Appunti di Elaborazione di Immagini e Visione Artificiale

A.A. 2022-2023

Matteo Brunello

Contents

1 Introduzione

1.1 Il sistema visivo umano

Le principali componenti dell'occhio umano sono:

- **Cornea** e **sclera**: la cornea è un tessuto duro e trasparente che copre la superficie anteriore dell'occhio, mentre la sclera è una membrana opaca che racchiude il resto dell'occhio che non è coperto dalla cornea;
- **Coroide**: tessuto ricco di vasi sanguigni per il trasporto di nutrienti alle cellule dell'occhio. L'**iride** è parte della coroide e permette di controllare la quantità di luce che entra nell'occhio per mezzo di contrazioni/espansioni, la sua funzione è quella di mettere a fuoco gli oggetti del mondo reale;
- **Lente**: la lente dell'occhio è un tessuto fibroso da cui passa la luce, questo tessuto assorbe relativamente poca luce visibile e molta luce non visibile;
- **Retina**: membrana che ricopre l'interno dell'occhio su cui viene riflessa l'immagine che si sta vedendo. Essa è costituita da:
  - **Coni**: recettori molto sensibili al colore. Sono localizzati in una zona centrale della retina detta *fovea*. C'è ne sono tra i 6 e 7 milioni. La vista di questi recettori è detta *fotopica*;
  - **Bastoncelli**: molto più numerosi dei coni, tra i 75 e 150 milioni. Molto sensibili all'intensità luminosa, servono a creare una visione generale dell'immagine (utile in condizioni di scarsa luminosità), detta anche visione *scotopica*.

Il numero di bastoncelli e coni decresce a 0 a circa 20 gradi di distanza (a sinistra) della fovea, chiamato *blind spot* cioè il punto in cui il nervo ottico è connesso alla retina. I coni sono più densi al centro della retina, decrescendo via via che ci si sposta (in gradi) verso destra e sinistra. D'altra parte, il punto con meno bastoncelli è la fovea, mentre crescono esponenzialmente per raggiungere il massimo a circa 20 gradi e per poi decrescere mano a mano. Tendenzialmente ciò riprende il fatto che i bastoncelli sono più presenti in numero per garantire una migliore visione periferica, mentre i coni per garantire una fedeltà migliore dell'immagine che si sta direttamente guardando.

La concentrazione della sensibilità dei colori dei coni è la seguente:

- 65% sensibili al giallo-arancione;
- 33% sensibili al verde;
- 2% sensibili al blu.

Il colore percepito è la somma di questi colori primari. È bene notare che la concentrazione di coni ripartita in questo modo, non permette di percepire tutti i colori nello spettro della luce visibile.

1.2 Fondamenti di immagini digitali

Un'immagine bidimensionale è una rappresentazione nel piano di una scena o di un fenomeno fisico che ha luogo nel mondo reale. Formalmente possiamo descriverla come una funzione $f(x, y)$ il cui valore è una quantità scalare positiva, il cui significato fisico dipende dalla sorgente dell'immagine. La funzione $f(x, y)$ può essere caratterizzata dal prodotto di due componenti: la quantità di *illuminazione* $i(x, y)$ della scena vista e la quantità di *riflettanza* $r(x, y)$ degli oggetti visti nella scena. Ovviamente, $i(x, y)$ è determinata dalla sorgente di illuminazione, mentre $r(x, y)$ è determinata dalle caratteristiche degli elementi che sono presenti nella scena. Possiamo definire diverse tipologie di immagini:

- **Visibili**: possono essere percepite da un sistema visivo umano oppure ottico;
- **Non direttamente visibili**: create da distribuzioni bi-dimensionali di quantità fisiche quali ad esempio temperatura, pressione o campo elettrico;
- **Definite Analiticamente**: definite da funzioni continue a due variabili oppure da funzioni discrete a due indici.

Le immagini possono essere ulteriormente categorizzate in immagini:

- **Analogiche**: caratterizzate dall'evoluzione *continua* dei valori x, y nello spazio, e dei valori di $f(x, y)$ che definiscono il colore/intesità luminosa;
- **Digitali**: sono ottenute discretizzando i valori di f e dei valori di x e y nello spazio. Il *campionamento* (sampling), è il processo di discretizzazione spaziale (dei valori x e y) e la *quantizzazione* (quantization), è la digitalizzazione del valore di f all'intero più vicino.

Le immagini salvate come una serie di linee (o righe) di pixels sono dette immagini *raster* (aka *bitmap format*). L'acquisizione di un immagine digitale prevede un processo di campionamento della grandezza fisica di riferimento (in quanto grandezza fisica continua). Questo processo può essere visto concettualmente come la superimposizione di una griglia bi-dimensionale. La scelta della grandezza delle celle determina la qualità dell'immagine in un certo senso. Più esse sono grosse, più si accentua il fenomeno in cui le immagini diventano *pixelate*. Questa grandezza è determinata direttamente dal sensore di acquisizione.

Per rappresentare un'immagine digitale, è necessario di fatto salvare in una matrice i valori (quantizzati) di $f(x, y)$. Il numero di bit che si scelgono per rappresentare questi valori determina il *color depth*, rappresentato in *bit per pixel* (bpp). In caso $Im(f) = \{0, 1, \ldots, L-1\}$, saranno necessari $log_2(L)$ bpp per rappresentarne i valori. Se abbiamo immagini catturate in *grayscale*, di solito si utilizza 1 byte (8 bpp). Molto comuni invece sono immagini catturate in RGB (chiamati anche *additive primaries*, in caso siano in CMY sono chiamate *subtractive primaries*), in questo caso si utilizza un byte per canale, per cui ogni pixel richiederà 24 bit. Il numero totale di colori rappresentabile con 24 bit è (2^8 per ogni componente spettrale) $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ colori possibili (circa 16.000.000).

> In generale, per calcolare il numero di bit per pixel, facciamo il logaritmo in base 2 del numero *massimo* di valori rappresentabile.

Siccome le imagini non sono altro che la rappresentazione di una grandezza fisica, vien da sè che è possibile fare diverse operazioni su di esse. Prime tra tutte sono le operazioni logiche quali `and` o `or`. Altre operazioni sono la somma, differenza, moltiplicazione e divisione. Il risultato è definito applicando l'operazione $(\cdot)$ nel modo seguente $h(x, y) = f(x, y) \cdot g(x, y)$.

Un caso di applicazione di queste operazioni è ad esempio la riduzione del rumore per mezzo della media (si veda esempio sulle slides). Altre tipologie di operazioni sono le **trasformazioni affini**, che sono in grado di mantenere linee e parallelismi. Appartententi a questa famiglia di trasformazioni sono le traslazioni, rotazioni, ridimensionamenti e ritagli.

Altro concetto molto importante è la **risoluzione** di un'immagine, che è in funzione della *grandezza* dell'immagine ($W \times H$) e della *dimensione spaziale del supporto* su cui è rappresentata. La risoluzione di un'immagine viene tipicamente misurata in *dots per inch* (dpi), ed esprime la quantità di pixel che vengono visualizzati per unità di lunghezza (in questo caso, un pollice). La formula per il calcolo della risoluzione è

$$\text{resolution} = \frac{\text{size}}{\text{spatial\_dimension}}$$

Come citato in precedenza, il campionamento del segnale analogico periodico (cioè che cambia nel tempo) determina in modo diretto quanto questo segnale viene rappresentato fedelmente a livello digitale. Tanto più frequentemente si acquisiscono i campioni di un segnale, tanto più questo sarà rappresentato fedelmente, valendo invece l'inverso se campionato troppo infrequentemente. Il teorema di **Nyquist-Shannon** ci da la certezza teorica che un segnale che contiene frequenze più piccole di f_c Hertz, può essere ricostruito completamente da campioni presi a intervalli non più grandi di $\frac{1}{2f_c}$ secondi.

2 Color Image Processing

Nonostante la sensazione del colore percepito sia un fenomeno fisiopsicologico, dal punto di vista formale non è nient'altro che una particolare lunghezza d'onda della luce. Il colore percepito di un oggetto è il risultato della luce assorbita e riflessa dello stesso. Ad esempio, se percepiamo un oggetto verde, significa che questo rifletterà lo spettro corrispondente al verde, assorbendo invece tutto il resto.

Ci sono diverse quantità di base che vengono utilizzate per descrivere le componenti delle luce cromatica:

- **Radianza**: quantità totale di energia *emessa* dalla sorgente luminosa (misurata in Watts);
- **Luminanza**: quantità di energia emessa dalla sorgente *percepita* da un osservatore (misurata in Lumen);
- **Luminosità**: intensità della componente *acromatica*, è soggettiva e non formalmente definita;
- **Lunghezza d'onda dominante** (hue): lunghezza d'onda della componente del colore più forte;
- **Saturazione**: rapporto tra la potenza della lunghezza d'onda dominante e del valore medio dello spettro (luce bianca);
- **Cromaticità** (chroma): Hue + Saturazione.

2.1 Spazi colore

Lo scopo di un modello colore (chiamato anche spazio colore o sistema colore), è principalmente quello di standardizzare la specifica dei colori. Un modello colori non è nient'altro che una specifica di un *sistema di coordinate* e un *sottospazio* all'interno di esso, in cui ogni colore è rappresentato da un singolo punto.

2.1.1 Modello RGB

Il modello più semplice è il modello RGB. Il sistema di coordinate è dato dalle 3 componenti principali del colore: rosso, verde e blu. Il sottospazio in cui sono definiti tutti i colori è il cubo unitario in cui i colori RGB primari sono ai 3 vertici, mentre negli altri 3 vertici ci sono il ciano, magenta e giallo. Nei rimanenti 2 vertici, ci sono il nero (che è situato all'origine) e il bianco (situato invece nel vertice più lontano dall'origine). Sulla diagonale principale (dal nero al bianco) del cubo sono situate tutte le possibili tonalità di grigio.

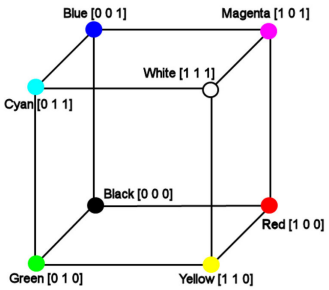

Figure 1: Rappresentazione grafica del cubo RGB.

Ogni valore all'interno del cubo è implicitamente normalizzato nel range $[0, 1]$. Il numero di bits che vengono utilizzati per rappresentare le singole componenti di colore è detto *pixel depth*. Tipicamente, il valore di color depth è circa 8 bit, per cui è possibile rappresentare $2^{24} = 16.777.216$ colori diversi con tale modello.

2.1.2 Modello HSI

I modelli HS* sono basati sulle caratteristiche della luce che nell'occhio umano definiscono direttamente un colore. Quando un essere umano vede un colore, lo descrive in termini delle sue caratteristiche di:

- **Hue** (o lunghezza d'onda dominante): definisce direttamente il colore *puro*;
- **Saturazione**: definisce quanto puro è il colore in termini di quanto questo viene *diluito* dalla componente di luce bianca;
- **Luminosità**: definisce quanto è luminoso il colore.

Siccome la luminosità è definita solo soggettivamente, il modello HSI introduce la componente acromatica di *intensità* del colore, che è invece definibile e direttamente collegata alla sensazione di luminosità. Dal punto di vista formale, questo modello si ottiene essenzialmente *ruotando* il cubo del modello RGB in modo tale da rendere verticale la diagonale. In questo modo, considerando tutti i punti ad una determinata altezza, otterremo un piano. Questo piano, detto *piano cromatico*, conterrà quindi tutti i possibili colori all'intensità luminosa specificata dall'altezza in cui è situato. È immediato notare che la saturazione dei colori aumenta in base alla distanza dal centro di questo piano (punto d'intersezione con la retta che rappresenta i valori di intensità). D'altra parte, se considerassimo un piano "verticale" definito ad esempio dai punti bianco, nero e ciano, su questo piano giaceranno tutti i valori di colori con lo stesso valore di hue (ciano in questo caso). Questo perché i valori all'interno di

tale piano verticale sono ottenuti come combinazione lineare dei valori dei vertici (i colori sono una miscela di questi tre vertici).

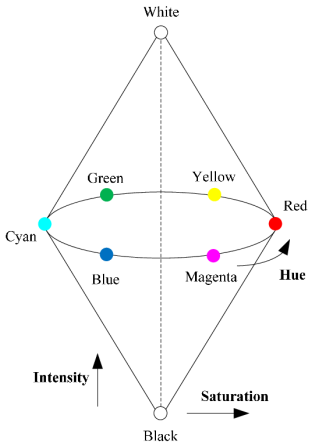

Figure 2: Rappresentazione grafica dello spazio HSI.

Più nel dettaglio, dato un punto nello spazio HSI, il valore di saturazione viene dato dalla lunghezza del vettore giacente sul piano cromatico, mentre il valore di hue viene determinato dall'angolo tra tale vettore e l'asse del rosso (0°: rosso, 60°: giallo, 120°: verde, 180°: ciano, 240°: blu, 300°: magenta). Il valore di hue, è quindi compreso tra $[0; 360]$. Questo tipo di modelli funziona particolarmente bene perché divide la componente cromatica (HS) dalla componente acromatica, luminosa (I). Grazie a questa stretta relazione tra i due modelli di colore possiamo convertire direttamente un valore di RGB in uno HSI e viceversa.

Dal punto di vista formale:

- $I = (r_\gamma + g_\gamma + b_\gamma)/3$;
- $S = 1 - min(r_\gamma + g_\gamma + b_\gamma)/I$ [1].

2.1.3 Modello HSV

Il modello HSV (Hue, Saturation, Value) è molto simile al modello HSI. Questo modello si ottiene trasformando gli angoli del cubo in modo tale da essere portati allo stesso livello del bianco ($V = 1$), ottenendo così uno spazio a forma di piramide rovesciata.

Dal punto di vista formale, siano:

- R: vertice del piano dell'esagono rappresentante il rosso;

[1] in realtà questo valore è definito come 0 in caso il denominatore sia 0 per evitare evidenti problemi.

- O: l'origine (centro) dell'esagono;
- P: il colore da rappresentare;
- $\gamma = (r_\gamma, g_\gamma, b_\gamma)$.

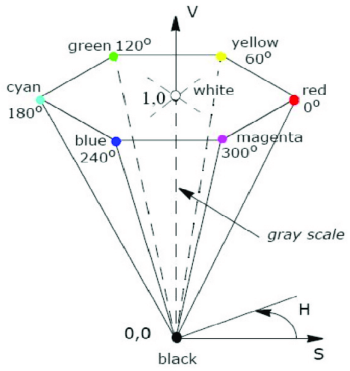

Figure 3: Rappresentazione grafica dello spazio HSV.

Allora, possiamo calcolare i valori:

- $hue_{360}(\gamma) = 360 \cdot \frac{RP}{6OR}$;
- $value(\gamma) = max(r_\gamma, g_\gamma, b_\gamma)$;
- $chroma(\gamma) = max(r_\gamma, g_\gamma, b_\gamma) - min(r_\gamma, g_\gamma, b_\gamma)$;
- $saturation(\gamma) = \frac{chroma(\gamma)}{value(\gamma)}$ [2].

2.1.4 Modello HSL

Anche in questo caso il modello HSL (Hue, Saturation, Lightness) è simile ai precedenti, solamente che i vertici del cubo vengono proiettati sul piano centrale (quello ad altezza $L = 0.5$), ottenendo di fatto una *doppia piramide*. Il vertice superiore della piramide corrisponde ad un valore di luminosity pari a 1 (bianco), mentre quello superiore pari a 0 (nero).

[2] in realtà questo valore è definito come 0 in caso il denominatore sia 0 per evitare evidenti problemi.

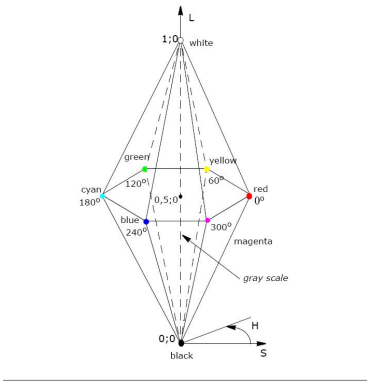

Figure 4: Rappresentazione grafica dello spazio HSL.

Come già detto, i valori di H e S sono calcolati nello stesso modo come nel modello HSV, mentre il valore di L viene calcolato nel modo seguente:

$$lightness(\gamma) = (max(r_\gamma, g_\gamma, b_\gamma) + min(r_\gamma, g_\gamma, b_\gamma))/2$$

2.2 Metodi di Color Image Processing

Esistono diverse tecniche di processamento delle immagini che riguardano direttamente il colore.

2.2.1 Intensity Slicing

Consiste nel trasformare un'immagine a scala di grigi in un'immagine a colori, mappando ogni range di intensità ad un colore specifico. Il razionale sull'impiego di questa tecnica, è che l'occhio umano è in grado di percepire e differenziare meglio i colori che diverse intensità di grigio, per cui potrebbe essere utile per analizzare particolari dettagli di immagini.

2.2.2 False Color Transformation

Un altro tipo di trasformazione consiste invece nel mappare i valori di rosso/verde e giallo, scambiandoli di ordine, oppure variandone le intensità. Questa trasformazione è facilmente implementabile per mezzo di una trasformazione lineare (moltiplicazione vettore/matrice).

3 Image processing types

Esistono diverse tipologie di processamento d'immagini, differenziate in primo luogo dal dominio di processamento, che può essere:

- **Dominio spaziale**: direttamente sui pixel dell'immagine;
- **Dominio delle frequenze**: rappresenta il contenuto delle frequenze spaziali, cioè *quanto* variano le insensità dei vari pixel in direzione orizzontale/verticale.

Le tecniche di image processing che avvengono nel dominio delle frequenze sono l'analogo delle tecniche di processamento del suono, solamente che il segnale si evolve nello spazio anzichè nel tempo, per questo motivo si parla di *frequenze spaziali*.

Le tipologie di tecniche per il processamento delle immagini possono essere categorizzate in modo più generale.

3.1 Image Enhancement

L'obiettivo principale dell'enhancement è di preparare l'immagine ad un'ulteriore processing in modo che questo processing sia più effettivo. Un esempio potrebbe essere quello di applicare un'operazione di rimozione del rumore per migliorare un task di riconoscimento di oggetti.

La valutazione dei risultati di questo tipo di processamento è puramente *soggettiva*, cioè non esistono metriche oggettive per valutare la qualità del risultato. In questo senso, possiamo definire l'image enhancement come un processo che migliora la qualità **percepita da un umano** di un'immagine. Ne deriva naturalmente che le tecniche utilizzate per migliorare la qualità di immagini sono fortemente *application-driven* (*es.* algoritmi usati nel campo medico potrebbero essere totalmente inutili in ambito astronomico).

3.2 Image Restoration

L'obiettivo della restoration è quello di migliorare la qualità dell'immagine, possibilmente ripristinandone parti o caratteristiche perse. La restoration viene applicata quando si ha una conoscenza (parziale o totale) sulla *funzione di degradazione*.

Contrariamente all'enhancement, la restoration è invece un procedimento il cui risultato può essere valutato *oggettivamente*.

3.3 Compression

La compressione ha invece l'obiettivo di ridurre lo spazio richiesto per salvare un'immagine, o ridurre il tempo richiesto per trasmettere tale immagine in un medium di trasmissione. Ci sono due tipologie principali di compressione:

- **Lossy**: l'immagine decompressa potrebbe avere differenze rispetto all'originale.
- **Lossless**: l'immagine decompressa è identica all'immagine originale.

Alcuni esempi di lossy compression possono esssere:

- **Subsampling**: in cui solo un sottoinsieme dei pixel viene salvato. In questo caso, il processo di decompressione dovrà far fronte alla mancanza di pixel impiegando tecniche di replicazione o di interpolazione. In ogni caso, l'immagine decompressa non sarà identica all'originale.
- **Quantizzazione**: in cui si decresce il livello di quantizzione, riducendo così il numero di possibili livelli di intensità, per cui saranno necessari meno bits per codificare il valore di un *color channel* (o livello di grigio). All'estremo di questa tecnica possiamo ottenere immagini a due livelli di colore (bianco e nero), dette immagini *binarizzate*.
- **Sampling e Quantizzazione**: l'idea è quella di modificare adattivamente sia il livello di quantizzazione che il livello di sampling. Questi due livelli vengono fatti variare in base ai livelli di dettaglio in una particolare area dell'immagine. In questa maniera, il numero corretto di campioni e di livelli d'intensità sarà utilizzato per rappresentare i pixel in quella determinata area.

3.4 Segmentation

La segmentazione è un processo di computer vision che consiste nel partizionare un'immagine per individuare le componenti *semanticamente* utili ad un determinato task. Tipicamente, per effettuare un task di segmentazione sono necessari metodi per:

- Rappresentare le parti segmentate (utilizzando la sua area interna, oppure i suoi bordi).
- Descrivere le parti segmentate (in termini di area, perimetro, lunghezza dei bordi).

4 Image Enhancement nel dominio spaziale

Le tipologie di image enhancement si suddividono in tipologie di trasformazione che possono essere:

- **Puntuali**: trasformazione da pixel a pixel, rappresentabili come una mappa $(x_t, y_t) = f(x, y)$;
- **Locali**: trasformazione da insieme di pixel a pixel. L'insieme di pixel è rappresentato da una *funzione di neighborhood* $I(x, y)$, per cui la trasformazione è rappresentata dalla mappa $(x_t, y_t) = f(I(x, y))$;
- **Globali**: trasforma un'intera immagine in un singolo punto.

4.1 Intensity Transformation

Una tipologia di image enhancement è l'*Intensity Transformation*. Abbiamo visto questa trasformazione in particolare per le immagini rappresentate in scala di grigi. Una trasformazione puntuale T di questo tipo, mappa ogni valore di intensità in un'altro valore. Questa funzione può essere implementata attraverso una tabella di conversione chiamata LUT (*Look Up Table*). Queste tabelle vengono rappresentate per mezzo di un grafico bidimensionale che esprime i nuovi valori di T in corrispondenza dei vari valori di grigio.

Una trasformazione T molto importante è la *funzione gamma*, che implementa la cosiddetta *gamma correction*. Tale operazione permettere di espandere/comprimere i livelli di grigio scuri/chiari. La funzione gamma è la seguente:

$$s = c \cdot r^{\gamma}$$

dove:

- c è una costante pre-definita;
- r è il livello di intensità di grigio in input;
- $\gamma > 0$ è l'iperparametro della funzione.

Valori di $\gamma > 1$ determinano un'amplificazione dei livelli di grigio scuri, attenuando il livello di quelli chiari. D'altra parte, con $0 < \gamma < 1$ si ottiene un'amplificazione dei livelli di grigio chiari, attenuando quelli scuri.

Altre trasformazioni degne di nota di questo tipo sono:

- **Ladder**: grafo a *"scaletta"* che riduce il numero di livelli di intensità, creando dei contorni falsi. Questa trasformazione ha il problema che introduce un tipo di compressione irreversibile perchè si perde informazione data dalle linee verticali della funzione (i "salti");
- **Ramp**: grafo che corrisponde ad una funzione a tratti collegata da una rapida ascesa tra i valori minimi e massimi di intensità in un intervallo compreso tra r_1 e r_2. Questa trasformazioe essen-

zialmente serve a ridurre il contrasto dei pixel che hanno intensità troppo bassa ($< r_1$) o troppo alta ($> r_2$). L'operazione è detta *contrast stretching*;
- **Binarization**: come suggerisce il nome, trasforma i valori di intensità in intensità massima (1) e minima (0). Il grafico di tale LUT è una funzione a tratti.

Fin'ora si è parlato di LUT per trasformazioni di immagini in scala di grigi. Nei casi invece di immagini a colori, si impiegano le *pseudocolor* LUT; tabelle che hanno una LUT per ogni canale di colore.

4.2 Bit-Plane Slicing

La tecnica di bit-plane slicing consiste nel rappresentare un'immagine (a scala di grigi) inizialmente in binario e poi creare un'immagine binarizzata per ogni posizione di bit di ogni pixel. Ad esempio, se ci concentriamo su un dato pixel con valore 010, l'operazione produrrà 3 piani in cui lo stesso pixel avrà valore 0 nel primo, 1 nel secondo e infine 0 nel terzo piano.

Concentrandosi ad esempio sui piani dei bit meno significativi si può ridurre il rumore, mentre su quelli più significativi si potrebbero enfatizzare i dettagli. Si può utilizzare questa tecnica anche per fare compressione, permettendo di salvare spazio, pur mantenendo le features dell'immagine intatte.

4.3 Histogram Processing

È una tecnica che ricade nelle tecniche di *global processing*. Innanzitutto, l'istogramma dei livelli di grigio di un'immagine è una funzione discreta h definita come

$$h(r_k) = n_k$$

dove:

- r_k: è k-esimo livello di grigio ($r_k \in [0; 255]$ nel caso siano 8bpp);
- n_k: è il numero (*frequenza*) di pixel nell'immagine con intensità r_k.

Tipicamente, per far si che i valori siano tutti tra 0 e 1, si utilizza una versione normalizzata dell'istrogramma:

$$p(r_k) = \frac{n_k}{M \cdot N}$$

dove M sono il numero di pixel per riga e N il numero di pixel per colonna.

La versione normalizzata di un istogramma definisce la probabilità che un qualsiasi pixel abbia uno specifico valore di grigio r_k. La sua rappresentazione grafica dà una descrizione globale dell'immagine.

In questo senso possiamo anche definire la *frequenza cumulativa* (o Cumulative Density Function)

$$F(r_k) = \sum_{i=0}^{k} p(r_i)$$

Gli istogrammi riassumono quindi la distribuzione dell'intensità di pixel nell'immagine. Ad esempio, istogrammi di immagini molto scure, avranno una distribuzione *skewed* verso valori bassi di r_k. Altri casi interessanti sono gli istogrammi delle immagini a basso contrasto (che appariranno come una distribuzione maggiormente concentrata verso i valori centrali), e quelle a basso contrasto (con una distribuzione molto omogenea su tutti i possibili valori).

4.3.1 Histogram Equalization

Come suggerisce il nome, il metodo della histogram equalization serve ad aumentare il *contrasto* globale dell'immagine. Lo scopo di questa tecnica è quindi quello di modificare le intensità dei pixel in modo che l'istogramma risultante sia il più possibile uniforme. Di conseguenza, i valori dei pixel dell'immagine saranno quindi distribuiti più uniformemente, sfruttando l'intero range dei valori disponibili.

Siano $p_r(r_k)$ la distribuzione dell'immagine originale e $p_s(s_k)$ la distribuzione target (*uniforme*). La trasformazione $s = T(r)$ per fare histogram equalization è:

$$T(r) = s = (L-1)\int_0^r p_r(z)dz$$

È possibile verificare che T genera di fatto in output una distribuzione uniforme sfruttando un un risultato della teoria delle probabilità:

$$p_s(s_k) = p_r(r_k)\left|\frac{dr}{ds}\right|$$

ma siccome

$$\begin{aligned}\frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= (L-1)\frac{[\int_0^r p_r(z)dz]}{dr} \\ &= (L-1)p_r(r)\end{aligned}$$

allora $\frac{dr}{ds} = \frac{1}{(L-1)p_r(r)}$, per cui possiamo sostituirlo nell'equazione precedente e verificare se la dis-

tribuzione risultante è uniforme:

$$\begin{aligned} p_s(s_k) &= p_r(r_k)\left|\frac{dr}{ds}\right| \\ &= (L-1)\left|\frac{1}{(L-1)p_r(r)}\right| \\ &= \frac{1}{(L-1)} \end{aligned}$$

È evidente che l'espressione coincide con quella di una distribuzione uniforme, per cui T è la trasformazione necessaria.

Si è trattato attualmente il caso le variabili siano continue (si noti l'integrale), per ottenere la stessa trasformazione nel discreto è sufficiente sostituire alla formula della trasformazione l'integrale con la distribuzione cumulativa ed eventualmente normalizzate per ottenere una probabilità. L'istogramma equalizzato sarà quindi ottenuto applicando la seguente trasformazione

$$T(r_k) = (L-1)\frac{H(r_k)}{MN}$$

dove:

- $H(r_k) = \sum_{i=0}^{k} h(r_i)$ è *l'istogramma cumulativo*[3];
- M numero di colonne e N numero di righe;
- $L-1$ livelli di intensità (255 per immagini a 8bpp).

Un problema di questa tecnica è che queste trasformazioni potrebbero mappare più valori di intensità sullo stesso valore, di fatto causando una perdita di dettagli nell'immagine originale.

4.3.2 Histogram Specification

L'histogram specification consiste nel mappare i valori di intensità in modo da cambiare la forma dell'istogramma, senza limitarsi solamente ad una forma uniforme come nell'equalization. In altri termini, l'histogram equalization non è nient'altro che un caso specifico di histogram specification, in cui la forma desiderata è una distribuzione uniforme.

Si vuole trasformare l'istogramma in un istogramma con distribuzione *target* $p_z(z)$. Si può seguire lo stesso ragionamento dell'*histogram equalization*, e ottenere la trasformazione $G(z)$ come

$$G(z) = s = (L-1)\int_0^z p_z(v)dv$$

[3] È come la distribuzione cumulativa, con la differenza che non viene fatta rispetto alle *probabilià*, ma rispetto alle *frequenze*.

Si ha quindi una trasformazione $T : r \to s$ discussa precedentemente, e una trasformazione $G : z \to s$. Quello che si vuole ottenere è però una trasformazione da r a z ($r \to z$).

$$G : z \to \mathbf{s} \leftarrow r : T$$

Per ottenerla basta fare l'inversa G^{-1}, per cui la mappa per ottenere z a partire da r sarà definita nel modo seguente

$$z = G^{-1}(T(r))$$

Per fare histogram specification bisogna eseguire 3 passi ben definiti:

1. Utilizza l'immagine originale per ottenere $T(r)$;
2. Utilizza la distribuzione desiderata $p_z(s)$ per calcolare $G(z)$, invertendola successivamente;
3. Per ogni pixel, ottieni la sua traformazione s applicando $T(r)$, poi applica G^{-1} al risultato s, ottenendo z.

Valgono le considerazioni fatte in precedenza per il caso discreto, per cui tutte le trasformazioni equivalenti nel discreto possono essere ottenute da quelle nel continuo, semplicemente sostituendo le distribuzioni cumulative continue con quelle discrete.

4.4 Filtri Spaziali

Un'altra tecnica di image enhancement consiste nell'applicare dei filtri spaziali. I filtri spaziali permettono di fare un processing *locale* all'immagine. L'idea è quella di definire una maschera bidimensionale che determina *"quanto"* la singola intensità deve essere amplificata o ridotta, e successivamente sovraimporre tale maschera all'immagine scorrendo via via tutte le possibili posizioni. Questa operazione è detta *convoluzione* ed è riassumibile (nel caso bidimensionale e discreto) dalla seguente formula

$$y(n,m) = \sum_{k=-a/2}^{a/2} \sum_{l=-b/2}^{b/2} x(n-k, m-l)h(k,l)$$

dove:

- $h(k,l)$ è la *maschera* definita nella regione $(-a,a)$, $(-b,b)$;
- $x(n,m)$ è l'intensità del pixel alla posizione (n,m).

Ovviamente bisogna tenere conto dei pixel ai bordi dell'immagine, poichè la maschera potrebbe "*uscire*" al di fuori dell'immagine. In questi casi si effettua un'operazione di *0-padding*, cioè si estende l'immagine opportunamente di valori pari a 0.

In generale i valori di h sono normalizzati a 1, alternativamente è necessario dividere la formula per la somma di tutti i pesi della maschera. I valori indicano se il tipo del filtro è passa-basso o passa-alto.

Valori $= 1$ lasciano passare il segnale inalterato, mentre $= 0$ lo filtrano.

Possiamo vedere questo tipo di processamento essenzialmente come una media pesata specificata dalla maschera (o filtro). Dal punto di vista operativo, i filtri passa basso vanno a rimuovere i dettagli dell'immagine, evidenziando aree grandi nell'immagine.

4.4.1 Ordered Statistics Filters

Un'altra tipologia di filtri è quella basata sulle statistiche ordinate come la *mediana*. Un filtro mediano funziona andando a fare una statistica dei pixel definiti nella finestra del filtro, ordinandoli in ordine ascendente e prendendo il valore centrale come l'output del fitro. Questo tipo di filtri è particolarmente utile per rimuovere del rumore di tipo *salt&pepper*.

4.4.2 Filtri Passa Alto

Vengono utilizzati in task come *sharpening* o *contour extraction*. Questa tipologia di filtri viene ottenuta andando a simulare un operatore di derivazione. Sono stati visti essenzialmente 2 operatori basati su questa tecnica: il Laplaciano e il Jacobiano.

4.4.3 Operatore Laplaciano

Il primo operatore visto è il *Laplaciano*. Questo operatore permette di fare *contour extraction*, cioè evidenzia i contoni (o bordi) degli oggetti presenti nelle immagini. L'idea nasce dal fatto che la derivata seconda è proporzionale alle variazioni di intensità nell'immagine. Essa è difatti $\neq 0$ solo in prossimità di cambiamenti dei valori di intensità. Seguendo questa intuizione, si introduce il Laplaciano, un operatore che approssima la derivata seconda di una funzione

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

nel caso delle immagini si utilizza il filtro Laplaciano isotropico (cioè un filtro *invariante rispetto alle rotazioni*).

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

Notiamo essenzialmente che la definizione di questa derivata non è nient'altro che una somma pesata per cui possimo definire un kernel (maschera) che potrà essere combinata eventualmente con

l'immagine facendone un'operazione di convoluzione. Il kernel risultante sarà:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Si noti che il kernel precedente non è nient'altro che la somma dei due kernel calcolati su una specifica direzione

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Una volta definito il Laplaciano, possiamo utilizzarlo per fare image sharpening.

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

in cui g è l'immagine risultato dell'operazione, mentre f è l'immagine originale. c è un coefficiente che può essere 1 oppure -1, tipicamente non si scelgono valori grandi poichè aumenterebbero il rumore nell'immagine. L'evidenza che l'operatore faccia sharpening è data dal fatto che la derivata seconda assume valori più grandi in prossimità in cui ci sono bruschi cambiamenti di intensità. L'operazione rinforzerà quindi i bordi degli oggetti nell'immagine, oscurando (rendendo nero) lo sfondo. Il fatto che si aggiunge il Laplaciano può essere visto intuitivamente come l'aggiunta di un segnale che ha valori alti su porzioni in cui cambia bruscamente l'immagine, mentre valori bassi in tutte le altre porzioni.

4.4.4 Operatore Gradiente

Il gradiente è invece la derivata prima di una funzione a più variabili (nel nostro caso parliamo di funzioni bivariate). Non è nien'altro che un *vettore* a due componenti che punta nella direzione che ha l'intensità massima.

$$\nabla f = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}$$

Per ottenere una singola componente numerica si considera il modulo M del gradiente

$$M(x, y) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Spesso però per motivi di linearità si considera l'approssimazione del modulo seguente:

$$M(x, y) = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

Tra gli operatori che vanno a calcolare il gradiente abbiamo l'operatore di *Roberts*, che calcola la derivata sulla direzione *diagonale*. Ad esempio, la maschera che implementa l'operatore di Roberts per calcolare il gradiente (rispetto alla diagonale in cui ci sono valori diversi da 0) è il seguente

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

È importante sottolineare che il pixel che viene considerato ogni volta durante l'operazione di convoluzioe viene "allineato" alla componente $(0, 0)$ della maschera. Questo perchè non esiste un centro ben definito essendo la maschera 4×4. Proprio per il fatto che utilizzare maschere di dimensioni pari provoca questo problema si preferisce utilizzare l'operatore di *Sobel*, che è invece definito da una maschera 3×3. Gli operatori di Sobel non si concentrano sulle componenti diagonali ma sulle direzioni verticali e orizzontali. Ad esempio, la seguente maschera implementa l'operatore di *Sobel* che enfatizza le linee orizzontali

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

La somma dell'operatore per le linee verticali e quello per le linee orizzontali ci da il gradiente.

Un'altro operatore che implementa il gradiente è l'operatore di *Prewitt*, essenzialmente questo operatore dà meno importanza al pixel centrare rispetto all'operatore di *Sobel*. L'operatore di Prewitt per le linee orizzontali è implementato nel modo seguente (quello per le linee verticali ne è semplicemente una rotazione di 90 gradi in senso antiorario)

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Anche in questo caso, per ottenere il gradiente si fa la somma delle due derivate direzionali (prima si calcola poi si fa la somma).

Per riassumere, sia il Gradiente che il Laplaciano servono a evidenziare i contorni (sono sensibili ad essi), ma è importante sottolineare varie differenze tra i due. In generale, il Laplaciano:

- Mantiene i dettagli;
- È più sensibile al rumore (più evidente in zone lisce);
- Viene utilizzato per fare *sharpening*.

Mentre il Gradiente ha una risposta più marcata in aree in cui ci sono transizioni di grigio molto significative, in cui i bordi sono particolarmente più spessi, per cui è preferibile per fare *edge detection*.

4.5 Contour Enhancement (Local Histogram Statistics)

Un'altro metodo di enhancement a livello spaziale è il cosiddetto contour enhancement. L'obiettivo ad esempio potrebbe essere quello di migliorare delle aree scure dell'immagine lasciando invariate quelle più chiare. L'idea alla base è quella di utilizzare delle statistiche *locali*, definite in un'opportuna finestra di dimensioni $N \times M$. Ad esempio, la media sarà

$$\mu = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{x=0}^{M-1} f(x, y)$$

ed eventualmente la varianza locale

$$\sigma^2 = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{x=0}^{M-1} [f(x, y) - \mu]^2$$

A questo punto, il processing andrà a considerare pixel per pixel, considerando opportunamente per ognuno di esso queste statistiche. Il processing verrà effettuato solo se i seguenti vincoli sono soddisfatti:

- Se $\mu_s \leq k_0 \cdot \mu_G$ (dove μ_s è il valore medio *locale*, μ_G la media globale e $0 \leq k_0 \leq 1$ un opportuno valore fissato), in modo tale da considerare solamente i valori al di sotto di una determinata soglia di threshold, ignorando quelli al di sopra della stessa;
- Se $\sigma_S \leq k_2 \sigma_G$, in modo da considerare solo i pixel che non sono già in una zona ad altro constrasto;
- Se $k_1 \sigma_G \leq \sigma_S$ (ovviamente $k_1 < k_2$) in modo tale da evitare di aumentare il contrasto in zone uniformi.

Mettendo insieme tutti questi vincoli, si ottiene la seguente formula riassuntiva per l'aumento locale di contrasto:

$$g(x, y) = \begin{cases} E \cdot f(x, y) & \text{if } \mu_s \leq k_0 \mu_G \text{ and } k_1 \sigma_G \leq \sigma_S \leq k_2 \sigma_G \\ f(x, y) & \text{otherwise} \end{cases}$$

Questo tipo di processamento può essere applicato per fare *contour enhancement*, in generale, zone che hanno una varianza locale molto alta si traducono in un'alta variabilità delle intensità di pixel (all'interno della finestra). Le zone che contengono bordi sono spesso tali da avere un'alta varianza, siccome contengono spesso molti pixel di varie intensità. D'altra parte, zone di *background* avranno un numero più ristretto di intensità, risultando quindi in zone con varianza più piccola.

5 La trasformata di Fourier

5.1 Introduzione

L'idea alla base dello studio dei segnali periodici di Fourier è che ogni funzione **periodica** può essere scritta come la **somma** di *seni* e *coseni*, di differenti *ampiezze* e *frequenze* (in caso solo un tipo di funzione venga utilizzato - seno o coseno - allora si utilizza anche la *fase*). Questa somma viene chiama **serie di Fourier**.

Nel caso invece delle funzioni **non-periodiche**, si possono rappresentare come un *integrale* di *seni* e *coseni* pesati (da un'opportuna funzione). Questa rappresentazione viene invece chiamata **trasformata di Fourier**.

> *La rappresentazione nel **dominio di Fourier** è* equivalente *all'originale.*

5.2 La serie di Fourier

Come già citato in precedenza, ogni funzione periodica di periodo T (o equivalentemente di frequenza $\omega = \frac{2\pi}{T}$) può essere espresso come

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{j\omega nt} \quad \text{con } n \in \mathbb{Z}$$

dove i coefficienti calcolati con la seguente formula

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-j\omega nt} dt$$

i coefficienti vanno ad indicare *quanto* la frequenza fondamentale ωn è presente all'interno del segnale $f(t)$. In altri termini, questo integrale va a calcolare la similarità tra il segnale *"sonda"* $f(t)$ e la frequenza fondamentale $e^{j\omega nt}$.

> **Formula di Eulero**: $e^{j\omega} = cos(\omega) + jsin(\omega)$

Si noti che T indica la *risoluzione* della pulsazione in frequenza. Tanto più T è grande, tanto più sarà grande.

5.3 La trasformata di Fourier

Per fare da *"ponte"* tra la serie e la trasformata di Fourier, è necessario introdurre un concetto fondamentale: la distribuzione *delta di Dirac*. Tale funzione (più correttamente *distribuzione*) è caratterizzata

da un impulso di ampiezza infinita e di una durata infinitamente piccola (puntuale), ed è definita matematicamente nel modo seguente

$$\delta(t) = \begin{cases} \infty & \text{if } t = 0 \\ 0 & \text{otherwise} \end{cases}$$

inoltre, essendo una distribuzione deve valere che

$$\int_{-\infty}^{\infty} \delta(t)dt = 1$$

ha una proprietà molto importante detta *shifting property* (oppure *sampling property*) tale per cui, presa una funzione $f(t)$ e moltiplicata per la *delta di Dirac*, ne ritorna il valore al punto 0 (in generale, nel punto in cui è centrata la funzione δ). Tale proprietà generale è riassumibile nel modo seguente

$$\int_{-\infty}^{\infty} f(t)\delta(t-u)dt = f(t-u)$$

L'intuizione è che siccome la distribuzione δ è 0 ovunque tranne che intorno ai valori $t-u$, l'integrale anche è nullo per cui si possono considerare solo gli estremi di integrazione in un intorno infinitesimamente piccolo di $t-u$. In questo intorno la funzione $f(t-u)$ non cambia, per cui è costante e può esser portata fuori dall'integrale. A questo punto, l'integrale della distribuzione δ è 1, per cui ne deriva la dimostrazione.

È importante notare che questa proprietà vale anche nel discreto, creando di fatto una connessione tra la trasformata e la serie che verrà discussa più avanti.

Avendo introdotto tutto il necessario, per ottenere la trasformata dalla serie facciamo tendere (nella definizione di serie di Fourier) $T \to \infty$. Focalizzandosi inizialmente solo sul calcolo dei coefficienti c_n, notiamo in primo luogo (dalla definizione data precedentemente), che l'estremo di integrazione diventa da $-\infty$ a $+\infty$. Inoltre, la variabile discreta n, siccome è divisa per T, indica delle frequenze arbitrariamente vicine, per cui $\frac{n}{T}$ viene sostituito da una variabile continua u. Infine, il termine moltiplicativo $\frac{1}{T}$ diventa per definizione il *differenziale* dell'integrale dt. Otteniamo quindi la cosiddetta **trasformata** di Fourier, riportata in seguito

$$F(u) = \int_{+\infty}^{-\infty} f(t)e^{-j2\pi ut}dt$$

Si noti che la funzione $F(u)$ è in funzione della *frequenza*, ed equivale alla rappresentazione di f nel dominio delle *frequenze*. È possibile anche definire la trasformata inversa, semplicemente applicando lo stesso processo al limite alla definizione della *serie*, e sostituendo le definizioni dei coefficienti con

la definizione di trasformata

$$f(t) = \int_{+\infty}^{-\infty} F(u) e^{j2\pi ut} du$$

tra la trasformata e l'inversa cambia solo il segno dell'esponenziale, per cui si dice che la trasformata rispetta la proprietà di **simmetria**.

Un'altra proprietà importante della trasformata di Fourier è che nonostante individui tutte le frequenze presenti in un segnale, le informazioni di tipo *spaziale* (o *temporale* nel caso di segnali nel tempo) vengono perse. Questo è dimostrabile considerando la trasformata di un segnale qualunque traslato poiché il risultato (in modulo) non cambia rispetto al segnale non traslato (dettagli matematici sule slides).

5.3.1 Trasformate importanti

5.3.1.1 Funzione Rettangolo Una delle poche funzioni di cui è necessaria una certa dimestichezza con la sua trasformata è la funzione $sinc$. Per introdurla, prendiamo un segnale a finestra (potrebbe, per esempio, essere una riga di pixel) definita nel modo seguente

$$f(t) = \begin{cases} A & \text{for } -a \leq t \leq a \\ 0 & \text{otherwise} \end{cases}$$

La trasformata F di tale funzione è la cosiddetta *funzione sinc*:

$$F(u) = \int_{-a}^{+a} A e^{-j2\pi ut} dt = \left[\frac{A}{-2j\pi u} e^{-2\pi ut} \right]_{-a}^{a} = \frac{sin(2a\pi u)}{2a\pi u} = a \cdot A \cdot sinc(2au)$$

> *Graficamente, la funzione $sinc$ ha valore 1 in $u = 0$, e oscilla con $u \to \infty$ avvicinandoci asintoticamente all'asse x.*

Ovviamente, possiamo anche definire la trasformata per segnali bi-dimensionali, per cui la trasformata di un segnale rettangolare bi-dimensionale (ad esempio una porzione d'immagine) in questo caso sarebbe una funzione $sinc$ bi-dimensionale.

5.3.1.2 Delta di Dirac Vale la pena considerare anche la trasformata della funzione δ introdotta precedentemente, poichè ha diverse caratteristiche degne di nota.

$$F(u) = \int_{+\infty}^{-\infty} \delta(t) e^{-j2\pi ut} dt = e^{-j2\pi u0} = 1$$

il fatto che la trasformata sia la funzione costante 1, indica che tutte le frequenze sono contenute all'interno di un implulso. In generale, vale che

$$F(u) = \int_{+\infty}^{-\infty} \delta(t - t_0) e^{-j2\pi ut} dt = e^{-j2\pi ut_0}$$

ciò significa che la trasformata della funzione delta di Dirac individua la frequenza fondamentale multipla di t_0.

Inoltre, grazie alla relazione della trasformata inversa otteniamo che

$$\delta(t) = \int_{+\infty}^{-\infty} e^{j2\pi ut} du$$

5.3.1.3 Segnale Monocromatico Consideriamo il segnale monocromatico $f(t) = Acos(2\pi u_0 t)$ e calcoliamone la trasformata

$$F(u) = \int_{+\infty}^{-\infty} A\ cos(2\pi u_0 t) e^{-j2\pi ut} dt = A\left(\delta(u - u_0) + \delta(u + u_0)\right)$$

ciò significa che la trasformata sarà composta da due $delta$ di Dirac centrate nei punti u_0 e $-u_0$. Questo è ragionevole poichè la trasformata è sempre simmetrica rispetto all'asse y, per cui la funzione $delta$ individua la frequenza u_0, che è effettivamente l'unica frequenza contenuta nella funzione originale.

5.4 Teorema della Convoluzione

Precedentemente abbiamo trattato la convoluzione come operazione di media mobile nel discreto, ma è possibile definirla anche nel continuo. La convoluzione di due funzioni h e f è così definita

$$f(t) \star h(t) = \int_{-\infty}^{\infty} f(\tau) h(t - \tau) d\tau$$

Teorema 1 (Di Convoluzione)**.** *La convoluzione di due funzioni h e f è pari al prodotto delle loro rispettive trasformate H e F, formalmente*

$$f(t) \star h(t) = F(u) \cdot H(u)$$

La dimostrazione del teorema è banale, poichè basta applicare la definizione di trasformata all'operazione di convoluzione. Calcoliamo innanzitutto la trasformata di $h(t - \tau)$ per semplificare i

calcoli successivamente

$$\begin{aligned}\mathscr{F}[h(t-\tau)] &= \int h(t-\tau)e^{-j2\pi ut}dt \\ &= \int h(v)e^{-j2\pi u(v+\tau)}dv \ (v=t-\tau) \\ &= e^{-j2\pi u\tau}\int h(v)e^{-j2\pi v}dv \\ &= H(u)\cdot e^{-j2\pi u\tau}\end{aligned}$$

passiamo ora a calcolare la trasformata della convoluzione, ottenendo quindi la dimostrazione. Il passaggio principale consiste nel cambiare l'ordine degli integrali (per linearità), sostituendo successivamente le varie definizioni che compaiono

$$\begin{aligned}\mathscr{F}[f\star h] &= \int\left(\int f(\tau)h(t-\tau)d\tau\right)e^{-j2\pi ut}dt \\ &= \int f(\tau)\left(\int h(t-\tau)e^{-j2\pi ut}dt\right)d\tau \\ &= \int f(\tau)H(u)e^{-j2\pi u\tau}d\tau \\ &= H(u)\int f(\tau)e^{-j2\pi u\tau}d\tau \\ &= F(u)\cdot H(u) \quad \blacksquare\end{aligned}$$

da questo teorema risulta evidente il perché h venga anche chiamato *filtro*. Vale quindi che tutte le operazioni che abbiamo definito nel dominio dello spazio per mezzo di un'operazione di convoluzione, possono essere fatte semplicemente per mezzo di una moltiplicazione nel dominio delle frequenze.

Vale anche l'inverso, cioè $f(t)\cdot h(t) = F\star H$.

5.5 Teorema del Campionamento

Per campionare un segnale possiamo utilizzare la proprietà vista in precedenza della funzione δ di Dirac. L'idea è quella di definire una serie di funzioni δ equispaziate di un certo ΔT (chiamato *periodo di campionamento*). Si costruisce quindi il seguente *treno di impulsi*

$$s(t) = \sum_{n=-\infty}^{+\infty}\delta(t-n\Delta T)$$

la cui trasformata sarà

$$S(u) = \frac{1}{\Delta T}\sum_{n=-\infty}^{+\infty}\delta(u-\frac{n}{\Delta T})$$

La trasformata è ancora un *treno di impulsi*, equispaziati di $\frac{n}{\Delta T}$. Si noti che tanto più gli impulsi sono vicini nella funzione iniziale (quindi valori ΔT piccoli che corrispondono ad una frequenza di campionamento alta), tanto più gli impulsi nella trasformata saranno distanti.

Intuitivamente, quindi, se cerchiamo di avvicinare le varie delta tra loro nel dominio spaziale (o temporale), le distanziamo nel dominio delle frequenze. Questo è ragionevole perché la trasformata andrà a coprire frequenze più alte.

Dal punto di vista formale, il campionamento di un segnale continuo $f(t)$ consiste nel moltiplicare il treno di impulsi per tale segnale. Dal teorema di campionamento, sappiamo che il prodotto di due funzioni nello spazio è la loro convoluzione nelle frequenze, per cui il sengale campionato nello spazio delle frequenze sarà cioè la convoluzione della trasformata di f e della trasformata del treno di impulsi s.

$$
\begin{aligned}
\tilde{F}(u) &= F(u) \star S(u) \\
&= \sum_{n=-\infty}^{+\infty} f(t) \cdot \delta(t - n\Delta T) \\
&= \int F(\tau) S(u - \tau) d\tau \\
&= \frac{1}{\Delta T} \int F(\tau) \sum_{n=-\infty}^{\infty} \delta\left(u - \tau - \frac{n}{\Delta T}\right) d\tau \\
&= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \int F(\tau) \delta\left(u - \tau - \frac{n}{\Delta T}\right) d\tau \quad (\delta \neq 0 \text{ solo quando } u - \tau - \frac{n}{\Delta T} = 0) \\
&= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(u - \frac{n}{\Delta T}\right) d\tau
\end{aligned}
$$

questo risultato ci dice che la trasformata di Fourier del segnale *digitalizzato* è una versione periodicizzata della trasformata di Fourier del segnale *originale* di periodo $\frac{1}{\Delta T}$.

Quindi, per ricostruire il segnale originale, si potrebbe prendere solo il primo periodo della funzione e poi andare ad applicare una trasformata inversa. Questo però può essere fatto quando ΔT è sufficientemente piccolo da distanziare bene le varie funzioni (si campiona a sufficienza). Se questo non succede, le funzioni potrebbero essere troppo vicine e sovrapporsi tra di loro (poichè la trasformata è una somma di tutte queste funzioni, per cui si sovrappongono tra loro). Questo fenomeno viene detto *aliasing*.

Un teorema già citato in precedenza che ci permette di scegliere la frequenza di campionamento con la garanzia che il segnale ricostruito sia indistinguibile da quello originale è il teorema di campionamento Nyquist-Shannon. L'assunto di questo teorema è che i segnali siano a banda limitata, cioè esista un limite superiore di frequenze che non danno più contributi significativi all'intero segnale. Questo è ragionevole dal punto di vista pratico, poichè oltre ad una certa risoluzione, ad esempio, l'occhio

non riesce a discernere alcuni particolari. Sia quindi μ_{max} la frequenza massima (definita in base al dominio), il teorema di Nyquist-Shannon ci dice che il segnale ricostruito è instinguibile da quello originale se

$$\mu_{max} < \frac{1}{\Delta T} \quad e \quad \frac{1}{\Delta T} > 2\mu_{max}$$

Quando le condizioni del teorema di campionamento non possono essere applicate, cioè il segnale con cui si ha a che fare non è limitato in banda (non ha un upper bound μ_{max}), si può risolvere applicando un'operazione di *anti-aliasing* a monte, in modo da eliminare tutte le frequenze al di sopra del limite prefissato. Così facendo, si evita che durante il campionamento tali frequenze compaiano e vadano ad interferire con le altre, creando di fatto dell'aliasing. Nel caso delle immagini, applicare l'anti-aliasing corrisponde semplicemente ad applicare un filtro passa basso, creando del *blurring*. Nelle immagini, queste zone di alta frequenza sono rappresentate spesso da valori che cambiano rapidamente (ad esempio in una scacchiera), per quello si parla di smoothing.

Mettendo tutto insieme, per ricostruire il segnale originale, si applica la trasformata inversa alla trasformata del segnale campionato, opportunamente filtrata in una determinata finestra.

$$f(t) = \mathscr{F}^{-1}(F(u)) = \mathscr{F}^{-1}(H(u) \cdot \tilde{F}(u))$$

(altri passaggi sono stati ommessi, per dettagli ulteriori, andare a pagina 47 delle slides)

5.6 Trasformata di Fourier (2 Dimensioni)

Fin'ora abbiamo trattato la versione ad una dimesnione della trasformata di Fourier, ma quando parliamo di immagini parliamo invece di funzioni a due variabili (come visto, $f(x, y)$ indica l'intensità di un singolo pixel). Vale la pena introdurre la versione bidimensionale della trasformata di Fourier

$$F(u, v) = \int_{\mathbb{R}} \int_{\mathbb{R}} f(t, z) e^{-j2\pi(ut+vz)} dt dz$$

Essa si ottiene semplicemente facendo prima la trasformata su una dimensione, ottenendo una funzione intermedia e poi facendo la trasformata sulla restante dimensione su questa funzione. La trasformata inversa è banale

$$f(t, z) = \int_{\mathbb{R}} \int_{\mathbb{R}} f(t, z) e^{j2\pi(ut+vz)} du dv$$

Ovviamente, tutto ciò che è stato discusso in una dimensione vale anche per due dimensioni, compreso il teorema del campionamento.

Per il processamento di immagini è necessaria la versione della trasformata 2D nel continuo (DFT)

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

sapendo ovviamente che $(x,y) \in M \times N$ è il dominio della funzione f (dimensioni dell'immagine).

5.7 Trasformata di Fourier e Immagini

Come detto, la Trasforamata ci permette di ottenere una rappresentazione in termini di frequenze. Se noi applichiamo la DFT (2D) ad un'immagine, otterremo la sua rapresentazion in termini di frequenze. Tipicamente se ne visualizza il modulo in una scala logaritmica normalizzata in modo da avere l'origine al centro. In questo modo, ovviamente si perde informazione di tipo spaziale, ma si hanno importanti insights sulla composizione dell'immagine. Ad esempio, bruschi cambiamenti (da bianco a nero) spesso corrispondono a linee di valori alti in modulo che sono nella stessa direzione delle stesse nel dominio spaziale. Questo perché la TF mantiene le informazioni di tipo *rotazionale*. D'altra parte, valori che cambiano con meno frequenza saranno ovviamente rappresentati da valori meno marcati, sempre sull'asse in cui queste variazioni occorrono.

(si veda l'esempio sulle slides, pagina 81)

Come detto, facendo il modulo si perde informazione di tipo spaziale, ma questa informazione è invece contenuta nelle *fasi* delle varie sinusoidi. In altri termini, la fase mantiene informazioni che rappresentano essenzialmente i *bordi* contenuti nell'immagine.

5.8 Elaborazione nel dominio delle frequenze

Alla base di ogni metodo di elaborazione abbiamo sempre l'operazione di convoluzione già vista in precedenza.

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

In frequenza, possiamo sfruttare il teorema della convoluzione e fare questo tipo di processing semplicemente per mezzo di una moltiplicazione. Ovviamente, quando si vanno a fare le trasformate, spesso si ottengono due matrici di dimensioni differenti. Spesso, infatti, il kernel (o filtro) ha dimensioni molto più ridotte rispetto all'immagine. Questo si risolve facendo *0-padding* sulla matrice della trasformata del kernel in modo che le dimensioni siano le stesse.

Quando facciamo le trasformate, dobbiamo vedere le immagini come dei segnali periodici, cioè che si ripetono all'infinito su tutte le dimensioni x e y. Questa periodicità può causare degli artefatti ai bordi delle immagini sotto opportune condizioni, chiamato problema della *circular convolution*. Per mitigare

questo fenomeno, si fa 0 padding in entrambe le immagini (kernel e immagine in input) in modo tale rendere le immagini entrambe di dimensioni $P \times Q$, dove

$$P = A + C - 1$$
$$Q = B + D - 1$$

in cui:

- $A \times B$ sono le dimensioni dell'immagine;
- $C \times D$ sono le dimensioni del kernel.

6 Image Enhancement nel dominio di Fourier

Reiterando quello che è stato visto nel capitolo precendente, nel dominio di Fourier vengono rappresentate le informazioni che riguardano l'intesità di *pattern* presenti nell'immagine (ad esempio, $F(0,0)$ contiene il valore medio dell'immagine). Quando facciamo image enhancement nel dominio di Fourier è necessario sempre considerare dei filtri che siano *simmetrici* rispetto all'origine. In caso contrario, quando si ricostruisce l'immagine elaborata si otterrebbero valori complessi.

> **Importante**: come già detto, i filtri già visti che operano nel dominio dello spazio sono interpretabili anche nel dominio delle frequenze, eccetto però i filtri *non-lineari*. La trasformata di Fourier, infatti, permette di implementare solo filtri che sono lineari, per cui, il filtro *mediano* ad esempio non è implementabile nel dominio delle frequenze (così come tutti i filtri che coinvolgono ad esempio un ordinamento).

L'unica cosa che considereremo sarà lo *spettro* (modulo) della trasformata, dimenticandoci in un certo senso della fase, che supporremo nulla in tutto il dominio (come già discusso, rimane però di centrale importanza).

Possiamo riassumere il processamento nel dominio delle frequenze nei seguenti passaggi:

$$f_{elab}(x,y) = \mathscr{F}^{-1}[H(u,v)F(u,v)]$$

da cui sono evidenti i singoli passaggi per effettuare l'elaborazione:

1. Trasformazione dell'immagine e del filtro tramite DFT;
2. Filtraggio delle frequenze per mezzo della moltiplicazione dell'immagine traformata e del filtro trasformato;
3. Ricostruzione dell'immagine elaborata tramite trafrormata inversa (IDFT).

6.1 Esempi di Filtri

6.1.1 Mean Removal

Consideriamo ad esempio il seguente filtro

$$H(u,v) = \begin{cases} 0 & u = v = 0 \\ 1 & \text{otherwise} \end{cases}$$

Tale filtro va a rimuovere tutte le componenti *continue*, cioè dove i valori non cambiano tra loro e quindi sono uniformi essendo che hanno frequenza pari a 0. Il risultato di questa applicazione è che

l'immagine risultante sarà a media nulla, cioè la media di tutti i valori di pixel sarà 0.

6.1.2 Filtri passa alto/basso ideali

Altri filtri degni di nota sono i filtri *passa alto* e *passa basso*. Il primo cancella del frequenze basse, per cui va ad accentuare essenzialmente i bordi riducendo l'intensità nelle zone omogenee. L'effetto del secondo è invece l'opposto, per cui si ottengono delle operazioni di *smoothing* dell'immagine. Un filtro passa basso ideale è implementato nel modo seguente:

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \leq D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

dove la funzione D indica la *distanza* di un punto dal centro:

$$D(u,v) = \sqrt{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2}$$

Di solito, in realtà, per fare processamento di immagini si preferisce mantenere il processing a livello spaziale. Quindi la fase di progettazione del filtro avviene nel dominio delle frequenze, per poi fare una IDF del filtro e ottenere la sua versione spaziale. Se suppponessimo di voler progettare un filtro passa basso ideale (rappresentato in una dimensione da una funzione rettangolare), ci scontreremo però subito con un problema: il *ringing*. L'idealità di un filtro passa-basso/alto nella frequenza compromette l'idealità nello spazio e viceversa. Questo può essere spiegato semplicemente guardando l'antitrasformata di un filtro rappresentato da una funzione a rettangolo, in cui ci sono diverse oscillazioni che generano rumore. Per andare a rappresentare un cambio brusco (continuo) il filtro risultante è infinito in dominio (funzione *sinc*). Perciò di solito si tende a "tagliarlo" oltre una certa soglia per renderlo utilizzabile (di solito i filtri hanno dimensioni molto più piccole delle immagini). Ciò che si ottiene, è evidente andando a fare la trasformata di questo filtro ottenuto, che conterrà meno sinusoidi e quindi conterrà una versione distorta della funzione a rettangolo con diversi artefatti introdotti dalle interazioni delle cosinusoidi. Questo perché avendo rimosso alcuni valori della funzione nel dominio dello spazio (andando a tagliarla) si vanno effettivamente a rimuovere delle informazioni cruciali (meno campioni) per ricostruire la funzione rettangolo nel dominio delle frequenze. Come già detto, il fenomeno che introduce queste discontinuità è detto *ringing*.

6.1.3 Gaussiano

Siccome questo problema viene dato dalla *discontinuità* del filtro (cioè non è derivabile nei punti in cui cambia di valore), si potrebbe evitare utilizzando un'approssimazione differenziabile di tale filtro. Una

possibile funzione per implementare il filtro passa basso è la curva *Gaussiana*.

$$H(u) = Ae^{-u^2/2\sigma^2}$$

La cosa interessante della gaussiana è che è invariante rispetto alla trasformata. L'unica cosa che cambia è la forma:

- Se σ è grande, allora la trasformata sarà larga e l'antitrasformata stretta;
- Se σ è piccolo, allora la trasformata sarà stretta e l'antitrasformata larga.

Se volessimo invece implementare un filtro passa basso, possiamo *invertire* la Gaussiana

$$H(u) = 1 - Ae^{-u^2/2\sigma^2}$$

L'unico problema della Gaussiana è che risulta difficile andare a rimuovere totalmente alcune frequenze, non essendo mai a 0.

La parte di Image Enhancement vista fin'ora è funzionale a due grandi classi di elaborazione d'immagini. La più grande è il miglioramento soggettivo dell'immagine (ad esempio, per la visualizzazione). La seconda, invece, è quella di utilizzare l'enhancement come fase di pre-processing in una pipeline di elaborazione delle immagini.

6.1.4 Butterworth

> TODO.

6.1.5 Laplaciano

Il Laplaciano che abbiamo visto precedentemente è rappresentabile anche nel dominio delle frequenze. È molto più semplici difatti fare la derivata nel dominio delle frequenze rispetto allo spazio.

$$\nabla^2 f(t,z) = \frac{\partial^2 f(t,z)}{\partial t^2} \frac{\partial^2 f(t,z)}{\partial z^2}$$

La derivata della trasformata corrisponde a moltiplicare le trasformate per $(j2\pi u)^2$:

$$\mathscr{F}[f(t,z)] = (j2\pi u)^2 \cdot F(u,v) + (j2\pi u)^2 \cdot F(u,v) = -4\pi^2(u^2+v^2)F(u,v)$$

per cui il filtro nel dominio delle frequenze sarà

$$H(u,v) = -4\pi^2(u^2+v^2)$$

il cui grafico del modulo rappresenta una *"coppa"*, per cui si tratta di un filtro *passa-alto*. Siccome il filtro "parte" dall'angolo in alto a sinistra $(0, 0)$, si preferisce sempre fare una traslazione in modo che sia al centro dell'immagine

$$H(u,v) = -4\pi^2 \left[\left(u - \frac{P}{2} \right)^2 + \left(v - \frac{Q}{2} \right)^2 \right] = -4\pi^2 D^2(u,v)$$

dove

$$D^2(u,v) = \sqrt{\left(u - \frac{P}{2} \right)^2 + \left(v - \frac{Q}{2} \right)^2}$$

Si ricorda l'applicazione del Laplaciano vista nei capitoli precedenti

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

per cui, se volessimo calcolarlo nel dominio delle frequenze

$$\begin{aligned} g(x,y) &= \mathscr{F}^{-1}[F(u,v) - H(u,v)F(u,v)] \\ &= \mathscr{F}^{-1}[F(u,v) - (c + 4\pi D^2(u,v))] \end{aligned}$$

implementare lo sharpening in frequenza corrisponde quindi a moltiplicare per un filtro che ha le sembianze di una coppa rovesciata traslata rispetto all'asse z di c, per cui lascia passare il valor medio in maniera proporzionale al valore di c. Il risultato dell'applicazione di questo filtro sono effetti in cui il valor medio permane, esaltando però i valori di alta frequenza (texture, bordi, pattern).

6.1.6 Filtraggio Omomorfico

È un tipo di filtering che serve a migliorare l'immagine in cui l'illuminazione non è soddisfacente. Si parte da un *modello* che descrive l'immagine, secondo il quale essa è caratterizzata da 2 componenti

$$f(x,y) = i(x,y)r(x,y)$$

dove:

- $i(x,y)$ è la luminosità del pixel, cioè il valore con cui il pixel in questione *riceve* la luce;
- $r(x,y)$ è la riflettanza, cioè il valore con cui il pixel in questione *riflette* la luce ricevuta.

La riflettanza descrive la *scena*, cioè gli oggetti presenti nell'immagine. Assumiamo nella maggior parte dei casi che ci si interessi solo l'oggettività della scena, cioè degli oggetti presenti nella scena. L'obiettivo è quello di rendere uniforme la luminosità dell'immagine, cioè in modo tale che tutta la scena sia illuminata in modo uniforme, senza che alcune zone siano più o meno illuminate di altre. Possiamo sfruttare il fatto che la componente luminosa non ha variazioni significative all'interno dell'immagine

(alte frequenze), per cui l'idea è quella di applicare un filtro high-pass alla componente i e nessun filtro a r. Per evitare difficoltà e separare le due componenti che sono due prodotti (anche perché per il teorema della convoluzione, la trasformata di un prodotto diventerebbe una convoluzione), si utilizza un *"trucco"* matematico che consiste nel considerare il *logaritmo* dell'immagine, cosicchè il prodotto diventi la somma dei logaritmi delle componenti, formalmente

$$z(x,y) = \log f(x,y) = \log i(x,y) + \log r(x,y)$$

quindi, la trasformata sarà ugualmente semplice

$$Z(u,v) = F_i(u,v) + F_r(u,v)$$

a questo punto si può filtrare nello spazio delle frequenze

$$S(u,v) = H(u,v)Z(u,v) = H(u,v)F_i(u,v) + H(u,v)F_r(u,v)$$

l'immagine elaborata (in scala logaritmica) sarà ovviamente l'antitrasformata di S, il che è composta per definizione dalla componente i' elaborata e dalla componente r' elaborata:

$$s(x,y) = \mathscr{F}^{-1}[S(u,v)] = i'(x,y) + r'(x,y)$$

l'algoritmo termina andando a rimuovere i logarimi, tornando alla scala normale, semplicemente facneod l'esponenziale di s:

$$g(x,y) = e^{s(x,y)} = i_0(x,y)r_0(x,y)$$

Come detto in precedenza, il filtro H deve andare ad agire solamente sulla componente luminosa i ed è descritto dalla seguente formula:

$$H(u,v) = (\gamma_H - \gamma_L)\left(1 - e^{-c(D^2(u,v)/D_0^2)}\right) + \gamma_L$$

questo filtro effettua essenzialmente uno sharpening al dominio logaritmico. Dalla formula, se:

- $\gamma_L < 1$ vengono attenuate le basse frequenze (illuminazione);
- $\gamma_H > 1$ vengono amplificate le alte frequenze (riflettanza), aumentando il contrasto.

6.1.7 Gaussian Pyramid

Per evitare di fare aliasing durante il downsampling (riduzione delle dimensioni dell'immagine), si può applicare un filtro passa basso a metà della frequenza (corrisponde essenzialmente al filtro D^2 in cui $D_0 = f/2$, ovviamente però si usa un Gaussiano o un qualsiasi altro filtro passa basso realizzabile).

Questa operazione di downsampling non sono è utile per fare un'operazione di resizing, ma serve per costruire delle rappresentazioni multi-risoluzione. Un idea è appunto quella di catturare un'immagine e poterla elaborare a diversi stadi di risoluzione. Un tipo di applicazione è ad esempio Google Earth, dove ovviamente non possono essere caricate tutte le immagini di risoluzione massima in tutta la memoria dell'intero globo, ma in base alla distanza viene aumentata la risoluzione su particolari zone di dettaglio. Un'altro approccio può essere quello di applicare diverse elaborazioni di Computer Vision a diverse risoluzioni per aumentare la precisione degli algoritmi.

Per gestire le varie risoluzioni dell'immagine si utilizza una struttura dati a *piramide*, rappresentata da un formato `TIFF`. La piramide si costruisce partendo dalla piena risoluzione e facendo mano a mano una *cascata* di resizing (applicando un filtro low pass e un successivo downsampling, mantenendo essenzialmente un pixel si e uno no per righe e per colonne). Se l'operazione di downsampling è implementato per mezzo di un filtro Gaussiano, questa struttura viene chiamata *Piramide Gaussiana* (Gaussian Pyramid).

6.1.8 Laplacian Pyramid

Un'altro esempio di struttura è la *Piramide Laplaciana*. Al posto di memorizzare ogni immagine sotto-campionata, l'idea è invece quella di andare a memorizzare l'*errore* (opportunamente pre-calcolato) che viene commesso considerando una versione a bassa risoluzione dell'immagine e scalandola per mezzo di un'interpolazione al livello successivo della piramide (ad esempio, $\times 2$). In questo caso, l'errore riguarda maggiormente le alte frequenze, poiché facendo questa interpolazione si perdono tutti i dettagli riguardandi i bordi ad esempio. Quindi, l'idea è quella di memorizzare questi dettagli persi (che sono di fatto il Laplaciano dell'immagine), in modo tale che quando l'immagine viene scalata per mezzo dell'interpolazione, si possono aggiungere questi dettagli, semplicemente sommandone il Laplaciano corrispondente.

La costruzione di un singolo step di questa piramide viene fatto per mezzo dei seguenti passaggi:

1. Parto dalla immagine in input (al primo step sarà l'immagine a risoluzione piena) e ne faccio il downsample ottenendo un'immagine a grandezza dimezzata.
2. Partendo dall'immagine downsampled al passo precedente, ne faccio upsampling per mezzo di un metodo di interpolazione[4] in modo da tornare alle dimensioni di partenza.
3. Si calcola la discrepanza tra l'immagine upsampled (risultato) e l'immagine originale (ground truth), semplicemente facendone la differenza.
4. Si reitera il processo prendendo come immagine di input l'immagine ottenuta facendo down-sampling.

[4] Per fare interpolazione si prende l'immagine upsampled composta dai pixel dell'immagine con i pixel mancanti posti a 0 (alternano tra pixel a 0 e pixel dell'immagine), e successivamente si applica un filtro Gaussiano per fare smoothing.

A questo punto, in memoria saranno memorizzate solo 2 componenti:

1. L'ultimo step di downsample, quindi l'immagine con risoluzione più bassa;
2. I vari errori (dettagli, Laplaciani) ottenuti in ogni step della piramide.

Tutte queste componenti hanno l'informazione necessaria e sufficiente a ricostruire tutte le varie risoluzioni, fino ad arrivare a quella di partenza. L'idea è quella già discussa; si parte semplicemente dal primo step, si fa interpolazione ottenendo una versione upsampled, e poi si somma il dettaglio corrispondente a quello step particolare. Ovviamente, per ottenere l'immagine a step n, devono essere ripercorsi tutti gli step precedenti, partendo dall'immagine originale.

La differenza tra piramide Gaussiana e Laplaciana è essenzialmente un trade-off tra tempo computazionale e memoria. Mentre la prima è meno efficiente dal punto di vista della memoria, non deve utilizzare nessuna elaborazione per fare retrieval di una determinata immagine ad una determinata risoluzione. D'altra parte, la piramide Laplaciana, è più efficiente in termini di memoria ma deve andare a fare una serie di step per recuperare l'immagine alla risoluzione desiderata, per cui è meno efficiente dal punto di vista computazionale.

Un'applicazione della piramide Laplaciana che abbiamo visto è il *matting*, cioè il blending di due immagini tra loro (metà e metà ad esempio). Con il *pyramid matting* si intende una tecnica che utilizza la piramide Laplaciana per fare un matting più *smooth* rispetto ad un matting netto.

6.1.9 Altri Filtri

L'idea alla base di questi filtri è esprimere queste funzioni in termini della distanza rispetto all'origine

$$D(u,v) = \sqrt{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2}$$

6.1.9.1 Esponenziale Il filtro esponenziale è rappresentato nel modo seguente:

$$H(u,v) = e^{-[D(u,v)/D_0]^n}$$

Dove i parametri sono D_0 (*cutoff frequency*) e n, che controlla quanto è accentuata la transizione. Valori alti indicano transizioni alte.

6.1.9.2 Trapezoidale Questo filtro ha essenzialmente due parametri:

- D_0 la banda passante;
- D_1 la banda di taglio (*cutoff frequency*).

Con $D_0 < D_1$. La regione delimitata da D_0 e D_1 viene detta *cutoff region*. Il filtro (cross section) in prossimità di D_0 ha valore 1, mentre ha valore 0 in prossimità di D_1.

6.1.9.3 Bandreject/Bandpass Questi filtri vanno a processare delle bande specifiche di frequenza, non solo alte o basse. I filtri bandreject possono essere implementati come i filtri che sono stati visti in precedenza: ideale, Butterworth e Gaussiano. Hanno però 2 parametri addizionali:

- D_0 che indica il punto di *cutoff* in cui sarà centrato il filtro, cioè il punto in cui viene centrata la finestra che delimita la banda di frequenza da rimuovere;
- W indica la larghezza totale della finestra.

Di seguito sono riportate le varie implementazioni dei filtri bandreject.

- **Ideale**:
$$H(u,v) = \begin{cases} 0 & \text{if } D_0 - \frac{W}{2} \leq D \leq D_0 + \frac{W}{2} \\ 1 & \text{otherwise} \end{cases}$$
- **Butterworth**:
$$H(u,v) = \frac{1}{1 + \left[\frac{DW}{D^2 - D_0^2}\right]^{2n}}$$
- **Gaussiano**:
$$H(u,v) = 1 - e^{\left[-\frac{D^2 - D_0^2}{DW}\right]}$$

Ovviamente, l'equivalente filtro passabanda si ottiene per simmetria del bandreject, ovvero

$$H_{BP}(u,v) = 1 - H_{BR}(u,v)$$

6.1.9.4 Filtri Notch Sono una famiglia di filtri che si concentrano su specifiche zone di frequenza. In questo senso, vanno a rimuovere o far passare (o incentivare) le frequenze in alcune aree del rettangolo (filtro). Ovviamente, come ogni filtro, per essere valido deve essere simmetrico rispetto all'origine. Questi filtri non sono più specificati per un range come i filtri visti precedentemente, ma per delle specifiche frequenze (u_k, v_k) (nel caso bi-dimensionale). Di queste frequenze possono esserne processate una moltitudine $(u_1, v_1), (u_2, v_2), \ldots, (u_S, v_S)$, ognuna differente dall'altra. Siccome si vuole processare differentemente una singola frequenza, si associa un filtro differente per ognuna $H_k(u,v)$, centrato nel punto (u_k, v_k) corrispondente, che effettuerà il processing voluto per la frequenza k-esima. A questo punto, il filtro Notch è definito dalla concatenazione dei vari filtri

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

dove H_{-k} viene inserito per non rompere la simmetria del filtro. I filtri H_k utilizzati sono tipicamente Gaussiani o Butterworth, per evitare i problemi (*ringing*) discussi in precedenza dei filtri ideali.

Per scrivere l'espressione eplicita dei filtri bisogna fare però qualche accortezza nel definire la distanza, poiché sarà rispetto al nuovo centro (u_k, v_k) del singolo filtro H_k. La distanza sarà quindi (assumendo il centro dell'immagine essere $(M/2, N/2)$):

$$D_k(u,v) = \sqrt{(u - M/2 - u_k)^2 + (v - N/2 - v_k)^2}$$

A questo punto, se volessimo un filtro Notch-reject Butterworth avremmo

$$H_{NR}(u,v) = \prod_{k=1}^{S} \left[\frac{1}{1 + (D_{0k}/D_k(u,v))^{2n}}\right] \left[\frac{1}{1 + (D_{0k}/D_{-k}(u,v))^{2n}}\right]$$

in cui D_{0k} sono le *cutoff frequencies* di ogni filtro H_k, che posono essere ovviamente tutte diverse. Così come tutti gli altri filtri, se volessimo invece un Notch-pass:

$$H_{NP}(u,v) = 1 - H_{NR}(u,v)$$

Uno degli esempi visto in cui è utile il Notch Filter è rimuovere pattern periodici tipo Moiré patterns, oppure pattern periodici anche solo in una particolare direzione. Questo processing si avvicina quindi molto al task di restoration.

7 Image Restoration

L'image restoration ha come obiettivo il *miglioramento* dell'immagine, ma a differenza dell'*enhancement* (che è un processo soggettivo), si tratta invece di un processo **oggettivo**. Il *modus operandi* consiste nel modellare il *difetto* dell'immagine, per poi poterlo riparare. Si utilizzano poi delle funzioni per misurare la *performance* della tecnica di restoration utilizzata, in modo da raggiungere un risultato ottimale. Così come l'enhancement, possiamo effetture la restoration sia nel dominio delle frequenze (es. *blur removal*), che nel dominio spaziale (es. *additive noise reduction*).

> Tipicamente quando ci sono problemi nell'immagine che hanno una località spaziale (che sono più prominenti in alcune zone specifiche dell'immagine) si preferisce un processing a livello spaziale. D'altra parte, se il problema è distribuito su tutta l'immagine, è preferibile un processing in frequenza (soprattutto se si tratta di un pattern periodico).

Ci sono due grandi problemi legati alla restoration:

- *Denoising*: in cui vengono utilizzati modelli che approssimano il rumore dell'immagine e un successivo filtering dello stesso. Di solito viene approcciato nel dominio spaziale;
- *Inversion*: consistono nel modellare la funzione degradante per ottenerne l'inversa, in modo tale che da invertire il processo degradante. Di solito viene approcciato nel dominio delle frequenze.

Questi due problemi non sono isolati tra loro ma bensì sono legati in diversi modi che saranno chiariti più avanti nel capitolo.

Prima di iniziare il capitolo, è necesario però introdurre il modello matematico che descrive il processo di degrado e restauro a cui si farà riferimento per il resto del capitolo. Si suppone di aver un'immagine ideale $f(x, y)$ a cui viene applicata una funzione di degrado H, utilizzata per modellare difetti di *acquisizione* (es. movimento del sensore, blurring atmosferico), e successivamente ne viene aggiunto un certo rumore, modellato dalla funzione $\eta(x, y)$ (es. rumore del sensore). L'immagine degradata da queste due operazioni è $g(x, y)$. A questo punto, si applicano i filtri di restoration che sfrutteranno delle conoscenze specifiche su H e η, e sarà tanto più efficace tanto più precise le approssimazioni delle funzioni degradanti. Il processo di restoration produrrà un'immagine $\hat{f}(x, y)$ che sarà una stima dell'immagine originle $f(x, y)$ (*ground-truth*).

7.1 Rumore statistico additivo

Ci concentreremo su una classe specifica di problemi, caratterizzata di funzioni di degrado *lineari* e *position-invariant* e rumore *additivo*. La funzione di degrato è deterministica, poiché spesso viene modellata sulla base di osservazioni o specifiche del sensore di acquisizione. Il rumore additivo, invece,

è determinato dall'ambiente per cui è un qualcosa di totalmente casuale ed è quindi un modello statistico.

Dal punto di vista spaziale, possiamo rappresentare l'intero processo di degrado come:

$$g(u,v) = h(u,v) \star f(u,v) + \eta(u,v)$$

Sfruttando il teorema di Convoluzione possiamo ottenere la rappresentazione equivalente nel dominio di Fourier:

$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

Il rumore η è affetto da diversi fattori, che possono essere anche introdotti dagli strumenti di cattura. Ogni sensore, infatti, ha una propria *cifra di rumore*, tant'è che i produttori di sensori provvedono a fornire una cosiddetta curva che indica il *signal-to-noise* ratio in funzione dell'apertura ISO. Per ridurre questo rumore, infatti, spesso si vanno a fare delle medie tra diverse misurazioni per avere delle stime più precise. Anche il rumore si può analizzare in frequenza. Degno di nota è il cosiddetto *white-noise*, un rumore uniforme ovunque che viene introdotto impostando a 0 i pixel in modo casuale. La componente spettrale di questo rumore sarà una funzione costante, cioè contiene tutte le frequenze di altezza $sigma^2$. In generale, inoltre, il rumore che considereremo è invariante per posizione.

Il rumore $\eta(x,y)$ viene costruito a partire da una densità di probabilità $f_\eta(z)$ che indica la probabilià che nella matrice η ci sia un certo valore di intensità. Possiamo modellare questa distribuzione con diverse distribuzioni note, ad esempio:

- **Gaussiana**: $p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\bar{z})^2/2\sigma^2}$, centrata nel valor medio $\bar{z}$ e avrà tanto più rumore tanto più σ sarà grande;
- **Uniforme**;
- **Rayleigh**;
- **Esponenziale**;
- **Impulso**.

Per capire sperimentalmente se nell'immagine c'è del rumore è confrontare la stima della proabilità dell'intensità dei pixel. L'istogramma è proprio una distribuzione dei livelli di intensità dell'immagine e può fornire un'approssimazione iniziale della distribuzione ricercata. Ovviamente questa è un'approssimazione molto grezza poichè modella le varie intensità in modo indipendente, nonostante queste siano dipendenti dalla loro posizione spaziale nell'immagine [5]. Ottenere un modello statistico che tenga conto anche della correlazione tra i pixel è molto difficile, ed esula dagli scopi di questo corso.

[5] Per convincersene, è sufficiente prendere l'istogramma di un'immagine e utilizzarlo per generare una nuova immagine, campionando semplicemente i valori dei pixel dall'istogramma in modo proporzionale ad esso. Quello che si otterrebbe non è un'immagine simile a quella ottenuta ma semplice rumore, proprio perché la distribuzione non tiene conto delle correlazioni spaziali tra pixel.

Abbandonando quindi l'idea di modellare statisticamente l'intera immagine, ci si può focalizzare invece sulla modellazione statistica del rumore. Esso è appunto un processo indipendente dal punto di vista statistico, per cui l'istogramma potrebbe rappresentare un'ottima approssimazione della distribuzione.

Se si fa appunto l'assuzione che il rumore sia uniforme e spazio invariante, focalizzandosi su determinate porzioni dell'immagine e calcolandone l'istogramma, saranno evidenti la dispersione rispetto al valore medio (il valore reale dell'immagine senza rumore) introdotta dal rumore. Difatti, se ci si focalizza su zone sufficientemente uniformi dell'immagine, l'istogramma assumerebbe proprio la forma di una distribuzione simile a quella del rumore stesso, centrata sul valore della porzione dell'immagine (il valore di pixel che è uguale per tutti i pixel della porzione siccome è uniforme).

Se supponiamo di non conoscere la distribuzione del rumore, l'idea è quindi quella di prendere più porzioni uniformi dell'immagine e calcolarne l'istogramma. Se la distribuzione è simile per tutti allora si può dedurre che il rumore è spazio invariante, additivo e bianco (scorrelato da punto a punto). Quindi la forma dell'istogramma rappresenta un modo di capire qual'è la densità di probabilità del rumore e tramite un processo di fitting è anche possibile andare a stimare quali siano i parametri di questa distribuzione.

Per rimuovere il rumore whitenoise si va a fare una media delle misurazioni in modo da abbassare la varianza σ^2. Tale operazione corrisponde di fatto a fare un filtro passa-basso nel dominio della frequenza. Se si immagina appunto il rumore presente nell'immagine, nel dominio di Fuorier comparirebbe come una funzione costante alta σ^2. L'applicazione di un filtro passa basso permette di rimuvere tutto quel rumore dopo la soglia del filtro, che tipicamente non contiene troppe informazioni riguardo l'immagine (frequenze alte), siccome tipicamente le immagini hanno un'alta concetrazione dell'energia nelle frequenze basse.

Un'altra tipologia di rumore è il rumore periodico, che a questo punto comparirà nella rappresentazione spaziale come una delta centrata nella frequenza dello stesso. Per rimuovere questo rumore si può utilizzare un Notch filter, visto in precedenza.

Possiamo concludere che il rumore possiamo processarlo sia in frequenza che nello spazio. Se il rumore è periodico si preferisce processarlo in frequenza, se invece si tratta di whitenoise, allora si preferisce il processing nello spazio.

7.1.1 Filtri lineari

Come già accennato, i filtri spaziali utilizzati sono principalmente delle medie (dove S_{xy} è l'intorno del pixel (x, y) di dimensioni $m \times n$):

- **Aritmetica**: in cui il rumore è ridotto dal blurring;

$$\frac{1}{nm}\sum_{(s,t)\in S_{xy}} g(s,t)$$

- **Geometrica**: i dettagli sono maggiormente preservati;

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

- **Armonica**: buoni per rumori *"salt"* e Gaussiano;

$$\hat{f}(x,y) = \frac{nm}{\sum_{(s,t)\in S_{xy}} 1/g(s,t)}$$

- **Contrarmonico**: generalizzazione delle medie armoniche e aritmetiche.

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

  - Con $Q > 0$ buono per la rimozione del rumore *"pepper"*;
  - Con $Q < 0$ buono per la rimozione del rumore *"salt"*;
  - Con $Q = 0$ corrisponde alla media *aritmetica*;
  - Con $Q = -1$ corrisponde alla media *armonica*.

È utile osservare che applicare questi smoothing non elimina completamente il rumore. Inoltre, anche la qualità dell'immagine viene intaccata un minimo. Bisogna quindi trovare il compromesso tra i due.

7.1.2 Filtri non lineari

Possiamo usare anche filtri non lineari per rimuovere il rumore, come diversi filtri basati su statistiche ordinate:

- **Mediana**: rimuove rumore a impulso;

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

- **Massimo**: riduce rumore pepe;

$$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}} g(s,t)$$

- **Minimo**: riduce rumore sale;
$$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}} g(s,t)$$
- **Punto medio**:
$$\hat{f}(x,y) = \left[\max_{(s,t)\in S_{xy}} g(s,t) \min_{(s,t)\in S_{xy}} g(s,t)\right]/2$$
- **Alpha Trimmed**: rimuove prima del calcolo della media, sia i $d/2$ valori più piccoli che più grandi. Lavora bene con rumore sale-pepe e Gaussiano mischiati insieme.
$$\hat{f}(x,y) = \frac{1}{mn-d} \sum_{(s,t)\in S_{xy}} g'(s,t)$$

7.1.3 Filtri adattivi

Sono filtri che adattano il loro comportamento al contenuto di una finestra locale $m \times n$. Questi filtri vengono applicati solamente a zone in cui non c'è molto rumore per evitare di incentivarlo. Una possibilità è quella di raccogliere delle statistiche locali di ogni finestra S_{xy} che si va a processare. Un esempio visto riguarda appunto il calcolo della *media* (che rappresenta la *luminosità* media) e la *varianza* (che rappresenta il *contrasto*) locali. L'idea è che in punti in cui la varianza è molto alta, è molto probabile che ci si trovi in prossimità di un bordo, per cui rappresenta un dettaglio che auspicabilmente non si deve processare per evitare di danneggiarlo. D'altra parte, se la varianza è bassa o è molto simile ad una varianza sperimentale (ad esempio data dal difetto del sensore di acquisizione), allora si è in prossimità di rumore, per cui si va a processare con un filtro. Un filtro adattivo di questo tipo è il seguente:
$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L]$$
in cui:

- σ_η^2 è la varianza del rumore del sensore, oppure di un rumore ottenuto sperimentalmente a partire da un dataset di immagini;
- σ_L^2 è la varianza locale;
- m_L è la media locale.

Dalla formula, è evidente che il rapporto tra σ_η^2 e σ_L^2 sia fondamentale, poiché determina il comportamento del filtro. Nello specifico, possiamo distinguere i diversi casi:

- $\sigma_\eta^2 = 0$ allora non c'è rumore, per cui non si processa e si ritorna $g(x,y)$;
- Se $\sigma_L^2 = \sigma_\eta^2$ allora $\hat{f}(x,y) = \text{mean}_{S_{xy}} g(x,y)$. Si è in prossimità di un punto che ha le stesse proprietà globali dell'immagine, per cui vogliamo processarla;

- Se $\sigma_L^2 >> \sigma_\eta^2$ allora $\hat{f}(x,y) \approx g(x,y)$. Si è in prossimità di un punto in cui ci sono bordi o dettagli importanti che non devono essere processati principalmente per due motivi: si rovinerebbero i dettagli e comunque il rumore sarebbe meno visibile poiché i dettagli lo *"coprirebbero"*.

Un'altra possibilità per creare filtri adattivi è quella di variare i parametri dei filtri in base all'immagine, al posto di decidere se applicare o meno il filtering in base ad essa. Un tipo di questo filtro è l'*adaptive median filter*, in cui viene cambiato l'unico parametro del filtro mediano: le dimensioni del filtro stesso. Infatti, l'idea alla base di questo filtro è quella di *ridurre* le dimensioni in prossimità di punti in cui la varianza è alta e di *accrescerle* in punti in cui è bassa. Vale la pena introdurre le notazioni che verranno utilizzate nell'algoritmo in seguito:

- S_{xy} è l'area centrata nel punto z_{xy}
- z_{min} è l'intensità minima nell'area considerata;
- z_{max} è l'intensità massima nell'area considerata;
- z_{med} è l'intensità mediana nell'area considerata;
- z_{xy} è l'intensità del pixel nella posizione (x,y);
- S_{max} è la grandezza massima dell'area consentita.

Algorithm 1 Adaptive Median Filter

$A1 \leftarrow z_{med} - z_{min}$
$A2 \leftarrow z_{med} - z_{max}$
if A1 > 0 and A2 < 0 **then** ▷ Apply the median filter
    $B1 \leftarrow z_{xy} - z_{min}$
    $B2 \leftarrow z_{xy} - z_{max}$
    if B1 > 0 and B2 < 0 **then**
        output z_{xy}
    else
        output z_{med}
    end if
else
    increase S_{xy}
end if
if $S_{xy} \leq S_{max}$ **then**
    goto start (recomputing min, max and med)
else
    output z_{med}
end if

7.1.4 Rumore additivo periodico

Come già accennato in precedenza, il rumore periodico viene invece processato nello spazio delle frequenza. L'idea principale è quella di applicare dei filtri bandreject o Notch per rimuovere le stesse

dall'immagine.

Come visto, la rimozione di queste frequenze viene spesso fatta per mezzo di filtri che hanno un profilo Gaussiano. Questo però ci crea dei problemi siccome un filtro moltro stretto nel dominio delle frequenze (con σ^2 piccolo) corrisponderà ad una Gaussiana molto larga nel dominio dello spazio (con $1/\sigma^2$). Per risolvere il problema l'idea è quella di isolare le componenti da rimuovere, moltiplicarle per un determinato peso e sottrarle dall'immagine rumorosa. Questo approccio è implementato dall'*optimal Notch filtering*. Anziché rimuovere direttamente le componenti di frequenza che causano rumore, l'approccio consiste nell'estrarre le componenti tramite un filtro Notch pass, per poi ottenere una stima del rumore facendone l'antitrasformata:

$$\eta(x,y) = \mathscr{F}^{-1}[H_{np}(u,v)G(u,v)]$$

A questo punto, si sottrae il rumore ottenuto, ma siccome è solo una stima, si vouole fare attenzione a sottrarlo solo nei punti in cui si è più sicuri che si tratti di rumore, per cui lo si pesa per un'opportuna funzione che ha il compito di minimizzare gli effetti dei componenti che non sono presenti in $\eta(x,y)$.

$$\hat{f}(x,y) = g(x,y) - w(x,y)\eta(x,y)$$

dove $w(x,y)$ è la funzione di *"pesatura"*. Tale funzione in un certo senso rende adattivo il filtro, poiché decide quando e di quanto attivare il filtro Notch. Il criterio per decidere se applicare o meno il filtro è quello di minimizzare la varianza all'interno dell'intorno locale $(2a+1) \times (2b+1)$. L'idea è che le immagini in generale hanno una varianza locale bassa per ogni (x,y), per cui si tratta di un criterio prioristico (si sta di fatto facendo una stima a priori sulla distribuzione delle immagini). Per arrivare alla formulazione della funzione w bisogna risolvere di fatto un problema di ottimizzazione. Siano:

- S_{xy} l'intorno di dimensioni $(2a+1) \times (2b+1)$, centrato in (x,y);
- $\mu(x,y) = \frac{1}{(2a+1)(2b+1)} \sum_{(x,y)\in S_{xy}} \hat{f}(x,y)$ il valor medio locale;
- $\sigma^2(x,y) = \frac{1}{(2a+1)(2b+1)} \sum_{(x,y)\in S_{xy}} \hat{f}(x,y) - \mu(x,y)$ la varianza locale;

Si sostituiscono inizialmente all'interno della definizione di σ^2 tutte le occorrenze di $\hat{f}$ con la definizione data poch'anzi. A questo punto, per poter ottimizzare questa formulazione si va a calcolare la derivata di w, per poi scegliere i punti di minimo in essa. Per "portar fuori" w dalla sommatoria, si fa l'assunzione che sia costante nella finestra S_{xy}. Calcolando quindi la derivata $\frac{\partial \sigma^2}{\partial w}$ e ponendola $= 0$ possiamo trovarne il minimo. I calcoli sono ommessi, si riporta solo l'ottimo:

$$w(x,y) = \frac{\overline{g(x,y)\eta(x,y)} - \overline{g}(x,y)\overline{\eta}(x,y)}{\overline{\eta^2}(x,y) - \overline{\eta}^2(x,y)}$$

dove $\bar{\cdot}$ è la **media** calcolata rispetto alla finestra S_{xy}.

7.2 Degradazione

In riferimento allo schema proposto ad inzio capitolo, ci si concentra ora su metodi per la rimozione della degradazione della funzione H. In questo caso si ipotizza quindi che non ci sia nessun tipo di rumore per non complicare il problema.

$$g(x,y) = H(f(x,y)) + \cancel{\eta(x,y)}$$

Ipotizziamo inoltre che H sia un filtro lineare (rispetta il principio di sovrapposizione degli effetti) e posizione invariante (l'output non cambia in base alla posizione), per cui ci si concentrerà su un sottoinsieme delle possibili funzioni di degradazione. Questa restrizione permette di riscrivere l'espressione di $g(x,y)$, siccome è dimostrabile che applicare un filtro lineare e posizione invariante corrisponde a fare la convoluzione tra H e f. Alla luce di questo fatto, è possibile quindi riscrivere la relazione precedente:

$$g(x,y) = f(x,y) \star h(x,y)$$

L'approccio principale è quindi quello di cercare di trovare un modello per la funzione di degradazione, costruirne il filtro inverso e applicarlo all'immagine. La restoration è anche detta per questo motivo *deconvoluzione*. Ci sono true approcci principali per determinare la funzione di degradazione: per *osservazione*, per *sperimentazione* e per *modellazione*. Nel primo caso, l'idea è quella di prende l'immagine corrotta e prenderne un'area specifica s che può essere facilmente restaurata (manualmente come desideriamo che sia). Considerando successivamente sia la trasformata di questa area $G_s(u,v)$, che la trasformata di quella non restaurata $\hat{F}_s(u,v)$, la stima della funzione di degradazione sarà data dal rapporto delle stesse (siccome $G(u,v) = H(u,v)F(u,v)$):

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

Se la degradazione è realmente lineare e posizione invariante allora la restaurazione è modellata correttamente e l'applicazione del filtro a tutta l'immagine restaurerà tutta l'immagine, sfruttando così l'invarianza posizionale.

La seconda idea è quella di acquisire un'immagine (template) con un singolo punto illuminato ad alta intensità. Ci si aspetterebbe un singolo pixel acceso, però si ottiene tipicamente un'ombra (dovuta al difetto di acquisizione H). A questo punto, la TF dell'immagine template è semplicemente A (che è l'ampiezza del punto acceso), per cui, sfruttando la relazione come fatto precedentemente si ottiene:

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

La terza idea consiste nel andare ad ottenere un modello del fenomeno fisico di riferimento per mezzo di

un modello matematico. Un approccio di questo tipo è la modellazione del movimento delle immagini. Avendo la traiettoria dello spostamento nel tempo in ogni direzione descritto dalle funzioni $x_0(t)$ e $y_0(t)$ (ottenendole ad esempio tramite accelerometro), è possibile modellare l'acquisizione come:

$$g(x,y) = \int_0^T f(x - x_0(t), y - y_0(t))dt$$

cioè non è nient'altro che *media* delle acquisizioni che fa il sensore durante il tempo di acquisizione T. Tramite questa modellazione si ottiene che la funzione di degradazione non è nient'altro che un filtro passa basso, cioè un filtro che fa blurring in funzione della *velocità*, il che coincide anche con l'intuizione.

> (*per la trattazione matematica dettagliata si rimanda alle slides*)

Dal punto di vista pratico, un filtro che modella il motion blurring è un filtro composto da tutti 0 e 1 verso la direzione del blurring. Ad esempio un filtro $N \times M$ che modella il blurring orizzontale potrebbe essere il seguente

$$h(x,y) = \begin{cases} 1 & \text{if } (x,y) \in [(0, N-1), (0, M-1)] \\ 0 & \text{otherwise} \end{cases}$$

La grandezza $N \times M$ è determinata inoltre dalla velocità direzionale relativa.

7.2.1 Filtro Inverso

Una volta ottenuta una modellazione sufficientemente buona del filtro di degradazione, è possibile utilizzare successivamente la relazione usata in precedenza per ottenere l'immagine restaurata:

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

Questa soluzione però evidenzia due problemi principali, introdotti dalla divisione per il filtro H. Il primo è la divisione per 0: il filtro potrebbe avere valori vicino a 0 per cui renderebbe la procedura numericamente instabile. Il secondo riguarda invece il rumore, esso difatti è stato ommesso, ma è pur sempre presente in un modello realistico di restoration. La divisione del rumore per valori piccoli di H, aumenterebbe significativamente lo stesso di conseguenza, ne consegue che il filtro inverso **può** aumentare il rumore. La soluzione è evitare di applicare il filtro inverso nelle frequenze in cui diventa instabile.

7.2.1.1 Filtro di Wiener

Un modo che permette di mitigare il problema del rumore discusso in precedenza è mediante l'utilizzo di un filtro di Weiner. Il filtro di Wiener è un filtro ottimo rispetto al

MSE (rispetto all'immagine ground truth) per invertire i sistemi lineari. L'obiettivo per trovare un filtro ottimo è quello di progettare un filtro h_i tale per cui

$$h_i = \arg\min_{h_i} E\left\{(f - \hat{f})^2\right\}$$

risolvere questo problema di ottimizzazione è possibile e il risultato è il seguente:

$$H_i(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + S_\nu(u,v)/S_f(u,v)}$$

dove:

- $H^*(u,v)$ è il *complesso coniugato* di H;
- $S_\nu(u,v) = |N(u,v)^2|$ spettro di potenza del rumore;
- $S_f(u,v) = |F(u,v)^2|$ spettro di potenza dell'immagine.

Per conoscere lo spettro di potenza dell'immagine originale bisogna conoscere la distribuzione dell'energia dello spettro dell'immagine. Siccome non si conosce a priori, è sufficiente una stima dello stesso. Lo stesso vale per il rumore.

Il rapporto tra i due spettri di potenza nella formula equivale al rapporto $\frac{1}{SNR} = NSR$. Più è grande, più rumore e meno segnale sarà presente in quella banda, per cui il filtro inverso dovrà essere applicato in minore intensità per evitare di aumentare il rumore.

Se analizziamo la formula, nel caso migliore (in cui SNR) tende a ∞, si ha che l'intero rapporto vale 0, per cui il filtro sarà semplicemente applicato così com'è. D'altra parte, nel caso in cui SNR tenda a 0, allora il rapporto tenderà a ∞, per cui il valore dell'intero filtro sarà molto basso, cancellando di fatto le frequenze dominate dal rumore (siccome è dominato dall'alto valore del denominatore).

Quando non si hanno indizi sullo spettro del rumore, si assume che ci si trovi in presenza di un rumore bianco, per cui si pone lo spettro a σ^2. Nel caso invece dell'immagine, come già accennato si può sia utilizzare un modello, oppure se non si conosce si utilizza semplicemente una costante.

7.3 Metriche di qualità

Ottenere delle metriche di qualità per la restoration è un problema ancora parzialmente irrisolto. Supponendo di avere sia l'immagine originale $f(x,y)$ (ground thuth) che l'immagine restaurata $\hat{f}(x,y)$, un modo potrebbe essere quello di calcolare l'errore tra le due

$$e(x,y) = f(x,y) - \hat{f}(x,y)$$

che però non è una metrica. Per avere una metrica per valutare l'errore si utilizza il *Mean Square Error* (MSE):

$$MSE = \frac{1}{W \cdot H} \sum_{(x,y) \in W \times H} e(x,y)^2$$

Questa metrica non tiene conto della qualità soggettiva, ma solo di una qualità in senso generale. In realtà nelle immagini si utilizza di solito il rapporto segnale/rumore:

$$SNR = \frac{S}{MSE}$$

dove $S = \frac{1}{W \cdot H} \sum_{(x,y)} f(x,y)^2$. Un'altro rapporto è il rapporto segnale di picco/rumore:

$$PSNR = \frac{P^2}{MSE}$$

dove $P = max_{(x,y)} f(x,y)$. Di solito valori $PSNR \approx 30dB$ sono ritenuti accettabili.

Deve essere chiaro che comunque queste metriche non sempre hanno una correlazione vera e propria con le qualità soggettive per un umano.

8 Visione Artificiale con Reti Neurali

In questo capitolo ci si concentrerà sulla visione artificiale, interamente per mezzo di approcci machine learning, nello specifico mediante l'impiego di reti neurali profonde. Inoltre, ci si concentrerà su problemi di apprendimento supervisionato come la **classificazione** o la **regressione**.

Un esempio di problema di apprendimento supervisionato nel campo della computer vision potrebbe essere quello di classificare con una determinata classe un oggetto presente in un immagine. Ad esempio, se un immagine contiene un gatto oppure un cane. Il dataset di input per un problema di apprendimento supervisionato è rappresentato da una matrice $\mathbf{X}$ con tante righe quante istanze presenti nel dataset e tante colonne quante *features* del problema. Inoltre si indicano le classi etichettate delle varie istanze con il vettore $\mathbf{t}$. Si supporrà inoltre che i dati siano normalizzati, cioè tali per cui abbiano media $\mu = 0$ e varianza $\sigma = 1$. Formalmente, per ogni vettore colonna $\mathbf{x}_i$, si applica la seguente formula:

$$\mathbf{x}_i = \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)}$$

8.1 Percettrone

La rete più semplice che si può immaginare è quella composta da un singolo neurone. Esso ha una serie di input x_i, una serie di parametri (*pesi*) w_i, un *bias* additivo e una funzione di attivazione *non-lineare* e differenziabile g. Si prende ispirazione dal funzionamento biologico di un neurone del sistema nervoso, che consiste nel rappresentare un neurone come una funzione che somma i semplicemente vari segnali in input (i neuroni ai quali è connesso) pesandoli per la forza di connessione che ha il neurone con essi. Questa somma poi viene fatta passare per una funzione di attivazione che indica la risposta del neurone, siccome nei neuroni biologici esiste una soglia di attivazione.

Mettendo insieme questi componenti, otteniamo che l'output del neurone è rappresentabile come

$$y = g\left(b + \sum_i \mathbf{w}_i \mathbf{x}_i\right)$$

L'apprendimento di questa semplice rete consiste inizialmente nell'impostare i valori dei pesi $\approx$ in modo completamente casuale. Successivamente, si calcola per ogni istanza di input $\mathbf{x}_i \in X$ (*vettore riga*) l'output y_i corrispondente del neurone. A questo punto si calcola un errore E differenziabile t_i, come il Mean Square Error $E_i(y_i, t_i) = (y_i - t_i)^2$. Siccome sia l'errore che l'output del neurone sono differenziabili, è possibile calcolare il *gradiente* dell'errore $\frac{\partial E}{\partial \mathbf{w}}$. Il calcolo del gradiente può essere fatto semplicemente per mezzo della regola di derivazione a catena. Difatti, espandendo la definzione di E

si ottiene:

$$E(y_i, t_i) = \left[g \left(\underbrace{b + \sum_j \mathbf{w}_j \mathbf{x}_i j}_{z_i} \right) - t_i \right]$$

Quindi, per la formula di derivazione a catena il gradiente corrispondente sarà:

$$\frac{\partial E}{\partial \mathbf{w}_i} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_i} \frac{\partial z_i}{\partial \mathbf{w}_i}$$

dove z_i è indicato nella formula precedente.

Una volta ottenuto il gradiente, per ogni $\mathbf{w}_i$ si applica una regola di update secondo il principio di *discesa del gradiente*. Essenzialmente questo principio permette di minimizzare l'errore modificando i pesi in modo da muoversi verso la direzione opposta all'aumento più grande dell'errore (e quindi, di conseguenza, verso la massima diminuizione). Dal punto di vista formale, la regola di update è la seguente:

$$\mathbf{w}_i' = \mathbf{w}_i - \eta \frac{\partial E}{\partial \mathbf{w}_i}$$

in cui $\mathbf{w}_i'$ è il nuovo valore del peso i-esimo e η è un iperparametro dell'ottimizzatore che indica il *learning rate* (o *step-size*). Questo passaggio viene re-iterato per tutte le istanze i del dataset, completando di fatto un *epoca*.

8.2 Fully Connected Layer

Il percettrone, pur rappresentando un fondamentale punto di partenza nel campo delle reti neurali, presenta alcune limitazioni. Come visto, difatti, rappresenta solo un output scalare y. È possibile concatenare diversi percettroni per ottenere un FCL (*Fully Connected Layer*). In caso se ne concatenino m, si otterranno m vettori $\mathbf{w}$ di parametri, per cui vengono rappresentati tutti in una matrice $\mathbf{W}$ con dimensioni $m \times n$ dove m è il numero di outputs (*percettroni*) e $(n + 1)$ il numero di inputs (dove il $+1$ può essere omesso e rappresenta il *bias*).

8.3 Multilayer Perceptron

Un layer completamente connesso permette di "mappare" le features in input su un nuovo spazio vettoriale. Concatenare vari layer uno dopo l'altro permette di "processare" l'informazione in input, mappandola in diversi spazi vettoriali. Ciò che la rete fa è essenzialmente apprendere delle features latenti del dataset che possono essere utilizzate per discriminare dei problemi anche non linearmente separabili (poiché esse lo sono). In un problema di multi-classificazione, l'ultimo layer sarà composto

da tanti neuroni quanti sono le classi che fanno parte del problema di classificazione, poiché i vari neuroni fungeranno da classificatori per la stessa classe. Inoltre, in questo tipo di problemi si applica una funzione *softmax* sul vettore dei neuroni di output $\mathbf{z}$, per cui l'output della rete sarà

$$\mathbf{y} = \frac{e^{\mathbf{z}}}{\sum_{k=1} e^{z_k}}$$

La probprietà della softmax è che crei una *distribuzione* sulle varie componenti del vettore, che in questo caso sono le varie classi. Per questo è possibile calcolare una metrica di errore che funziona meglio per problemi di classificazione, cioè la *cross-entropy loss*:

$$\mathscr{L}(\mathbf{y}, \mathbf{t}) = -\sum_{j} t_j log(y_j)$$

8.3.1 Overfitting

Più aumenta la complessità del modello (numero di layer e neuroni), più esso sarà in grado di adattarsi al il dataset di train iniziale. Bisogna però fare attenzione al tradeoff bias-variance. Un classificatore troppo coomplesso si adatterà troppo ai dati, provocando *overfitting* (cioè un fenomeno in cui il classificatore modella così bene i dati di input che non è in grado di generalizzare). Tipicamente si introduce una regolarizzazione dei pesi in normal $\mathscr{L}^2$ in modo tale da penalizzare le soluzioni con valori dei pesi troppo grandi.

In un problema di machine learning, per capire se ci si trova di fronte ad un modello overfitted, si divide il dataset originale in dataset di train e dataset di test. Si apprende il modello sullo split del dataset di train e si testa su quello di test. L'overfit si verifica quando l'errore di train è più alto dell'errore di test.

8.4 Reti Feed Forward e Immagini

Fino ad ora si è supposto che le features di un problema di classificazione fossero delle qualità intrinseche che potessero rappresentare le varie istanze, come ad esempio il peso, l'altezza, la grandezza ecc. Viene naturale pensare che le immagini stesse possano essere viste come un insieme di features, cioè ogni valore di pixel rappresenterebbe di per se una feature. In questo senso, un'immagine $N \times M$ può essere rappresentata in un vettore lungo $N \cdot M$, con i valori delle features corrispondenti ai valori di intensità dei pixel. Nel dataset MNIST, composto da immagini 28×28 di cifre scritte a mano ed etichettate con la classe corretta $(0 - 9)$, le immagini sono per l'appunto espresse in vettori lunghi $28 \cdot 28 = 768$ elementi.

8.4.1 LeNet300

LeNet300 fu un'architettura introdotta alla fine degli anni 80 per fare riconoscimento di cifre scritte a mano e classificarle (*handwritten digit recognition*). La rete era una rete completamente connessa (composta da soli layer fully connected), la cui architettura era composta da:

- Input layer di dimensioni 768 (il che è ragionevole poiché l'input sono immagini lunghe 768);
- Layer hidden di dimensioni 300;
- Layer hidden di dimensioni 100;
- Layer di output di dimensioni 10 (il numero di classi che coincide con il numero di cifre).

Nel paper originale venivano fatte diverse esplorazioni di parametri dell'architettura, ma quella con performare migliori fu appunto quella presentata precedentemente. Il problema di questo approccio è che questo tipo di rete non permette di sfruttare la correlazione spaziale dei pixel che esiste in un'immagine qualsiasi. Questo perché le immagini sono trattate semplicemente come dei vettori, rimuovendo di fatto questa struttura bidimensionale e di conseguenza l'informazione spaziale che ne consegue.

8.5 Reti Neurali Convoluzionali

Le reti convoluzionali sono un'evoluzione di questo approccio, nate per l'appunto per risolvere questa problematica e sfruttare la correlazione spaziale dei pixel nelle immagini.

Gli oggetti nelle immagini sono caratterizzati da features spaziali ad alto livello, come ad esempio *bordi*, *angoli*, ecc.. L'idea alla base delle reti convoluzionali è quella di far si che la rete possa apprendere queste features locali. La rete quindi utilizza dei *features detectors*, che data in input un'immagine rappresentata per pixel, sottolineano le features significative. Questa operazione viene implementata per mezzo di una convoluzione, in cui i valori del filtro vengono appresi dalla rete durante il processo di apprendimento. Il risultato dell'applicazione di questo filtro su un'immagine, produce una *feature map*, cioè una rappresentazione dell'immagine nello spazio delle feature che sono individuate dal filtro.

8.5.1 Layer Convoluzionale

L'operazione di convoluzione è implementata da un Neurone Convoluzionale, che non è nient'altro che un neurone in cui gli sono input vincolati spazialmente, cioè sono connessi solamente ad un sottoinsieme dei neuroni a cui sono connessi in input. Ovviamente, i valori del filtro (*pesi*) possono essere appresi attraverso la backpropagation. Formalmente l'output del neurone i, j viene calcolato

come:

$$\mathbf{y}_{i,j} = \sum_{l=0}^{L} \sum_{k=0}^{K} \mathbf{w}_{l,k} \cdot \mathbf{x}_{(S \cdot i)+l,(S \cdot j)+k}$$

Questa formula implementa un kernel di dimensioni $L \times K$ e stride S. La dimensione del kernel è un iperparametro che definisce l'estensione spaziale della connettività dei neuroni rispetto al layer precedente, e viene chiamato anche *receptive field* (campo recettivo). D'ora in poi si supporrà che la dimensione del kernel sia sempre $F \times F$. Lo striding indiva invece di quanto sono spaziati tra di loro i kernel nel layer di input. Siccome la feature map risultante è più piccola in dimensioni del layer in input, si applica uno *zero-padding* ($P = (F-1)/2$), in modo tale che l'output preservi le dimensioni del layer in input. Ovviamente è possibile applicare questa tipologia di rete anche ad immagini RGB, semplicemente applicando un filtro per ogni canale e poi si sommandoi risultati in modo da accorpare l'informazione delle varie features apprese nei vari canali.

Si noti che il numero di parametri è più ristretto rispetto ad un layer fully connected, siccome ogni neurone è connesso solo ad un sottoinsieme ristretto di neuroni di input ($F \times F$). Più precisamente, il numero di parametri per un layer convoluzionale è pari a:

$$n_{out} \cdot (S + F^2) \cdot n_{ch}$$

dove:

- n_{out} numero di neuroni convoluzionali;
- n_{ch} numero di canali ($= 1$) per immagini *grayscale* e $= 3$ per immagini RGB;
- F dimensione del kernel ($F \times F$).

Questo minore numero di parametri permette sia di alleggerire molto la fase di apprendimento che di migliorare la generalizzazione del modello risultante.

Si rappresenta quindi un layer convoluzionale con la notaziona $Conv(F, S, P)$, in cui ovviamente F è la dimensione del filtro, S lo stride e P lo zero-padding.

8.5.2 MaxPooling Layer

Siccome le features map estratte dai layer convoluzionali possono essere viste come delle immagini *filtrate*, anche queste immagini filtrate possono essere sotto-campionate proprio come le immagini. Questa operazione permette di ridurre ulteriormente i parametri totali della rete, siccome permette di ridurre la dimensione dell'output, mantenendo però una certa quantità di informazione.

Il layer di max-pooling permette di campionare dei valori dall'immagine secondo il criterio di *massimo*. L'idea è sempre quella di applicare un filtro *non-lineare* che faccia passare solamente il massimo tra tutti gli elementi della finestra definita dal filtro. Anche in questo caso il filtro sarà $F \times F$ e può essere

applicato con un determinato stride S. Il layer di max-pooling si applica tipicamente dopo un layer di convoluzione.

8.5.3 Optical Character Recognition

È possibile usare le reti convoluzionali per fare OCR. Ad esempio, si può aggiungere una classe "undefined" (o background) che non rappresenta nessun carattere. La rete poi può essere applicata a tutte le posizioni dell'immagine attraverso un approccio a *sliding-window*. Per fare questa operazione in modo più efficiente, però, si può sfruttare il dualismo tra layer fully connected e convoluzionale. Essenzialmente, se un layer convoluzionale ha le stesse dimensioni del kernel del suo input, diventa equivalente ad un layer fully connected. La trasformazione dei layer fully connected in convolutional, ha però il vantaggio di poter applicare la rete in parallelo con un approccio sliding window su immagini di grandezza arbitraria. L'output saranno non più una singola classe ma 10 feature maps con la grandezza della finestra (sliding window), in cui i valori di ogni feature map indicano il valore di classe per quel particolare punto in cui può essere posizionata la finestra. Questa accortezza permette di ridurre notevolmente la complessità computazionale richiesta, anzichè applicare per ogni posizione della sliding window la rete.

> Le reti che hanno solo layer convoluzionali e non fully-connected sono ragionevolmente chiamate *fully-convolutional*.

Una volta ottenuti i caratteri si va a fare poi una fase di postprocessing che tramite tecniche statistiche e probabilistiche ri-aggiusta il risultato. Ci sono quindi due possibilità per apprendere una rete in un task di OCR:

1. Apprendere una rete convoluzionale con layer fully connected su singoli caratteri (quindi campioni piccoli), per poi fare un reshape dei layer fully connected in layer convoluzionali, mantenendone i pesi appresi (si prendono dai layer fully connected);
2. Apprendere una rete già *fully-convolutional* direttamente sulle immagini. In questo caso però le immagini devono essere tutte etichettate nelle varie posizioni.

8.5.4 LeNet5

Questa rete fu una delle prime reti convoluzionali in cui si applicava il pattern *convolve-and-pool*. Di seguito verrà utilizzata una versione leggermente modificata di LeNet5. L'input sono immagini 32×32, a cui si applica un layer convoluzionale composto da 6 neuroni convoluzionali $Conv(5, 2, 1)$. In questo modo, vengono così generate in output 6 differenti features map, ognuna in grado di estrarre una particolare feature dall'input. Dopo questo layer ne segue un layer di max-pooling su tutte le features

map risultanti. Si ri-appplicano poi un layer di convoluzione uguale al precedente, questa volta però composto da 16 neuroni, seguito da un layer di max-pooling (2×2). Infine, seguono due layer fully connected, in modo da implementare la parte discriminativa della rete. Riassumendo, l'architettura di LeNet5 è così composta:

- Layer input (immagine padded 32×32);
- Layer di 6 neuroni convoluzionali $Conv(5, 2, 1)$, Output: $6 \times 28 \times 28$;
- Layer di max pool 2×2, Output: $6 \times 14 \times 14$;
- Layer di 16 neuroni convoluzionali $Conv(5, 2, 1)$, Output: $16 \times 14 \times 14$;
- Layer di max pool 2×2, Output: $16 \times 7 \times 7$;
- *(Layer di flatten)*;
- Layer fully connected 784×100;
- Layer fully connected 100×10 (seguito da un layer di *softmax*).

Nel caso invece si voglia considerare la versione fully-convolutional della rete, basta rispettare la regola precedente, per cui bisogna sostituire i layer fully connected con i seguenti layer (in ordine):

- Layer di 100 neuroni convoluzionali $Conv(7, 1, 0)$ (cioè 7×7);
- Layer di 10 neuroni convoluzionali $Conv(10, 1, 0)$ (cioè 1×1).

8.6 Tecniche avanzate di apprendimento

Come visto in precedenza, l'apprendimento delle reti neurali viene fatto per mezzo di un algoritmo di discesa del gradiente. Il *learning rate* è un iperparametro dell'algoritmo di ottimizzazione molto importante, perché può fare la differenza nell'ottenimento di un modello di buona qualità. Scegliere il valore corretto però non è semplice, siccome valori di η molto piccoli, renderebbero lento l'apprendimento e porterebbero spesso ad una configurazione associata ad un minimo locale. D'altra parte, un alto valore di η renderebbe l'apprendimento troppo instabile (provoncando un effetto "bouncing", o di "overshoot" dell'errore intorno al valore minimo).

Tipicamente il learning rate viene ridotto nel tempo in modo da dare inizialmente la possibilità all'ottimizzatore di non finire subito in minimi locali. Col crescere delle iterazioni, però, il learning rate descresce in modo tale da focalizzare la convergenza sul minimo trovato.

Un modo per implementare questa strategia è il cosiddetto *esponential decay*, in cui il learning rate decresce esponenzialmente col crescere delle epoche

$$\eta = \eta_0 \cdot e^{-kt}$$

Questa tipologia di aggiustamento del learning rate permette anche di evitare l'overfitting.