ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Sistemi Intelligenti

Ricerca automatica finalizzata alla decodifica di codici Data Matrix tramite tecniche di Intelligenza Artificiale

CANDIDATO
Silvio Olivastri

RELATORE
Chiar.ma Prof.ssa Michela Milano

CORRELATORI
Francesco D'Ercoli
Marco Lippi

Anno Accademico 2016/2017
Sessione II

Indice

1 Introduzione

Lo scopo della tesi è quello di creare un sistema intelligente che, in modo autonomo, trovi i parametri necessari per decodificare codici Data Matrix. Questo sistema si interfaccia con un lettore di codice a barre professionale dotato di camera fotografica e una libreria interna, che ha lo scopo di interpretare il codice all'interno dell'immagine acquisita.

I Data Matrix vengono generalmente impressi tramite il processo di Direct Part Marking (DPM), che impone una marcatura permanente del codice su un prodotto tramite varie tipologie di strumenti. In questo modo è possibile identificare il prodotto in ogni fase della sua distribuzione e/o assemblaggio.

Le diverse tipologie di stampa impongono che la lettura venga affidata a dei lettori molto avanzati che permettono di modificare numerosi parametri, al fine di rendere il codice visibile quanto basta per dare la possibilità alla libreria di decodificarlo. Alcuni esempi di questi parametri sono: posizione dei led accesi, luce emessa dai led, distanza focale e contrasto dell'immagine.

Avvalendosi di un tecnico, i lettori vengono posizionati in punti prefissati, e per ogni prodotto viene ricercata manualmente una ricetta (combinazioni di parametri) che permette di leggere le informazioni impresse. In questo modo, l'utilizzatore dovrà semplicemente inserire i parametri specificati a seconda del prodotto da scansionare. Il problema si pone soprattutto quando un nuovo prodotto viene inserito nella linea, in questo caso, se le ricette precedenti non sono riutilizzabili, il tecnico deve intervenire introducendo un costo per l'azienda. Da qui parte l'esigenza di creare un sistema esperto che permette di svolgere questo compito in maniera automatica.

In questo contesto è stata proposta una metodologia che suddivide il problema in tre fasi: localizzazione, autofocus e ricerca dei parametri. Mentre la localizzazione del codice nell'immagine viene svolta da una rete neurale profonda, le atre due fasi vengono evase tramite algoritmi di ottimizzazione su uno spazio di ricerca accuratamente discretizzato.

Questa pubblicazione è suddivisa in 8 capitoli. Dopo la presente introduzione seguirà il capitolo 2, dove è possibile trovare la descrizione del

problema, il campo applicativo e lo strumento utilizzato con lo studio dei relativi parametri. Il capitolo 3 fornisce uno sguardo generale sui principali algoritmi di ottimizzazione, soffermandosi anche su quelli utilizzati per l'allenamento di reti neurali profonde. Il capitolo 4 fa riferimento al Machine Learning, coprendo solo la parte relativa ai problemi di classificazione. Esso è introduttivo al quinto capitolo, utilizzato per illustrare le tecniche di Deep Learning, soprattutto riguardo le architetture CNN e i framework realizzati per la localizzazione di oggetti in immagini. Nel capitolo 6 è descritto accuratamente l'approccio di risoluzione del problema con i relativi risultati, per poi concludere con il 7, nel quale vengono espressi pareri sul lavoro svolto ed eventuali sviluppi da eseguire in futuro.

2 Definizione del problema

Il seguente capitolo viene proposto per argomentare il problema affrontato in questo studio. Particolare attenzione viene posta alla definizione del contesto applicativo e agli strumenti a disposizione.

2.1 Codici e metodi per il marking

Il Direct Part Marking (DPM) è il processo attraverso il quale le parti di un prodotto vengono marchiate permanentemente, inserendo dei dati riguardanti il numero di serie e altre informazioni aggiuntive, con lo scopo di tracciare il prodotto per tutto il suo ciclo di vita. È utilizzato in molti settori, come quello automobilistico, aereospaziale ed elettronico (Cognex Corporation, 2017, A).

Il prodotto viene contrassegnato da un codice, tipicamente un Data Matrix (figura 2.1.a) o un QR-Code (figura 2.1.b). In questo studio verrà considerata solo la prima tipologia.

Figura 2.1.a

Esempio di Data Matrix

Figura 2.1.b

Esempio di QR-Code

Fonte: https://en.wikipedia.org/wiki/Data_Matrix; https://en.wikipedia.org/wiki/QR_code

Il Data Matrix è un simbolo 2D, formato da una griglia solitamente quadrata o rettangolare. All'interno del suo schema, ogni elemento della griglia è chiamato modulo (o cella), esso è solitamente quadrato ma è possibile trovarlo anche in forma rotonda. La caratteristica principale di questo codice sono i quattro bordi, quelli totalmente neri (nella parte a sinistra e in basso) vengono chiamati "L finder pattern", mentre gli altri sono alternati e formano il "Clock Track". I primi sono molto importanti per trovare il codice e

determinano la grandezza e l'orientamento, mentre i secondi definiscono la struttura base e aiutano a indentificare l'eventuale distorsione (GS1, 2016).

Questo codice viene stampato sopra diversi tipi di materiali (carta, metallo, plastica, vetro, PCB, etc.), in superfici curve o piane. A seconda del tipo di materiale è possibile applicare vari tipi di stampa, in generale le più comuni sono: inchiostro (ink), laser, pallinatura e marcatura elettrolitica (Videojet, 2014). Di seguito sono riportate le principali combinazioni prese in considerazione in questo studio:

- Metallo: ink (figura 2.2.a), laser, puntellato (figura 2.2.b), marcatura elettrolitica;
- Plastica: ink, laser, puntellato;
- Vetro: ink, laser (figura 2.2.c);
- PCB (policlorobifenili): laser (figura 2.2.d).

Figura 2.2.a

Data Matrix su metallo, ink

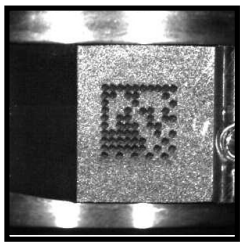


Figura 2.2.b

Data Matrix su metallo, puntellato

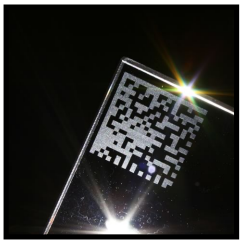


Figura 2.2.c

Data Matrix su vetro, laser

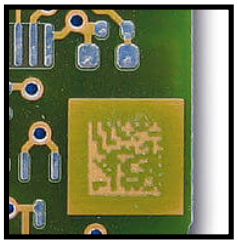


Figura 2.2.d

Data Matrix su PCB, laser

Fonte: https://www.pryormarking.com/products/inkjet-marking; http://www.etchmark.co.uk/products/data-matrix/;

http://www.unitedspectrum.in/applications.php;
https://it.pinterest.com/pin/440367669787124783/

2.2 Campo applicativo e il problema dell'auto-learn

La lettura dei codici viene solitamente effettuata nella linea di produzione, soprattutto subito dopo la fase di marcatura. In questa linea il prodotto passa sotto un marcatore che imprime il codice, dopodiché, per verificare la bontà della stampa e la correttezza dei dati inseriti, viene fatto passare sotto un lettore (in maniera statica o dinamica). Il lettore effettua l'acquisizione, ovvero crea una fotografia, osserva se all'interno è presente il codice e cerca di codificarlo.

Poiché le combinazioni materiali/tipologia di stampa sono molto variegate, i lettori che effettuano la rilevazione sono molto complessi. Analizzando i top di gamma è possibile trovare delle caratteristiche comuni come: possibilità di accensione e spegnimento di led posti in punti differenti, scelta della distanza focale, applicazione di filtri o altre tecniche di elaborazione delle immagini per aumentare o diminuire il contrasto della foto (Cognex Corporation, 2017, B; Datalogic, 2016, A). Per dare un'idea delle varie situazioni che possono verificarsi, si immagini di voler leggere un codice impresso sopra ad una superficie di plastica nera e poco riflettente, in questo caso l'intensità di luce da applicare per poterlo osservare in foto dovrà essere relativamente alta. Questa condizione però non potrà essere utilizzata per un oggetto con superficie metallica argentata poiché la sovra-illuminazione porterebbe a coprire il codice o una parte di esso. Un altro problema è quello riguardante la distanza del sensore dal codice. Premettendo che il sensore venga installato in una posizione fissa e non removibile, i prodotti che scorreranno sotto di esso saranno di altezze diverse, per cui è indispensabile cambiare ogni volta la distanza focale.

È chiaro dunque che a seconda della situazione, i parametri da inserire nel lettore saranno diversi. Il processo che ricerca automaticamente la combinazione di parametri, o "recipe" (ricetta) migliore, è chiamato comunemente "auto-learn". È importante notare che la fase di ricerca può durare anche molto tempo (cinque o dieci minuti) poiché, una volta trovata la

ricetta giusta, durante la produzione si utilizzerà sempre quest'ultima per leggere il pezzo. Infatti, l'operazione viene svolta soltanto quando l'azienda crea un nuovo prodotto o per ogni prodotto quando viene acquistato il lettore, anche se, in questi casi, di solito viene mandato un tecnico ad installare lo scanner e a ricercare manualmente le ricette. Il vero problema si pone nel primo caso dove, in teoria, bisognerebbe chiamare ancora il tecnico per creare una nuova ricetta apposita, portando l'azienda al consumo di denaro e tempo.

2.3 Verifica della qualità del codice

Una volta che il codice viene marchiato è importante verificare la bontà della stampa effettuata, a tale scopo il lettore dovrà anche valutare l'accuratezza in termini di qualità. A seconda dello standard, esistono diversi parametri, verificati tramite degli algoritmi, che un codice deve rispettare. Di seguito viene esposta la tabella 2.1 che riassume i parametri tenuti in considerazione dallo standard AIM DPM (Microscan, 2013; Cognex Corporation, 2006; Keyence Corporation, 2015).

Parametri	Descrizione	Esempio	Valori
Axial non-uniformity	Deviazione lungo gli assi principali.		A ($v \leq .06$) B ($.06 < v \leq .08$) C ($.08 < v \leq .1$) D ($.1 < v \leq .12$) F ($v > .12$)
Cell contrast	Differenza tra gli elementi chiari e scuri.		A ($v \geq .3$) B ($.3 < v \leq .25$) C ($.25 < v \leq .2$) D ($.2 < v \leq .15$) F ($v < .15$)
Cell modulation	Deviazione del colore dei moduli.		A ($v \geq .5$) B ($.5 < v \leq .2$) F ($v < .2$)

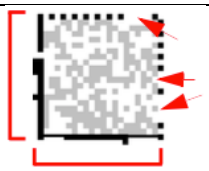
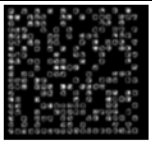
Decodability	Leggibilità rispetto ad un algoritmo di decodifica.		N.D.
Fixed pattern damage	Danno riscontrato nel perimetro del codice.		N.D.
Grid non-uniformity	Deviazione dell'intersezione della griglia.		A ($v \leq .38$) B ($.38 < v \leq .5$) C ($.5 < v \leq .63$) D ($.63 < v \leq .75$) F ($v > .75$)
Minimum reflectance	Riflettanza minima degli elementi chiari.		A ($v \geq .05$) F ($v < .02$)
Unused error correction	Possibile correzione degli errori.		A ($v \geq .62$) B ($.62 < v \leq .5$) C ($.5 < v \leq .37$) D ($.37 < v \leq .25$) F ($v < .25$)

Tabella 2.1

La tabella espone i parametri che distinguono lo standard AIM DPM. Nella colonna "Valori" sono riportati gli intervalli di bontà (in percentuale): quelli verdi rappresentano una condizione "Good", quelli arancioni "Fair", mentre quelli rossi "Poor".

2.4 Matrix 300N

In questo studio è stato utilizzato il Matrix 300N, il prodotto di punta della Datalogic per applicazioni che riguardano il DPM (figura 2.3).

Figura 2.3

Una versione del Matrix 300N.

Fonte: http://www.datalogic.com/eng/products/manufacturing-transportation-logistics/stationary-industrial-scanners/matrix-visiset-series-pd-657.html

Questo strumento permette di ottenere un'elevata configurazione di parametri in grado di adattarsi a qualsiasi condizione d'uso. Normalmente, questi parametri sono configurati da un tecnico (o dall'utilizzatore) tramite l'applicazione desktop DL.CODE, che comunica con il dispositivo tramite un protocollo a livello di applicazione, chiamato Host Mode Programming (HMP). Nel progetto vengono considerati solo i parametri tipici utilizzati nella lettura dei codici Data Matrix, in modalità "Standard", i quali vengono di seguito riportati con le relative descrizioni:

- Gain: incrementa di un fattore moltiplicativo intero l'intensità luminosa di ogni pixel dell'immagine. Più questo valore è alto, più l'intensità luminosa media risulterà alta. Ha lo svantaggio di aumentare il rumore dei pixel outlier presenti nell'immagine;
    - Intervallo del dominio: $[4, 48]$.
- Gain Multiplexer: moltiplica il parametro Gain per un'ulteriore fattore intero;

- Intervallo del dominio: $[1, 10]$.

- Exposure Time: intervallo di tempo durante il quale l'otturatore della camera del sensore rimane aperto. Determina la luminosità media dell'immagine acquisita;

  - Il dominio dipende dal valore di Internal Lighting (tabella 2.2) ed è disponibile per valori interi.

Intervallo del dominio	Internal Lighting
$[1, 500]$	2
$[100, 3300]$	3
N.D.	0,1

Tabella 2.2

Dominio del parametro Exposure Time (in microsecondi) al variare del parametro Internal Lighting.

- Internal Lighting: determina la modalità di funzionamento di tutti i led del sensore;

  - Dominio: $\{0, 1, 2, 3\}$ (tabella 2.3).

Valore	Nome	Descrizione
0	Disabled	Tutti i led rimangono costantemente spenti.
1	Always On	Tutti i led rimangono costantemente accesi.
2	Very High-Power Strobed	I led sono accesi solo per pochi nano secondi. La luce emessa è maggiore in confronto alla modalità High-Power Strobed.

3	High-Power Strobed	I led sono accesi solo per pochi nano secondi. La luce emessa è maggiore in confronto alla modalità Always On.

Tabella 2.3

Valori e relativa descrizione del parametro Internal Lighting.

- Reading Distance: distanza focale;
    - Intervallo del dominio: $[22, 452]$ millimetri con valori interi.
- Sectors: settori in cui i led sono accessi. Sono presenti quattro settori per ogni LED Group, ognuno dei quali può risultare acceso o spento (figura 2.4.a, figura 2.4.b);
    - Dominio: $\{0,1\}$ per ogni settore. Il dominio complessivo può essere visto come la permutazione dei quattro settori e ha cardinalità pari a 16 (comprendendo anche il caso in cui tutti i settori sono spenti).
- LED Group: catena dei led utilizzata. È possibile utilizzare una sola catena di led alla volta, una è presente nella parte centrale, l'altra nella parte periferica;
    - Dominio: $\{0, 1\}$ (tabella 2.4).

Valore	Nome
0	Central
1	Peripheral

Tabella 2.4

Nomi dei valori del dominio LED Group.

- ROI (Region of Interest): regione di acquisizione. Il Matrix 300N può ottenere un'immagine ad una dimensione massima di 1280x1024. Questa funzione permette di diminuire l'area, aumentandone la velocità di acquisizione. Partendo dall'angolo in alto a sinistra, questo parametro agisce su 4 sub-parametri: X (pixel di inizio immagine sull'asse orizzontale), Y (pixel di inizio immagine sull'asse verticale), Width (larghezza della finestra) e Height (altezza della finestra);

    - Intervalli dei domini:
        - X: $[0, 1216]$ pixel.
        - Y: $[0, 974]$ pixel.
        - Width: $[64, 1280]$ pixel (con incremento di 32 pixel).
        - Height: $[50, 1024]$ pixel.

È possibile anche inserire dei filtri all'immagine (ad esempio, erosione o dilatazione) ma il loro uso non è molto comune, per cui questa opzione non è stata considerata (Datalogic, 2016, B; Datalogic, 2016, C; Datalogic, 2014).

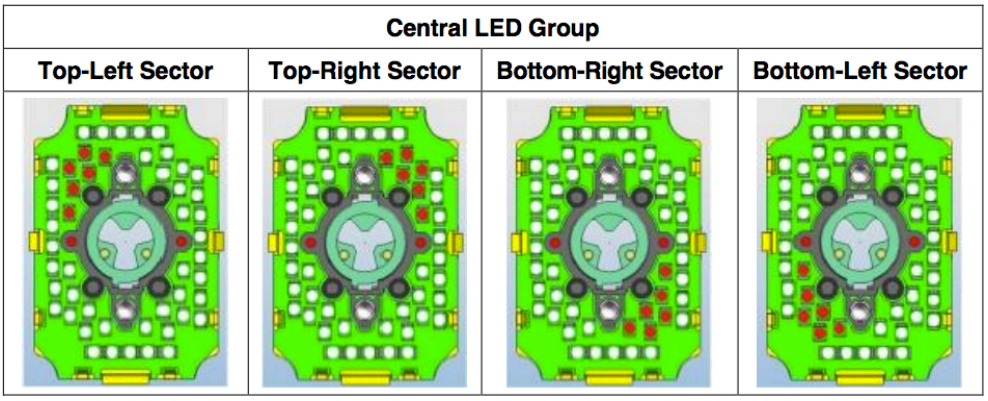


Figura 2.4.a

Settori dei LED group centrali.

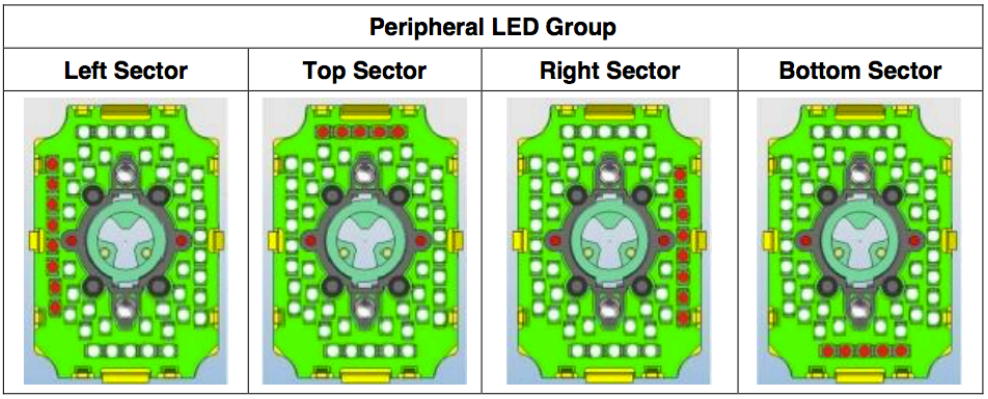


Figura 2.4.b

Settori dei LED Group periferici.

Il dispositivo contiene al suo interno una libreria di decodifica chiamata Virtual Library (VL). Nel nostro studio questa libreria viene vista come un sistema a scatola chiusa in cui in ingresso viene fornita un'immagine e in uscita si possono avere due situazioni:

- "Read". Nell'immagine è identificato un codice di tipo Data Matrix. Vengono poi mostrati posizione, tempo di decodifica e i valori di qualità di stampa (Axial non-uniformity, Cell contrast, Cell modulation, Grid non-uniformity, Minimum reflectance e Unused error correction);
- "No Read". Non vengono identificati codici Data Matrix nell'immagine.

3 Algoritmi di ottimizzazione

Gli algoritmi di ottimizzazione svolgono un ruolo molto importante in tutto il progetto. Nelle fasi di ricerca vengono utilizzati degli approcci euristici Greedy, mentre le reti neurali, sfruttate per la localizzazione del codice, si servono delle tecniche di discesa del gradiente per modificare i pesi.

Un qualsiasi problema di ottimizzazione può essere espresso nella seguente formulazione:

$$\begin{cases} \min f(x) \\ g_i(x) \geq 0 \\ h_j(x) = 0 \\ x \in X \subseteq \mathbb{R}^n \end{cases}$$

dove

- x è il vettore delle variabili decisionali a n componenti;
- X è un sottoinsieme dello spazio euclideo $\mathbb{R}^n$;
- f, g_i, h_j sono funzioni generiche dipendenti dal problema affrontato.

L'obiettivo è identificare il valore di x che, rispettando i vincoli g_i e h_j, minimizza il valore della funzione obiettivo $f(x)$.

Questo capitolo espone i principali metodi di ottimizzazione euristica, meta-euristica e discesa del gradiente.

3.1 Euristici

Gli algoritmi euristici sono delle tecniche finalizzate alla ricerca dell'ottimo locale, che in alcuni casi specifici (ad esempio spazi di ricerca convessi) coincide con l'ottimo globale. Hanno il vantaggio di essere molto semplici come costruzione e spesso vengono utilizzati per trovare una soluzione ammissibile iniziale quando si vogliono utilizzare successivamente delle tecniche meta-euristiche.

La ricerca nello spazio delle soluzioni viene guidata da una funzione euristica che fornisce una previsione della bontà di una soluzione ammissibile non ancora esplorata, in questo modo è possibile scartare a priori delle soluzioni piuttosto che altre aumentando la velocità di ricerca (Ventura, 2017).

3.1.1 Greedy

Questo approccio è molto conosciuto per la sua semplicità ed è utilizzato per la risoluzione di molti problemi noti, come Knapsack problem, Traveling salesman problem e Minimum spanning tree problem (Frangioni, 2010; Zhang, 2014). Una caratteristica molto rilevante è quella che gli algoritmi basati su ricerca Greedy riescono a determinare delle soluzioni relativamente accettabili per problemi di elevata complessità, come nel caso dello studio corrente. Una versione dello pseudocodice è descritta in algoritmo 3.1, le idee base sono:

- costruzione della soluzione per passi;
- partendo da una soluzione vuota, ad ogni iterazione aggiunge un elemento alla soluzione;
- in ogni iterazione seleziona la scelta migliore sulla base della soluzione attuale (scelta miopica);
- le decisioni non vengono ridiscusse (non esegue backtracking);
- si eseguono un numero di iterazioni note a priori.

Algoritmo 3.1: Greedy search

Input: E, F
Output: $maximal(S)$
1. $S \leftarrow \emptyset$
2. $Q \leftarrow E$
3. **While** Q **is Empty or** $maximal(S)$ **do**
4. $\quad e \leftarrow best(Q)$
5. $\quad Q \leftarrow Q - \{e\}$
6. $\quad$ **If** $S \cup \{e\} \in F$ **do**
7. $\quad\quad S \leftarrow S \cup \{e\}$

Questo tipo di strategia miope implica che la bontà della soluzione dipende dal problema affrontato (Carello, 2015). Per ottimizzare al meglio il funzionamento è doveroso ricorrere ad una buona rappresentazione del problema e un'ottima selezione della funzione euristica. Ad esempio in alcuni

casi è possibile rappresentare il problema in un albero nel quale, partendo dalla radice, si esplora iterativamente il nodo più promettente. Una tecnica di esplorazione potrebbe essere basata su una strategia di ricerca informata Greedy Best-First (Zhang, 2014).

3.1.2 Ricerca locale

Gli algoritmi di ricerca locale (algoritmo 3.2) si basano su una strategia molto semplice: definita una soluzione iniziale x_i, iterativamente si cerca di rimpiazzare la soluzione corrente con una "migliore" cercando nell' intorno di essa $\sigma(x)$.

Le modalità su come esplorare l'intorno possono essere varie, ad esempio casuali, greedy o utilizzando discesa del gradiente.

Algoritmo 3.2: Local search

Input: F, x_i

Output: x

1. $x \leftarrow \sigma(x_i)$
2. **While** $score(x_i) > score(x)$ **do**
3. $\quad x \leftarrow \sigma(x_i)$

Il problema principale di questa strategia è quello di imbattersi in un ottimo locale, ovvero, delle situazioni in cui tutti gli stati vicini non portano ad un miglioramento di x_i. In tal caso è possibile ampliare l'intorno (Mello, 2017).

Uno degli algoritmi più conosciuti è chiamato Hill climbing e viene utilizzato soprattutto in problemi nei quali le variabili assumono valori discreti, mentre, in situazioni in cui le variabili sono continue e differenziabili è possibile applicare il metodo della discesa del gradiente (Mausam, 2011). Tale metodo procede iterativamente, basandosi sull'intuizione che il valore della funzione obbiettivo decresca più velocemente nella direzione negativa del gradiente (il vettore che ha per componenti le derivate parziali calcolate su tutte le direzioni) (figura 3.1).

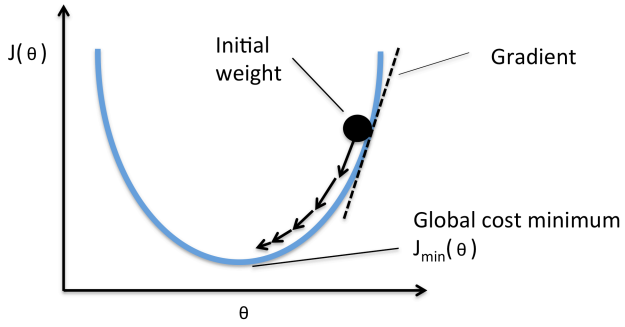


Figura 3.1

Esempio della procedura di discesa del gradiente.

Fonte: https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

Lo scopo è quello di trovare i valori da associare alle variabili θ_j che minimizzano la funzione $J(\theta_0, \dots, \theta_n)$. Per fare questo, è possibile utilizzare la discesa del gradiente, partendo da una soluzione non ottima, e iterativamente aggiornare ogni θ_j come

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n);\ \forall j$$

in modo da diminuire il più possibile la funzione di costo. L'algoritmo itera tale procedura fino ad un valore minimo di convergenza desiderato. La variabile α (definita nella fase iniziale) è chiamata learning rate e determina il coefficiente di aggiornamento di tutte le variabili. Se questo valore è troppo alto o troppo basso, l'algoritmo farà fatica a convergere perché nel primo caso l'avanzamento sarà molto brusco e il processo girerà attorno alla soluzione senza mai arrivarci, mentre nel secondo caso, il numero di iterazioni per arrivare alla soluzione saranno elevati (figura 3.2) (Raschka, 2015).

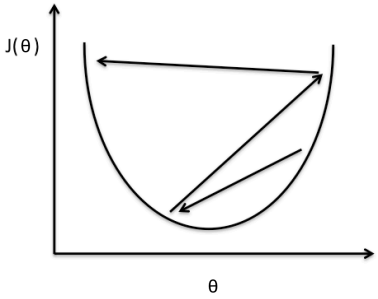


Figura 3.2.a

Discesa del gradiente con learning rate alto.

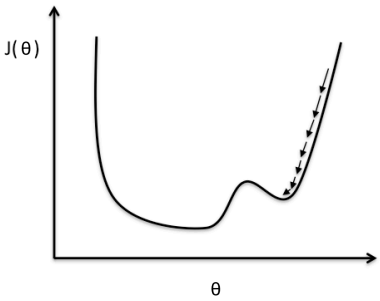


Figura 3.2.b

Discesa del gradiente con learning rate basso.

Fonte: http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

3.2 Meta-euristici

Mentre gli algoritmi euristici esplorano lo spazio sfruttando una funzione di score dipendente dal problema, quelli meta-euristici definiscono una strategia generica sfruttando dei metodi basati sull'osservazione dei fenomeni fisici o di comportamenti naturali, in questo modo è possibile costruire una tecnica di esplorazione indipendente dal problema. Per contro, richiedono un'elevata capacità computazionale, infatti vengono utilizzati comunemente per risolvere problemi di ottimizzazione NP-Hard che risultano estremamente complessi da analizzare analiticamente.

3.2.1 Simulated Anealling

Anealling, ovvero, "ricottura" è riferito ad un comportamento di alcuni metalli, vetri o cristalli che assumono una perfetta forma cristallina dopo aver subito un processo di riscaldamento sopra il loro punto di fusione, un mantenimento costante della temperatura e infine un raffreddamento molto lento. Attraverso la simulazione di questo processo è stato possibile costruire un algoritmo di ottimizzazione molto noto: il Simulated Anealling. In questa analogia, gli stati del materiale corrispondono alle soluzioni del problema, l'energia E al costo della soluzione e la temperatura T funge come controllo dei parametri.

Dalla statistica della termodinamica, la probabilità P che un sistema fisico sia in uno stato avente energia E alla temperatura T è

$$P = \frac{1}{Z} e^{-\frac{E}{k_\beta}T}$$

dove Z è un fattore di normalizzazione e k_β è la costante di Boltzmann. Come è possibile notare, se T è molto grande allora la distribuzione è molto uniforme, questo significa che l'algoritmo esplora lo spazio in modo omogeneo. Con il diminuire di T solo gli stati ad energia minima hanno una probabilità non nulla di occorrenza, ovvero, vengono esplorate solo le soluzioni più promettenti, fino ad un desiderato valore relativamente basso di T (algoritmo 3.3).

Algoritmo 3.3: Simulated Anealling

Input: $T_{min}, T_{max}, \Delta T, N$

Output: x

$x \leftarrow random\_initial\_solution()$

$T \leftarrow T_{max}$

While $T \geq T_{min}$ **do**

    For i **in** $range(0, N)$ **do**

        $\Delta x \leftarrow random\_perturbation(x)$

        $\Delta E \leftarrow E(x + \Delta x) - E(x)$

        If $\Delta E < 0$ **or** $random() < e^{-\frac{\Delta E}{T}}$ **do**

            $x \leftarrow x + \Delta x$

    $T \leftarrow T - \Delta T$

Questo algoritmo risente molto della scelta dei parametri iniziali come la temperatura iniziale, quella finale e la velocità di decadimento ΔT. Ad esempio, avere un ΔT troppo grande può portare a minimi locali (Du, 2016).

3.2.2 Evolutivi

Questo tipo di algoritmi si basano sulla procedura di selezione naturale in campo genetico. Nello specifico, durante la vita, gli individui (genitori) si accoppiano producendo nuove generazioni (figli) il cui corredo genetico rispecchia quello ereditato, con in più delle piccole mutazioni. La legge della selezione naturale impone che solo i soggetti capaci di adattarsi all'ambiente circostante possano continuare la propria esistenza e procreare a loro volta.

Le caratteristiche di un organismo sono determinate dai *geni* (le variabili del problema) ognuno dei quali può assumere diversi *alleli* (valori associati alle variabili). L'insieme di geni è detto *genotipo* e corrisponde ad una possibile soluzione del problema. Il valore di adattamento all'ambiente viene chiamato *fitness* (valore della funzione obiettivo), ragion per cui lo scopo dell'algoritmo è individuare gli alleli relativi ai geni che massimizzano questa variabile.

I passi sono i seguenti:

1. si generano di un insieme di possibili soluzioni, ciascuna con un valore di fitness, formando la popolazione iniziale;
2. si selezionano coppie di individui;
3. per ogni coppia viene applicato un operatore genetico (crossover e/o mutazione) al fine di produrre nuove soluzioni;
4. si calcola la fitness relativa ai figli aggiungendoli alla popolazione totale, dopodiché una parte di essa sarà eliminata;
5. il processo viene ripetuto fino ad una procedura di arresto.

Riguardo agli operatori genetici, il crossover crea un nuovo individuo unendo parti dei geni di entrambi i genitori, mentre la mutazione consiste nella modifica pseudocasuale di alcuni geni.

Gli algoritmi genetici, a confronto del Simulated Anealling, non sono molto sensibili ai parametri iniziali, piuttosto, possono risentire di una convergenza prematura della popolazione. Inoltre, la scelta del criterio di arresto può determinare drasticamente i tempi di calcolo (Bruno, 2010).

3.2.3 Swarm Intelligence

La Swarm Intelligence è una tecnica di intelligenza artificiale basata sullo studio del comportamento collettivo in sistemi decentralizzati e autogestiti. L'idea base è quella di creare una popolazione di agenti semplici che interagiscono localmente tra loro e con il loro ambiente. Anche se non esiste una struttura di controllo centralizzato che detta come i singoli agenti dovrebbero comportarsi, le interazioni locali fra tali agenti spesso portano alla nascita di un comportamento globale. La procedura di ricerca meta-euristica è basa su comportamenti di animali che vivono in colonie come Ant-colony, Bee-colony e Particle Swarm Optimization (PSO).

Il PSO (algoritmo 3.4) prende come esempio uno stormo di uccelli, dove ogni agente deve seguire i suoi vicini, rimanere nel gruppo ed evitare di urtarli con lo scopo comune di individuare una fonte di cibo. Ogni individuo che nel suo movimento scorge una fonte di cibo si trova di fronte a due alternative: allontanarsi dal gruppo per raggiungerlo (individualismo) o rimanere nel gruppo (socialità). Se più individui si dirigono verso il cibo anche altri membri possono cambiare la loro direzione per sfruttare la stessa fonte di nutrimento. Il gruppo cambia gradualmente direzione verso le zone più promettenti (l'informazione viene propagata gradualmente a tutti gli agenti). In un problema di ottimizzazione, gli individui campionano la funzione obiettivo (exploration) e ad ogni iterazione un individuo trae vantaggio dalle ricerche degli altri, dirigendosi verso la regione del punto migliore globalmente trovata (exploitation). La strategia di ricerca può essere espressa come bilanciamento tra exploration e exploitation. Un'altra caratteristica che risulta essere importante nella ricerca è legata al concetto di vicinanza (definita a priori tra gli agenti):

a) gli individui sono influenzati da quelli ad essi più vicini (sotto-gruppi);

b) gli individui fanno parte di più sotto-gruppi (si garantisce la circolazione dell'informazione globale).

Algoritmo 3.4: Particle Swarm Optimization

```
Input: termination_criterion(), number_of_particles
Output: g
1.  P ← initial_swarm(number_of_particles)
2.  g ← best_score(P)
3.  While termination_criterion() do
4.      For each particle in P do
5.          uptade_velocity(particle)
6.          uptade_position(particle)
7.          If score(particle) > particle.best_score do
8.              particle.best_score ← score(particle)
9.              If particle.best_score > g do
10.                 g ← particle.best_score
```

Anche in questo caso l'algoritmo risente della dipendenza dai parametri di input che possono compromettere il risultato finale, anche se esistono delle metodologie per calcolare questi parametri (Milano, 2017, A).

3.3 Algoritmi per l'addestramento di reti neurali

Gli algoritmi meta-euristici possono essere utilizzati nella ricerca delle variabili ottime nel processo di apprendimento di una rete neurale (Ojha, 2017), ma nella realtà dei fatti il metodo più utilizzato si basa sulla discesa del gradiente perché molto efficace su variabili continue. Di seguito vengono esposti i principali algoritmi per l'addestramento delle reti neurali.

3.3.1 Principali varianti della discesa del gradiente

Una rete neurale classica impara osservando molte coppie di esempi (x, y), in questo contesto chiameremo tale insieme data set. A seconda della quantità dei dati utilizzata per calcolare i gradienti della funzione obiettivo, è possibile avere tre varianti della discesa del gradiente:

1) Batch gradient descent: calcola i gradienti su tutto il data set e aggiorna in una sola volta tutte le variabili. È un approccio molto lento e inutilizzabile per insieme di dati troppo grande (in confronto alla memoria del calcolatore utilizzato). Converge nell'ottimo globale in superfici convesse, mentre in un ottimo locale in quelle non convesse.

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

2) Stocastic gradient descent (SGD): questa tipologia calcola il gradiente per ogni elemento del data set. È di gran lunga più veloce rispetto al Batch gradient descent con lo svantaggio di presentare computazioni ridondanti in presenza di data set molto grandi. L'alta frequenza di aggiornamento delle variabili rende la funzione obiettivo molto instabile, permettendo di uscire da eventuali minimi locali. Dall'altra parte questo comportamento diminuisce la probabilità di convergenza dell'algoritmo.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

3) Mini-batch gradient descent: questo metodo espande quello precedente ad un insieme di n campioni per il calcolo del gradiente. rendendo la convergenza più stabile durante il processo di ottimizzazione. Ad oggi, questo metodo è quello utilizzato durante il processo di allenamento delle reti neurali profonde.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i+n)}; y^{(i+n)})$$

La tecnica Mini-batch non è sufficiente però a garantire la buona convergenza. Alcuni problemi irrisolti sono:

- Difficoltà di scelta del parametro di learning rate;
- Modificare il learning rate durante l'allenamento può portare a dei miglioramenti, ma è molto difficile programmare quando e come il

cambiamento avvenga in modo da adattarlo ai dati specifici da analizzare;

- Si potrebbe volere che il coefficiente di aggiornamento cambi drasticamente solo per determinati campioni. Ad esempio, aumentarlo in presenza di dati a bassa frequenza e diminuirlo in quelli ad elevata frequenza;
- Nel tentavo di minimizzare la funzione in uno spazio non convesso (situazione molto comune nelle reti neurali) è possibile incrociare numerosi tipi di minimi locali. Quelli più insidiosi sono detti "saddle points" (punti di sella), in cui una dimensione presenta valori molto alti mentre le altre hanno valori molto bassi. Questi punti sono di solito accerchiati da un spazio fondamentalmente piano che rende la fuga da quest'ultimi molto ostica.

3.3.2 Momentum

Comunemente l'SGD presenta dei problemi di convergenza quando attraversa un minimo locale, procedendo nelle varie iterazioni in senso opposto (figura 3.3.a). Il momentum è un metodo che permette di stabilizzare l'andamento verso l'ottimo globale.

Figura 3.3.a

SGD senza momentum.

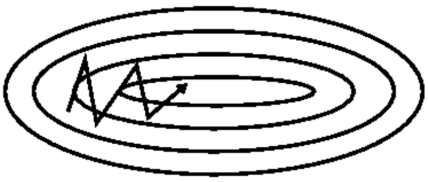


Figura 3.3.b

SGD con momentum.

Fonte: https://arxiv.org/pdf/1609.04747.pdf

In generale spinge l'algoritmo a proseguire in una direzione più rettilinea (figura 3.3.b), aggiungendo alla nuova iterazione una frazione γ (comunemente uguale a 0.9) del vecchio valore di aggiornamento:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

3.3.3 Nesterov accelerated gradient

L'algoritmo momentum, non tiene conto della direzione di esplorazione intrapresa. Con Nesterov (NAG) è possibile dare un sguardo in avanti calcolando un'approssimazione della prossima posizione

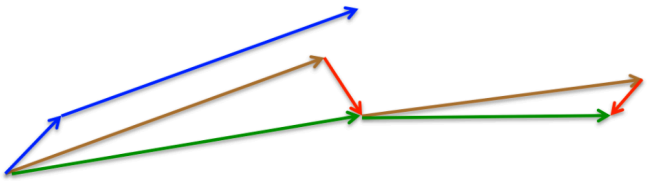


Figura 3.4

Confronto aggiornamento direzione del gradiente utilizzando momentum (blu) e Nesterov (verde).

Fonte: https://arxiv.org/pdf/1609.04747.pdf

In figura 3.4 è presente la direzione intrapresa dal momentum (blu) e dal NAG (verde) il quale produce inizialmente un grande passo in avanti verso la direzione accumulata dal gradiente (marrone), misura la pendenza (rosso) e poi calcola la correlazione (verde). Questo aggiornamento anticipato impedisce l'andamento veloce verso un'eventuale soluzione sub-ottima e porta significativi miglioramenti nelle reti ricorrenti in molti utilizzi.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

3.3.4 Adagrad

Adagrad è un algoritmo che adatta gli aggiornamenti ad ogni parametro individuale, aumentando o diminuendo il carico a seconda della sua importanza. In pratica adatta il learning rate ai parametri, aumentandolo in quelli più frequenti e riducendolo in quelli meno frequenti, per questo motivo è molto utilizzato quando i dati sono molto sparsi.

Sia $g_{t,i}$ il gradiente della funzione obiettivo al tempo t riferito al parametro θ_i, l'aggiornamento è dato da

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

dove $G_{t,ii}$ è una matrice diagonale dove gli elementi i,i sono la somma dei quadrati dei gradienti e ϵ è un termine molto piccolo (comunemente $1e-8$) che evita la divisione per zero.

Il maggior beneficio apportato da questo approccio è che non necessita del cambio manuale del learning rate durante l'allenamento, bisogna solo decidere il valore iniziale.

3.3.5 Adadelta

Adadelta è un'estensione di Adagrad e cerca di ridurre l'aggressivo decadimento monotono del learning rate prodotto dalla matrice $G_{t,ii}$, che crescendo all'aumentare di t determina l'insufficienza di aggiornamento delle nuove variabili. Invece di accumulare tutti i gradienti passati, questo nuovo metodo tiene in memoria la media E di tutti i gradienti passati e di quelli presenti come

$$E[g^2]_t = \gamma\, E[g^2]_{t-1} + (1-\gamma) g_t^2$$

e tiene in considerazione anche il decadimento medio dei parametri di aggiornamento

$$E[\Delta\theta^2]_t = \gamma\, E[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

da cui deriva la nuova metodologia di aggiornamento dei parametri:

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$
$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t}\, g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Come è possibile notare, questo algoritmo non utilizza il valore di learning rate ma solo quello del momentum γ, tipicamente impostato a 0.9.

3.3.6 RMSprop

RMSprop utilizza lo stesso processo del primo aggiornamento di Adadelta rintroducendo il learning rate:

$$E[g^2]_t = \gamma\, E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{RMS[g]_t}\, g_t$$

3.3.7 Adam

Adaptive Moment Estimation (Adam) è un altro metodo che calcola il learning rate adattandolo ad ogni parametro. Oltre a calcolare il decadimento medio dei passati gradienti come RMSprop e Adadelta (qui chiamato v_t piuttosto che $E[g^2]_t$), esso mantiene anche il decadimento esponenziale dei passati gradienti m_t:

$$m_t = \beta_1\, m_{t-1} + (1-\beta_1)\, g_t$$
$$v_t = \beta_2\, v_{t-1} + (1-\beta_2) g_t^2$$

Gli autori dell'algoritmo hanno notato che inizialmente, essendo m_t e v_t uguali a 0, nelle prime iterazioni sono propensi a rimanere vicino allo zero. Per evitare questo hanno riscritto questi valori come:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$

da cui deriva la formula di aggiornamento dei parametri:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

I valori standard sono $\beta_1 = 0.999$, $\beta_2 = 0.999$ e $\epsilon = 10^{-8}$.

3.3.8 Utilizzo degli algoritmi

La figura 3.5 fornisce un'informale intuizione del comportamento degli algoritmi appena citati in superfici problematiche.

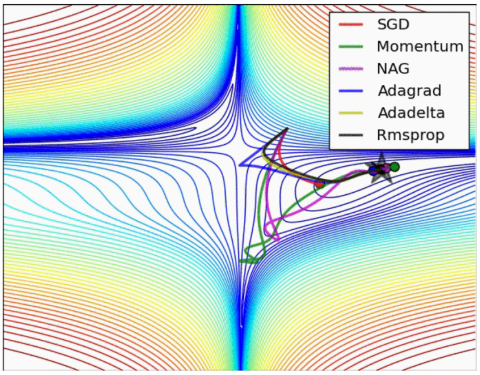


Figura 3.5.a

Ottimizzazione con punti non ottimi nei contorni.

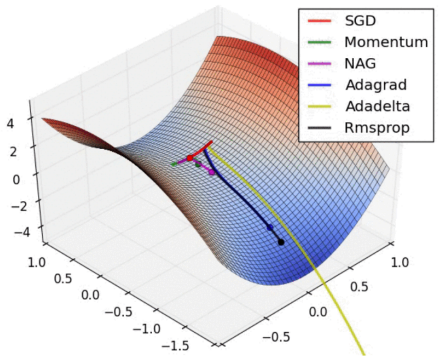


Figura 3.6.b

Ottimizzazione in un punto di sella.

Fonte: https://arxiv.org/pdf/1609.04747.pdf

Come viene indicato in (Ruder, 2017), con dati in input molto sparsi è preferibile utilizzare uno dei metodi che permettono l'auto-adattamento del learning rate con il benefit aggiuntivo di non dover cambiare nessun valore durante l'allenamento. RMSprop, Adadelta e Adam sono algoritmi molto simili come metodologie ma Adam permette di superare l'RMSprop durante la fine del processo di ottimizzazione quando i gradienti diventano sparsi.

Ultimamente, in molti studi si preferisce allenare le reti neurali profonde utilizzando l'algoritmo SGD base senza momentum inserendo una schedulazione ben progettata del learning rate secondo i dati a disposizione. Questa procedura però riscontra tutte le problematiche riportate in 3.3.1.

In sintesi, se l'obiettivo è quello di trovare una buona convergenza in tempi rapidi è sempre meglio utilizzare un metodo che prevede l'auto-adattamento del learning rate.

4 Machine Learning per la classificazione

Prima di passare alla spiegazione delle tecniche di Deep Learning, è importante definire e illustrare gli algoritmi principali utilizzati nel Machine Learning. In questo capitolo è discussa solo parte riguardante la classificazione. Dopo una breve introduzione sui concetti base, vengono introdotti gli alberi decisionali, le Support Vector Machines e infine le reti neurali classiche. Per concludere sono citate le metriche per valutare un classificatore.

4.1 Concetti base di classificazione

L'apprendimento automatico (Machine Learning) è un insieme di tecniche poste a creare un sistema induttivo, che a partire da fatti, osservazioni di un insegnante o dall'ambiente circostante apprenda la conoscenza e la generalizzi in modo che sia valida anche per casi non ancora osservati. Esistono due tipi di apprendimento induttivo:

- *supervisionato* (apprendimento da esempi): la conoscenza è acquisita a partire da un insieme di esempi positivi che sono istanze del concetto da imparare e di esempi negativi che non sono istanze del concetto;
- *non supervisionato* (apprendimento di regolarità): non c'è un concetto da imparare, l'obiettivo è quello di trovare regolarità (caratteristiche comuni) nelle istanze date.

Sia E l'insieme degli esempi positivi (e^+) e negativi (e^-) che esprimono un concetto, l'obbiettivo dell'apprendimento supervisionato è quello trovare l'ipotesi H (il concetto da imparare), tale che:

$$copre(H, e^+) = true \; \forall \; e^+ \in E^+; \; copre(H, e^-) = false \; \forall \; e^- \in E^-$$

dove $copre(H, e)$ è una funzione che ritorna valore positivo se un fatto e soddisfa l'ipotesi H. Un'ipotesi H si dice *completa* se copre tutti gli esempi

positivi mentre è *consistente* se non compre nessun esempio negativo (Milano, 2017, B; Russell & Norvig, 2016).

Nella maggior parte dei casi, i modelli di apprendimento automatico acquisiscono i fatti dai dati a disposizione. L'insieme delle informazioni disponibili è chiamato *data set*. Nella classificazione si presuppone che a fronte di n features (dati numerici vettoriali) che rappresentano un esempio del data set sia associata un'etichetta (*label*), in questo modo il modello cerca di associare determinate features ad una classe specifica. L'intero processo di classificazione può essere scomposto in cinque fasi:

1. *preparazione dei dati*: spesso i dati in formato grezzo non sono ancora pronti per essere utilizzati. Situazioni come dati mancanti o ridondanti e etichettature errate possono portare il modello ad apprendere informazioni non corrette. Per questo motivo, il data set viene filtrato e ripulito dagli esempi corrotti o non necessari. In altri casi i dati devono essere rappresentati in vettori (ad esempio testi o variabili qualitative);
2. *suddivisione*: il data set viene suddiviso in modo casuale in due porzioni distinte, il *training set* e il *test set* (tipicamente le percentuali sono rispettivamente 80% e 20%);
3. *addestramento*: il modello viene allenato con i dati del training set;
4. *stima dell'accuratezza*: il modello viene validato con degli esempi che non ha mai visto in precedenza ma di cui è possibile risalire all'etichetta reale (test set). in questo modo è possibile calcolare l'errore e in caso svolgere un altro allenamento per diminuirlo;
5. *inferenza*: una volta minimizzato l'errore è possibile classificare nuovi elementi di cui non è nota l'etichetta.

Potenzialmente, il modello creato deve generalizzare i concetti imparati il più possibile. Quando questo non accade si può incorrere a due tipi di comportamento:

- *overfitting*: il modello impara dettagli o dati errati presenti nel training set ed evidenzia questo comportamento in maniera evidente quando vengono sottoposti dati fuori dal data set;
- *underfitting*: il modello non riesce a generalizzare sia i dati del test set e sia i nuovi dati.

A seconda del tipo di classificatore utilizzato, è possibile applicare delle tecniche specifiche per evitare queste due tipologie di problemi (Sartori, 2012, A).

4.2 Tecniche di classificazione

Di seguito sono esposti i principali metodi di classificazione tramite apprendimento automatico.

4.2.1 Alberi decisionali

Gli alberi decisionali rappresentano uno degli strumenti più classici dell'apprendimento automatico. Il modello è composto da un piano strutturato ad albero che individua una successione di test da svolgere sulle features del dato in ingresso allo scopo di predire una feature non nota in uscita (label). In questa strutta un nodo rappresenta un test da svolgere su un determinato attributo, i suoi rami in uscita corrispondono alle possibili soluzioni di un dato test, mentre una foglia rappresenta il valore predetto. Questa metodologia è molto efficace quando:

- le istanze del data set sono rappresentati come coppie attributo-valore;
- l'uscita del modello è discreta;
- il data set può contiene anche dati errati o mancanti.

Uno degli algoritmi più noti in letteratura è il C4.5. Sia T un set si esempi, questo algoritmo costruisce il modello in maniera ricorsiva nei seguenti passi:

1. se T contiene uno o più esempi tutti appartenenti alla stessa classe C, viene creato un nodo foglia etichettato con C;
2. se T contiene esempi appartenenti a più classi, allora T viene partizionato secondo un attributo di test. In questo caso si procede alla creazione di un nodo associato a tale test con un sotto albero per ogni possibile risultato. A questo punto viene chiamato l'algoritmo ricorsivamente per ogni nodo creato dalla partizione;
3. se T non contiene esempi, allora si crea un nodo foglia etichettato con la classe più frequente nel nodo padre.

In linea di principio l'algoritmo generale terminerebbe quando tutte le foglie contengono esempi omogenei. In realtà C4.5 termina la procedura ricorsiva nei passi 1 e 3 (Milano, 2017, B).

Potenzialmente è possibile generare il modello scegliendo nel passo 2 uno qualsiasi degli attributi come test, ma in questo modo la profondità dell'albero aumenterebbe in modo drastico e con essa la complessità nel trovare la classe da predire. Per generare il più piccolo albero possibile è preferibile scegliere il test in modo che le partizioni di T siano insiemi il più "puri" possibili. Per eseguire tale scelta si utilizza il concetto di Information Gain (IG)

$$information\_gain(X) = H(T) - H_X(T)$$

dove $H(T)$ è l'entropia del set prima della partizione mentre $H_X(T)$ è l'entropia dopo la partizione di T secondo il test X. In pratica l'Information Gain di un attributo X è dato dalla differenza tra l'informazione originale e quella dopo la partizione su X. Nella pratica, nel punto 2 dell'algoritmo C4.5 si calcola IG per ogni test possibile e si utilizza quello con il valore finale più alto.

Ricapitolando, C4.5 esegue una ricerca locale di tipo Hill-Climbing nello spazio di tutti i possibili alberi decisionali. L'assenza di backtracking favorisce il rischio di incombere in un ottimo locale, anche se, l'utilizzo di tutti gli esempi disponibili per calcolare i vari IG rende tale algoritmo meno sensibile agli errori dovuti ai dati individuali. Un altro vantaggio nell'utilizzo degli alberi

decisionali è quello di poter risalire alla motivazione della classificazione di un certo dato, osservando i test svolti nel percorso dalla radice fino alla foglia (Milano, 2017, B; Sartori, 2012, A).

4.2.2 Support Vector Machine

Una *Support Vector Machine* (SVM) è un classificatore formalmente definito come un separatore di iperpiani. In altre parole, dato l'intero training set l'algoritmo restituisce come output i migliori iperpiani che categorizzano i dati. Ad esempio, in uno spazio bidimensionale, tale iperpiano è definito da una retta che massimizza la distanza geometrica tra le due classi di attributi (figura 4.1).

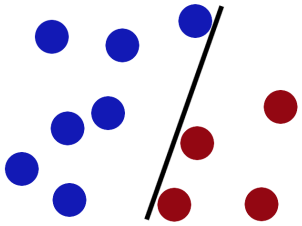


Figura 4.1

Iperpiano (linea nera) prodotto da una SVM che separa due classi differenti.

Fonte: https://www.linkedin.com/pulse/support-vector-machine-srinivas-kulkarni

Qualora il confine non fosse linearmente separabile, occorre effettuare una trasformazione dei dati mediante un *kernel* ϕ in uno spazio avente un numero di dimensioni più elevato (figura 4.2).

La classificazione mediante SVM è quindi formulata come un problema di ottimizzazione convessa. Tale tecnica non ha bisogno di molti dati per raggiungere alte performance, anzi, un numero troppo elevato può presentare un costo in termini di tempo nel calcolo degli iperpiani. Inoltre, presenta problemi di performance nei casi in cui i dati sono sovrapposti, in questo caso, bisogna procedere utilizzando dei parametri di regolarizzazione per migliorare il risultato (Manning et al., 2009; Sartori, 2012, B).

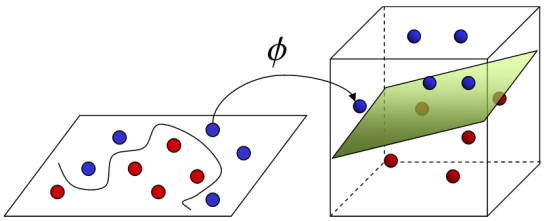


Figura 4.2

Tramite il kernel ϕ i dati sono mappati in un nuovo spazio (parte destra), dentro il quale è possibile svolgere una separazione tramite SVM.

Fonte: https://www.linkedin.com/pulse/support-vector-machine-srinivas-kulkarni

4.2.3 Reti neurali

Un altro tipo di approccio, totalmente differente da quelli visti fino ad ora, è quello basato sulla simulazione dei meccanismi che avvengono nel cervello umano, composto da miliardi di neuroni interconnessi fra loro. L'idea è quella di costruire un modello formato da una rete di neuroni artificiali che apprenda dai dati autonomamente (senza imporre particolari regole).

Il neurone biologico è composto principalmente da nucleo, assone e dendriti. Il collegamento che un neurone crea con l'altro è chiamato sinapsi. Il flusso di "informazioni" che attraversa le varie cellule in questione parte dalle sinapsi, passa per i dendriti per poi venire "elaborato" dal nucleo. In questo modo la nuova informazione viene ricondotta in uscita ad altri neuroni tramite gli assoni. Questo processo viene effettuato tramite scariche elettriche che, passando attraverso il neurone, possono essere propagate o meno a seconda della differenza di potenziale necessaria per l'attivazione (Milano, 2017, C).

Partendo da queste semplici osservazioni, nel 1962, Rosenblatt ha elaborato il primo modello matematico che apprende da esempi: il *Percettrone*. Esso è composto da un numero finito di parametri di input, ognuno dei quali riferito ad un segnale di ingresso diverso x_i, che a sua volta viene amplificato o ridotto a seconda del valore del peso w_i. Questi segnali vengono quindi agglomerati secondo delle specifiche (solitamente sommati tra loro inserendo inoltre un bias b). Infine, l'uscita viene distorta tramite una funzione di soglia f (figura 4.3).

Le funzioni di soglia possono essere di vario tipo, le più comuni sono funzioni sigmoidee, lineari, gradino o gaussiane.

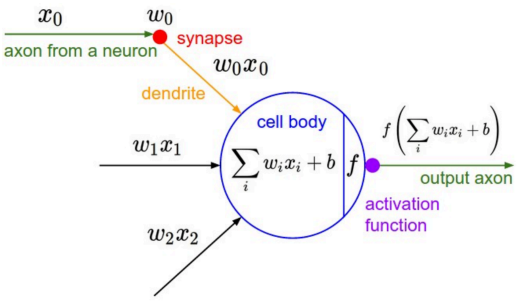


Figura 4.3

Modello matematico del Perceptron.

Fonte: http://cs231n.github.io/convolutional-networks/

Nel 1969 Minsky e Papert dimostrarono le limitazioni di questo modello, ovvero, la sua capacità di riconoscere solamente classi di problemi con soluzioni linearmente separabili. Questo problema fu superato verso la metà degli anni 70 con il *Multi-Layers Perceptron* (MLP) il quale permette di avere più livelli composti ognuno da più Percettroni, i quali sono connessi con tutti i neuroni del livello successivo e antecedente, ma non tra neuroni artificiali dello stesso livello (figura 3.4). Questo tipo di strato viene anche chiamato *fully conneceted layer*.

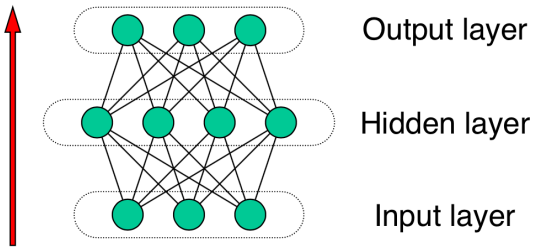


Figura 4.4

Multi-Layers Perceptron a tre strati. I neuroni sono rappresentati in verde.

Fonte: http://ai.unibo.it/webfm_send/238

Questo modello con tre livelli riesce a separare regioni convesse (se il numero di angoli è minore o uguale al numero di neuroni presenti nello strato nascosto), mentre con quattro livelli, potenzialmente, può separare regioni di qualsiasi tipo. L'aggiunta di nuovi strati non aumenta le abilità di classificazione. Un altro aspetto importante è che se i neuroni sono lineari, allora è possibile ridurre la rete fino a soli due strati, quindi per classificare oggetti complessi bisogna che i neuroni presentino funzioni di attivazione non lineari (Milano, 2017, C).

L'allenamento di una rete neurale viene svolto tramite l'algoritmo *Backpropagation* (algoritmo 4.1) il quale utilizza metodi di ottimizzazione a discesa del gradiente per far in modo che il modello apprenda e generalizzi i dati del training set, in modo tale da raggiungere una adeguata convergenza.

Algoritmo 4.1: Backpropagation

Input: $network$, $training\_set$, $learning\_rate$, min_error
Output: $best\_node$
1. $network.initialize\_weights\_randomly()$
2. **While** $global\_error > min\_error$ **do**
3. $\quad global\_error \leftarrow 0$
4. $\quad$ **For each** $example$ **in** $training\_set$ **do**
5. $\quad\quad error \leftarrow network.forward\_step(example)$
6. $\quad\quad gradients \leftarrow network.compute\_gradients(error)$
7. $\quad\quad network.update\_weights(gradients, learning\_rate)$
8. $\quad\quad global\_error \leftarrow global\_error + error$

Come è possibile osservare in Algoritmo 4.1, inizialmente la Backpropagation effettua una predizione utilizzando i pesi correnti (*forward step*), e successivamente calcola iterativamente i gradienti per ogni livello in maniera inversa, partendo da quello in uscita fino a ritornare al primo (*backward step*). L'algoritmo opera questo processo per ogni esempio del training set, dopodiché decide se ripetere una nuova *epoca* (allenamento univoco su tutto il training

set) a seconda dell'errore globale raggiunto. Per migliorare l'apprendimento, vengono svolte le seguenti accortezze:

- l'inizializzazione dei pesi viene svolta in modo che abbiano valori molto piccoli e normalizzati;
- essendo un algoritmo di ricerca locale è buona norma ripetere più volte l'allenamento;
- nella fase di preparazione dei dati è opportuno eliminare quelli inconsistenti. Infatti, la convergenza non è garantita se gli esempi contengono etichette errate;
- per favorire la generalizzazione ed evitare overfitting il dataset viene suddiviso in tre: training set, test set e *validation set*. Quest'ultimo è utilizzato in fase di allenamento per controllare la convergenza del modello. Ad esempio, se l'accuratezza finale tra test set e validation set è molto differente, allora si ha overfitting e quindi bisogna ripetere l'allenamento.

Il modello MLP è uno strumento altamente potente e può essere considerato un framework di approssimazione universale. Purtroppo in pratica, i problemi dovuti alla convergenza e la difficoltà di decidere il numero di neuroni interni rendono questo modello altamente complesso da allenare (Milano, 2017, C).

4.3 Valutazione di un classificatore

Una volta creato il modello di classificazione è importante valutare le sue prestazioni servendosi del test set. Nel caso più semplice è possibile calcolare l'accuratezza tramite il rapporto tra numero di classificazioni corrette e numero di classificazioni totali. A volte, solo questa informazione non è sufficiente per valutare il corretto funzionamento del classificatore, quindi si ricorre ad altri tipi di metriche più specifiche. Nella tabella 4.1 sono indicate le metriche più comuni (Sartori, 2012, C).

Metodologia	Formula	Descrizione
Accuratezza (*accuracy*)	$\frac{TP + TN}{TP + TN + FP + FN}$	Percentuale delle istanze classificate correttamente.
Precisione (*precision*)	$\frac{TP}{TP + FP}$	Percentuale dei positivi classificati correttamente.
Recupero (*recall*)	$\frac{TP}{TP + FN}$	Tasso di veri positivi.
F1	$2\frac{precision \cdot recall}{precision + recall}$	Media armonica tra precisione e recupero.

Tabella 4.1

Metriche principali per la valutazione di un classificatore, le quali hanno dominio tra 0 e 1. Si prenda come riferimento la figura 4.5.

	Predicted	
	Positive	Negative
Actual True	TP	FN
Actual False	FP	TN

Figura 4.5

Possibili combinazioni di risultati in un processo di valutazione.

Fonte: https://docs.microsoft.com/it-it/azure/machine-learning/studio/evaluate-model-performance

In questo studio vengono utilizzate anche altre due metriche, impiegate quando si creano modelli regressivi: la *Root Mean Squared Error* (RMSE) e la *Mean Absolute Error* (MAE), rispettivamente calcolate come

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

dove n è il numero di campioni nel test set, y_i è la classe reale, mentre $\hat{y}_i$ la predizione effettuata dal classificatore. Entrambe rappresentano l'errore medio di predizione del classificatore con dominio $[0, \infty]$, inoltre, la RMSE è sempre minore o uguale alla MEA. La differenza sostanziale tra le due è che la RMSE individua anche la varianza tra gli errori dei singoli campioni (Chudzicki, 2012).

5 Deep Learning

Le tecniche di Machine Learning tradizionali sono molto efficaci ma negli ultimi anni, il Deep Learning ha portato una grande rivoluzione, migliorando le performance e imponendosi in campi come Speech recognition, Machine translation, Object recognition, Video undertanding, Natuaral Language Processing e Game playing (Milano, 2017, D).

5.1 Tecniche principali

Queste nuove tecniche derivano delle reti neurali e introducono metodologie che permettono al modello di superare i limiti del Multi-Layers Perceptron. Le reti MLP con almeno uno strato nascosto hanno la possibilità di imparare qualsiasi schema non lineare dai dati, rappresentando un vero e proprio framework di approssimazione universale. Sfortunatamente, quando i dati da classificare sono molto complessi, il livello nascosto deve contenere molti neuroni e apprendere un numero molto grande informazioni, di conseguenza, aumenta la difficoltà di trovare appropriati valori dei pesi e il rischio di overfitting. Questo paragrafo espone una visione generale delle innovazioni proposte dal Deep Learning, che permettono di aumentare il numero di livelli nascosti, specializzando ognuno di essi a riconoscere una determinata tipologia di caratteristica in modo automatico (Milano, 2017, D).

5.1.1 Autoencoders e Deep Belief Networks

Gli Autoencoders sono un tipo di reti non supervisionate, le quali hanno la particolarità di imparare una rappresentazione di input (x) e ricostruirla in output (r) cercando di catturare le caratteristiche più rilevanti (figura 5.1). Sia h lo strato nascosto, vengono definite le funzioni

$$h = f(x);\ r = g(h)$$

dove f è chiamato *encoder* mentre g *decoder*. Il modello ha lo scopo di minimizzare la dissimilarità tra x e r.

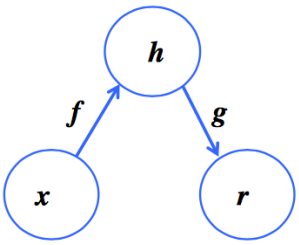


Figura 5.1

Rappresentazione di un Autoencoder con un livello nascosto di encoder e decoder.

Fonte: http://ai.unibo.it/webfm_send/239

Una particolarità di queste reti è che nel caso in cui il numero di neuroni di output è maggiore del numero dei neuroni di input, il modello estrarrà le caratteristiche più rilevanti dei dati. Mentre, nel caso contrario è possibile scoprire relazioni non lineari.

L'esempio con cui è stato illustrato il funzionamento è costituito da un solo layer. In realtà, nella maggior parte dei casi, vengono utilizzati più livelli in cascata piuttosto che un singolo livello molto largo, infatti, la profondità introduce dei vantaggi molto importanti come: ridurre il costo computazionale, ridurre il numero di dati necessari del training set e aumentare l'accuratezza. Come è possibile osservare in figura 5.2, l'allenamento di tipo non supervisionato, in questo caso, avviene a livelli e una volta terminato si può procedere anche con alcuni step di apprendimento supervisionato tramite back-propagation (fine-tuning). Tale procedimento viene chiamato apprendimento semi-supervisionato ed è molto più performante rispetto al classico MLP supervisionato, perché allenare l'intero modello tramite la sola back-propagation porta a problemi di aggiornamento dei pesi (*vanishing gradient*) nei layer più lontani dall'output (Goodfellow et al., 2016; Lippi, 2016, A; Milano, 2017, D).

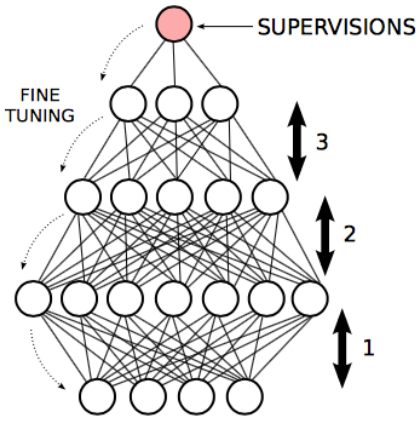


Figura 5.2

Rappresentazione di un allenamento semi-supervisionato: inizialmente vengono allenati gli Autoencoders in cascata (1,2,3) e successivamente si passa alla fase supervisionata tramite fine-tuning.

Fonte: http://lia.disi.unibo.it/Staff/MarcoLippi/teaching/lecture_2.pdf

Un'altra tipologia di rete introdotta, è la Deep Belief Networks, la quale ha lo stesso schema delle Autoencoders (figura 5.1) solo che ogni layer è una Restricted Boltzman Machine, ovvero, introduce una fattorizzazione di probabilità tra i layer. Queste ultime hanno avuto molto successo nella classificazione di immagini e documenti (Goodfellow et al., 2016; Lippi, 2016, A; Milano, 2017, D).

5.1.2 Dropout

Il *dropout* è una metodologia che ha lo scopo di evitare overfitting. L'idea principale è quella di chiudere delle connessioni in maniera casuale solo durante l'allenamento, riducendo nell'atto pratico la grandezza della rete (figura 5.3).

In questo modo una rete con n unità può essere vista come la combinazione di 2^n sub-reti. Quindi, in altri termini, la combinazione di più reti porta ad un'accuratezza più elevata. Lo svantaggio è quello di rendere più difficoltosa la fase di apprendimento (Goodfellow et al., 2016; Lippi, 2016, B; Milano, 2017, D).

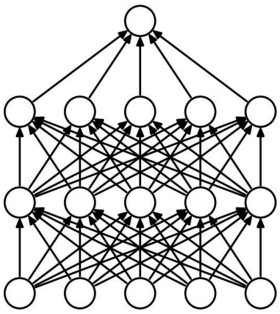


Figura 5.3.a

Rete neurale standard.

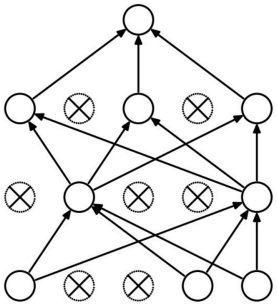


Figura 5.3.b

Rete neurale dopo l'applicazione del dropout.

Fonte: http://ai.unibo.it/webfm_send/239

5.1.3 Convolutional Neural Network

Le reti MLP utilizzano un'architettura fully connected, nella quale ogni neurone di ciascun layer è collegato a tutti i neuroni del layer precedente. Questo meccanismo non è indicato se l'input ha dimensioni molto grandi, per via dei problemi di scalabilità. Le Convolutional Neural Networks (CNN) sono delle reti neurali ottimizzate per processare immagini, o in alcuni casi sequenze di dati, introducendo nuovi tipi di livelli che hanno permesso la creazione di modelli molto profondi, complessi e soprattutto accurati (Goodfellow et al., 2016; Milano, 2017, D).

Lo strumento principale è il convolutional layer, un filtro che esegue l'operazione matematica di convoluzione tra il livello precedente e un kernel contenente i valori dei pesi allenabili. In questo modo è possibile connettere localmente ogni neurone con una sola regione di input. Per esempio, supponiamo di avere un ingresso con dimensione $[32, 32, 3]$, se la grandezza del filtro è 5x5, ogni neurone del nuovo layer avrà un volume di $[5, 5, 3]$ per un totale di 75 pesi più un parametro di bias (figura 5.4) (Lippi, 2016, B).

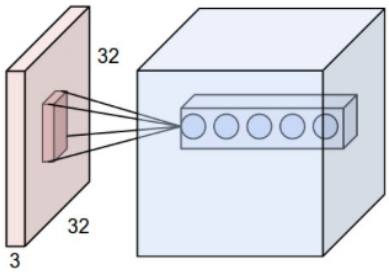


Figura 5.4

Esempio di un operazione di convoluzione svolta nel convolutional layer.

Fonte: http://cs231n.github.io/convolutional-networks/

La grandezza di ogni livello sarà quindi dipendente dal filtro e dalla modalità con cui viene utilizzato. Tali caratteristiche vengono impostate tramite quattro parametri:

- dimensione del filtro (F);
- numero di filtri da applicare (K);
- stride (S): l'offset di scorrimento del filtro;
- padding (P): permette di inserire una cornice formata da elementi uguali a 0 intorno all'input (P indica la grandezza della cornice).

In definitiva, dato un input di dimensioni $[W_i, H_i, D_i]$ è possibile calcolare la grandezza del livello successivo $[W_o, H_o, D_o]$ come

$$W_o = (W_i - F + 2P)/(S + 1)$$
$$H_o = (H_i - F + 2P)/(S + 1)$$
$$D_o = K$$

dove D rappresenta anche il numero di feature maps (Goodfellow et al., 2016; Lippi, 2016, B). Un esempio è osservabile in figura 5.5.

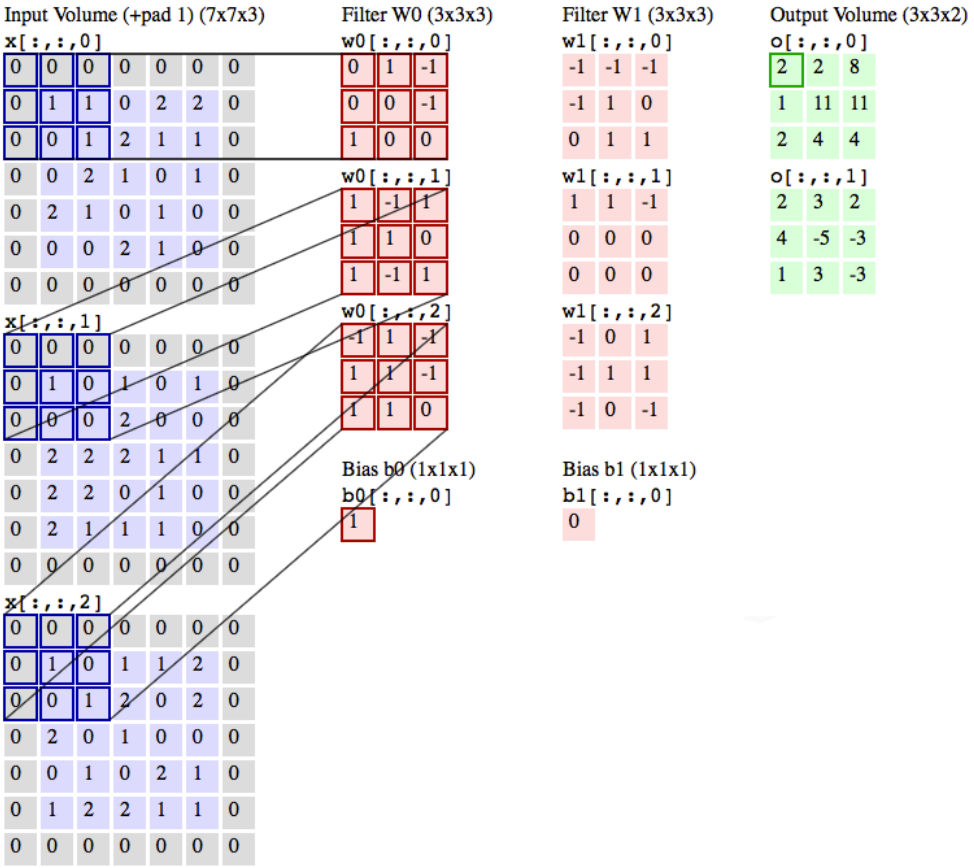


Figura 5.5

Esempio di creazione di un nuovo convolutional layer, utilizzando come input tre feature maps 7x7 e applicando i seguenti criteri: $F = 3$, $K = 2$, $S = 1$ *e* $P = 1$.

Fonte: http://cs231n.github.io/convolutional-networks/

Un'altra tipologia di strato è il pooling layer, il quale non contiene pesi allenabili. Dati i parametri F, K, S e P, si esegue una semplice operazione matematica, ad esempio portando in uscita il massimo valore tra gli elementi della finestra di input (*max pooling*, figura 5.6) (Goodfellow et al., 2016; Lippi, 2016, B).

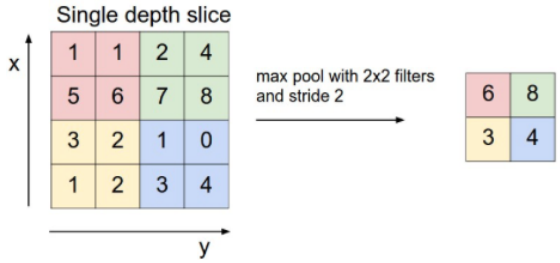


Figura 5.6

Esempio di un max pooling layer.

Fonte: https://nrupatunga.github.io/2016/05/14/convolution-arithmetic-in-deep-learning-part-1/

Grazie a questi due nuovi tipi di layers, le CNN sfruttano tre importanti idee che migliorano un sistema di Machine Learning (Goodfellow et al., 2016; Milano, 2017, D):

1. le connessioni locali permettono di utilizzare meno parametri e di conseguenza aumentare l'efficienza;
2. la condivisione del medesimo filtro lungo tutta l'immagine è un altro fattore che permette a questi sistemi di diminuire il costo computazionale;
3. questi sistemi sono invarianti al cambio di scala e di rotazione.

5.1.4 Recurrent Neural Network

Le reti osservate in 5.1.1 e 5.1.3 ricavano tutta l'informazione necessaria solo dal dato in ingresso al tempo t, nello specifico la predizione svolta al tempo $t + 1$ non tiene conto anche dell'input precedente ma solo di quello corrente. Le Recurrent Neural Network (RNN) permettono di superare questo limite e gestire dati di tipo sequenziale memorizzando internamente anche le informazioni introdotte nei passi precedenti. Un altro vantaggio, è la capacità di creare svariati tipi di configurazioni input/output a seconda del problema da risolvere (figura 5.7) (Goodfellow et al., 2016; Lippi, 2016, C).

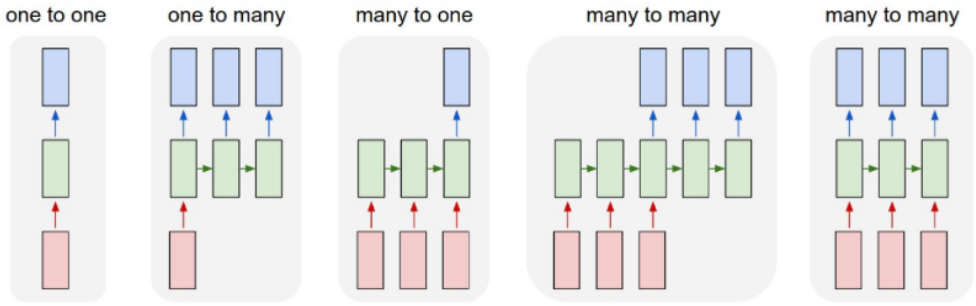


Figura 5.7

Le possibili configurazioni di un layer RNN.

Fonte: http://lia.disi.unibo.it/Staff/MarcoLippi/teaching/lecture_4.pdf

Come è possibile osservare in figura 5.8, una rete ricorrente calcola l'uscita h_t prendendo in ingresso la nuova informazione x_t e l'uscita precedente h_{t-1}. Un layer è dunque formato inserendo più neuroni in cascata.

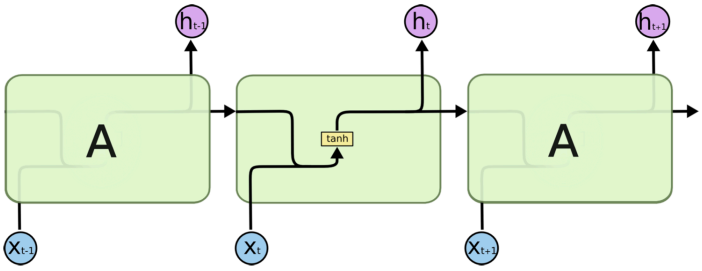


Figura 5.8

Struttura interna di una RNN. Il riquadro giallo indica un livello di neural network classico con la relativa funzione di attivazione a tangente iperbolica (tanh).

Fonte: http://lia.disi.unibo.it/Staff/MarcoLippi/teaching/lecture_4.pdf

Il problema di questi layer è quello di non riuscire a imparare dipendenze a lungo termine, per questo motivo sono state sviluppate delle nuove celle di memoria chiamate Long Short-Time Memory (LSTM) che sostituiscono il classico neurone. I layer LSTM (figura 5.8) possono essere utilizzati singolarmente (ad esempio per la classificazione di testi), oppure dopo dei

convolutional layes per classificare oggetti in movimento in un video (Goodfellow et al., 2016; Lippi, 2016, C).

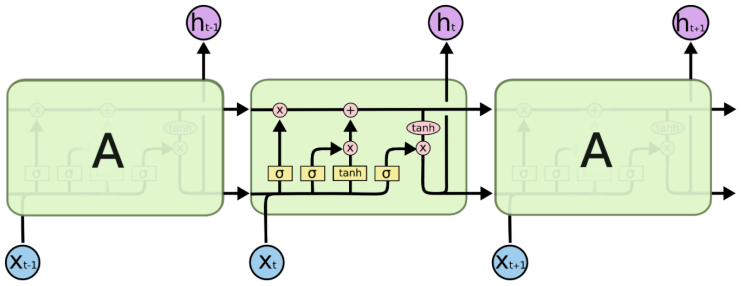


Figura 5.9

Struttura interna di una LSTM. I riquadri gialli indicano un livello di neural network classico con la relativa funzione di attivazione: sigmoidea (σ) o tangente iperbolica (tanh). Le parti in rosa indicano delle operazioni pointwise.

Fonte: http://lia.disi.unibo.it/Staff/MarcoLippi/teaching/lecture_4.pdf

5.2 Transfer learning

Il processo di creazione di un modello di apprendimento automatico tramite reti neurali profonde richiede un insieme di esempi molto grande e prevede molti tentativi prima di creare una struttura che porti ad un buon risultato. In pratica, poche persone allenano le reti neurali partendo da pesi inizializzati in modo casuale perché è molto raro avere una grande disponibilità di dati che permettono la buona convergenza della rete. Invece, è molto comune prendere un modello già costruito e funzionante su un data set molto grande, per poi allenarlo di nuovo sui propri dati. Questo processo è noto con transfer learning e prevede tre casi di applicazione:

1. prendere una rete allenata precedentemente, rimuovere il suo ultimo layer di classificazione e sostituirlo con uno nuovo inizializzato con pesi casuali. Dopodiché allenare solo l'ultimo livello trattando il resto del modello come un estrattore di features;

2. sostituire l'ultimo livello come nel caso 1 e poi allenare tutta la rete. In pratica si utilizzano i pesi della rete pre-allenata come inizializzazione del processo di training (fine-tuning);
3. utilizzare la tecnica discussa nel caso 2 ma partendo da pesi presi durante la fase di allenamento della rete di riferimento, piuttosto che alla fine.

Nella tabella 5.1 sono descritte delle pratiche di riferimento su come utilizzare queste tecniche in base alla grandezza dei data set.

	D_{new} ha contenuti simili rispetto a D_{old}	D_{new} ha contenuti diversi rispetto a D_{old}
La dimensione di D_{new} è piccola	Caso 1	Casi 3 e 2
La dimensione di D_{new} è grande	Caso 2	Caso 2

Tabella 5.1

Pratiche comuni di utilizzo delle tecniche di tranfer learning. D_{old} è il data set utilizzato per la rete pre-allenata, mentre D_{new} rappresenta il nuovo data set su cui si desidera estrarre le nuove informazioni.

Inoltre, in caso si utilizzasse fine-tuning, è buona regola tener conto del fatto che il learning rate iniziale dovrà essere più basso in confronto a una classificazione di uno o più livelli casualmente inizializzati (Karpathy, 2017).

5.3 Architetture CNN

Le architetture sviluppate utilizzando tecniche di Deep Learning sono altamente personalizzabili, per questo presentano strutture molto più complesse rispetto alle reti neurali classiche (paragrafo 4.2.3), soprattutto se si utilizzano le CNN. Per questo motivo, nei casi in cui si voglia classificare delle immagini o localizzare oggetti al loro interno è preferibile utilizzare delle architetture

generiche che hanno riscontrato un ottimo funzionamento in data set standard, come MNIST o ImageNet, per poi adattarle al caso d'uso specifico.

La struttura base di una CNN è costituita inizialmente da una serie di livelli di convoluzione e max pooling alternati tra loro, in modo tale da estrarre le caratteristiche principali dell'immagine e sottoporle successivamente come input ad una classica rete neurale fully connected, di solito a 3 livelli. Questa struttura è stata ideata da Yann LeCun et al. nel 1998 ed impiegata nel riconoscimento di numeri scritti a mano (data set MNIST). Recentemente, le CNN più utilizzate derivano da quelle che ottengono ottimi risultati ad una competizione che le università di Stanford e Princeton organizzano ogni anno per scopi di ricerca. Un obiettivo, ad esempio, è quello di creare modelli avanzati per classificare e localizzare oggetti in un'immagine impiegando un data set, estrapolato da ImageNet, con 1000 classi differenti. Di seguito vengono esposti i modelli più rilevanti.

5.3.1 AlexNet

AlexNet è stata la prima rete neurale profonda che ha ottenuto risultati considerevoli su ImageNet (LSVRC-2010). Essa utilizza una composizione molto simile a quella di LeNet-5, aggiungendo delle nuove tecniche come il dropout tra i livelli fully connected, un ReLu dopo ogni livello di convoluzione e una normalizzazione nei due primi livelli di max pooling. Inoltre, per aumentare la velocità durante l'allenamento, sono state utilizzate due GPU in parallelo che operano nello stesso modello (figura 5.10) (Krizhevsky et al., 2012).

Come è possibile notare in figura 5.10, l'immagine iniziale di grandezza 224x244 viene filtrata da 96 convoluzioni 11x11, in questo modo vengono create altrettante feauture maps utilizzate come ingresso per il filtro successivo. Andando avanti con i livelli, il numero di feature-maps aumenta mentre diminuisce la grandezza di ognuna di esse, in questo modo durante l'allenamento la rete impara caratteristiche sempre più specifiche ad ogni livello, le quali, alla fine del quinto, vengono utilizzate per la classificazione tramite layer fully connected. La struttura così composta è formata da più di

62 milioni di pesi, di cui 58 solo nei livelli di fully connected (Krizhevsky et al., 2012).

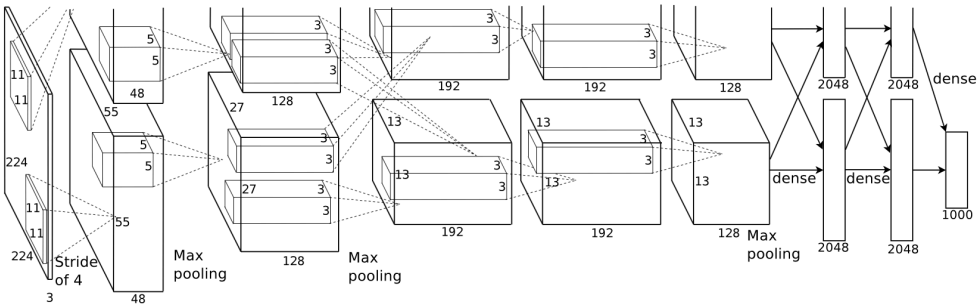


Figura 5.10

Struttura della rete AlexNet con due GPU in parallelo. Durante l'allenamento i parametri vengono condivisi tra il secondo e il terzo livello e tra gli ultimi tre.

Fonte: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

5.3.2 VGG

Le VGG sono un insieme di reti presentante nel corso del ILSVRC-2014. Come è visibile in figura 5.11, queste reti sono molto simili tra loro ma il fattore discriminante è il numero di livelli, variabile da 11 a 19. La struttura, rispetto ad AlexNet, è rimasta la stessa ma, in confronto, le VGG sono più profonde e presentano molti più parametri allenabili: tra i 133 ai 144 milioni.

È possibile notare che non sono presenti convolutional layers con grandezza del filtro superiore a 3. L'idea è stata quella di simulare filtri di dimensione maggiore utilizzando in cascata delle convoluzioni 3x3, in questo modo si sono introdotte più funzioni di ReLu (alla fine di ogni layer) incrementando la capacità della rete di apprendere relazioni non lineari. Nella VGG-16 sono presenti anche convoluzioni 1x1, anch'essi con lo scopo di aumentare la non linearità, senza diminuire la dimensione delle feature maps in uscita. Tutti i layers di max pooling operano senza sovrapporre la finestra di scorrimento (kernel e stride pari a 2), mentre i convolutional layers hanno uno stride pari a 1 (Simonyan & Zisserman, 2015).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 5.11

Struttura delle reti VGG. In grassetto è possibile notare le differenze con la configurazione più a sinistra.

Fonte: https://arxiv.org/pdf/1409.1556.pdf

Tramite questi espedienti è stato dimostrato che una rete più profonda apprende più informazioni, aumentando l'accuratezza. Per contro, la rete occupa molto spazio disco (più di 500 MB) e il numero maggiore di parametri da allenare impone che l'hardware da utilizzare sia molto rilevante, infatti, sono state impiegate 4 NVIDIA Titan Black con 6 GB di memoria ciascuna durante l'allenamento (Simonyan & Zisserman, 2015).

5.3.3 Network-in-Network

Le NIN (Network-In-Network) sono reti che rappresentano un nuovo tipo di standard. Il team di progettazione è dell'idea che i filtri convoluzionali hanno un livello di astrazione basso e di fatto non rappresentano il concetto di simulazione del sistema biologico umano. In pratica, il classico MLP viene

rivisitato in una nuova struttura, l'MLP Convolutional Layer (MLPCL), formata consecutivamente da:

- layer di convoluzione classico (con valori personalizzabili);
- ReLu;
- layer di convoluzione 1x1 e stride 1
- ReLu;
- layer di convoluzione 1x1 e stride 1
- ReLu;
- max pooling 3x3 con stride pari a 2.

Viene proposta quindi una variante del MLP nella quale i fully connected vengono sostituiti con delle convoluzioni 1x1 (Min et al., 2014).

Una rete NIN (figura 5.12) è formata da una serie di MLPCL e infine un nuovo tipo di pooling, chiamato *global average pooling*.

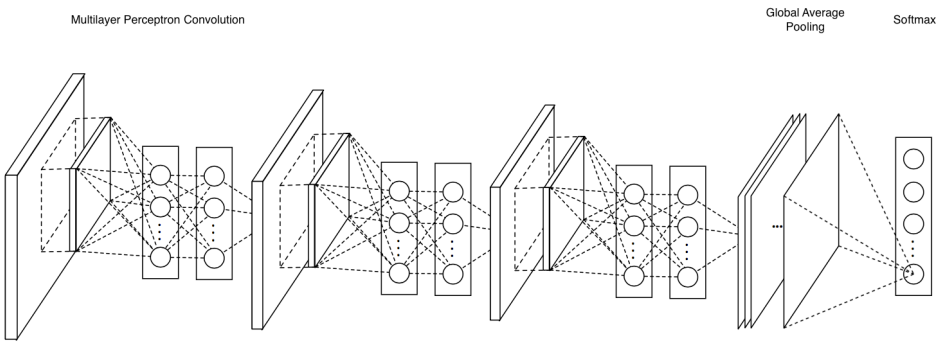


Figura 5.12

Struttura della rete NIN.

Fonte: https://arxiv.org/pdf/1312.4400.pdf

Il global averenge pooling è stato introdotto per la prima volta in questo contesto con lo scopo di sostituire gli ultimi layer fully connected. La sua struttura è composta da una feature map per ogni classe da predire, effettuando una media su quelle in ingresso per poi calcolare la softmax (Min et al., 2014).

Non essendoci parametri allenabili, questo tipo di operazione previene l'overfitting risparmiando anche molta memoria, infatti la NIN presenta solo 7.5 milioni di parametri allenabili, ha performance molto simili a quelle di AlexNet occupando però solo 30 MB di spazio disco (Min et al., 2014).

5.3.4 GoogLeNet

Come sottolineato nel paragrafo 5.3.2, aumentare la profondità della rete può portare all'aumento di accuratezza. Continuando con questa filosofia, il team di Google ha formulato una rete molto profonda chiamata GoogLeNet (che prende il nome dalla rete LeNet-5), successivamente rinominata Inception-v1. La struttura presenta ben 12 livelli e, come le VGG, contiene filtri convoluzionali di dimensioni ridotte: 1x1, 3x3 e 5x5.

Quando si tenta di creare una rete molto profonda, il numero dei parametri e il conseguente costo computazionale aumentano vertiginosamente. Per questo motivo sono state adottate delle contromisure:

1. la GoogLeNet non fa uso di molti layers fully connected, se non nell'ultimo livello (il team spiega inoltre che non è essenziale la sua presenza ma che è stata aggiunta solo per migliorare ulteriormente le prestazioni);
2. le convoluzioni 1x1 sono utilizzate anche per ridurre il numero di feature maps tra un livello e l'altro prima di effettuare convoluzioni più onerose, rimuovendo eventuali colli di bottiglia.

In questo modo è possibile sviluppare la rete in lunghezza senza un significativo calo delle performance (Szegedy et al., 2014).

Come nelle NIN, anche in questo caso è stato progettato un nuovo tipo di strato chiamato *inception layer*, composto da:

- convoluzione 1x1;
- convoluzione 3x3;
- convoluzione 5x5;

- max pooling 3x3.

Questi layer sono posizionati in parallelo, in modo tale che l'uscita dell'inception layer sia la concatenazione delle loro uscite. Il motivo per cui è stato creato in questo modo parte dall'osservazione che, a seconda del layer utilizzato, vengono estratte alcune caratteristiche piuttosto che altre. In questo modo è possibile apprendere molta più informazione, in confronto alle altre reti AlexNet, VGG e NIN. In figura 5.13 è visibile la versione utilizzata per formare la GoogLeNet. Come descritto in precedenza, l'aggiunta delle convoluzioni 1x1 previene l'aumento del numero di parametri da allenare, inoltre, è importante sottolineare che si ha sempre un ReLu dopo una convoluzione (Szegedy et al., 2014).

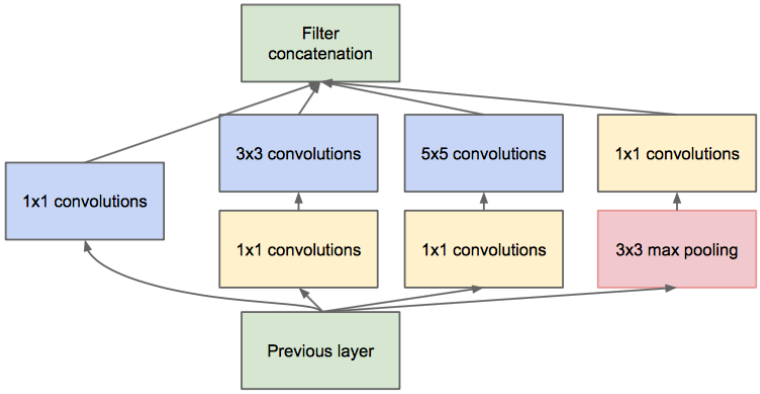


Figura 5.13

Struttura di un inception layer.

Fonte: https://arxiv.org/pdf/1409.4842.pdf

La struttura finale della GoogLeNet (figura 5.14) è formata da una parte iniziale di convoluzioni e max pooling, per poi proseguire con vari inception layer in cascata. Da notare che, come le NIN, è presente un global average pooling prima dell'ultimo fully connected layer. Questa procedura è stata ideata per permettere il fine-tuning quando si vuole cambiare il numero di classi. Essendo l'architettura molto profonda, in alcune versioni vengono inserite altre uscite nei livelli intermedi, in modo tale da poter recuperare le features che

potrebbero essere importanti per la classificazione, ma che vengono perse durante il tragitto (Szegedy et al., 2014).

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figura 5.14

Architettura della rete GoogLeNet.

Fonte: https://arxiv.org/pdf/1409.4842.pdf

5.4 Metodi di localizzazione tramite CNN

Fino ad ora, sono stati esaminati vari tipi di reti neurali profonde, molto accurate e complesse, in grado però di svolgere un solo compito: classificare. In questo studio le tecniche di Deep Learning sono state utilizzate per localizzare un codice Data Matrix dentro un'immagine. Per questo motivo è doveroso introdurre i principali framework che permettono di classificare e localizzare oggetti contemporaneamente.

5.4.1 OverFeat

Creato con la collaborazione di Yann LeCun, OverFeat è uno dei primi framework il cui intento è quello di creare una rete neurale che allo stesso tempo svolga più mansioni: classificare e localizzare uno o più oggetti in un'immagine. Questo approccio ha permesso di vincere la competizione "ImageNet Large Scale Visual Recognition Challenge" del 2013. Di seguito

vengono esposti i vari passi che hanno portato ad ottenere tale risultato (Sermanet et al., 2014).

La costruzione della rete è ispirata a quella di AlexNet con alcune modifiche (figura 5.15):

- vengono eliminati i layer di normalizzazione;
- non c'è sovrapposizione nei layer di max pooling;
- i primi due livelli hanno più feature maps ma stride più piccoli per aumentare l'accuratezza.

Layer	1	2	3	4	5	6	7	Output 8
Stage	conv + max	conv + max	conv	conv	conv + max	full	full	full
# channels	96	256	512	1024	1024	3072	4096	1000
Filter size	11x11	5x5	3x3	3x3	3x3	-	-	-
Conv. stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling size	2x2	2x2	-	-	2x2	-	-	-
Pooling stride	2x2	2x2	-	-	2x2	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1

Figura 5.15

Struttura della rete di OverFeat.

Fonte: https://arxiv.org/pdf/1312.6229.pdf

Durante l'allenamento, la rete riduce le immagini del training set in modo da ricondurle alla dimensione di 231x231px. In fase di test, invece, è possibile analizzare immagini più grandi tramite un approccio a finestra scorrevole, avendo in uscita molteplici classificazioni. Questo metodo è molto lento e soprattutto inefficiente perché porta a calcolare più volte la parti in comune tra due finestre che si sovrappongono. Perciò, con l'obbiettivo di migliorare le performance, una volta allenata la rete, ogni fully connected layer viene sostituito con un convolutional layer 1x1, senza modificare il valore dei pesi. In uscita, si ha quindi una matrice con più classificazioni, ognuna riferita ad una parte specifica dell'immagine (figura 5.16) (Sermanet et al., 2014).

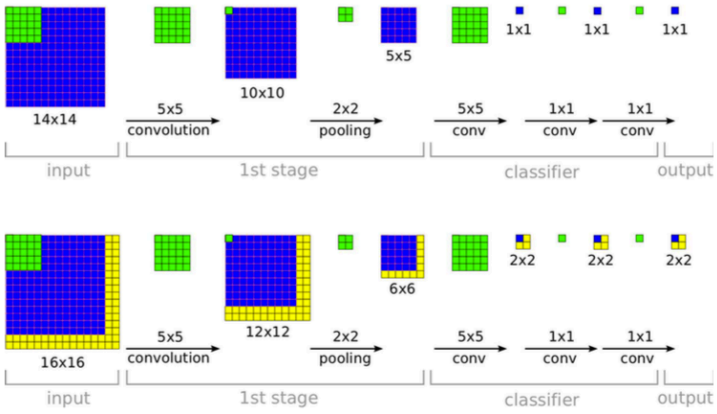


Figura 5.16

Utilizzando una rete con solo convolutional layer, all'aumentare delle dimensioni in input (parte gialla), l'output classifica i nuovi dati ampliando la sua dimensione, come in un approccio a finestra scorrevole. In verde sono raffigurati i filtri di convoluzione applicati.

Fonte: https://arxiv.org/pdf/1312.6229.pdf

Una volta trovato il valore desiderato di accuratezza nella classificazione, è possibile riutilizzare parte della rete come estrattrice di features al fine di localizzare l'oggetto. Per generare la predizione della bounding-box l'idea è quella di rendere invariabili i pesi dei primi 5 livelli e sostituire gli ultimi 3 layers con dei nuovi fully connected. In particolare i nuovi livelli saranno 3, rispettivamente dotati di 4096, 1024 e 4 neuroni. Ognuno dei 4 neuroni dell'ultimo livello indicherà i quattro angoli della regione in cui si troverà l'oggetto nell'immagine. In questo caso l'ottimizzazione sarà basata su una regressione, per cui la funzione da minimizzare è la distanza euclidea tra la predizione e quella reale. L'allenamento della localizzazione viene svolto solo con le immagini in cui l'oggetto è visibile almeno al 50%, dopodiché ogni immagine viene mostrata più volte a scale differenti al fine di aumentare la robustezza dell'apprendimento. Una volta terminato, anche in questo caso i fully connected layers vengono sostituiti da convoluzioni 1x1 (Sermanet et al., 2014).

Per svolgere i processi di classificazione e localizzazione in maniera simultanea, i layer di classificazione vengono aggiunti di nuovo al modello (figura 5.17)

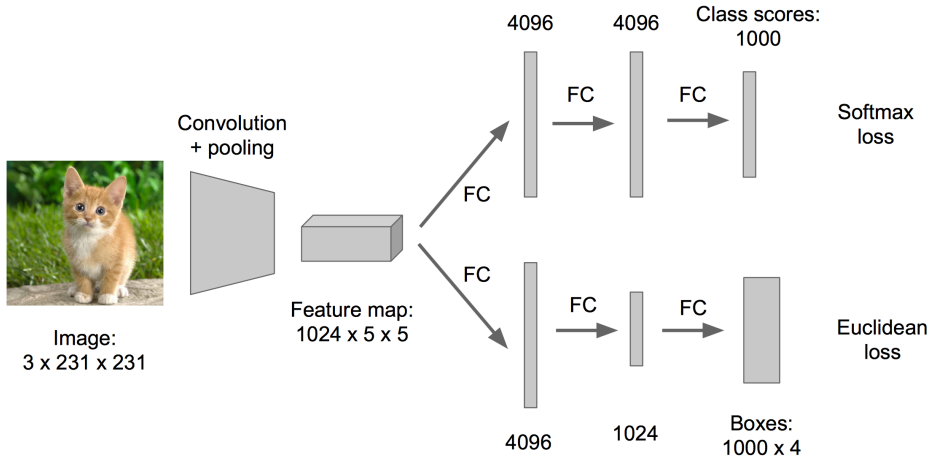


Figura 5.17

Struttura finale della rete utilizzata dal framework OverFeat.

Fonte: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Come detto, in fase di test, saranno predette più bounding-box B relative a più classificazioni C. Per scegliere la migliore viene adottata la seguente tecnica greedy:

a) Si assegna a C_s l'insieme delle migliori k predizioni per ogni scala $s \in 1, \dots, 6$.
b) Si assegna a B_s l'insieme delle bounding box predette per ogni classificazione C_s.
c) Si inizializza $B \leftarrow \cup B_s$
d) Viene ripetuta la seguente procedura fino alla condizione di stop:
    i. $(b_1^*, b_2^*) = argmin_{b_1 \neq b_2 \in B}(match\_score(b_1, b_2))$
    ii. **if** $match\_score(b_1^*, b_2^*) > t$, **stop**
    iii. **else** $B \leftarrow B \setminus \{b_1^*, b_2^*\} \cup box\_merge(b_1^*, b_2^*)$

dove $match\_score$ è la somma delle distanze dei centri delle due bounding-box b_1 e b_2, aggiunta all'intersezione dell'area; mentre box_merge calcola la media delle coordinate per ogni angolo. Questa tecnica permette di scegliere la migliore predizione, unendo inoltre le bounding-box migliori che fanno riferimento alla stessa classe (Sermanet et al., 2014).

5.4.2 R-CNN

Allo scopo di localizzare più oggetti in una scena, OverFeat produrrà quattro valori per ogni entità. Questa caratteristica obbliga (in allenamento) ad osservare gli oggetti in più scale e in più posizioni, aumentando la difficoltà di apprendimento.

Per ridurre il numero di proposte della stessa immagine, è possibile estrarre da essa solo le regioni più interessanti (RoI, Region-of-Interest) tramite una "Selective Search". Questo approccio è chiamato R-CNN, e come OverFeat modifica una rete neurale convoluzionale standard (paragrafo 5.3) sostituendo i layer finali con un meccanismo che processa sia le RoI che l'immagine di partenza. Nel tempo si sono susseguite più tipologie di questo framework, le più rilvanti sono: Fast R-CNN e Faster R-CNN.

La Fast R-CNN analizza l'input utilizzando i livelli di convoluzione e pooling di una rete neurale a scelta, per poi proiettare le RoI direttamente nelle ultime feature maps. Solo le parti dentro a queste regioni vengono poi processate da un "RoI pooling layer", che presenta le feature estratte a dei fully connected layer che produrranno in output una classificazione e una bounding-box per ogni regione proposta (figura 5.18.a). In confronto ad OverFeat, l'allenamento è svolto una sola volta, sviluppando una funzione di loss unica per entrambi gli output (Fei-Fei et al., 2016; Parthasarathy, 2017).

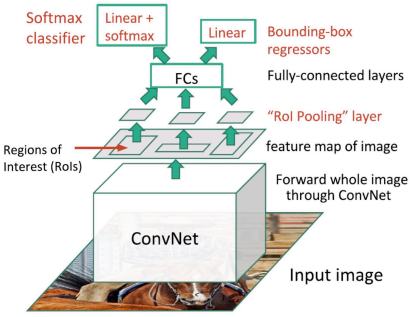


Figura 5.18.a

Fast R-CNN.

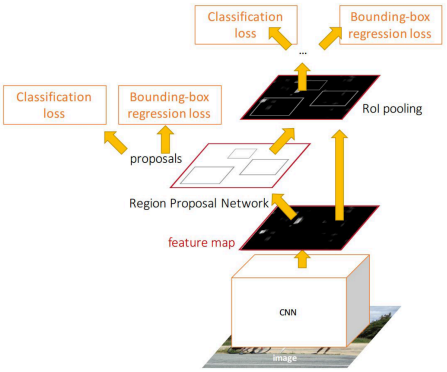


Figura 5.18.b

Faster R-CNN.

Fonte: https://zhuanlan.zhihu.com/p/29953111; https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html

La Faster R-CNN è un'evoluzione della Fast R-CNN nella quale la "Selective Search" viene sostituita da una rete più efficiente chiamata "Region Proposal Network" (RPN) e inserita direttamente dentro la Fast R-CNN. Lo scopo di questa seconda rete è individuare, un insieme di oggetti rettangolari ognuno con uno score preciso, il quale indica la probabilità che l'area selezionata sia un oggetto o meno. Queste informazioni vengono unite a quelle precedenti per calcolare in uscita classificazione e regressione. Come primo passo viene allenata solo la RPN, dopo di che viene immessa nel processo di training della Fast R-CNN (figura 5.18.b) (Fei-Fei et al., 2016; Parthasarathy, 2017).

5.4.3 YOLO

YOLO, abbreviazione di "You Only Look One", è un altro framework molto famoso utilizzato per la localizzazione di molteplici oggetti in un'immagine. In confronto ad OverFeat e Faster R-CNN, utilizza una rete neurale ben definita e non personalizzabile (figura 5.19). In questo modello, il layer di uscita suddivide l'immagine iniziale in una griglia usata per predire gli oggetti e la loro posizione. Tale approccio è 100 volte più veloce della Fast R-CNN, infatti, è

utilizzato per la localizzazione di oggetti in tempo reale sui video (ovviamente, con l'ausilio di GPU) (Redmon et al., 2016).

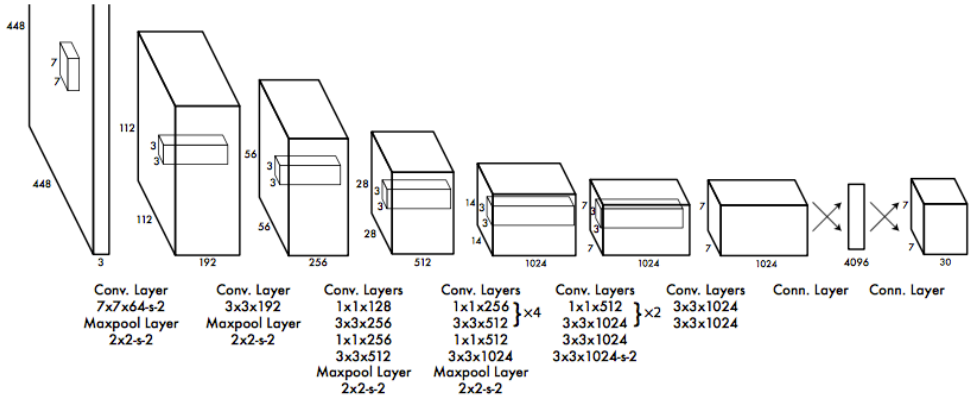


Figura 5.19

Rete neurale del framework YOLO.

Fonte: https://arxiv.org/pdf/1506.02640.pdf

La caratteristica principale che contraddistingue questo sistema è quella di unificare classificazione e regressione in un unico output, utilizzando solo l'intera immagine, senza l'aggiunta di ulteriori parti o modifiche. Questa rete produce in un'uscita un tensore con dimensioni $[S, S, (5 * B + C)]$ dove:

- S è la quantità di righe e colonne in cui si vuole suddividere l'immagine iniziale. Il totale di celle create sarà dunque $S \times S$;
- B è il numero di bounding-box predette per ogni cella. Esso è composto da 4 coordinate (che spaziano su tutta l'immagine ma hanno al centro all'interno della cella) più la confidenza, che indica la probabilità che in quella regione ci sia un'oggetto;
- C il numero di classi da predire. Ogni valore indica la probabilità che in quella cella ci sia una classe di un determinato tipo.

Ad esempio, in figura 5.19, si hanno i seguenti valori: $S = 7$, $B = 2$, e $C = 20$. Per attingere alla predizione finale, data una cella, la sua confidenza viene moltiplicata per ogni valore di C, in questo modo, per ogni cella si hanno B

bounding-box relative a C classi. Queste informazioni vengono utilizzate dall'algoritmo di "Non-maximum suppression" per eliminare le regioni ridondanti. Successivamente, vengono selezionate solo le classi predette sopra ad un determinato valore di soglia (figura 5.20) (Redmon et al., 2016).

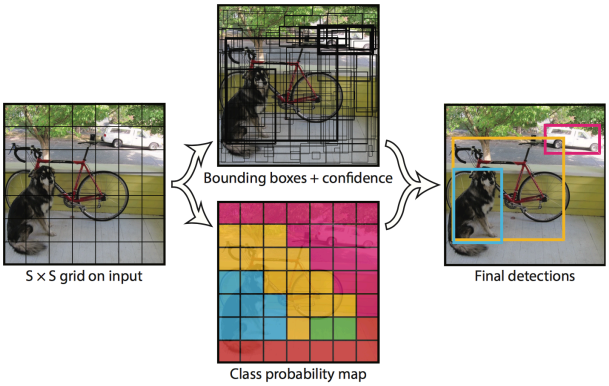


Figura 5.20

Predizione in uscita utilizzando il framework YOLO.

Fonte: https://arxiv.org/pdf/1506.02640.pdf

Anche se molto efficiente, YOLO riscontra problemi a riconoscere oggetti molti piccoli rispetto alla grandezza di una cella della griglia, ad esempio, piccoli animali lontani dalla telecamera (Redmon et al., 2016).

6 Matrix Auto-learn

In questo capitolo sarà descritta l'implementazione dell'algoritmo di auto-learn, costruito tramite tecniche di intelligenza artificiale. Partendo dall'introduzione, nella quale è possibile osservare la strategia generale, si passerà ad esporre le varie fasi di creazione dell'algoritmo, per poi concludere con i risultati ottenuti.

6.1 Strategia generale

Osservando il problema da un punto di vista più astratto si può riassumere che lo scopo di questa tesi è quello di creare una procedura che identifichi il valore di alcuni parametri in modo tale da soddisfare una determinata condizione dipendente da un algoritmo (quello della VL). Ciò significa che, se quest'ultimo cambia, allora le condizioni di ottimalità potrebbero cambiare. Le tecniche citate nei capitoli precedenti possono essere utilizzate per creare una possibile soluzione unica, ad esempio, tramite algoritmi di ottimizzazione genetici è possibile limitare la dipendenza al costo di mantenere un'elevata complessità di calcolo che potrebbe portare a tempi di risoluzione molto lunghi. Un'altra idea è optare per le reti neurali profonde, costruendo un modello (formato anche da un sistema complesso di reti) che, data un'immagine e i parametri associati di acquisizione, predica tramite una regressione (o classificazione se vengono utilizzati parametri discretizzati) la ricetta migliore. In questo caso però non solo il processo è fortemente dipendente dalla VL ma sussiste un'elevata complessità nella costruzione del data set su cui le reti dovranno apprendere le informazioni. Inoltre se l'algoritmo della libreria viene modificato radicalmente, bisognerebbe ripetere tutto il processo di training e formazione degli esempi.

Partendo da questi presupposti teorici, è stato deciso di scomporre il problema in più parti cercando di diminuire o eliminare i problemi discussi in precedenza. Nello specifico la strategia generale è composta da tre algoritmi che si susseguono in cascata, ognuno con un obbiettivo ben preciso:

1. Localizzazione del codice. Determina il parametro ROI diminuendo l'area di ricerca aumentando la velocità di acquisizione per i prossimi passi;
2. Messa a fuoco del codice. Viene identificato il parametro Reading Distance;
3. Ricerca dei parametri rimasti legati alla luminosità della scena.

In aggiunta, allo scopo di diminuire la complessità computazionale, alcuni parametri vengono discretizzati.

6.2 Studio dei parametri

6.2.1 Discretizzazione

Il Matrix 300N permette una configurazione molto ampia e precisa, infatti, come è possibile osservare dal paragrafo 2.4, ogni parametro ha un intervallo di valori molto grande che porta ad aumentare la complessità dell'algoritmo di auto-learn.

Sorge spontanea l'intenzione di ridurre (laddove è possibile) il dominio di ogni parametro in modo tale da esaminare solo i valori che determinano un cambiamento significativo nella decodifica del codice. La discretizzazione effettuata è disponibile nella tabella 6.1, i parametri assenti non sono stati modificati. L'intero processo è stato validato e supervisionato da un tecnico esperto.

Parametro	Incremento	Intervallo dominio	Osservazioni
Gain	2	$[4, 48]$	-
Gain Multiplexer	1	$[1, 1]$	Di norma non vengono utilizzati valori diversi da 1.
Exposure Time (Internal Lighting = 2)	50	$[50, 500]$	-

Exposure Time (Internal Lighting = 3)	300	$[300, 3300]$	-
Internal Lighting	1	[2,3]	Le combinazioni con tutti i led spenti o accesi determinavano una forte dipendenza dall'illuminazione circostante.
Reading Distance	10	$[30, 330]$	Di solito, i lettori non vengono installati ad una distanza maggiore di 330 millimetri dal piano su cui il prodotto viene posto.
Led Group	1	$[peripheral]$	Vengono utilizzati solo i led periferici in quanto offrono una migliore distribuzione della luce a confronto di quelli centrali.

Tabella 6.1

Discretizzazione dei parametri del Matrix 300N. Le caselle che non contengono osservazioni indicano che la discretizzazione è data solo dall'aumento del valore di incremento, il quale permette di selezionare i valori che effettuano un cambiamento drastico nella rilevazione fotografica.

6.2.2 Luminance

Durante lo studio di questi parametri è stata notata empiricamente una certa somiglianza tra Gain e Exposure Time nel modo in cui trasformano l'immagine a livello di luminosità media. Ad esempio, si osservino attentamente le figure 6.1.a e 6.1.b. Il valore medio dei relativi istogrammi è di 95 e 97.5 ma i

parametri del Matrix 300N con cui sono state effettuate le foto sono completamente diversi e in entrambi i casi il codice è stato letto correttamente.

Figura 6.1.a

Immagine eseguita con un Matrix 300N con Internal Lighting = 3, Exposure Time = 2700 e Gain = 4.

Figura 6.1.b

Immagine eseguita con un Matrix 300N con Internal Lighting = 2, Exposure Time = 300 e Gain = 14.

Da questa osservazione (riscontrata in varie occasioni) deriva l'ipotesi che esistono delle combinazioni di parametri potenzialmente equiparabili, o quantomeno simili a livello di lettura del codice. Nel tentativo di identificarle è stato creato un nuovo dominio su cui mappare i valori di Gain, Exposure Time e Internal Lighting: la Luminance. Considerando tutti gli altri valori costanti, la Luminance determina una classificazione di tutte le combinazioni dei tre parametri citati, partendo da quella che fornisce l'acquisizione fotografica più scura a quella più chiara, in termini di valor medio dell'istogramma. Il nome della variabile deriva dalla componente Y riferita allo spazio di colore YUV la quale indica la luminanza dell'immagine.

La Luminance è stata calcolata in modo empirico, attraverso ripetute acquisizioni di un pattern noto (figura 6.2), considerando una Reading Distance media pari a 165 millimetri, ponendo il Matrix 300N centralmente (figura 6.3.a) e utilizzando solo i led periferici. Questo tipo di pattern è stato scelto perché:

- contiene tutta la gamma dei valori di grigi. Pattern chiari portano ad una classificazione poco accurata in corrispondenza di combinazioni che saturano l'immagine (ad esempio: Gain = 40, Exposure Time = 500 e Internal Lighting = 2), viceversa per combinazioni che portano ad immagini scure (ad esempio: Gain = 4, Exposure Time = 300 e Internal Lighting = 3);

- il fascio di led di un settore viene proiettato sempre lungo l'intera gamma della scala di grigi. Si ipotizza che questa caratteristica porti omogeneità al risultato finale, ed è stata preferita ad altri pattern. Si pensi ad esempio ad un'immagine che contenga una gamma di grigi distribuita da sinistra a destra. Con questa scelta alcune combinazioni di settori porteranno ad illuminare poco la parte chiara e molto la parte scura, invece altre illumineranno tutta la gamma (figura 6.3.b). Questa diversità potrebbe portare ad un risultato non accurato.

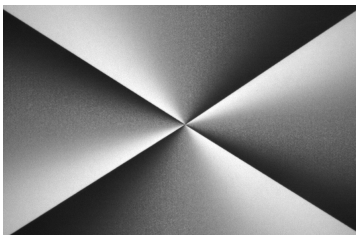


Figura 6.2

Una rilevazione del pattern utilizzato per la ricavare il dominio del parametro Luminance.

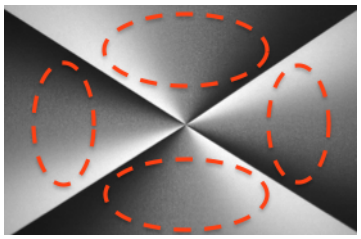


Figura 6.3.a

Corrispondenza foto con l'accensione led del Matrix 300N (aree in rosso) nel pattern utilizzato.

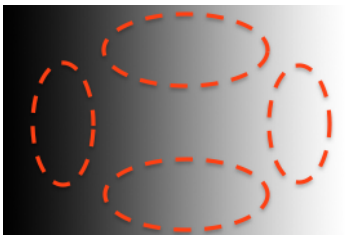


Figura 6.3.b

Corrispondenza accensione led del Matrix 300N (aree in rosso) nel pattern ipoteticamente non corretto.

Una volta scelte le specifiche iniziali, viene definito l'insieme C, come

$$C = G \times ET \times IL \mid C \subset M \qquad \#C = 483$$

dove G, ET e IL sono rispettivamente gli insiemi Gain, Exposure Time e Internal Lighting discretizzati. M è l'insieme delle possibili combinazioni valide nel Matrix 300N utilizzando i tre parametri citati. Sia S l'insieme dei Sectors, per ogni suo elemento s, viene effettuata una rilevazione fotografica del pattern noto per ogni elemento c dell'insieme C come in algoritmo 6.1.

Algoritmo 6.1: per ogni immagine rilevata (utilizzando le combinazioni ottenute tramite S e C) calcola la media dell'istogramma.

Input: S, C

Output: R

1. $R \leftarrow \emptyset$
2. **For each** s **in** S **do**
3. $\quad List \leftarrow \emptyset$
4. $\quad$ **For each** c **in** C **do**
5. $\quad\quad image \leftarrow get\_image\_from\_matrix300N(s, c)$
6. $\quad\quad h \leftarrow histogram(image)$
7. $\quad\quad m \leftarrow mean(h)$
8. $\quad\quad$ **Add** (c, m) **into** $List$
9. $\quad$ **Order** $List$ **by** m
10. $\quad$ **Add** $List$ **into** R

Dalle righe 5, 6, 7 si nota che da ogni immagine generata viene calcolata la media dell'istogramma h come

$$m = \frac{\sum_i p_i}{N}$$

dove N è il numero di pixel dell'immagine e p_i è il valore (in scala di grigi a 265 valori) del pixel i-esimo. In riga 8 tutte le combinazioni di C sono ordinate seguendo i valori di m. In uscita si ha una lista R contenente $\#S$ liste ordinate C_s.

È stato notato che, prendendo una combinazione qualsiasi c in C_s, quest'ultima si trovava in posizioni simili (se non uguali) in tutti gli ordinamenti presenti in

R. Infatti, analizzando una coppia qualsiasi di liste appartenenti a R tramite funzione di correlazione si è osservato che il risultato finale oscillava tra il 70% e il 90% di similitudine. Per questo motivo si può concludere che è possibile creare un solo ed unico ordinamento di questi valori in modo tale da essere indipendente dalla combinazione s-esima (algoritmo 6.2).

Algoritmo 6.2: partendo dai valori contenuti in R, calcola la Luminance

Input: R

Output: $Luminance$

1. $Luminance \leftarrow \emptyset$
2. **For each** c **in** C **do**
3. $\quad position \leftarrow \emptyset$
4. $\quad$ **For each** s **in** S **do**
5. $\quad\quad List \leftarrow R.get\_element(s)$
6. $\quad\quad i \leftarrow List.get\_index(c)$
7. $\quad\quad position \leftarrow position + i$
8. $\quad position \leftarrow position \,/\, lenght(C)$
9. $\quad$ **Add** $(c, position)$ **into** $Luminance$
10. **Order** $Luminance$ **by** $position$
11. Overwrite all $position$ inside $Luminance$ with those index

In altre parole, la Luminance può essere vista come un valore univoco da associare ad ogni $c \in C$, che rappresenta la potenzialità della combinazione c di illuminare la scena considerando costanti gli altri parametri del lettore.

Infine, è stato osservato il comportamento di questo nuovo dominio sui vari campioni delle tipologie citate in 2.1, per verificare la risposta della Virtual Library. Come è possibile osservare in figura 6.4, a parità degli altri parametri, i valori in cui la libreria riesce a identificare il codice si addensano in un intervallo ben preciso. Tale condizione è stata sempre verificata in tutte le prove svolte. Un'altra osservazione molto importante da sottolineare è che gli estremi di tale intervallo non sono netti ma presentano delle oscillazioni. Esaminando attentamente si nota che questo è dovuto a dei valori di

Luminance in cui il Gain era molto alto, mentre quello di Exposure Time era basso. Come detto in 2.4 nella descrizione dei parametri del Matrix 300N, un Gain alto porta ad aumentare il rumore nell'immagine che, trovandosi già in una condizione di luminosità limite, aumenta di conseguenza la difficoltà di decodifica. Non si esclude che una causa aggiuntiva possa essere l'ordinamento finale svolto per ricavare i valori di Luminance.

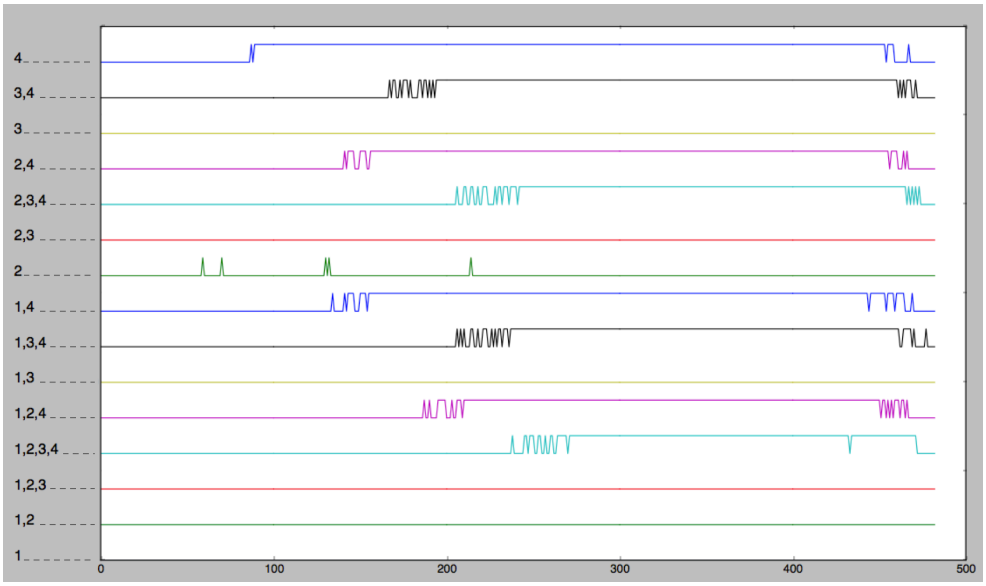


Figura 6.4

Andamento della lettura della VL per i vari valori di Luminance (ascisse). Quando il segnale è alto, allora la VL codifica il Data Matrix, quando è basso, la libreria non identifica il codice. Nell'asse delle ordinate è possibile osservare il test replicato per ogni valore di Sectors utilizzando i led perimetrali. La Reading Distance è stata calcolata manualmente ed è sempre costante, come tutti gli altri parametri.

6.2.3 Raggruppamento

Concludendo, di seguito è riportata la Tabella 6.2 che indica come i vari parametri vengono raggruppati in modo tale da renderli più utilizzabili nelle applicazioni a seguire.

Vecchie variabili	Nuove variabili	Intervallo	Incremento
Exposure Time, Internal Lighting, Gain, Gain Multiplexer	Luminance	$[0, 482]$	1
Led Group, Sectors	Led Position	$[0, 14]$	1
Reading Distance	Reading Distance	$[30, 330]$	10

Tabella 6.2

Sintesi dei nuovi parametri derivati dal raggruppamento e discretizzazione di quelli base del Matrix 300N.

6.3 Localizzazione del codice

6.3.1 Creazione del data set

Il data set è considerato una parte fondamentale per la riuscita dell'esperimento, infatti, se fosse inconsistente (contiene etichette che non rispecchiano il contenuto dei dati associati) i risultati sarebbero inevitabilmente compromessi perché il modello apprenderebbe informazioni errate. Con lo scopo di immagazzinare i dati in maniera efficiente e accurata è stata creata un'applicazione desktop chiamata DPM DB, la quale permette con una certa facilità di salvare immagini e dati numerici provenienti dal lettore. DPM DB (figura 6.5) è disponibile per i sistemi Windows, quindi utilizzabile dalla maggior parte dei dispositivi della rete aziendale Datalogic, e viene implementata attraverso le seguenti tecnologie:

- C# (preferito al C++ per la sua facilità d'uso e scalabilità nel contesto);
- LINQ (per l'interfacciamento automatico con il database);
- SQLite (database dotato di un'ottima semplicità d'uso e portabilità essendo un sistema serverless);
- SQLite Studio (per la visualizzazione e gestione manuale del database).

L'architettura è stata sviluppata tramite Visual Studio 2010, comunica automaticamente con il Matrix 300N, ed è predisposta (apportando lievi modifiche) all'aggiunta di altri dispositivi. Questo strumento, permette di salvare:

- acquisizioni di ricette singole;
- le informazioni principali del campione (*Sample Information*) e le sue relative informazioni di posizionamento nel momento dell'acquisizione (*Position Informations*);
- una serie di combinazioni personalizzabili di Reading Distance, Luminance e Led Position (*Test and set*, *Acquire images and data*);
- tutte le combinazioni di Luminance per osservare il comportamento di lettura della VL (*Acqure all combo of luminance without images*).

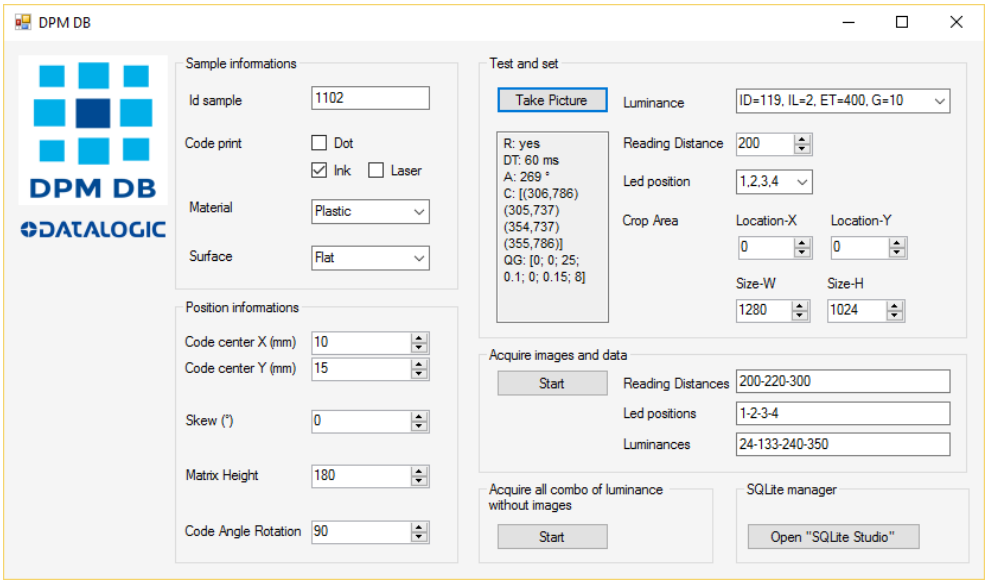


Figura 6.5

DPM DB: interfaccia utente.

Sono stati scelti accuratamente 36 campioni di vari materiali con varie modalità di stampa del codice, in modo da coprire la maggior parte delle casistiche che possono verificarsi nelle applicazioni reali (la lista completa viene

proposta nel paragrafo 2.1). Per questa fase, e in quella di test dell'algoritmo di auto-learn, è stato utilizzato un supporto munito di un braccio (su cui viene applicato il lettore) che permette di regolare in altezza la distanza della camera dal piano di acquisizione.

Prima di iniziare, è importante capire cosa deve imparare la rete neurale. In questo caso il modello si occuperà della ricerca di un codice in un'immagine, tale codice però potrebbe apparire sfocato, poco illuminato o parzialmente visibile, in quanto nella prima fase non si hanno vincoli sui valori dei parametri e non si è a conoscenza del posizionamento dell'oggetto nella scena. Questo significa che le immagini ottenute dovranno rispecchiare anche queste caratteristiche. È importante precisare che, per semplificare l'applicazione, si è deciso di porre il vincolo che nell'immagine catturata dal lettore debba essere presente sempre un solo codice potenzialmente visibile nell'inquadratura.

La procedura di popolamento del database utilizzando il DPM DB è stata quindi la seguente:

1. Viene scelto il campione;
2. Il pezzo viene poggiato su una parte casuale nella pedana;
3. Vengono inseriti i dati relativi al campione e al suo posizionamento;
4. Si effettuano alcuni test utilizzando vari tipi di Luminance, Led Position e Reading Distance mirati a capire:
    a. Quali combinazioni permettono la visibilità del codice totale, parziale e nulla;
    b. Quali valori di Reading Distance determinano un'immagine sfocata o nitida;
    c. Quale combinazione permette la lettura del codice tramite la VL;
5. Una volta inseriti i valori trovati, viene avviato il sistema di rilevazione automatica;
6. La procedura viene ripetuta più volte per lo stesso campione, aumentando o diminuendo la distanza codice-sensore e spostando il pezzo in diverse posizioni.

Molte scelte sono state effettuate in modo arbitrario e sfruttando alcune indicazioni fornite dai tecnici, in modo tale da rispecchiare i casi d'uso reali. Ad esempio, si è preferito non spostare il codice in zone troppo lontane dal centro della scena e, quando è stato possibile, non inclinare il piano del codice rispetto a quello del lettore per non alterare i risultati di axial non-uniformity forniti dagli algoritmi che si occupano della qualità del codice. In più, è stato imposto che la grandezza del codice nell'immagine deve essere compresa tra i 170 px e i 40 px, questo solo per facilitare la creazione del data set e l'implementazione della rete neurale. In un'applicazione futura sarà possibile espandere questa condizione semplicemente inserendo nuovi esempi e modificando il modello.

Le immagini salavate hanno una grandezza di 1280x1024px, quindi è importante capire dov'è situata la bounding-box contenente il codice (la quale occupa meno del 15% dell'area totale). Tramite uno script in python realizzato appositamente, sono state osservate tutte le immagini in modo tale da aggiungere manualmente le informazioni mancanti o scartare alcune rilevazioni non idonee. Nei casi in cui la VL riusciva a leggere il codice, era possibile usufruire dei dati generati in automatico. Dopo questo processo, necessario per essere sicuri di creare un data set consistente, sono state prese in totale 632 immagini valide.

Per ampliare ulteriormente il data set, la Datalogic ha fornito ulteriori 200 immagini contenenti codici Data Matrix di varie dimensioni decodificabili dall'algoritmo della VL. Questa condizione però, non essendo perfettamente adatta allo scopo (poiché l'obbiettivo è individuare il codice in condizioni sfavorevoli), per ogni immagine è stata creata una copia modificata in modo da simulare una sfocatura tramite un filtro gaussiano. Il kernel di questo filtro veniva modificato arbitrariamente a seconda della foto, in modo da rendere comunque riconoscibile il codice ad occhio umano. Questa procedura è stata svolta modificando lo script python utilizzato precedentemente. Scartando alcune immagini, si è raggiunta la cifra di 390 campioni da aggiungere ai 632 rilevati tramite il DPM DB per un totale di 1022.

Con l'idea di fornire i dati in maniera veloce ed efficace alla rete neurale, sono state ricavate da ogni immagine le parti in cui non era presente il codice. Come è visibile in figura 6.6 ogni immagine è stata scomposta in 4 parti e da

ognuna di esse è stata ricavata una nuova immagine se il lato minore era superiore di 170 pixel.

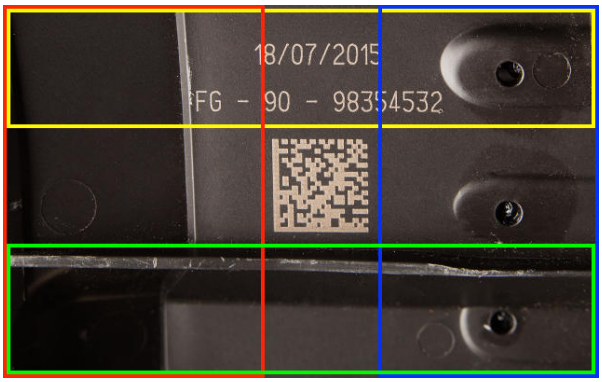


Figura 6.6

Scomposizione di un'immagine per ricavare le quattro parti (giallo, rosso, verde, blu) che non contengono il codice.

Il dataset finale è stato quindi composto e contrassegnato nel seguente modo:

- Classe "Code": 1022 immagini 1280x1024 px contenenti il codice con l'informazione relativa alla bounding box;
- Classe "Other": 2985 immagini di varie dimensioni che non contengono il codice.

Il 20% degli elementi di ogni classe è stato utilizzato come test-set (rispettivamente 204 e 597 campioni) mentre il restante 80% è stato utilizzato per il training-set. Lo split è stato effettuato in maniera casuale.

6.3.2 Modello di classificazione

Individuare un oggetto in un'immagine tramite tecniche di Deep Learning significa risolvere due sub-task: classificazione e localizzazione (Fei-Fei et al., 2016). Nel primo caso, il modello deve capire se esiste l'oggetto nella scena, nel secondo deve indicare la posizione nell'immagine tramite una bounding box. Nello stato dell'arte odierno, esistono 3 framework principali che implementano

queste caratteristiche: OverFeat, R-CNN e YOLO. La scelta finale è ricaduta su OverFeat per i seguenti motivi:

- Facile implementazione su TensorFlow;
- Approccio scalabile (è possibile cambiare rete mantenendo il metodo invariato);
- Nel momento dell'implementazione (settembre 2016) non erano fruibili modelli pre-trained di R-CNN e YOLO utilizzabili su TensorFlow.

Il framework originale di OverFeat utilizza la rete AlexNet, una delle prime CNN, la quale è molto grande in termini spazio su disco (circa 250 MB) ma soprattutto è poco accurata rispetto alle nuove reti scoperte negli anni avvenire. Al posto di AlexNet si è preferito utilizzare la GoogLeNet (Inception-v1), leggera (circa 40 MB), veloce e più accurata (Canziani et al, 2016). Tenendo conto che dovrà classificare solo due label ("Code" e "Other") la GoogLeNet dovrà essere modificata e adattata al framework OverFeat (figura 6.7).

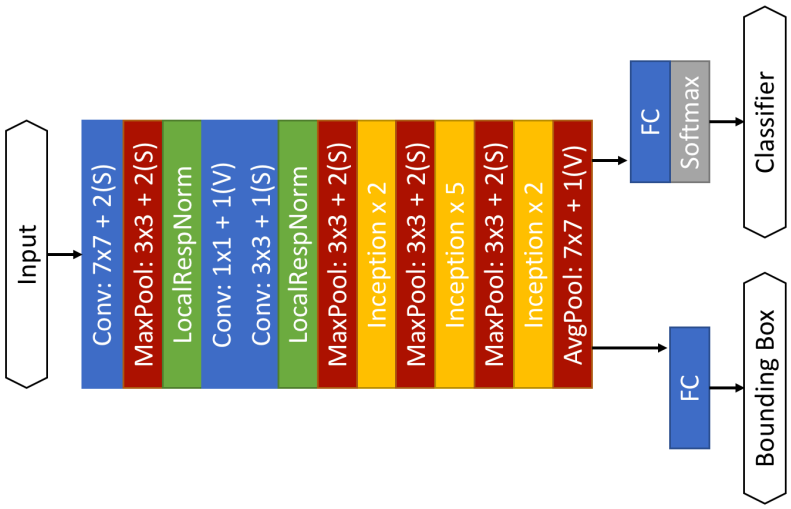


Figura 6.7

Modello utilizzato per la classificazione e localizzazione del codice in un'immagine fornita dal Matrix 300N.

6.3.3 Training

Le reti neurali profonde hanno bisogno di moltissimi esempi per apprendere e classificare in maniera adeguata; la quantità di campioni dipende da molti fattori come il numero di classi, il modello o la grandezza dei dati di input. La GoogLeNet è stata progettata per un dataset ricavato da ImageNet il quale contiene 1000 classi con più di 1.2 milioni di esempi solo per il training-set Sarà quindi opportuno adottare delle strategie di transfer learning e fine tuning per ottenere dei risultati accettabili dal training set ricavato in 6.3.1 (Szegedy et al., 2014).

Prendendo spunto dalle indicazioni riportare in (Sermanet et al., 2014), il processo di allenamento avviene secondo due fasi:

1) Presa la GoogLeNet pre-trained, si sostituisce l'ultimo livello con uno nuovo (fully-conncted) contenente tanti neuroni quante sono le etichette da classificare. Quest'ultimi, inizializzati in modo casuale, si occuperanno di classificare l'immagine in input (di grandezza 224x224 px). A questo punto la rete viene allenata per la classificazione fino a raggiungere l'accuratezza voluta.
2) Il secondo step, prevede di agganciare all'ultimo livello di convoluzione un nuovo layer fully-connected con quattro neuroni che individueranno la bounding box dell'oggetto da identificare. Le coordinate in uscita dal modello saranno quelle del punto centrale del codice, più altezza e larghezza, partendo da quest'ultimo (Fei-Fei et al., 2016). L'algoritmo di ottimizzazione ha come obiettivo minimizzare la distanza L2 tra la predizione e i punti reali, allenando solo l'ultimo layer. In questa fase la rete osserva solo immagini con codice, infatti, questo output viene interrogato solamente nel caso in cui la classificazione abbia trovato l'oggetto desiderato.

Durante tutti i vari allenamenti, è importante che i dati vengano leggermente modificati prima di essere inseriti nel modello. Seguendo le strategie comuni (Sermanet et al., 2014) ogni immagine viene casualmente

ruotata, traslata e scalata con lo scopo di evitare l'overfitting. Le immagini del data set sono molto grandi rispetto all'input della rete, quindi è opportuno sottolineare che le trasformazioni sono create in modo tale da formare un'immagine da 224x224 px in cui, nel caso che si prenda un'immagine da classificare come "Code", debba essere rispettata la grandezza del codice rispetto le specifiche. Mentre, nel caso di "Other" viene presa una parte casuale dell'immagine dopo essere stata opportunamente modificata.

Durante l'allenamento viene controllata periodicamente l'accuratezza del modello tramite il validation-set ottenuto dai campioni dell'ultimo batch di ogni epoca. Per velocizzare l'apprendimento sono state impiegate 2 GPU NVIDIA GeForce 780 in parallelo. Nella figura 6.8 è possibile osservare lo schema del training in parallelo utilizzato nel caso preso in esame.

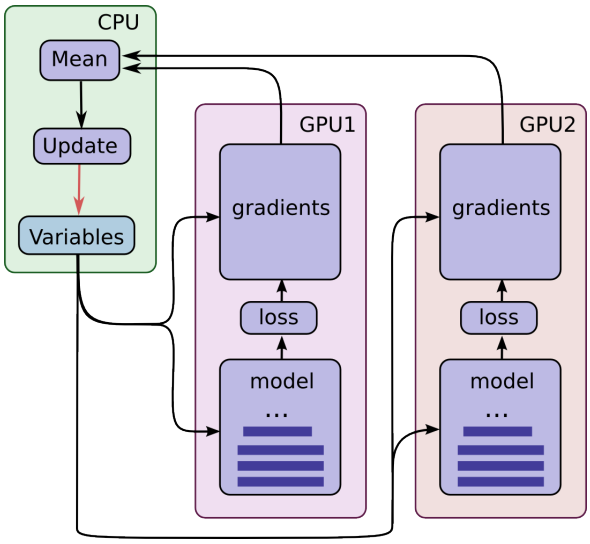


Figura 6.8

Configurazione di training con due GPU utilizzando il framework TensorFlow.

Fonte: https://www.tensorflow.org/tutorials/deep_cnn

6.3.4 Risultati

Il training di una rete neurale richiede di solito molto tempo perché bisogna provare molte combinazioni dei parametri globali, come learning rate, batch size, numero di epoche e il tipo di algoritmo (ad esempio Nesterov, RMSprop o

Adam). Nella tabella 6.3 è messo in evidenza il risultato migliore ottenuto con i relativi valori.

Batch size	Learning rate	Epoche	Algoritmo di ottimizzazione	Accuratezza classificazione ("Code" /"Other")	Errore regressione (MAE)
64	0.00001	300	*Adam*	91.95%/91.99%	20 px

Tabella 6.3

Parametri e risultati del modello migliore trovato.

È possibile notare che il layer di regressione riporta un errore molto elevato. In pratica, ogni punto della bounding-box predetta ha un errore medio di 20px, dunque troppo alto per essere utilizzato. Al contrario, è possibile affermare che il layer di classificazione ha riportato un esito molto positivo e pienamente accettabile.

Seguendo l'approccio descritto in (Sermanet et al., 2014) è possibile comunque sfruttare la classificazione come localizzatore, sostituendo in fase di test tutti i neuroni dei layer fully-connected con delle convoluzioni 1x1 senza cambiare i valori dei pesi trovati dopo l'allenamento. Come discusso in 2.4.1, in questo modo si ha lo stesso risultato di un approccio a finestra scorrevole, con il vantaggio di non calcolare per ogni finestra l'intero grafo della rete, inoltre, è possibile inserire in ingresso un'immagine di qualsivoglia dimensione. Nel caso in esame, la grandezza è di 1280x1024px, per cui l'uscita sarà una matrice di 33x25 elementi. Preso l'output i_o, j_o, esso corrisponderà alla predizione riguardante l'area in input pari a

$$(31(i_o - 1), 31(j_o - 1), 31i_o + 257, 31j_o + 257) \qquad \begin{matrix} \forall\, i_o \in [1, 33] \\ \forall\, j_o \in [1, 25] \end{matrix}$$

definita come (x, y, x + lunghezza, y + altezza), dove lunghezza e altezza sono pari alla dimensione massima nella quale si ha ancora una sola predizione in uscita. Per contro, questo metodo impone che ogni output si riferisca ad una

finestra di input fissa, perciò codici per i quali la bounding-box supera i $257x257$px non potranno essere localizzati.

6.4 Autofocus

L'algoritmo di autofocus ha il compito di trovare il parametro di Reading Distance. L'idea principale per risolvere tale problema è quella di provare varie combinazioni di questo parametro, acquisire un'immagine per ognuna di esse, valutare il livello di nitidezza e infine, scegliere il valore che ha portato al risultato migliore. È molto importante considerare che questa procedura avviene dopo la localizzazione del codice, infatti questo assicura (nei limiti della bontà di predizione della rete neurale) che il codice sia presente nell'immagine, occupando buona parte di essa, e soprattutto sia illuminato in modo tale da renderlo visibile. Queste due condizioni semplificano e aumentano le prestazioni dell'autofocus, diminuendo la possibilità di mettere in evidenza eventuali altri oggetti presenti.

Per ottenere questo valore manualmente è possibile calcolare la distanza tra il codice e il lettore con un qualsiasi nastro metrico, in questo modo si è sempre sicuri di ottenere la migliore nitidezza possibile. In generale, non è necessario recuperare il valore preciso, l'importante è individuarne uno che si aggira attorno alla misura ottima, in modo che il codice risulti ancora nitido. Per recuperare tale intervallo, invece che esplorare tutti i valori del dominio, si è pensato di velocizzare la ricerca distribuendo dicotomicamente le distanze in un albero appositamente costruito ed utilizzare un semplice algoritmo Greedy Best-First per prendere le decisioni di esplorazione (figura 6.9). Tale algoritmo è stato studiato anche perché si presuppone che la nitidezza aumenti fino ad un punto massimo per poi decrescere, in questo modo l'ottimo locale trovato sarà prossimo a quello globale e quindi accettabile per l'obiettivo prefissato.

Come funzione euristica (f) è stata scelta la varianza di tutti gli elementi m_i della matrice M ricavata tramite convoluzione tra l'immagine in input e il filtro Laplaciano riportato in seguito:

$$M = I * \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad f = \frac{\sum_{i=0}^{n}(m_i - \mu)}{n-1}$$

dove μ è la media di tutti valori di M e n è pari al prodotto righe per colonne di M. Più f è alto più l'immagine risulterà nitida (Pech-Pacheco et. al, 2000; Mir et al., 2014). Il kernel del filtro è stato individuato valutando la sua efficacia su un campionamento del data set preso in 6.3.1, nel quale data un'immagine nitida si ha anche la corrispettiva sfocata. In questo modo è stato possibile creare un test set per osservare se il kernel è in grado di discriminare le caratteristiche cercate.

Non avendo a disposizione un criterio di stop (perché l'eventuale valore di soglia dell'euristica è dipendente dall'immagine di input), l'algoritmo esplorerebbe comunque tutte le soluzioni. Per evitare questo, è stato imposto che, una volta scelto un ramo dell'albero, non è più possibile tornare indietro, quindi ad ogni iterazione l'algoritmo scenderà sempre di un livello e terminerà una volta raggiunte le foglie dell'albero. Il valore finale sarà quello migliore trovato in tutto il cammino. Questo espediente implica che la prima scelta è fondamentale e nello scopo di non intraprendere fin dall'inizio la strada sbagliata, sul primo livello vengono svolte tre rilevazioni per ogni soluzione da esplorare, preferendo poi quella che ha la varianza media più alta (figura 6.9).

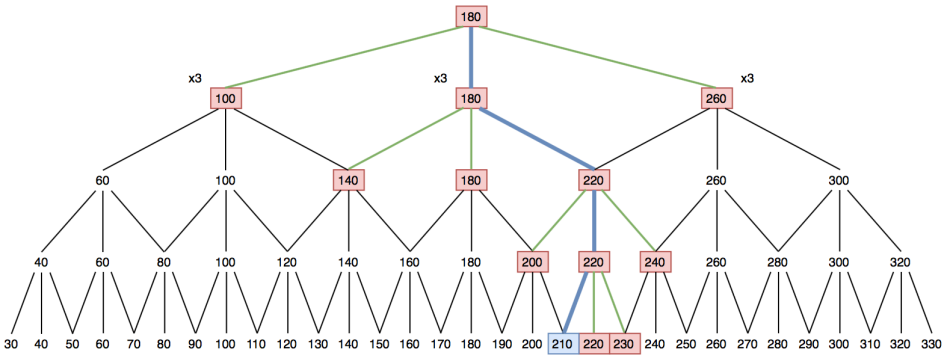


Figura 6.9

Esempio di esplorazione dell'albero di ricerca riguardante il dominio Reading Distance. In blu il percorso ottimo e la soluzione finale, mentre in rosso e in verde sono rispettivamente i nodi e i rami esplorati.

Inoltre, per evitare di esplorare gli stessi nodi più volte, è stata aggiunta una lista tabu.

6.5 Ricerca dei parametri di Luminance e Led Position

Lo scopo di questo algoritmo è trovare i valori di Luminance e Led Position in modo tale che la Virtual Library decodifichi il codice rispettando delle condizioni minime sufficienti. In questa fase si presuppone che il codice sia presente nell'immagine e che sia stato individuato un valore accettabile di Reading Distance.

Nel paragrafo 6.2.2 è stato verificato empiricamente che le condizioni in cui è possibile decodificare il codice al variare della Luminance si addensano in un intervallo. Quindi, seguendo l'approccio descritto in 6.4, è possibile applicare anche in questo caso un metodo Greedy Best-First ad un albero di ricerca pre-costruito in quanto, comunemente, esistono più soluzioni che possono soddisfare le esigenze richieste. Quindi con un'euristica ben definita è possibile condurre l'algoritmo inizialmente alla ricerca di una soluzione che permetta almeno la decodifica e successivamente, se non vengono ancora rispecchiati i requisiti minimi, continui a cercare dentro all'intervallo.

Lasciando da parte la ricerca del parametro di Led Position, come in 6.4, è possibile costruire l'albero di ricerca seguendo una strategia dicotomica. Tale strategia però non tiene conto che, preso un intorno da esaminare, in alcuni casi è preferibile analizzare prima i valori con Exposure Time più alti e Gain bassi piuttosto che il contrario, in quanto, aumentare il Gain porta inevitabilmente ad accrescere il rumore nell'immagine (paragrafo 2.4). Di conseguenza, l'approccio generico prevede che una volta individuato il punto da esaminare, prima di calcolare il valore euristico, si effettui una local search su n valori più vicini, esaminando il nodo con rapporto Exposure Time-Gain più alto. Dal punto di vista pratico si è deciso di creare l'albero di ricerca in modalità bottom-up. Nel livello più basso l'intero dominio di Luminance viene suddiviso in 81 sub-intervalli da 6 valori ciascuno (tranne l'ultimo che ne contiene 5 e il primo che ne contiene 4). I livelli successivi vengono creati unendo i valori in gruppi da 3 elementi fino ad arrivare alla radice. Infine, con

lo scopo di esaminare un intorno più ampio in prossimità dei nodi foglia, si è deciso di aggiungere al nodo padre del penultimo livello una foglia del padre antecedente e 2 di quello successivo. La scelta di questi parametri è stata svolta dopo varie sperimentazioni nelle quali si è cercato di ottimizzare l'equa distribuzione dei valori nei vari livelli e ovviamente spingere i valori con alto rapporto Exposure Time-Gain ad occupare posizioni verso i livelli più alti (figura 6.10).

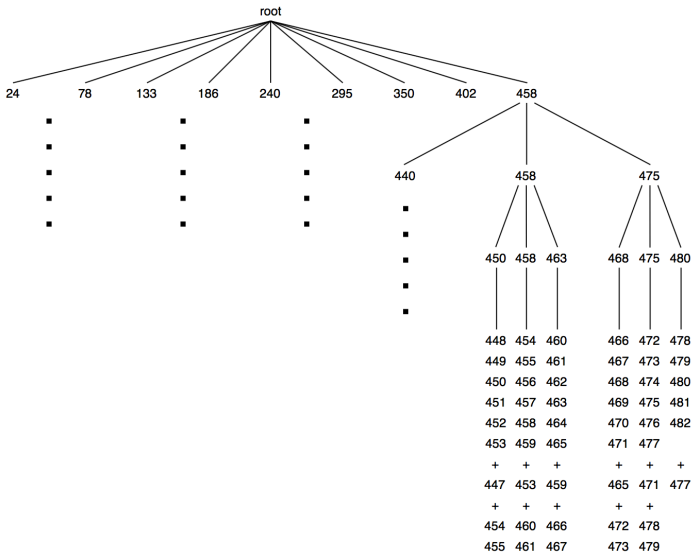


Figura 6.10

Una parte dell'albero di ricerca utilizzato per l'esplorazione del dominio Luminance.

Per aggiungere l'inserimento del parametro di Led Group nella ricerca, la metodologia studiata è stata estesa creando 15 copie dell'albero, in ognuna delle quali il valore di questo parametro è diverso. Un'ulteriore decisione è stata quella di iniziare la ricerca a partire dal secondo livello di ogni albero contenente 9 valori ritenuti, tramite le prove empiriche svolte, sufficienti per esplorare lo spazio di ricerca in modo distribuito. In pratica, l'algoritmo esamina inizialmente 9x15 combinazioni rispettivamente di Luminance e Led Group, per poi decidere tramite l'euristica in quali rami proseguire (riga 1 in algoritmo 6.3). Inoltre, come nella ricerca della Reading Distance, anche in

questo caso, per evitare di esplorare due volte lo stesso nodo, è stata inserita una lista tabu nell'algoritmo.

Algoritmo 6.3: ricerca dei parametri di Luminance e Led Position

```
Input: terminal_conditions
Output: best_node
1.  tabu_list, node_list ← expand_initial_nodes(terminal_conditions)
2.  best_node ← node_list[0]
3.  While terminal_conditions(best_node) is False and node_list not is ∅ do
4.      nodes ← expand(best_node)
5.      For each node in nodes do
6.          If node in tabu_list do
7.              node.scores ← get_scores_from_matrix300N()
8.              Add node into tabu_list
9.          Else node.scores ← tabu_list[node].scores
10.         Add node into node_list
11.     node_list ← sort_by_heuristic(node_list)
12.     best_node ← node_list[0]
```

Una volta scelto il nodo da esaminare, viene acquisita l'immagine e, in caso di decodifica, vengono presi anche i parametri riguardanti il tempo di tale operazione e i risultati relativi agli algoritmi di verifica della qualità del codice (paragrafo 2.3). Queste informazioni sono state utilizzate per identificare l'euristica che raggruppa e ordina i nodi esplorati in base a più indicatori, partendo da quello più rilevante fino ad arrivare a quello meno significativo. Nel caso in esame (algoritmo 6.4), il primo ordinamento è ovviamente posto ad individuare quali sono i nodi che permettono la decodifica (i quali contengono le informazioni aggiuntive utili per l'esplorazione). Proseguendo, vengono considerati i parametri di qualità Minimum Reflectance, Cell Contrast e Cell Modulation usati come indicatori di visibilità e la nitidezza dei moduli del codice, per poi passare a Axial Non-Uniformity, Unused Error Correction e Grid Non-Uniformity. Infine, l'ultima discriminante è il tempo di decodifica

perché può dipendere da vari fattori non noti interni alla libreria. In generale, non si è certi che un tempo breve di lettura corrisponda indiscutibilmente ad una corretta scelta dei parametri del codice, anche se ragionevolmente si tenta di avere il valore più basso possibile.

Algoritmo 6.4: procedura "*sort_by_heuristic*" (Algoritmo 3)

```
    Input: node_list
    Output: node_list
1.  node_list ← node_list.sort (
2.          Compare by Node.read then by
3.          Node.minimum_reflectance then by
4.          Node.cell_contrast then by
5.          Node.cell_modulation then by
6.          Node.axial_non_uniformity then by
7.          Node.unused_error_correction then by
8.          Node.grid_non_uniformity then by
9.          Node.decoding_time
10.         )
```

Il criterio di terminazione è personalizzabile e consiste nel definire un valore minimo che ogni indicatore deve rispettare. Inoltre, in qualsiasi momento, se l'euristica appartenente ad un nodo soddisfa i requisiti minimi, l'algoritmo terminerà.

6.6 Auto-learn

Una volta esaminate tutte le parti che compongono l'algoritmo Auto-learn è possibile documentare l'intero processo e i risultati ottenuti. Il codice è stato sviluppato in Python utilizzando il framework Tensorflow per la parte di Deep Learning. L'algoritmo globale è composto da tre parti che vengono eseguite in sequenza:

1. Localizzazione del codice e successivo restringimento della finestra di visibilità;
2. Autofocus, che fornisce in output il valore di Reading Distance;
3. Ricerca Greedy che restituisce gli ultimi parametri di Led Position e Luminance;

Per le fasi 2 e 3 non sono necessari ulteriori processi, mentre per la fase 1 è necessario indicare dei parametri prestabiliti per poter acquisire l'immagine e proporla alla rete neurale. Inoltre, bisogna imporre una soglia che indichi se un codice viene localizzato o meno, utilizzando una strategia simile a quella di OverFeat (algoritmo 6.5).

Algoritmo 6.5: ricerca della migliore bounding box

Input: $None$
Output: $box, confidence$

$reading\_distance = [180, 100, 260]$
$luminance = [295, 186, 402, 78]$
$led\_position = [1,2,3,4]$
For each rd **in** $reading\_distance$ **do**
    For each l **in** $luminance$ **do**
        For each lp **in** $led\_position$ **do**
            $image \leftarrow get\_image\_from\_matrix300N(rd, l, lp)$
            $box, confidence \leftarrow get\_max\_inference(image)$
            If $confidence \geq 0.7$ **do End**
            Else Add $(box, confidence)$ **into** $inference\_list$
$box, confidence \leftarrow$ **Get item with max** $confidence$ **into** $inference\_list$

I valori di Luminance sono stati limitati a 4 per evitare di rallentare il processo iniziale, inoltre, sono stati inseriti in maniera decrescente perché si è osservato che in molti casi è possibile riconoscere il codice, utilizzando un alto dosaggio di luce nella scena. Come Led Position, vengono utilizzati i 4 valori che permettono l'accensione di un solo settore per poter proiettare la luce in

posizioni differenti. Riguardo la Reading Distance sono stati utilizzati i 3 valori del primo livello dell'albero di ricerca dando la precedenza a quello medio.

Come test, sono stati esaminati 19 campioni eterogenei tra loro, i quali hanno permesso di sperimentare tutte le casistiche più comuni (paragrafo 2.1). Ogni campione viene utilizzato più volte, cambiando arbitrariamente l'altezza e la posizione del codice rispetto al Matrix300N, per un totale di 37 test.

Fase	Processo	Percentuale di successo	Tempo minimo	Tempo medio	Tempo massimo
1	Localizzazione del codice	86.8%	5 *s*	16 *s*	83 *s*
2	Autofocus	93.5%	6 *s*	9 *s*	17 *s*
3	Ricerca parametri di Luminance e Led position	97.2%	2 *s*	62 *s*	171 *s*

Tabella 6.4

Risultati dei test effettuati sui tre algoritmi descritti in 6.3, 6.4 e 6.5.

I risultati dei test sui tre processi sono descritti nella tabella 6.4. Essendo sequenziali tra loro, le percentuali di successo descritte sono relative ai campioni che hanno passato le fasi precedenti. Nello specifico è stato possibile individuare il codice in modo corretto nel 86.8% delle 37 prove svolte, per un totale di 32. Quest'ultime hanno passato la fase 1 e sono state sottoposte all'algoritmo di Autofocus, il quale è riuscito ad identificare un valore accettabile Reading Distance nel 93.5% dei casi. Quindi, 30 prove su 37 hanno passato le prime due fasi e sono state impiegate nell'ultima procedura, la quale ha trovato sempre una soluzione tranne in un caso. Nella tabella 6.4 è visibile il risultato complessivo che indica i casi in cui, dato un campione, è stato possibile individuare una ricetta in grado di decodificare il codice rispettando tutti i vari vincoli presenti nelle tre fasi. È importante notare che i tempi di ricerca sono accettabili considerato l'utilizzo sporadico in condizioni che non impongono una risposta immediata.

Algoritmo	Accuratezza	Tempo minimo	Tempo medio	Tempo massimo
Auto-learn	72.8%	14 *s*	90 *s*	285 *s*

Tabella 6.5

Risultato finale complessivo dell'algoritmo Auto-Learn.

I test svolti mettono in evidenza pregi e difetti di ogni fase dell'algoritmo di Auto-learn proposto. Si è osservato che il processo di localizzazione del codice è molto abile nel suo compito, ma soffre di precisione quando la bounding-box del codice è grande quasi quanto l'area di una delle finestre in input. A differenza di quest'ultimo, l'algoritmo di autofocus non è riuscito ad individuare il giusto valore di Reading Distance quando l'immagine risultava uniforme a livello di istogramma, rendendo difficile l'identificazione della varianza maggiore nelle rilevazioni iniziali. Infine, l'algoritmo applicato nell'ultima fase risulta molto efficace, ma la discretizzazione dei parametri ha reso noto il fatto che esistono delle condizioni in cui bisogna saturare molto l'immagine, ricorrendo ad utilizzi del valore di Gain Multiplexer maggiore di 1, i quali sono stati scartati. Ad esempio, quando si utilizzano prodotti di colore scuro e i moduli del codice non sono nettamente messi in risalto dallo sfondo.

7 Conclusioni

In questa tesi è stato affrontato un problema di ricerca automatica al fine di decodificare codici Data Matrix utilizzando un lettore professionale (Matrix300N). Sono state quindi affrontate due delle principali tematiche dell'intelligenza artificiale, fondamentali per raggiungere l'obiettivo prefissato, gli algoritmi di ottimizzazione e il Machine Learning.

I risultati ottenuti, osservabili in 6.6, mostrano che l'intero sistema, seppur in versione prototipale, riesce ad ottenere dei risultati concreti e accettabili. Scomporre il sistema in tre parti permette di ottenere algoritmi ottimizzati, semplici e veloci. Tale metodologia presenta come problematica principale il fatto che se uno dei tre processi interni fallisce, allora la soluzione finale sarà compromessa. Questo accade soprattutto quando la rete neurale non localizza bene il codice o quando l'autofocus predilige risaltare parti dell'immagine che non sono nello stesso piano del codice.

È necessario, quindi, continuare a migliorare ogni singolo passo. Partendo dalla localizzazione, sarebbe opportuno ampliare il data set (utilizzando DPM DB) per poi provare a costruire un modello più accurato. Un altro esperimento potrebbe essere quello di utilizzare algoritmi genetici o PSO per l'ottimizzazione negli ultimi due processi.

Bibliografia

[Bruno, 2010] Bruno G. (2010). *Gli Algoritmi Genetici*, www.federica.unina.it, pp 1-30.

[Canziani et al, 2016] Canziani Alfredo, Culurciello Eugenio, Paszke Adam. (20 maggio 2016). *An Analysis of Deep Neural Network Models for Practical Applications*, arxiv.org, pp 1-7.

[Carello, 2015] Carello G. (2015). *Tecniche euristiche -- greedy*, home.deib.polimi.it, pp 1-16.

[Chudzicki, 2012] Chudzicki D. (06 Giugno 2012). *Metrics*, kaggle.com. Estratto il 12 Dicembre 2017, da https://www.kaggle.com/wiki/Metrics/history/1362

[Cognex Corporation, 2017, A] Cognex Corporation. (2017). *Direct Part Marking*, cognex.com. Estratto il 19 marzo 2017, da http://www.cognex.com/symbologies/direct-part-marking/?pageid=14040&langtype=1033

[Cognex Corporation, 2017, B] Cognex Corporation. (2017). *DataMan 200/300 Series Barcode Readers*, cognex.com. Estratto il 26 marzo 2017, da http://www.cognex.com/products/barcode-readers-scanners/dataman-300-fixed-mount-dpm-barcode-reader/

[Cognex Corporation, 2006] Cognex Corporation. (2006). *Implementing Direct Part Mark Verification - Control Design*, cognex.com, pp 11-12.

[Datalogic, 2016, A] Datalogic. (2016). *Matrix 300N™*, datalogic.com. Estratto il 26 marzo 2017, da http://www.datalogic.com/eng/products/industrial-automation/fixed-industrial-barcode-readers/matrix-300n-pd-659.html

[Datalogic, 2016, B] Datalogic. (2016). *Matrix 300N™ Reference Manual*, datalogic.com, pp 1-182.

[Datalogic, 2016, C] Datalogic. (2016). *Matrix N Family, Guide to Lighting System Selection for DPM Applications*, pp 12-27.

[Datalogic, 2014] Datalogic. (2014). *Matrix 300N™ Host Mode Programming*, datalogic.com, pp 1-69.

[Du, 2016] Du L-K & Swamy MNS. (2016). *Search and Optimization by Metaheuristics*, Springer, pp 29-33.

[Fei-Fei et al., 2016] Fei-Fei L, Andrej K & Justin J. (1 Febbraio 2016). *Lecture 8: Spatial Localization and Detection*, cs231n.stanford.edu, pp 1-90.

[Frangioni, 2010] Frangioni A. (2010). *Algoritmi Euristici*, di.unipi.it, pp 217-265.

[Goodfellow et al., 2016] Goodfellow I, Bengio Y, Courville A. (2016). *Deep Learning*, MIT Press, pp 1-523.

[GS1, 2016] GS1. (2016). *GS1 DataMatrix Guideline*, gs1.org, pp 8-12.

[Karpathy, 2017] Karpathy A. (2017). *Transfer Learning and Fine-tuning Convolutional Neural Networks*, cs231n.github.io. Estratto il 03 Dicembre 2017, da http://cs231n.github.io/transfer-learning/

[Keyence Corporation, 2015] Keyence Corporation. (2015). *SR-1000 User Manual*, keyence.com, pp 41-46.

[Krizhevsky et al., 2012] Krizhevsky A, Sutskever I, Hinton GE. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*, papers.nips.cc, pp 1-8.

[LeCun et al., 1998] LeCun Y, Bottou L, Bengio Y, Haffner P. (Novembre 1998). *Gradient-Based Learning Applied to Document Recognition*, yann.lecun.com, pp 1-44.

[Lippi, 2016, A] Lippi M. (2016). *Deep Learning: a revolution for AI*, lia.disi.unibo.it, pp 3-46.

[Lippi, 2016, B] Lippi M. (2016). *Convolutional Neural Networks*, lia.disi.unibo.it, pp 1-24.

[Lippi, 2016, C] Lippi M. (2016). *Recurrent Neural Networks*, lia.disi.unibo.it, pp 1-27.

[Manning et al., 2009] Manning CD, Raghavan P, Schütze H. (1 Aprile 2009). *An Introduction to Information Retrival*, Cambridge University Press, pp 319-333.

[Mausam, 2011] Mausam. (2011). *Local Search and Optimization*, courses.cs.washington.edu, p 37.

[Mello, 2017] Mello P. (2017). *Strategie di Ricerca informate*, lia.disi.unibo.it, pp 43-50.

[Microscan, 2013] Microscan. (2013). *Understanding Machine Vision Verification of 1D and 2D Barcodes*, microscan.com, pp 1-4.

[Milano, 2017, A] Milano M. (2017). *Swarm Intelligence*, ai.unibo.it, pp 1-39.

[Milano, 2017, B] Milano M. (2017). *Machine Learning*, ai.unibo.it, pp 1-62.

[Milano, 2017, C] Milano M. (2017). *Neural Networks*, ai.unibo.it, pp 1-105.

[Milano, 2017, D] Milano M. (2017). *Deep Learning*, ai.unibo.it, pp 1-80.

[Min et al., 2014] Min L, Qiang C, Shuicheng Y. (4 Marzo 2014). *Network In Network*, arxiv.org, pp 1-4.

[Mir et al., 2014] Mir H, Xu P & Van Beek P. (2014). *An extensive empirical evaluation of focus measures for digital photography*, cs.uwaterloo.ca, pp 1-11.

[Ojha, 2017] Ojha VK, Abraham A & Snášel V. (16 Maggio 2017). *Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research*, arxiv.org, pp 1-35.

[Parthasarathy, 2017] Parthasarathy D. (22 Aprile 2017). *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*, blog.athelas.com. Estratto il 03 Dicembre 2017, da https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

[Pech-Pacheco et. al, 2000] Pech-Pacheco JL, Cristóbal G, Chamorro-Martínez J & Fernández-Valdivia J. (2000). *Diatom autofocusing in brightfield microscopy: a comparative study*, optica.csic.es, pp 1-4.

[Raschka, 2015] Raschka S. (24 Marzo 2015). *Single-Layer Neural Networks and Gradient Descent*, sebastianraschka.com. Estratto il 05 Novembre 2017, da http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

[Redmon et al., 2016] Redmon J, Divvala S, Girshick R, Farhadi A. (9 Maggio 2016). *You Only Look Once: Unified, Real-Time Object Detection*, arxiv.org, pp 1-6.

[Ruder, 2017] Ruder S. (15 Giugno 2017). *An overview of gradient descent optimization algorithms*, arxiv.org, pp 1-10.

[Russell & Norvig, 2016] Russell S & Norvig P. (2016). Goodfellow I, Bengio Y, Courville A. (2016). *Artificial Intelligence: A Modern Approach*, Pearson.

[Sartori, 2012, A] Sartori C. (2012). *Classificatori con Alberi di Decisione*, campus.unibo.it, pp 29-123.

[Sartori, 2012, B] Sartori C. (2012). *Classificatori: altri metodi*, campus.unibo.it, pp 64-76.

[Sartori, 2012, C] Sartori C. (2012). *Valutazione di un classificatore*, campus.unibo.it, pp 1-23.

[Sermanet et al., 2014] Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R & LeCun Y. (2014). *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*, arxiv.org, pp 1-12.

[Simonyan & Zisserman, 2015] Simonyan K, Zisserman A. (10 Aprile 2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arxiv.org, pp 1-8.

[Szegedy et al., 2014] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Erhan D, Vanhoucke V & Rabinovich A. (2014). *Going deeper with convolutions*, arxiv.org, pp 1-12.

[Ventura, 2017] Ventura V. (2017). *METODI GREEDY*, uniroma2.it, pp 3-23.

[Videojet, 2014] Videojet. (2014). *Methods for direct part marking*, videojet.com, pp 3-6.

[Zhang, 2014] Zhang H. (23 Settembre 2014). *Informed Search*, homepage.divms.uiowa.edu, pp 1-4.