

Simple Book Example

TeXstudio Team

January 2013

Contents

1	Introduzione	1
1.1	Cos'è un'immagine?	1
1.1.1	Definizioni	1
1.1.2	Ottenere un'immagine	2
1.1.3	Quantizzazione	2
1.2	Operazioni con le immagini	2
1.2.1	Operazioni pixel a pixel	2
1.2.2	Operazioni su più pixel	3
1.3	Dimensione dell'immagine	3
1.3.1	Risoluzione	3
1.4	Campionamento	4
2	Il Colore	5
2.1	Descrizione fisica	5
2.1.1	Definizioni	5
2.2	Modelli di colore	5
2.2.1	Cubo RGB	6
2.2.2	Modelli HS*	6
2.2.3	Spazio HSV	6
2.2.4	Spazio HSL	7
2.2.5	Spazio HSI	7
2.3	Trasformazioni di colore	7
2.3.1	Trasformazioni in Pseudocolore	7
2.3.2	Trasformazione in falsi colori	8
3	Image Processing	9
3.1	Image enhancement	9
3.2	Image restoration	9
3.3	Compressione	10
3.3.1	Compressione lossless	10
3.3.2	Compressione lossy	10

3.4 Segmentation	10
4 Enhancement nel dominio spaziale	11
4.1 Trasformazioni puntuali	11
4.1.1 Look Up Table	12
4.1.2 Bit-plane slicing	12
4.2 Trasformazioni globali	13
4.2.1 Istogramma	13
4.3 Filtri	15
4.3.1 Caratteristiche generali	15
4.3.2 Filtri passa-basso	15
4.3.3 Filtri passa-alto	16
4.4 Combinare gli strumenti	18
5 Processing di immagini a colori	19
5.1 Color slicing	19
5.1.1 Segmentare col modello HSI	19
5.1.2 Segmentare col modello RGB	20
6 Enhancement con Fourier	21
6.1 Trasformata di Fourier continua	21
6.1.1 Trasformate notevoli	21
6.1.2 Convoluzione	23
6.2 Sampling e DTFT	23
6.2.1 Teorema del campionamento	24
6.2.2 Aliasing	24
6.3 Trasformata di Fourier Discreta	25
6.4 Trasformata 2D continua	25
6.5 Sampling 2D	25
6.5.1 Teorema del campionamento	26
6.5.2 Aliasing 2D	26
6.6 Trasformata 2D discreta	27
6.6.1 Traslazioni e rotazioni	27
6.6.2 Convoluzione 2D	27
6.7 Filtraggio nel dominio della frequenza	28
6.7.1 Zero Padding	28
6.7.2 Progettare un filtro	28
7 Image Restoration	31
7.1 Rumore	32
7.1.1 Stimare il rumore	32

7.2	Rumore spazio invariante	33
7.2.1	Filtri semplici	33
7.2.2	Filtri adattativi	34
7.3	Rumore spazio dipendente	36
7.3.1	Notch Filter	36
7.4	Danni lineari spazio invarianti	38
7.4.1	Filtro inverso	38
7.4.2	Wiener filter	40
7.4.3	CLSF	41
7.4.4	Filtro con media geometrica	42
7.4.5	Esempi	42
8	Low Level Vision	43
8.1	Segmentation	43
8.1.1	Thresholding	43
8.1.2	Region growing	47
8.1.3	Region splitting and merging	47
8.2	Feature Extraction	48
8.2.1	Edge Detection	48
8.2.2	Line Detection	50
9	AI per la visione artificiale	51
9.1	Percettrone	51
9.1.1	Funzionamento	51
9.2	Multilayer networks	52
9.2.1	Funzionamento	53
9.2.2	Overfitting	53
9.2.3	Multiclass Classification	54
9.2.4	Loss function	55
9.3	Riconoscere immagini	55
9.3.1	LeNet 300	56
9.3.2	Un approccio diverso	56
10	Convolutional Neural Network	57
10.1	Neurone convoluzionale	57
10.2	Complessità del layer convoluzionale	57
10.2.1	Complessità della rete	58
10.2.2	Ridurre la complessità	58
10.3	LeNet5	59
10.4	Receptive Field	59
10.5	Da fully connected a fully convolutional	60

10.5.1 Riconoscimento del testo	60
10.5.2 Sliding Window	60
10.5.3 Equivalente convoluzionale	61
10.5.4 Interpretare i risultati	62
11 Advanced Learning	63
11.1 Learning rate	63
11.1.1 Decadimento del learning rate	64
11.2 Stochastic gradient descent	65
11.2.1 Aggiunta della quantità di moto	65
11.2.2 Tecniche avanzate di discesa del gradiente	65
11.3 Batch Training	66
11.3.1 Minibatch	66
11.4 Overfitting	67
11.4.1 Data Augmentation	67
11.4.2 Committees of Neural Networks	67
11.4.3 Dropout	68

Chapter 1

Introduzione

1.1 Cos'è un immagine?

1.1.1 Definizioni

Un'immagine $f(x, y)$ è una rappresentazione su un piano 2D di:

- una scena o un fenomeno fisico misurabile;
- un modello definito analiticamente.

A seconda di come si ottiene l'immagine si parla di:

Immagini visibili quelle che possono essere viste dall'occhio umano o sono catturate da un sistema ottico;

Immagini non direttamente visibili generate a partire da una distribuzione di grandezze fisiche come ad esempio una mappa di temperature;

Immagini definite da modelli analitici ottenute a partire da funzioni continue o discrete, come la sintesi di modelli 3D.

Le immagini possono essere analogiche, caratterizzate da funzioni continue $f(x, y)$ che definiscono il colore e l'intensità luminosa oppure **digitali**, nelle quali la funzione di prima viene approssimata due volte:

- campionando: il piano XY viene diviso con una griglia e consideriamo un solo valore di colore per ogni cella della griglia (la chiameremo pixel)
- quantizzando: dovendo immagazzinare l'informazione del pixel in un calcolatore non abbiamo a disposizione infinita precisione, dovremo perciò troncare il valore

1.1.2 Ottenerne un'immagine

Dal punto di vista tecnico per ottenere un'immagine campionata e quantizzata con una fotocamera si fa arrivare con il sistema ottico la luce al sensore, il sensore è diviso in celle che permettono di ottenere ciascuna un pixel (campionamento o sampling), ogni cella traduce l'intensità luminosa che la colpisce in un segnale elettrico che viene tradotto in un valore tipicamente intero tra 0 e 255 (quantizzazione). Per ottenere l'immagine a colori si sovrappone al sensore una griglia colorata che permetta a ogni cella del sensore di assorbire solo la luce di un certo colore (RGB)

1.1.3 Quantizzazione

il valore di ogni pixel deve essere espresso attraverso un certo numero finito di bit, più bit usiamo più saremo in grado di definire colori differenti, il numero di bit è quindi detto **color depth** o **profondità di colore** e si indica con l'acronimo **bpp** cioè bit per pixel. Se un'immagine è in bianco e nero significa che per ogni pixel salviamo un solo bit, l'immagine è detta **binarizzata**

Normalmente usiamo 8 bit per esprimere un valore, da questo segue che un'immagine in scala di grigi avrà 256 livelli di grigio. Per le immagini a colori dividiamo il valore del pixel nelle componenti RGB detti anche **canali**, destiniamo a ogni canale 8 bit ottenendo 16 milioni di colori circa.

1.2 Operazioni con le immagini

Possiamo definire delle operazioni sulle immagini che prendano in input una o due immagini e restituiscano uno scalare, un vettore o un'altra immagine

1.2.1 Operazioni pixel a pixel

Operazioni logiche

Si applicano a due immagini binarizzate aventi la stessa dimensione e si ottiene una terza immagine in cui ogni pixel è il risultato dell'operazione logica sui pixel corrispondenti nelle immagini iniziali.

Operazioni aritmetiche

Possiamo applicarle a due immagini di qualsiasi tipo purché con la stessa dimensione. Il risultato è un'immagine in cui ogni pixel è ottenuto applicando una certa funzione aritmetica ai pixel di partenza corrispondenti. Quando si

lavora con queste operazioni si deve tenere conto della possibilità di sforare il range di rappresentazione dei pixel, il risultato dovrà essere quindi opportunamente clippato o scalato.

Trasformazioni affini

Queste operazioni permettono di traslare, scalare, ruotare e fare shear di un'immagine, facendo in modo che le linee dritte rimangano tali. Queste operazioni si implementano con **matrici** di **trasformazione** T .

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad (1.1)$$

1.2.2 Operazioni su più pixel

Le operazioni che coinvolgono più pixel di una certa immagine sono tipicamente implementate mediante **filtri** queste operazioni sono importantissime e ne parleremo più approfonditamente in seguito (??).

1.3 Dimensione dell'immagine

Quando si parla di dimensione si intende il numero di pixel che costituiscono l'immagine, dato che le immagini sono tipicamente rettangolari e i pixel sono organizzati in una griglia la dimensione dell'immagine si ottiene moltiplicando la base per l'altezza.

Una volta ottenuta la dimensione è facile calcolare lo **spazio occupato** dall'immagine, se usiamo 8 bit per canale e abbiamo 3 canali basta moltiplicare per 3 la dimensione dell'immagine per avere la sua occupazione in byte. Si vede che questa, anche per immagini non grandissime è molto elevata, nel tempo quindi si sono sviluppati efficientissimi algoritmi di compressione delle immagini; ne parleremo più avanti (??).

1.3.1 Risoluzione

La risoluzione ci da un'idea di quanto possa essere gradevole guardare un'immagine senza vedere la grana dei pixel. Per questa ragione è una funzione della dimensione e dell'area fisica su cui l'immagine viene stampata o proiettata. In particolare la risoluzione si calcola in **punti per pollice** (dpi) e vale la seguente relazione:

$$\text{dimensione} = \text{area} \cdot \text{risoluzione} \quad (1.2)$$

1.4 Campionamento

Quando acquistiamo un’immagine reale stiamo discretizzando un fenomeno fisico, dalla teoria dell’informazione conosciamo il **teorema di Nyquist-Shannon** che afferma che se vogliamo campionare correttamente dobbiamo usare una frequenza di campionamento almeno pari al doppio della frequenza massima presente nel segnale che vogliamo acquisire.

Nelle immagini le alte frequenze corrispondono a bordi di colori molto diversi ravvicinati fra loro, mentre la frequenza di campionamento è data da quanto è fitta la griglia di celle del nostro sensore. Quando acquistiamo un’immagine non possiamo fare come nell’audio e usare dei filtri passa basso per eliminare le frequenze troppo alte, perché questo equivale a sfocare l’immagine. Possono pertanto presentarsi degli artefatti dovuti all’errato campionamento di determinati pattern; questi sono detti **Moiré**.

Chapter 2

Il Colore

2.1 Descrizione fisica

2.1.1 Definizioni

Il colore è la traduzione da parte del nostro cervello di un fenomeno fisico. Possiamo descrivere questo fenomeno come uno spettro che assegna a ogni lunghezza (o frequenza) d'onda visibile la sua energia. Sullo spettro definiamo:

Lunghezza d'onda dominante la lunghezza d'onda della componente più energetica, è detta anche **hue**

Saturazione il rapporto tra l'energia della hue e il valor medio dello spettro (luce bianca)

Radianza l'energia emessa da una certa sorgente (è l'integrale dello spettro)

Luminanza energia percepita da un osservatore

Brightness intensità della componente acromatica

Cromaticity o **chroma** hue + saturazione

2.2 Modelli di colore

Per descrivere un colore si possono usare vari modelli che differiscono nel modo di rappresentare l'informazione, ma sono tutti equivalenti e hanno la stessa capacità espressiva. Questi modelli definiscono dei volumi di colore in cui ciasun punto rappresenta un colore diverso, sono per questo detti **spazi colore**.

2.2.1 Cubo RGB

In questo modello si definisce un cubo di lato 1 avente un vertice nell'origine degli assi. Ogni asse rappresenta una componente e ogni punto all'interno del cubo sarà un colore avente un ben preciso rapporto tra le componenti. Questo modello è molto semplice e adatto a un calcolatore, ma è difficile da usare per gli umani, sono state quindi proposte alternative.

2.2.2 Modelli HS*

Questa famiglia di modelli si basano sul modo umano di rappresentare il colore e descrivono

- **Hue:** la tinta del colore
- **Saturazione:** quanto il colore è puro
- **Luminosità:** quanto ci appare brillante

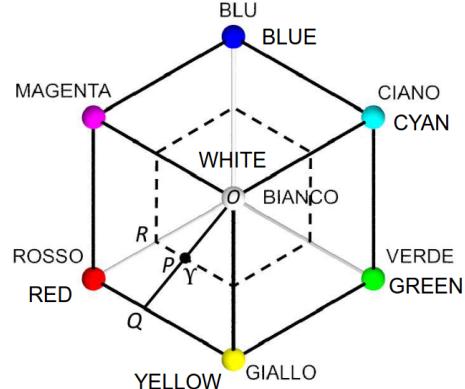
Tutti i modelli HS* sono derivati applicando delle trasformazioni al cubo RGB, in particolare in tutti i modelli si ruota il cubo in modo che la diagonale che congiunge il bianco al nero sia verticale.

Il piano che si ottiene guardando dall'alto il cubo ruotato è detto **chromaticity plane**. Su questo si definisce la **chroma** e la **hue** del colore Υ

$$\text{chroma}(\Upsilon) = \frac{OP}{OQ} \quad (2.1)$$

$$\text{hue}(\Upsilon) = \frac{RP}{6OR} \quad (2.2)$$

Cioè la chroma è il rapporto tra la distanza del colore dal bianco e la distanza dal bianco della tinta pura; mentre la hue è il rapporto tra la distanza dal rosso (in senso orario) e il perimetro dell'esagono.



2.2.3 Spazio HSV

In questo spazio il piano della cromaticità viene alzato fino al bianco, si rende questo esagono un cerchio e la base si trasforma in un cerchio nero, ottenendo

alla fine un cilindro. Hue e croma sono definite come prima, si aggiunge il $value(\Upsilon)$ che rappresenta l'altezza sul cilindro. Si definisce poi la saturazione come:

$$saturation(\Upsilon) = \begin{cases} 0 & \text{if } value(\Upsilon) = 0 \\ \frac{chroma(\Upsilon)}{value(\Upsilon)} & \text{otherwise} \end{cases} \quad (2.3)$$



2.2.4 Spazio HSL

Anche in questo spazio colore si ottiene un cilindro ma il piano della cromaticità rimane al centro, mentre la faccia superiore è completamente bianca e quella inferiore completamente nera. In questo caso l'altezza sul cilindro si chiama $lightness(\Upsilon)$ (abbreviata nella formula sotto con $light$). La saturazione invece è:

$$saturation(\Upsilon) = \begin{cases} 0 & light(\Upsilon) = 0 \text{ or } 1 \\ \frac{chroma(\Upsilon)}{1-|2\cdot light(\Upsilon)-1|} & otherwise \end{cases} \quad (2.4)$$



2.2.5 Spazio HSI

È un modello che cerca il migliore compromesso tra comodità di rappresentazione ed efficienza sul calcolatore. Si parte sempre dal cubo ruotato, per poi normalizzarlo in modo che la diagonale sia lunga 1. In questo modello definiamo l'intensità I come la somma delle componenti colorate diviso tre e la saturazione come 1 meno la componente più piccola, tutto diviso l'intensità. Con queste definizioni il modello rimane comodo per un utilizzatore e la conversione nelle componenti RGB risulta molto efficiente.

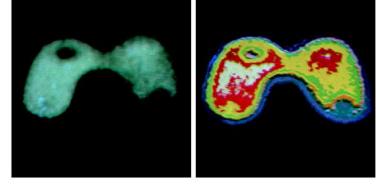
2.3 Trasformazioni di colore

2.3.1 Trasformazioni in Pseudocolore

Un immagine in scala di grigi con una profondità colore di 8 bit permette di avere 256 livelli di grigio differenti, nelle immagini mediche la profondità colore è ancora maggiore. L'occhio umano però non è sensibile a così tanti livelli di grigio, per migliorare la percezione si può trasformare una immagine monocromatica in una a falsi colori.

Intensity slicing

È una particolare trasformazione in pseudocolori che consiste nell'associare a ogni range di valori in scala di grigi un colore. In questo modo ogni pixel è colorato in base al suo livello di intensità. Come si vede dall'esempio questo metodo permette di distinguere molto meglio le varie zone dell'immagine.



2.3.2 Trasformazione in falsi colori

In questo caso si parte da un'immagine a colori e si mappano i vari colori su altri colori, a esempio per evidenziare particolari dettagli. Una trasformazione di questo tipo può essere espressa come matrice di trasformazione che prende come input il vettore che contiene le componenti di colore di ciascun pixel:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.5)$$

Chapter 3

Image Processing

In questo capitolo vediamo brevemente i principali tipi di processing di immagini e tipo per tipo quali sono gli obiettivi della manipolazione.

3.1 Image enhancement

Questo tipo di processing ha lo scopo di migliorare l'immagine in modo **soggettivo** e mirato alle elaborazioni future, è quindi un trattamento a monte prima di utilizzare l'immagine per ulteriori manipolazioni. Elaborazioni tipiche di enhancement sono la riduzione del rumore e miglioramento del contrasto.

L'image enhancement può anche essere l'unico step di processing e in questo caso ha lo scopo di rendere più gradevole l'immagine a un osservatore umano.

L'aspetto che caratterizza questa elaborazione è quella di essere application-driven, nel senso che la manipolazione è pensata ad hoc per il passo successivo a cui andrà incontro l'immagine (che sia quello di essere osservata oppure di essere ulteriormente elaborata).

3.2 Image restoration

Questa elaborazione ha lo scopo di aumentare la qualità dell'immagine in modo **oggettivo**, andando ad esempio a recuperare delle informazioni perse o correggendo errori fatti durante il processo di acquisizione.

Per operare questo recupero di informazioni è necessario avere una conoscenza del dominio specifico e della funzione di degradazione che ha comportato la perdita di informazioni.

3.3 Compressione

Come accennato nel paragrafo sulla dimensione delle immagini (1.3) queste possono arrivare a occupare notevole spazio in memoria, la compressione si pone allora l'obbiettivo di ridurre lo spazio occupato dalle immagini, sia per risparmiare memoria sia per ridurre i tempi di trasmissione. Distinguiamo tra due tipi di compressione:

3.3.1 Compressione lossless

In cui l'immagine decompressa è esattamente identica all'immagine originale. Con questo tipo non si riescono a raggiungere livelli di compressione molto elevati, ha comunque senso in particolari applicazioni in cui è importante mantenere tutti i dati (ambito medico a esempio).

3.3.2 Compressione lossy

L'immagine decompressa è diversa dall'immagine originale, tipicamente siamo interessati a ottenere dei difetti impercettibili per l'occhio umano. Tecniche di compressione lossy si possono basare su

- **subsampling**: ridurre il numero di pixel memorizzati, per fare questo bisogna fare attenzione a non introdurre artefatti come le moiré;
- **ridurre la quantizzazione**: per ogni pixel riduciamo la profondità colore.

Queste tecniche si possono anche mischiare e rendere adattative, ad esempio usando una grana fine e una buona profondità colore per i soggetti a fuoco e riducendo la qualità per le parti non in evidenza.

3.4 Segmentation

Consiste nel partizionare un'immagine estraendo quelle parti alle quali siamo interessati. Anche questa elaborazione può essere fatta a monte di altre che ad esempio tentano di classificare le parti segmentate o cercano difetti nei componenti prodotti a cui è stata scattata una foto.

Dopo la segmentazione quindi è importante riuscire a rappresentare le parti segmentate e associare ad esse delle caratteristiche alle quali siamo interessati, ad esempio l'area di queste parti, il perimetro, la presenza di difetti, curvature ecc...

Chapter 4

Enhancement nel dominio spaziale

In questo capitolo vediamo le principali tecniche di enhancement nel dominio dello spazio, ovvero modificando il valore dei pixel attraverso funzioni che prendono in input uno o più pixel dell'immagine che intendiamo trasformare.

Dividiamo il tipo di funzione che applichiamo all'immagine a seconda degli argomenti che prende in input:

Trasformazioni puntuale che operano pixel a pixel e sostanzialmente mappano i livelli di intensità in altri livelli;

Trasformazioni locali prendono in input un pixel e i suoi vicini, sono quelle in assoluto più usate e sono chiamate **filtri**;

Trasformazioni globali prendono come argomento l'intera immagine, tipicamente non ci permettono di ottenere una nuova immagine, ma servono a estrarre informazioni globali sull'immagine.

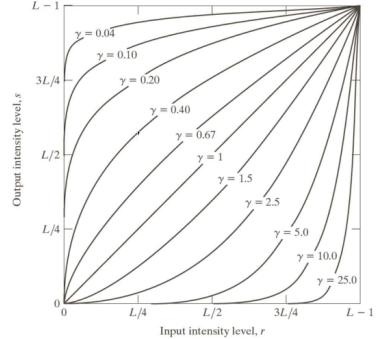
4.1 Trasformazioni puntuale

Anche se queste funzioni permettono semplicemente di rimappare il livello di intensità pixel a pixel sono molto utili per ottenere immagini più ricche di informazione e correggere eventuali errori dovuti all'acquisizione o allo strumento di visualizzazione dell'immagine.

Sono inoltre queste trasformazioni che ci permettono di ottenere immagini in pseudocolori o in falsi colori. Queste trasformazioni sono particolarmente utili in ambito medico e scientifico, dove è necessario evidenziare dettagli su immagini ottenute attraverso sensori che catturano fenomeni non osservabili otticamente.

4.1.1 Look Up Table

I metodi che prevedono una funzione per rimappare i livelli di grigio solitamente si basano su Look Up Table (**LUT**) che specificano per ogni livello di grigio in ingresso il nuovo livello di grigio del pixel in uscita. Nell’immagine si può vedere una tipica trasformazione generata con una Power-Law, questa trasformazione serviva per correggere la gamma dei televisori a tubo catodico e prende quindi il nome di **gamma correction**.



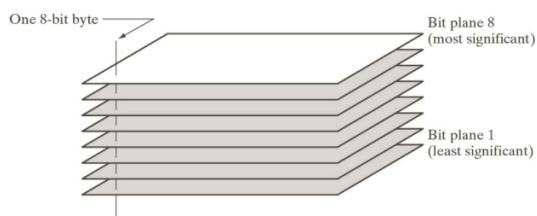
Tipologie di LUT

Non c’è nessuna regola sul come costruire una LUT per rimappare i livelli di grigio, a seconda dell’obiettivo che si vuole raggiungere si può costruire la LUT più adatta. Vediamo alcune tipologie particolarmente importanti:

- **scala**: serve per ridurre il numero di livelli di grigio, opera una compressione lossy e può generare falsi bordi;
- **rampa**: serve per incrementare il contrasto in una particolare regione, ovviamente se si incrementa da una parte si riduce al di fuori della rampa;
- **binario**: permette di binarizzare l’immagine mettendo al massimo i livelli in un certo range e tutti gli altri a zero.
- **pseudocolor**: permette di colorare un’immagine in scala di grigi, consiste in realtà in tre diverse LUT che permettono di ottenere ciascuna una diversa componente RGB

4.1.2 Bit-plane slicing

Questa trasformazione permette di ottenere sempre immagini binarizzate. Per fare questo consideriamo l’immagine suddivisa in livelli, se la profondità colore è 8 ci saranno 8 livelli e il terzo sarà costituito da tutti i bit in



terza posizione. Se consideriamo solo un livello otteniamo l'immagine binarizzata, possiamo poi unire diversi livelli assieme.

A seconda di che livello consideriamo possiamo ottenere differenti informazioni, perfino il livello associato al bit meno significativo (che genera un'immagine apparentemente solo rumorosa) può portare informazioni utili, oppure può permettere di inserire informazioni utili non visibili come watermark.

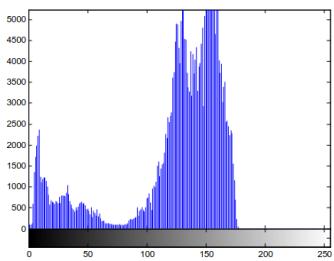
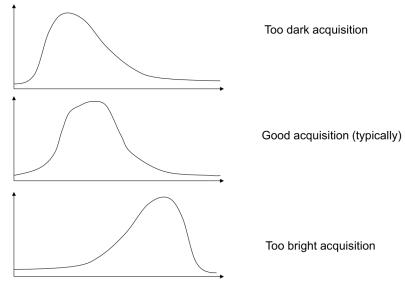
4.2 Trasformazioni globali

Permettono di ottenere informazioni utili sull'immagine e usarle per migliorare difetti tipici quali la sotto/sovra esposizione o basso contrasto.

4.2.1 Iistogramma

Un indicatore molto utilizzato nelle trasformazioni globali è l'istogramma, che permette di ottenere la distribuzione in frequenza dei vari livelli di grigio di un'immagine.

Semplicemente guardando la forma dell'istogramma ci si può rendere conto della bontà dell'immagine, o per lo meno della corretta esposizione. In un'immagine ci aspettiamo una distribuzione normale dai vari livelli di grigio, con pochi pixel che saturano il sensore in alto o in basso.



L'istogramma può inoltre essere utile per stimare la soglia di threshold più adatta a binarizzare un'immagine, infatti se siamo di fronte a un istogramma bimodale possiamo essere ragionevolmente certi che un picco rappresenti lo sfondo, mentre l'altro gli oggetti di nostro interesse. Tipicamente questa situazione si ritrova nei contesti di visione industriale e raramente nelle immagini naturali. Nell'immagine vediamo una situazione in cui è particolarmente evidente il valore che ci permette di avere una binarizzazione ragionevole dell'immagine; siamo quindi in una situazione molto favorevole.

Equalizzazione dell'istogramma

Una volta ottenuto l'istogramma di un immagine si può tentare di equalizzarlo in modo da migliorare l'immagine. Un istogramma concentrato solo su alcuni valori fa perdere dinamicità all'immagine, può quindi avere senso cercare di ripristinare una distribuzione uniforme dei valori dei pixel. Alla fine dell'equalizzazione vorremmo essere in una situazione come quella mostrata in figura qui sotto.

Per ottenere questo risultato consideriamo l'istogramma dell'immagine: $h(r)$ che associa a ogni livello di grigio r il numero di pixel con quel valore. Definiamo l'istogramma cumulativo

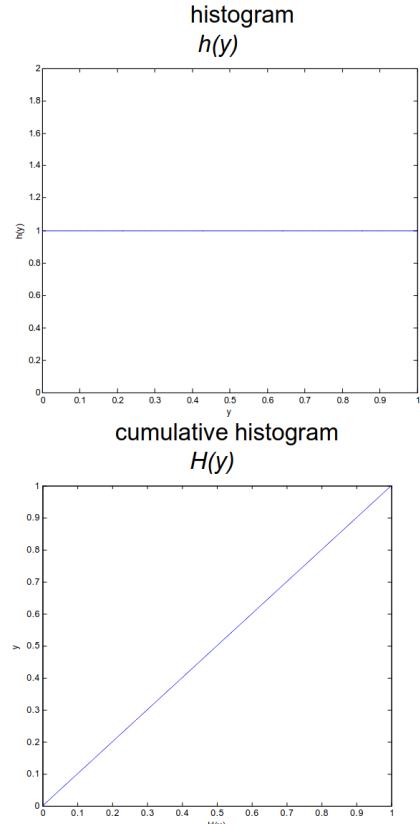
$$H(r) = \sum_{i=0}^r h(i) \quad (4.1)$$

e la funzione di trasformazione

$$T(r) = (L - 1) \cdot \frac{H(r)}{M \cdot N} \quad (4.2)$$

Dove L è il numero di livelli di grigio, e M e N sono le dimensioni dell'immagine. Infine per ottenere l'immagine equalizzata trasformiamo ciascun pixel con

$$y(i, j) = T[x(i, j)] \quad (4.3)$$



Applicando queste trasformazioni possiamo ottenere un immagine con un istogramma equalizzato. È possibile che diversi livelli di grigio dell'immagine di partenza vengano mappati nello stesso livello di grigio dell'immagine di arrivo, è quindi necessari prestare attenzione alla possibile perdita di informazioni.

Si può costruire una trasformazione anche per ottenere una nuova immagine con un istogramma specificato, non per forza con distribuzione uniforme. Per fare questo costruisce T che equalizza l'immagine di partenza, G che equalizza l'istogramma che vogliamo ottenere, infine la trasformazione che permette di ottenere l'istogramma voluto sarà $z = G^{-1}(T(r))$.

4.3 Filtri

4.3.1 Caratteristiche generali

I filtri permettono di applicare modifiche locali all’immagine e possono essere progettati sia nel dominio dello spazio che della frequenza. In questa prima e più semplice trattazione ci concentriamo sui filtri nel dominio dello spazio, indagando come certe caratteristiche dell’immagine possano essere messe in evidenza o nascoste a seconda di come progettiamo il filtro.

I filtri sono trasformazioni locali che si basano sulla **convoluzione** fra i pixel dell’immagine e una matrice che rappresenta il filtro stesso. Ci occuperemo di filtri:

- Spazio invarianti;
- Lineari.

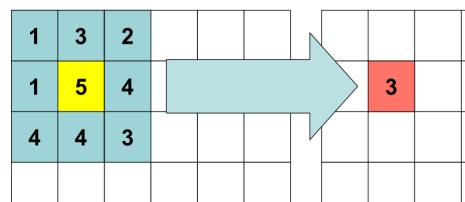
Date queste caratteristiche possiamo studiare l’effetto di un filtro semplicemente osservando la **risposta all’impulso**. Siccome progetteremo sempre filtri 2D simmetrici fare la convoluzione sarà equivalente a sovrapporre la matrice quadrata che rappresenta il filtro all’immagine, moltiplicare i pixel per i pesi corrispondenti, sommare i risultati e immagazzinare il nuovo valore nel pixel centrale.

4.3.2 Filtri passa-basso

I filtri passa basso come suggerito dal nome sono filtri che eliminano le componenti a frequenza più alta, cioè si concentrano sui bordi dell’immagine sfumandoli.

Media mobile

Il più semplice filtro passa-basso è la media mobile, consiste nello scegliere una certa dimensione del filtro (**finestra**) e fare la media di tutti i pixel che si trovano nella finestra. Vedremo che un approccio di questo tipo può generare artefatti quando progetteremo filtri nel dominio della frequenza ??.



Mediana

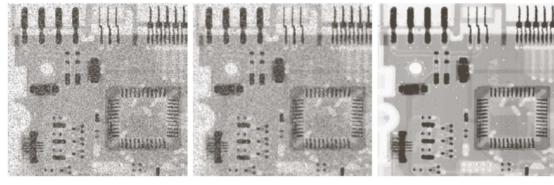


FIGURE 3.35 (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

che potrebbero essere presenti. Si comporta particolarmente bene per la rimozione del rumore sale e pepe.

Questo è un filtro non lineare, dato che la funzione che applica non è lineare, è comunque un filtro interessante dato che fa una statistica dei pixel all'interno della finestra e prende la mediana tra i pixel. Per questa ragione riesce a eliminare gli outlier particolarmente bene per la

4.3.3 Filtri passa-alto

I filtri passa alto servono per aumentare la nitidezza dell'immagine (**sharpening**) ed estrarre i bordi. Si ottengono cercando di calcolare le derivate dell'immagine, basandosi sull'idea che nei pressi dei bordi la derivata sarà elevata dato che c'è un cambio repentino nel valore dei pixel.

Istogramma

Prima di progettare dei filtri ad hoc proviamo ad applicare strumenti già visti per aumentare il contrasto lungo i bordi. Possiamo ridefinire il concetto di equalizzazione in modo locale, andando considerare l'immagine costituita da sotto-immagini e equalizzando ciascuna parte in modo indipendente. Questo permette di aumentare il contrasto solo di alcune zone, ad esempio confrontando il valore medio dei pixel della zona con il valore medio dei pixel dell'immagine intera.

Derivata dell'immagine

Consideriamo la coordinata x e definiamo la derivata nel caso discreto semplicemente come la differenza tra un campione e il successivo.

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \quad (4.4)$$

Data questa definizione possiamo applicare l'operatore derivata (che è lineare) alla derivata prima e ottenere la derivata seconda

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2 \cdot f(x) \quad (4.5)$$

Laplaciano

Usando la derivata seconda possiamo costruire l'operatore laplaciano, questo operatore è invariante per rotazione e permette di aumentare la nitidezza dell'immagine.

Definiamo il laplaciano come:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (4.6)$$

Possiamo ottenere un'immagine più nitida applicando la trasformazione

$$g(x, y) = f(x, y) + c \cdot \nabla^2 f(x, y) \quad (4.7)$$

Dove c è un parametro che normalmente vale ± 1 .

Nell'immagine si vede il risultato del laplaciano applicato all'immagine (non è stato fatto sharpening, è solo stato applicato l'operatore ∇^2).



Gradiente

Considerando la derivata prima possiamo definire il gradiente dell'immagine in un dato pixel. Il gradiente è un vettore con una sua direzione e modulo. Possiamo usare il gradiente per costruire filtri che mettano in evidenza i bordi dell'immagine, dato che il gradiente ha una direzione possiamo considerare solo alcune componenti per enfatizzare i bordi orizzontali, verticali od obliqui; Oppure possiamo sommare tutte le componenti per evidenziare tutti i bordi.

Il gradiente è definito come:



$$\nabla f = \frac{\partial f}{\partial x} \vec{i} + \frac{\partial f}{\partial y} \vec{j} \quad (4.8)$$

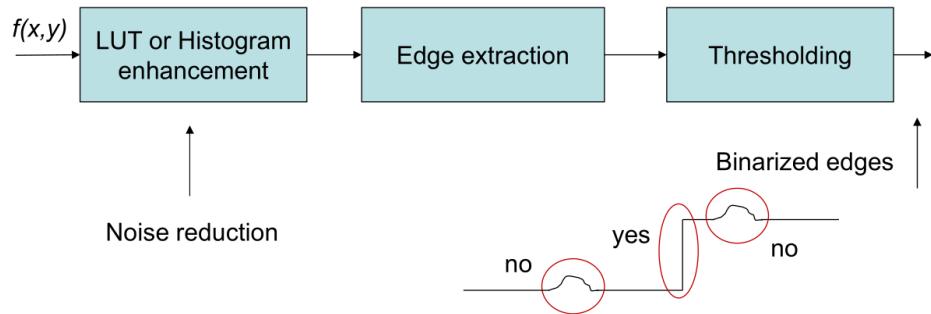
Ed è sempre ortogonale alla direzione dei bordi dell'immagine. Se consideriamo solo le componenti orizzontali o verticali possiamo costruire gli operatori di Sobel mostrati qui sotto.

$$S_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (4.9)$$

L'immagine di esempio è stata ottenuta sommando il risultato dei due operatori di Sobel che evidenziano rispettivamente bordi orizzontali e verticali.

4.4 Combinare gli strumenti

Tutti gli strumenti presentati possono essere applicati in cascata per ottenere l'immagine desiderata. Ogni trasformazione ha lo scopo di semplificare il lavoro degli operatori a valle.



Nell'immagine si vede una tipica pipeline di elaborazione di un'immagine, ovviamente ogni passaggio è optional e la sua applicazione o meno dipende dall'immagine di partenza, ma soprattutto dal risultato che si vuole ottenere; ricordando che l'enhancement è application-driven.

Chapter 5

Processing di immagini a colori

Tutti gli strumenti presentati nel capitolo precedente possono essere applicati con successo anche alle immagini a colori, occorre però prestare attenzione e scegliere con giudizio come trattare i diversi canali. In 2.2 abbiamo presentato diversi spazi colore, scegliere quello corretto per rappresentare i dati che vogliamo elaborare è fondamentale ai fini di ottenere un buon risultato.

5.1 Color slicing

Avere a disposizione un’immagine colorata ci permette di segmentare l’immagine non solamente considerando l’energia totale (valore del livello di grigio), ma anche in base al colore.

5.1.1 Segmentare col modello HSI

Il modo più ovvio per segmentare un’immagine è quello di passare al modello HSI, in questo modello infatti possiamo definire direttamente la tinta (H) che ci interessa e selezionare tutti i pixel che hanno una tinta vicina.

In certi casi può essere utile avere delle soglie anche sulla saturazione (S), in modo da avere degli algoritmi robusti quando si dovessero incontrare colori la cui tinta non è ben definita come i grigi. Mentre non si usa mai l’intensità che non porta alcuna informazione sul colore.

Quando si segmenta con questo spazio colore è necessario prestare attenzione alla definizione del modello stesso, siccome la hue è la distanza dal rosso in corrispondenza del rosso c’è un salto di continuità, questo significa che tinte percettivamente simili al rosso come il magenta avranno in realtà distanza massima da quest’ultimo. Se si devono segmentare oggetti rossi bisogna prendere in considerazione questa discontinuità.

5.1.2 Segmentare col modello RGB

Per segmentare con lo spazio colore RGB abbiamo bisogno di definire il colore medio degli oggetti che vogliamo evidenziare, il modo più comodo per fare questo è selezionare a mano un certo numero di pixel che sappiamo appartenere all'oggetto di interesse e computare poi la media.

Una volta stabilito il colore medio abbiamo bisogno di un modo per accettare o rifiutare altri colori. Nello spazio HSI questo era molto semplice, si definiva una soglia e si valutava se la distanza dalla tinta selezionata era maggiore o minore. Nel modello RGB invece possiamo definire una distanza geometrica tra il colore del pixel che stiamo analizzando e il colore target (questo perché i colori si trovano in un cubo).

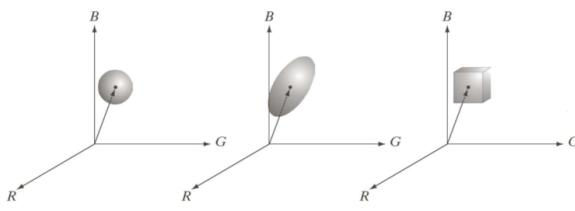
Abbiamo tipicamente tre alternative per decidere se accettare o meno un pixel:

- **cubo:** costruiamo un cubo intorno al colore target, tutti i pixel il cui colore cade dentro al cubetto saranno selezionati;
- **sfera:** si definisce una distanza di soglia, tutti i pixel il cui colore dista da quello target meno della soglia saranno accettati;
- **distanza di Mahalanobis:** è una distanza adattativa che tiene conto che alcune componenti di colore possano variare più di altre.

Distanza di Mahalanobis

Usare una sfera per segmentare prevede che ciascuna componente possa variare fino a una soglia massima uguale per tutte le componenti. Ci sono casi in cui la componente verde del target rimane molto stabile mentre la componente rossa varia molto, usare un sfera in questo caso porterà per forza a errori.

Quando si selezionano i pixel target si calcola anche una matrice di covarianza, questa ci permette di descrivere il comportamento di una componente rispetto alle altre e genera un'area di accettazione dei pixel a forma di ellissoide ruotato in modo da rispettare la distribuzione dei pixel che abbiamo selezionato per definire il colore target.



Nella figura sono rappresentate le diverse forme per definire la regione di accettazione dei pixel. Solo la distanza di Mahalanobis si adatta alla distribuzione dei pixel target.

Chapter 6

Enhancement con Fourier

6.1 Trasformata di Fourier continua

Una funzione arbitraria può essere riscritta come l'integrale in frequenza di funzioni goniometriche elementari (seni e coseni) opportunamente pesati. Questo permette di passare da un dominio spaziale, quello dei campioni del segnale, al dominio di Fourier.

Scriviamo la trasformata di Fourier (FT) indicata con F per un segnale monodimensionale. Considerando tempo t e frequenze ω continue, è possibile derivare la trasformata di una funzione $f(t)$ (scriviamo \mathfrak{F} per indicare che vogliamo calcolare la trasformata di una funzione):

$$\mathfrak{F}(f(t)) = F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i2\pi\omega t} dt \quad (6.1)$$

e la sua inversa:

$$\mathfrak{F}^{-1}(F(\omega)) = f(t) = \int_{-\infty}^{\infty} F(\omega)e^{i2\pi\omega t} d\omega \quad (6.2)$$

Siccome cambia solo il segno dell'esponenziale possiamo derivare la proprietà di **simmetria**: se $F(\omega)$ è la trasformata di $f(t)$ allora $f(-\omega)$ è la trasformata di $F(t)$.

6.1.1 Trasformate notevoli

Impulso

L'impulso è un segnale che vale 1 in zero e 0 in ogni altro istante. La sua trasformata sarà:

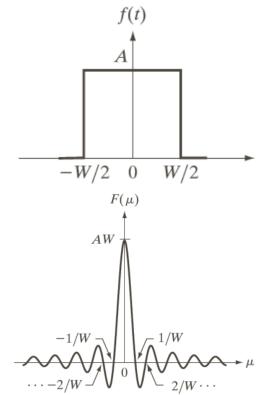
$$F(\omega) = \int_{-\infty}^{\infty} \delta(t)e^{-i2\pi\omega t} dt = e^{-i2\pi\omega 0} = 1 \quad (6.3)$$

Questo significa che l'impulso contiene tutte le frequenze, inoltre ricordando le proprietà della trasformata (6.1) la trasformata di una funzione costante è un impulso.

Gradino

Per una funzione a gradino, definita cioè da:

$$f(t) = \begin{cases} A & \text{if } -W/2 \leq t \leq W/2 \\ 0 & \text{otherwise} \end{cases} \quad (6.4)$$



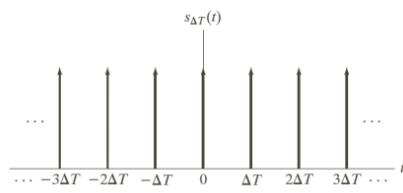
La trasformata è un seno cardinale, cioè

$$F(\omega) = AW \frac{\sin(\pi\omega W)}{\pi\omega W} = AW \text{sinc}(\omega W) \quad (6.5)$$

Coseno

La trasformata del coseno è una coppia di impulsi centrati nell'origine che si trovano uno in ω (la frequenza del coseno), l'altro in $-\omega$

Treno di impulsi



La trasformata di un treno di impulsi è nuovamente un treno di impulsi, se ogni impulso arriva dopo un tempo ΔT , gli impulsi della trasformata si trovano a una distanza di $\frac{1}{\Delta T}$

Funzioni pari e dispari

Per le proprietà degli esponenziali una funzione pari trasformata rimane pari e una dispari rimane dispari. Lo stesso vale per l'antitrasformata, che preserva la parità della funzione di partenza.

Funzioni traslate

Considero una funzione $f(t)$, la sua trasformata $F(\omega)$ e una traslazione di t_0 . Si ottiene

$$\begin{aligned}\mathfrak{F}(f(t - t_0)) &= \int_{-\infty}^{\infty} f(t - t_0) e^{-i2\pi\omega t} dt = \\ &= \int_{-\infty}^{\infty} f(s) e^{-i2\pi\omega(s+t_0)} ds = \\ &= e^{-i2\pi\omega t_0} \int_{-\infty}^{\infty} f(s) e^{-i2\pi\omega s} ds = \\ &= e^{-i2\pi\omega t_0} F(\omega)\end{aligned}\tag{6.6}$$

Il risultato è ottenuto con la sostituzione di variabile $s = t - t_0 \rightarrow t = s + t_0$.

6.1.2 Convoluzione

La convoluzione è una funzione binaria $*$ che si applica a due funzioni $f(t)$ e $h(t)$. La sua definizione è:

$$f(t) * h(t) = \int_{-\infty}^{\infty} f(\tau) \cdot h(t - \tau) d\tau\tag{6.7}$$

Ricordando il risultato appena ottenuto della trasformata di funzioni traslate possiamo calcolare la trasformata della convoluzione ottenendo:

$$\mathfrak{F}(f(t) * h(t)) = F(\omega) \cdot H(\omega)\tag{6.8}$$

Cioè la covoluzione diventa un prodotto nel dominio trasformato. Similmente la convoluzione nel dominio di Fourier diventa un prodotto nel dominio del tempo. Questo risultato prende il nome di **teorema della convoluzione**.

6.2 Sampling e DTFT

Quando acquisiamo digitalmente un segnale, che sia mono o bi dimensionale lo stiamo campionando, è come se lo osservassimo solo in certi momenti. Possiamo quindi vedere il segnale campionato come il segnale originale moltiplicato per un treno di impulsi. La sua trasformata sarà quindi la convoluzione delle trasformate. Chiamiamo questa trasformata DTFT cioè Trasformata di Fourier a tempi discreti e la indichiamo con \tilde{F} .

$$\tilde{F}(\omega) = F(\omega) * S(\omega) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F(\omega - \frac{n}{\Delta T})\tag{6.9}$$

Dove $S(\omega)$ è la trasformata del treno di impulsi che usiamo per il campionamento (sampling) e, di conseguenza, $\frac{1}{\Delta T}$ è la frequenza di campionamento. Otteniamo cioè infinite copie della trasformata della funzione originale, ciascuna traslata dalla precedente di $\frac{1}{\Delta T}$.

6.2.1 Teorema del campionamento

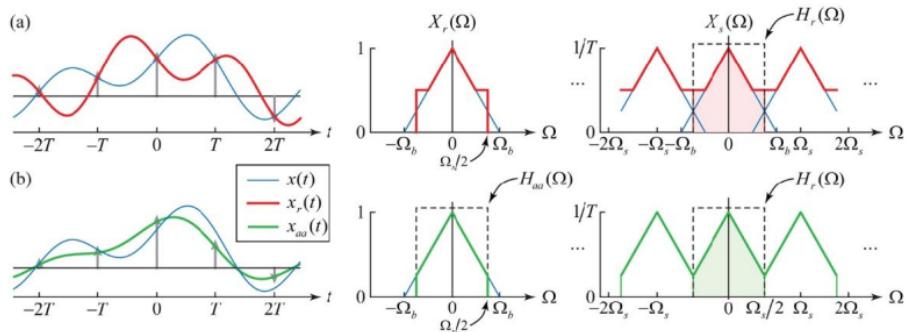
Per fare in modo di riuscire a ricostruire fedelmente il segnale di partenza devo fare in modo che le varie copie non si intersechino (sommandosi); significa cioè che la frequenza di campionamento deve essere almeno doppia della frequenza massima presente nel segnale. Questo risultato è noto come **teorema del campionamento** o di Nyquist-Shannon.

Purtroppo per garantire che le frequenze presenti in un segnale siano limitate questo deve avere durata infinita, dato che questo non può mai verificarsi saremo costretti ad accettare un minimo di sovrapposizione.

6.2.2 Aliasing

Quando le copie della trasformata si sovrappongono si verifica il fenomeno dell'aliasing, ovvero le frequenze più alte del segnale originale vanno ad alterare le frequenze più basse del segnale ricostruito. L'unica soluzione per poter ricostruire il segnale originale è quindi quella di aumentare la frequenza di campionamento.

Quando non è possibile modificare il processo di acquisizione del segnale si possono applicare (prima dell'acquisizione) dei filtri per limitare le frequenze più alte (filtri passa basso). Questi filtri sono detti anti aliasing, modificheranno il segnale, che non sarà più quello originale, ma spesso questa alterazione è molto minore dell'aliasing.



L'immagine mostra sengale originale (azzurro), ricostruito e trasformata in assenza (a) e presenza (b) di filtri anti aliasing, si vede bene come il segnale verde si avvicini di più all'originale del segnale rosso.

6.3 Trasformata di Fourier Discreta

Prima abbiamo considerato la trasformata di un segnale a tempi discreti, ora proviamo a discretizzare anche le frequenze e per fare questo consideriamo la periodicità della trasformata \tilde{F} e la campioniamo per un periodo ottenendo M campioni. Si può allora ottenere la Trasformata di Fourier discreta DFT (sia nel dominio dello spazio che della frequenza):

$$F(m) = \sum_{x=0}^{M-1} f(x)e^{-\frac{i2\pi mx}{M}} \quad (6.10)$$

e la sua inversa:

$$f(x) = \frac{1}{M} \sum_{m=0}^{M-1} F(m)e^{\frac{i2\pi mx}{M}} \quad (6.11)$$

Dove sia $x = 0, 1, \dots, M$ che $m = 0, 1, \dots, M$ sono intere e rappresentano rispettivamente gli istanti discreti di tempo e le frequenze discrete.

È facile mostrare che sia la trasformata che l'inversa sono periodiche di periodo M . È possibile interpretare la trasformata di Fourier come un cambio di base che ci permette di passare dal dominio spaziale al dominio temporale tramite una matrice di trasformazione (il kernel della trasformata).

6.4 Trasformata 2D continua

Possiamo estendere il concetto di trasformata Fourier al caso bidimensionale ottenendo le seguenti espressioni:

$$\mathfrak{F}(f(t, z)) = F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z) e^{-i2\pi(ut+vx)} dt dz \quad (6.12)$$

$$\mathfrak{F}^{-1}(F(u, v)) = f(t, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{i2\pi(ut+vx)} du dv \quad (6.13)$$

Notiamo che siccome le variabili sono indipendenti posso l'integrale è separabile, quindi è possibile integrare prima su una variabile poi sull'altra.

6.5 Sampling 2D

Possiamo ripetere i ragionamenti fatti nel caso monodimensionale quando passiamo al caso bidimensionale. In 2D per campionare non usiamo un treno di impulsi ma una matrice e possiamo ottenere tutti i risultati del caso 1D.

6.5.1 Teorema del campionamento

Possiamo essere sicuri di ricostruire fedelmente un segnale se lo campioniamo a una frequenza almeno doppia della frequenza massima presente nel segnale. Siccome usiamo per acquisire un sensore rettangolare che rappresenta la matrice di impulsi possiamo considerare separatamente frequenze verticali e orizzontali.

6.5.2 Aliasing 2D

Solo funzioni illimitate nel dominio dello spazio sono limitate nel dominio della frequenza, e viceversa, quando acquisiamo un'immagine o un video questo sarà per forza limitato nello spazio e nel tempo, questo porta ad avere un segnale a banda illimitata.

Per quanto detto prima la frequenza di acquisizione sarà sicuramente minore della frequenza massima presente nel segnale. Per limitare il fenomeno dell'aliasing sarà quindi necessario applicare dei filtri passa basso prima dell'acquisizione.

Aliasing e scalamento



Quando riduciamo le dimensioni dell'immagine non è sufficiente prendere un pixel ogni due od ogni tre, perché in questo modo si riduce la frequenza di campionamento senza ridurre le frequenze presenti nell'immagine, quindi

potrebbero presentarsi artefatti dovuti al rimpicciolimento. Per evitare questo fenomeno è necessario, prima di fare lo scalamento applicare un filtro di blurring che elimini le frequenze troppo alte dall'immagine. Nell'immagine si vede l'immagine originale seguita da un esempio di scalamento prima senza filtro e poi con filtro che elimina le Moirè

6.6 Trasformata 2D discreta

Vediamo ora l'effetto del campionamento sulla trasformata di Fourier:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-i2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (6.14)$$

e la sua inversa:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{i2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (6.15)$$

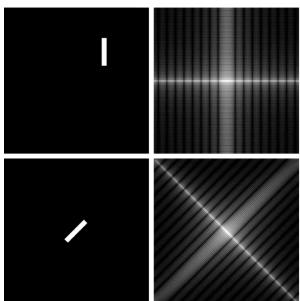
Dove la distanza spaziale tra due campioni orizzontali è ΔT e tra due campioni verticali è ΔZ , mentre la separazione tra due frequenze orizzontali e verticali è rispettivamente $\Delta u = \frac{1}{M\Delta T}$ e $\Delta v = \frac{1}{N\Delta Z}$

Modulo e fase

Come nel caso monodimensionale anche in 2D la trasformata di un **segnale reale** ha modulo pari e fase dispari.

Se proviamo a fare il plot della fase spesso ci troviamo di fronte un'immagine che pare di puro rumore, nonostante questo è la fase che trasporta le informazioni sulla forma dell'immagine, provando a ricostruire un'immagine solo con la fase si riesce ancora spesso a capire il soggetto, è impossibile invece con il modulo.

6.6.1 Traslazioni e rotazioni



La trasformata di Fourier è invariante per traslazione, mentre ruotare l'immagine significa ruotare la trasformata e ruotare la trasformata significa ruotare l'antitrasformata (l'immagine). Nell'esempio si vede prima il rettangolo bianco traslato in alto a destra, che non altera la trasformata, poi il rettangolo ruotato che ha come effetto la rotazione della trasformata.

6.6.2 Convoluzione 2D

In due dimensioni la convoluzione è definita come:

$$f(x, y) * h(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n) \quad (6.16)$$

Si può dimostrare il **teorema della convoluzione 2D** che afferma che cambiando dominio la convoluzione diventa un prodotto e viceversa, formalmente si ha che:

$$f(x, y) * h(x, y) \iff F(u, v)H(u, v) \quad (6.17)$$

$$f(x, y)h(x, y) \iff F(u, v) * H(u, v) \quad (6.18)$$

6.7 Filtraggio nel dominio della frequenza

L'idea di base è quella di passare al dominio della frequenza per poter moltiplicare la trasformata del segnale con la trasformata del filtro, poi tornare al dominio spaziale per ottenere il risultato.

6.7.1 Zero Padding

Siccome quando facciamo la trasformata di un segnale 2D di dimensione $N \times M$ otteniamo una trasformata con le stesse dimensioni e per moltiplicare due segnali o due trasformate è necessario che entrambi abbiano le stesse dimensioni dobbiamo fare in modo che i segnali di partenza abbiano la stessa dimensione. Per fare questo possiamo fare **zero padding**, cioè aggiungere alle funzioni (segnali) degli zeri, in modo che abbiano la stessa dimensione.

Aggiungere degli zeri alla fine di una funzione può però generare componenti di alta frequenza, dato che sarebbe come moltiplicare il segnale per una finestra rettangolare, che abbiamo visto avere un seno cardinale come trasformata (6.1.1); questo fenomeno si chiama **frequency leakage**.

Per mitigare il frequency leakage al posto che moltiplicare la funzione per un gradino si usano finestre più morbide, che vanno a zero senza salti netti; le finestre più usate sono Gaussiana, di Hann, di Hamming, di Barlett.

6.7.2 Progettare un filtro

Passare al dominio della frequenza ci permette di progettare i filtri in modo nuovo, concentrandoci sulle frequenze e i pattern che vogliamo eliminare dall'immagine.

Purtroppo quando operiamo nel dominio della frequenza dovremmo evitare di creare filtri ideali, dato che questi si traducono in filtri di dimensione infinita nel dominio dello spazio e portano al frequency leakage.

Inoltre quando progettiamo un filtro non vogliamo andare a deformare l'immagine di partenza, dato che l'informazione sulle forme è portata dalla fase della trasformata vorremmo che il filtro non alteri la fase.

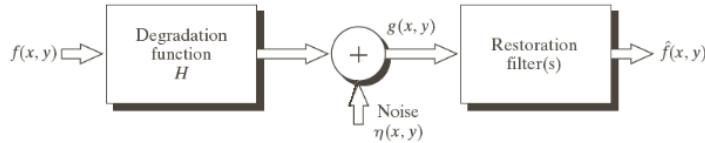
In generale quando progettiamo un filtro per un'immagine $f(x, y)$ di dimensioni $M \times N$ dovremmo seguire i seguenti passaggi, che nella maggior parte delle applicazioni portano a dei risultati accettabili:

1. fare zero padding dell'immagine arrivando alla dimensione $P \times Q = 2M \times 2N$
2. moltiplicare l'immagine per $(-1)^{x+y}$ in modo da centrare la trasformata nell'origine
3. calcolare la trasformata $F(u, v)$
4. generare il filtro $H(u, v)$ di dimensioni $P \times Q$ centrato in $P/2, Q/2$
5. calcolare la trasformata del risultato come $G(u, v) = H(u, v)F(u, v)$
6. calcolare la parte reale dell'antitrasformata e trasstrarla nell'origine moltiplicando di nuovo per $(-1)^{x+y}$
7. ritagliare l'immagine alla dimensione $M \times N$

Chapter 7

Image Restoration

L’image restoration è un processo oggettivo, che passa attraverso la modellazione del danneggiamento subito dall’immagine per poi correggerla. Si appoggia a delle metriche per stabilire se è stato raggiunto l’ottimo nella curva di riparazione.



In figura si vede un tipico processo di restoration: si parte dall’immagine originale ($f(x, y)$), perfetta, si passa attraverso un primo filtro che danneggia l’immagine (ad esempio la macchina non ha messo correttamente a fuoco), assumeremo che questo filtro sia lineare e spazio invarianti. Durante il processo di acquisizione poi potrebbe essere sommato all’immagine del rumore (tipicamente dovuto a imperfezioni del sensore) e si arriva all’immagine degradata ($g(x, y)$). All’immagine rovinata si possono applicare uno o più filtri di restoration per ottenere una stima dell’immagine originale ($\hat{f}(x, y)$).

Siccome Assumiamo che i filtri che rovinano l’immagine siano lineari e spazio invarianti possiamo scrivere

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y) \quad (7.1)$$

$$G(u, v) = H(u, v) \cdot F(u, v) + N(u, v) \quad (7.2)$$

Dove le lettere maiuscole indicano la trasformata di Fourier della funzione con la minuscola.

7.1 Rumore

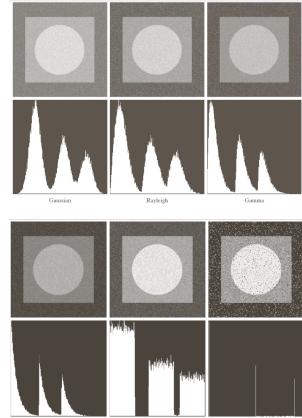
È un danno all’immagine dovuto all’ambiente e alla tipologia di acquisizione dell’immagine. Possiamo studiarne le proprietà attraverso lo spettro in frequenza (ad esempio spettro piatto per il rumore bianco) e in generale assumiamo sia spazio invariante.

7.1.1 Stimare il rumore

Oltre al rumore bianco l’immagine può essere affetta da rumori distribuiti in molti modi. Per stimare i parametri del rumore si possono adottare diverse tecniche.

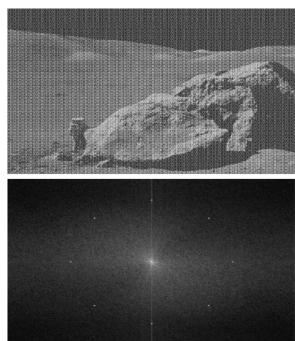
Rumore spazio invariante

Siamo interessati a stimare la PDF del rumore, per fare questo si può considerare una zona dell’immagine che sappiamo avere uno stesso livello di grigio e studiare l’istogramma di quella zona, per osservare la distribuzione del rumore. Nelle immagini vediamo alcuni esempi di come differenti distribuzioni del rumore danneggiano immagini sintetizzate. Anche in questi esempi molto semplici è molto difficile capire il tipo di rumore senza guardare l’istogramma.



Capire la PDF del rumore è importante dato che sono stati ingegnerizzati filtri apositi che, sfruttando le conoscenze a priori della PDF, correggono meglio i danni provocati da un certo tipo di rumore.

Rumore spazio dipendente



Normalmente questo è dovuto al particolare processo di acquisizione, in questo caso ci concentriamo sul rumore periodico, che è un particolare tipo di rumore spazio dipendente. Il rumore periodico può essere individuato da dei picchi visibili nella trasformata di Fourier dell’immagine, è in questo dominio che progettiamo i filtri per rimuovere il rumore periodico. Ogni componente del rumore genera 2 picchi simmetrici nella trasformata.

7.2 Rumore spazio invariante

Progettiamo dei **filtri nel dominio spaziale** per ridurre il rumore presente in un'immagine. Questi filtri sono adatti quando nell'immagine è presente rumore additivo. Il modello di degradazione è:

$$g(x, y) = f(x, y) + \eta(x, y) \quad (7.3)$$

$$G(u, v) = F(u, v) + N(u, v) \quad (7.4)$$

Nel progettare questi filtri prenderemo in considerazione un pixel e i suoi vicini, questi possono essere disposti a croce, gli 8 tutti attorno oppure possiamo considerare un quadrato più grande, chiameremo S_{xy} il vicinato (**neighbourhood**) del pixel a coordinate (x, y)

7.2.1 Filtri semplici

Filtri che operano una media

Aritmetica

$$\hat{f} = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t) \quad (7.5)$$

Il rumore è ridotto grazie al blurring, è l'unico filtro lineare (solo per questo ha senso studiare la risposta in frequenza);

Geometrica

$$\hat{f} = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}} \quad (7.6)$$

Riduce il rumore ma preserva più dettagli rispetto alla media aritmetica;

Armonica

$$\hat{f} = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s, t)}} \quad (7.7)$$

Adatta a ridurre il rumore gaussiano e di tipo sale, fallisce con il rumore pepe;

Contrarmonica

$$\hat{f} = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^Q} \quad (7.8)$$

Q è l'ordine della media, se $Q > 0$ adatta per rumore pepe, se $Q < 0$ adatta per rumore sale, se $Q = 0$ o $Q = -1$ si riade in media aritmetica o armonica rispettivamente.

Filtri basati sull'ordinamento dei livelli d'intensità

Questi filtri sono adatti alla rimozione degli outlier.

Mediana

$$\hat{f} = \text{median}_{(s,t) \in S_{xy}} g(s, t) \quad (7.9)$$

Adatto a rimuovere rumore impulsivo, può rovinare i bori sottili;

Max/Min

$$\hat{f} = \max_{(s,t) \in S_{xy}} \min g(s, t) \quad (7.10)$$

Adatti a rimuovere rispettivamente rumore pepe e sale;

Punto medio

$$\hat{f} = \frac{\max_{(s,t) \in S_{xy}} g(s, t) + \min_{(s,t) \in S_{xy}} g(s, t)}{2} \quad (7.11)$$

Adatto a rimuovere rumore casuale gaussiano o uniforme;

Alfa trimmed

$$\hat{f} = \frac{1}{mn - d} \sum_{(s,t) \in S_{xy}} g'(s, t) \quad (7.12)$$

Prima di calcolare la media vengono scartati i $d/2$ valori più piccoli e i $d/2$ valori più grandi. È adatto per rimuovere rumore sale e pepe e gaussiano assieme.

7.2.2 Filtri adattativi

Sono filtri che adattano il loro comportamento in base alle caratteristiche locali dell'immagine, tipicamente si considera una finestra $m \times n$ che rappresenta il vicinato di un pixel e si confrontano le caratteristiche della finestra con quelle dell'immagine intera per dosare il comportamento del filtro.

Media adattativa

$$\hat{f}(x, y) = g(x, y) - \frac{\sigma_\eta^2}{\sigma_L^2} [g(x, y) - m_L] \quad (7.13)$$

Dove σ_η è la varianza del rumore, σ_L è la varianza locale e m_L è la media locale. La media locale rappresenta la luminanza nella zona che stiamo considerando, mentre la varianza locale il contrasto.

A seconda del rapporto tra la varianza del rumore e la varianza locale il filtro si comporta in modo diverso:

- $\sigma_\eta = 0$ non c'è rumore quindi $\hat{f}(x, y) = g(x, y) = f(x, y)$;
- $\sigma_L \gg \sigma_\eta$ molto contrasto, ci sono dei bordi, quindi non bisogna filtrare troppo per non rovinare i gettagli $\hat{f}(x, y) \approx g(x, y)$;
- $\sigma_L = \sigma_\eta$ la finestra ha le stesse caratteristiche dell'immagine, il filtro si comporta esattamente come la media;
- $\sigma_\eta > \sigma_L$ si forza $\sigma_\eta = \sigma_L$ per non ottenere dei valori negativi, questo potrebbe rendere necessario riscalare i valori dell'immagine per non perdere della dinamica dell'immagine.

Affinché questo filtro possa funzionare correttamente è essenziale avere una stima accurata della varianza del rumore.

Mediana adattativa

Questo filtro ha lo scopo di preservare i bordi, rimuovere il rumore impulsivo, fare smoothing del rumore random senza allargare o assottigliare i bordi.

```

function ADAPTATIVEMEDIAN( )
     $S_{xy} \leftarrow S_{min}$ 
    while  $S_{xy} \leq S_{max}$  do
         $A_1 \leftarrow z_{med} - z_{min}$ 
         $A_2 \leftarrow z_{med} - z_{max}$ 
        if  $A_1 > 0$  and  $A_2 < 0$  then
             $B_1 \leftarrow z_{xy} - z_{min}$ 
             $B_2 \leftarrow z_{xy} - z_{max}$ 
            if  $B_1 > 0$  and  $B_2 < 0$  then
                return  $z_{xy}$ 
            else
                return  $z_{med}$ 
            end if
        else
             $S_{xy} \leftarrow \text{INCREMENTNEIGHBOURHOOD}( )$ 
        end if
    end while
    return  $z_{med}$ 
end function

```

Dove:

- S_{xy} : Area del vicinato del pixel (x, y) ;
- z_{min} : Intensità minima nell'area considerata;
- z_{max} : Intensità massima nell'area considerata;
- z_{med} : Intensità media nell'area considerata;
- z_{xy} : Intensità del pixel (x, y) ;
- S_{min} : Area di partenza del vicinato;
- S_{max} : Massima area del vicinato consentita.

Come si vede questo filtro è in grado di modificare, entro certi limiti, la dimensione della finestra di vicinato. Il filtro calcola la mediana, se questa corrisponde al minimo o al massimo dell'intensità si aumenta la dimensione della finestra, altrimenti si decide se restituire la mediana o il valore del pixel. Si restituisce il valore del pixel se questo ha le stesse caratteristiche che abbiamo richiesto per la mediana (non è né un minimo né un massimo di intensità).

7.3 Rumore spazio dipendente

Progettiamo dei filtri **nel dominio della frequenza** per ridurre il rumore additivo spazio dipendente. Per descrivere il rumore osserveremo quali sono le sue componenti nella trasformata di Fourier e cereremo di isolare l'immagine rispetto al rumore.

7.3.1 Notch Filter

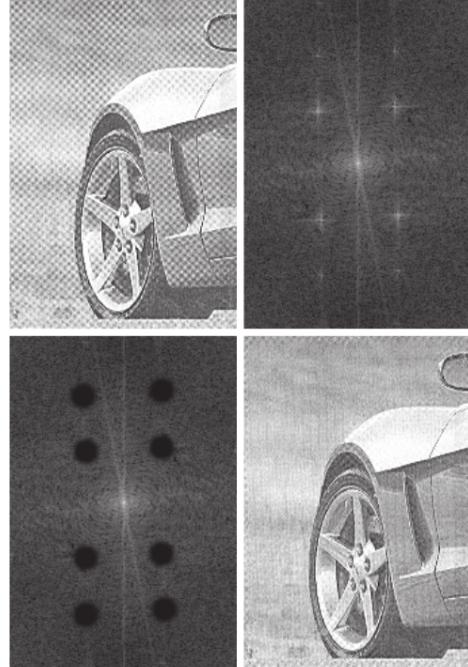
Il rumore periodico si individua facendo la trasformata e cercando dei picchi nella stessa. Ogni componente genera 2 picchi simmetrici rispetto all'origine. Per rimuovere il rumore periodico si può usare un filtro **notch** H_{NR} . Questo è costituito dal prodotto di filtri passa alto centrati sui picchi di rumore.

$$H_{NR}(u, v) = \prod_{k=1}^Q H_k(u, v) H_{-k}(u, v) \quad (7.14)$$

Dove $H_k(u, v)$ è il filtro passa alto centrato in u_k, v_k e $H_{-k}(u, v)$ è il simmetrico rispetto all'origine.

Ciascun H_k può essere costruito utilizzando diverse forme (ideale, gaussiano, butterworth, ecc...) indipendentemente dagli altri filtri passa alto, l'importante è che $H_k(u, v)$ e $H_{-k}(u, v)$ abbiano la stessa forma.

Per fare image enhancement è sufficiente centrare i filtri passa alto nei picchi di rumore visibili nella trasformata di Fourier e filtrare l'immagine utilizzando il filtro notch ottenuto componendo i passa alto. Il risultato che si ottiene è spesso più soddisfacente rispetto all'immagine che si otterrebbe con dei filtri di blurring. Inoltre se i picchi di rumore sono sufficientemente distanti dalle frequenze che caratterizzano l'immagine il filtro notch non rovina i dettagli, neppure quelli più fini. L'immagine mostra la situazione originale e l'applicazione dei filtri passa alto in corrispondenza dei picchi di rumore, come si vede il risultato è soddisfacente.



Optimum Notch

Costruiamo a partire dal filtro notch un filtro adattativo che pesi l'effetto del filtro notch a seconda delle caratteristiche locali dell'immagine. Come sempre consideriamo il vicinato S_{xy} del pixel (x, y) e ricaviamo un filtro che minimizzi la varianza locale dell'immagine ricostruita $\hat{f}(x, y)$. Per costruire l'optimum notch abbiamo bisogno di:

- filtro che isoli il rumore: $H_{NP} = 1 - H_{NR}$
- rumore stimato: $\eta(x, y) = \mathfrak{F}^{-1}\{N(u, v)\} = \mathfrak{F}^{-1}\{H_{NP}(u, v)G(u, v)\}$
- immagine ricostruita: $\hat{f} = g(x, y) - w(x, y)\eta(x, y)$

Dove pesiamo il rumore con $w(x, y)$ perché non è il rumore esatto, ma solo una stima.

Siamo pronti per calcolare la varianza locale dell'immagine ricostruita; chiamiamo il valor medio di una misura come la misura stessa con una bar-

retta sopra, ad esempio $\bar{\eta}$ è il valor medio del rumore nella finestra considerata.

$$\begin{aligned}\sigma^2(x, y) &= \frac{1}{mn} \sum_{(r,c) \in S_{xy}} [\hat{f}(r, c) - \bar{\hat{f}}]^2 = \\ &= \frac{1}{mn} \sum_{(r,c) \in S_{xy}} \left\{ [g(c, r) - w(c, r)\eta(c, r)] - [\bar{g} - \bar{w}\bar{\eta}] \right\}^2 = \quad \text{def. di } \hat{f} \\ &= \frac{1}{mn} \sum_{(r,c) \in S_{xy}} \left\{ [g(c, r) - w(c, r)\eta(c, r)] - [\bar{g} - w(x, y)\bar{\eta}] \right\}^2 \quad \bar{w} = w(x, y)\end{aligned}\tag{7.15}$$

Nell'ultimo passaggio approssimo i pesi della finestra al peso del pixel centrale ($w(c, r) = w(x, y)$), quindi la media sarà il peso del pixel centrale.

Possiamo derivare la varianza in funzione di $w(x, y)$ e imponerla a 0, si ottiene il vincolo su $w(x, y)$:

$$w(x, y) = \frac{\bar{g} \cdot \bar{\eta} - \bar{g} \cdot \bar{\eta}}{\bar{\eta}^2 - \bar{\eta}^2} \tag{7.16}$$

7.4 Danni lineari spazio invarianti

Le immagini che prendiamo in considerazione ora sono affette da una degradazione lineare spazio invariante, cioè l'immagine è stata trasformata da un filtro lineare spazio invariante. Questo significa che possiamo modellare l'immagine acquisita come:

$$g(x, y) = h(x, y) * f(x, y) \tag{7.17}$$

$$G(u, v) = H(u, v) \cdot F(u, v) \tag{7.18}$$

Per ripristinare l'immagine quindi è necessario operare una deconvoluzione nel dominio spaziale o una semplice divisione nel dominio della frequenza, data la semplicità si sceglie sempre il secondo. La deconvoluzione di filtri non lineari o spazio dipendenti è molto più complessa o, in certi casi, addirittura irrisolvibile.

7.4.1 Filtro inverso

L'idea più semplice per recuperare un'immagine danneggiata da un filtro lineare spazio invariante è quella di costruire il filtro inverso. Il filtro inverso, se esiste ed è stabile permette di ricostruire l'immagine di partenza. Vediamo ora alcuni metodi per risalire al filtro che ha danneggiato l'immagine e quindi per poter stimare il filtro inverso.

Stima per osservazione

Per stimare il filtro che ha rovinato l'immagine $g(x, y)$ possiamo isolare una piccola area $g_s(x, y)$ dell'immagine che sia facile da ricostruire, ripararla empiricamente (anche con un semplice programma di fotoritocco) in modo da ottenere $\hat{f}_s(x, y)$; ora possiamo ottenere il filtro che ha degradato la piccola area come:

$$H_s(u, v) = \frac{G_s(u, v)}{\hat{F}_s(u, v)} \quad (7.19)$$

Possiamo poi costruire il filtro $H(u, v)$ a partire da $H_s(u, v)$ per scalamento.

Stima sperimentale

Dato che stiamo considerando filtri lineari spazio invarianti questi possono essere caratterizzati dalla risposta all'impulso. Per studiare la risposta all'impulso del sistema ottico acquisisco un singolo punto luminoso (molto intenso per evitare il rumore) nelle stesse condizioni in cui ho acquisito l'immagine che voglio recuperare. Dato che la DFT dell'impulso è nota ed è una costante A ricavare il filtro è molto semplice:

$$H(u, v) = \frac{G(u, v)}{A} \quad (7.20)$$

Stima con modello

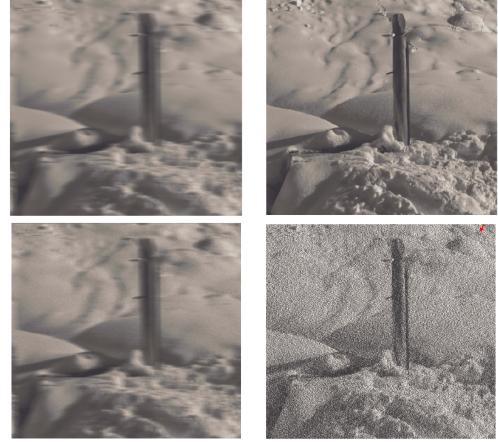
Esistono diversi modelli che descrivono alcuni fenomeni fisici comuni, come la turbolenza atmosferica, la presenza di nebbia o foschia, ecc... oppure delle imprecisioni durante l'acquisizione dell'immagine come il movimento della camera. Se si ha a disposizione il modello di degradazione per progettare il filtro che ha rovinato l'immagine è sufficiente impostare correttamente i parametri del modello per ricostruire filtro e filtro inverso.

Problemi del filtro inverso

Questo approccio funziona se l'immagine acquisita $G(u, v)$ non è affetta da altri tipi di rumore, infatti in generale l'immagine che otteniamo dopo il ripristino sarà data da:

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} = F(u, v) + \frac{N(u, v)}{H(u, v)} \quad (7.21)$$

Dove la frazione indica la deconvoluzione tra il rumore e il filtro che abbiamo usato per ripristinare l'immagine. Dei valori prossimi a zero nel filtro possono aumentare molto la componente rumorosa e l'immagine che si ottiene risulta essere fortemente rovinata, nella figura si vede un esempio di immagine ricostruita con il filtro inverso che è stato modellato seguendo il movimento della camera. La prima coppia di immagini non è affetta da rumore, mentre la seconda sì e anche se nell'immagine mossa il rumore si nota poco in quella ricostruita predomina sull'immagine.



7.4.2 Wiener filter

Il problema del filtro inverso è stato studiatissimo e sono state proposte molte soluzioni che riducessero gli effetti mostrati sopra. Il filtro di Wiener migliora le prestazioni del filtro inverso minimizzando lo scarto quadratico medio (MSE Mean Square Error). Vogliamo minimizzare

$$E\{(f - \hat{f})^2\} \quad (7.22)$$

Se assumiamo oltre al filtro lineare spazio invariante che il rumore sia decorrelato dall'immagine e che almeno uno tra immagine e rumore abbia media nulla si può ricavare che il minimo del MSE è dato da un'immagine ricostruita con questa forma:

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + S_\eta(u, v)/S_f(u, v)} \right] G(u, v) \quad (7.23)$$

Dove:

- $H(u, v)$: filtro che ha danneggiato l'immagine
- $H^*(u, v)$: complesso coniugato di $H(u, v)$
- $|H(u, v)|^2 = H^*(u, v)H(u, v)$
- $S_\eta(u, v) = |N(u, v)|^2$: spettro di potenza del rumore
- $S_f(u, v) = |F(u, v)|^2$: spettro di potenza dell'immagine originale

Al denominatore vediamo l'inverso del rapporto segnale rumore, infatti $S_\eta(u, v)/S_f(u, v) = 1/SNR$. Questo ci dice che il filtro di Wiener diventa il filtro inverso se a una data frequenza il rapporto segnale rumore tende a infinito, mentre cancella le frequenze dominate dal rumore.

È possibile semplificare questo filtro assumendo che il rumore sia bianco, quindi $S_\eta(u, v)/S_f(u, v) \rightarrow \sigma^2/S_f(u, v)$, inoltre dato che in generale lo spettro dell'immagine originale non è noto assumiamo che il rapporto segnale rumore sia costante e lo consideriamo un parametro K del filtro. Il filtro di Wiener diventa quindi:

$$H(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + K} = \frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + K} \quad (7.24)$$

7.4.3 CLSF

Il filtro di Wiener necessita dello spettro dell'immagine e del rumore, o almeno di una loro stima, dato che queste informazioni non sempre sono disponibili sono stati sviluppati altri filtri, come il Constrained Least Squares Filter. Questo filtro parte dall'idea che il filtro inverso fallisce perché il rumore che ha solitamente spettro piatto viene aumentato moltissimo specie alle alte frequenze dove il filtro ha valori molto bassi (ricordiamo che il filtro si trova a denominatore). Per impedire questo effetto si cerca un risultato più smooth minimizzando il laplaciano:

$$\sum_{x=0}^M \sum_{y=0}^N [\nabla^2 f(x, y)] \quad (7.25)$$

La soluzione che minimizza il laplaciano è:

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \gamma |P(u, v)|^2} \right] G(u, v) \quad (7.26)$$

Dove:

- $P(u, v)$ è la trasformata dell'operatore laplaciano: $P(u, v) = \mathfrak{F}\left(\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}\right)$
- γ è un parametro che si trova **iterativamente** in modo da rispettare:

$$\|g(x, y) - h(x, y) * \hat{f}(x, y)\|^2 = \|\eta(x, y)\|^2 \quad (7.27)$$

7.4.4 Filtro con media geometrica

È possibile generalizzare il filtro di Wiener in uno che ricordi la media geometrica. Questo filtro è molto utile dato che controllando i suoi 2 parametri si possono implementare filtri molto diversi utili in tanti ambiti. Il filtro generale genera un'immagine ricostruita in questo modo:

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2} \right]^\alpha \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \beta \left[\frac{S_\eta(u, v)}{S_f(u, v)} \right]} \right]^{1-\alpha} G(u, v) \quad (7.28)$$

Dove $\alpha \geq 0$ e $\beta \geq 0$ sono i parametri e a seconda del loro valore si hanno diversi comportamenti:

- $\alpha = 1$: filtro inverso;
- $\alpha = 0$: filtro di Wiener parametrico ($\beta = 1$ Wiener standard);
- $\alpha = \frac{1}{2}$: si calcola la media geometrica delle quantità tra parentesi
- $\alpha = \frac{1}{2}$ e $\beta = 1$: filtro equalizzatore dello spettro;
- $\alpha < \frac{1}{2}$: il filtro si comporta più come Wiener;
- $\alpha > \frac{1}{2}$: il filtro si comporta più come filtro inverso.

7.4.5 Esempi

Le immagini sono disposte $(\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix})$ e mostrano l'applicazione di diversi filtri a un'immagine rovinata da turbolenza atmosferica e affetta da rumore additivo:

1. Immagine originale;
2. Filtro inverso;
3. Filtro Wiener;
4. Filtro CLSF.



Chapter 8

Low Level Vision

La low level vision si basa su algoritmi che ricalcano proprietà matematiche per estrarre informazioni utili dalle immagini. Questa branca della computer vision raramente fa uso di intelligenza artificiale e si adatta bene a contesti semplici e controllati come la visione industriale.

La low level vision ha come obiettivo quello di estrarre delle regioni di interesse dalle immagini, evidenziare bordi e punti di interesse di un'immagine.

8.1 Segmentation

Vogliamo suddividere l'immagine in aree di interesse, cioè creare una maschera che ritagli delle aree dell'immagine semanticamente differenti. La maschera più semplice a cui possiamo pensare ha solo 2 livelli e permette di ottenere un'immagine binarizzata in cui distinguiamo la ragione di interesse dallo sfondo.

8.1.1 Thresholding

È uno dei metodi più intuitivi per fare segmentazione di un'immagine, è molto utilizzato date le se buone proprietà e la pochissima complessità computazionale. Il thresholding permette di dividere i pixel in regioni basandosi sulle proprietà dei pixel stessi.

La forma più semplice di thresholding consiste nel decidere una soglia T e dividere i pixel con intensità minore di T da quelli con intensità maggiore; questo approccio è detto **thresholding globale**. Se la soglia cambia parliamo invece di thresholding variabile, questo si divide in **thresholding locale** se T dipende dalle caratteristiche del vicinato del pixel in esame, oppure **thresholding adattativo** se T dipende dalla posizione nell'immagine.

Lo scopo del thresholding è quello di risolvere un problema decisionale statistico. Vogliamo minimizzare il numero di pixel etichettati nel modo sbagliato.

Questo significa conoscere la distribuzione dei pixel che rappresentano lo sfondo e la distribuzione dei pixel che rappresentano il soggetto. Stimare le PDF è un problema tutt'altro che semplice, e anche se assumiamo siano distribuzioni normali trovare la soglia ccorretta non è semplice.

Thresholding globale semplice

Questo metodo è adatto quando le distribuzioni di pixel sono chiaramente bimodali e la separazione è relativamente facile. La soglia T viene trovata iterativamente raffinando delle stime. L'algoritmo seguito per trovare T è il seguente:

```
function BASICTHRESHOLDING( )
     $\Delta T \leftarrow \infty$ 
     $T \leftarrow T_{init}$ 
    while  $\Delta T < \Delta T_{min}$  do
         $G_1 \leftarrow \text{PIXELLESS}(T)$ 
         $G_2 \leftarrow \text{PIXELGREATER}(T)$ 
         $m_1 \leftarrow \text{INTENSITYMEAN}(G_1)$ 
         $m_2 \leftarrow \text{INTENSITYMEAN}(G_2)$ 
         $T_{new} \leftarrow \frac{m_1 + m_2}{2}$ 
         $\Delta T \leftarrow |T - T_{new}|$ 
         $T \leftarrow T_{new}$ 
    end while
end function
```

Thresholding di Otsu

Al posto di minimizzare i pixel etichettati in modo sbagliato ci concentriamo sul massimizzare la differenza tra le due classi di pixel che separiamo. Questo consiste nel massimizzare la varianza inter-classe (tra le classi).

Il metodo di Otsu è vantaggioso anche perché permette di lavorare unicamente con l'istogramma, che è facile da ottenere e permette di costruire un algoritmo molto efficiente dal punto di vista computazionale.

Avendo L livelli di intensità, un'immagine di dimensione $N \times M$ e chiamando n_i il numero di pixel aventi intensità i possiamo la probabilità che un pixel dell'immagine abbia intensità i cioè $p_i = n_i/MN$.

Se scegliamo una soglia k possiamo ora derivare la probabilità che un certo pixel appartenga alla prima (c_1) o alla seconda (c_2) classe:

$$P_1(k) = \sum_{i=0}^k p_i \quad (8.1)$$

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1 \quad (8.2)$$

Possiamo calcolare l'intensità media dei pixel in c_1 , in c_2 e la media cumulativa fino al livello k come:

$$m_1(k) = \sum_{i=0}^k i P(i/c_1) = \frac{1}{P_1} \sum_{i=0}^k i p_i \quad (8.3)$$

$$m_2(k) = \sum_{i=k+1}^{L-1} i P(i/c_2) = \frac{1}{P_2} \sum_{i=k+1}^{L-1} i p_i \quad (8.4)$$

$$m(k) = \sum_{i=0}^k i p_i \quad (8.5)$$

Definiamo ora la varianza inter-classe come :

$$\begin{aligned} \sigma_G^2 &= P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 = \\ &= P_1 P_2 (m_1 - m_2) = \\ &= \frac{(m_g P_1 - m)^2}{P_1 P_2} \end{aligned} \quad (8.6)$$

Possiamo ricavare il valore ottimo per la soglia k^* che permette di massimizzare la varianza

$$\sigma_B^2(k^*) = \max_{0 \leq k \leq L-1} \sigma_B^2(k) = \max_{0 \leq k \leq L-1} \left(\frac{[m_g P_1(k) - m(k)]^2}{P_1(k) P_2(k)} \right) \quad (8.7)$$

Migliorare il thresholding globale

Per migliorare il thresholding globale possiamo usare diverse tecniche:

1. smoothing dell'immagine;
2. usare i bordi per individuare la soglia corretta.

La prima tecnica consiste nell'applicare un filtro passa basso, facendo **blurring dell'immagine** possiamo:

- ridurre il rumore;
- scegliere la dimensione dei dettagli da evidenziare;
- passare da un istogramma confuso a uno bimodale.

La seconda tecnica si basa sul fatto che tipicamente siamo interessati a separare il soggetto dallo sfondo, se il soggetto ha dei bordi sufficientemente marcati possiamo evidenziarli (ad esempio con il laplaciano) e trovare la soglia di threshold ottima solo sui pixel di bordo. Per fare questo costruiamo una maschera a partire dal laplaciano e la usiamo per selezionare i bordi dell'immagine di partenza. Una volta trovata la soglia ottima possiamo applicarla a tutta l'immagine.

Threshold variabile

Compensa un'illuminazione non uniforme definendo per ogni pixel una soglia attraverso strumenti statistici.

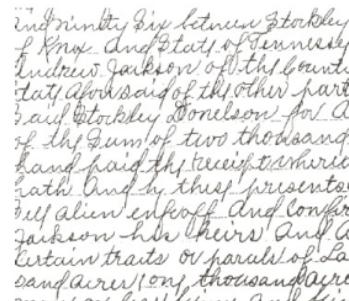
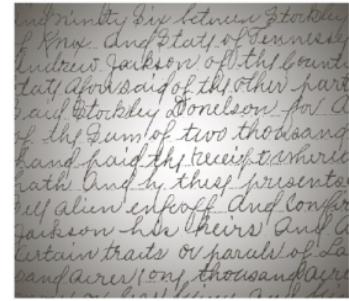
Esempi di definizione di soglia sono:

$$\begin{aligned} T_{xy} &= a\sigma xy + bm_{xy} \\ T_{xy} &= a\sigma xy + bm \end{aligned}$$

Un esempio più complicato di statistica per scegliere la soglia è quello della **media mobile**; questa consiste nello scandire l'immagine per riga e risulta particolarmente utile per il riconoscimento del testo. Definiamo la media al passo $k+1$ come

$$\begin{aligned} m(k+1) &= \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i = \\ &= m(k) \frac{z_{k+i} - z_{k-n}}{n} \end{aligned} \quad (8.8)$$

Dove z_i è l'intensità del pixel i e $m(1) = z_1/n$. Nella figura vediamo l'immagine originale, quella binarizzata col metodo di Otsu e infine quella con la media mobile.



8.1.2 Region growing

Costruiamo le regioni di interesse a partire da un punto che sicuramente vi appartiene (**seed**) e facciamo crescere la regione aggiungendo pixel che soddisfano una determinata proprietà Q .

La parte più complessa di questa procedura è quella di individuare i seed, per fare questo può essere necessario esaminare tutti i pixel dell'immagine per cercare candidati adatti.

Una volta individuati i seed l'algoritmo di region growing è relativamente semplice: per ogni seed individuato si invoca la seguente funzione ricorsiva:

```
function REGIONGROWING(seed)
    neighbors  $\leftarrow$  8-CONNECTED(seed)
    for all pixel in neighbors do
        if SATISFY( $Q$ ,pixel) then
            pixel.valid  $\leftarrow$  true
            REGIONGROWING(pixel)
        end if
    end for
end function
```

Al termine di ogni chiamata si ottiene una nuova regione di pixel marcati come validi, a questa regione si può poi dare un'etichetta per costruire una maschera.

8.1.3 Region splitting and merging

Questa procedura permette di dividere l'immagine in regioni ricorsivamente più piccole e di raggruppare le regioni adiacenti che rispettano una data proprietà Q . La struttura che descrive l'immagine al termine della procedura è un **quad-tree**; il quad-tree è una struttura molto compatta di rappresentare l'informazione e consente di decidere a priori la grana con cui ricostruire le regini di interesse.

La procedura si divide in due parti, la prima si occupa di dividere le regioni R in sotto-regioni, la seconda di unire le sotto-regioni se entrambe rispettano una proprietà.

```
function SPLIT(R)
    if (not SATISFY( $Q$ ,R)) and (R  $>$   $R_{min}$ ) then
         $R_1, R_2, R_3, R_4 \leftarrow$  DIVIDE(R)
        SPLIT( $R_1$ ), SPLIT( $R_2$ ), SPLIT( $R_3$ ), SPLIT( $R_4$ )
    end if
end function
```

```

function MERGE(region-set R)
  do
    old_R  $\leftarrow R$ 
    for all  $R_a \in R$  do
      for all  $R_b \in \text{ADJACENT}(R_a)$  do
        if SATISFAY( $Q, R_a \cup R_b$ ) then
           $R_{ab} = R_a \cup R_b$ 
           $R \leftarrow R / \{R_a, R_b\} \cup R_{ab}$ 
        end if
      end for
    end for
    while  $R \neq old_R$ 
  end function

```

8.2 Feature Extraction

Lo scopo della feature extraction è quello di estrarre caratteristiche utili da un’immagine. Non esiste una definizione precisa e condivisa di cosa sia una feature; in generale le caratteristiche che cerchiamo dipendono dall’applicazione e dall’elaborazione che seguirà l’estrazione.

A prescindere da cosa consideriamo feature il processo di estrazione si divide in due fasi principali:

1. feature detection;
2. feature description.

Vogliamo cioè non solo individuare le caratteristiche utili, ma essere in grado di descriverle per trovare similitudini e differenze tra feature estratte da diversi punti dell’immagine o tra immagini diverse.

8.2.1 Edge Detection

Sviluppiamo dei metodi per identificare i bordi che siano più robusti e affidabili della semplice derivata (gradiente o laplaciano).

Marr-Hildreth edge detector

Questo metodo è molto semplice, non assicura risultati ottimi ma permette di migliorare le prestazioni del detector rispetto al semplice laplaciano. L’idea è che il rumore presente in un’immagine può generare falsi positivi, possiamo rimuovere parte di questo disturbo con un filtro gaussiano passa basso. Il

filtro ha anche lo scopo di selezionare la dimensione minima dei bordi che vogliamo tenere in considerazione. La procedura per estrarre i bordi è la seguente:

1. filtrare l'immagine con un filtro gaussiano passa basso $n \times n$ ($G(x, y)$);
2. applicare l'operatore laplaciano (∇^2) al risultato;
3. trovare i punti di zero crossing.

Possiamo scrivere l'immagine ottenuta al passo 2 come

$$g(x, y) = \nabla^2 [G(x, y) * f(x, y)] \quad (8.9)$$

Il punto chiave di questa procedura è quello considerare bordi solo i pixel che si trovano a metà tra due pixel con segno opposto (zero crossing), questo unito al fatto di poter scegliere una soglia minima per il modulo della differenza dei pixel di segno opposto permette di ottenere buoni risultati etichettando bordi fini e precisi.

Canny edge detector

È l'algoritmo per l'estrazione dei bordi più complesso, ma che da risultati migliori. Questo metodo ha 3 obiettivi:

- **basso tasso d'errore:** tutti i bordi devono essere identificati;
- **bordi ben localizzati:** la distanza tra il punto etichettato come bordo e il bordo reale deve essere la minima possibile
- **single point response:** per ogni punto del bordo deve essere etichettato un solo punto, cioè non si devono etichettare i massimi locali nelle prossimità di un bordo.

L'algoritmo è diviso in 4 fasi principali:

1. Smoothing dell'immagine con un filtro gaussiano;
2. Calcolo del gradiente con angolo e modulo;
3. applicazione della Non Maxima Suppression al modulo del gradiente;
4. Doppia soglia di threshold per rilevare e unire i bordi.

Passo 1: lo smoothing dell’immagine serve a scegliere la dimensione dei dettagli che ci interessano e permette di rimuovere il rumore che potrebbe portare a falsi positivi.

Passo 2: Calcoliamo il gradiente di ciasun pixel, questo passaggio può essere fatto ad esempio con l’operatore di Sobel che ci fornisce la componente verticale e orizzontale del gradiente, da questa siamo poi in grado di risalire a modulo e direzione.

Passo 3: Costruiamo una prima approssimazione dei bordi, chiamiamo questa immagine $g_N(x, y)$ e inizializziamo tutti i suoi pixel a 0. Consideriamo i bordi come paralleli a 4 direzioni (verticali, orizzontali e le 2 diagonali), non ci interessa il verso ma solo la direzione. Dato il vettore gradiente del pixel (x, y) in f chiamiamo d_k la direzione (tra le 4 definite prima) più prossima all’angolo del gradiente e $K = \|\nabla f_s\|$ il modulo del gradiente. Se K è minore di almeno uno fra i $\|\nabla f_s\|$ dei picel adiacenti a (x, y) lungo la direzione d_k allora $g(x, y) = 0$ altrimenti $g_N(x, y) = K$.

↗	↗	↗	↗	↗	↗	↗	↗	↗
1	2	3	5	3	2	1	0	
1	2	3	4	3	2	1	0	
↗	2	3	↗	8	4↑	4↑	4↑	4↑
7↑	7↑	8↑	9↑	9↑	8↑	7↑	6↑	
6↑	6↑	7↑	7↑	7↑	6↑	8	5↑	
4↑	4↑	4↑	4↑	6↑	5↑	9	7↑	
3↑	4↑	4↑	4↑	5↑	4↑	8	6↑	
2↑	4↑	4↑	4↑	4↑	5↑	11↑	9↑	
1↑	1↑	0↑	1↑	1↑	6	8↑	4↑	

In questo modo individuiamo un solo punto per ogni bordo, man mano che il gradiente aumenta avvicinandosi al bordo stesso.

Passo 4 hysteresis thresholding: Consideriamo 2 soglie T_H e T_L in rapporto variabile (solitamente tra 2 : 1 e 3 : 1) e costruiamo 2 immagini bordi forti g_{NH} e bordi deboli g_{NL} utilizzando rispettivamente prima e seconda soglia.

In g_{NH} , avendo usato una soglia più alta, ci saranno meno pixel. Aggiorno g_{NL} in modo che contenga solo i pixel che mancano a g_{NH} in questo modo: $g_{NL} = g_{NL} - g_{NH}$. I Pixel in g_{NH} vengono marcati tutti come validi, poi per riempire i buchi di questa maschera procedo come segue:

1. etichetto tutti i pixel di g_{NH} come non visitati;
2. per ogni pixel (x, y) etichettato come non visitato:
 - (a) etichetto come bordo valido tutti i pixel in g_{NL} adiacenti a (x, y) ;
 - (b) etichetto (x, y) come visitato;

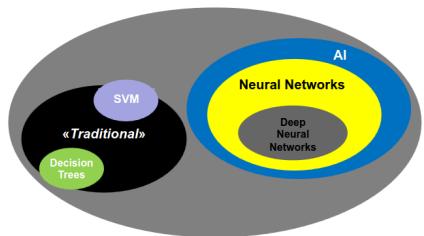
Alla fine della procedura tutti i pixel etichettati come validi saranno considerati bordi.

8.2.2 Line Detection

Chapter 9

AI per la visione artificiale

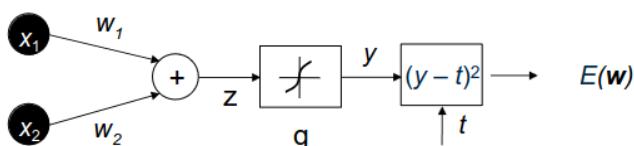
L'intelligenza artificiale oggi viene usata in moltissime applicazioni di computer vision e processing di immagini; Le architetture si basano su reti neurali convoluzionali (CNN). In questa trattazione vedremo come si sono evolute le reti neurali partendo dall'idea di percettrone fino ad arrivare alle reti profonde. Per quanto riguarda il metodo di apprendimento ci concentreremo in particolare sull'apprendimento supervisionato.



9.1 Percettrone

Il percettrone è il precursore della moderna AI e cerca di risolvere dei problemi di **classificazione binaria**. Un problema di questo tipo prevede di riuscire ad etichettare correttamente istanze appartenenti a due classi differenti (esempio: è un cane o un gatto)

9.1.1 Funzionamento



come parametri i pesi w_i degli input. La funzione calcolata è la combinazione

Nello schema si vede il funzionamento del percettrone, questo calcola una funzione che ha come variabili gli input che riceve e

lineare degli input pesati e questa serve come input per una funzione di attivazione non lineare, ma differenziabile (tipicamente una **sigmoide**).

Aggiornamento dei pesi

Una volta che è stata calcolata la risposta del percettrone si può calcolare l'errore commesso come scarto quadratico ($E(\vec{w}) = (y - t)^2$) dove t è la classe reale dell'oggetto e può valere 0 o 1. Si può quindi derivare l'errore rispetto al singolo peso con la chain-rule:

$$\frac{\partial E(\vec{w})}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_i} \quad (9.1)$$

Quindi possiamo aggiornare i pesi considerando il gradiente della funzione errore. Questo metodo si chiama **stochastic gradient descent**. Passando dall'iterazione k alla $k + 1$ possiamo scrivere:

$$\vec{w}^{k+1} = \vec{w}^k - \eta \nabla_{\vec{w}} E(\vec{w}) \quad (9.2)$$

Dove il parametro η è detto **learning rate**, è sempre minore di 1 e influisce su quanto l'errore pesa sull'aggiornamento dei parametri del percettrone.

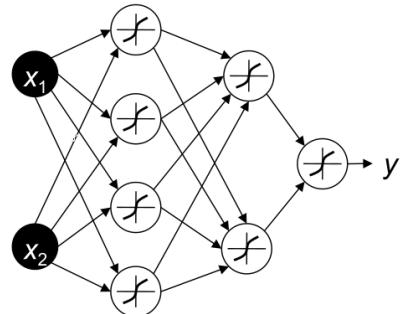
Training

Si inizializzano i pesi casualmente, si calcola l'output per un dato input e si valuta l'errore. Si aggiornano i pesi e si procede in questo modo fino a quando non si esauriscono tutti gli esempi nella classe di training. L'algoritmo

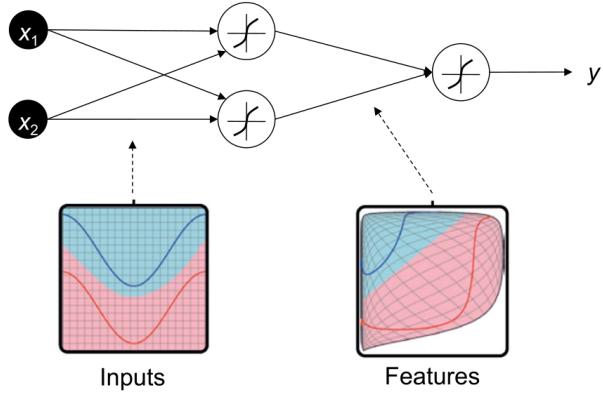
- converge se le classi sono linearmente separabili
- da come risultato un output binario

9.2 Multilayer networks

Il singolo neurone permette di classificare classi linearmente separabili. Per poter aumentare la generalità del classificatore si passa alle FCNs (Multi-layer Fully Connected Networks). In queste reti sono presenti uno o più livelli nascosti (hidden), non ci sono cicli e vi è un layer di output.



9.2.1 Funzionamento



I livelli hidden hanno lo scopo di rendere le features linearmente separabili in modo che il livello di output possa operare come descritto prima per il perceptron. L'operazione dei livelli nascosti è quindi una trasformazione dallo spazio degli input allo **spazio latente** dove si spera che la classificazione sia più semplice.

Più sono i livelli hidden più è complessa la funzione che si può apprendere per separare le classi.

Complessità

La complessità è data dal numero di parametri che devono essere appresi. Per ogni livello il numero di pesi sarà il prodotto tra il numero di input di ogni neurone (+ 1 per il bias) e il numero di neuroni del layer ovvero il numero di output del layer stesso. Formalmente:

$$N_{par} = n_{out} \cdot (1 + n_i n) \quad (9.3)$$

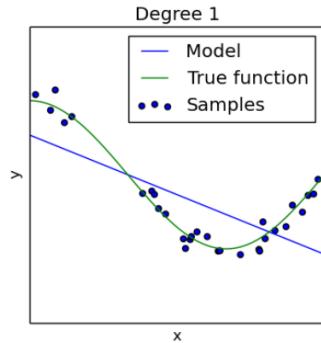
9.2.2 Overfitting

Man mano che si aggiungono livelli e si aumenta la complessità della rete i le funzioni che possono essere apprese diventano sempre più complesse, questo può portare al fenomeno dell'**overfitting**; che si verifica quando la rete apprende troppo bene i parametri per minimizzare l'errore sul set di training e perde quindi di generalità. Questo fenomeno è evidenziato da un grosso divario tra i punteggi realizzati sul set di training e sul set di validation.

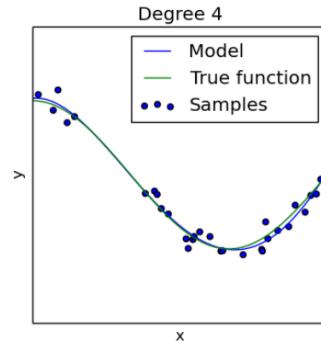
Esempio

Consideriamo di dover interpolare dei punti con una curva, l'immagine mostra come al variare del numero di gradi di libertà abbiamo

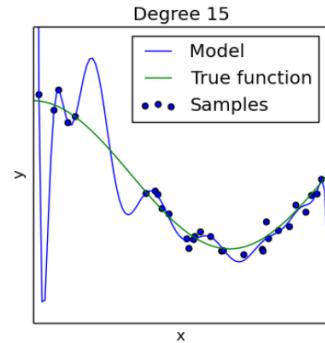
- underfitting



- good fitting



- overfitting



Nell'ultimo caso stiamo fittando il rumore e non i dati.

Regularization

Per impedire l'overfitting si ricorre alla regolarizzazione. Dall'esempio si può immaginare che per far fluttuare così rapidamente la funzione che fa overfitting è necessario che i suoi coefficienti abbiano un “grande” valore assoluto. Si cerca di penalizzare le soluzioni con pesi elevati, minimizzando

$$J(\vec{w}) = E(\vec{w}) + \lambda R(\vec{w}) = E(\vec{w}) + \lambda \|\vec{w}\|^2 \quad (9.4)$$

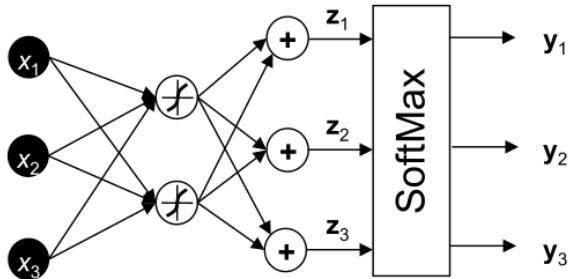
Dove $R(\vec{w})$ è un fattore di normalizzazione che solitamente si traduce nella norma al quadrato le vettore dei pesi e λ è un parametro che normalmente è qualche ordine di grandezza più piccolo del learning rate η .

9.2.3 Multiclass Classification

Quando abbiamo più di due classi usiamo un livello di output che contiene tanti neuroni quante sono le classi da riconoscere, in questo modo quando incontriamo un oggetto appartenente a una classe si attiva solo il neurone corrispondente e gli altri dovrebbero rimanere spenti. Nella pratica però è raro che si attivi un solo neurone al 100% e gli altri rimangano spenti. Nei casi fortunati avremo un neurone molto attivo e gli altri poco, non è detto però che la somma degli output dei vari neuroni sommi a 1.

Livello Soft-Max

Se la somma delle attivazioni dei neuroni non è 1 non possiamo trattarla come una distribuzione di probabilità (PDF). Per risolvere questo problema si aggiunge un livello di normalizzazione che scala gli output dei neuroni



e ci assicura di poter trattare il risultato come una PDF. Per trasformare il risultato dei neuroni in una PDF si usa la formula:

$$y_i = \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{k=1}^n e^{z_k}} \quad \text{con } i = 1, 2, \dots, n \quad (9.5)$$

9.2.4 Loss function

Fino ad ora abbiamo calcolato la funzione d'errore come scarto quadratico (MSE). Questa politica però ha dei problemi, dato che nella funzione del calcolo dell'errore rientrano anche gli errori dovuti alle classi sbagliate che tendono a spingere \vec{y} verso 0; questo può portare a una distribuzione meno mirata dell'errore. Definiamo allora la **Cross Entropy Loss Function** come

$$L(y, t) = - \sum_i t_i \log(y_i) = - \log(y_{i*}) \quad (9.6)$$

Dove y_{i*} è la risposta del neurone associato alla classe corretta e si ottiene dalla sommatoria dato che tutti gli altri t_i valgono 0.

Il risultato di usare la Cross Entropy è:

- l'errore e quindi il gradiente propagato attraverso la rete deriva solo dalla classe corretta $i*$ e ne consegue un aggiornamento dei pesi può mirato;
- all'inizio del training i gradienti sono più grandi e questo porta alla convergenza più rapida dei pesi.

9.3 Riconoscere immagini

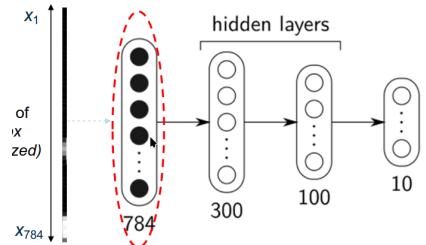
Ora che abbiamo costruito un sistema in grado di apprendere proviamo a insegnargli a riconoscere le immagini. Il primo tentativo in tal senso è stato fatto per riconoscere le cifre da 0 a 9.

9.3.1 LeNet 300

Il primo compito di classificazione delle immagini consisteva nel riconoscimento di cifre. Queste erano salvate come immagini 28×28 , in scala di grigi con profondità di 8 bit e contenute in un dataset chiamato **MNIST**

Architettura

La LeNet 300 è una FCN prevede un livello di input costituito da 784 neuroni, uno per ogni pixel dell'immagine. Ci sono poi 2 livelli hidden costituiti da 300 e 100 neuroni rispettivamente, infine un livello di output costituito da 10 neuroni. La funzione di attivazione del livello di input e dei livelli hidden è una sigmoide, mentre per il livello di output si usa una Soft-Max.



Pro e contro

La LeNet migliora le sue performance man mano che si aggiungono livelli hidden e raggiunge, nella configurazione presentata, un tasso di errore del 3.05%.

Aggiungere altri livelli però produrrebbe un grosso incremento della complessità e richiederebbe set di immagini molto grandi per evitare overfitting. Inoltre una FCN non si adatta bene allo scalare delle dimensioni dell'immagine, infatti per ogni pixel è necessario un neurone sul layer di input, questo significa moltissimi neuroni appena si utilizzano immagini di risoluzione maggiore.

9.3.2 Un approccio diverso

Le FCNs prevedono una serializzazione dell'immagine, ogni pixel è mandato a un neurone. Questo però non è come un'immagine dovrebbe essere trattata. Le immagini infatti sono caratterizzate da elevata correlazione spaziale, se si serializzano si conserva la correlazione in una sola dimensione.

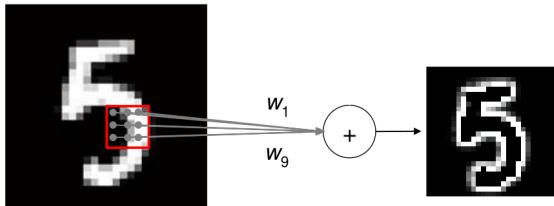
I filtri, d'altro canto, si comportano così bene sulle immagini perché mantengono la correlazione tra i dati.

Viene quindi naturale pensare di poter applicare l'apprendimento automatico alla costruzione di filtri per riuscire a estrarre le features che più ci interessano per raggiungere il nostro scopo. I pesi dei filtri saranno proprio i parametri che la rete deve apprendere al fine di estrarre le features importanti. Nascono in questo modo le **reti convoluzionali**.

Chapter 10

Convolutional Neural Network

10.1 Neurone convoluzionale



È un neurone che applica un filtro all'immagine ed estrae la tessa feature da tutta l'immagine. Per fare questo opera come un normale neurone che calcola una combinazione lineare degli input, ma gli input avranno dei vincoli spaziali, in particolare saranno una matrice quadrata di pixel che con la convoluzione viene filtrata dai parametri del neurone. I parametri saranno appresi attraverso la backpropagation e non si dovranno più progettare a mano i filtri per estrarre le features.

Per fare in modo che dopo l'operazione di filtraggio il risultato abbia le stesse dimensioni dell'immagine originale si fa zero padding.

Se stiamo trattando immagini colorate trattiamo ogni canale in modo indipendente, ad esempio se costruiamo dei filtri 3×3 ogni neurone apprenderà 27 parametri, 9 per ogni filtro. I risultati del filtraggio dei tre canali è poi sommato per generare la feature map.

10.2 Complessità del layer convoluzionale

Il numero di parametri che devono essere appresi per ogni livello convoluzionale sarà dato dal numero di feature map che vogliamo ottenere n_{out} (numero di filtri nel livello) per la dimensione del filtro + 1 per il numero di canali in ingresso; questi saranno 3 per le immagini a colori 1 per le immagini in scala

di grigi e il numero di feature map estratte al livello precedente se stiamo considerando un generico layer all'interno della rete. In formule possiamo scrivere:

$$N_{par} = n_{out} \cdot (1 + F^2) \cdot n_{ch} \quad (10.1)$$

Dove F rappresenta il lato del filtro.

10.2.1 Complessità della rete

Ogni neurone convoluzionale restituisce come output un'immagine delle stesse dimensioni dell'immagine originale. Dopo che abbiamo estratto tutte le feature che ci interessano quindi dobbiamo tornare a una FCN per poter fare la classificazione. Per fare questo semplicemente serializziamo ogni feature map e la concateniamo con le altre.

Il collo di bottiglia diventa quindi il layer di passaggio tra CNN e FCN, dato che al posto di avere un neurone per ogni pixel dell'immagine originale avremo un neurone per ogni pixel di ogni feature map, e queste possono essere anche molte.

10.2.2 Ridurre la complessità

Per ridurre la complessità possiamo trattare le feature map come immagini e sottocampionarle. Per ridurre le dimensioni delle feature map abbiamo due alternative.

MaxPooling

Il MaxPooling consiste nel dividere l'immagine in celle da 4 pixel e per ogni cella prendere solo il valore maggiore. In questo modo dimezziamo la dimensione dell'immagine e dato che prendiamo solo il valore più grande e non facciamo la media eseguiamo anche un'operazione di **Non Maxima Suppression**, si è visto che questo aggiunge robustezza alle traslazioni.



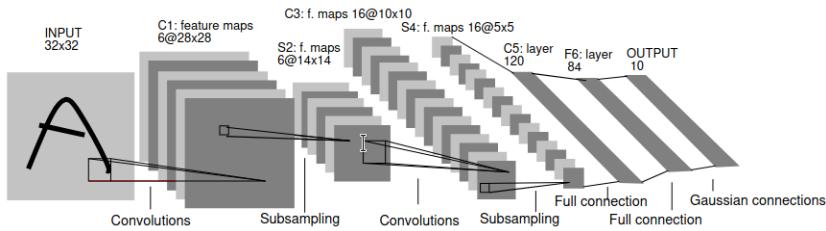
Convolve and Stride

In questo caso ogni convoluzione ha un passo maggiore di 1, questo significa che il filtro non si sposta sull'immagine pixel per pixel ma fa dei salti.

Questa tecnica non prevede la Non Maxima Suppression, ma è parte integrante del processo di filtraggio e non deve essere aggiunta come layer intermedio tra 2 layer di neuroni. Questo rende la tecnica del convolve and stride più flessibile e di fatto ha soppiantato il MaxPooling nelle moderne architetture.

10.3 LeNet5

Presentiamo l'architettura della LeNet5 che prevede 2 livelli convoluzionali da 6 e 16 neuroni, intervallati da 2 livelli di MaxPooling, si passa poi 3 livelli FC.



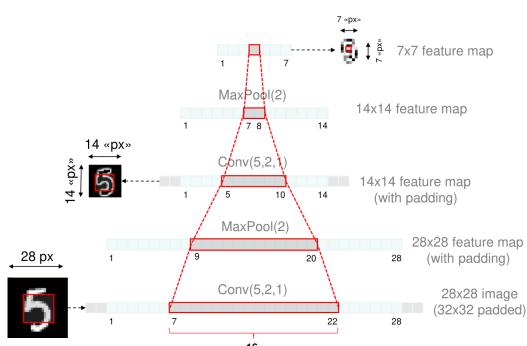
Il tasso d'errore della LeNet5 con l'architettura in figura è di 0.95%, questo miglioramento nelle performance è accompagnato anche da una riduzione della complessità della rete rispetto alla LeNet300, in particolare il numero di parametri necessari alla LeNet5 è meno di $\frac{1}{3}$ dei parametri necessari alla LeNet300.

10.4 Receptive Field

Il receptive field di una feature indica quanti pixel dell'immagine originale sono stati usati per generare la feature. Ad esempio un filtro 3×3 ha un receptive field di 9 pixel.

Nell'immagine si vede una rappresentazione 2D del receptive field della LeNet5. Notiamo che:

- i livelli di MaxPooling sono quelli che più di tutti influiscono sul receptive field, raddoppiandolo;
- i filtri aumentano la dimensione del receptive field per effetto del padding.



Che si usi il MaxPooling oppure il convolve and stride il subsampling dell'immagine consente di aumentare il receptive field. Questo è molto importante perché vogliamo che ogni feature (che al passando dall'ultimo livello convoluzionale al primo fully connected è costituita da un pixel) trasporti più informazione possibile.

10.5 Da fully connected a fully convolutional

10.5.1 Riconoscimento del testo

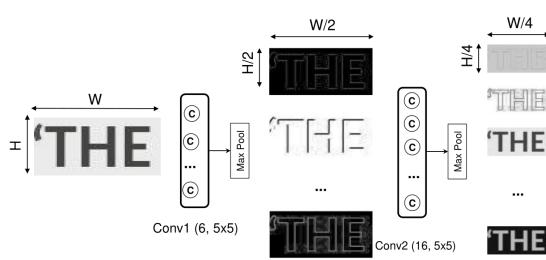
Ora che abbiamo progettato una rete in grado di raggiungere elevata accuratezza nel riconoscimento dei singoli caratteri vorremmo poter leggere il testo scritto su in immagine. Per fare questo abbiamo bisogno di individuare i caratteri prima di riconoscerli. L'approccio naif consiste nel:

- aggiungere una classe di background che non corrisponde a nessun carattere;
- aggiungere una classe che corrisponde all'intersezione di 2 caratteri, anche questa non corrisponde a nessun carattere;
- fare zigzag per l'immagine con passo di 1 pixel per individuare tutti i caratteri.

Questo approccio è molto inefficiente, dato che ogni pixel viene elaborato molte volte, tutte le elaborazioni sono indipendenti e la maggior parte di queste sono ridondanti o non portano informazione.

10.5.2 Sliding Window

Al posto che ritagliare l'immagine in quadrati della stessa dimensione delle immagini di training posso usare delle finestre di dimensione arbitraria, tanto i livelli convoluzionali si comportano come filtri e possono essere applicati a qualsiasi immagine.



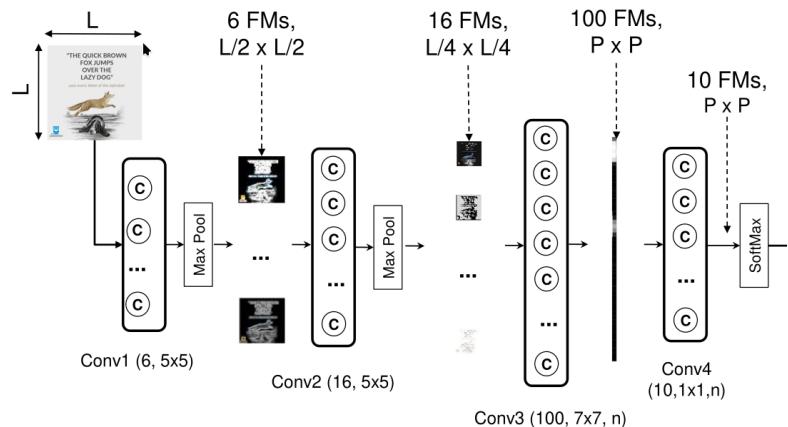
Nella figura si vede l'effetto dei neuroni convoluzionali che permettono di estrarre le feature per le quali sono stati addestrati da un immagine qualsiasi. Si vede anche l'effetto del MaxPooling che dopo ogni livello dimezza la dimensione delle features.

Rimane però un problema, dato che la rete termina con dei livelli fully connected questi hanno bisogno di un vettore di input di dimensione fissata e non possono adattarsi alle dimensioni della finestra arbitraria che abbiamo scelto.

10.5.3 Equivalente convoluzionale

Sostituiamo quindi i livelli fully connected con dei livelli convoluzionali. Consideriamo il primo livello FC, questo riceve in input un certo numero di features map (FM) di dimensione fissa. Per ogni FM avrà un peso per ogni pixel. Possiamo quindi trasformare questo neurone in un filtro semplicemente trattandolo come tale senza bisogno di cambiare i pesi. I neuroni convoluzionali ottenuti trasformando un layer FC sono detti **narrow**. Anche il livello di output può essere sostituito con un livello convoluzionale collegato poi alla funzione Soft-Max.

Una volta fatte queste sostituzioni siamo pronti per applicare la rete fully convolutional (CNN) all'intera immagine



Dove:

$$P = 1 + \frac{L - (f_{dim} \cdot 2^{n_{pools}})}{4} \quad (10.2)$$

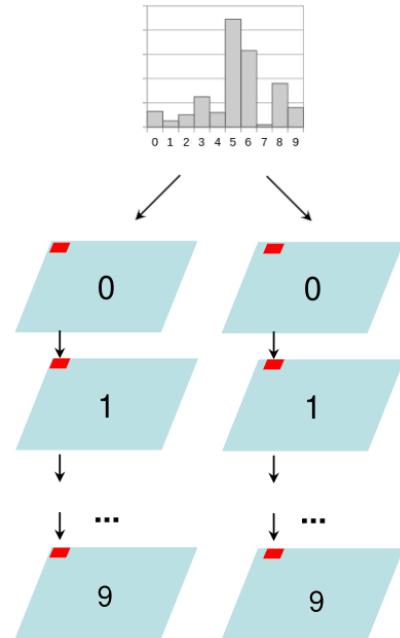
e f_{dim} è la dimensione del filtro narrow del primo livello FC tradotto (in questo caso la dimensione è 7), e n_{pools} è lo stride (In questa rete quindi si combinano convolve and stride e MaxPooling).

10.5.4 Interpretare i risultati

DA RISCRIVERE

Ora che abbiamo ottenuto delle feature map dobbiamo interpretarle. Per fare questo possiamo riutilizzare il classificatore FC che avevamo prima convertito in convoluzionale e passare a questo dei vettori di lunghezza 10, ciascun vettore conterrà il pixel i, j di ciascuna delle 10 FM che abbiamo estratto con la rete CNN. Ogni vettore può quindi essere classificato e fornirci in output un carattere. Nel caso della LeNet5 si ottengono 10 FM 19×19 , saremo quindi in grado di riconoscere al massimo $19 \times 19 = 361$ caratteri.

Il risultato di questa prima interpretazione potrebbe essere ancora poco soddisfacente, potrebbero infatti esserci dei doppiioni e delle lettere sbagliate. A valle di questo passaggio quindi si usano dei language model e altre reti per raffinare il risultato da un punto di vista semantico.



Chapter 11

Advanced Learning

Nei capitoli precedenti (9.1.1) abbiamo visto come minimizzare una certa funzione di loss usando il gradiente

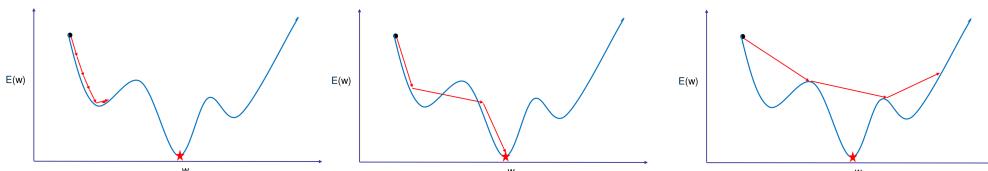
$$\vec{w}^{k+1} = \vec{w}^k - \eta \nabla_{\vec{w}} E(\vec{w}) \quad (11.1)$$

Ci domandiamo ora come rendere efficiente il training, sia modificando i parametri dell'equazione appena mostrata sia modificando proprio la strategia di learning.

11.1 Learning rate

La scelta del learning rate corretto è fondamentale per avere una rete che converga al minimo globale dell'errore, infatti se abbiamo learning rate

- troppo piccolo finiremo incastriati in minimi locali;
- corretto riusciamo a individuare il minimo globale;
- troppo grande la rete non converge a un minimo.



L'idea è che all'inizio dell'allenamento la rete si strovi a grande distanza dal minimo globale, man mano che vengono appresi i pesi ci si avvicina e gli step verso il minimo si fanno più piccoli. Per sfruttare questa idea si può modificare il learning rate durante l'apprendimento.

11.1.1 Decadimento del learning rate

Per modificare il learning rate possiamo seguire diverse politiche, in questo paragrafo ci concentriamo sulle più semplici che sono monotone decrescenti, ma esistono altri modi di far variare il learning rate che non sono monotoni e possono riuscire a non incastrarsi in minimi locali.

Step decay

Dopo in certo numero di iterazioni (**epoch**) il learning rate viene moltiplicato per un valore < 1 in modo da rendere il suo grafico un decadimento esponenziale a gradini. Le epoch in cui avviene la riduzione del learning rate sono dette **milestones**. Formalmente a ogni milestones il learning rate viene aggiornato in questo modo:

$$\eta = \eta_0 \cdot e^{-kt} \quad (11.2)$$

dove η_0 è il learning rate iniziale, k è il numero della milestones e t è un parametro.

Time decay

Dopo ogni epoca il learning rate iniziale viene diviso per il numero di epoch opportunamente pesato. In questo caso il grafico del learning rate è un ramo di iperbole. Formalmente

$$\eta = \frac{\eta_0}{1 + k \cdot t} \quad (11.3)$$

dove η_0 è il learning rate iniziale, k è il numero dell'epoca attuale e t è un parametro.

Exponential decay

Come lo step decay, ma l'aggiornamento viene fatto dopo ogni epoca. Formalmente

$$\eta = \eta_0 \cdot e^{-kt} \quad (11.4)$$

dove η_0 è il learning rate iniziale, k è il numero dell'epoca attuale e t è un parametro.

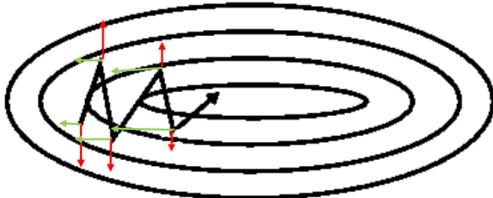
11.2 Stochastic gradient descent

Lo stochastic gradient descent SGD presenta dei problemi quando incontriamo una valle, infatti se osserviamo la situazione dell'immagine è facile convincersi che mentre le frecce verdi rimarranno costanti e orientate verso il minimo le frecce rosse invertono il loro segno ogni volta, questo provoca una discesa a zigzag dell'errore (poco efficiente). Quando si verifica lo zigzag diciamo che il **gradiente è rumoroso** in quella dimensione, dato che cambia tutte le volte di segno.



11.2.1 Aggiunta della quantità di moto

Per limitare il rumore di alcune componenti del gradiente si può ricorrere a delle similitudini con la fisica. Una pallina lanciata nel punto 1 farà poche oscillazioni ampie, per poi accelerare maggiormente verso il minimo. Esattamente come mostrato nell'immagine.



Per replicare questo comportamento si aggiunge una sorta di conservazione della quantità di moto (in inglese momentum) che si oppone all'oscillare del segno delle componenti rumorose del gradiente. La componente inerziale che aggiungiamo al gradiente è semplicemente

il gradiente calcolato all'iterazione precedente. Formalmente all'iterazione $k + 1$ avremo:

$$\vec{w}^{k+1} = \vec{w}^k - \eta(\nabla_{\vec{w}} E(\vec{w}^k) + \nabla_{\vec{w}} E(\vec{w}^k)) \quad (11.5)$$

11.2.2 Tecniche avanzate di discesa del gradiente

Sono stati ingegnerizzati moltissimi altri ottimizzatori ciascuno che punta a risolvere le criticità dei precedenti. La maggior parte di questi è già implementata nei framework per l'AI, Pytorch offre:

- Adadelta
- Adam
- LBFGS
- SDG
- Adagrad
- ASDG
- Rprop
- Ecc...

11.3 Batch Training

In uno scenario ideale tutti i campioni del set di training ci portano verso il minimo globale, nella realtà però i dati del set possono essere effetti da errore, questo fa sì che campione per campione il minimo si trovi in posizione diversa e questo porta a oscillare intorno al minimo piuttosto che a raggiungerlo.

Per mitigare l'effetto degli errori e del rumore nei campioni di training si ricorre a tecniche statistiche.

11.3.1 Minibatch

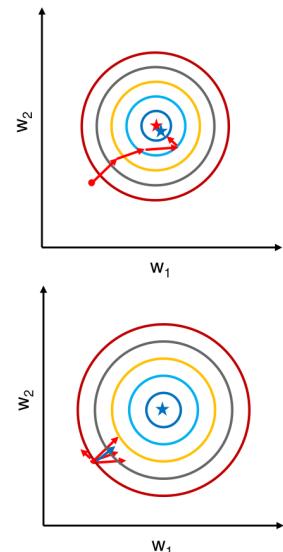
Al posto che addestrare la rete campione per campione si divide il set di training in minibatch di m campioni. E si usano i risultati dell'addestramento sull'intero mini batch per aggiornare i pesi della rete.

```

function MINIBATCHTRAINING
  for all minibatch  $b$  in trainingSet do
     $\vec{G} \leftarrow 0$                                  $\triangleright$  init. accumulatore gradiente
    for all sample  $x$  in  $b$  do
       $y \leftarrow \text{NETINFERENCE}(x)$                  $\triangleright$  risultato della rete
       $E \leftarrow \text{LossFUNCTION}(y)$ 
       $\vec{g} \leftarrow \nabla_{\vec{w}} E(\vec{w})$                $\triangleright$  Calcolo gradiente
       $\vec{G} \leftarrow \vec{G} + \vec{g}$                        $\triangleright$  Accumulo gradienti
    end for
     $\vec{w}^{b+1} \leftarrow \vec{b} + \eta \frac{1}{m} \vec{G}$        $\triangleright$  Aggiorno pesi con la media del gradiente
  end for
end function
```

L'immagine mostra le differenze tra un addestramento sample per sample e uno fatto con i minibatch. Questo approccio

- permette convergenza più rapida dovuta ai minori errori
- è la chiave delle architetture profonde
- rendere computazionalmente efficiente il training
- permette di mischiare immagini di classi diverse in un solo aggiornamento dei pesi



11.4 Overfitting

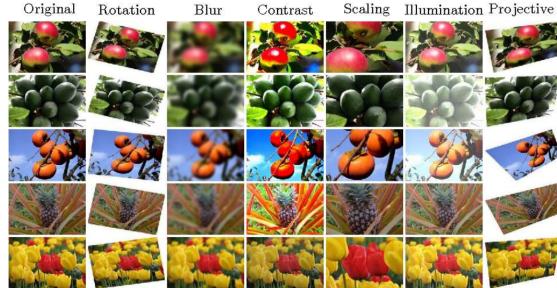
Nel capitolo introduttivo (9.2.2) abbiamo già parlato di come la regolarizzazione possa prevenire l'overfitting, vediamo ora altre tecniche che di solito vengono usate in aggiunta alla regolarizzazione.

11.4.1 Data Augmentation

Dato che le reti neurali hanno bisogno di moltissimi esempi per l'addestramento quelli nel set di training possono non essere sufficienti. Si ricorre allora alla data augmentation che consiste nel sintetizzare nuove immagini a partire da quelle presenti nel set.

Per sintetizzare nuove immagini si può ricorrere a diverse trasformazioni, modificando ad esempio:

- Scala
- Contrasto
- Rotazione
- Luminosità
- Traslazione
- Proiezione
- Blur
- Ecc...



L'immagine mostra esempi di data augmentation relativi a un set di immagini per il riconoscimento dei frutti. Ovviamente non tutte le trasformazioni sono ammissibili, ad esempio se dobbiamo riconoscere del testo non possiamo ruotare di 180° i caratteri.

Grazie alla data augmentation possiamo aumentare la diversità intraclasse dei dati, si è visto sperimentalmente che questo approccio porta a vantaggi notevoli, ad esempio la LeNet300 che abbiamo presentato prima (9.3.1) dopo la data augmentation passa da un tasso di errore del 3.05% a uno del 2.45%

11.4.2 Committees of Neural Networks

L'overfitting è spesso causato dal fatto che ci sono troppi pochi campioni perchè una rete complessa possa apprendere la generalità del problema. Questo approccio quindi consiste nel dividere la rete complessa in tante reti più semplici e unire i risultati degli output di ciascuna per prendere la decisione finale. In questo modo si riduce il numero di parametri di ciascuna rete e si previene l'overfitting.

È importantissimo in questo caso inizializzare i pesi delle sotto-reti in modo attento, non vogliamo avere tante copie di una rete, ma molte reti diverse ciascuna che ha appreso in modo diverso come risolvere il problema.

In base alla tipologia del problema si potranno unire i risultati in modo diverso:

- **Classificazione:** il risultato definitivo sarà quello più votato dalle sotto-reti;
- **Regressione:** il risultato può essere ottenuto facendo la media dei risultati forniti.

11.4.3 Dropout

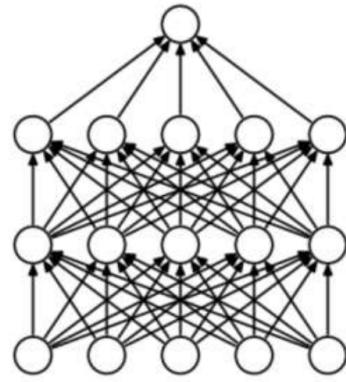
Per rendere più snella la rete un fase di training si possono interrompere alcuni collegamenti nei vari layer, i neuroni alimentati da questi collegamenti resteranno spenti.

La strategia tipica è quella di scegliere un certo tasso di dropout e ogni neurone avrà quella probabilità di essere disattivato.

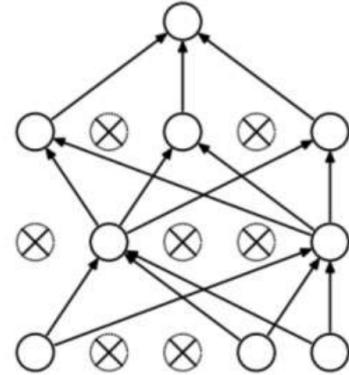
I neuroni spenti non partecipano alla creazione dell'output, né al calcolo del gradiente, inoltre i loro pesi non saranno aggiornati.

Terminato il training tutti i neuroni partecipano all'inferenza, ma l'output di ciascun neurone sarà ridotto in proporzione al tasso di dropout, per compensare l'assenza delle attivazioni durante il training.

Nell'immagine si vede un esempio di rete sottoposta a dropout. Questa tecnica funziona perché permette alla rete di distribuire l'apprendimento su tutti i neuroni e non far affidamento solo su alcuni (o su specifiche connessioni tra neuroni), è come se si addestrassero tante sotto-reti tutte diverse ad ogni iterazione; rendendo la rete più robusta a piccole variazioni nel training set e rendendo l'apprendimento più generale



(a) Standard Neural Net



(b) After applying dropout.