



---

UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI FISICA

Corso di Laurea Magistrale in Fisica dei Sistemi Complessi

## Studio sulle reti neurali addestrate su dataset sintetici

Relatori:

Michele Caselle

Antonio Mastropietro

Candidato:

Federico Di Credico

matricola 848610

Anno Accademico 2017-2018

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Neurone artificiale . . . . .	3
1.2	Reti neurali artificiali . . . . .	3
1.3	Addestrare una rete neurale . . . . .	5
1.3.1	Discesa del gradiente . . . . .	5
1.3.2	Discesa stocastica del gradiente (stochastic gradient descent - SGD) . . . . .	6
1.3.3	Algoritmo di ottimizzazione della discesa del gradiente - Momentum . . . . .	6
<b>2</b>	<b>Computer vision</b>	<b>8</b>
2.1	Compiti della visione artificiale (computer vision tasks) . . . . .	8
<b>3</b>	<b>Classificazione e reti neurali convoluzionali - VGG</b>	<b>12</b>
3.1	Architettura . . . . .	12
3.2	Layer di convoluzione . . . . .	13
3.3	Rectified linear unit . . . . .	15
3.4	Max-pooling . . . . .	15
3.5	Fully-connected layer . . . . .	15
3.6	Softmax . . . . .	16
<b>4</b>	<b>Reti di segmentazione - SegNet</b>	<b>17</b>
4.1	Innovazione . . . . .	18
4.2	Architettura . . . . .	19
4.3	Batch normalization layer . . . . .	21

4.3.1	Internal covariate shift . . . . .	21
4.3.2	Algoritmo . . . . .	22
<b>5</b>	<b>Metodi di indagine per le reti neurali profonde</b>	<b>25</b>
5.1	Metodi di interpretazione . . . . .	26
5.1.1	Activation maximization . . . . .	27
5.1.2	DeconvNet . . . . .	28
5.1.3	Guided Backpropagation . . . . .	29
5.2	Metodi di spiegazione . . . . .	30
5.2.1	Sensitivity analysis . . . . .	31
5.2.2	Pixel-wise decomposition . . . . .	32
5.2.3	Simple taylor decomposition . . . . .	33
5.2.4	Layer-wise relevance propagation - LRP . . . . .	34
<b>6</b>	<b>LRP per reti multistrato</b>	<b>40</b>
6.1	Alfa-Beta rule . . . . .	41
6.1.1	Considerazioni sul bias . . . . .	42
<b>7</b>	<b>Grad-CAM</b>	<b>45</b>
7.1	Identificare e rimuovere i bias in un dataset . . . . .	47
7.2	Grad-Cam su Segnet . . . . .	48
7.2.1	Campo recettivo . . . . .	50
<b>8</b>	<b>LRP su SegNet</b>	<b>54</b>
8.1	Aspetti tecnici . . . . .	54
8.2	Inizializzazione dello score . . . . .	55
8.3	Algoritmo . . . . .	57
8.4	heatmap . . . . .	58
<b>9</b>	<b>Esperimenti</b>	<b>61</b>
9.1	Dataset sintetici e reali . . . . .	61
9.2	Dettagli dell'addestramento . . . . .	63
9.3	Visualizzazione delle feature più importanti tramite LRP . . . . .	63
9.4	Somma delle feature . . . . .	64
9.5	Score distribuito sulle singole feature . . . . .	65

9.6	Feature values . . . . .	67
9.7	LRP su un singolo pixel e nuova visualizzazione della heatmap .	69
9.8	Separazione delle feature . . . . .	71
9.9	Confronto tra diverse classi . . . . .	74
9.10	LRP su immagini reali con rete addestrata su dataset sintetico .	76
9.11	Osservazioni sul bias . . . . .	77
<b>10</b>	<b>Conclusioni</b>	<b>81</b>

## **Sommario**

Nel campo della computer vision è ben noto il problema che i dataset creati artificialmente non sono in grado di rappresentare bene la realtà quando si tratta di utilizzarli per l'addestramento di algoritmi di machine learning. In questo lavoro di tesi ci siamo posti l'obiettivo di studiare il fenomeno attraverso le più moderne tecniche di spiegazione per le reti neurali artificiali.

# Capitolo 1

## Introduzione

Algoritmi di machine learning come le reti neurali profonde (deep neural network - DNN) sono diventate uno strumento indispensabile per un'ampia gamma di applicazioni, come ad esempio la classificazione di immagini [1], il riconoscimento vocale [2] o l'elaborazione del linguaggio naturale [3]. Queste tecniche hanno raggiunto un'accuratezza predittiva estremamente alta e in alcuni casi al pari delle capacità umane [4]. Tale accuratezza è ottenibile soltanto con l'utilizzo di una considerevole mole di dati annotati manualmente [5]. L'annotazione manuale è un'operazione che viene compiuta da parte dell'uomo e consiste nell'estrazione e nella registrazione di informazioni che sono implicitamente presenti nei dati, le stesse informazioni che si vuole che l'algoritmo impari ad ottenere [6]. Pensiamo ad esempio alla foto di un gatto. L'immagine è il dato che abbiamo a nostra disposizione: una matrice numerica immagazzinata nella memoria di un computer. L'informazione "gatto" è esplicitamente registrata dall'uomo, mentre la relazione tra il dato e l'informazione è quella che la rete neurale deve imparare ad apprendere. Un insieme di coppie dato-annotazione è definito con il nome di dataset. Annotare i dati manualmente risulta essere molto spesso un'operazione lunga e dispendiosa. E' in risposta a tali problematiche che nasce l'idea di creare dataset sintetici (artificiali) [7].

Un dataset sintetico è un dataset in cui i dati e le annotazioni sono il risultato di un processo di generazione automatizzata al calcolatore. Dati di questo tipo non solo sopperiscono alla forte dipendenza da un procedimento umano, e

quindi soggetto ad errori, ma permettono di avere anche un controllo completo sulla forma e il contenuto dei dati di cui si ha bisogno [8]. Una rete neurale addestrata su un dataset sintetico però fallisce nel momento in cui la si applica ai dati reali [9]. I dati creati artificialmente, per quanto possano essere per certamente realistici, non garantiscono l'apprendimento della vera relazione che c'è tra quelli che si vuole imitare e l'informazione implicita da estrarre.

Cercare di capire i motivi che stanno alla base di questo fenomeno è di per sé già un problema: le reti neurali artificiali infatti sono considerate delle "black box", la loro estrema capacità decisionale si paga con una quasi inesistente interpretabilità logica [10].

In questo lavoro di tesi abbiamo studiato Segnet [11], una DNN che fa segmentazione di immagini, ovvero che categorizza ogni singolo pixel dell'immagine che prende in input. L'abbiamo addestrata su un dataset sintetico e ne abbiamo studiato le proprietà attraverso due metodi di indagine per le reti neurali artificiali che sono allo stato dell'arte: Grad-CAM [12] e la Layer-wise Relevance Propagation (LRP) [13]. L'obiettivo che ci siamo posti è capire qual'è la fonte dell'errore della rete quando la si applica a immagini reali: con ciò intendiamo non solo la porzione dell'immagine che genera l'errore, inteso come inesattezza dell'informazione estratta, ma anche la parte dell'architettura della rete che non è in grado di gestirlo

In questo primo capitolo e nel successivo vedremo rispettivamente un riassunto teorico di base sulle reti neurali artificiali e i principali compiti della computer vision. Essi ci permetteranno di comprendere l'architettura delle reti neurali convoluzionali (convolutional neural network - CNN, capitolo 3) ed in particolare di SegNet (capitolo 4), la rete che è stata oggetto dei nostri studi. Nel capitolo 5 faremo una panoramica sui principali metodi di indagine presenti in letteratura e tra questi ne vedremo due in maniera più approfondita: Grad-CAM nel capitolo 7 e la layer-wise relevance propagation (LRP) nel capitolo 6. Nel capitolo 8 entreremo nei dettagli dell'implementazione di LRP su Segnet, la tecnica principale utilizzata negli esperimenti raccontati nel capitolo 9. L'ultimo capitolo infine sarà dedicato alle riflessioni e alle conclusioni.

## 1.1 Neurone artificiale

Il neurone artificiale è l'elemento base che compone i modelli matematici che abbiamo studiato nel nostro lavoro. Dato un vettore di input  $\vec{x} \in \mathbb{R}^d$ , un vettore di pesi (parametri)  $\vec{w} \in \mathbb{R}^d$ , un bias  $b \in \mathbb{R}$  e una funzione di attivazione non lineare  $g : \mathbb{R} \rightarrow \mathbb{R}$ , il neurone artificiale è un'unità di calcolo definibile attraverso una funzione  $f$ :

$$f : \mathbb{R}^d \rightarrow \mathbb{R} \quad (1.1)$$

$$f(\vec{x}; \vec{w}, b) = g\left(b + \sum_{i=1}^d x_i \cdot w_i\right) \quad (1.2)$$

Chiamiamo il valore  $f(\vec{x})$  *attivazione* del neurone. Dati due neuroni  $\alpha$  e  $\beta$ , se l'attivazione di  $\alpha$  è una variabile di input per  $\beta$  (esempio in figura 1.1) allora essi sono definiti *connessi*, la connessione è rappresentabile con una freccia che in questo caso va da  $\alpha$  a  $\beta$  ( $\alpha \rightarrow \beta$ ), e il peso relativo al valore per cui  $\alpha$  è di input a  $\beta$  è detto peso della connessione.

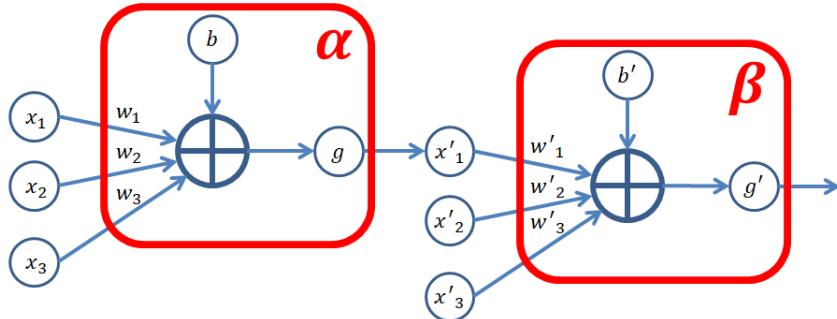


Figura 1.1: Esempio di due neuroni connessi: l'attivazione del neurone  $\alpha$  è di input per il neurone  $\beta$ , il peso  $w'_1$  è il peso della connessione

## 1.2 Reti neurali artificiali

Una rete neurale artificiale è un modello matematico descrivibile attraverso un grafo diretto in cui i nodi sono rappresentati dai neuroni, e i link sono le connessioni tra neuroni definite nel paragrafo precedente. Una composizione di

questo tipo modellizza una funzione  $h(\vec{x}; W)$

$$h : \mathbb{R}^l \rightarrow \mathbb{R}^m \quad (1.3)$$

dove  $l$  è il numero di neuroni della rete (nodi del grafo) che non hanno connessioni in ingresso,  $m$  è il numero di neuroni che non hanno connessioni in uscita e  $W$  è l'insieme dei parametri della rete. Chiameremo i primi: neuroni di input della rete; e i secondi: neuroni di output. Le reti neurali più complesse hanno una architettura organizzata a strati, detti più comunemente layer. Un layer è un insieme di neuroni che non sono interconnessi, e che hanno connessioni soltanto con neuroni non appartenenti all'insieme. Definiamo il layer che contiene i

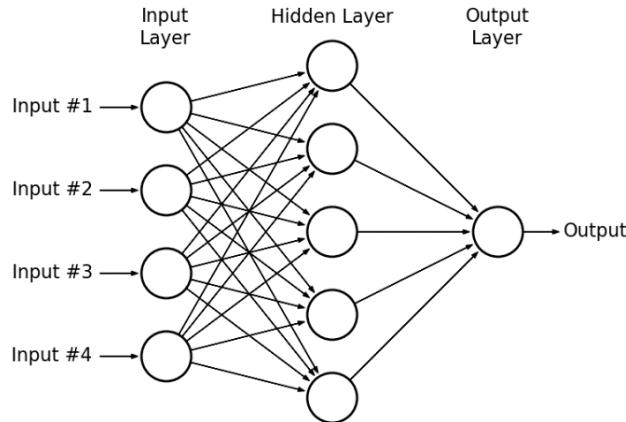


Figura 1.2: Esempio di rete neurale multistrato

neuroni di input: *layer di input* o *input layer*; analogamente definiamo il *layer di output* o *output layer*. Se in una rete neurale artificiale ci sono dei layer che non sono né di input né di output allora li definiamo *nascosti* (*hidden layer*). Spesso la parola *layer* è accompagnata dal nome dell'operazione che definisce i neuroni di quel layer. Ad esempio un "layer di convoluzione" è il risultato di un'operazione di convoluzione, ovvero è un insieme di neuroni le cui attivazioni dipendono da essa.

## 1.3 Addestrare una rete neurale

Addestrare una rete neurale non risulta un compito semplice, trattandosi come abbiamo visto di una complessa funzione composta  $h(\vec{x}; W)$  del vettore di input  $\vec{x}$  e dell'insieme dei parametri  $W$ . Innanzitutto è necessario scegliere opportunamente le funzioni di attivazione  $g$  in modo che tale funzione risulti differenziabile. Si tratta quindi di risolvere, dato un dataset  $\{\vec{x}_i, \vec{y}_i\}_1^n$  e una funzione di perdita  $LOSS[\vec{y}, h(\vec{x})]$ , un problema di minimizzazione:

$$\min_W \left\{ \frac{1}{n} \sum_{i=1}^n LOSS(\vec{y}_i, f(\vec{x}_i; W)) + \lambda J(W) \right\} \quad (1.4)$$

dove  $J(W)$  è un eventuale termine non negativo di regolarizzazione sugli elementi di  $W$ , con  $\lambda$  relativo parametro di regolazione. Le funzioni di perdita  $LOSS[\vec{y}, f(\vec{x})]$  hanno un minimo globale in  $\vec{y} = f(\vec{x})$  e sono generalmente convesse in  $f$ , ma non negli elementi di  $W$ , con la conseguenza che ci troviamo a dover risolvere un problema di minimizzazione su una superficie che presenta una grande quantità di minimi locali [14]. Una soluzione è quella di effettuare più stime dello stesso modello con differenti inizializzazioni dei parametri, per scegliere infine quello che risulta essere migliore. Una procedura di questo tipo può però richiedere molto tempo, pertanto generalmente ci si tende ad accontentare di buoni minimi locali.

### 1.3.1 Discesa del gradiente

I principali metodi utilizzati per la risoluzione della formula (1.4) sono basati sulla tecnica della discesa del gradiente [15]. Indicizziamo con  $k$  tutti i parametri (pesi)  $W^{(k)}$  della rete. Per il calcolo del gradiente globale sarà sufficiente effettuare la media lungo tutti gli elementi del dataset:

$$\Delta W^{(k)} = \frac{1}{n} \sum_{i=1}^n \frac{\partial LOSS[\vec{y}_i, f(\vec{x}_i; W)]}{\partial W^{(k)}} \quad (1.5)$$

Una volta calcolato il gradiente è possibile procedere con l'aggiornamento dei pesi:

$$W^{(k)} \leftarrow W^{(k)} - \alpha(\Delta W^{(k)} + \lambda W^{(k)}) \quad \text{per } k = 1, \dots, K \quad (1.6)$$

Il calcolo del gradiente ci fornisce la direzione lungo la quale la superficie da minimizzare è più ripida, ma nessuna informazione circa la lunghezza del passo che dovremmo compiere. Tale quantità, rappresentata dal parametro  $\alpha$  nella (1.6) viene denominata *learning-rate* e costituisce l'iperparametro più importante da regolare in una rete neurale: valori troppo alti portano ad una convergenza più veloce ma sono rischiosi dal momento che potrebbero saltare il minimo ottimale o fluttuare intorno ad esso, mentre valori troppo bassi sono responsabili di una convergenza lenta e possono comportare l'arresto dell'algoritmo in un minimo locale non ottimale.

### 1.3.2 Discesa stocastica del gradiente (stochastic gradient descent - SGD)

Esistono alcune varianti dell'algoritmo di discesa del gradiente che differiscono nella quantità di dati utilizzati nel calcolo del gradiente prima di effettuare l'aggiornamento dei parametri. La formula (1.6) utilizza ad ogni iterazione l'intero dataset, e viene denominata batch gradient descent. Tuttavia, può spesso risultare più efficiente processare piccole quantità di dati (mini-batch) per volta, generalmente campionate in maniera casuale (da qui il termine discesa stocastica) [16]. Tale scelta è obbligata quando le dimensioni del dataset sono tali da non poter essere caricato in memoria. Indipendentemente dal numero di iterazioni e dalla dimensione del batch prescelta, ogni volta che tutte le  $n$  osservazioni del dataset vengono utilizzate per il calcolo del gradiente si dice che viene completata un'epoca.

### 1.3.3 Algoritmo di ottimizzazione della discesa del gradiente - Momentum

La discesa stocastica del gradiente presenta alcune difficoltà in prossimità di aree dove la superficie della funzione di perdita (*LOSS*), nello spazio dei parametri, presenta una curvatura molto più accentuata in una direzione rispetto all'altra. In questo scenario infatti l'algoritmo tende ad oscillare lungo il versante più ripido, rallentando così la convergenza verso il minimo locale ottimale (figura 1.3). Il momentum [17] è un metodo che aiuta ad accelerare la discesa del gra-

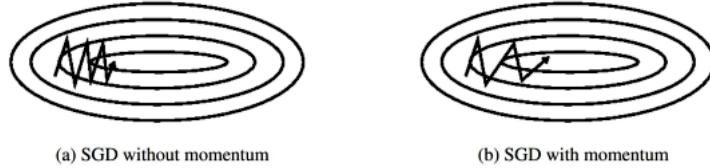


Figura 1.3: Discesa del gradiente senza (a) e con (b) momentum

diente verso la direzione corretta, smorzando l'effetto indotto dalle oscillazioni. Tale risultato si ottiene aggiungendo al termine di aggiornamento corrente una frazione  $\gamma$  del vettore di aggiornamento precedente:

$$v_t = \gamma v_{t-1} + \alpha \nabla_{W\psi}(W^{(k)}) \quad (1.7)$$

$$W^{(k)} \leftarrow W^{(k)} - v_t \quad (1.8)$$

dove  $\nabla_{W\psi}(W^{(k)})$  è il gradiente rispetto a  $W$  della funzione da minimizzare (argomento della formula (1.4)).

## Capitolo 2

# Computer vision

In questo capitolo cerchiamo di inserire il nostro lavoro nel contesto più ampio del campo della computer vision. La visione artificiale (computer vision) è un campo interdisciplinare che riguarda il modo in cui i computer possono essere realizzati per ottenere una comprensione di alto livello da immagini o video digitali. Dal punto di vista dell'ingegneria, essa cerca di automatizzare le attività che il sistema visivo umano può svolgere.

Un sistema di visione artificiale è costituito dall'integrazione di componenti ottiche, elettroniche e meccaniche che permettono di acquisire, registrare ed elaborare immagini sia nello spettro della luce visibile che al di fuori di essa (infrarosso, ultravioletto, raggi X, ecc.). Il risultato dell'elaborazione è il riconoscimento di determinate caratteristiche dell'immagine per varie finalità di controllo, classificazione, selezione, ecc.

### 2.1 Compiti della visione artificiale (computer vision tasks)

I compiti della computer vision si dividono in base al tipo di informazioni che si vogliono estrarre da una immagine. Quelli principali sono:

1. classificazione
2. localizzazione

### 3. segmentazione

### 4. detection

1) Dato un sistema di visione artificiale e una immagine di input per il sistema, la classificazione è il processo mediante il quale il sistema assegna un'etichetta all'immagine, ovvero un nome, una classe o una categoria rappresentativa di che cosa l'immagine contiene (un esempio di immagini classificate è mostrato in figura 2.1). Per sistemi di questo tipo, il numero di classi tramite le quali si possono distinguere gli oggetti è limitato e dipende dall'architettura del sistema. L'esempio più famoso di task di classificazione è costituito dal dataset ImageNet [18] sul quale sono state addestrate negli ultimi anni le più moderne reti neurali convoluzionali.

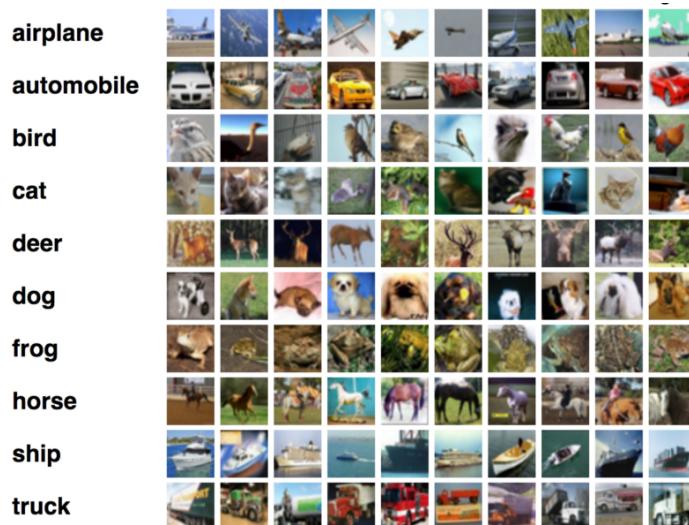


Figura 2.1: Insieme di immagini raggruppate in base alla categoria di appartenenza (classificazione)

- 2) Alla classificazione segue la localizzazione. Data una immagine e una categoria di appartenenza (il risultato della classificazione), con la localizzazione si vuole costruire un rettangolo nell'immagine (bounding box) che contiene l'oggetto rappresentativo di quella categoria (esempio in figura 2.2). Il rettangolo contiene quindi le informazioni sulla posizione ed estensione (intesa come dimensione in pixel) dell'oggetto nell'immagine.
- 3) Dato un sistema di visione artificiale in grado di fare detection, l'insieme delle categorie di oggetti che è in grado di distinguere e una immagine di input,

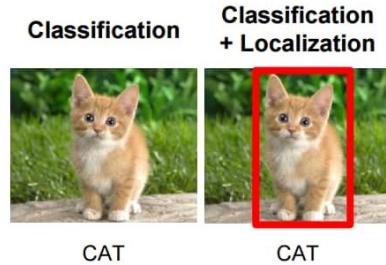


Figura 2.2: A sinistra la classificazione: l'immagine viene etichettata come "gatto". A destra la localizzazione: il bounding box racchiude la porzione di immagine che contiene l'oggetto "gatto" determinante per la classificazione

la detection consiste nel classificare e nell'individuare con un bounding box i molteplici oggetti presenti nell'immagine, come è mostrato in figura 2.3.

### Object Detection

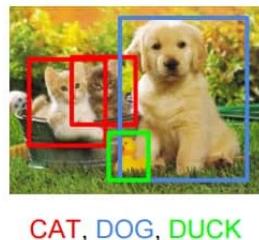


Figura 2.3: Esempio di object detection: a differenza della classificazione dove ad una immagine corrisponde una e una sola etichetta, nella detection la categorizzazione è multipla

- 4) Con la segmentazione infine intendiamo il compito di un sistema di visione artificiale di classificare ogni singolo pixel dell'immagine in input, un esempio è mostrato in figura 2.4.



Figura 2.4: A sinistra: immagine di input. A destra: la segmentazione dell'immagine (ad ogni colore diverso è associata una categoria, ad esempio: strada, albero, macchina ecc)

## Capitolo 3

# Classificazione e reti neurali convoluzionali - VGG

Un tipico sistema di visione artificiale utilizzato per la classificazione di immagini è la rete neurale convoluzionale (convolutional neural network - CNN). In questo paragrafo vedremo quali sono le parti principali che compongono questa classe di sistemi analizzando l'architettura di VGG [19].

### 3.1 Architettura

VGG è una CNN che classifica immagini. E' composta da una serie di layer di convoluzione+ReLU e max-pooling, termina con dei fully-connected layer e un softmax. Il primo layer è quello di input in cui vengono caricate le immagini da classificare. Le immagini si possono vedere come delle matrici di dimensioni  $(3 \times 224 \times 224)$  dove le prime due corrispondono all'altezza e larghezza dell'immagine, e tre sono i canali RGB con cui vengono codificati i colori (red, green, blue). L'output della rete è un vettore di dimensione 1000, come sono 1000 le categorie di classificazione. Ogni elemento del vettore di output è un numero che esprime la probabilità che l'immagine in input rappresenti la classe corrispondente. L'architettura completa è mostrata in figura 3.1

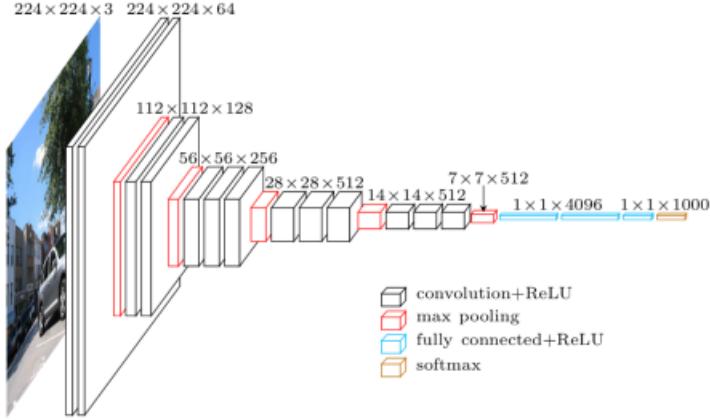


Figura 3.1: Architettura di VGG

### 3.2 Layer di convoluzione

Il layer di convoluzione è la componente fondamentale delle CNN, fu utilizzata per la prima volta nel campo della computer vision con AlexNet [20] nel 2012. Per layer di convoluzione intendiamo l'insieme dei neuroni le cui attivazioni derivano dall'operazione di convoluzione fatta sul layer che li precede. L'operazione in sé è peculiare per il modo in cui definisce le connessioni tra i neuroni di due layer consecutivi. Consideriamo un layer di dimensioni  $(c \times h \times w)$  dove  $h$  e  $w$  sono l'altezza e la larghezza del layer e  $c$  è la profondità (canali del layer). Il layer appena definito sarà l'input dell'operazione di convoluzione ed essa dipende da 4 fattori:

1. L'estensione del kernel (o filtro)  $d$
2. Il numero di canali dell'output  $c'$
3. Lo stride del kernel  $s$
4. Il padding dell'input  $p$

Il kernel è una matrice numerica di dimensioni  $(c \times d \times d)$ . Esso sovrapposto alla matrice di input e opera eseguendo una moltiplicazione elemento per elemento (point-wise) con le attivazioni dei neuroni, a cui segue una somma come mostrato in figura 3.2. L'altezza  $h'$  e la larghezza  $w'$  dell'output dipendono dallo stride e dal padding. Lo stride è un parametro che definisce di quanto deve essere

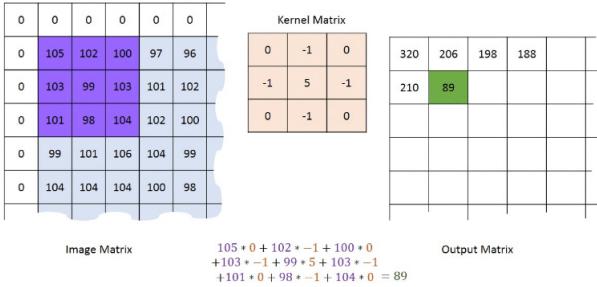


Figura 3.2: Esempio di convoluzione con stride  $s = 1$ , padding  $p = 1$ , e kernel di estensione  $d = 3$

traslato il kernel prima di eseguire una nuova moltiplicazione+somma, mentre il padding è un artificio che consiste nel circondare con  $p$  righe di zeri l'input layer in modo tale che il risultato della convoluzione generi un layer della medesima altezza e larghezza dell'input ( $h' = h, w' = w$ ). Ad esempio, per come avviene in ogni convoluzione di VGG, se si utilizza un kernel di dimensioni  $(c \times 3 \times 3)$ , per ottenere un output delle stesse dimensioni, lo stride  $s$  e il padding  $p$  devono essere uguali a 1. Nel modo appena descritto un singolo kernel produce una matrice di dimensioni  $(h \times w)$ . Da qui in avanti chiameremo queste matrici bidimensionali con il nome di *feature*.

**Definizione 1.** Una *feature* (o *feature map*) è una matrice bidimensionale che deriva dall'operazione di convoluzione+ReLU di un singolo kernel su un layer, gli elementi della matrice sono le attivazioni dei neuroni. Dato un qualsiasi layer di dimensioni  $(c \times h \times w)$  dove  $c$  è il numero dei canali ed  $h$  e  $w$  sono rispettivamente l'altezza e la larghezza del layer, più in generale chiamiamo *feature* ognuna delle  $c$  sezioni di dimensioni  $(h \times w)$  del layer

Se uso un numero  $c'$  di kernel (filter bank) ottengo  $c'$  feature di dimensioni  $(h \times w)$ , raggruppabili in un unico blocco di ouput di dimensioni  $(c' \times h \times w)$ . Ogni elemento di questo blocco (layer) è quindi un neurone, le sue connessioni sono definite dal passaggio del kernel sul layer in input all'operazione (neuroni di input) e i pesi delle connessioni sono gli elementi del kernel, che sono i parametri addestrabili della rete.

### 3.3 Rectified linear unit

In VGG la funzione di attivazione di ogni neurone è la ReLU (rectified linear unit), e viene applicata successivamente ad ogni operazione di convoluzione.

$$ReLU(x) = \max(0, x) = \begin{cases} x & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases} \quad (3.1)$$

### 3.4 Max-pooling

In VGG ci sono 5 layer di Max-pooling. La funzione del max-pooling è quella di ridurre progressivamente le dimensioni spaziali  $h$  e  $w$  dei layer per diminuire il numero dei parametri nella rete e quindi i tempi di calcolo. Il max-pooling opera indipendentemente su ogni feature del layer usando la funzione  $MAX$ . In VGG una finestra di  $(2 \times 2)$  scorre sull'intera feature con uno stride  $s = 2$  (in questo modo non c'è una sovrapposizione delle finestre), calcola il  $MAX$  tra gli elementi evidenziati, conserva esso e scarta gli altri, un esempio è mostrato in figura 3.3. Dato un layer di dimensioni  $(c \times h \times w)$ , l'operazione di max-pooling

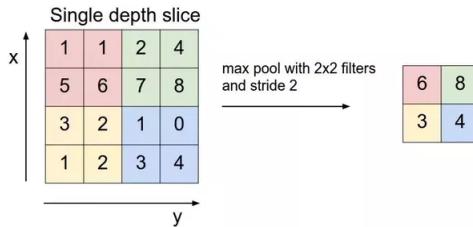


Figura 3.3: Esempio di max-pooling

genera un nuovo layer di dimensioni  $(c \times \frac{h}{2} \times \frac{w}{2})$

### 3.5 Fully-connected layer

In VGG l'ultima operazione di max-pooling genera un layer di dimensioni  $(7 \times 7 \times 512)$  per un totale di 4096 neuroni. I due layer successivi hanno ancora 4096 neuroni e sono completamente connessi (fully-connected), ovvero ognuno di essi ha una connessione con ogni neurone del layer precedente. L'ultimo layer della rete è sempre fully-connected ma ha 1000 neuroni invece di 4096. Per

tutti i neuroni dei layer fully-connected, tranne l'ultimo layer, la funzione di attivazione è sempre la ReLU.

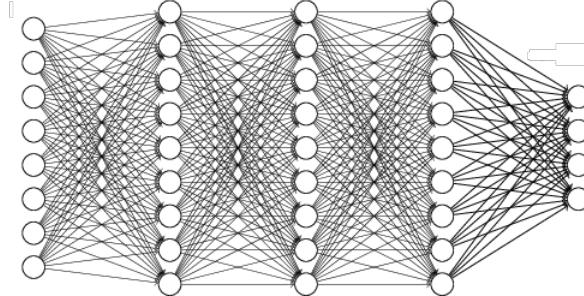


Figura 3.4: Esempio di successione di layer fully-connected, ogni neurone è connesso con tutti i neuroni del layer precedente

### 3.6 Softmax

Per l'ultimo layer le attivazioni dei neuroni sono date in input al softmax invece che alla ReLU. Il softmax è una funzione che mappa i  $k$  valori reali di un vettore  $\vec{z}$   $k$ -dimensionale in un vettore  $\vec{\sigma}(\vec{z})$  sempre di dimensione  $k$  con valori compresi tra 0 e 1.

$$\sigma : \mathbb{R} \rightarrow \left\{ \vec{z} \in \mathbb{R}^K \mid z_i > 0, \sum_{i=1}^K z_i = 1 \right\} \quad (3.2)$$

$$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{i=1}^K e^{z_i}} \quad (3.3)$$

Quando si utilizza VGG per fare inferenza su un'immagine, l'output della rete è un vettore di dimensione 1000 normalizzato ad 1, ovvero ognuno degli elementi (uno per ogni categoria di classificazione) rappresenta la probabilità che l'immagine mostrata alla rete appartenga a quella categoria. La decisione della rete è la categoria con la probabilità più alta.

**Definizione 2.** *l'attivazione  $z_i$  che viene data di input al softmax viene definita "punteggio" della classe  $i$*

## Capitolo 4

# Reti di segmentazione - SegNet

SegNet [11] è una rete neurale artificiale fully-convolutional per la segmentazione di immagini. La segmentazione è uno dei task principali della computer vision: per segmentazione si intende la procedura con la quale si assegna ad ogni pixel di una immagine un label (etichetta), in base alle classi di oggetti che è in grado di riconoscere.

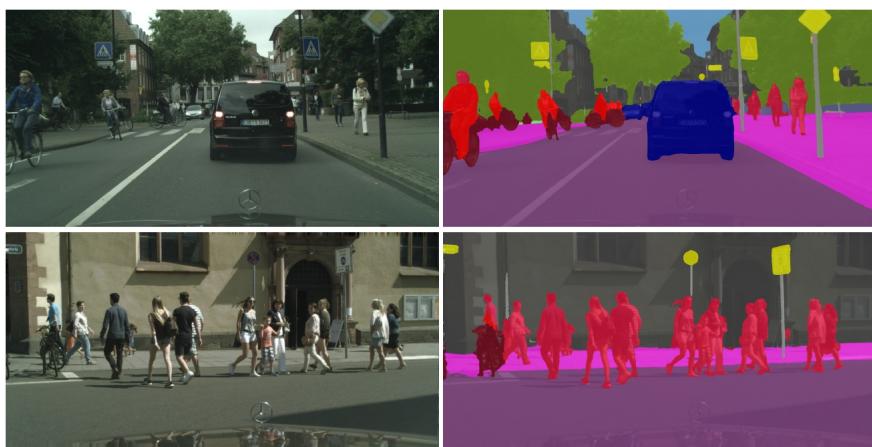


Figura 4.1: A sinistra: immagini di input. A destra: output segmentato, in questo caso la rete è in grado di riconoscere i seguenti oggetti: albero, asfalto, pedone, macchina, cartello stradale, marciapiede ecc

## 4.1 Innovazione

Quando ha iniziato ad esserci un interesse attivo per la risoluzione di questo tipo di task, i primi approcci consistevano soltanto in una parziale modifica delle più efficienti architetture di classificazione di immagini, come ad esempio VGG [19]. Il trend era quello di sostituire i layer fully-connected con un altro layer di convoluzione con  $c$  feature, dove  $c$  è il numero di categorie di classificazione di ogni pixel. Nostante i risultati fossero incoraggianti, essi apparivano grossolani [21]. Tutto ciò era dovuto ai layer di max-pooling e al sub-sampling in generale (per sub-sampling si intende l'insieme di tutte quelle operazioni, come ad esempio il max-pooling, che diminuiscono le dimensioni dei layer). Le segmentazioni ottenute avevano le dimensioni delle feature dell'ultimo layer di convoluzione, dopodichè venivano riscalate alle dimensioni originali tramite interpolazione lineare. Il particolare design di SegNet è stato sviluppato in risposta all'esigenza di risolvere questa problematica: esso mappa le feature a bassa risoluzione in una risoluzione che è quella dell'input, ovvero dell'immagine originale. Questo approccio permette di migliorare la definizione dei contorni che separano un oggetto e l'altro nell'immagine.

SegNet è stata pensata per la segmentazione di immagini che rappresentano ambienti stradali, questo compito richiede la capacità di modellizzare l'aspetto e la forma degli oggetti (strada, marciapiede, macchine, pedoni, strisce, ecc) e di capire le relazioni spaziali tra i vari oggetti (ad esempio il marciapiede si trova ai lati della strada). In questi tipi di ambienti, la maggior parte dei pixel appartengono ad oggetti di dimensioni estese (ad esempio la strada e il background), e quindi è di fondamentale importanza riconoscere i bordi in modo ben definito. Una rete di segmentazione deve anche essere in grado di riconoscere oggetti di dimensioni relativamente inferiori (strisce, segnaletica ecc). Inoltre SegNet deve essere efficiente dal punto di vista computazionale, nei termini di tempi di calcolo e memoria utilizzata in fase di inferenza. Poterla addestrare per ottimizzare congiuntamente tutti i pesi della rete usando una efficiente tecnica di aggiornamento dei pesi come la stochastic gradient descent (SGD, capitolo 1.3.2), è un vantaggio aggiuntivo. Il design di SegNet nasce dal bisogno di combinare tutti questi criteri.

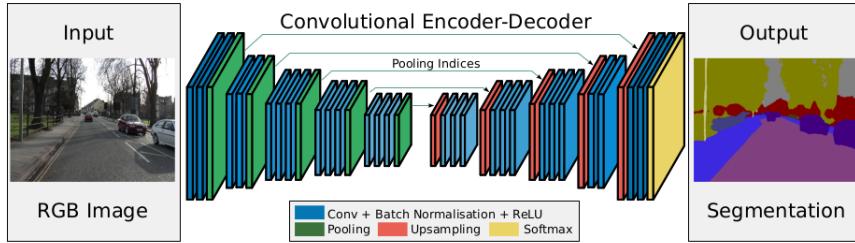


Figura 4.2: Un’illustrazione dell’architettura di SegNet. Non ci sono fully-connected layer, ma solo layer convoluzionali. Un decoder ricostruisce l’input utilizzando gli indici dei layer di max-pooling trasferiti dall’encoder, per produrre una feature map sparsa. A queste vengono seguite delle convoluzioni con filtri addestrabili che ridensificano le feature. Le ultime feature del decoder sono date in input al softmax

## 4.2 Architettura

SegNet è formato da due parti principali, un encoder e un decoder, e termina con un layer di classificazione pixel per pixel. La sua architettura è mostrata nella figura 4.2. L’encoder è costituito da 13 layer convoluzionali che corrispondono esattamente ai primi 13 layer della rete VGG [19], progettata per la classificazione di immagini. Questa caratteristica è fondamentale perché si ha la possibilità di inizializzare l’encoder di SegNet con i pesi generati dall’addestramento di una VGG. Il layer finale fully-connected viene scartato, permettendo di diminuire notevolmente il numero di parametri da addestrare rispetto ad altre architetture come [22]. Ogni layer dell’encoder ha un corrispettivo layer nel decoder, quindi anche il decoder ha 13 layer di convoluzione. L’output del decoder, infine, termina nel softmax, per produrre indipendentemente la probabilità delle classi per ogni singolo pixel.

Ogni blocco nell’encoder della rete esegue una o più convoluzioni con un filter bank per produrre un insieme di feature map. Ad ogni layer di convoluzione segue uno di batch normalization (capitolo 4.3) [23]. Poi viene applicata una ReLU (Rectified linear unit) su ogni attivazione del layer. Al termine del blocco, un max-pooling con stride 2 e una finestra  $2 \times 2$  riduce le dimensioni delle feature di un fattore 2. Il max-pooling viene utilizzato per raggiungere l’invarianza spaziale per piccole traslazioni dell’immagine di input, in quanto i neuroni nei layer più profondi hanno un maggiore campo recettivo (capitolo 7.2.1). Ma questo determina una diminuzione della risoluzione, compromettendo la capacità

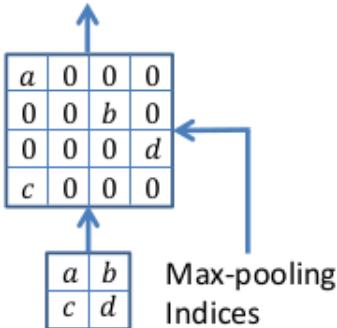


Figura 4.3: Meccanismo di upsampling. Gli indici in cui vengono distribuite le attivazioni sono gli stessi di quelli ricavati dal max-pooling nell'encoder

della rete di produrre segmentazioni ben definite.

E’ necessario quindi catturare le informazioni fondamentali sui contorni degli oggetti e memorizzarle, per riutilizzarle nel decoder in fase di ricostruzione dell’immagine. Bastano 2 bit per memorizzare questa informazione, ogni 2 bit viene indicato l’indice posizionale nella finestra  $2 \times 2$  del max-pooling dell’encoder.

Ogni blocco decodificante invece è del tutto uguale al corrispettivo nell’encoder, tranne per il layer di max-pooling. Il primo layer di ogni blocco riceve di input il risultato dell’operazione di up-sampling dal blocco precedente. L’up-sampling consiste nel distribuire le attivazioni in una matrice sparsa, di dimensioni doppie in larghezza e altezza, dove ogni singolo neurone viene posizionato nella corrispettiva finestra  $2 \times 2$  nella posizione che è stata memorizzata nel layer corrispondente dell’encoder. Un esempio di questa procedura è mostrato nella figura 4.3. Nel blocco del decoder, la mappa sparsa ottenuta dall’upsampling viene densificata attraverso una o più convoluzioni. Ogni convoluzione, come nei layer del decoder, è seguita da una batch normalization e una ReLU. Da notare che il numero di feature generate dalla convoluzione nel decoder sono le stesse per ogni layer al corrispettivo nell’encoder. Solo l’ultimo layer della rete è diverso, non produce 3 feature corrispondenti ai canali RGB dell’input, ma ne produce  $K$ , dove  $K$  è il numero di classi da segmentare. Al termine del decoder avremo quindi un layer di dimensioni  $(K \times h \times w)$ , dove  $h$  e  $w$  sono le dimensioni dell’immagine di input, quindi l’operazione di softmax viene applicata sui  $K$  canali per ogni elemento  $(h, w)$  del layer, ognuno dei quali corrisponde proprio

ad un pixel dell’immagine

### 4.3 Batch normalization layer

l’addestramento delle reti neurali profonde (deep neural networks - DNN) è complicato dal fatto che la distribuzione dell’input di ogni layer cambia continuamente durante questa fase, poichè cambiano i parametri dei layer che li precedono ad ogni step della stochastic gradient descent. Questo fenomeno allunga le tempistiche di addestramento perchè esso necessita di una diminuzione del learning-rate (definizione nel capitolo 1.3.1) e di una attenta inizializzazione dei parametri. Ci riferiamo a questo fenomeno col nome di *internal covariate shift*, e il *batch normalization* layer è lo strumento che si utilizza per contenerlo. Essa trae forza dal rendere la normalizzazione dell’input del layer come parte del modello dell’architettura, ed eseguendo la normalizzazione per ogni mini-batch del training. La batch normalization permette di utilizzare un learning-rate molto più alto e di essere meno scrupolosi sull’inizializzazione dei parametri.

#### 4.3.1 Internal covariate shift

Vediamo un po’ più nel dettaglio che cos’è l’internal covariate shift. Consideriamo una rete che calcola

$$l = F_2(F_1(u, \Theta_1), \Theta_2) \quad (4.1)$$

dove  $F_1$  e  $F_2$  sono trasformazioni arbitrarie, i parametri  $\Theta_1$  e  $\Theta_2$  sono gli insiemi dei parametri da addestrare per minimizzare la loss  $l$  e  $u$  è l’input di  $F_1$ . Addestrare  $\Theta_2$  può essere visto come se  $x = F_1(u, \Theta_1)$  fosse dato in input alla sub-network

$$l = F_2(x, \Theta_2) \quad (4.2)$$

Ad esempio, uno step di gradient descent

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2} \quad (4.3)$$

(per un batch di dimensione  $m$  e un learning rate  $\alpha$ ) è esattamente equivalente a quello per la sola network  $F_2$  con input  $x$ . Quindi, le proprietà di distribuzione dell'input che rendono il training più efficiente si applicano allo stesso modo alla sub-network. Perciò è vantaggioso per la distribuzione di  $x$  rimanere costante nel tempo. Quindi,  $\Theta_2$  non deve riaggiustarsi per compensare i cambiamenti della distribuzione  $x$ .

### 4.3.2 Algoritmo

Consideriamo un layer di una rete neurale con i relativi neuroni di input e di output. L'algoritmo di batch normalization prevede la sostituzione dell'input con una trasformazione dello stesso. Questa trasformazione permette di mantenere la distribuzione di input costante durante l'addestramento. Consideriamo l'input come un vettore  $x = (x^{(1)} \dots x^{(k)})$  di dimensione  $k$ , normalizziamo ogni dimensione

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}} \quad (4.4)$$

dove la media  $\mathbb{E}$  e la varianza  $\text{Var}$  sono calcolate su tutto il training set. Per assicurarci che la trasformazione inserita nella rete rappresenti una identità, introduciamo per ogni attivazione  $x^{(k)}$ , una coppia di parametri  $\gamma^{(k)}, \beta^{(k)}$ , che scalano e traslano il valore normalizzato

$$y^{(k)} = \gamma^{(k)} \hat{x}^k + \beta^{(k)} \quad (4.5)$$

Questi parametri sono soggetti anch'essi all'addestramento, e ristabiliscono il potere di rappresentazione della rete. Nella pratica è raro poter utilizzare tutto il training set in ogni step dell'addestramento, soprattutto nel campo della computer vision, i limiti sono imposti dalle grosse dimensioni delle immagini, che richiedono molta RAM per essere processate. Per questo motivo bisogna fare una modifica. Consideriamo un mini-batch  $B$  di dimensione  $m$ . Siccome la normalizzazione è applicata ad ogni attivazione indipendentemente, concentriamoci su una particolare attivazione  $x^{(k)}$  e omettiamo  $k$  per semplicità. Abbiamo gli  $m$  valori di attivazione del mini-batch di addestramento,

$$\mathcal{B} = \{x_{1\dots m}\} \quad (4.6)$$

<b>Input:</b>	Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$ ;
	Parameters to be learned: $\gamma, \beta$
<b>Output:</b>	$\{y_i = BN_{\gamma, \beta}(x_i)\}$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$	// scale and shift

Figura 4.4: Algoritmo di batch normalization

Siano  $\hat{x}_{1\dots m}$  i valori normalizzati, e le loro trasformazioni lineari  $y_{1\dots m}$ . Ci riferiamo alla trasformazione

$$BN_{\gamma, \beta} : x_{1\dots m} \rightarrow y_{1\dots m} \quad (4.7)$$

come *Batch Normalizing Transform*. Presentiamo la trasformata BN nell'algoritmo mostrato in figura 4.4. Ricapitolando, l'algoritmo appena mostrato rappresenta ciò che viene fatto durante l'addestramento della rete: una trasformazione al layer di input è stata aggiunta e ci sono due parametri in più  $\gamma^{(k)}$  e  $\beta^{(k)}$ , uno per ogni attivazione di input, che devono essere addestrati usando il batch gradient descent.

Ora vediamo cosa accade durante la fase di inferenza. La normalizzazione, in fase di training, dipendeva dal batch e serviva per rendere il training stesso più efficiente. In fase di inferenza questa cosa non è più necessaria o desiderabile; vogliamo che l'output dipenda unicamente dall'input in modo deterministico. Per questo motivo, una volta che la rete è stata addestrata usiamo la normalizzazione

$$\hat{x} = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x]}} \quad (4.8)$$

usando l'intera popolazione dei dati piuttosto che il mini-batch. Queste attivazioni normalizzate avranno media 0 e varianza 1 proprio come durante il

training. Usiamo la varianza campionaria corretta  $\text{Var}[x] = \frac{m}{m-1} \cdot E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$ , dove il valore atteso è sul mini-batch di dimensione  $m$  del training e  $\sigma_{\mathcal{B}}^2$  è la rispettiva varianza. Siccome media e varianza sono fissate durante l'inferenza, la normalizzazione è semplicemente una trasformazione lineare applicata a ciascuna attivazione. In figura 4.5 è mostrato l'intero processo di addestramento di

<b>Input:</b> Network $N$ with trainable parameters $\Theta$ ; subset of activations $\{x^{(k)}\}_{k=1}^K$
<b>Output:</b> Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$
1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network 2: <b>for</b> $k = 1 \dots K$ <b>do</b> 3:   Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. I) 4:   Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead 5: <b>end for</b> 6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$ 7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters 8: <b>for</b> $k = 1 \dots K$ <b>do</b> 9:   // For clarity, $x \equiv x^{(k)}$ , $\gamma \equiv \gamma^{(k)}$ , $\mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$ , etc. 10:   Process multiple training mini-batches $\mathcal{B}$ , each of size $m$ , and average over them: $E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ $\text{Var}[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$ 11:   In $N_{\text{BN}}^{\text{inf}}$ , replace the transform $y = \text{BN}_{\gamma, \beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$ 12: <b>end for</b>

Figura 4.5: Addestramento di una rete batch-normalizzata

una rete con layer di batch-normalization, ulteriori dettagli sono descritti in [23]

## Capitolo 5

# Metodi di indagine per le reti neurali profonde

Le tecniche di machine learning come le reti neurali profonde (deep neural network - DNN) sono diventate uno strumento indispensabile per una vasta gamma di applicazioni come ad esempio la classificazione di immagini, il riconoscimento vocale e l'elaborazione del linguaggio naturale (natural language processing). Queste tecniche hanno raggiunto un'elevatissima accuratezza predittiva, e in alcuni casi al pari delle prestazioni umane.

Nella pratica, è anche essenziale verificare per un certo compito, che l'accuratezza derivi da un uso appropriato della rappresentazione del problema, e non dallo sfruttamento di artefatti presenti nei dati. Tecniche per interpretare e capire che cosa il modello ha imparato sono perciò diventate un ingrediente chiave per una robusta procedura di validazione. L'interpretabilità è importante soprattutto in applicazioni come ad esempio la medicina o le auto che si guidano da sole (self-driving cars), dove la dipendenza del modello dalle corrette caratteristiche nei dati deve essere garantita.

In questo capitolo ci concentriamo sull'interpretabilità *a posteriori* delle reti, ovvero dato un modello già addestrato il nostro obiettivo è quello di capire che cosa il modello predice nei termini di che cosa è accessibilmente interpretabile (ad esempio le variabili di input). Vedremo una serie di metodologie che abbiamo raggruppato in due filoni principali che si distinguono per il tipo di approccio

al problema, prima di presentarli però diamo le seguenti definizioni.

**Definizione 3.** *Per "interpretazione" si intende il mappare un concetto astratto (ad esempio una classe di predizione) in un dominio umanamente comprensibile*

Un esempio di dominio umanamente comprensibile sono le immagini (array di pixel) o i testi (sequenze di parole). Un umano può guardare le immagini e leggere i testi, e quindi interpretarli

**Definizione 4.** *Una "spiegazione" è l'insieme delle caratteristiche nel dominio interpretabile che hanno contribuito alla decisione di un modello.*

Una spiegazione può essere, ad esempio, una mappa heatmap (mappa di calore) che evidenzia quali pixel dell'immagine di input sostengono maggiormente la predizione di un modello che fa classificazione.

Per heatmap si intende un certo modo di visualizzare dei valori numerici immagazzinati in una matrice bidimensionale. La matrice viene tradotta in una immagine dove i pixel corrispondono a suoi elementi, e i valori numerici vengono trasformati in intensità di colore .

## 5.1 Metodi di interpretazione

I metodi di interpretazione sono un insieme di tecniche che si utilizzano per mappare concetti appresi da una DNN in un dominio umanamente comprensibile. Una DNN è un insieme di neuroni organizzati in una sequenza di layer, dove ognuno di questi neuroni riceve in input le attivazioni di neuroni appartenenti al layer precedente. L'insieme di tutti questi neuroni compongono congiuntamente una complessa funzione non lineare che mappa dai dati di input all'output. L'insieme dei parametri che definiscono la funzione vengono addestrati con una tecnica che si chiama *discesa del gradiente* (capitolo 1.3.2). Addestrare i parametri significa utilizzare un metodo iterativo che li faccia variare, in modo tale che la funzione si adatti ad imparare la relazione implicita che c'è tra i dati di input e quelli di output. I concetti appresi dalla DNN che si vogliono interpretare, solitamente sono le attivazioni dei neuroni nei layer più profondi della rete. Questi neuroni sono caratterizzati dal fatto che le loro attivazioni dipendono da pattern nell'input molto più astratti e complessi rispetto agli altri neuroni nella

rete, e quello che si vuole fare con i metodi di interpretazione è proiettare questi concetti nello spazio dell'input (dominio interpretabile).

### 5.1.1 Activation maximization

L'activation maximization [24] consiste nel trovare un pattern nello spazio dell'input che massimizzi l'attivazione di un neurone nella rete che si vuole interpretare. Consideriamo un classificatore DNN che mappa dai dati di input  $\vec{x}$  ad un insieme di classi  $(\omega_c)_c$ . I neuroni di output codificano il modello della probabilità della classe  $p(\omega_c|\vec{x})$ . Un prototipo  $\vec{x}^*$  si può trovare ottimizzando:

$$\max_{\vec{x}} [\log(p(\omega_c|\vec{x})) - \lambda \|\vec{x}\|^2] \quad (5.1)$$

Il termine a destra è un regolarizzatore, serve per dare un'importanza maggiore

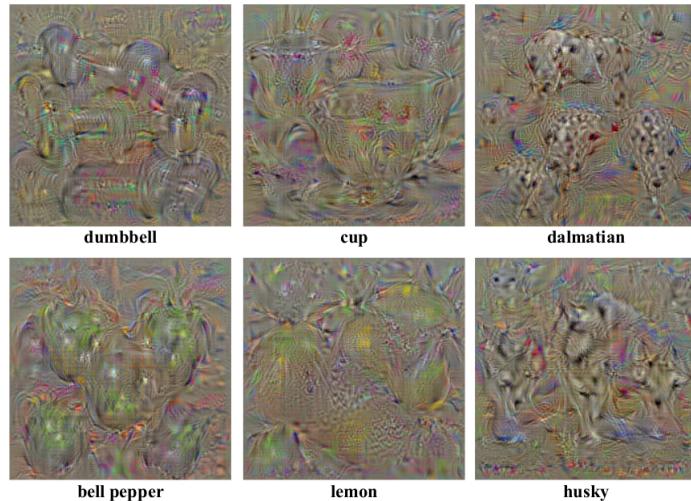


Figura 5.1: Esempio di immagini prodotte dalla procedura activation maximization. Le immagini mostrate vengono generate a partire da una di rumore bianco (pixel inizializzati in modo casuale), con l'ascesa del gradiente i pixel vengono modificati ad ogni step facendo aumentare sempre di più l'attivazione del neurone di output corrispondente ad una determinata classe. L'aspetto interessante dei risultati ottenuti è che possiamo notare la presenza di forme, linee e colori riconducibili agli oggetti rappresentativi di quella classe. Ad esempio per la classe "dalmata" possiamo osservare le macchie nere tipiche di quella razza di cane, si può dire allora che la DNN abbia bisogno di riconoscere quel concetto ("macchie") per quella determinata predizione.

alle  $\vec{x}$  che sono più vicine all'origine. L'operazione descritta è molto semplice da eseguire in quanto le moderne tecniche di implementazione delle reti neurali artificiali permettono di calcolare agevolmente i gradienti delle funzioni. Quin-

di con l'ascesa del gradiente in questo caso si può cercare il risultato voluto (esempio in figura 5.1).

### 5.1.2 DeconvNet

DeconvNet [25] è un'architettura che mappa nell'input space le attivazioni di un qualsiasi neurone nella DNN. Il concetto è lo stesso della activation maximization ma la modalità è diversa. DeconvNet sfrutta le componenti della rete che servono per fare inferenza ma facendole operare in maniera contraria. Parliamo dei layer di convoluzione, di pooling e la ReLU. Come si può vedere in figura 5.2,

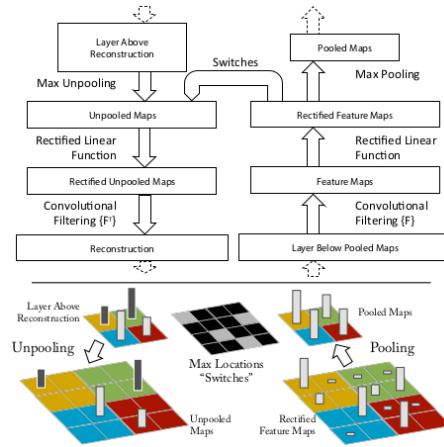


Figura 5.2: Schema di funzionamento di DeconvNet

DeconvNet lavora in maniera parallela alla rete convoluzionale ma in direzione opposta, fornendo ad ogni neurone nella rete un percorso all'indietro per rimappare la propria attivazione nell'input space. In DeconvNet però le componenti della rete operano in questa maniera:

1. Max-unpooling: nella rete di convoluzione l'operazione di max-pooling non è invertibile, tuttavia si può ottenere un'inversa approssimata registrando le posizioni dei massimi locali in fase di inferenza, e riutilizzandole nell'operazione di deconvoluzione.
2. Rectified linear unit: nella rete di convoluzione la funzione di attivazione dei neuroni (ReLU) garantisce che tutte le feature di tutti i layer siano positive. Per ottenere quindi una valida riscostruzione delle feature ad ogni

layer (che dovrebbero essere positive), viene operata una ReLU anche in deconvoluzione.

3. Convoluzione: la rete di convoluzione utilizza filtri (kernel) addestrati per ottenere nuove feature dai layer precedenti. Per invertire questo procedimento, in DeconvNet si utilizza una versione trasposta degli stessi filtri.

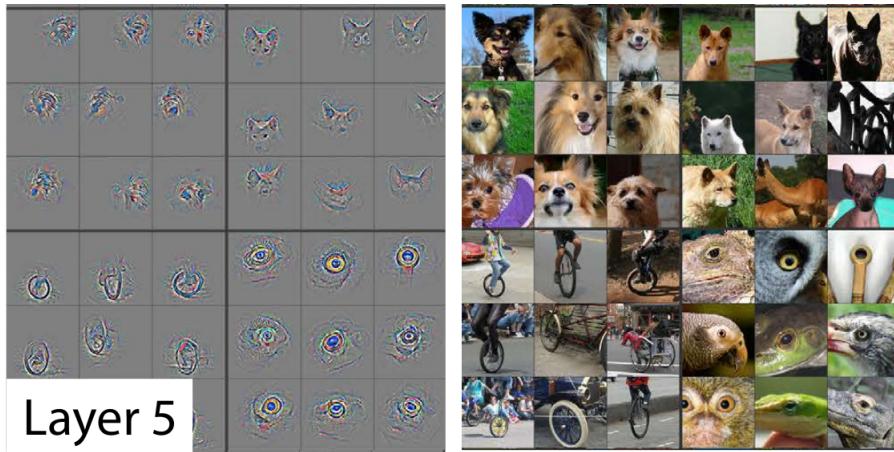


Figura 5.3: A destra vediamo le immagini processate dalla rete e a sinistra invece vediamo il risultato del processo di DeconvNet per alcuni neuroni fortemente attivati in fase di inferenza

Possiamo vedere in figura 5.5 alcune immagini prodotte dall’architettura DeconvNet. Questa metodologia permette di visualizzare con una certa definizione i pattern dell’immagine nell’input space che determinano l’attivazione di un neurone qualsiasi nella rete.

### 5.1.3 Guided Backpropagation

La guided backpropagation [26] è un’evoluzione di DeconvNet, funziona esattamente allo stesso modo ma aggiunge una modifica nell’operazione ReLU durante la fase di ricostruzione. La modifica nasce dalla considerazione che i neuroni la cui attivazione viene annullata in fase di inferenza dalla ReLU, non dovrebbero essere utilizzati in fase di ricostruzione in quanto non sono riusciti a contribuire alla decisione presa dalla rete. In figura 5.4 possiamo vedere la differenza tra le due metodologie, guided backpropagation e DeconvNet. La guided backprop-

b)

Forward pass	<table border="1"><tr><td>1</td><td>-1</td><td>5</td></tr><tr><td>2</td><td>-5</td><td>-7</td></tr><tr><td>-3</td><td>2</td><td>4</td></tr></table>	1	-1	5	2	-5	-7	-3	2	4	$\rightarrow$	<table border="1"><tr><td>1</td><td>0</td><td>5</td></tr><tr><td>2</td><td>0</td><td>0</td></tr><tr><td>0</td><td>2</td><td>4</td></tr></table>	1	0	5	2	0	0	0	2	4
1	-1	5																			
2	-5	-7																			
-3	2	4																			
1	0	5																			
2	0	0																			
0	2	4																			
Backward pass: backpropagation	<table border="1"><tr><td>-2</td><td>0</td><td>-1</td></tr><tr><td>6</td><td>0</td><td>0</td></tr><tr><td>0</td><td>-1</td><td>3</td></tr></table>	-2	0	-1	6	0	0	0	-1	3	$\leftarrow$	<table border="1"><tr><td>6</td><td>3</td><td>-1</td></tr><tr><td>2</td><td>-3</td><td>1</td></tr><tr><td>2</td><td>-1</td><td>3</td></tr></table>	6	3	-1	2	-3	1	2	-1	3
-2	0	-1																			
6	0	0																			
0	-1	3																			
6	3	-1																			
2	-3	1																			
2	-1	3																			
Backward pass: "deconvnet"	<table border="1"><tr><td>0</td><td>3</td><td>0</td></tr><tr><td>6</td><td>0</td><td>1</td></tr><tr><td>2</td><td>0</td><td>3</td></tr></table>	0	3	0	6	0	1	2	0	3	$\leftarrow$	<table border="1"><tr><td>6</td><td>3</td><td>-1</td></tr><tr><td>2</td><td>-3</td><td>1</td></tr><tr><td>2</td><td>-1</td><td>3</td></tr></table>	6	3	-1	2	-3	1	2	-1	3
0	3	0																			
6	0	1																			
2	0	3																			
6	3	-1																			
2	-3	1																			
2	-1	3																			
Backward pass: <i>guided backpropagation</i>	<table border="1"><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>6</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>3</td></tr></table>	0	0	0	6	0	0	0	0	3	$\leftarrow$	<table border="1"><tr><td>6</td><td>3</td><td>-1</td></tr><tr><td>2</td><td>-3</td><td>1</td></tr><tr><td>2</td><td>-1</td><td>3</td></tr></table>	6	3	-1	2	-3	1	2	-1	3
0	0	0																			
6	0	0																			
0	0	3																			
6	3	-1																			
2	-3	1																			
2	-1	3																			

Figura 5.4: Operazione di ReLU nella guided backpropagation

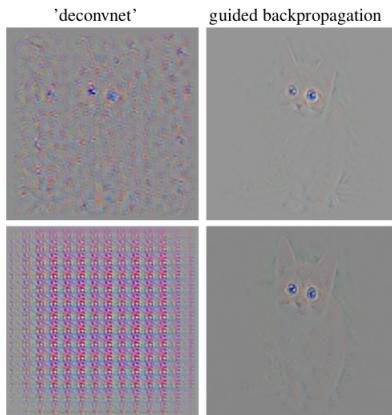


Figura 5.5: Differenza tra le immagini prodotte con DeconvNet e guided backpropagation a partire dallo stesso neurone nella rete. La guided backpropagation produce immagini di interpretazione molto più definite

agation opera azzerando le attivazioni che sono state annullate dalla ReLU in fase di inferenza, indipendentemente dal loro segno.

## 5.2 Metodi di spiegazione

Prendiamo in considerazione una DNN che fa classificazione di immagini come una funzione non lineare che prende di input dei valori  $\vec{x}$  e genera in output un valore  $\omega_c$ , che è l'attivazione di un neurone rappresentativo di una classe  $c$ . Si può vedere  $\omega_c$  come una funzione  $f(\vec{x})$  dell'input e  $\vec{x}$  come un insieme di elementi  $(x_i)_{i=1}^d$  dove  $d$  è la dimensione dell'input. Con i metodi di spiegazione si vuole

assegnare un punteggio di importanza (score)  $R_i$  ad ognuna delle componenti di  $\vec{x}$ . Un esempio è mostrato in figura 5.6

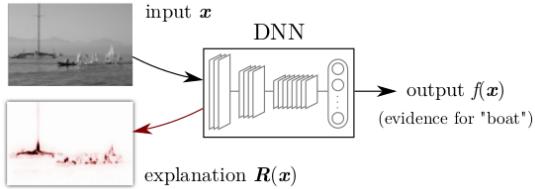


Figura 5.6: Spiegazione della predizione "barca" della DNN per una immagine  $\vec{x}$  data di input, gli  $R_i$  vengono visualizzati con una heatmap

### 5.2.1 Sensitivity analysis

L'idea che sta alla base della sensitivity analysis [27] è molto semplice. Dato un pixel  $x_i$  dell'immagine di input, la sua importanza  $R_i$  per la decisione  $\omega_c$  del modello, è assegnata in base alla derivata parziale di  $f(\vec{x})$  rispetto a  $x_i$

$$R_i(\mathbf{x}) = \left( \frac{\partial f}{\partial x_i} \right)^2 \quad (5.2)$$

L'interpretazione che si può dare a  $R_i$  è questa: un pixel è importante per la classificazione, ovvero per il valore  $\omega_c$ , se al variare della sua intensità corrisponde una distinta variazione di  $f(\mathbf{x}) = \omega_c$ . Nella figura 5.7 possiamo osservare che se rappresentiamo in scala di colori (heatmap) i risultati ottenuti dalla sensitivity analysis, le immagini prodotte non sono molto interpretabili, questo procedimento funziona molto bene se si vuole ricavare un'informazione spaziale sulla posizione nell'immagine dell'oggetto rappresentativo della categoria di predizione, ma nulla di più.

La sensitivity analysis non produce una spiegazione del valore della  $f(\mathbf{x})$ , ma piuttosto una sua variazione, infatti se sommiamo tutti i punteggi  $R_i$  otteniamo

$$\sum_{i=1}^d R_i(\mathbf{x}) = \|\nabla f(\mathbf{x})\|^2 \quad (5.3)$$

Gli  $R_i$  sono una decomposizione della variazione locale di  $f(\mathbf{x})$  misurata con la norma del gradiente.

Ricapitolando la sensitivity analysis è un procedimento in grado di rispondere

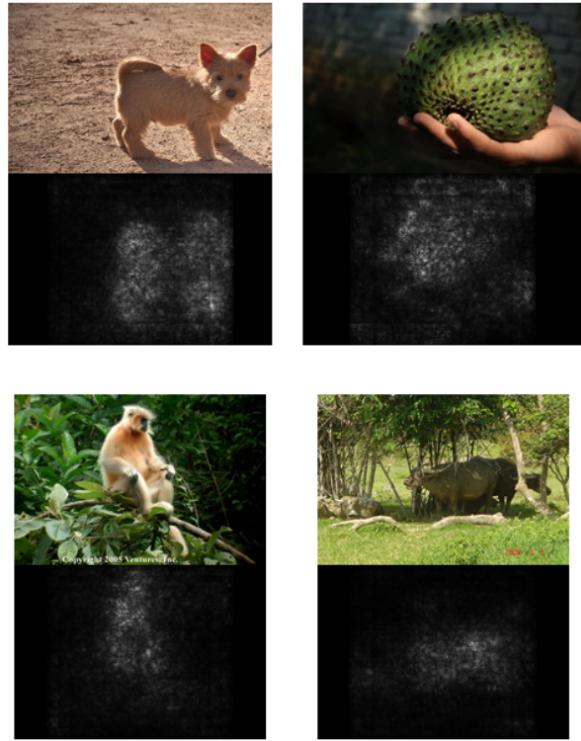


Figura 5.7: Esempi di heatmap prodotte dalla sensitivity analysis

ad una domanda del tipo "che cosa rende questa immagine più o meno una macchina?", piuttosto che "che cosa rende questa immagine una macchina?"

### 5.2.2 Pixel-wise decomposition

Introduciamo una metodologia che permette di visualizzare i contributi dei singoli pixel, utili alla predizione, nelle reti neurali multilayer. L'idea principale

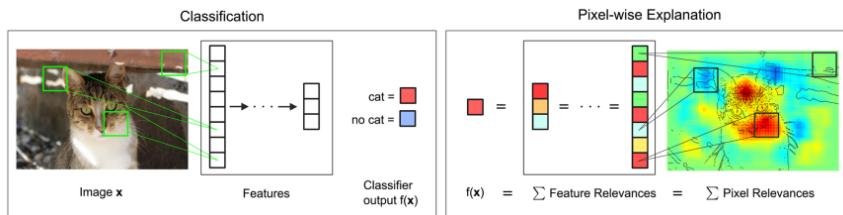


Figura 5.8: Visualizzazione del processo di pixel-wise decomposition

della pixel-wise decomposition [13] è quella di capire il contributo di ogni singolo pixel di un'immagine per la predizione  $f(x)$ , fatta da un classificatore  $f$

su una immagine  $x$ . Lo scopo della metodologia è comprendere, separatamente per ogni immagine  $x$ , quali pixel contribuiscono e in che misura a un risultato di classificazione, positivo o negativo. Un possibile modo consiste nel decomporre la predizione  $f(x)$  in una somma di termini di rilevanza  $R_d$  corrispondenti ai pixel  $x_d$  tanti quanti la dimensione  $V$  dell'input space:

$$f(x) \approx \sum_{d=1}^V R_d \quad (5.4)$$

**Definizione 5.** Da qui in avanti chiameremo i termini  $R_d$  con il nome di "score", punteggio di "rilevanza" o "importanza"

Un'interpretazione qualitativa al segno di  $R_d$  è che se  $R_d > 0$ , esso segnala la presenza di un pattern utile ad una determinata classificazione, mentre se  $R_d < 0$ , esso segnala la presenza di un pattern che tende a diminuire il valore di  $f(x)$ .

La rilevanza  $R_d$  ricavata per ogni pixel di input  $x_{(d)}$  può essere mappata dunque nello spazio dei colori, generando una mappa colorata in cui l'intensità del colore di ogni pixel dipende dalla sua rilevanza, il risultato ottenuto è una heatmap (un esempio di heatmap è presente nella parte destra della figura 5.8). L'interpretazione qualitativa appena descritta sarà un principio fondamentale della metodologia che introdurremo in seguito.

### 5.2.3 Simple taylor decomposition

La taylor decomposition [13] è un metodo che spiega la decisione del modello decomponendo il valore della funzione  $f(\mathbf{x})$  in una somma di score  $R_i$ . Gli score si ottengono identificando i termini dell'espansione di taylor al primo ordine della funzione intorno ad una sua radice (root point), ovvero un punto  $\tilde{\mathbf{x}}$  tale per cui  $f(\tilde{\mathbf{x}}) = 0$ . L'espansione ci permette di scrivere la funzione come:

$$f(\mathbf{x}) = \sum_{i=1}^d R_i(\mathbf{x}) + \mathcal{O}(\mathbf{x}\mathbf{x}^T) \quad (5.5)$$

dove i punteggi di importanza  $R_i$  sono

$$R_i(\mathbf{x}) = \left. \frac{\partial f}{\partial x_i} \right|_{\mathbf{x}=\tilde{\mathbf{x}}} \cdot (x_i - \tilde{x}_i) \quad (5.6)$$

Un esempio di questa metodologia è mostrato in figura 5.9. La simple Taylor

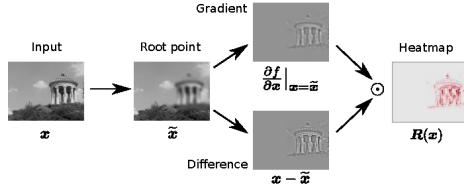


Figura 5.9: Esempio sul funzionamento della simple taylor decomposition

decomposition per quanto possa essere matematicamente ragionevole è troppo dipendente dalla ricerca del root point, che tra tutti quelli esistenti deve essere quello più vicino a  $\vec{x}$ .

#### 5.2.4 Layer-wise relevance propagation - LRP

Introduciamo la Layer-Wise Relevance Propagation (LRP) [13] come una metodologia definita da un insieme di vincoli. Ogni soluzione che soddisfa questi vincoli sarà considerata appartenente a questa metodologia.

La Layer-Wise Relevance Propagation nella sua forma generale presuppone che il classificatore possa essere decomposto in una serie di layer di computazione. Il primo layer è quello di input, ovvero i pixel dell'immagine, l'ultimo layer è l'output a valori reali della predizione del classificatore  $f$ . L' $l$ -esimo layer è modellizzato come un vettore  $z = (z_d^{(l)})_{d=1}^{V(l)}$  di dimensione  $V(l)$ . La layer-wise relevance propagation presuppone che abbiamo un Relevance score  $R_d^{(l+1)}$  per ogni dimensione  $z_d^{(l+1)}$  del vettore  $z$  al layer  $l + 1$ . L'idea è quella di trovare un Relevance score  $R_d^l$  per ogni dimensione  $z_d^{(l)}$  del vettore  $z$  al layer successivo, nella direzione dell'input layer, in modo che venga rispettata la seguente equazione:

$$f(x) = \dots = \sum_{d \in l+1} R_d^{(l+1)} = \sum_{d \in l} R_d^{(l)} = \dots = \sum_d R_d^{(1)} \quad (5.7)$$

Iterando l'equazione (5.7) dall'ultimo layer, che è l'output del classificatore  $f(x)$ , fino al layer di input  $x$ , costituito dai pixel dell'immagine, otteniamo l'equazione (5.4) che desideravamo. Chiamiamo la formula (5.7) 'legge di conservazione', dove si intende la conservazione della rilevanza  $R$  tra un layer e l'altro. Intuitivamente il Relevance score sarà il contributo locale alla predizione di  $f(x)$

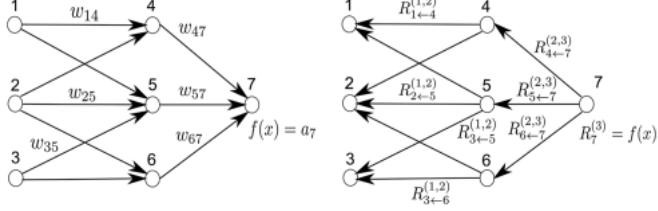


Figura 5.10: Sinistra: classificatore nella forma di rete neurale artificiale durante la fase di predizione. Destra: lo stesso classificatore durante la layer-wise relevance propagation

Vediamolo con un esempio. La parte sinistra della figura 5.10 mostra un classificatore nella forma di una rete neurale artificiale, composta da neuroni e pesi  $w_{ij}$  delle connessioni tra neuroni. Ogni neurone  $i$  ha un output  $a_i$  che dipende dalla funzione di attivazione. Da qui in poi con il termine *input* ci riferiremo al significato di essere input per un neurone nella direzione definita durante la fase di classificazione; ad esempio nella figura 5.10 i neuroni 1 e 2 sono di input per il neurone 4.

L'ultimo layer è costituito da un solo neurone di output, indicato con l'indice 7. Per ogni neurone  $i$  vorremmo calcolare la rilevanza  $R_i$ . Inizializziamo la rilevanza per l'ultimo layer  $R_7^{(3)}$  con il valore della funzione, ovvero  $R_7^{(3)} = f(x)$ . La regola di conservazione della LRP richiede che valgano le seguenti relazioni

$$R_7^{(3)} = R_4^{(2)} + R_5^{(2)} + R_6^{(2)} \quad (5.8)$$

$$R_4^{(2)} + R_5^{(2)} + R_6^{(2)} = R_1^{(1)} + R_2^{(1)} + R_3^{(1)} \quad (5.9)$$

Faremo due assunzioni.

1. Esprimiamo la layer-wise relevance in termini di messaggi  $R_{i \leftarrow j}^{(l,l+1)}$  tra i neuroni  $i \in l$  e  $j \in (l+1)$  che possono essere mandati attraverso la connessione. Questi messaggi, tuttavia, sono mandati da un neurone verso i suoi neuroni di input, nel verso contrario rispetto a quello che accade in fase di predizione, come mostrato nella parte destra della figura 5.10.
2. Definiamo la rilevanza di ogni neurone, eccetto che per il 7 come la somma

ma dei messaggi in entrata:

$$R_i^{(l)} = \sum_{k: i \text{ is input for neuron } k} R_{i \leftarrow k}^{(l,l+1)} \quad (5.10)$$

Ad esempio  $R_3^{(l)} = R_{3 \leftarrow 5}^{(l,l+1)} + R_{3 \leftarrow 6}^{(l,l+1)}$ .

Da qui in avanti ci riferiremo all'equazione (5.10) come "regola di distribuzione forward", dove per distribuzione si intende il modo in cui il valore dello score viene ripartito da un neurone ai neuroni del layer successivo (forward). Sottolineiamo la differenza tra le notazioni  $R_i^l$  e  $R_{i \leftarrow j}^{(l,l+1)}$ , la prima indica lo score vero e proprio di un neurone  $i$  nel layer  $l$ , mentre la seconda, con il doppio indice ad apice e pedice, indica il messaggio, ovvero la porzione dello score che viene ripartita dal neurone  $j$  nel layer  $(l+1)$ , ad un neurone  $i$  al layer precedente  $l$ . Notiamo che il neurone 7 non ha messaggi in ingresso. La sua rilevanza, diversamente da come accade per gli altri neuroni, è definita come  $R_7^{(3)} = f(x)$ . Date le due assunzioni, codificate nella regola di distribuzione forward (5.10), la legge di conservazione di LRP eq:2 può essere soddisfatta dalle seguenti condizioni sufficienti:

$$R_7^{(3)} = R_{4 \leftarrow 7}^{(2,3)} + R_{5 \leftarrow 7}^{(2,3)} + R_{6 \leftarrow 7}^{(2,3)} \quad (5.11)$$

$$R_4^{(2)} = R_{1 \leftarrow 4}^{(1,2)} + R_{2 \leftarrow 4}^{(1,2)} \quad (5.12)$$

$$R_5^{(2)} = R_{1 \leftarrow 5}^{(1,2)} + R_{2 \leftarrow 5}^{(1,2)} + R_{3 \leftarrow 5}^{(1,2)} \quad (5.13)$$

$$R_6^{(2)} = R_{2 \leftarrow 6}^{(1,2)} + R_{3 \leftarrow 6}^{(1,2)} \quad (5.14)$$

In generale questa condizione può essere espressa come:

$$\sum_{i: i \text{ is input for neuron } k} R_{i \leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \quad (5.15)$$

Chiamiamo l'equazione (5.15) come "regola di distribuzione backward", per il motivo analogo alla regola di distribuzione forward. La differenza tra queste due risiede nel fatto che nella prima la somma corre sul layer  $l$  per un neurone fissato al layer  $l+1$ , mentre nella seconda la somma corre sui neuroni del layer  $l+1$  che hanno di input il neurone  $i$  al layer  $l$ . Quando usiamo regola di distribuzione forward (5.10) per definire la rilevanza di un neurone tramite i

messaggi che riceve, la regola di distribuzione backward è una condizione sufficiente affinche valga la legge di conservazione dello score (5.7). Se al primo e secondo membro della regola distribuzione backward (5.15), sommiamo su tutti i neuroni del layer  $(l + 1)$  otteniamo

$$\begin{aligned} \sum_k R_k^{(l+1)} &= \sum_k \sum_{i: i \text{ is input for neuron } k} R_{i \leftarrow k}^{(l,l+1)} \\ &= \sum_i \sum_{k: i \text{ is input for neuron } k} R_{i \leftarrow k}^{(l,l+1)} \\ &= \sum_i R_i^{(l)} \end{aligned}$$

Nel primo passaggio abbiamo semplicemente intercambiato le sommatorie su  $k$  e su  $i$ , nel secondo passaggio invece abbiamo fatto una sostituzione utilizzando la regola di distribuzione forward (5.10). Si può interpretare la condizione (5.15) dicendo che i messaggi  $R_{i \leftarrow k}^{(l,l+1)}$  dal neurone  $k \in (l + 1)$  al neurone  $i \in l$ , sono utilizzati per distribuire la rilevanza  $R_k^{(l+1)}$  del neurone  $k$  ai suoi neuroni di input nel layer  $l$ . Impostiamo le regole di distribuzione forward (5.10) e backward (5.15) come i principali vincoli nel definire la layer-wise relevance propagation. Sempre nei termini dell'esempio mostrato in figura 5.10, ora è necessario trovare una soluzione che segua tale concetto per definire i messaggi  $R_{i \leftarrow k}^{(l,l+1)}$  in accordo ai vincoli che abbiamo definito. Sappiamo che durante la fase di classificazione, un neurone  $i$  manda in input al neurone  $k$  il valore  $a_i w_{ik}$ , a patto che esista una connessione tra  $i$  e  $k$ . Perciò possiamo riscrivere il primo membro delle equazioni (5.11) e (5.12) in modo che possa corrispondere alla struttura del secondo membro delle stesse equazioni, come segue

$$R_7^{(3)} = R_7^{(3)} \frac{a_4 w_{47}}{\sum_{i=4,5,6} a_i w_{i7}} + R_7^{(3)} \frac{a_5 w_{57}}{\sum_{i=4,5,6} a_i w_{i7}} + R_7^{(3)} \frac{a_6 w_{67}}{\sum_{i=4,5,6} a_i w_{i7}} \quad (5.16)$$

$$R_4^{(2)} = R_4^{(2)} \frac{a_1 w_{14}}{\sum_{i=1,2} a_i w_{i4}} + R_4^{(2)} \frac{a_2 w_{24}}{\sum_{i=1,2} a_i w_{i4}} \quad (5.17)$$

La corrispondenza tra il secondo membro delle equazioni (5.11) e (5.12) e il secondo membro delle equazioni (5.16) e (5.17) può essere espressa in generale come

$$R_{i \leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_i w_{ik}}{\sum_k a_h w_{hk}} \quad (5.18)$$

Nonostante la soluzione (5.18) abbia bisogno di essere modificata in modo tale che sia utilizzabile anche quando il denominatore è zero, questo esempio fornisce un'idea di che cosa debba essere il messaggio  $R_{i \leftarrow k}^{(l,l+1)}$ , ovvero la rilevanza di un neurone  $R_k^{(l+1)}$  pesata in modo proporzionale all'input del neurone  $i$  del layer precedente  $l$ .

Dalla formula (5.18) si evince anche che il segno della rilevanza distribuita attraverso il messaggio  $R_{i \leftarrow k}^{(l,l+1)}$ , diventa invertito se il contributo di un neurone  $a_i w_{ik}$  ha un segno differente rispetto alla somma dei contributi di tutti i neuroni di input.

In linea di principio si potrebbero aggiungere altri vincoli alla proprietà di conservazione, riducendo ulteriormente il numero di soluzioni ammissibili. Ad esempio, si potrebbero vincolare i messaggi  $R_{i \leftarrow k}^{(l,l+1)}$ , che risultano dalla ridistribuzione dello score ai nodi dei layer inferiori, in modo tale che siano consistenti con il contributo delle attivazioni di questi ai neuroni ai layer superiori durante la fase di inferenza. Sia  $i$  un neurone nel layer  $l$  che è di input per il neurone  $k$  nel layer  $(l+1)$ , siano  $a_i$  la sua attivazione e  $z_{ik} = a_i w_{ik}$  la sua attivazione pesata, dove  $w_{ik}$  è il peso della connessione tra  $i$  e  $k$ . Se  $i$  ha una maggiore attivazione pesata  $z_{ik}$  rispetto ad altri neuroni che sono di input per  $k$ , allora dovrebbe ricevere una frazione maggiore del relevance score  $R_k^{l+1}$  del nodo  $k$ . In particolare, per tutti i nodi  $k$  che soddisfano  $R_k, \sum_i z_{ik} > 0$ , si potrebbe definire il vincolo  $0 < z_{ik} < z_{i'k} \Rightarrow R_{i \leftarrow k}^{(l,l+1)} <= R_{i' \leftarrow k}^{(l,l+1)}$ .

Ricapitolando, abbiamo introdotto la layer-wise relevance propagation in una rete feed-forward. In base alla definizione data, la rilevanza totale è vincolata a preservarsi tra un layer e l'altro, e la rilevanza totale di un neurone deve essere uguale alla somma dei messaggi in uscita dal neurone stesso. E' importante notare che la definizione *non* è data come un algoritmo o una soluzione. Piuttosto algoritmi diversi, con diverse soluzioni, possono essere ammissibili con questo insieme di vincoli. C'è da notare che nell'esempio fornito (figura 5.10) non sono presenti i bias. Una estensione a reti con i bias, della metodologia mostrata in questo paragrafo, è fattibile ma richiede una riformulazione della legge di conservazione della LRP (lo vedremo nel prossimo paragrafo).

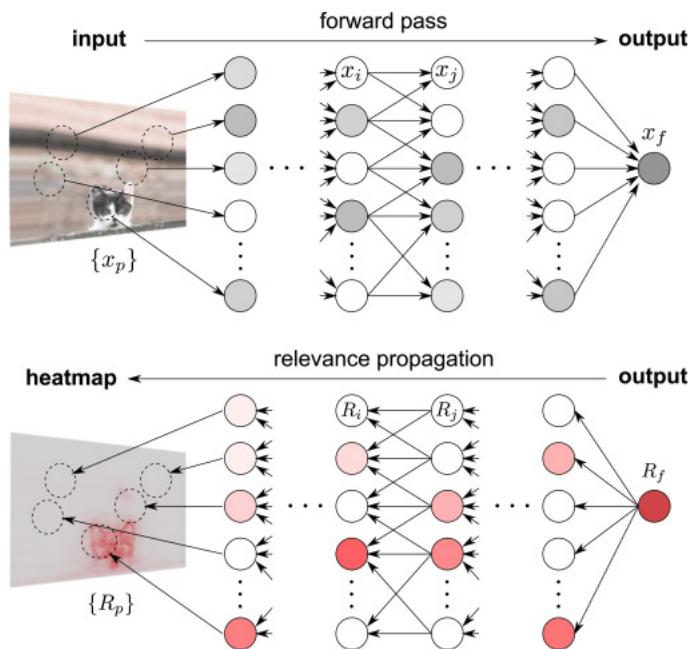


Figura 5.11: Visualizzazione esplicativa della tecnica di layer-wise relevance propagation applicata ad una rete feedforward che fa classificazione di immagini. In alto vediamo la fase di inferenza, l'immagine di input viene processata per ottenere lo score di output relativo ad una certa classe (in questo esempio la classe "gatto"). In basso vediamo la layer-wise relevance propagation, lo score dell'output viene distribuito nel layer che lo precede e così via, fino a raggiungere l'input space. Da notare la direzione dei due processi, durante l'inferenza il calcolo procede dall'input all'output, la LRP invece va dall'output all'input

## Capitolo 6

# LRP per reti multistrato

Abbiamo riassunto nel capitolo 5 i principali metodi di indagine per le reti neurali artificiali. In letteratura esistono tantissime altre tecniche che che fondamentalmente derivano da queste e si basano sugli stessi principi. In questo capitolo e nel prossimo vedremo separatamente i due algoritmi che abbiamo implementato per SegNet, e che per questo motivo meritano una trattazione più approfondita e dettagliata.

La struttura delle reti multistrato (multilayer) consiste in un insieme di neuroni interconnessi raggruppabili in più layer. La combinazione di questi elementi definisce una funzione matematica che mappa il primo layer di neuroni (input) sull'ultimo layer (output). Indichiamo ogni attivazione dei neuroni con  $x_i$  dove  $i$  è un indice per il neurone nel layer. Per convenzione, associamo differenti indici per ogni layer della rete. Indichiamo con " $\sum_i$ " la somma su tutti i neuroni di un certo layer, e con " $\sum_j$ " la somma su tutti i neuroni del layer successivo. Un modo comune di mappare le attivazioni da un layer ad un altro consiste in una combinazione lineare seguita da una funzione non lineare:

$$z_{ij} = x_i w_{ij}, \quad (6.1)$$

$$z_j = \sum_i z_{ij} + b_j, \quad (6.2)$$

$$x_j = g(z_j), \quad (6.3)$$

dove  $w_{ij}$  è il peso che connette il neurone  $i$  al neurone  $j$ ,  $b_j$  è un termine di bias, e  $g$  è una funzione di attivazione non lineare. Da qui in avanti ci riferiremo a  $z_j$  anche con il termine di preattivazione. Questa formulazione per le reti neurali è abbastanza generale da racchiudere dentro di sé una grande varietà di architetture, dal semplice multilayer perceptron [28], alle reti neurali convoluzionali [29].

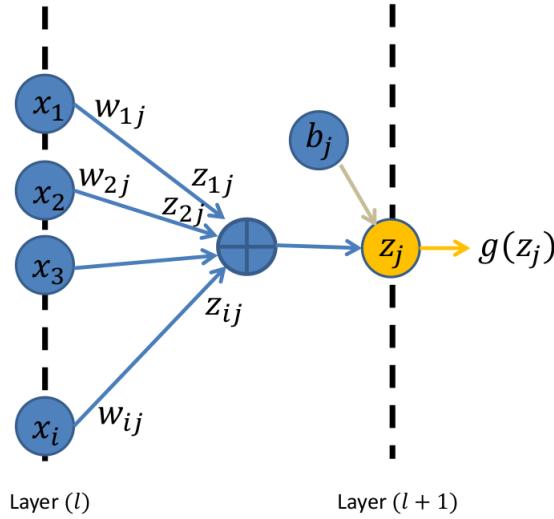


Figura 6.1: Schema rappresentativo della notazione utilizzata in questo capitolo

## 6.1 Alfa-Beta rule

Il metodo funziona in questo modo: conoscendo la rilevanza di un certo neurone  $R_j^{(l+1)}$  per la classificazione  $f(x)$ , si vuole ottenere una decomposizione di tale rilevanza nei termini dei messaggi mandati ai neuroni del layer precedente (sono gli stessi messaggi definiti nel capitolo 5.2.4). Chiamiamo questi messaggi  $R_{i \leftarrow j}$ . In particolare, come espresso nelle regole di distribuzione forward e backward (equazioni (5.10) e (5.15)), la proprietà di conservazione

$$\sum_i R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \quad (6.4)$$

deve valere. Un metodo stabile dal punto di vista della conservazione della rilevanza totale tra un layer e l'altro consiste nel trattare le pre-attivazioni

positive e negative separatamente. Sia  $z_j^+ = \sum_i z_{ij}^+ + b_j^+$  e  $z_j^- = \sum_i z_{ij}^- + b_j^-$  dove "−" e "+" denotano le parti negative e positive di  $z_{ij}$  e  $b_j$ . Le parti positive e negative sono definite così:

$$f^+(x) = \max\{f(x), 0\} = \begin{cases} f(x) & \text{se } f(x) > 0 \\ 0 & \text{altrimenti} \end{cases} \quad (6.5)$$

$$f^-(x) = -\min\{f(x), 0\} = \begin{cases} -f(x) & \text{se } f(x) < 0 \\ 0 & \text{altrimenti} \end{cases} \quad (6.6)$$

La propagazione della rilevanza nella alfa-beta rule è definita come:

$$R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left( \alpha \cdot \frac{z_{ij}^+}{z_j^+} + \beta \cdot \frac{z_{ij}^-}{z_j^-} \right) \quad (6.7)$$

dove  $\alpha + \beta = 1$ .

### 6.1.1 Considerazioni sul bias

In questo paragrafo vediamo in che modo la alfa-beta rule, appena definita, garantisca la validità della legge di conservazione dello score soltanto in assenza di bias. Poniamo ad esempio  $\alpha = \beta = \frac{1}{2}$  nella formula (6.7), e vediamo come diventa in questo caso la regola di distribuzione backward (equazione (5.15)):

$$R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left( \frac{z_{ij}^+}{2z_j^+} + \frac{z_{ij}^-}{2z_j^-} \right) \quad (6.8)$$

sommiamo su tutti i neuroni  $i$  a primo e secondo membro

$$\sum_i R_{i \leftarrow j}^{(l,l+1)} = \sum_i R_j^{(l+1)} \cdot \left( \frac{z_{ij}^+}{2z_j^+} + \frac{z_{ij}^-}{2z_j^-} \right) = \quad (6.9)$$

$$= R_j^{(l+1)} \cdot \left( \frac{\sum_i z_{ij}^+}{2z_j^+} + \frac{\sum_i z_{ij}^-}{2z_j^-} \right) \quad (6.10)$$

ricordandoci la (6.2) abbiamo che

$$\sum_i z_{ij}^+ = z_j^+ - b_j^+ \text{ e } \sum_i z_{ij}^- = z_j^- - b_j^- \quad (6.11)$$

sostituiamo nella (6.10)

$$\sum_i R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left( \frac{z_j^+ - b_j^+}{2z_j^+} + \frac{z_j^- - b_j^-}{2z_j^-} \right) = \quad (6.12)$$

$$= R_j^{(l+1)} \cdot \left( 1 - \frac{b_j^+}{2z_j^+} - \frac{b_j^-}{2z_j^-} \right) \quad (6.13)$$

Quello che volevamo ottenere è la regola di distribuzione backward (equazione (5.15)), ma a partire dalla (6.13) vale soltanto se il bias  $b_j$  è zero. L'interpretazione di quello che viene descritto nella (6.13) è che nella alfa-beta rule si può considerare il bias  $b_j$  come un neurone di input rispetto al neurone  $j$  che ha rilevanza  $R_j^{(l+1)}$ , e quindi è come se ricevesse una parte dello score nella ridistribuzione di questo ai neuroni di input. Questa parte della rilevanza però non viene più ripropagata all'indietro poiché i bias, se considerati come neuroni, non hanno connessioni di input, ma soltanto in output. Perciò la rilevanza assegnata al bias viene dispersa. Questa trattazione si adatta benissimo ai layer di convoluzione delle CNN e anche ai layer di max-pooling. Infatti la convoluzione non è altro che un modo di definire le connessioni tra neuroni appartenenti a layer diversi. In modo analogo il max-pooling si può vedere come una finestra ( $2 \times 2$ ) di connessioni tra 4 neuroni del layer in cui è applicata questa operazione e 1 neurone del layer successivo (generato dal risultato dell'operazione). Per 3 dei 4 pesi la connessione è zero, mentre per il neurone di input che ha l'attivazione più grande il peso è uno.

Un'importante osservazione da fare è che la propagazione della rilevanza è invariante rispetto alla scelta della funzione di attivazione  $g_j$  per calcolare la rilevanza  $R_i$  del neurone  $i$  che è di input per  $j$ . Questa osservazione permette di avere a che fare con funzioni di attivazione non lineari. La funzione  $g_j$  tuttavia eserciterà un'influenza nel calcolare  $R_i$ , perché da  $i$  dipende l'attivazione di  $j$  attraverso  $g_j$ , e quindi del suo score ridistribuito dai layer più profondi fino a  $i$ .



Figura 6.2: Immagini appartenenti al dataset MNIST



Figura 6.3: Esempio di heatmap prodotte dalla layer-wise relevance propagation applicata ad un classificatore di cifre scritte (dataset MNIST). La tecnica riconosce il pattern inibitorio (colore blu) sulla cifra 8: il tratto non è completo, non viene completato il cerchio

## Capitolo 7

# Grad-CAM

Proponiamo ora una tecnica per produrre "spiegazioni visive" per le decisioni di una grande varietà di modelli che si basano sulle reti neurali convoluzionali (convolutional neural network - CNN, capitolo 3), rendendoli più trasparenti. L'approccio Gradient-weighted Class Activation Mapping (Grad-CAM) usa il gradiente di qualunque classe scelta (ad esempio la classe 'gatto' per una rete che fa classificazione di immagini), che fluisce nell'ultimo layer convoluzionale, per produrre una heatmap (mappa di colore) che evidenzia le regioni dell'immagine importanti per la predizione di quella classe.

In molti lavori precedenti a questo, si sostiene che più sono profonde le rappresentazioni in una CNN e più saranno astratti i costrutti visuali che la rete è in grado di modellizzare [30] [31]. Inoltre, feature convoluzionali conservano un'informazione spaziale che invece è persa nei fully-connected layer (capitolo 3.5), così ci può aspettare che l'ultimo layer di convoluzione rappresenti il miglior compromesso tra 'semantica di alto livello' e informazioni spaziali dettagliate. I neuroni in questi layer catturano pattern semantici di informazione, nell'immagine, specifici per le classi (ad esempio le parti di un oggetto).

Grad-CAM utilizza l'informazione del gradiente calcolato nell'ultimo layer convoluzionale (ultimo a partire da quello di input) della CNN per capire l'importanza di ogni neurone per una decisione di interesse. Per ottenere la mappa Grad-CAM  $L_{GradCAM}^c \in R^{u \times v}$  di larghezza  $u$  e altezza  $v$  per una qualsiasi classe  $c$ , prima calcoliamo il gradiente del punteggio  $y^c$  per la classe  $c$  (definizione

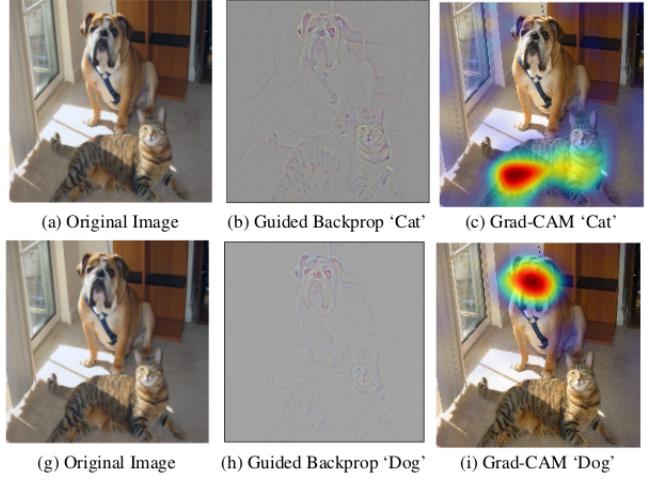


Figura 7.1: Grad-CAM in modo sostanziale può rispondere a domande del tipo: "che cosa fa di questa immagine l'immagine di un gatto?", o riformulata diversamente "quale porzione dell'immagine cattura il concetto di gatto, o di cane?"

2 nel capitolo 3.6), rispetto alla  $k$ -esima feature  $A^k$  di un layer convoluzionale, cioè  $\frac{\partial y^c}{\partial A^k}$ . Il gradiente viene poi mediato su tutta la feature, per ottenere l'importanza di  $A^k$  rispetto alla classe  $c$ , espressa dal coefficiente  $\alpha_k^c$ :

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \quad (7.1)$$

$Z$  indica il numero totale di neuroni che compongono la feature  $A^k$  mentre  $A_{ij}^k$  è l'attivazione del neurone  $(i, j)$  nella  $k$ -esima feature del layer. Dopodiché facciamo una somma pesata sui coefficienti descritti nella (7.1), seguita da una *ReLU* (rectified linear unit, capitolo 3.3), per ottenere:

$$L_{Grad-CAM}^c = ReLU \left( \sum_k \alpha_k^c A^k \right) \quad (7.2)$$

Da notare che la heatmap che si ottiene da questo procedimento è delle stesse dimensioni delle feature ( $u \times v$ ). Applichiamo *ReLU* alla combinazione lineare delle feature  $A^k$  perché siamo interessati soltanto alle feature che hanno un'influenza positiva per la classe di interesse, ovvero che sono composte da attivazioni il cui valore dovrebbe essere aumentato per aumentare  $y^c$ .

## 7.1 Identificare e rimuovere i bias in un dataset

In questa sezione viene mostrato un esempio qualitativo di come si possa utilizzare Grad-CAM per individuare bias nel dataset. Per bias in un dataset si intende una caratteristica nascosta o esplicita nei dati che può influire in modo negativo sulla rete sui quali è addestrata. Se si utilizza un dataset con questa problematica per addestrare ad esempio una CNN che fa classificazione di immagini, essa può fallire in fase di inferenza perchè impara a catturare concetti sbagliati per fare la predizione. In un esperimento descritto nell'articolo dove

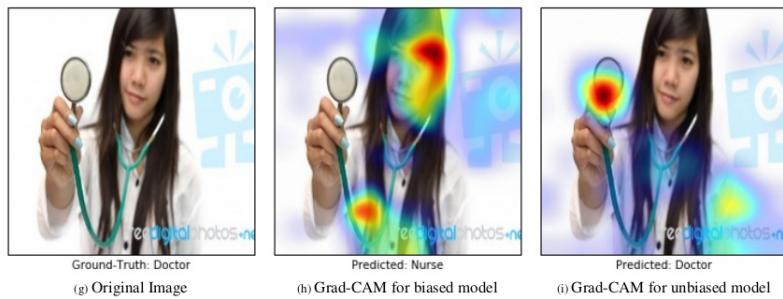


Figura 7.2: Con Grad-CAM possiamo vedere che la rete sembra concentrarsi sul volto della ragazza per fare la predizione "infermiera", ipotizzando quindi una classificazione in base al sesso ("è una infermiera perchè è una donna")

viene presentato Grad-CAM [12], si è addestrata una CNN in modo tale che imparasse a classificare immagini di 'dottori' e 'infermieri', il dataset è stato assemblato scaricando le immagini da internet. All'addestramento è seguita una fase di test, ovvero inferenza sul validation set e poi Grad-CAM sulle stesse immagini.

Dalle immagini rappresentate in figura 7.2 e 7.3 possiamo osservare che la rete

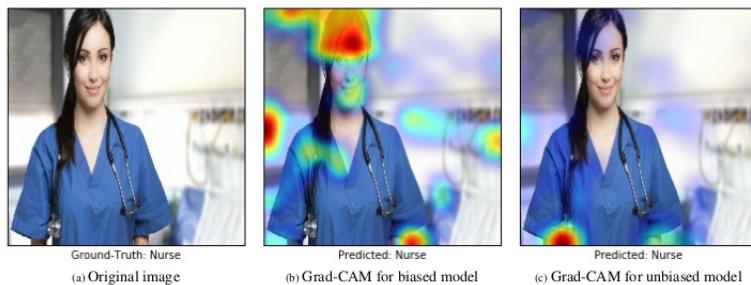


Figura 7.3: L'ipotesi sembra essere confermata da questa predizione sbagliata, sembra sempre che la rete riconosca il volto femminile e che quindi fa l'associazione 'donna'  $\Rightarrow$  'infermiera'

sembra abbia imparato a classificare in base al sesso gli 'infermieri' e i 'dottori'. Il motivo di questo è dovuto al fatto che il dataset creato è composto da troppe immagini di infermieri di sesso femminile, e troppe immagini di dottori di sesso maschile. Grazie a questa osservazione, sempre nello stesso esperimento mostrato nell'articolo di Grad-CAM [12], si è modificato il dataset in modo da riequilibrare le immagini in cui fosse presente un sesso e l'altro. Successivamente è stata riaddestrata la rete e sono stati rifatti i test. Ora la rete ha imparato ad utilizzare concetti più pertinenti a questo task di classificazione, come ad esempio la presenza dello 'stetoscopio' (ultima immagine a destra delle figure 7.3 e 5.11).

## 7.2 Grad-Cam su Segnet

Abbiamo implementato Grad-CAM su SegNet. L'idea nasce dall'osservazione che una rete di segmentazione si può considerare come un insieme di classificatori, uno per ogni pixel.

Per poter dare un supporto teorico all'idea che mostreremo in seguito, siamo andati prima di tutto a vedere in che modo Caffe calcolasse i gradienti. Caffe è il framework di deep learning con il quale è stato implementato SegNet. Abbiamo scoperto che se per una network non è definito un layer di *LOSS* (capitolo 1.3), allora Caffe ne genera uno in automatico che è dato semplicemente dalla somma di tutte le attivazioni dell'ultimo layer. Se indichiamo con *output* l'insieme di tutti i neuroni appartenenti all'ultimo layer della rete prima del softmax, e con *y* le loro attivazioni, possiamo scrivere la *LOSS* in questo modo

$$LOSS = \sum_{n \in output} y_n \quad (7.3)$$

Data l'attivazione di un neurone qualsiasi nella rete, Caffe è in grado di calcolare la derivata parziale della *LOSS* rispetto a quella attivazione

$$\frac{\partial LOSS}{\partial A_{ij}^k} \quad (7.4)$$

dove  $(i, j)$  sono gli indici che girano sulla  $k$ -esima feature bidimensionale  $A^k$  di un certo layer nascosto della rete. Ora riscriviamo la (7.4) sfruttando la regola



Figura 7.4: Esempio di maschera, a destra abbiamo quella relativa alla classe 'macchina'

della catena

$$\frac{\partial \text{LOSS}}{\partial A_{ij}^k} = \sum_{n \in \text{output}} \frac{\partial \text{LOSS}}{\partial y_n^c} \cdot \frac{\partial y_n^c}{\partial A_{ij}^k} \quad (7.5)$$

Sempre in Caffe possiamo intervenire prima del calcolo del gradiente e settare i differenziali  $\frac{\partial \text{LOSS}}{\partial y_n^c}$  con un valore a nostro piacimento e inizializzare la backpropagation utilizzando quei valori, ad esempio 0 o 1 nei nostri esperimenti (otteniamo una maschera come mostrato in figura 7.4). Otteniamo così

$$\frac{\partial \text{LOSS}}{\partial A_{ij}^k} = \sum_{n \in \text{mask}} \frac{\partial y_n^c}{\partial A_{ij}^k} \quad (7.6)$$

dove  $\text{mask}$  è un sottoinsieme di  $\text{output}$  ( $\text{mask} \subseteq \text{output}$ ). Ridefiniamo il coefficiente dell'equazione (7.1) sostituendo al posto del termine di sommatoria quello che abbiamo appena ottenuto

$$\alpha_k^c \rightarrow [\alpha_k^c]_{\text{SegNet}} = \frac{1}{Z} \sum_i \sum_j \sum_{n \in \text{mask}} \frac{\partial y_n^c}{\partial A_{ij}^k} \quad (7.7)$$

riarrangiamo le sommatorie per ottenere

$$[\alpha_k^c]_{\text{SegNet}} = \sum_{n \in \text{mask}} \frac{1}{Z} \sum_i \sum_j \frac{\partial y_n^c}{\partial A_{ij}^k} \quad (7.8)$$

possiamo scrivere

$$\frac{1}{Z} \sum_i \sum_j \frac{\partial y_n^c}{\partial A_{ij}^k} = [\alpha_k^c]_n \quad (7.9)$$

e quindi

$$\alpha_k^c = \sum_{n \in mask} [\alpha_k^c]_n \quad (7.10)$$

Dove gli  $[\alpha_k^c]_n$  sono quelli dell'equazione (7.1), . Questa è una generalizzazione rispetto a quello che avveniva per le semplici reti di classificazione. In quel caso infatti il coefficiente  $\alpha_k^c$  veniva calcolato in base al punteggio della classe  $c$  di interesse, in SegNet invece usiamo il punteggio di tutti i pixel appartenenti a quella classe, ottenendo che il peso della mappa è dato dalla somma dei pesi, calcolati come se considerassi i pixel come singoli classificatori. Si può facilmente tornare alla definizione originale di Grad-CAM in modo semplice usando come maschera un singolo pixel, vuol dire che stiamo considerando un singolo classificatore.

### 7.2.1 Campo recettivo

Nei nostri esperimenti ci siamo resi conto che dato un singolo neurone nel layer di output e dato un qualsiasi layer intermedio nella rete, per come è strutturata SegNet, non tutti i neuroni di quel layer hanno un'influenza sul neurone di output. Questa cosa si può vedere propagando all'indietro il gradiente rispetto al punteggio relativo ad una certa classe  $c$  di un singolo pixel, fino alla feature  $k$  di un layer  $l$ . Consideriamo una LOSS a cui ha contribuito soltanto un neurone  $n'$  nel layer di output, ovvero

$$\frac{\partial LOSS}{\partial y_n^c} = \begin{cases} 1 & \text{se } n = n' \\ 0 & \text{se } n \neq n' \end{cases} \quad (7.11)$$

la (7.5) diventa

$$\frac{\partial LOSS}{\partial A_{ij}^k} = \sum_{n \in output} \frac{\partial LOSS}{\partial y_n^c} \cdot \frac{\partial y_n^c}{\partial A_{ij}^k} = \frac{\partial y_{n'}^c}{\partial A_{ij}^k} \quad (7.12)$$

Ora scegliamo una feature  $k$  in un layer  $l$  nella rete e calcoliamo il gradiente della LOSS rispetto a quella feature e rappresentiamo tramite una mappa in scala di

colori i valori  $\frac{\partial \text{LOSS}}{\partial A_{ij}^k} = \frac{\partial y_{n'}}{a_{ij}^k}$  che otteniamo. Possiamo appunto vedere che ci

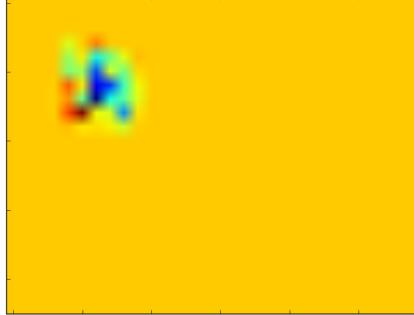


Figura 7.5: Campo recettivo di un pixel, il colore giallo predominante rappresenta i valori nulli

sono soltanto un gruppo di elementi diversi da zero, significa che al variare delle attivazioni di questi neuroni corrisponderà una variazione dell'attivazione del neurone di output, mentre per gli altri neuroni, la derivata  $\frac{\partial y_{n'}}{A_{ij}^k}$  vale zero (quelli gialli in figura). Significa che per tutti questi valori, una variazione del loro valore di attivazione, non fa mutare  $y_{n'}$ . Dati due neuroni in layer consecutivi  $a \in l$  e  $b \in (l+1)$ , definiamo il link  $e_{ab}$  che ha come vertici  $a$  e  $b$  se esiste un peso non nullo (peso nei kernel di convoluzione) che collega i due neuroni. Se  $\frac{\partial y_{n'}}{A_{ij}^k} = 0$  per un neurone in posizione  $(i,j)$  nella feature  $k$  di un layer  $l$  nella rete, che per comodità chiameremo  $x$ , allora vuol dire che non esiste un cammino  $L = \{e_{xx_1}, e_{x_1x_2}, \dots, e_{x_ny'}\}$  che collega  $x$  a  $y'$ , ovvero non c'è modo che  $x$ , o una sua variazione dell'attivazione, possa influenzare  $y'$ . Il concetto di campo recettivo di un pixel è estendibile a più pixel. La figura 7.6 mostra

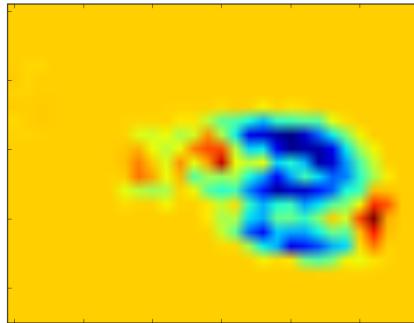


Figura 7.6: Campo recettivo di un gruppo di pixel

il campo recettivo di un gruppo di pixel. Basta intervenire sempre sulla (7.5), cambiando la (7.11). Dato un insieme  $I = n_1, n_2, \dots, n_f$  di neuroni avremo

quindi

$$\frac{\partial \text{LOSS}}{\partial y_n^c} = \begin{cases} 1 & \text{se } n \in I \\ 0 & \text{se } n \notin I \end{cases} \quad (7.13)$$

ottenendo

$$\frac{\partial \text{LOSS}}{\partial A_{ij}^k} = \sum_{n \in \text{output}} \frac{\partial \text{LOSS}}{\partial y_n^c} \cdot \frac{\partial y_n^c}{\partial A_{ij}^k} = \sum_{n \in I} \frac{\partial y_n^c}{\partial A_{ij}^k} \quad (7.14)$$

Abbiamo pensato che nella generalizzazione di Grad-CAM a SegNet fosse giusto ridefinire il modo in cui viene fatta la somma pesata delle feature (equazione (7.2)). Se in una feature ci sono dei neuroni che non contribuiscono minimamente alle attivazioni dei neuroni nell'output che abbiamo selezionato, come abbiamo appena visto, allora non ha senso che questi contribuiscano alla produzione della heatmap  $L_{\text{Grad-CAM}}^c$ . Allora abbiamo sfruttato le mappe dei gradienti (come quelle mostrate in figura 7.5 e 7.6) per produrre delle masche, con le quali filtrare le attivazioni delle feature map  $A^k$ . Il procedimento è

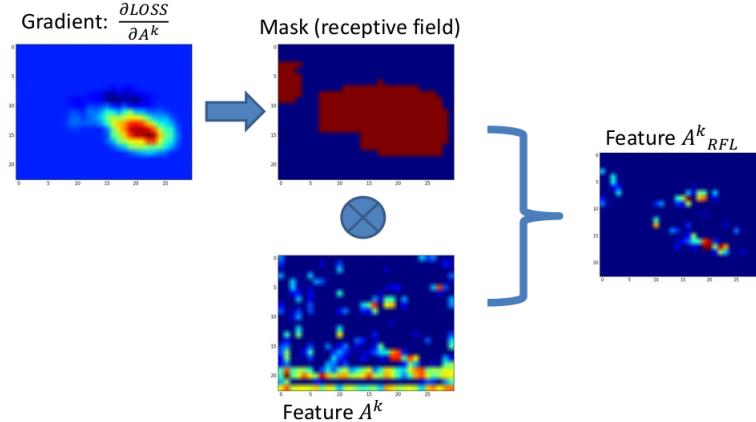


Figura 7.7: Schema del processo di ritaglio delle feature

mostrato in figura 7.7. Abbiamo infine definito le nuove feature  $A^k \rightarrow A_{RFL}^k$  e ridefinito Grad-CAM in questo modo

$$L_{\text{Grad-CAM}}^c = \text{ReLU}(\sum_k [\alpha_k^c]_{\text{SegNet}} \cdot A_{RFL}^k) \quad (7.15)$$

Ora che abbiamo ridefinito Grad-CAM nell'applicazione a questo particolare tipo di rete, vediamo in modo riassuntivo come si può utilizzare. Data una immagine, la segmentazione di output (della rete), e la relativa ground-truth,

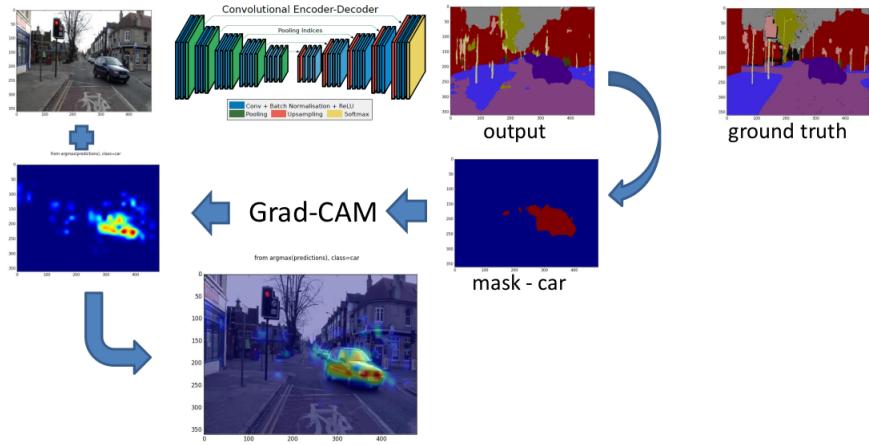


Figura 7.8: Schema del procedimento completo sull'utilizzo di Grad-CAM su SegNet

rimane soltanto da selezionare la parte dell'output che si vuole analizzare, come ad esempio i True Positive di una certa classe, oppure i False Negative, ecc. Si processa il concetto selezionato con l'algoritmo di Grad-CAM e si produce una heatmap. Questa poi si può sovrapporre all'immagine per migliorare la visualizzazione. Uno schema riassuntivo di tutto quanto è mostrato in figura 7.8

# Capitolo 8

## LRP su SegNet

In questo capitolo vediamo il modo in cui abbiamo implementato la layer-wise relevance backpropagation su SegNet. Faremo prima di tutto una breve panoramica sugli aspetti tecnici e alcune considerazioni sulle condizioni di utilizzo di LRP, poi definiremo l'algoritmo di calcolo e infine daremo una descrizione dei principali metodi di utilizzo dell'algoritmo, che saranno la base degli esperimenti finali sullo studio delle reti neurali addestrate sui dataset sintetici.

### 8.1 Aspetti tecnici

Per implementare LRP su Segnet dobbiamo innanzitutto vedere se l'architettura della rete è compatibile con questo algoritmo, ovvero dobbiamo verificare che per tutte le operazioni di SegNet, che mappano l'immagine dall'input all'output, si possa definire il modo di propagare gli score  $R_i$  di ogni neurone durante LRP. Abbiamo visto nel capitolo 4.2 che le operazioni principali di SegNet sono: convoluzione, batch normalization, ReLU, max-pooling, up-sampling e softmax. Per quanto riguarda le operazioni di convoluzione e ReLU la layer-wise relevance backpropagation è ben definita (come abbiamo visto nel capitolo 6.1). L'operazione di batch-normalization (capitolo 4.3) si può aggirare, infatti esiste una procedura, che noi abbiamo utilizzato, che permette di modificare i pesi del modello (parametri della rete: pesi delle connessioni e bias), in modo tale da poter togliere completamente i layer di batch normalization dall'architettura di SegNet. Il risultato di questa procedura permette quindi di ottenere

SegNet con un’architettura diversa, ovvero dove non sono presenti i layer di batch normalization, ma che in fase di inferenza genera gli stessi risultati. Per quanto riguarda l’operazione di up-sampling abbiamo pensato che fosse automatico e giusto corrisponderle un’operazione di max-pooling in fase di layer-wise relevance propagation. I dettagli di quest’ultima considerazione li vedremo nei paragrafi successivi. Infine abbiamo tolto il softmax dalla rete in quanto non era utile per i nostri scopi.

Prima di procedere con la trattazione poniamo l’accento su una considerazione importante. La layer-wise relevance propagation appartiene all’insieme di metodologie definite con il nome di pixel-wise decomposition (capitolo 5.2.2), e quindi è una tecnica che permette di assegnare un punteggio di importanza ad ogni singolo pixel dell’immagine. L’importanza è riferita ad un qualsiasi concetto della rete inteso come attivazione di un neurone, che può essere di output e quindi adibito alla classificazione di una certa categoria di classificazione, oppure appartenente ai layer nascosti della rete. Gli score (punteggi di importanza) si possono infine visualizzare con delle heatmap, che è lo scopo di questa procedura, ovvero mappare un concetto appreso dalla rete in un dominio umanamente comprensibile (metodi di spiegazione 5.2). La ridistribuzione dello score tra un layer e l’altro (formula 6.7) dipende sia dai pesi addestrati della rete che dalle attivazioni dei neuroni ottenute processando un’immagine. Quello su cui vogliamo porre l’accento è quindi che ogni heatmap prodotta da LRP dipende dall’immagine che è stata processata, dai pesi della rete e dal concetto che si è scelto di mappare nell’input space.

## 8.2 Inizializzazione dello score

Prima di dare una struttura all’implementazione di LRP su SegNet facciamo qualche considerazione sul layer di output. Per layer di output intendiamo l’insieme dei neuroni le cui attivazioni sono date dall’ultima operazione di convoluzione della rete (al termine del decoder). Essi sono organizzati in un blocco tridimensionale di dimensioni  $(c \times h \times w)$ , dove  $h$  e  $w$  sono l’altezza e la larghezza dell’immagine segmentata (corrispondono esattamente a quelle di input), mentre  $c$  è il numero delle classi con cui la rete è in grado di categorizzare ogni pixel.

Ad esempio, prendiamo in considerazione una rete addestrata a distinguere 10 classi diverse, il suo blocco finale sarà di dimensioni  $(10 \times 360 \times 480)$ . L'attivazione del neurone  $(c', x, y)$  corrisponde al punteggio prima del softmax della classe  $c'$  per il pixel  $(x, y)$ . Possiamo utilizzare LRP per mappare l'attivazione di uno di questi neuroni nell'input space, ma nulla ci vieta di sceglierne più di uno. Per poter inizializzare LRP dobbiamo prima di tutto assegnare uno score ad ogni singolo neurone del layer di output. Questa scelta è completamente arbitraria ma deve essere interpretabile affinché abbia senso, ad esempio se scelgo di assegnare  $R_{i=(3,15,18)} = 1$  e  $R_{i \neq (3,15,18)} = 0$  sto cercando di rispondere alla domanda "quale parte dell'immagine è importante per categorizzare il pixel  $(15, 18)$  con la classe 3?".

Abbiamo appena descritto un esempio di come si potrebbero inizializzare gli score del layer di output di SegNet dandone un'interpretazione. In linea di principio non esistono limiti in LRP sull'inizializzazione degli score, l'algoritmo funziona indipendentemente dal fatto che i valori siano positivi, negativi o nulli, o che i valori non nulli siano uno o più di uno. Nei nostri esperimenti ci siamo limitati e vincolati soltanto ai tipi di inizializzazione che fossero interpretabili. Separiamo l'operazione di "inizializzazione", che è completamente arbitraria, in "scelta" e "assegnazione". Per "scelta" intendiamo proprio scegliere i concetti da mappare nell'input space, intesi come le attivazioni dei neuroni. Negli esperimenti ci siamo limitati ai seguenti tipi di scelta che sono facilmente interpretabili

1. scegliere un singolo neurone in base alla classe e al pixel di interesse
2. scegliere un gruppo di neuroni appartenenti alla stessa classe

Il primo tipo di inizializzazione è banale da giustificare, una rete di segmentazione si può vedere come un insieme di classificatori, uno per ogni pixel, per questo motivo scegliere soltanto un neurone ci riconduce al caso elementare. Scegliere più pixel appartenenti alla stessa classe significa produrre un'heatmap dove l'importanza di un pixel dipende da più di un pixel nell'output segmentato. Se un pixel è molto importante per la classificazione di tanti tra quelli di output che sono stati scelti avrà uno score più elevato, rispetto ad un pixel che è importante o meno per la classificazione di pochi o nessun pixel in output.

Abbiamo detto che una volta scelti i neuroni "assegnamo" a loro uno score sen-

za specificarne il valore. Nei nostri esperimenti, nel caso di una selezione di un gruppo di pixel, abbiamo usato tre tipi diversi di assegnazione:

1.  $R_i = y_i$
2.  $R_i = P(i)$
3.  $R_i = 1$

L'indice  $i$  è riferito ai neuroni "scelti" mentre  $P(i)$  indica la probabilità prima del softmax della classe  $c$  per il pixel  $(x, y)$  se esplicitiamo  $i$  con l'indicazione  $i = (c, x, y)$ . In questo caso abbiamo pensato che fosse più utile fare un esperimento per verificare le differenze tra i risultati ottenuti con un'assegnazione o l'altra, piuttosto che cercare di darne una interpretazione a priori. Abbiamo visto che le heatmap prodotte sono praticamente identiche, questo è probabilmente dovuto alla forte correlazione che c'è tra i valori dei tre tipi di assegnazione. Nelle esecuzioni di LRP dove facciamo propagare lo score di un singolo neurone, abbiamo sempre assegnato il valore 1. Qualsiasi altro valore determina la produzione di heatmap completamente identiche, infatti i valori di tutti gli score propagati fino al layer di input, vengono congiuntamente riscalati se cambio il valore dello score del neurone di output.

### 8.3 Algoritmo

Vediamo nel dettaglio l'algoritmo da noi implementato.

1. *LoadWeights* indica l'operazione di caricamento dei pesi  $W$  della rete, i valori dei pesi dipendono dai parametri di addestramento della rete e dal dataset utilizzato.
2. *InitializeConcept* indica l'operazione descritta nel capitolo 8.2, essa dipende dall'output della rete rispetto all'immagine processata e dalla sua segmentazione "vera". Ad esempio posso scegliere di inizializzare l'output con il punteggio prima del softmax dei "true positive" di una certa classe.
3. *InputLayer* è una funzione che restituisce il nome del layer precedente a quello passato come argomento, ad esempio  $\text{InputLayer}(\text{conv1}_1D) \rightarrow \text{conv1}_2D$ , ovvero  $\text{conv1}_2D$  è il layer che precede  $\text{conv1}_1D$ .

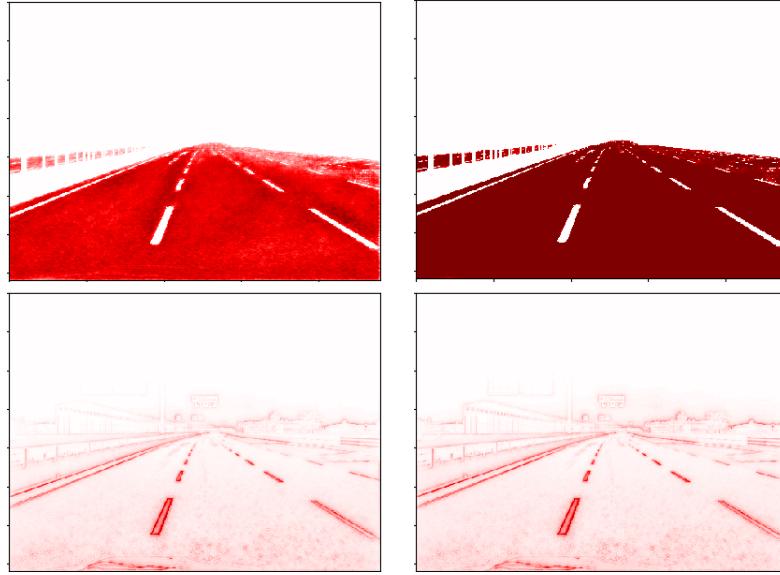


Figura 8.1: In alto abbiamo due diversi tipi di assegnazione: punteggio prima del softmax a sinistra e probabilità a destra. In basso abbiamo le relative heatmap. Da notare che le heatmap prodotte dai due diversi tipi di assegnazione sono molto simili

4. *Pooling* è l'operazione inversa dell'up-sample.
5. *UpSample* è l'operazione inversa del max-pooling, essa utilizza gli stessi indici dell'operazione MAX fatta in inferenza.
6. *LRP* è l'operazione di layer-wise relevance propagation, viene fatta considerando una feature alla volta del layer da cui viene ridistribuito lo score.

## 8.4 heatmap

LRP produce una matrice di dimensioni  $(3 \times 360 \times 480)$  dove 3 sono i canali del colore e le altre due dimensioni sono rispettivamente l'altezza e la larghezza dell'immagine. Quindi non abbiamo uno score per ogni pixel ma tre, uno per ogni canale. Abbiamo seguito il consiglio dato in [32] che suggerisce di fare la somma degli score dei tre canali per ogni pixel per ottenerne il definitivo punteggio di importanza. Dato un pixel  $i$  nell'input lo score  $R_i$  si ottiene facendo

$$R_i = R_i^R + R_i^G + R_i^B \quad (8.1)$$

```

Input : Image  $I$ , Ground-Truth  $GT$ , Weights  $W$ 
Output:  $Score$  for all Segnet layers

 $SegNet_W \leftarrow \text{LoadWeights}(SegNet, W);$ 
 $\text{Output } O \leftarrow SegNet_W(I);$ 
 $Score[conv1_1] \leftarrow \text{InitializeConcept}(O, GT);$ 

for  $LAYER$  in  $SegNet$  layers do
     $Score[\text{InputLayer}(LAYER)] \leftarrow 0;$ 
    if  $LAYER$  is an up-sample layer then
         $Score[\text{InputLayer}(LAYER)] \leftarrow \text{Pooling}(Score[LAYER]);$ 
    else if  $LAYER$  is pooling layer then
         $Score[\text{InputLayer}(LAYER)] \leftarrow \text{UpSample}(Score[LAYER], MAX\_INDICES);$ 
    else //  $LAYER$  is convolutional layer
        for  $feature$  in  $Score[LAYER]$  do
             $Score[\text{InputLayer}(LAYER)] \leftarrow Score[\text{InputLayer}(LAYER)] + \text{LRP}(feature);$ 
        end
    end
end

```

dove gli apici  $R$ ,  $G$ , e  $B$  si spiegano da soli. Una volta ottenuti gli  $R_i$ , li traduciamo in scala di colore per poter osservare visivamente il risultato. Per definire la scala dobbiamo prima calcolarne gli estremi

$$ext_{\pm} = \pm \max_i(|R_i|) \quad (8.2)$$

Dato un pixel  $i$ , se  $R_i = ext_-$  verrà colorato di blu, se  $R_i = ext_+$  verrà colorato di rosso, se  $R_i = 0$  verrà colorato di bianco. Per tutti gli altri avremo le transizioni dal blu al bianco per i valori negativi, e transizioni dal rosso al bianco per quelli positivi. Un esempio di heatmap è mostrato in figura 8.2

layer: conv1\_1\_D class: road alfa e beta: 2 -1 -- sum

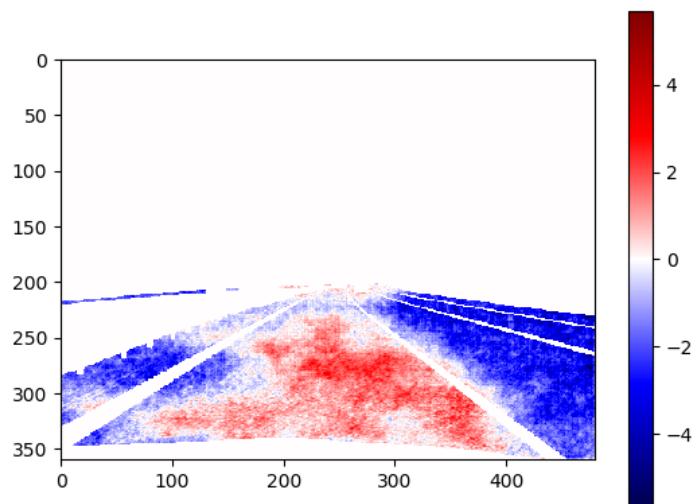


Figura 8.2: Esempio di heatmap prodotta con LRP

# Capitolo 9

## Esperimenti

### 9.1 Dataset sintetici e reali

Nei nostri esperimenti abbiamo addestrato SegNet con Mapillary [33] e Camvid [34], essi sono il frutto di un lavoro di ricerca precedente che mette a disposizione della comunità scientifica un insieme di dati standardizzato in modo che i risultati ottenuti siano confrontabili. Le immagini che li compongono sono state acquisite posizionando una videocamera sul cofano di un automobile. Tutte le immagini di Camvid e Mapillary rappresentano quindi ambienti stradali e per ogni singola immagine si ha anche la relativa segmentazione. Oltre ad avere a disposizione questi due dataset, abbiamo avuto la possibilità di generare dei dataset sintetici tramite la piattaforma di creazione di ambienti virtuali Unreal. Abbiamo generato con Unreal un percorso autostradale estremamente realistico e ricco di elementi. Nell'ambiente abbiamo simulato il moto di una automobile e le immagini le abbiamo catturate attraverso una finestra di acquisizione ad imitare una fotocamera posta sul cofano (proprio come per i dataset Mapillary e Camvid). Nei nostri primi esperimenti non avevamo a disposizione Unreal, ma Synthia [35]

Le segmentazioni delle immagini di Camvid e Mapillary sono il prodotto di un lavoro umano, significa che una o più persone hanno speso il proprio tempo e le proprie energie per classificare ogni singolo pixel dell'immagine. Riteniamo che sia molto importante sottolineare questo aspetto, perché esso è alla base del

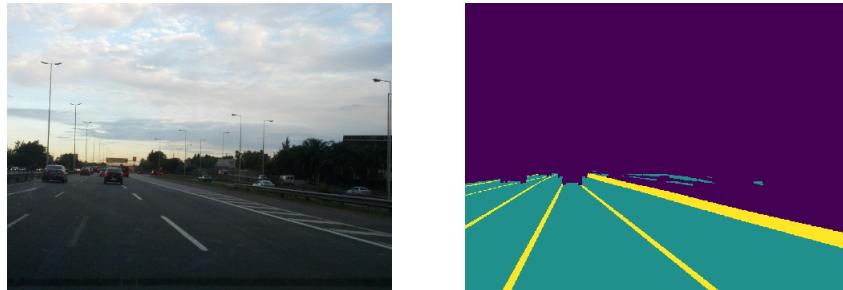


Figura 9.1: Esempio di una immagine del dataset di Mapillary e la relativa segmentazione

motivo per cui è desiderabile avere la possibilità di generare un dataset sintetico atto ad imitare la realtà. Le segmentazioni delle immagini generate con Unreal, diversamente da quelle di Mapillary e Camvid, derivano invece da un processo automatizzato e computerizzato. Per ottenerle è bastato ripetere le simulazioni del moto dell'automobile facendo in modo che le uniche sorgenti di luce fossero gli oggetti stessi nello scenario, che non ci fossero effetti di riflessione, diffusione e diffrazione della luce, e che il colore degli oggetti fosse monocromatico. Un esempio è mostrato nell'immagine a destra della figura 9.2.

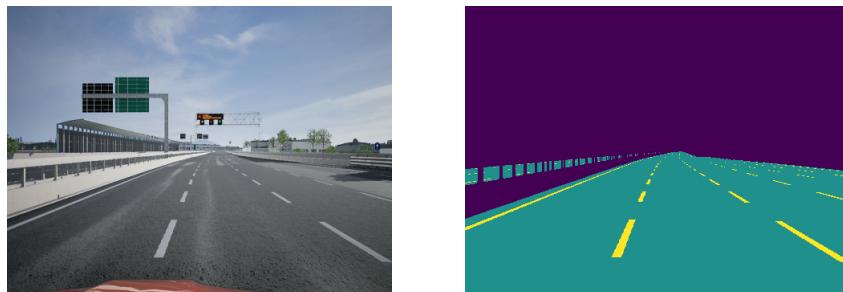


Figura 9.2: Esempio di una immagine del dataset generato con Unreal e la relativa segmentazione

Nei nostri esperimenti non abbiamo utilizzato tutte le classi di segmentazione disponibili, ma soltanto tre di esse: strada, strisce (segnaletica orizzontale) e sfondo (background, in questa categoria sono incluse tutte le altre classi che non appartengono alle prime due). Tutti i dataset a nostra disposizione li abbiamo riorganizzati e divisi in training set, validation set e test set. Da qui in avanti ci riferiremo al dataset generato con il framework Unreal con "dataset di Unreal" o "Unreal".

## 9.2 Dettagli dell'addestramento

In questa sezione elenchiamo l'insieme dei parametri che abbiamo utilizzato per addestrare SegNet

- dimensione del batch:  $m = 4$
  - learning rate:  $lr = 0.001$
  - momentum:  $\gamma = 0.9$
  - weight dacay:  $\lambda = 0.0005$
  - dimensione del training set:  $n = 12995$  per Mapillary,  $n = 10000$  per Unreal
  - numero di iterazioni di SGD: 100k per Unreal e 300k per Mapillary

Il significato di questi parametri è spiegato nel capitolo 1.3, le notazioni sono le medesime

### 9.3 Visualizzazione delle feature più importanti tramite LRP

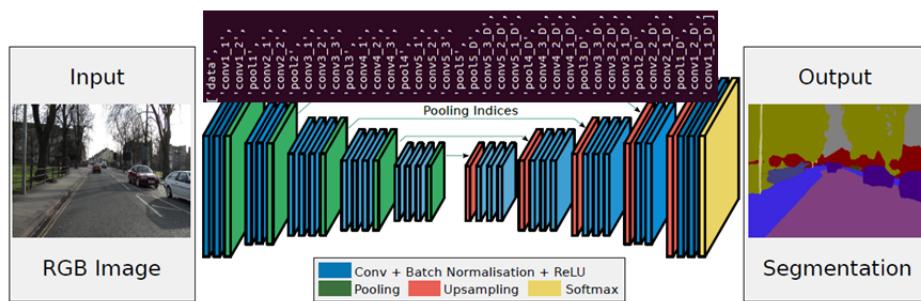


Figura 9.3: In questa figura sono mostrati i nomi dei layer di SegNet

In questo capitolo mostreremo i risultati ottenuti dagli esperimenti che avevano come obiettivo quello di capire, nel modo più dettagliato possibile, quali fossero le parti dell'immagine più importanti per la classificazione dei pixel rappresentativi della classe "strada" da parte di SegNet. Ogni metodologia

utilizzata e ogni considerazione fatta è ripetibile e generalizzabile anche alle altre due classi "strisce" e "background". Vediamo prima di tutto quali sono le componenti dell'apparato sperimentale.

1. Rete SegNet addestrata sul training set di Unreal
2. Immagine e relativa segmentazione prese dal validation set di Unreal
3. Architettura LRP ( $\alpha = 1, \beta = 0$ )

## 9.4 Somma delle feature

Per prima cosa abbiamo processato l'immagine con SegNet per ottenere in output la relativa segmentazione. Poi abbiamo confrontato l'output con la ground-truth per selezionare i true-positive della strada (operazione di inizializzazione descritta nel capitolo 8.2), ovvero l'insieme di tutti i pixel che sono stati classificati "strada" in modo esatto. Abbiamo inizializzato LRP (capitolo 8.2) assegnando come score ai neuroni scelti i loro punteggi (definizione 2, capitolo 3.6). Una visualizzazione del modo in cui abbiamo inizializzato il layer di output si trova nell'immagine in alto a sinistra della figura 9.4. Dopodichè abbiamo usato LRP per calcolare gli score layer per layer fino a quello di input. Ogni volta che eseguiamo LRP conserviamo in memoria lo score distribuito su tutte le feature di tutti i layer di SegNet. Un primo modo per vedere in che modo avviene la ridistribuzione dello score layer per layer è quello di produrre delle heatmap date dalla somma di tutte le feature che appartengono allo stesso layer. Dato un layer di dimensioni  $(c \times h \times w)$  in cui sono già stati assegnati tutti gli score  $R_{(k,y,x)}$  (le tre coordinate a pedice indicano la posizione del neurone nel layer), le heatmap  $R^{heatmap}$  prodotte avranno dimensioni  $(h \times w)$  dove ogni elemento è dato da:

$$R_{(y,x)}^{heatmap} = \sum_{k=1}^c R_{(k,y,x)} \quad (9.1)$$

In figura 9.4 sono mostrate soltanto 12 heatmap delle 37 ottenute (sono 37 i layer totali di Segnet), e già con queste possiamo fare delle osservazioni interessanti:

1. Lo score sembra conservare l'informazione spaziale, ovvero esso è distribuito principalmente nella parte bassa del layer perché nell'immagine di input la strada si trova posizionata giustamente in basso

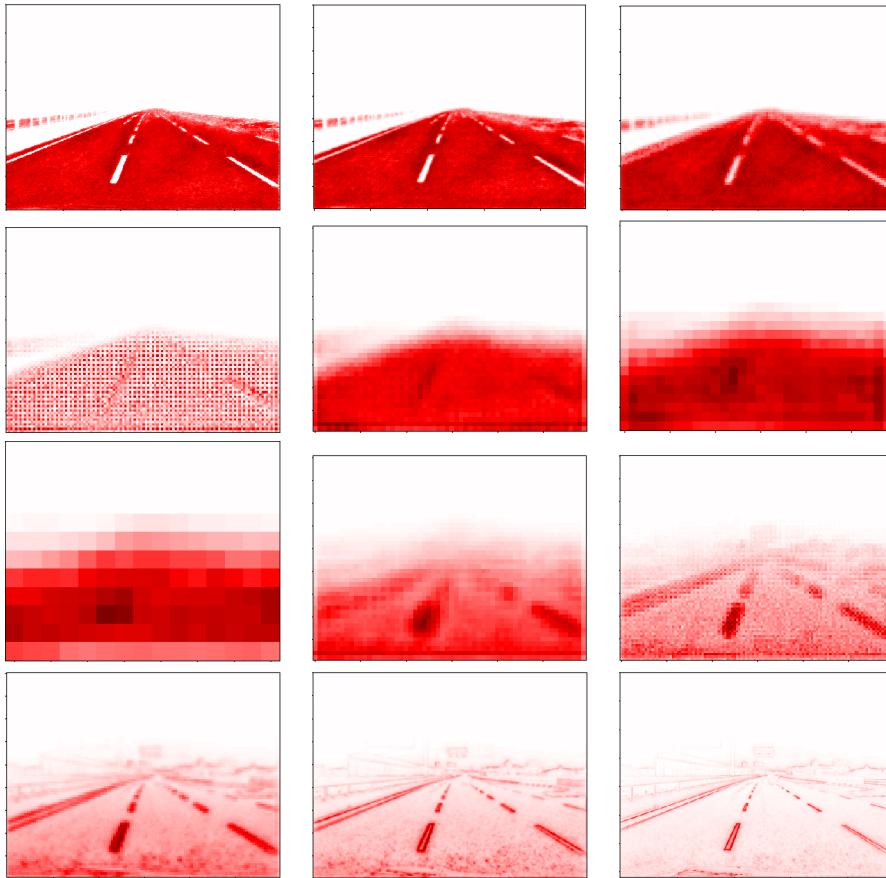


Figura 9.4: In questa immagine si può vedere la propagazione layer per layer (layer-wise) dello score a partire dal layer di output "conv11D" in alto a sinistra fino al layer di input "data" in basso a destra, secondo la alfa-beta rule (equazione 6.7).

2. Una volta che lo score è distribuito sul layer di input si può osservare che esso è estremamente concentrato sulle strisce della strada piuttosto che sull'asfalto, sembra che questo elemento sia fondamentale per la classificazione di tutti quei pixel

## 9.5 Score distribuito sulle singole feature

Nella seconda parte dell'esperimento abbiamo recuperato l'informazione persa nel calcolo dell'equazione 9.1 e siamo andati a vedere gli score di ogni layer distribuiti sulle singole feature sempre attraverso delle heatmap che chiameremo  $R_k^{heatmap}$ . Per poterle confrontare e capire quali feature catturano più score e

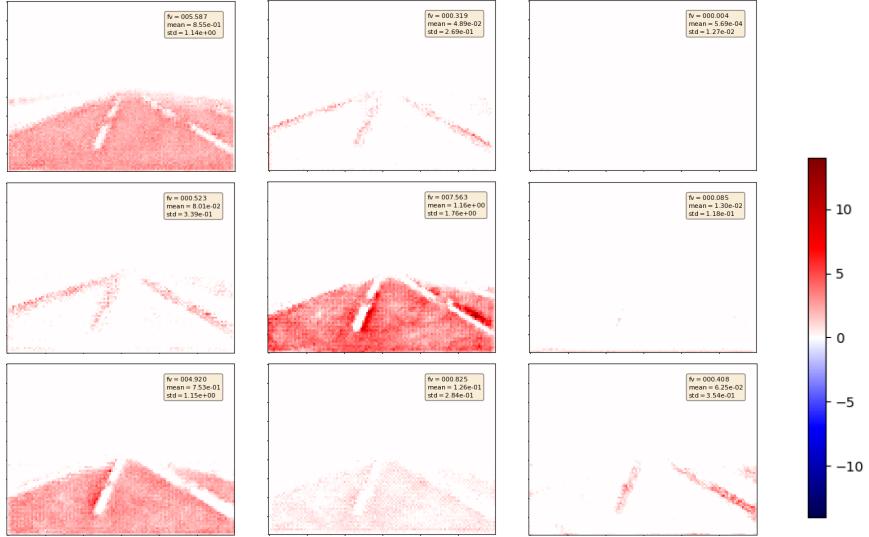


Figura 9.5: In questa figura possiamo osservare lo score distribuito su diverse feature del layer "conv3\_1D". Da notare che tutte le heatmap utilizzano la medesima scala di colori, in questo modo possiamo confrontarle nei termini per cui possiamo dire che alcune feature sono molto importanti per la classificazione del gruppo di pixel che abbiamo selezionato all'inizio dell'esperimento, e altre meno, tanto che poco o niente si riesce a visualizzare, come nelle immagini in alto a destra e a destra.

quali meno abbiamo fissato una scala di colore globale i cui estremi sono calcolati in questo modo:

$$ext_{\pm} = \pm \max_{(k,y,x)} (|R_{(k,y,x)}|) \quad (9.2)$$

Ovvero gli estremi sono calcolati prendendo in considerazione il neurone con lo score più distante dallo zero in tutte le feature del layer.

Possiamo vedere nella figura 9.5 il risultato della visualizzazione appena descritta per nove delle 128 features di "conv3\_1D". Abbiamo fatto le seguenti osservazioni:

- Ci sono delle feature in cui è maggiore la quantità di score rispetto alle altre, la distinzione è abbastanza netta e si nota soltanto usando una scala di colori globale (equazione 9.2)
- Le feature non solo si distinguono per la quantità di score ma anche per la forma della distribuzione dello score: in alcune lo score è concentrato in pochi neuroni situati in corrispondenza delle strisce bianche, in altre è distribuito in maniera più estesa e uniforme, in zone della feature

corrispondenti all'asfalto della strada

I layer di convoluzione sono caratteristici per la capacità di conservare gran parte dell'informazione spaziale dell'immagine che viene processata. Che cosa significa ciò? Significa che l'attivazione di un neurone in una posizione  $(x, y)$  di una feature qualsiasi di un layer nascosto della rete, dipende da una zona localizzata dell'immagine, centrata in  $(x', y')$  che sono le medesime coordinate riscalate alle dimensioni dell'input space. Le heatmap delle feature che abbiamo ottenuto, siccome dipendono dagli score e quindi dalle attivazioni dei neuroni, riflettono in una certa misura qual è la parte dell'immagine che le rende tali. Ad esempio, per la feature che ha lo score distribuito in maniera netta sulle strisce bianche, si può dire che essa catturi in prima approssimazione proprio tale concetto, ovvero le attivazioni di tale feature si attivano in maniera netta e distinta quando riconoscono quel determinato pattern nell'immagine. Se osserviamo tutte le feature di un layer possiamo già intuire quali siano i concetti che la rete utilizza per predire la classificazione di tutti i pixel presi in esame.

## 9.6 Feature values

Dopo aver fatto delle considerazioni sul ritenere più o meno una feature importante rispetto ad un'altra, ci è sembrato lecito dare un numero a questo attributo. L'abbiamo chiamato "feature value". Dato un layer di dimensioni  $(c \times h \times w)$ , la feature value  $f_v^k$  della feature  $k$  si misura in questo modo:

$$f_v^k = \frac{\sum_x \sum_y R_{(k,y,x)}}{\sum_k \sum_x \sum_y R_{(k,y,x)}} \cdot c \quad (9.3)$$

Il primo fattore è semplicemente la frazione di score distribuita sulla feature, rispetto a quella totale del layer, il secondo fattore invece è stato aggiunto perché ci permette di confrontare feature appartenenti a layer diversi della rete. Senza il termine  $c$  la feature value rappresenterebbe semplicemente un valore percentuale, ed esso sarebbe significativo soltanto in relazione al numero di feature del layer di appartenenza. Consideriamo ad esempio una feature value  $f_v^k = 0.1$  calcolata senza il fattore  $c$ , questo numero dice che il 10% dello score del layer è distribuito su tale feature, ma tale informazione ha un significato

diverso se in un layer ci sono 5 feature piuttosto che 50. Avere un decimo dello score totale che è stato frazionato in cinque parti è molto meno significativo rispetto ad avere un decimo dello score totale che è stato frazionato invece in 50 parti. Se moltiplichiamo per  $c$  i due valori otteniamo  $f_{v1}^k = 0.5$  e  $f_{v2}^k = 5$ .

Dopo aver assegnato un numero di importanza ad ogni feature sulla base dello score che gli appartiene, abbiamo fatto un grafico di questi valori per ogni layer della rete. Possiamo vedere in figura 9.6 in che modo sono distribuiti questi

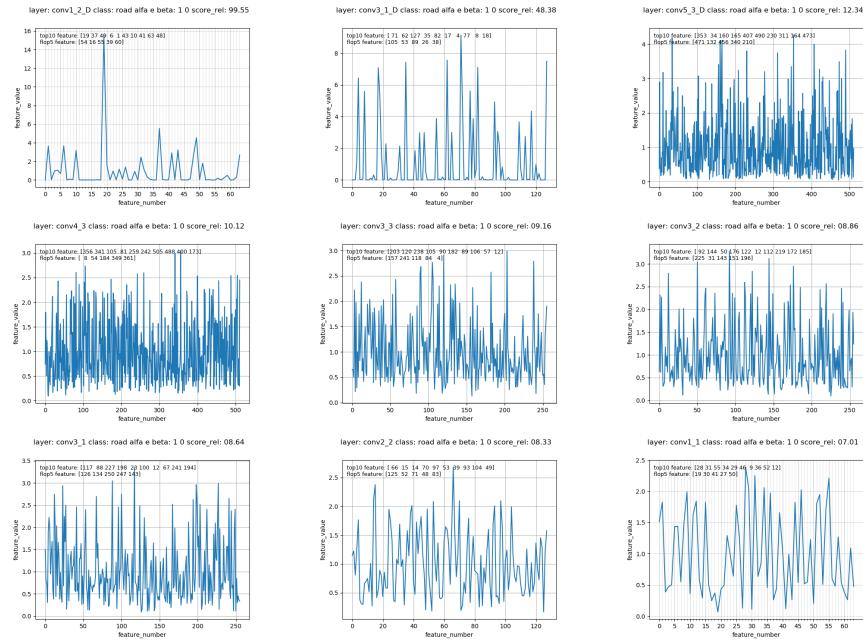


Figura 9.6: In questa figura vengono mostrati 9 grafici che corrispondono a 9 dei 37 layer di SegNet. In ognuno di essi vengono rappresentati i feature value di tutte le feature del layer

valori. Abbiamo fatto le seguenti osservazioni:

- Nei layer più profondi della rete le feature value sono molto più piccate rispetto agli altri layer
- La feature value sembra essere un'ottima metrica da poter usare come termine di confronto tra due feature appartenenti allo stesso layer, tra due feature appartenenti a layer diversi e tra le medesime feature generate da due immagini diverse.

Fin da subito i grafici delle feature values ci sono sembrati molto significativi, è come se fossero un'impronta o un segnale lasciato dal processo LRP. Vedremo

più avanti in che modo abbiamo utilizzato questi grafici come termine di confronto per diverse esecuzioni di LRP (con classi diverse oppure con immagini appartenenti a dataset diversi).

## 9.7 LRP su un singolo pixel e nuova visualizzazione della heatmap

In questo esperimento abbiamo applicato LRP processando un singolo pixel. L'apparato è sempre lo stesso: SegNet addestrata su Unreal, immagine di Unreal, architettura LRP con  $\alpha = 1$  e  $\beta = 0$ , classe strada. Possiamo vedere nella

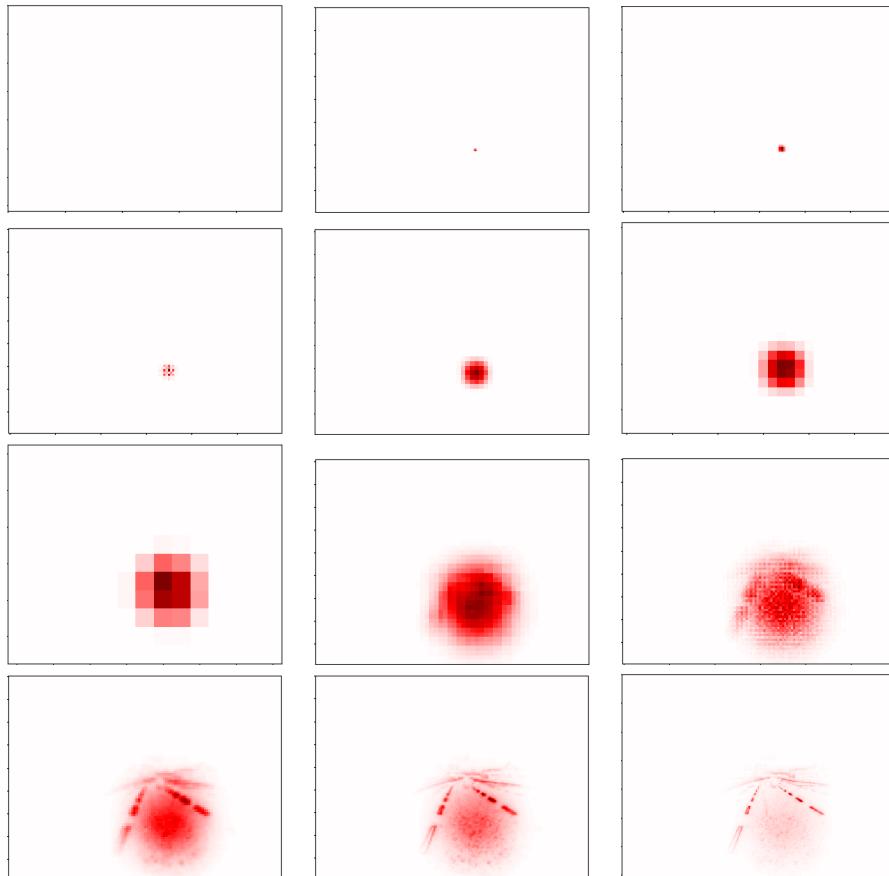


Figura 9.7: In questa immagine si può vedere la propagazione layer per layer (layer-wise) dello score di un singolo pixel a partire dal layer di output "conv1\_1D" in alto a sinistra fino al layer di input "data" in basso a destra, secondo la alfa-beta rule (equazione 6.7). L'immagine originale è quella a sinistra in figura 9.2

figura 9.7 quanto siano dettagliate le strisce della strada rispetto ad altri elementi dell’immagine. Rispetto alla figura 9.4 questo dettaglio è molto più netto. L’interpretazione che abbiamo dato a queste immagini è: la rete per classificare quel singolo pixel con la classe asfalto, non solo ha bisogno di riconoscere le strisce della strada, ma anche un certo intorno di pixel simili (alone rotondo rosso).

Per poter visualizzare meglio la distribuzione dello score nella heatmap del layer di input, abbiamo prodotto un’immagine monocromatica delle stesse dimensioni. Successivamente spegniamo in questa immagine i pixel, uno alla volta, che vanno dallo score più basso a quello più alto, mettendo in evidenza la percentuale di score rimasta e la porzione di pixel (figura 9.8). A partire dalle immagini

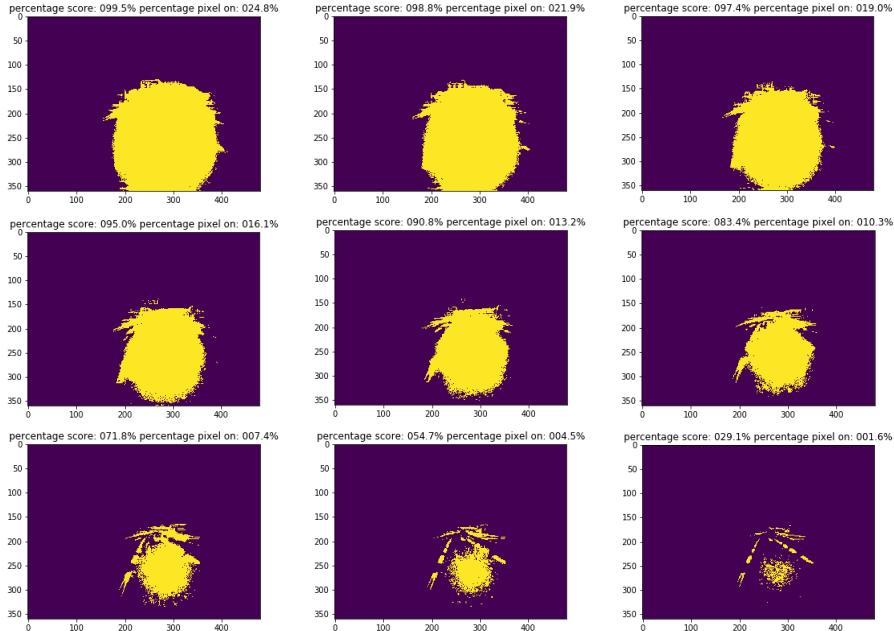


Figura 9.8: Metodo alternativo di visualizzazione della distribuzione dello score. In giallo i pixel accesi e in viola quelli spenti. L’immagine in alto a sinistra ci dice che circa il 99% dello score è distribuito nel 24,8% dei pixel dell’immagine. Se continuiamo a spegnere i pixel da quelli meno importanti a quelli più importanti arriviamo all’ultima immagine in basso a destra. Vediamo che se spengo più del 98% dei pixel in questo modo mi rimane ancora il 30% dello score totale di partenza. Possiamo anche vedere la marcata distribuzione dei pixel rimasti sulle strisce e sulla porzione di asfalto, a formare un cerchio intorno al pixel processato con LRP (l’immagine originale è sempre quella a sinistra nella figura 9.2)

prodotte in figura 9.8 abbiamo fatto le seguenti osservazioni e considerazioni:

1. E' elevatissima la definizione con cui riusciamo a catture tramite LRP i pattern nell'input, utilizzati dalla rete per la classificazione dei pixel.
2. Nelle ultime due immagini in basso a destra possiamo osservare 3 oggetti distinti: i pixel distribuiti sulle strisce, i pixel sull'asfalto e il gruppo di pixel distribuiti su alcuni oggetti nello sfondo.

Ci siamo chiesti se questi oggetti fossero catturati dalla rete in modo separato tramite feature diverse di uno stesso layer e poi utilizzati dalla rete in fase di inferenza per la classificazione dei pixel mediante una specie di composizione. La risposta a questa domanda la vedremo nell'esperimento descritto nel prossimo paragrafo.

## 9.8 Separazione delle feature

In tutti gli esperimenti visti fino ad adesso abbiamo processato LRP a partire da una inizializzazione fatta sul layer di output di Segnet. Prendiamo in considerazione i dati prodotti da una di queste esecuzioni, per ogni neurone in tutti i layer di SegNet abbiamo definito un valore di importanza detto score. Ora scegliamo un layer e una feature, azzeriamo tutti gli score di tutte le feature tranne quella scelta ed eseguiamo LRP fino layer di input della rete. Stiamo cercando di rispondere alla domanda: "Quali sono i pixel dell'immagine che influiscono positivamente sulle attivazioni dei neuroni di quella feature in quel layer?". Abbiamo eseguito questa procedura per tutte le feature del layer *conv2\_1*, che sono 128, ottenendo 128 heatmap diverse e le abbiamo concatenate in un unico blocco di dimensioni  $(128 \times 360 \times 480)$ . Successivamente abbiamo utilizzato una procedura analoga a quella del paragrafo precedente, produciamo 128 immagini monocromatiche in cui accendiamo i pixel in ordine dal più importante al meno importante, però su tutto il blocco appena definito. In base alle immagini prodotte nelle figure 9.9 e 9.10 abbiamo fatto le seguenti considerazioni.

1. Ogni feature cattura un pattern diverso, si può vedere nella figura 9.10, la feature centrale mostra un pattern totalmente diverso rispetto a quella in alto e a quella a destra.

relative\_score: 30.00%

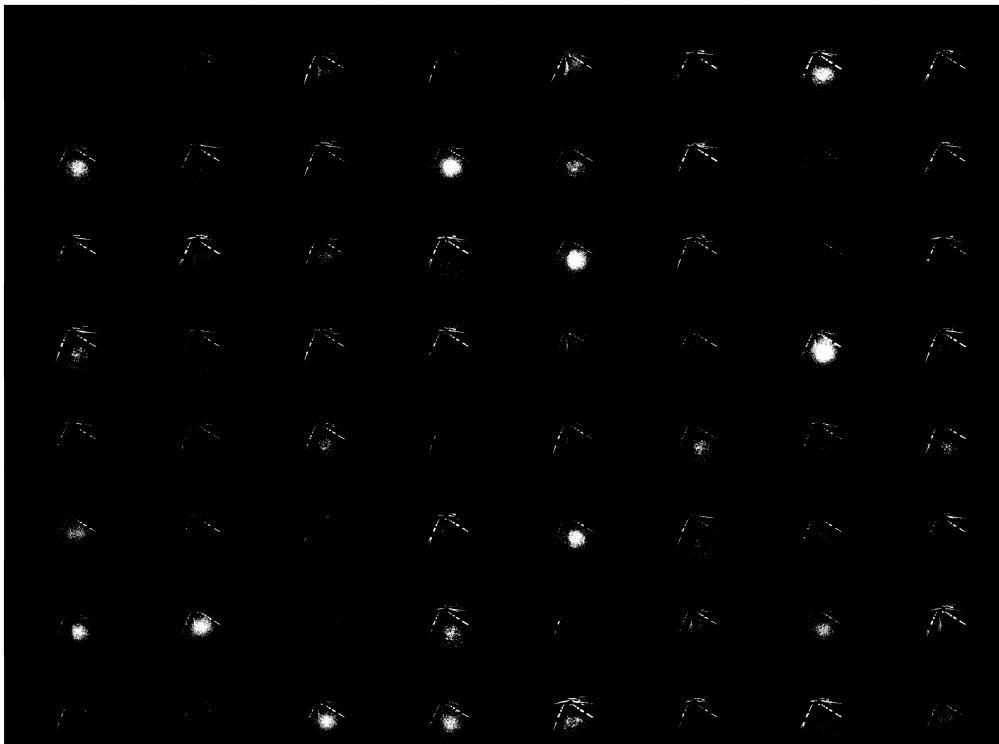


Figura 9.9: In questa figura sono mostrate 64 delle 128 proiezioni dello score delle feature all'input space. Sono stati accesi i pixel dal più importante al meno importante che contengono il 30% dello score

2. Ci sono feature più importanti di altre, si può vedere nella figura 9.9, dopo aver acceso i pixel fino al 30% dello score alcune feature rimangono ancora spente del tutto.

Ricapitolando non solo abbiamo inventato una metrica in grado di esprimere, dato un layer, l'importanza di ogni feature, ma abbiamo trovato un metodo per visualizzare i diversi pattern nell'input space che esse sono in grado di catturare. Ci siamo allora chiesti se una feature potesse essere importante soltanto per una classe di segmentazione oppure per più classi, se ci fossero feature importanti per nessuna classe oppure per tutte allo stesso modo. Abbiamo cercato di dare la risposta a questa domanda con il prossimo esperimento.

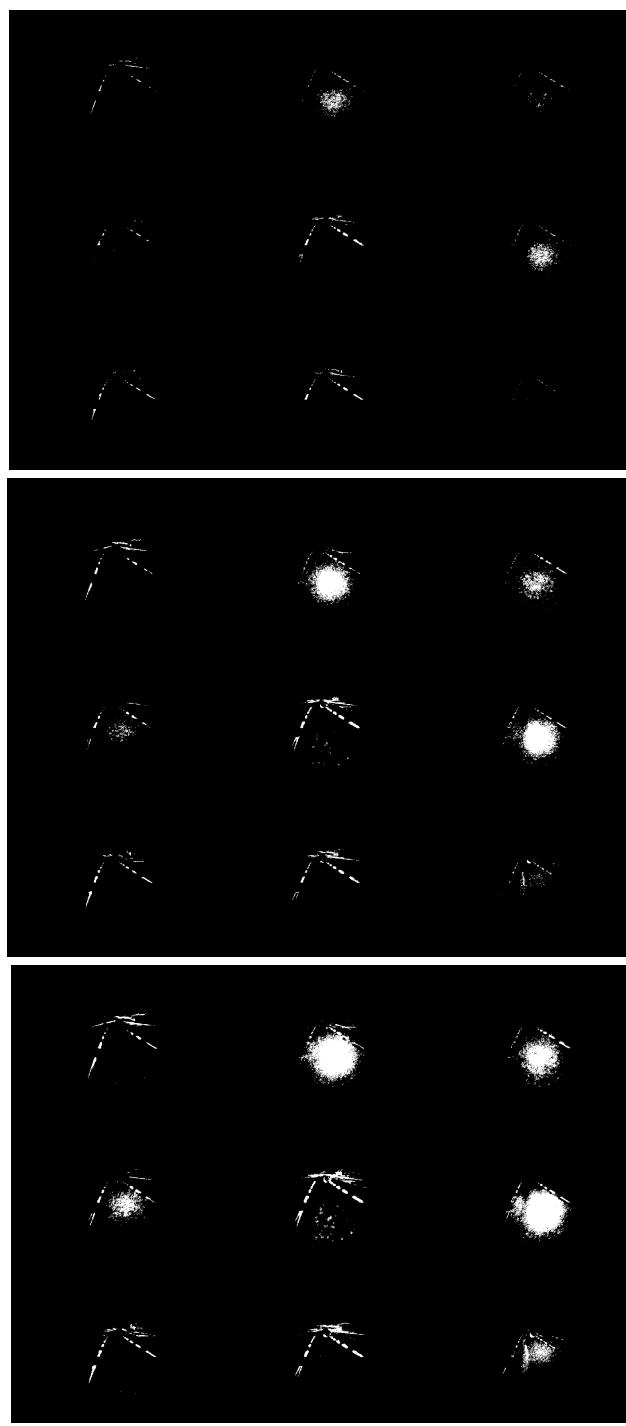


Figura 9.10: In questa figura viene mostrata l'evoluzione di 9 delle 128 feature ad una percentuale di score accesa del 15%, 35%, 50%

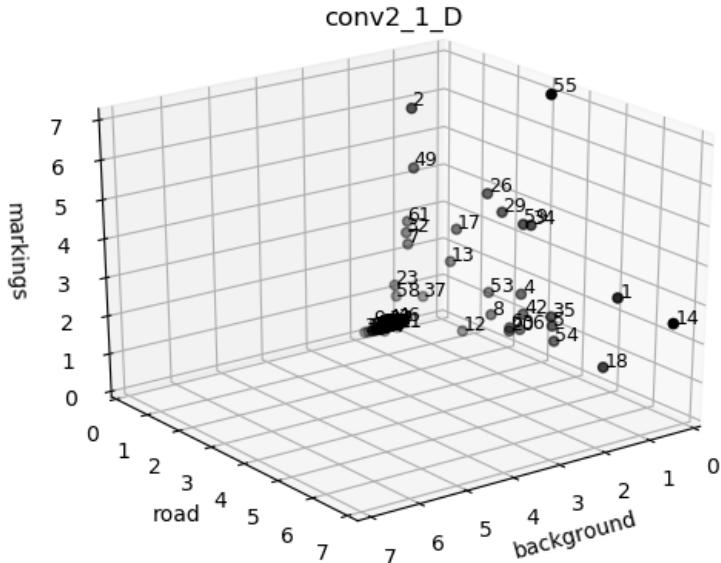


Figura 9.11: Ogni punto nello spazio tridimensionale rappresentato in figura corrisponde ad una feature del layer  $conv2_1D$ . Le coordinate dei punti corrispondono alle feature value per la classe "strada", "background" e "strisce"

## 9.9 Confronto tra diverse classi

Processiamo con LRP i true-positive delle tre classi: strada, strisce, background.

Prendiamo in considerazione un qualsiasi layer di SegNet, ad esempio  $conv2_1D$ , per ogni feature  $i$  avremo 3 feature value  $f_i$ :  $f_i^{road}$ ,  $f_i^{markings}$ ,  $f_i^{background}$ .

Possiamo definire un punto  $P_i$  di coordinate  $P_i = (f_i^r, f_i^m, f_i^b)$  per ogni feature del layer, e visualizzarli in uno spazio tridimensionale. Si possono osservare cose molto interessanti dalla figura 9.11:

1. Per il gruppetto di feature che si possono vedere in basso a sinistra si può dire che siano importanti per la classificazione "background" e sicuramente non per le altre due classi (le coordinate rispetto a queste sono praticamente nulle).
2. Possiamo vedere nel piano "markings"- "road" alcune feature molto vicine all'asse "markings" che si possono dire essere importanti per quella classe,

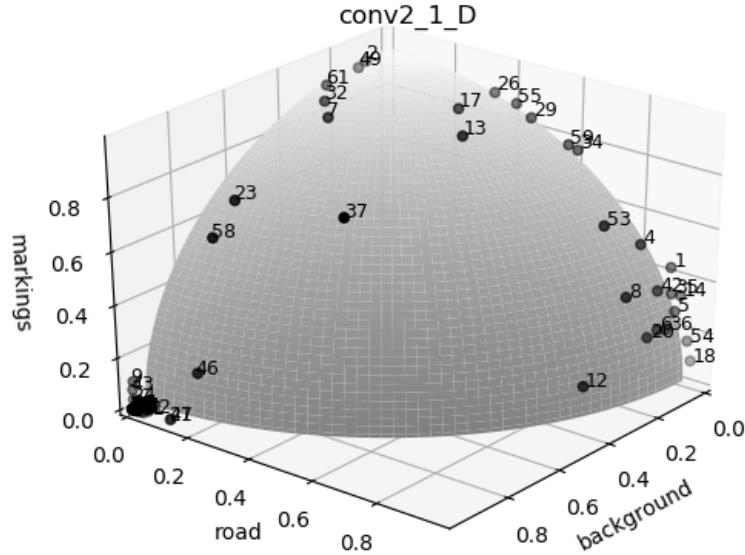


Figura 9.12: Questa visualizzazione permette di vedere meglio quali feature sono importanti per una determinata classe. Le feature che sono molto vicine al vertice in basso a sinistra del triangoloide sono importanti per la classe "background", quelle in alto sono importanti per la classe "markings", e quelle in basso a destra per la classe "road"

analogamente si può dire di alcune feature per la classe "road"

### 3. Non ci sono feature sull'origine degli assi

Una rappresentazione di questo tipo sembra dirci molto circa il modo in cui la rete utilizza le feature relativamente alla classificazione dei pixel. Ci sono feature importanti per una sola categoria di classificazione e altre sono importanti per due classi (come ad esempio quelle nel piano "markings"- "road"). Possiamo anche normalizzare a 1 le distanze di tutti i punti dall'origine degli assi e ripetere o stesso tipo di visualizzazione tridimensionale, come è mostrato ad esempio in figura 9.12.

$$P_i = (f_i^r, f_i^m, f_v^b) \rightarrow P'_i = \frac{(f_i^r, f_i^m, f_v^b)}{\sqrt{(f_i^r)^2 + (f_i^m)^2 + (f_v^b)^2}} \quad (9.4)$$

## 9.10 LRP su immagini reali con rete addestrata su dataset sintetico

Nell'introduzione abbiamo parlato del fatto che i dataset sintetici, come ad esempio quello di Unreal, per quanto possano essere percettibilmente realistici, non sono in grado di rappresentare bene la realtà, almeno nei termini di efficienza di algoritmi di machine learning. In questo paragrafo cercheremo di argomentare le motivazioni secondo le quali riteniamo che la metrica della feature value, introdotta nel paragrafo 9.6, possa essere un ottimo strumento per l'analisi di tale problema.

Per prima cosa vediamo un esempio esplicativo di che cosa significhi "non rappresentare bene la realtà". Abbiamo preso dal validation set di Mapillary 100 tra le immagini che più assomigliavano a quelle del dataset di Unreal, per prospettiva, illuminazione, definizione delle strisce, colore dell'asfalto e elementi nel background, e le abbiamo tutte quante processate con SegNet. Soltanto per pochissime di queste immagini si può dire che la segmentazione in output è "buona", per tutte le altre la rete non è in grado di riconoscere alcun elemento, come si può vedere in figura 9.13.

Prima di procedere facciamo questo esperimento: consideriamo SegNet addestrata su Unreal e operiamo con LRP su due neuroni appartenenti a pixel diversi della classe "strada", di una stessa immagine, sempre di Unreal. Confrontiamo le feature value per le due esecuzioni. Possiamo osservare nella figura 9.14 che la classificazione dei due pixel, nonostante la loro distanza nell'immagine, produce feature value praticamente identiche. Sembra assolutamente deterministico il modo in cui la rete utilizzi le proprie componenti quando si tratta di prendere una netta decisione. Proviamo ora a confrontare le feature value prodotte da un pixel preso nell'immagine sintetica e uno preso in un'immagine di Mapillary segmentata bene dalla rete. Possiamo osservare nella figura 9.15 che in questo caso le feature value sono confrontabili soltanto nei layer più profondi della rete. Nei layer più vicini all'input invece possiamo notare delle nette differenze.

Vediamo ora cosa succede quando facciamo LRP su un pixel di una immagine reale che è stata segmentata male. Possiamo osservare nella figura 9.16 che ci sono grosse differenze tra le feature value ottenute, soprattutto nei layer più

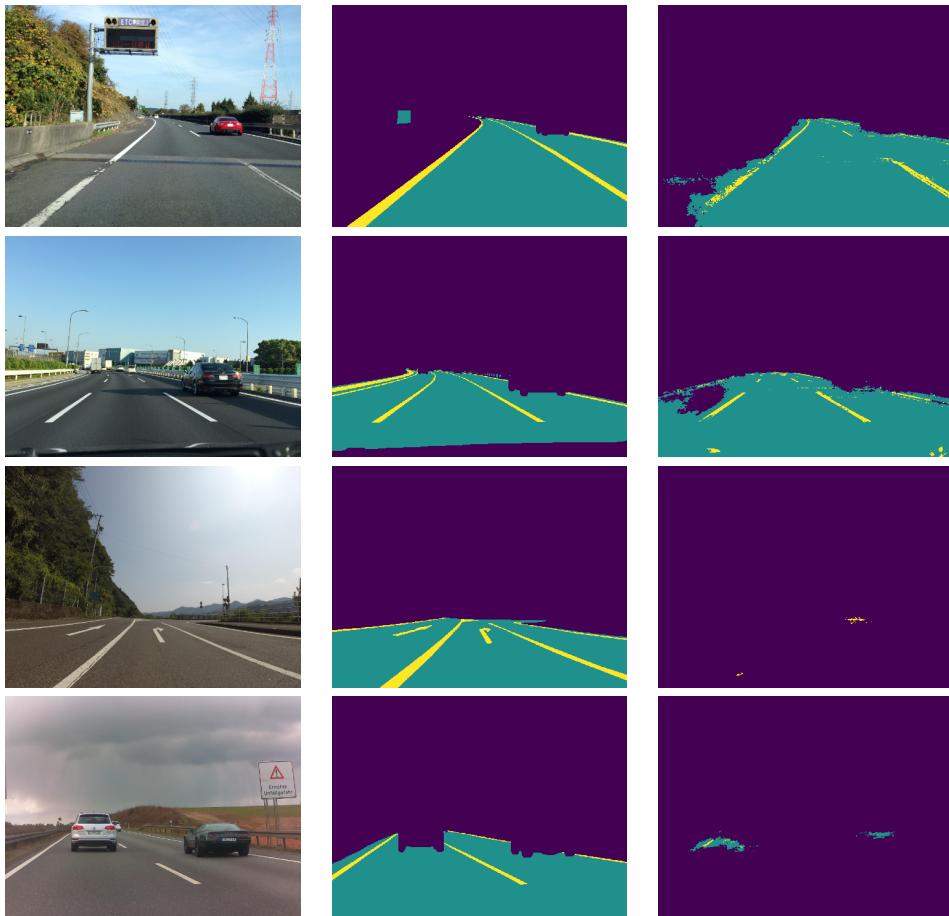


Figura 9.13: Nella colonna di sinistra vengono mostrate alcune immagini di Mapillary, al centro le relative segmentazioni vere (ground-truth), e a destra il risultato della segmentazione da parte di SegNet che è stata addestrata su Unreal

profondi della rete.

## 9.11 Osservazioni sul bias

Non lo abbiamo mostrato nei risultati ma usando la alfa-beta rule ( $\alpha = 1, \beta = 0$ ) lo score ridistribuito fino al layer di input conserva tra il 5% e il 50% del valore iniziale. Il motivo per cui ciò accade dipende dai bias ed è spiegato nel capitolo 6.1.1. Dai dati che abbiamo potuto osservare abbiamo notato che sembra esistere una forte correlazione tra la percentuale di score persa con LRP e la probabilità di classificazione. Quando con LRP propago lo score di un neurone di output

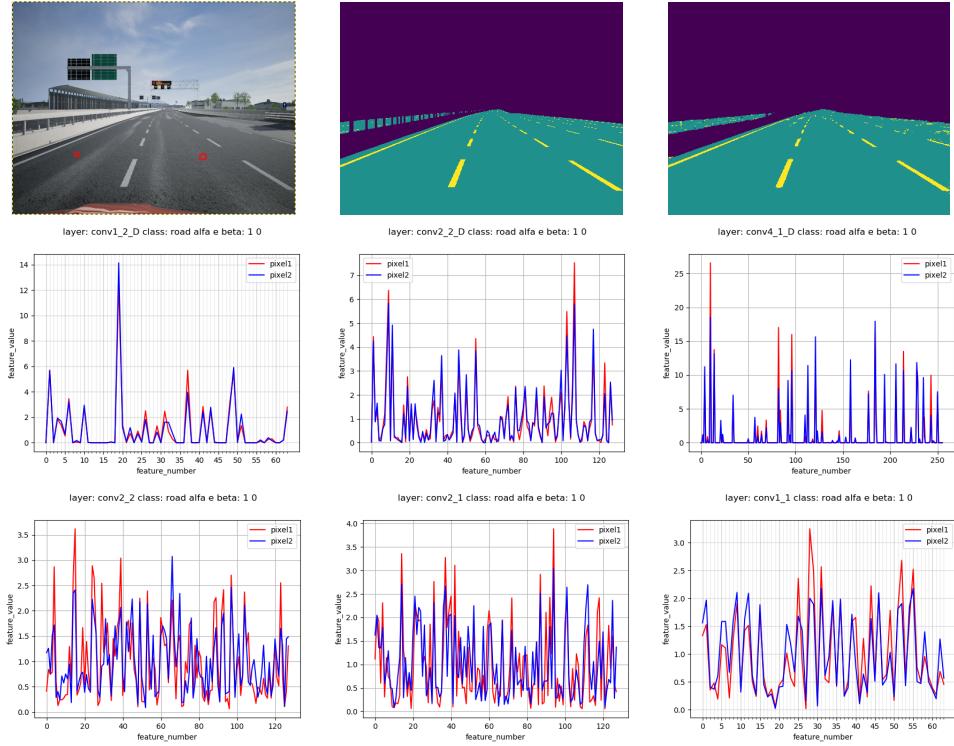


Figura 9.14: Confronto tra feature value prodotte processando con LRP due pixel diversi della stessa immagine, appartenenti alla classe "strada". Nell'immagine in alto a sinistra sono cerchiati di rosso i due pixel e nella stessa riga vediamo la relativa segmentazione e output di SegNet, nelle immagini in basso si vedono i grafici delle feature value per layer diversi della rete. Da notare che i valori sono identici a meno di piccole fluttuazioni.

adibito al punteggio di una certa classe per un certo pixel, se la probabilità di classificare quel pixel con quella classe è molto elevata allora lo score viene perso ed assorbito dai bias fino al 95% ad arrivare al layer di input, diversamente se la probabilità è molto bassa o nulla lo score perso è di circa la metà.

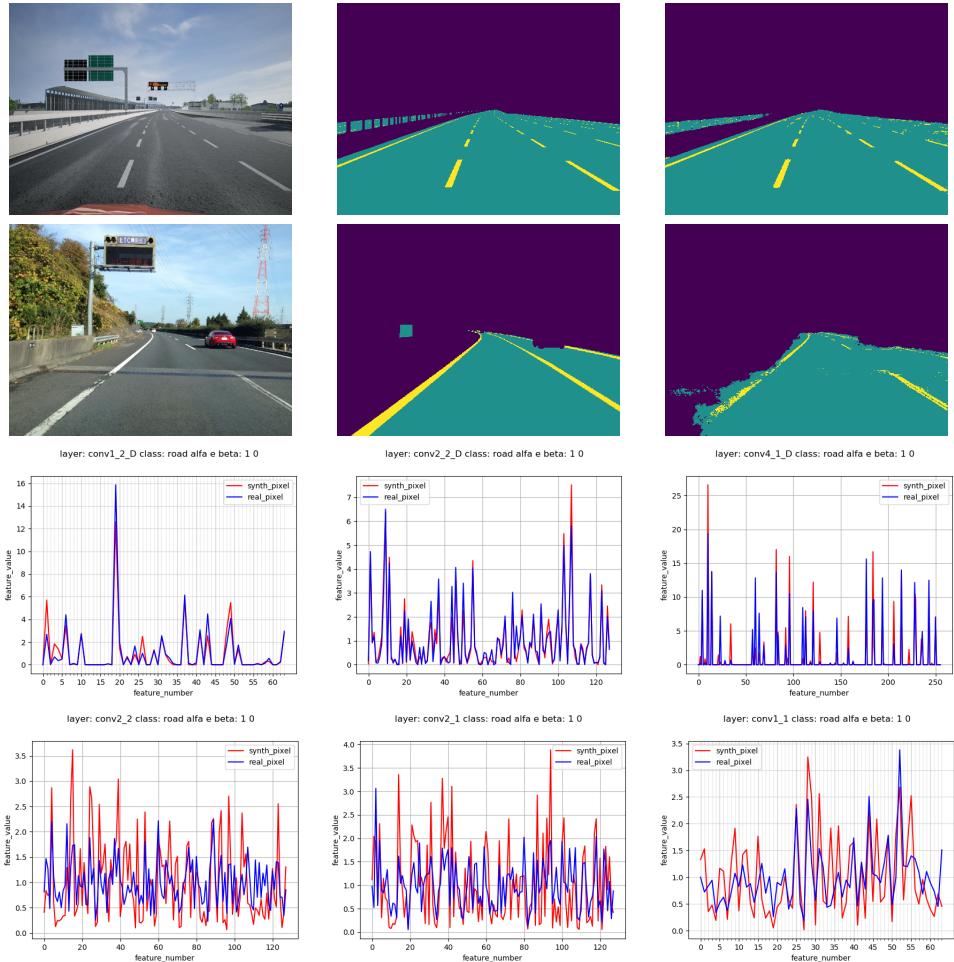


Figura 9.15: In alto nella figura vediamo l'immagine sintetica di Unreal, quella reale di Mapillary, le relative segmentazioni vere e l'output si SegNet. In basso abbiamo il confronto tra feature value di layer diversi di SegNet, ottenute processando con LRP un pixel della classe "strada" dell'immagine di Mapillary e un pixel della classe "strada" dell'immagine di Unreal.

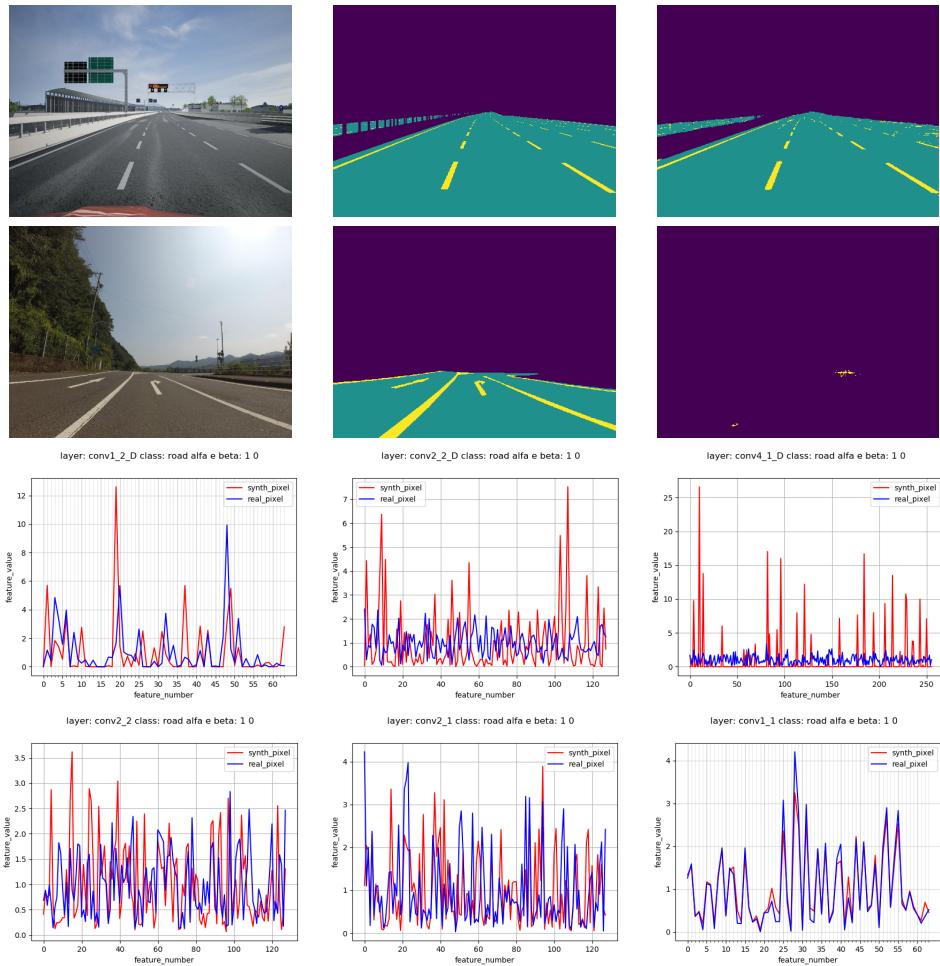


Figura 9.16: In alto nella figura vediamo l'immagine sintetica di Unreal, quella reale di Mapillary, le relative segmentazioni vere e l'output si SegNet. In basso abbiamo il confronto tra feature value di layer diversi di SegNet, ottenute processando con LRP un pixel della classe "strada" dell'immagine di Mapillary e un pixel della classe "strada" dell'immagine di Unreal.

# Capitolo 10

## Conclusioni

SegNet si è rivelata essere una buona rete per i tipi di analisi che abbiamo effettuato. Nonostante la complessità della sua architettura, che è inevitabilmente commisurata al tipo di compito che deve svolgere, ha risposto bene e quasi sempre in maniera deterministica alle tecniche di analisi a cui l'abbiamo sottoposta. Ci teniamo a precisare che LRP e Grad-CAM sono state pensate e introdotte per reti che svolgono compiti più semplici come la classificazione di immagini; la generalizzazione e l'implementazione con successo ad una rete che fa segmentazione come SegNet è un nostro risultato innovativo.

Riteniamo che tutte le tecniche di visualizzazione e tutte le analisi mostrate negli esperimenti siano un buon punto di partenza per uno studio più approfondito del problema delle reti neurali addestrate sui dataset sintetici. Tale considerazione è supportata dal fatto che la tecnica della Layer-wise Relevance Propagation, e la metrica della feature value che abbiamo introdotto, sembrano essere sensibili alla "qualità" dell'output generato dalla rete. Il punto di forza della Layer-wise Relevance Propagation è la capacità di conservare tutta l'informazione, diversamente da quello che accade per Grad-CAM dove gran parte viene persa a causa della somma pesata delle feature.

Un'ulteriore analisi andrebbe effettuata sui bias dei kernel di convoluzione. Il fatto che con la alfa-beta rule ( $\alpha = 1, \beta = 0$ ) implementata su SegNet non valga la legge di conservazione descritta dall'equazione (5.7) nel capitolo 5.2.4, non lo ritieniamo un limite della metodologia. Tutto ciò infatti va incontro al para-

digma della pixel-wise decomposition descritta nel capitolo 5.2.2, ma può essere ripensato e rivalutato in relazione alla correlazione tra la probabilità dell'output e lo score perso descritta nel capitolo 9.11.

Relativamente ai pochi ma apparentemente significativi risultati che abbiamo ottenuto, riteniamo che sia fondamentale proseguire lo studio cercando di capire se effettivamente le visualizzazioni (heatmap) producono una spiegazione della decisione della rete. Ad esempio, considerando la forte concentrazione dello score sulle strisce della strada suggerisce che esse siano un elemento fondamentale per la classificazione dei pixel. Ci sentiamo di dire che la rete fallisce quando non è in grado di riconoscere questo pattern o altre componenti che emergono dalle tecniche di visualizzazione che abbiamo introdotto. Pensiamo che dati questi risultati, il dataset sintetico debba essere generato utilizzando tante combinazioni di texture diverse sia per le strisce che per l'asfalto, e tante combinazioni ambientali di luce. Questa procedura potrebbe permettere alla rete di imparare a riconoscere questi pattern in molti più casi, e quindi di inferenziare in maniera più performante sulle immagini reali che sono estremamente variabili. Prove di questo tipo potrebbero dare un maggiore supporto alle tecniche di spiegazione e potrebbero magari portare alla luce aspetti e caratteristiche delle reti neurali artificiali ancora nascoste.

# Bibliografia

- [1] K. Simonyan & A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556 (2014).
- [2] L. Deng, G. Hinton, & B. Kingsbury, *New types of deep neural network learning for speech recognition and related applications: An overview*, in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603, IEEE (2013).
- [3] G. G. Chowdhury, *Natural language processing, Annual review of information science and technology*, 37(1):51–89 (2003).
- [4] O. Russakovsky et al., *Imagenet large scale visual recognition challenge*, *International Journal of Computer Vision*, 115(3):211–252 (2015).
- [5] H. A. Alhaija et al., *Augmented reality meets deep learning for car instance segmentation in urban scenes*, in *British Machine Vision Conference*, volume 1, page 2 (2017).
- [6] N. M. Nasrabadi, *Pattern recognition and machine learning*, *Journal of electronic imaging*, 16(4):049901 (2007).
- [7] J. Tobin et al., *Domain randomization for transferring deep neural networks from simulation to the real world*, in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30, IEEE (2017).
- [8] D. Dwibedi, I. Misra, & M. Hebert, *Cut, paste and learn: Surprisingly easy synthesis for instance detection*, in *The IEEE international conference on computer vision (ICCV)* (2017).

- [9] X. Peng et al., *Learning deep object detectors from 3d models*, in *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286 (2015).
- [10] J. D. Olden & D. A. Jackson, *Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks*, *Ecological modelling*, 154(1-2):135–150 (2002).
- [11] V. Badrinarayanan, A. Kendall, & R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*, *arXiv preprint arXiv:1511.00561* (2015).
- [12] R. R. Selvaraju et al., *Grad-cam: Visual explanations from deep networks via gradient-based localization.*, in *ICCV*, pages 618–626 (2017).
- [13] S. Bach et al., *On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation*, *PloS one*, 10(7):e0130140 (2015).
- [14] V. Vapnik, *The nature of statistical learning theory*, Springer science & business media (2013).
- [15] L. Bottou, *Large-scale machine learning with stochastic gradient descent*, in *Proceedings of COMPSTAT’2010*, pages 177–186, Springer (2010).
- [16] G. Hinton, N. Srivastava, & K. Swersky, *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*, *Cited on*, page 14 (2012).
- [17] I. Sutskever et al., *On the importance of initialization and momentum in deep learning*, in *International conference on machine learning*, pages 1139–1147 (2013).
- [18] O. Russakovsky et al., *ImageNet Large Scale Visual Recognition Challenge*, *International Journal of Computer Vision (IJCV)*, 115(3):211–252 (2015).
- [19] K. Simonyan & A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, *arXiv preprint arXiv:1409.1556* (2014).
- [20] A. Krizhevsky, I. Sutskever, & G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *Advances in neural information processing systems*, pages 1097–1105 (2012).

- [21] L.-C. Chen et al., *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848 (2018).
- [22] J. Long, E. Shelhamer, & T. Darrell, *Fully convolutional networks for semantic segmentation*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440 (2015).
- [23] S. Ioffe & C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *arXiv preprint arXiv:1502.03167* (2015).
- [24] D. Erhan et al., *Visualizing higher-layer features of a deep network*, *University of Montreal*, 1341(3):1 (2009).
- [25] M. D. Zeiler & R. Fergus, *Visualizing and understanding convolutional networks*, in *European conference on computer vision*, pages 818–833, Springer (2014).
- [26] J. T. Springenberg et al., *Striving for simplicity: The all convolutional net*, *arXiv preprint arXiv:1412.6806* (2014).
- [27] K. Simonyan, A. Vedaldi, & A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, *arXiv preprint arXiv:1312.6034* (2013).
- [28] D. E. Rumelhart, G. E. Hinton, & R. J. Williams, *Learning representations by back-propagating errors*, *nature*, 323(6088):533 (1986).
- [29] L. Bottou et al., *Comparison of classifier methods: a case study in handwritten digit recognition*, in *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing.*, volume 2, pages 77–82, IEEE.
- [30] Y. Bengio, A. Courville, & P. Vincent, *Representation learning: A review and new perspectives*, *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828 (2013).

- [31] A. Mahendran & A. Vedaldi, *Visualizing deep convolutional neural networks using natural pre-images*, *International Journal of Computer Vision*, 120(3):233–255 (2016).
- [32] G. Montavon, W. Samek, & K.-R. Müller, *Methods for interpreting and understanding deep neural networks*, *Digital Signal Processing* (2017).
- [33] G. Neuhold et al., *The mapillary vistas dataset for semantic understanding of street scenes*, in *International Conference on Computer Vision (ICCV)* (2017).
- [34] G. J. Brostow, J. Fauqueur, & R. Cipolla, *Semantic object classes in video: A high-definition ground truth database*, *Pattern Recognition Letters*, xx(x):xx–xx (2008).
- [35] G. Ros et al., *The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243 (2016).