza. La notazione $\|v\|$ indica, invece, la funzione di lunghezza per un vettore v. C prende il nome di funzione di costo quadratico, a volte è anche noto come errore quadratico medio o solo MSE (Mean Squared Error).
L'algoritmo di gradient descent si pone come obiettivo di ridurre al minimo il costo $C(w, b)$ in funzione dei pesi e delle bias, poiché $C(w, b) \approx 0$, quando $y(x)$ è approssimativamente uguale all'output a, per tutti gli input di training, x.
Per minimizzare C la si può considerare come una funzione di sole due variabili v_1 e v_2.

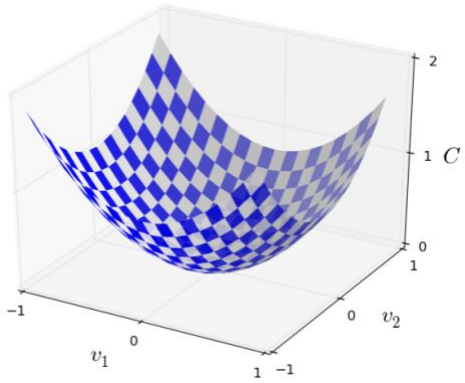


Figura 2.1: generica funzione di due variabili

Quello che si deve trovare è dove *C* raggiunge il suo minimo globale.
Modificando la variabile v_1 di una piccola quantità Δv_1 e v_2 di Δv_2, *C* cambia come nella (2.2).

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \qquad (2.2)$$

Si definisce $\Delta v = (\Delta v1, \Delta v2)^T$ vettore delle variazioni, dove *T* è l'operazione di trasposizione che trasforma vettori riga in vettori colonna.
Si definisce $\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$ gradiente di *C* come il vettore delle derivate parziali, con questa definizione è possibile riscrivere la (2.2):

$$\Delta C \approx \nabla C \cdot \Delta v \quad (2.3)$$

Per ottenere ΔC negativo si sceglie

$$\Delta v = -\eta \nabla C \quad (2.4)$$

dove η è un piccolo parametro positivo, noto come tasso di apprendimento.
Quindi dalla (2.3) si ottiene che

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \parallel \nabla C \parallel^2 \quad (2.5)$$

Poiché $\parallel \nabla C \parallel^2 \geq 0$, ne deriva che $\Delta C \leq 0$, ovvero *C* diminuisce sempre se si cambia *v* in base alla (2.4).
Quindi *v* si modifica in v' secondo la seguente:

$$v \rightarrow v' = v - \eta \nabla C \quad (2.6)$$

Questa regola di aggiornamento viene utilizzata ripetutamente fino a raggiungere un minimo globale.
Riassumendo, l'algoritmo gradient descent consiste nel calcolare ripetutamente il gradiente ∇C utilizzandolo per minimizzare la funzione di costo *C*.
Per farlo funzionare correttamente bisogna scegliere il tasso di apprendimento η abbastanza piccolo da rendere l'equazione (2.3) una buona approssimazione.
Allo stesso tempo, non vogliamo che η sia troppo piccolo, poiché ciò renderà le modifiche Δv minuscole, e quindi l'algoritmo gradient descent funzionerà molto lentamente.

È stato illustrato l'algoritmo per funzioni di due variabili ma può essere generalizzato, senza incorrere in errori, anche quando *C* è una funzione di molte più variabili.
Inoltre, applicando la (2.6) a pesi e bias della rete neurale si ottengono le seguenti:

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \qquad (2.7)$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \qquad (2.8)$$

Si noti che la funzione di costo (2.1) ha la forma C = $\frac{1}{n}\sum_x C_x$, ovvero una media sui costi Cx ≡ $\frac{\|y(x)-a\|^2}{2}$ per singoli esempi di formazione. In pratica, per calcolare il gradiente ∇C bisogna calcolare i gradienti ∇C_x separatamente per ogni input di allenamento, *x*, e quindi mediarli, ∇C = $\frac{1}{n}\sum_x \nabla C_x$. Sfortunatamente, quando il numero di input di allenamento è molto elevato, ciò può richiedere molto tempo e l'apprendimento avviene lentamente.
Una variante dell'algoritmo, chiamata stochastic gradient descent, può essere utilizzata per accelerare l'apprendimento. L'idea è di stimare il gradiente ∇C calcolando ∇C_x per un piccolo campione di input di allenamento scelti casualmente. Facendo una media su questo piccolo campione risulta che è possibile rapidamente ottenere una buona stima del gradiente reale ∇C, e questo aiuta ad accelerare la gradient descent, e quindi l'apprendimento.
Non ci soffermiamo ulteriormente su questa tecnica per non appesantire l'elaborato.

2.2 Algoritmo di Error-Backpropagation

L'algoritmo di Error-Back Propagation viene in aiuto del gradient descent per favorire il calcolo del gradiente, esso si pone l'obiettivo di calcolare le derivate parziali $\frac{\partial C}{\partial w}$ e $\frac{\partial C}{\partial b}$ della funzione di costo *C* rispetto a qualsiasi peso *w* o bias *b* nella rete.
Si consideri la seguente notazione: con $w^l{}_{kl}$ si indica il peso per la connessione dal k^{th} nel $(l-1)^{\text{th}}$ strato al j^{th} neurone nel l^{th} strato; mentre con $\text{b}^{\text{l}}_{\text{j}}$ si indica le bias del j^{th} neurone nello strato l^{th}; infine, con $\text{a}^{\text{l}}_{\text{j}}$ si indica l'attivazione del j^{th} neurone nello strato l^{th}.

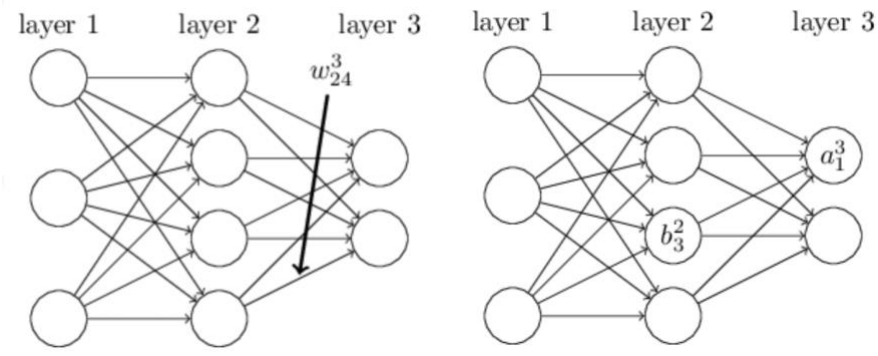


Figura 2.2: (1) peso su una connessione dal quarto neurone nel secondo strato al secondo neurone nel terzo strato di una rete, (2) bias del terzo neurone nel secondo strato e attivazione del primo neurone nel terzo strato

Con queste notazioni, l'attivazione $\text{a}^{\text{l}}_{\text{j}}$ del j^{th} neurone nello strato l^{th} è correlata alle attivazioni dello strato $(l-1)^{\text{th}}$ per l'equazione (2.9)

$$a^l{}_j = \sigma\left(\sum_k w^l{}_{jk} a^{l-1}{}_k + b^l{}_j\right) \qquad (2.9)$$

dove la somma è fatta su tutti i neuroni *k* nello strato $(l-1)^{\text{th}}$. Per riscrivere l'espressione in forma matriciale si definisce la matrice dei pesi w^l, ovvero i pesi che connettono lo strato di neuroni l^{th}, cioè la entry nella riga j^{th} e colonna k^{th} è $w^l{}_{jk}$.
Allo stesso modo, per ogni livello l si definisce un vettore di bias, b^l .
L'equazione (2.9) può essere riscritta, in modo compatto, nella forma vettoriale

$$a^l = \sigma(w^l a^{l-1} + b^l) \qquad (2.10)$$

Questa espressione ci fa capire come le attivazioni in un livello si relazionano con le attivazioni nel livello precedente.
Quando si usa l'equazione (2.10) per calcolare a^l, si calcola la quantità intermedia $z^l \equiv w^l\ a^{l-1}+b^l$, che prende il nome di input pesato per i neuroni nello strato *l*.
La funzione di costo (2.1) viene riscritta come segue:

$$C = \frac{1}{2n}\sum_x \|y(x) - a^L(x)\|^2 \qquad (2.11)$$

Nella quale *L* indica il numero di strati della rete e $a^L = a^L$ (x) è il vettore delle attivazioni emesse dalla rete quando viene immesso *x*.

La (2.11) può essere ancora riscritta come media C = $\frac{1}{n}\sum_x C_x$ rispetto alle funzioni di costo C_x per singoli esempi di addestramento, x. In questa, il costo per singoli esempi di addestramento è $C_x = \frac{1}{2} \| \text{y} - a^L \|^2$. Con questa riscrittura è possibile calcolare la derivata parziale $\frac{\partial C_x}{\partial w}$ e $\frac{\partial C_x}{\partial b}$ per un singolo esempio di addestramento e di conseguenza si ottengono $\frac{\partial C}{\partial w}$ e $\frac{\partial C}{\partial b}$ facendo una media degli esempi di allenamento. Supponendo che l'esempio di addestramento *x* sia stato corretto si può lasciar cadere il pedice *x*, scrivendo il costo C_x come C.

Si definisce, inoltre, $\delta^l{}_j$ l'errore del neurone j nello strato l:

$$\delta^l{}_j \equiv \frac{\partial \text{C}}{\partial \text{z}^l{}_j} \qquad (2.12)$$

La Back-Propagation si basa su quattro equazioni fondamentali che permettono, insieme, di calcolare sia l'errore δ^l che il gradiente della funzione di costo.
La prima equazione considera l'errore nello strato di output:

$$\delta^L{}_j = \frac{\partial \text{C}}{\partial \text{a}^L{}_j}\, \sigma'(z^L{}_j) \quad \text{(BP1)} \qquad (2.13)$$

Nella quale $\frac{\partial \text{C}}{\partial \text{a}^L{}_j}$ misura la velocità con la quale il costo sta cambiando, in funzione all'attivazione dell'uscita j^{th}. Mentre, $\sigma'(z^L{}_j)$ indica la velocità con cui la funzione di attivazione σ sta cambiando in $z^L{}_j$.

La (BP1) (2.13) può esse anche riscritta in forma matriciale

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1a)} \qquad (2.14)$$

Nella quale $\nabla_a C$ è un vettore che ha come componenti le derivate parziali $\frac{\partial C}{\partial a^L{}_j}$.
La seconda equazione considera l'errore in termini dell'errore dello strato successivo, δ^{l+1}:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)} \qquad (2.15)$$

Dove $(w^{l+1})^T$ è la trasposizione della matrice di peso w^{l+1} per il $(l+1)^{th}$ layer.

Si suppone di conoscere l'errore δ^{l+1} al livello $(l+1)^{th}$.
Quando si applica la trasposta alla matrice dei pesi, $(w^{l+1})^T$, intuitivamente, è come spostare l'errore indietro attraverso la rete, dando una sorta di misura dell'errore all'uscita del l^{th} strato. Il prodotto Hadamard $\odot\sigma'(z^l)$ sposta indietro l'errore attraverso la funzione di attivazione nel livello *l*, dando δ^l nell'input pesato nel livello *l*.
Combinando (BP2) (2.15) con (BP1) (2.13) è possibile calcolare l'errore δ^l per qualsiasi livello della rete. Si inizia usando (BP1) (2.13) per calcolare δ^L, quindi si applica l'equazione (BP2) per calcolare δ^{L-1}, quindi di nuovo l'equazione (BP2) (2.15) per calcolare δ^{L-2}, e così via, fino alla fine, scorrendo tutta la rete.
La terza equazione riguarda il tasso di variazione del costo rispetto a eventuali bias della rete:

$$\frac{\partial C}{\partial b^l{}_j} = \delta^l{}_j \qquad \text{(BP3)} \qquad (2.16)$$

Questa indica che l'errore $\delta^L{}_j$ è esattamente uguale alla velocità di variazione $\frac{\partial C}{\partial b^l{}_j}$.
Infine, la quarta equazione riguarda il tasso di variazione del costo rispetto a qualsiasi peso nella rete:

$$\frac{\partial C}{\partial w^l{}_{kj}} = a_k{}^{l-1} \delta^l{}_j \qquad \text{(BP4)} \qquad (2.17)$$

La (BP4) (2.17) permette di calcolare le derivate parziali $\frac{\partial C}{\partial w^l{}_{kj}}$ in termini di quantità δ^l e a^{l-1}.
Le quattro equazioni di back-propagation scritte sotto forma di algoritmo risultano:

1. Input x: impostare l'attivazione a^1 corrispondente per il livello di input.
2. Feedforward: per ogni *l* = 2, 3, …, L calcolare $z^l \equiv w^l a^{l-1} + b^l$ e $a^l = \sigma(z^l)$.
3. Output error δ^L: calcolare il vettore $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. Back-propagate the error: per ogni *l* = L-1, L-2, …, 2 calcolare $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. Output: il gradiente della funzione di costo è dato da $\frac{\partial C}{\partial w^l{}_{kj}} = a_k{}^{l-1} \delta^l{}_j$ e $\frac{\partial C}{\partial b^l{}_j} = \delta^l{}_j$.

Capitolo 3: Interpretabilità

Nonostante le notevoli capacità prestazionali e la loro diffusione capillare, i modelli di Machine Learning più complessi rimanevano per lo più delle black-box, ossia delle scatole nere, impossibili da scrutare. Come affermato chiaramente in [Pedreschi et al., 2018], essi "mappano degli utenti su delle classi, basandosi su delle features attribuibili agli stessi, senza spiegare il perché della scelta effettuata". Questo aspetto è decisamente preoccupante e va studiato correttamente.
Per meglio comprendere questa esigenza è utile porsi una domanda: "Se fossi ritenuto responsabile per la decisione presa da un macchina, in un contesto che abbia un forte impatto su un individuo in termini finanziari, di sicurezza, o di salute, ti fideresti ciecamente delle decisioni di tale macchina?" [Doran et al., 2017].
Ad esempio, si consideri un sistema di riconoscimento delle immagini addestrato per rilevare un tumore. Anche se questo funzionasse molto bene in termini di accuratezza sia sulla convalida che sul set di test, quando vengono presentati i risultati ai clienti e questi si chiedono da quali parti dell'immagine il modello sta imparando o quale sia la causa principale di questo output, sarebbe necessario avere una risposta precisa altrimenti essi non accetteranno mai il sistema, poiché è in gioco la vita umana. [Mishra 2019]
Anche in [Doran et al., 2017] viene sottolineato questo problema:

"È difficile immaginare una persona che si sentirebbe a proprio agio nel concordare ciecamente con la decisione di un sistema in situazioni essenziali ed etiche senza una profonda comprensione della logica decisionale del sistema".
Un altro esempio potrebbe essere l'applicazione delle reti neurali profonde in ambito forense, multimedia forensics, ad esempio capire se un filmato è originale o contraffatto è un'azione importante e diffusa oggi, ma se non si riesce a dare una motivazione del risultato ottenuto dalla rete questo non assume valore di prova in tribunale.
Un ulteriore esempio è la possibilità che la rete, inavvertitamente, prenda decisioni sbagliate, condizionata da artefatti o correlazioni spurie nei dati di addestramento. Come ad esempio riconoscere un oggetto in un'immagine dalle proprietà di uno sfondo o dall'illuminazione, a causa di una sistematica distorsione nella raccolta dei dati [Pedreschi 2017]
Già dai primi studi di interpretabilità delle reti sono venute fuori delle sorprendenti scoperte su come la rete lavorava. Infatti, nel campo della classificazione delle immagini, dove le reti convoluzionali profonde hanno dimostrato prestazioni predittive molto elevate, i primi metodi di spiegazione hanno permesso di aprire queste "scatole nere" e ottenere informazioni su ciò che i modelli avevano imparato e come arrivavano alle loro previsioni. Ad esempio, la classe manubrio veniva riconosciuta correttamente solo se era sistematicamente dotato di un braccio collegato, dimostrando che i neuroni in uscita non si attivavano solo per l'oggetto di interesse, ma anche per le features correlate. [Samek 2003]

Figura 3.1: manubrio con braccio Inceptionism: Going Deeper into Neural Networks Wednesday, June 17, 2015. Posted by Alexander Mordvintsev, Software Engineer, Christopher Olah, Software Engineering Intern and Mike Tyka, Software Engineer

Un'altra scoperta sorprendente fu fatta su una rete che partecipava ad una sfida di classificazione degli oggetti visivi PASCAL (VOC), in questo caso le tecniche di interpretabilità permisero di individuare quali pixel venivano utilizzati per la previsione, e si scoprì che in realtà essa basava, in parte, le sue elevate prestazioni su artefatti. Nel riconoscere la classe "cavallo" essa faceva affidamento su un tag di copyright presente nell'angolo in basso a sinistra presente su molte immagini di cavalli del dataset, piuttosto che rilevare il cavallo effettivo nell'immagine. [Samek 2003]

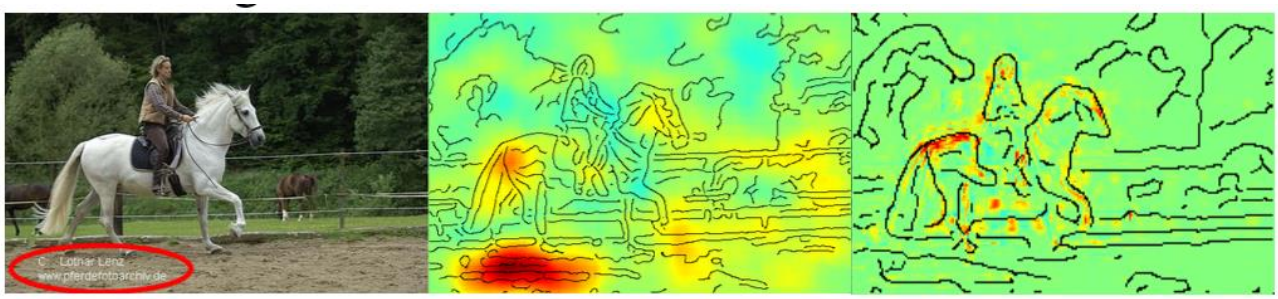


Figura 3.2: S. Lapuschkin, A. Binder, G. Montavon, K.-R. Mu ̈ller, and W. Samek. Analyzing classifiers: Fisher vectors and deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2912–2920, 2016

Altri simili esempi furono segnalati per altre classi e set di dati, in particolare, si dimostrò che per distinguere tra la classe "husky" e "wolf" veniva preso in considerazione la presenza o l'assenza della neve sullo sfondo. [Samek 2003]

3.1 Concetti base dell'apprendimento automatico interpretabile nelle reti neurali

Consideriamo una classe generale di modelli di apprendimento automatico, presumiamo che il modello sia stato completamente sottoposto a training e che il suo comportamento di stima sia descritto da una funzione f: $R^d \rightarrow R$. Essa riceve come input un vettore di feature reali $x = (x_1, x_2, \ldots, x_\mathrm{d})$ e produce come output un punteggio con valore reale su cui si basa la decisione. I risultati della classificazione vengono quindi ottenuti verificando se l'output è superiore a una determinata soglia o superiore all'output di altre funzioni che rappresentano le classi rimanenti, sostanzialmente, l'output della funzione può essere interpretata come la quantità di prove per o contro la scelta a favore di una certa classe. [Samek 2003]
Per far luce sulle previsioni di apprendimento automatico ci si serve di svariati approcci, alcuni come l'"activation maximization" mirano ad un'interpretazione **globale** del modello identificando i casi prototipi per la quantità di uscita $x^* = arg\,max_x\, f(x)$ e consentendo in linea di principio di verificare che la funzione abbia un valore elevato solo per i casi validi. Questo approccio, pur essendo utile per la convalida del modello, è di scarsa utilità per comprendere in un dato esempio *x* quali features giocano a favore o contro l'output $f(x)$ del modello.
Volendo risolvere tale problema abbiamo bisogno di un'analisi **locale** della funzione decisionale.
Per i modelli semplici con una non linearità limitata, la funzione decisionale può essere approssimata localmente come funzione lineare:

$$f(x) \approx \sum_{i=1}^{d} \left[\nabla f(\tilde{x})\right]_i \cdot (x_i - \tilde{x_i}) \qquad (3.1)$$

dove $\tilde{x}$ è un punto radice. Questa espansione assume la forma di una somma pesata sulle features di input, dove l'addendo $R_i = \left[\nabla f(\tilde{x})\right]_i \cdot (x_i - \tilde{x_i})$ può essere interpretato come il contributo della

feature i alla previsione. In particolare, osservando meglio R_i, si nota che ad una feature x_i sarà attribuita forte rilevanza se vengono soddisfatte le seguenti due condizioni:
- la feature deve essere espressa nei dati, in altre parole deve differire dal valore di riferimento $\tilde{x_i}$;
- l'output del modello deve essere sensibile alla presenza di tale feature, cioè $\left[\nabla f\left(\tilde{x}\right)\right]_i \neq 0$.

Una spiegazione per la previsione può quindi essere formata dal vettore dei punteggi di pertinenza $(R_i)_i$. Può essere somministrato all'utente come istogramma sulle feature di input o come mappa termica. È importante sapere che quando si utilizzano queste tecniche di spiegazione, si presuppone implicitamente che tali features di input siano interpretabili per il ricevitore.

3.2 Deep Neural Network (DNN) e loro problematiche

I modelli lineari o le reti neurali poco profonde potrebbero non essere sufficienti nel prevedere in modo corretto. Da qui nasce l'esigenza di reti neurali profonde, o DNN, per avere modelli più predittivi, esse possono essere astratte come una sequenza di livelli

$$f(x) = f_L \circ \ldots \circ f_1(x) \qquad (3.2)$$

dove ogni livello applica una trasformazione lineare seguita da una non linearità per elemento.
Un numero elevato di questi livelli conferisce al modello un'elevata potenza di stima.
Tuttavia, essi diventano anche molto più complessi e non lineari, e le variabili che entrano nel modello di spiegazione semplice della (3.1) diventano considerevolmente più difficili da calcolare e stimare in modo affidabile.
Una prima difficoltà deriva dalla natura multi-scala e distribuita delle rappresentazioni della rete neurale. Alcuni neuroni sono attivati solo per pochi data point, mentre altri si applicano più globalmente. La previsione è quindi una somma di effetti locali e globali, che rende difficile (o impossibile) trovare un punto radice $\tilde{x}$ che si estende linearmente alla previsione per il data point di interesse. La transizione dall'effetto globale a quello locale introduce infatti una non linearità, che la (3.1) non può catturare.
Una seconda fonte di instabilità deriva dall'alta profondità delle recenti reti neurali, dove è stato osservato un effetto "shattered gradient", notando che il gradiente assomiglia localmente al rumore bianco. In particolare, si può dimostrare che per le reti di rettifica profonda, il numero di discontinuità del gradiente può crescere nel peggiore dei casi in modo esponenziale con la profondità.
Un'ultima difficoltà deriva dalla sfida di cercare un punto radice $\tilde{x}$ su cui basare la spiegazione, esso deve essere vicino ai dati ma non deve essere un esempio contradditorio, cioè tale punto, non apportando differenza visiva per i dati originali, non deve far cambiare in maniera drastica l'output della funzione.
Il problema degli esempi contraddittori può essere spiegato dal rumore del gradiente, che fa sì che il modello 'reagisca in modo eccessivo' a determinate perturbazioni a livello di pixel, e anche dall'alta dimensione dei dati in cui molti piccoli effetti a livello di pixel si accumulano in un grande effetto sull'output del modello.

3.3 Metodi pratici per spiegare le DNN

In questo paragrafo ci si concentra su quattro famiglie di tecniche di spiegazione: Interpretable Local Surrogates, Occlusion Analysis, Gradient-based techniques e Layer-Wise Relevance Propagation.
La prima categoria, Interpretable Local Surrogates, mira a sostituire la funzione decisionale con un modello locale surrogato, strutturato in modo tale che sia auto-esplicativo.

La spiegazione può essere ottenuta definendo prima alcune distribuzioni locali $p_x(\xi)$ intorno al data point *x*, imparando il parametro *v* del modello lineare che meglio corrisponde alla funzione localmente:

$$min_v \int [f(\xi) - v^T \xi]^2 \cdot dp_x(\xi) \qquad (3.3)$$

e poi estraendo dei contributi dalle features locali. Il vantaggio di questa tecnica è dato dal fatto che il modello surrogato appreso può essere basato su un proprio set di features interpretabili, consentendo, in questo modo, di produrre spiegazioni in termini di features che risultano comprensibili per l'uomo.
Strutture interpretabili sono anche contenute in modelli molto più complessi. Ad esempio, l'architettura CAM è formata da una sequenza di livelli convoluzionali seguita da un Global Average Pooling di primo livello che aggrega le features di classe in varie posizioni spaziali. Quelle rilevanti vengono quindi fornite facilmente dalle attivazioni che entrano in questo layer di pooling di primo livello. Alla CAM verrà dedicato un capitolo a parte.
La seconda categoria, Occlusion Analysis è un particolare tipo di analisi della perturbazione.
Consiste nel testare, ripetutamente, l'effetto sull'output della rete neurale, dell'occlusione di patch o singole features nell'immagine di input:

$$R_i = f(x) - f(x \odot (1 - m_i)) \qquad (3.4)$$

nella quale m_i è una funzione indicatore per la patch o la feature da rimuovere e '$\odot$' indica il prodotto per elemento. Attraverso questi punteggi è possibile costruire una mappa termica $(R_i)_i$ che evidenzia posizioni in cui l'occlusione ha causato la diminuzione più forte della funzione.
Gli Integrated Gradients, invece, spiegano integrando il gradiente $\nabla f(x)$ lungo una certa traiettoria nello spazio di ingresso che collega un punto radice $\tilde{x}$ al data point *x*.
Nella forma più semplice, la traiettoria è scelta come segmento [$\tilde{x}$, x] collegando un punto radice ai dati. Gli Integrated Gradients definiscono i punteggi per quanto riguarda le features come:

$$R_i(x) = (x_i - \tilde{x_i}) \cdot \int_0^1 \left[\nabla f(\tilde{x} + t \cdot (x - \tilde{x}))\right]_i dt \qquad (3.5)$$

Si può dimostrare che questi punteggi soddisfano $\sum_i Ri(x) = f(x)$ e quindi costituiscono una spiegazione completa. Se necessario, il metodo può essere facilmente esteso a qualsiasi traiettoria nello spazio di input. Ai fini dell'implementazione, gli Integrated Gradients devono essere discretizzati. In particolare, la traiettoria continua è approssimata da una sequenza di data point $x^1, \ldots, x^N$.
Gli Integrated Gradients vengono implementati secondo il seguente algoritmo:

$R = 0$
$for\ n = 1\ to\ N - 1\ do$
$R = R + \nabla f(x^{(n)}) \odot (x^{(n+1)} - x^{(n)})$
$end\ for$
$return\ R$

(Algoritmo 3.1)

Maggiore è il numero di passaggi di discretizzazione, più l'output si avvicina alla forma integrale, ma la procedura diventa più costosa dal punto di vista computazionale.

Un altro metodo di spiegazione basato sul gradiente è lo SmoothGrad. Il gradiente della funzione viene mediato su un numero elevato di posizioni corrispondenti a piccole perturbazioni casuali del data point originale *x*:

$$\nabla_{smooth} f(x) = E_{\varepsilon \sim N(0,\sigma^2 I)}[\nabla f(x + \varepsilon)] \qquad (3.6)$$

Infine, il Layer-wise Relevance Propagation fa un uso esplicito della struttura a più livelli della rete neurale e opera in modo iterativo per produrre la spiegazione. Inizialmente vengono calcolate le attivazioni ad ogni livello della rete neurale fino a raggiungere il livello di output. Il punteggio di attivazione ottenuto nel livello di output forma la stima. Successivamente viene applicata una propagazione inversa, in cui il punteggio di output viene progressivamente ridistribuito, livello dopo livello, fino a raggiungere le variabili di input. In particolare, tutta la "pertinenza" (relevance) che fluisce in un neurone a un dato livello scorre verso i neuroni dello strato sottostante. A livello generale, la procedura LRP può essere implementata come “forward-backward loopcome” mostrata nell'algoritmo:

$a^{(0)} = x$
$for\ l = 1 \ldots L\ do$
$a^{(l)} = f_l(a^{(l-1)})$
$end\ for$
$R^{(L)} = a^{(L)}$
$for\ l = L \ldots 1\ do$
$R^{(l-1)} = relprop(a^{(l-1)}, R^{(l)}, f_l)$
$end\ for$
$return\ R^{(0)}$

(Algoritmo 3.2)

La funzione “relprop” esegue la ridistribuzione da un livello a quello sottostante e si basa su "regole di propagazione" che definiscono il criterio di ridistribuzione esatto. La procedura LRP è mostrata graficamente in Figura 3.3.

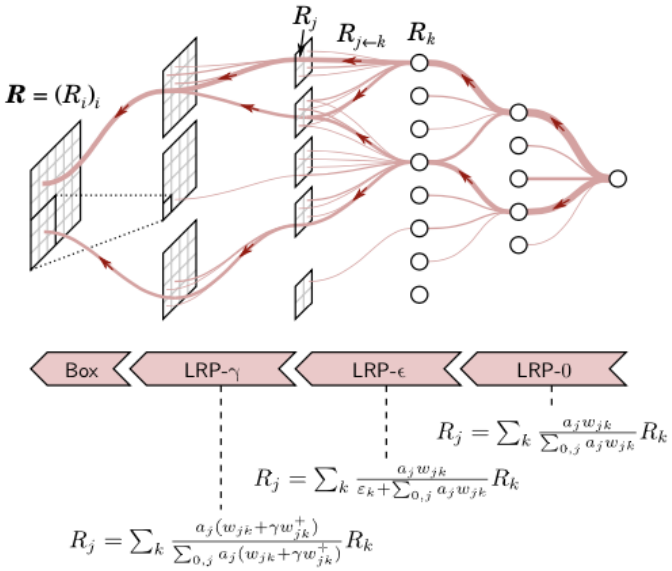


Figura 3.3: Illustrazione della procedura di propagazione LRP applicata a una rete neurale. La previsione in uscita viene propagata all'indietro nella rete, utilizzando varie regole di propagazione, fino a raggiungere le caratteristiche di input. Il flusso di propagazione è mostrato in rosso. [Samek 2003]

Capitolo 4: Class Activation Map (CAM)

La CAM è stata introdotta nel documento “Learning Deep Features for Discriminative Localization”. In esso viene rivisitato il livello di pooling medio globale, e viene fatta luce su come consente esplicitamente alla rete neurale convoluzionale (CNN) di avere notevole capacità di localizzazione nonostante sia addestrata sulle etichette a livello di immagine.
Il pooling medio globale era stato precedentemente proposto come mezzo per regolarizzare la formazione, in realtà costruisce una rappresentazione localizzabile e generica che evidenzia l'attenzione implicita delle CNN su un'immagine.
In questo capitolo viene descritta la procedura per la generazione di mappe di attivazione delle classi (CAM) utilizzando il pooling medio globale (GAP) nelle CNN. Una mappa di attivazione della classe per una particolare categoria indica le regioni discriminatorie dell’immagine utilizzate dalla CNN per identificare quella categoria (ad esempio, Figura 4.1).

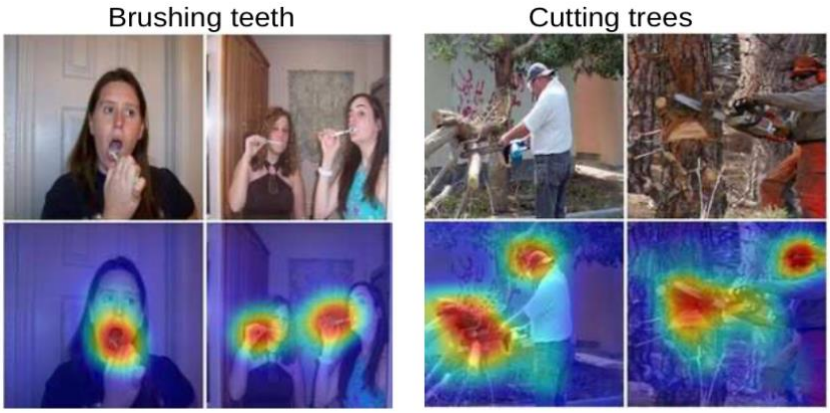


Figura 4.1: Una semplice modifica del livello di pooling medio globale combinato con la tecnica CAM consente alla CNN sia di classificare l'immagine che di localizzarne le regioni specifiche della classe in un unico passaggio in avanti, ad esempio lo spazzolino da denti per lavarsi i denti e la motosega per tagliare gli alberi. [Zhou et al., 2016]

4.1 Architettura della CAM

La rete è costituita in gran parte da livelli convoluzionali, e poco prima del livello di output finale (softmax in caso di categorizzazione), si forma il pooling medio globale sulle mappe di features convoluzionali, queste vengono usate come funzionalità per un livello completamente connesso che produce l'output desiderato (categorico o altrimenti). Data questa semplice struttura di connessioni, è possibile identificare l'importanza delle aree dell'immagine proiettando indietro i pesi del livello di output sulle mappe delle feature convoluzionali, questa tecnica è conosciuta come Class Activation Map.

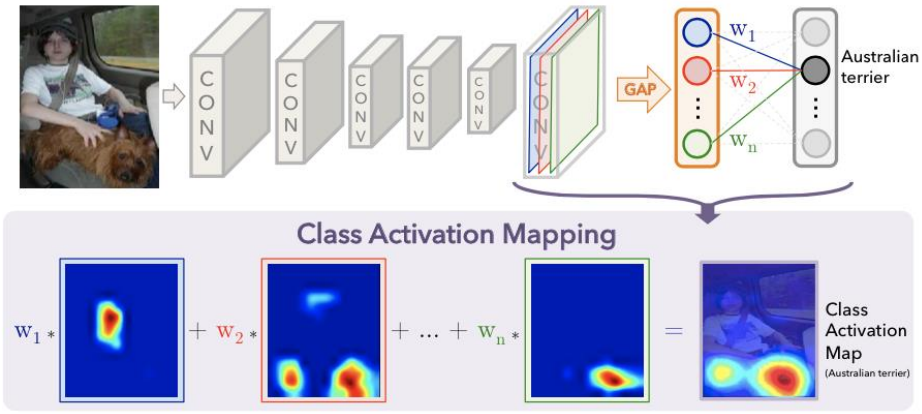


Figure 4.2: procedura per la generazione delle mappe. Il punteggio della classe previsto viene mappato al livello convoluzionale precedente per generare le mappe di attivazione della classe (CAM). La CAM evidenzia le regioni discriminanti specifiche per classe. [Zhou et al., 2016]

Come illustrato nella Figura 4.2, il pooling medio globale restituisce la media spaziale della mappa delle features di ogni unità all'ultimo layer convoluzionale. Una somma ponderata di questi valori viene utilizzata per generare l'output finale. Analogamente, viene calcolata una somma ponderata delle mappe delle features dell'ultimo layer convoluzionale per ottenere le mappe di attivazione della classe.
Lo descriviamo più formalmente di seguito per il caso di softmax.
Per una determinata immagine, $f_k(x, y)$ rappresenta l'attivazione dell'unità k nell'ultimo livello convoluzionale in posizione spaziale (x, y). Quindi, per l'unità k, il risultato dell'esecuzione di un pooling medio globale, F^k è $\sum_{x,y} f_k(x, y)$. Così, per una data classe c, l'input al softmax, S_c, è $\sum_k {w^c}_k F_k$ dove ${w^c}_k$ è il peso corrispondente alla classe c per l'unità k. Essenzialmente, ${w^c}_k$ indica l'importanza di F_k per la classe c. Infine l'output del softmax per la classe c, P_c è dato da $\frac{e^{S_c}}{\sum_c e^{S_c}}$.
È stato ignorato il termine di bias impostando in modo esplicito la bias di input del softmax su 0 in quanto ha poco o nessun impatto sulle prestazioni di classificazione.
Collegando $F_k = \sum_{x,y} f_k(x, y)$ nel punteggio della classe, S_c, si ottiene

$$S_c = \sum_k {w^c}_k \sum_{x,y} f_k(x, y) = \sum_{x,y} \cdot \sum_k {w^c}_k f_k(x, y) \qquad (4.1)$$

Si definisce M_c come mappa di attivazione per la classe c, in cui ogni elemento spaziale è dato da

$$M_c(x, y) = \sum_k {w^c}_k f_k(x, y) \qquad (4.2)$$

Così, $S_c = \sum_{x,y} M_c(x, y)$, e quindi $M_c(x, y)$ indica direttamente l'importanza dell'attivazione alla griglia spaziale *(x, y)* che porta alla classificazione di un'immagine alla classe *c*.
Ci si aspetta che ogni unità venga attivata da un qualche pattern visivo all'interno del suo campo ricettivo. Così f_k è la mappa della presenza di questo pattern visivo. La mappa di attivazione della classe è semplicemente una somma lineare ponderata della presenza di questi pattern visivi in diverse posizioni spaziali. Eseguendo semplicemente l'up-sampling della mappa di attivazione della classe in base alle dimensioni dell'immagine di input, è possibile identificare le aree dell'immagine più rilevanti per la categoria specifica.

In Figura 4.3, vengono mostrati alcuni esempi dell'output CAM utilizzando l'approccio precedente. Si può vedere che le regioni discriminatorie delle immagini per le varie classi sono evidenziate.

Figura 4.3: I CAM di due classi di ILSVRC. Le mappe evidenziano le regioni dell'immagine discriminante utilizzate per la classificazione delle immagini. [Zhou et al., 2016]

In Figura 4.4 vengono evidenziate le differenze nelle CAM per una singola immagine quando si utilizzano classi diverse *c* per generare le mappe. È possibile osservare che le regioni discriminatorie per le diverse categorie sono diverse anche per una stessa immagine. Ciò suggerisce che questo approccio funziona come previsto.

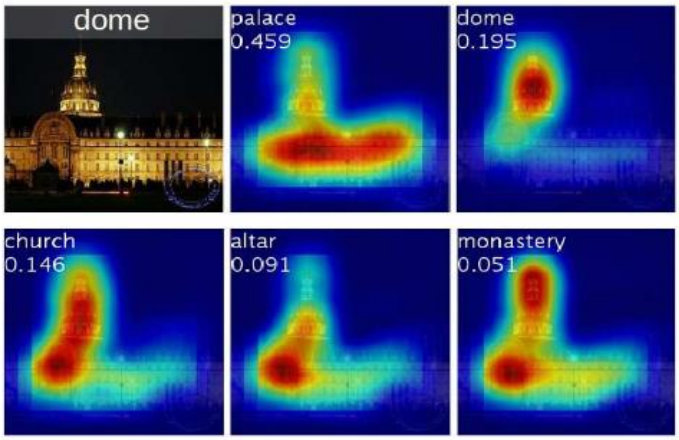


Figura 4.4: esempi delle CAM generate dalle prime 5 categorie previste per l'immagine data con la cupola come grund-truth. La classe prevista e il suo punteggio sono mostrati sopra ogni mappa di attivazione della classe. Le regioni evidenziate variano tra le classi previste, ad esempio la cupola attiva la parte rotonda superiore mentre il palazzo attiva la parte piatta inferiore.

4.2 Global Average Pooling (GAP) vs Global Max Pooling (GMP)

È stato usato il GAP ma poteva essere usato il GMP (utilizzato in [Oquab et al. 2015]). La scelta è stata fatta perché i livelli GAP aiutano ad identificare l'estensione completa dell'oggetto rispetto al livello GMP che identifica solo una parte discriminante, questo perché in GAP prendiamo una media sulle attivazioni che aiuta a trovare tutte le regioni discriminanti, mentre il livello GMP considera solo quella più discriminante.

4.3 Implementazione CAM

Ciò che fondamentalmente fa la CAM è creare una mappa di calore sull'immagine di input, essa è una griglia 2D di punteggi associati ad una specifica classe di output, calcolata per ogni posizione in qualsiasi immagine di input, che indica quanto sia importante ciascuna posizione rispetto alla classe considerata. Molto semplicemente ci dice quali caratteristiche sta cercando il modello.
Analizziamo nel dettaglio l'implementazione fatta per sperimentare il funzionamento della CAM utilizzando Keras come framework di deep learning.
Per prima cosa si importano le librerie

```
from keras.applications.vgg16 import VGG16
import matplotlib.image as mpimg
from keras import backend as K
import matplotlib.pyplot as plt
%matplotlib inline
K.clear_session()
from google.colab import drive
drive.mount('/content/drive')
```

Il modello di rete neurale utilizzato è VGG16 che ha già sostenuto la fase di apprendimento utilizzando il training set di "ImageNet"

```
model = VGG16(weights='imagenet')
```

Si considera l'immagine hummingbird.jpg (Figura 4.5) come esempio

```
img_path = 'G:/hummingbird.jpg'
img=mpimg.imread(img_path)
plt.imshow(img)
```

Figura 4.5: hummingbird.jpg

Si modifica l'immagine per adattarla alle dimensioni di input di VGG

```
from keras.preprocessing import image
img = image.load_img(img_path, target_size=(224,224))
```

La si converte in un array numpy

```
x = image.img_to_array(img)
```

Si rimodellano i dati in modo che siano in formato "batch" poiché il modello accetta solo input di questo tipo

```
import numpy as np
x = np.expand_dims(x, axis=0)
x.shape
```

Si applica il pre-processing di VGG16 all'immagine

```
from keras.applications.vgg16 import preprocess_input
x = preprocess_input(x)
```

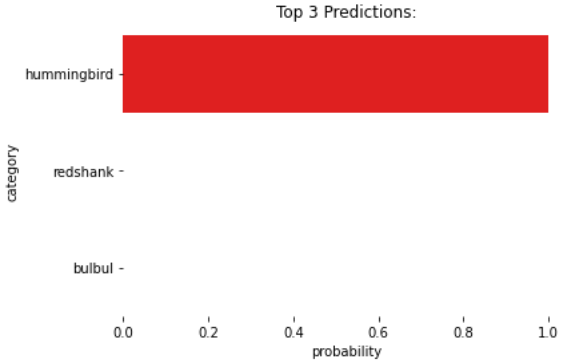
Si esegue la predizione

```
import pandas as pd
from keras.applications.vgg16 import decode_predictions
preds = model.predict(x)
predictions = pd.DataFrame(decode_predictions(preds,
top=3)[0],columns=['col1','category','probability']).iloc[:,1:]
print('PREDICTION:',predictions.loc[0,'category'])
import seaborn as sns
f = sns.barplot(x='probability',y='category',data=predictions,color="red")
sns.set_style(style='white')
f.grid(False)
f.spines["top"].set_visible(False)
f.spines["right"].set_visible(False)
f.spines["bottom"].set_visible(False)
f.spines["left"].set_visible(False)
f.set_title('Top 3 Predictions:')
```

la quale restituisce la seguente uscita:

```
PREDICTION: hummingbird

Text(0.5, 1.0, 'Top 3 Predictions:')
```



Si ottiene l'indice della predizione

```
argmax = np.argmax(preds[0])
output = model.output[:, argmax]
```

Si esplicita l'architettura della rete

```
model.layers_by_depth
```

quest'ultima istruzione restituisce il seguente output:

```
[<tensorflow.python.keras.engine.input_layer.InputLayer at 0x7fd181fd7ef0>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14e04e630>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14e04eb00>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x7fd14d81c5f8>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d81ccf8>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d7b4a20>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x7fd14d7cbeb8>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d7c48d0>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d7d3780>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d7de438>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x7fd14d76c198>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d76c668>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d779198>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d781f28>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x7fd14d78dc88>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d78df28>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d799c88>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fd14d727a58>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x7fd14d7324e0>,
 <tensorflow.python.keras.layers.core.Flatten at 0x7fd14d7329e8>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fd14d73e630>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fd14d73e748>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fd14d73ee48>]
```

Si memorizza in una variabile lo strato convoluzionale finale

```
last_conv_layer = model.get_layer('block5_conv3')
```

Si calcola il gradiente

```
grads = K.gradients(output, last_conv_layer.output)[0]
```

Ogni voce di questo tensore è l'intensità media del gradiente su uno specifico canale della mappa delle caratteristiche

```
pooled_grads = K.mean(grads, axis=(0, 1, 2))
```

Si accede ai valori delle quantità appena definite

```
iterate = K.function([model.input], [pooled_grads,
last_conv_layer.output[0]])
pooled_grads_value, conv_layer_output_value = iterate([x])
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
```

Si traccia la mappa termica (vedi Figura 4.6)

```
heatmap = np.mean(conv_layer_output_value, axis=-1)
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```

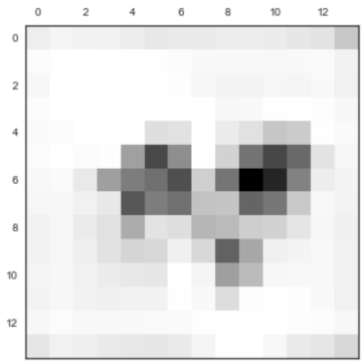


Figura 4.6: mappa termica di hummingbird.jpg

Si carica l'immagine con CV2

```
import cv2
img = cv2.imread(img_path)
```

Si ridimensiona la mappa termica, la si converte in RGB e la si applica all'immagine originale

```
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
```

Si applica il fattore di intensità della mappa termica e si visualizza il risultato (vedi Figura 4.7)

```
hif = .8
superimposed_img = heatmap * hif + img
output = 'G:/output.jpeg'
cv2.imwrite(output, superimposed_img)
img=mpimg.imread(output)
plt.imshow(img)
plt.axis('off')
plt.title(predictions.loc[0,'category'])
```

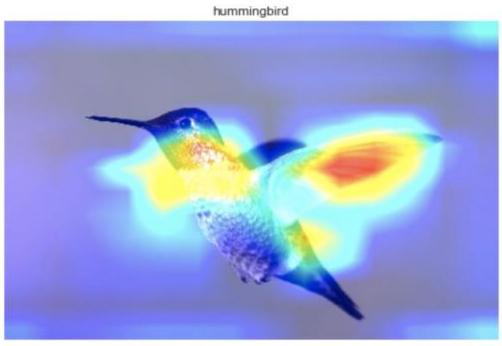


Figura 4.7

Capitolo 5: Esperimenti

In questo capitolo si valuta la capacità di localizzazione della CAM quando la rete è addestrata sul set di dati di benchmark ILSVRC 2014.
Viene descritta innanzitutto l'impostazione sperimentale e le varie CNN utilizzate. Quindi, si verifica che la tecnica non influisca negativamente sulle prestazioni di classificazione e vengono forniti risultati dettagliati sulla localizzazione, scarsamente supervisionata, di oggetti.

5.1 Setup

Attraverso gli esperimenti viene valutato l'effetto dell'utilizzo della CAM sulle seguenti CNN: AlexNet, VG-Gnet e GoogLeNet. In generale, per ciascuna di queste reti vengono rimossi i livelli completamente connessi prima dell'uscita finale e vengono sostituiti con GAP seguiti da uno strato softmax completamente connesso. Si noti che la rimozione dei layer completamente connessi riduce in gran parte i parametri della rete (ad esempio il 90% in meno di parametri per VG-Gnet), ma comporta anche un certo calo delle prestazioni di classificazione.
È stato scoperto che la capacità di localizzazione delle reti è migliorata quando l'ultimo strato convoluzionale, prima di GAP, ha una risoluzione spaziale più alta, che viene chiamata la risoluzione di mappatura. Per fare questo, sono stati rimossi diversi livelli convoluzionali da alcune delle reti. In particolare, sono state apportate le seguenti modifiche:
per AlexNet, sono stati rimossi i livelli dopo conv5 (cioè, pool5 a prob) con conseguente risoluzione di mappatura di 13 x 13.
Per VG-Gnet, sono stati rimossi gli strati dopo conv5-3 (cioè pool5 a prob), ciò ha portato ad una risoluzione di mappatura di 14 x 14.
Per GoogLeNet, sono stati rimossi i livelli dopo inception4e (cioè, pool4 a prob), con conseguente risoluzione di mappatura di 14 x 14.
A ciascuna delle reti di cui sopra, è stato aggiunto uno strato convoluzionale di dimensione 3 x 3, passo 1, con 1024 unità, seguito da uno strato GAP e uno strato softmax. Ognuna di queste reti è stata poi messa a punto sulle 1.3 milioni di immagini di formazione di ILSVRC per la classificazione degli oggetti, ottenendo cosi, le reti AlexNet-GAP, VGGnet-GAP e GoogLeNet-GAP.
Per la classificazione, vengono confrontate con l'originale AlexNet, VG-Gnet e GoogLeNet, e vengono forniti anche risultati per Network in Network (NIN).
Invece, per la localizzazione vengono confrontate con l'originale GoogLeNet, NIN e utilizzando la backpropagation anziché la CAM. Inoltre, per confrontare il pooling medio con il pooling massimo, vengono forniti anche risultati per GoogLeNet addestrato utilizzando il pooling max globale (GoogLeNet-GMP).
Vengono utilizzate le stesse metriche di errore (top-1, top-5) di ILSVRC sia per la classificazione che per la localizzazione. Per la classificazione, la valutazione è fatta sul set di dati di validazione ILSVRC e per la localizzazione si valuta sia la convalida che il set di test.

5.2 Risultati

In primo luogo, vengono riportati i risultati sulla classificazione degli oggetti per dimostrare che questo approccio non danneggia in modo significativo le prestazioni di classificazione. Poi viene dimostrato che questo approccio è efficace nella localizzazione di oggetti debolmente supervisionati.

5.2.1 Classificazione

La Tabella 5.1 riassume le performance di classificazione sia della rete originale che delle reti GAP. Nella maggior parte dei casi c'è un piccolo calo delle prestazioni di 1 - 2% quando si rimuovono

gli strati aggiuntivi dalle varie reti. AlexNet è il più colpito dalla rimozione dei livelli completamente connessi. Per compensare, vengono aggiunti due livelli convoluzionali appena prima di GAP, ottenendo cosi la rete AlexNet*-GAP. Questa esegue in modo comparabile ad AlexNet. Pertanto, nel complesso, le prestazioni di classificazione sono ampiamente preservate per le reti GAP. Inoltre, GoogLeNet-GAP e GoogLeNet-GMP hanno prestazioni simili in termini di classificazione. Si noti che è importante che le reti funzionino bene sulla classificazione, al fine di ottenere prestazioni elevate sulla localizzazione, poiché implica l'identificazione accurata sia della categoria dell'oggetto che della posizione del riquadro di delimitazione.

Networks	top-1 val. error	top-5 val. error
VGGnet-GAP	33.4	12.2
GoogLeNet-GAP	35.0	13.2
AlexNet*-GAP	44.9	20.9
AlexNet-GAP	51.1	26.3
GoogLeNet	31.9	11.3
VGGnet	31.2	11.4
AlexNet	42.6	19.5
NIN	41.9	19.6
GoogLeNet-GMP	35.6	13.9

Tabella 5.1: Errore di classificazione sul set di convalida ILSVRC. [Zhou et al., 2016]

5.2.2 Localizzazione

Per eseguire la localizzazione, è necessario generare un riquadro di delimitazione e la sua categoria associata all'oggetto. Per generare un riquadro di delimitazione dalla CAM, è stata utilizzata una semplice tecnica di soglia per segmentare la mappa termica. Per prima cosa si segmentano le regioni il cui valore è superiore al 20% del valore massimo della CAM. Poi viene preso il riquadro di delimitazione che copre il componente connesso più grande nella mappa di segmentazione. Questa operazione viene eseguita per ciascuna delle prime 5 classi previste per la metrica di valutazione della localizzazione Top-5.
Le prestazioni di localizzazione nel set di convalida ILSVRC vengono visualizzate in Tabella 5.2 e gli output di esempio in Figura 5.1.

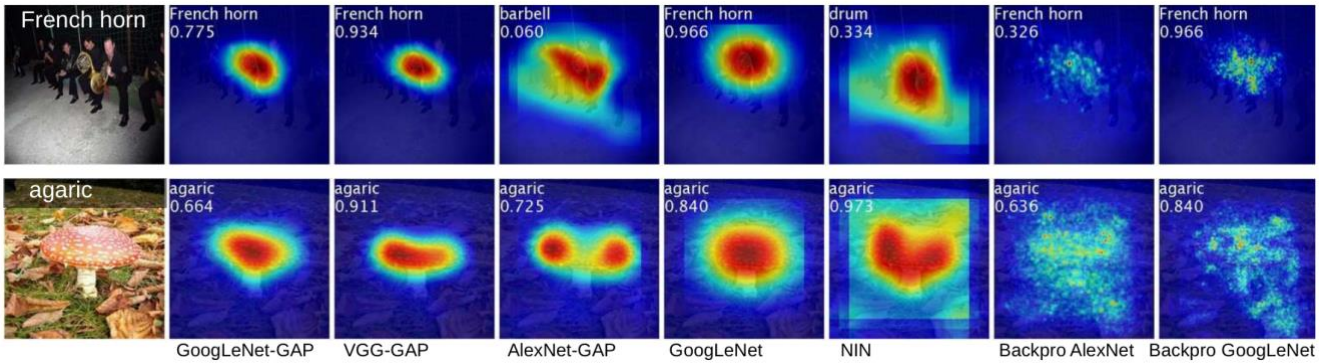


Figura 5.1: Mappe di attivazione di classe da CNN-GAP e mappa di salienza specifica per classe dai metodi di backpropagation. [Zhou et al., 2016]

Method	top-1 val.error	top-5 val. error
GoogLeNet-GAP	**56.40**	**43.00**
VGGnet-GAP	57.20	45.14
GoogLeNet	60.09	49.34
AlexNet*-GAP	63.75	49.53
AlexNet-GAP	67.19	52.16
NIN	65.47	54.19
Backprop on GoogLeNet	61.31	50.55
Backprop on VGGnet	61.12	51.46
Backprop on AlexNet	65.17	52.64
GoogLeNet-GMP	57.78	45.26

Tabella 5.2: Errore di localizzazione sul set di convalida ILSVRC. [Zhou et al., 2016]

In Figura 5.2(1) sono mostrati alcuni esempi di riquadri di delimitazione generati con questa tecnica.

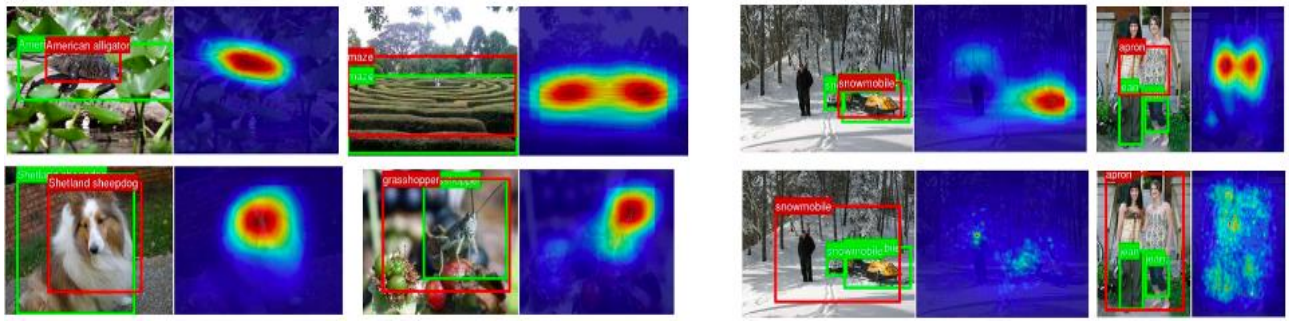


Figura 5.2: (1) Esempi di localizzazione da GoogLeNet-GAP. (2) Confronto tra la localizzazione da GoogLeNet-GAP (due superiori) e la backpropagation utilizzando AlexNet (due inferiori). Le caselle della verità sul terreno sono in verde e le caselle di delimitazione previste dalla mappa di attivazione della classe sono in rosso. [Zhou et al., 2016]

Le reti GAP superano tutti gli approcci di base con GoogLeNet-GAP che raggiunge l'errore di localizzazione più basso del 43% sui Top-5. Ciò è notevole, dato che questa rete non è stata addestrata su un singolo riquadro di delimitazione annotato.
L'approccio CAM supera significativamente l'approccio di retropropagazione (vedi Figura 5.2(2) per un confronto delle uscite). Inoltre, GoogLeNet-GAP supera significativamente GoogLeNet sulla localizzazione, nonostante valga l'inverso per la classificazione. Infine, GoogLeNet-GAP supera GoogLeNet-GMP con un margine ragionevole che illustra l'importanza del pooling medio rispetto al pooling massimo per identificare l'estensione degli oggetti.
Per confrontare ulteriormente questo approccio con i metodi CNN esistenti con supervisione debole e completamente supervisionato, vengono valutate le prestazioni di GoogLeNet-GAP sul set di test ILSVRC. In questo caso viene seguita una strategia di selezione del riquadro di delimitazione leggermente diversa: vengono selezionate due bounding-box (una stretta e una larga) dalla CAM della prima e seconda classe prevista e una bounding-box larga dalla terza classe prevista. Questa euristica è un compromesso tra l'accuratezza della classificazione e quella della localizzazione.

Le performance sono riassunte in Tabella 5.3.

Method	supervision	top-5 test error
GoogLeNet-GAP (heuristics)	weakly	**37.1**
GoogLeNet-GAP	weakly	42.9
Backprop	weakly	46.4
GoogLeNet	full	26.7
OverFeat	full	29.9
AlexNet	full	34.2

Tabella 5.3: Errore di localizzazione sul set di test ILSVRC per vari metodi con supervisione debole e completa. [Zhou et al., 2016]

GoogLeNet-GAP con euristica raggiunge un tasso di errore top-5 del 37,1% in un'impostazione debolmente supervisionata, che è sorprendentemente vicina al tasso di errore top-5 di AlexNet (34,2%) in un ambiente completamente supervisionato. Anche se i risultati sono sorprendentemente positivi, c'è ancora molta strada da fare quando si comparano, per la localizzazione, le reti completamente supervisionate con la stessa architettura (cioè, GoogLeNet-GAP debolmente supervisionato vs GoogLeNet completamente supervisionato).

5.3 Prove pratiche di implementazione classificazione e localizzazione

In questo paragrafo si illustrano dei semplici esperimenti fatti, personalmente, per testare le nozioni fin qui spiegate.
Si confrontano quattro diverse reti neurali già addestrate presenti all'interno del framework Keras: INCEPTIONV3, RESNET101, XCEPTION e VGG16. Le prime tre possiedono lo strato GAP prima dell'ultimo livello completamente connesso, mentre l'ultima rete utilizza il "max-pooling" al posto dell'"average pooling". Si valutano le reti in termini di prestazioni sulla classificazione, effettuata sul set di dati validation di ILSVRC 2014, per mettere in evidenza la differenza tra pooling medio e pooling massimo e successivamente si presentano degli esempi di immagini sulle quali, per confrontare le prestazioni di localizzazione delle quattro reti, si è calcolato le bounding-box utilizzando una soglia al 20% del valore massimo della CAM dell'immagine.

Il codice utilizzato è il seguente

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import matplotlib.image as mpimg
from keras import backend as K
import skimage.io
import matplotlib.pyplot as plt
import cv2
import glob
from skimage.transform import resize
from keras.applications.xception import Xception,preprocess_input,
decode_predictions
import numpy as np
from tqdm import tqdm
import keras
```

```
from keras.utils import to_categorical
%matplotlib inline
K.clear_session()
model = Xception(weights='imagenet')
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=
keras.optimizers.SGD(lr=0.01),metrics = ['accuracy'])
num_classes=1000
file = open("ILSVRC2014_clsloc_validation_ground_truth.txt","r")
label = file.readlines()
file.close()
label =[int(item) for item in label]
import scipy.io
meta = scipy.io.loadmat("meta_clsloc.mat")
original_idx_to_synset = {}
synset_to_name = {}
for i in range(1000):
    ilsvrc2014_id = int(meta["synsets"][0,i][0][0][0])
    synset = meta["synsets"][0,i][1][0]
    name = meta["synsets"][0,i][2][0]
    original_idx_to_synset[ilsvrc2014_id] = synset
    synset_to_name[synset] = name
synset_to_keras_idx = {}
keras_idx_to_name = {}
f = open("synset_words.txt","r")
idx = 0
for line in f:
    parts = line.split(" ")
    synset_to_keras_idx[parts[0]] = idx
    keras_idx_to_name[idx] = " ".join(parts[1:])
    idx += 1
f.close()
def convert_original_idx_to_keras_idx(idx):
    return synset_to_keras_idx[original_idx_to_synset[idx]]
label = np.array([convert_original_idx_to_keras_idx(idx) for idx in label])
label = to_categorical(label , num_classes)
pbar = tqdm(total=50000)
img_rows=299; img_cols=299; img_channels=3;
x_val = []
y_val = []
test_accuracy_list = []
test_loss_list = []
j=1
i=1
k=0
for filename in sorted(glob.glob("ILSVRC2014_images_val/*.JPEG")):
    img = np.float32(skimage.io.imread(filename))
    img = resize(img, (img_rows, img_cols), order=1)
    if img.shape!=(299,299,3):
      img = np.stack((img,img,img), -1)
    img = preprocess_input(img)
```

```
    x_val.append(img)
    if i==5000:
      x_val = np.reshape(x_val , (5000,img_rows,img_cols,img_channels))
      y_val=label[k:(k+5000)]
      test_loss, test_accuracy=model.evaluate(x_val, y_val, verbose=True)
      test_loss_list.append(test_loss)
      test_accuracy_list.append(test_accuracy)
      print('Test loss:',test_loss)
      print('Test accuracy:',test_accuracy)
      x_val = []
      y_val = []
      i=0
      j=j+1
      k=k+5000
    i=i+1
    pbar.update()
pbar.close()
accuracy_media=sum(test_accuracy_list) / float(len(test_accuracy_list))
print("Accuracy media:",accuracy_media)
```

Si è riportato il codice relativo all'implementazione della rete Xception, ma esso è analogo a quello utilizzato per le altre tre reti.
Si sono valutate le prestazioni della classificazione sulle 50.000 immagini del validation set di ILSVRC 2014 suddividendolo in lotti di 5.000 immagini e facendo, successivamente una media dei vari risultati.
In Tabella 5.4 sono sintetizzate le prestazioni delle quattro reti.

NETWORK	ACCURACY
VGG16	0.66
RESNET101	0.73
XCEPTION	0.78
INCEPTIONV3	0.77

Tabella 5.4

È possibile osservare la differenza di prestazioni tra le reti che usano il pooling medio (INCEPTIONV3, RESNET101, XCEPTION), le quali superano tutte lo 0.7 di accuratezza, rispetto alla rete VGG16, che usa il pooling massimo e ottiene un'accuratezza di 0.66.
È ancora più marcata la differenza tra pooling medio e pooling massimo nella localizzazione ed è possibile osservarlo nella Figura 5.3.

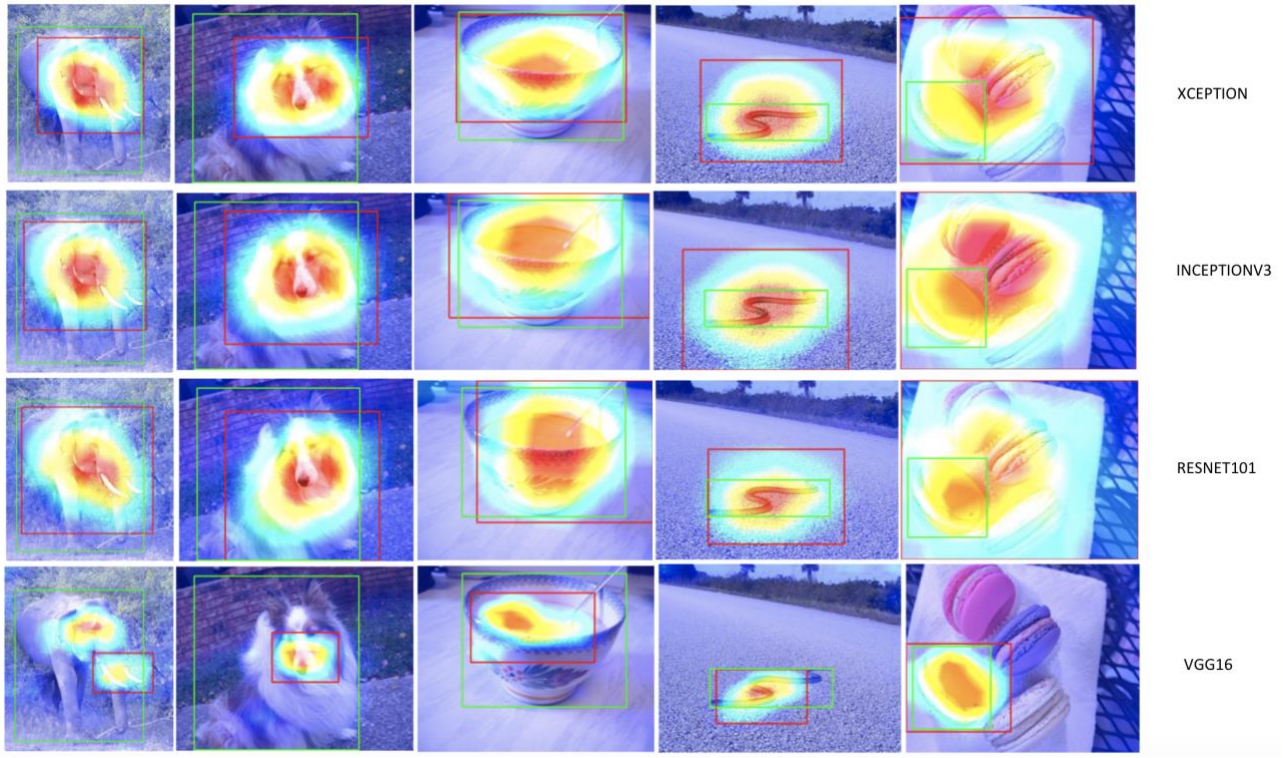


Figura 5.3: (1) Xception, (2) InceptionV3, (3) ResNet101, (4) VGG16.

Le cinque immagini presenti in Figura 5.3 sono state prelevate dal validation set di ILSVRC 2014. La bounding-box verde rappresenta quella corretta relativa all'immagine, mentre la bounding-box rossa è stata calcolata con il metodo di cui sopra, utilizzando la CAM.

È possibile notare che, mentre per quanto riguarda la classificazione la rete VGG16 ha buone prestazioni, nella localizzazione sono presenti degli esempi, come le prime due immagini dove la rete non si "comporta bene" ed ha prestazioni inferiori rispetto alle altre tre. Ciò conferma che i livelli di pooling medio aiutano ad identificare l'estensione completa dell'oggetto rispetto al livello di pooling massimo che identifica solo una parte discriminante.

Il codice utilizzato per generare le bounding-box è il seguente

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
import cv2
from skimage import color
%matplotlib inline
K.clear_session()
from keras.applications.xception import Xception
model = Xception(weights='imagenet')
def cam(img_path,index):
    from keras.applications.xception import Xception
    import matplotlib.image as mpimg
    %matplotlib inline
    K.clear_session()
```

```
    model = Xception(weights='imagenet')
    img=mpimg.imread(img_path)
    plt.imshow(img)
    from keras.preprocessing import image
    img = image.load_img(img_path, target_size=(299, 299))
    x = image.img_to_array(img)
    #import numpy as np
    x = np.expand_dims(x, axis=0)
    from keras.applications.xception import preprocess_input,
decode_predictions
    x = preprocess_input(x)
    preds = model.predict(x)
    import pandas as pd
    predictions = pd.DataFrame(decode_predictions(preds,
top=3)[0],columns=['col1','category','probability']).iloc[:,1:]
    argmax = np.argmax(preds[0])
    output = model.output[:, argmax]
    last_conv_layer = model.get_layer('block14_sepconv2_act')
    grads = K.gradients(output, last_conv_layer.output)[0]
    pooled_grads = K.mean(grads, axis=(0, 1, 2))
    iterate = K.function([model.input], [pooled_grads,
last_conv_layer.output[0]])
    pooled_grads_value, conv_layer_output_value = iterate([x])
    for i in range(512):
        conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
    heatmap = np.mean(conv_layer_output_value, axis=-1)
    heatmap = np.maximum(heatmap, 0)
    heatmap /= np.max(heatmap)
    img = cv2.imread(img_path)
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap = np.uint8(255 * heatmap)
    heatmapN = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    hif = .8
    superimposed_img = heatmapN * hif + img
    if index==67:
      output = 'output67.jpeg'
    if index==3:
      output = 'output3.jpeg'
    if index==4:
      output = 'output4.jpeg'
    if index==6:
      output = 'output6.jpeg'
    if index==8:
      output = 'output8.jpeg'
    cv2.imwrite(output, superimposed_img)
    img=mpimg.imread(output)
    plt.imshow(img)
    plt.axis('off')
    plt.title(predictions.loc[0,'category'].upper())
    return heatmap,img
```

```
heatmap67,img67 = cam('ILSVRC2012_val_00000067.JPEG',67)
max67=heatmap67.max()
threshold=0.2*max67
ret67,img67_thresh= cv2.threshold(heatmap67,threshold,255,cv2.THRESH_BINARY)
contours67,hierarchy67 = cv2.findContours(img67_thresh, 1, 2)
cnt67 = contours67[0]
x67,y67,w67,h67 = cv2.boundingRect(np.float32(cnt67))
plt.figure()
plt.imshow(cv2.rectangle(img67,(x67,y67),(x67+w67,y67+h67),(255,0,0),2))
plt.imshow(cv2.rectangle(img67,(28,473),(377,63),(0,255,0),2))
plt.axis('off')
heatmap3,img3 = cam('ILSVRC2012_val_00000003.JPEG',3)
max3=heatmap3.max()
threshold=0.2*max3
ret3,img3_thresh= cv2.threshold(heatmap3,threshold,255,cv2.THRESH_BINARY)
contours3,hierarchy3 = cv2.findContours(img3_thresh, 1, 2)
cnt3 = contours3[0]
x3,y3,w3,h3 = cv2.boundingRect(np.float32(cnt3))
plt.figure()
plt.imshow(cv2.rectangle(img3,(x3,y3),(x3+w3,y3+h3),(255,0,0),2))
plt.imshow(cv2.rectangle(img3,(38,373),(385,19),(0,255,0),2))
plt.axis('off')
heatmap4,img4 = cam('ILSVRC2012_val_00000004.JPEG',4)
max4=heatmap4.max()
threshold=0.2*max4
ret4,img4_thresh= cv2.threshold(heatmap4,threshold,255,cv2.THRESH_BINARY)
contours4,hierarchy4 = cv2.findContours(img4_thresh, 1, 2)
cnt4 = contours4[0]
x4,y4,w4,h4 = cv2.boundingRect(np.float32(cnt4))
plt.figure()
plt.imshow(cv2.rectangle(img4,(x4,y4),(x4+w4,y4+h4),(255,0,0),2))
plt.imshow(cv2.rectangle(img4,(94,284),(441,15),(0,255,0),2))
plt.axis('off')
heatmap6,img6 = cam('ILSVRC2012_val_00000006.JPEG',6)
max6=heatmap6.max()
threshold=0.2*max6
ret6,img6_thresh= cv2.threshold(heatmap6,threshold,255,cv2.THRESH_BINARY)
contours6,hierarchy6 = cv2.findContours(img6_thresh, 1, 2)
cnt6 = contours6[0]
x6,y6,w6,h6 = cv2.boundingRect(np.float32(cnt6))
plt.figure()
plt.imshow(cv2.rectangle(img6,(x6,y6),(x6+w6,y6+h6),(255,0,0),2))
plt.imshow(cv2.rectangle(img6,(105,279),(358,204),(0,255,0),2))
plt.axis('off')
heatmap8,img8 = cam('ILSVRC2012_val_00000008.JPEG',8)
max8=heatmap8.max()
threshold=0.2*max8
ret8,img8_thresh= cv2.threshold(heatmap8,threshold,255,cv2.THRESH_BINARY)
contours8,hierarchy8 = cv2.findContours(img8_thresh, 1, 2)
```

```
cnt8 = contours8[0]
x8,y8,w8,h8 = cv2.boundingRect(np.float32(cnt8))
plt.figure()
plt.imshow(cv2.rectangle(img8,(x8,y8),(x8+w8,y8+h8),(255,0,0),2))
plt.imshow(cv2.rectangle(img8,(14,328),(181,163),(0,255,0),2))
plt.axis('off')
```

Anche in questo caso si è riportato il codice relativo all'implementazione della rete Xception, ma esso è analogo a quello utilizzato per le altre tre reti.

Conclusioni

Le mappe di attivazione delle classi o CAM sono un ottimo modo per capire cosa vedono i modelli durante l'utilizzo, esse offrono approfondimenti allo sviluppatore e agli eventuali clienti che possono essere cruciali per l'usabilità e il ciclo di vita del modello.
Con l'aiuto della CAM lo sviluppatore può vedere quale regione dell'immagine il modello sta guardando, ed il modello, diventato interpretabile, risulta essere più robusto e utilizzabile in modo corretto nel mondo reale. Attraverso questo approccio i modelli di Machine Learning non risultano più delle black-box, poiché è possibile comprendere le ragioni che hanno portato ad una determinata previsione della rete, ciò in primo luogo permette di rilevare eventuali errori, e in secondo luogo rende il sistema più affidabile. Infatti, potendo spiegare l'output generato dalla rete, è possibile allargare i campi di utilizzo di questi strumenti anche in contesti che hanno un forte impatto su un individuo in termini finanziari, di sicurezza, o di salute.

Bibliografia

[1] Derek Doran, Sarah Schulz, and Tarek R. Besold, “What Does Explainable AI Really Mean? A New Conceptualization of Perspectives”, CoRR, abs/1710.00794 ,2017.

[2] Divyanshu Mishra, “Demystifying Convolutional Neural Networks Using Class Activation Maps”, 2019.

[3] Michael Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015.

[4] Maxime Oquab, Léon Bottou, Ivan Laptev, Josef Sivic, “Is object localization for free? – Weakly-supervised learning with convolutional neural networks”, Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, 685-694.

[5] Dino Pedreschi, Fosca Giannotti, Riccardo Guidotti, Anna Monreale, Luca Pappalardo, Salvatore Ruggieri, Franco Turini, “Open the Black Box Data-Driven Explanation of Black Box Decision Systems”, CoRR, abs/1806.09936, 2018.

[6] Stuart Russell, Peter Norvig, “Artificial Intelligence: a modern approach”, Prentice Hall, 2010.

[7] Wojciech Samek, Member, IEEE, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, and Klaus-Robert Müller, “Toward Interpretable Machine Learning: Transparent Deep Neural Networks and Beyond”, In: arXiv preprint arXiv:2003.07631, 2003.

[8] Matthew D. Zeiler, Rob Fergus, “Visualizing and understanding convolutional networks”, European conference on computer vision (ECCV), Springer International Publishing Switzerland, 2014, 818-833.

[9] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, “Dive into Deep Learning”, 2020.

[10] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba, “Learning Deep Features for Discriminative Localization”, Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, 2921–2929.