



**Università
degli Studi
di Palermo**

Università degli Studi di Palermo

Visione Artificiale

Docente:

Lo Presti Liliana

Autore:

Castelli Giovanni

Corso di Laurea:

LM-32 Ingegneria Informatica

Anno Accademico 2023/2024

Indice

Indice	2
INTRODUZIONE	11
0 Introduzione	12
ACQUISIZIONE DELLE IMMAGINI	13
1 Acquisizione delle immagini	14
1.1 Introduzione	14
1.2 La luce	14
Lunghezza d'onda	14
Riflessi e rifrazione	15
1.3 L'occhio umano	15
Occhio Umano e Corteccia Visiva	15
Il Sistema Visivo Umano	15
Rods	16
Coni	17
1.4 Cos'è il Colore?	18
Percezione del Colore	18
Luce Acromatica - Rappresentazione dell'Intensità	18
Rappresentazione del Colore	18
Combinazione Additiva dei Colori	18
Luce Cromatica - Rappresentazione del Colore	19
Tonalità e Saturazione	19
1.5 Modelli di Colore	19
Scopo dei Modelli di Colore	19
Spazio Colore RGB	19
Spazio Colore HSI	20
Spazio Colore YCbCr	20
Similitudini tra l'occhio umano e le telecamere	20
1.6 Immagini Digitali	21
Immagini in Scala di Grigi	21
Immagini a Colori	21
Quantizzazione	21
1.7 Video Digitali	21
Caratteristiche	21
Memorizzazione	22
Compressione	22
Codec Video	22
1.8 Dispositivi di Acquisizione delle Immagini	23
Webcam e CCTV	23

Telecamere IP	23
Tipi di Telecamere IP	23
Sistemi di Sorveglianza Industriale	23
Altri Dispositivi di Acquisizione	24
1.9 Rendering	24
Sistemi di Colore Additivo	24
Sistemi di Colore Sottrattivo	24
1.10 OpenCV	24
Introduzione	24
Moduli	25
IMAGE FORMATION	26
2 Image Formation	27
2.1 Formazione dell'Immagine	27
2.2 Imaging	27
2.3 Primitivi Geometrici	28
Punti 2D	28
Punti 3D	28
2.4 Traslazione	29
2.5 Rotazione	29
2.6 Roto-traslazione	29
2.7 Scaling	30
2.8 Trasformazione Affine	31
2.9 Trasformazione Proiettiva	31
2.10 Rotazione in 3D	32
2.11 Rotazione Generica	32
2.12 Proiezioni 3D su 2D	33
Proiezione Ortografica	33
Proiezione Prospettica	33
2.13 Modello della Fotocamera Stenopeica	35
Intrinsici della Fotocamera	35
Corrispondenza tra Pixel dell'Immagine e Punti Proiettati	36
2.14 Matrice di Proiezione M_s	37
2.15 Stima di K	37
2.16 Aggiunta di un nuovo sistema di coordinate	38
2.17 Stima dei parametri della fotocamera	38
Vincolo sulla forma di K	39
2.18 Stima dei parametri della fotocamera	39
2.19 Pattern di Calibrazione	40
2.20 Stima della Matrice della Fotocamera	41
Stima di $K, (R, t)$	42
Alcuni dettagli implementativi	43
2.21 Calibrazione della fotocamera di Zhang	44
Procedura di calibrazione	44
2.22 Distorsione della Lente	48
Distorsione Radiale	48

2.23 Distorsione Radiale	48
Modello Brown-Conrady	48
2.24 Distorsione Tangenziale	49
2.25 Distorsione Radiale e Tangenziale	49
Stima dei Parametri	49
2.26 Lente Fisheye	50
Modello di Proiezione	50
2.27 Formazione dell'Immagine Fotometrica	51
2.28 Fotocamere Digitali	51
2.29 Chip Sensore	52
CCD	52
CMOS	52
2.30 Fattori che Influenzano le Prestazioni dei Sensori di Immagine	53
2.31 Campionamento e Quantizzazione (ADC)	53
2.32 Aliasing	53
Aliasing e Anti-Aliasing	54
2.33 Percezione del Colore	54
2.34 Sistema di Colori Additivi	55
2.35 Sistema di Colori Sottrattivi	55
Descrizione dei Colori	55
Acquisizione di Immagini a Colori	56
IMAGE PROCESSING	58
3 Image Processing	59
3.1 Elaborazione delle Immagini	59
Operatori su Immagini e Operatori di Punto	59
Operatori di Punto per Immagini a Colori	59
Operatori Puntuali	60
Istogrammi	61
Come estrarre un'immagine monocromatica da un'immagine a colori	61
Equalizzazione dell'Iistogramma	61
Relazioni di base tra i pixel: Vicinato	62
Relazioni di base tra i pixel: Adiacenza	62
Componenti Connesse	62
Relazioni di base tra i pixel	63
Filtraggio Lineare	63
Padding	65
Correlazione e Convoluzione vs Filtraggio Spaziale Lineare	65
Filtraggio Lineare Reso Più Veloce	66
Correlation and Convolution	67
A cosa servono i filtri lineari?	67
Smoothing Immagine	68
Cosa succede se applichiamo due volte un filtro di media?	68
Filtri Gaussiani	68
Mascheratura non nitida	70
Filtri Derivati	70

Nitidezza tramite Laplaciano	72
Gradiente di Immagine	73
Operatori di Sobel	73
Filtri Separabili	74
Filtro LoG	75
3.2 Filtri Non Lineari	76
Elaborazione di Immagini Binari	76
Operatori Morfologici	76
Multi-risoluzione	77
Esempio - Rilevamento Pedoni	78
Trasformazioni Geometriche	78
Interpolazione Bilineare	79
3.3 Ripristino delle Immagini	80
Il processo di degradazione/ripristino	80
3.4 Modelli di rumore	80
Alcuni tra i modelli di rumore più comuni nelle applicazioni di elaborazione delle immagini sono:	81
Stima dei parametri del rumore	84
MODEL FITTING AND OPTIMITAZION	86
4 Model fitting and Optimitzation	87
4.1 Interpolazione di Dati Sparsi	87
4.2 Approssimazione di Dati Sparsi	87
4.3 Cosa è f	88
Come trovare i parametri del modello - Intuizioni	88
Approssimazione di Dati Sparsi	88
Funzioni a Base Radiale	89
4.4 Overfitting e Underfitting	92
Compromesso Bias-Varianza	93
Selezione del Modello	95
Selezione del Modello con la Validazione Incrociata (Cross-Validation)	96
4.5 Campo Aleatorio di Markov per la riduzione del rumore	97
Metodi Variazionali	97
4.6 Metodi Variazionali e Minimizzazione dell'Energia	97
4.7 Metodi Variativi per la smoothness	98
4.8 Minimizzazione dell'Energia Discreta	98
4.9 Metodi Variativi per Immagini	99
4.10 Prospettiva Bayesiana	99
4.11 Massima Verosimiglianza a Posteriori	100
4.12 Campi Aleatori di Markov	101
4.13 Campo Aleatorio di Markov come Modello Grafico	101
4.14 Campi Aleatori di Markov Binari	102
4.15 Binary Markov Random Fields and Graph-Cut	104
Min Cut per Immagini Binari	106
Interpretazione 1-D	107

LEARNING	108
5 Learning	109
5.1 Evoluzione della Visione Artificiale	109
5.2 Problemi da Risolvere	109
5.3 Apprendimento	109
5.4 Apprendimento Supervisionato	110
Regressione	110
5.5 Cosa significa "massimizzare l'accordo"?	111
5.6 Pre-elaborazione dei Dati	111
5.7 Approcci di Apprendimento Supervisionato Popolari	112
Nearest Neighbor (NN) e KNN	112
Classificazione Bayesiana	114
Regressione Logistica	115
classificatore lineare binario	115
Support Vector Machines	116
Alberi Decisionali	117
5.8 Approcci di Apprendimento Non Supervisionato Popolari	117
5.9 Riconoscimento Visivo	117
FACES	119
6 Faces	120
6.1 Face Processing	120
Face Processing	121
Face Detection – Approccio Classico	121
Haar-like Features	122
Integral Image (summed area table)	124
6.2 Face Detection - Viola-Jones	125
6.3 Riconoscimento Facciale - Approcci Classici	127
6.4 EigenFaces - Turk Pentland	128
6.5 Face Space	129
6.6 EigenFaces and Face Recognition	129
6.7 Forma del Viso e Orientamento della Testa	130
6.8 Modelli Statistici di Forma	130
6.9 Active Shape Models	131
6.10 Modelli di Aspetto Attivi (AAM)	131
6.11 Applicazioni dei Modelli di Aspetto Attivi (AAM)	132
TECNICHE DI COMPUTER VISION MODERNE	133
7 Tecniche di Computer Vision moderne	134
7.1 Reti Neurali Profonde (Deep Neural Network)	134
Panoramica Storica delle Reti Neurali	134
7.2 Addestramento End-to-End (Apprendimento Supervisionato)	134
7.3 Neurone	135
Significato Geometrico del Neurone	136

Apprendimento dei parametri del Neurone	137
Minimizzazione	138
7.4 Neurone come Rilevatore di Caratteristiche	139
7.5 Perceptron e Regola Delta	142
Regola del Perceptron	142
Minimizzazione dell'Errore – Regola Delta	142
7.6 Capacità di un Neurone	142
Aumentare la Capacità del Classificatore	143
7.7 Rete Multi-Neurone	143
Seconda Strategia	143
7.8 Strati	143
Esempio	144
7.9 Lavorare con vettori di input multipli - il batch	144
Singolo Strato vs Multi-Strato vs Rete Neurale Profonda	145
7.10 Perceptron Multistrato - Rete Feed Forward	145
7.11 Propagazione in Avanti (Forward Propagation)	147
Propagazione in Avanti – Esempio Numerico	147
7.12 Funzioni di Attivazione	148
7.13 Funzione Softmax	149
7.14 Libreria TensorFlow	149
Classi Principali di TensorFlow-Keras	150
7.15 Functional API	150
7.16 Verso l'apprendimento...	151
7.17 Funzione di Perdita	151
7.18 Binary Cross-Entropy	152
7.19 Cross-Entropy	152
7.20 Backpropagation (Hinton – 1986)	152
7.21 Grafico Computazionale	157
7.22 Grafico Operativo (da TensorBoard)	157
7.23 Ispezione dei Risultati da fit	157
7.24 Regolarizzazione tramite p-norma	157
Regolarizzazione tramite p-norma in TensorFlow	158
7.25 Regolarizzazione tramite Data Augmentation	158
7.26 Regolarizzazione tramite Dropout	159
7.27 Regolarizzazione tramite Batch Normalization	159
Regolarizzazione tramite Batch Normalization in TensorFlow	159
7.28 Ottimizzatori	160
Esempio Completo	161
RETI CONVOLUZIONALI	163
8 Reti convoluzionali	164
8.1 Convolutional Neural Network	164
8.2 Layer Convoluzionali	164
8.3 Convoluzione 1D	166
8.4 Layer Convoluzionale vs Layer Denso	166
8.5 Layer Convoluzionale – Altri Parametri	167

8.6 Grouping	168
8.7 Layer Convoluzionale in Tensorflow.keras	169
8.8 CNN in Tensorflow	169
8.9 Pooling	170
8.10 Pooling in TensorFlow	171
8.11 Max-Pooling e Backpropagation	171
8.12 Unpooling	172
8.13 Convoluzione Trasposta - Deconvoluzione	172
8.14 LeNet-5	173
8.15 CIFAR-10	173
8.16 Fashion-MNIST	173
8.17 AlexNet	173
8.18 ImageNet	174
8.19 VGG	175
8.20 GoogLeNet e Inception	175
8.21 GoogLeNet	176
8.22 ResNet	177
8.23 Più sui Blocchi Residuali	177
8.24 Blocchi Residuali e Backpropagation	177
8.25 DenseNet	178
8.26 MobileNet	179
8.27 Migliorare l'Efficienza	179
8.28 Uso di Modelli Pre-addestrati – Fine Tuning	180
8.29 Manifold Learning	180
8.30 Visualizzazione di Pesi e Attivazioni	181
8.31 Zeiler e Fergus (2014)	181
8.32 Qual è l'effetto dell'apprendimento dei kernel convoluzionali?	183
8.33 GradCam	184
8.34 Deep Learning per Volumi e Video	184
8.35 TimeDistributed in TensorFlow	185
8.36 3D-Conv	185
8.37 Reti Neurali Ricorrenti (RNN)	186
8.38 Long Short-Term Memory (LSTM)	187
8.39 Memoria a lungo breve termine (LSTM)	187
Variabili	188
Funzioni di attivazione	188
Allenamento	188
ConvLSTM	189
Autoencoder	189
Autoencoder - Addestramento	190
Esempio: Image Denoising	191
Encoder-Decoder	193
U-Net	193
Reti Multi-Rami	193
Rete Siamese	193
Rete Siamese - Funzioni di perdita	194
Generatore di Dati	195

Early Stopping	196
OBJECT DETECTION VIA DEEP LEARNING	197
9 Object Detection via Deep Learning	198
9.1 Rilevamento degli Oggetti	198
Rilevamento dei Pedoni – Un Caso Speciale	198
In un SVM Lineare	199
Il Mio Rilevatore è un Buon Rilevatore?	200
Intersezione su Unione (IoU)	200
Soppressione Non Massima	201
Precisione - Richiamo	201
Curva Precisione-Richiamo e AP	202
Rilevatori di Oggetti Moderni	202
Proposta di Regione	203
R-CNN Basato su Regione (R-CNN) 2014	204
R-CNN Veloce (Fast R-CNN) 2015	204
Pooling della Regione di Interesse (ROI Pooling)	205
9.2 Faster R-CNN, 2015	206
9.3 Ancoraggi e la loro Rappresentazione	207
9.4 RPN (Rete di Proposte di Regioni)	208
9.5 Mappa di Caratteristiche Singola vs. Più Mappe di Caratteristiche da una Piramide di Immagini	209
9.6 Piramide delle Mappe di Caratteristiche	209
9.7 Rete a Piramide di Caratteristiche	210
TensorflowHub	210
Faster RCNN in Tensorflow	211
Confronto Qualitativo: HoG Detector vs Faster RCNN	212
Segmentazione di Istanza e Mask RCNN	212
Definizione della Maschera	213
Output di Mask-RCNN	213
Testata di Faster R-CNN con un Backbone ResNet (res5) a cui è aggiunto un ramo di maschera.	214
RoIAlign – Mask RCNN	214
Architettura di Mask-RCNN	215
Addestramento di Mask-RCNN	216
9.8 Mask-RCNN per il Rilevamento della Posizione Umana	216
9.9 Detectron	217
9.10 Model Zoo – Mask RCNN	218
Mask-RCNN su Model Zoo	218
9.11 Two Stage vs Single Stage	218
9.12 SSD - 2016	218
9.13 SSD	219
9.14 Architettura SSD	220
9.15 YOLO – You Only Look Once, dal 2015	220
9.16 YOLOv1 (Redmon, Divvala, Girshick, Farhadi, 2016)	221
9.17 YOLOv2	222

9.18 YOLOv2 – lavorare con gli anchor	222
9.19 YOLOv3	223
9.20 YOLOv4, 2020	224
9.21 YOLOv5-YOLOv8 passando per YoloX	224
9.22 RetinaNet	225
TRACKING	227
10 Tracking	228
10.1 Tracciamento Visivo	228
Come inizializzare un tracker?	228
Movimento per differenza tra frame	228
Modellazione di Primo Piano e Sfondo	230
Come inizializzare un tracker?	232
Quali sono gli spunti importanti per eseguire il monitoraggio?	233
10.2 Tracking e telecamere	233
10.3 Tracking Visivo di Oggetti	234
Esempio di SOT: Tracking-by-Detection	234
GOTURN	235
Esempio di SOT: MDNet	235
MDNet e il Tracking di Oggetto Singolo (Sot)	236
10.4 Tracking di Oggetti Multipli	236
10.5 Grafico Bipartito	237
Matching in un Grafico Bipartito	237
Problema di Assegnazione nel Tracking	238
10.6 SORT	239
DeepSort	239
10.7 Come valutare il nostro tracker?	240

INTRODUZIONE

Introduzione

0

La visione Artificiale è un campo dell'Intelligenza Artificiale che permette ai computer di ricavare informazioni da immagini digitali, video o altri input visivi e intraprendere azioni o formulare segnalazioni sulla base di tali informazioni

ACQUISIZIONE DELLE IMMAGINI

Acquisizione delle immagini

1.1 Introduzione

Alla base della computer vision vi sono chiaramente dei dispositivi che hanno la capacità di acquisire lo scenario del mondo reale e trasformarlo in dati digitali da poter elaborare per mezzo di un calcolatore secondo un determinato scopo (nel caso specifico, la computer vision). Tali sono detti **dispositivi di acquisizione immagini**. Esempi di dispositivi di acquisizione immagini sono gli scanner, le fotocamere, le videocamere.

In generale, il principio di funzionamento di dispositivi come le fotocamere e le videocamere consiste nel catturare la **luce** dell'ambiente circostante e trasformarla in segnali elettronici da appunto processare. Le videocamere hanno in aggiunta la capacità di catturare più immagini in un secondo.

1.2 La luce

La **luce** è una radiazione elettromagnetica che può essere percepita dall'occhio umano sulla base di due caratteristiche principali:

- **Intensità:** indica la potenza emessa dalla sorgente luminosa in una determinata direzione;
- **Lunghezza d'onda:** inversamente proporzionale alla frequenza della luce, permette di classificarla nelle categorie visibili e non visibili.

Per esempio, la luce percepibile dall'occhio umano ha una lunghezza d'onda compresa tra i **400 nm** e i **700 nm**. Infatti, i raggi ultravioletti (la cui lunghezza d'onda è compresa tra i **100 nm** e i **400 nm**) rientra tra le categorie di luce non visibili.

Gli umani non percepiscono la luce allo stesso modo, sebbene sia noto appunto il range di lunghezze d'onde visibili.

Lunghezza d'onda

La **lunghezza d'onda** caratterizza il **colore** della luce, in particolare:

- Al **minimo** valore percepibile della lunghezza d'onda (400 nm) corrisponde il colore **blu**;
- Al **massimo** valore percepibile della lunghezza d'onda (700 nm) corrisponde il colore **rosso**.

Riflessi e rifrazione

Nel vuoto, la luce si muove come una linea retta, tuttavia può **riflettere** in direzioni diverse sulla base della superficie e gli ostacoli che incontra durante il percorso. Per esempio:

- ▶ Nel caso di superfici **opache** può riflettere in più direzioni a seconda della scabrosità di esse, causando la comparsa di **ombre**;
- ▶ Nel caso di superfici **riflettenti**, la luce riflette in una direzione specifica;

La luce può inoltre passare attraverso alcuni materiali, comportando tuttavia un cambio di velocità qualora la densità del materiale sia diversa. Questo cambio di velocità comporta altresì un cambio di direzione e, di conseguenza delle distorsioni. Questo cambio di direzione è detto **rifrazione**.

1.3 L'occhio umano

Occhio Umano e Corteccia Visiva

- ▶ L'occhio ha una lente che proietta il mondo 3D su un'immagine 2D sulla retina
- ▶ Parte dell'elaborazione dell'immagine viene effettuata sulla retina per ridurre l'immagine. L'immagine ridotta viaggia attraverso il nervo ottico al Nucleo Genicolato Laterale, che la invia alla corteccia visiva primaria
- ▶ La corteccia visiva è composta da diverse aree (V1, V2, ...) ciascuna specializzata in compiti specifici (analisi delle forme, colore, texture, movimento...)
- ▶ Sappiamo quale parte del cervello risolve un compito ma non ancora come. Sappiamo che i neuroni lo fanno ma non sappiamo come sono collegati

Il Sistema Visivo Umano

- ▶ L'occhio umano ha approssimativamente la forma di una sfera, con un diametro medio di circa 20 mm. Tre strati circondano l'occhio:
 - Lo strato più esterno è composto da:
 - * la cornea (parte trasparente e anteriore)
 - * la sclera (parte opaca e posteriore)
 - Lo strato intermedio consiste di:
 - * coroide (fortemente pigmentata e ricca di vasi sanguigni)
 - * corpo ciliare, che controlla la forma della lente per vedere oggetti a diverse distanze (mettendo a fuoco)
 - * iride, un diaframma che si contrae e si espande per controllare la quantità di luce che entra nell'occhio. L'apertura centrale dell'iride è la pupilla

- Lo strato più interno è la retina, contenente le cellule fotosensibili (in un certo senso, il piano dell'immagine)
- Per mettere a fuoco oggetti posti a varie distanze, è necessario modificare la distanza tra la lente e il piano dell'immagine
- In una fotocamera, la lente ha normalmente una lunghezza focale fissa. Nell'occhio umano è vero il contrario: la distanza tra la lente e il piano dell'immagine (la retina) è fissa. Le variazioni della lunghezza focale si ottengono modificando la curvatura delle superfici della lente cristallina
- Sono i muscoli ciliari che danno alla lente una forma più o meno piatta, permettendo così di mettere a fuoco oggetti più o meno distanti
- La lente cristallina assorbe circa l'8% della luce entrante nell'occhio, ma non allo stesso modo sull'intero spettro visibile. L'assorbimento è maggiore nella regione blu

L'occhio umano è sensibile alla radiazione elettromagnetica nello spettro visibile, tra circa 360 nm e 800 nm (in realtà la sensibilità dell'occhio è largamente ridotta nelle bande 360-410 nm e 720-800 nm):

L'energia luminosa viene assorbita dai pigmenti fotosensibili dei recettori: coni e bastoncelli

- Quando un oggetto o una scena è messa a fuoco, un'immagine viene proiettata sulla retina
- Le cellule fotosensibili distribuite sulla superficie della retina convertono l'energia della luce incidente in impulsi nervosi che vengono trasportati al cervello attraverso il nervo ottico
- I fotorecettori della retina sono di due tipi: coni e bastoncelli
- I coni (tra 6 e 7 milioni in ogni occhio) sono localizzati principalmente nella porzione centrale della retina, chiamata fovea, e sono altamente sensibili al colore. Quindi, la visione dei coni è chiamata visione diurna
- I coni hanno tre tipi di pigmenti, quindi sono sensibili a tre diverse bande dello spettro visibile e in grado di percepire i colori
- La distribuzione dei recettori luminosi sulla superficie della retina è radialmente simmetrica intorno alla fovea (eccetto per la regione con il nervo ottico, detto punto cieco)

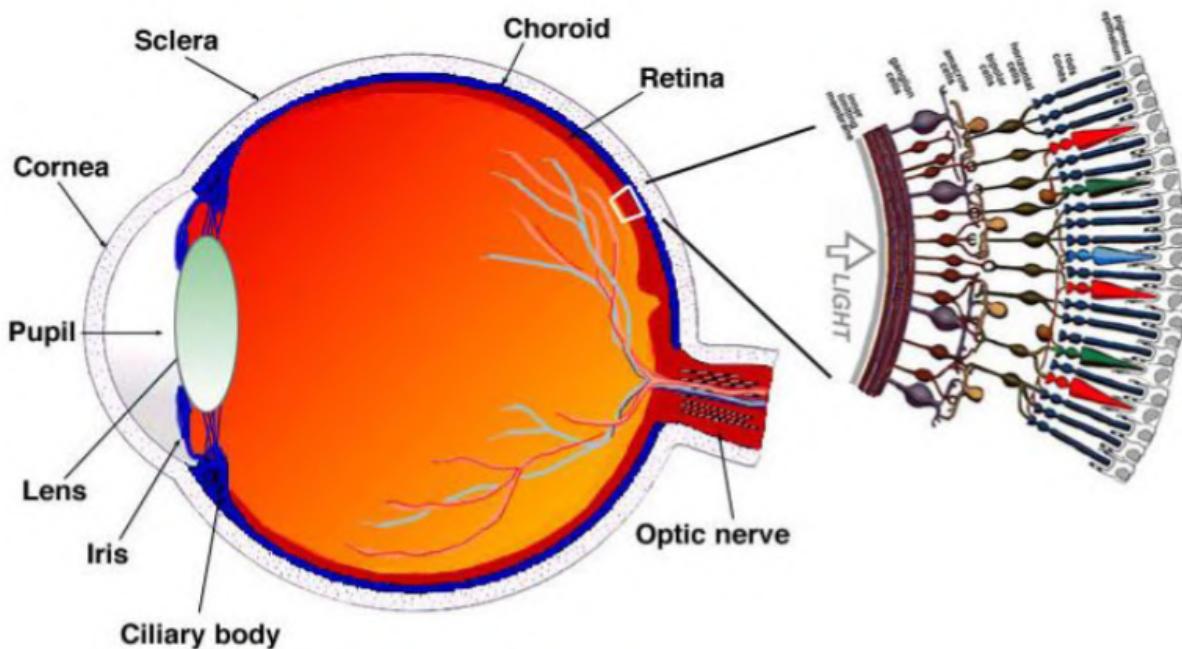
Rods

Il numero di bastoncelli (rods) è molto maggiore e sono distribuiti in modo più uniforme su tutta la superficie retinica. I bastoncelli sono quasi insensibili ai colori e hanno un solo tipo di pigmento sensibile alla luce. Sono quindi quasi interamente responsabili della visione notturna. La visione notturna è quasi monocromatica o incolore. I bastoncelli forniscono un'immagine generale, complessiva e meno precisa del campo visivo.

Coni

I coni sono sensibili a livelli relativamente alti di illuminazione, quindi la visione dei coni è chiamata visione diurna o fotopica. La distribuzione dei recettori luminosi sulla superficie retinica è radialmente simmetrica intorno alla fovea (tranne per la regione di emergenza del nervo ottico, detta punto cieco).

La regione della retina dove passa il nervo ottico è detta punto cieco a causa dell'assenza di fotorecettori. Normalmente il punto cieco non è percepito poiché alcuni processi nel cervello interpolano il punto cieco basandosi sui dettagli circostanti, sulle informazioni provenienti dall'altro occhio e sui movimenti continui del bulbo oculare.



1.4 Cos'è il Colore?

Percezione del Colore

La luce visibile è composta da una banda relativamente stretta di frequenze. I colori che gli esseri umani percepiscono sono determinati dalla natura della luce riflessa dall'oggetto. Un oggetto che riflette la luce in tutte le lunghezze d'onda visibili appare bianco. Un oggetto la cui superficie favorisce la riflettanza in un intervallo limitato dello spettro visibile appare colorato, esibendo sfumature di colore corrispondenti alle lunghezze d'onda contenute nella luce riflessa. Ad esempio, gli oggetti verdi riflettono la luce con lunghezze d'onda principalmente nell'intervallo da 500 a 570 nm.

Luce Acromatica - Rappresentazione dell'Intensità

Se la luce è acromatica (priva di colore, nero, grigio e bianco), il suo unico attributo è l'intensità o quantità. Una misura scalare dell'intensità è data dai livelli di grigio, solitamente varianti da 0 (nero) a un valore massimo, ad esempio 255 (bianco). Questi valori possono essere rappresentati con soli 8 bit e sono anche chiamati livelli di grigio.

Rappresentazione del Colore

Negli esseri umani il colore è percepito grazie ai coni, la cui assorbanza della luce copre tre diverse gamme di frequenza, corrispondenti approssimativamente a blu, verde e rosso. Pertanto, ogni colore percepito può essere visto come una combinazione dei cosiddetti colori primari, rosso (R), verde (G) e blu (B). Le definizioni in termini di lunghezze d'onda di R, G, B sono soggettive. Nessun colore singolo può essere chiamato rosso, verde o blu. Ai fini della standardizzazione, la CIE (Commission Internationale de l'Eclairage) ha designato nel 1931 le seguenti lunghezze d'onda specifiche ai tre colori primari: B=435.8nm, G=546.1nm, R=700nm.

Combinazione Additiva dei Colori

La combinazione additiva di due colori primari produce un colore secondario:

- ▶ Magenta (M) è rosso più blu,
- ▶ Ciano (C) è verde più blu,
- ▶ Giallo (Y) è rosso più verde.

Mescolando i tre colori primari, o un colore secondario con il suo colore primario opposto, a intensità piene si produce luce bianca. I colori intermedi si ottengono variando le intensità individuali dei colori primari.

Luce Cromatica - Rappresentazione del Colore

Per la luce cromatica, ogni pixel dell'immagine deve essere descritto tenendo conto dei tre colori primari la cui combinazione determina il colore del pixel. In pratica, un colore è caratterizzato da due componenti:

- ▶ Luminosità (ovvero intensità del colore)
- ▶ Cromaticità (caratterizza le lunghezze d'onda che determinano il colore)

Un modo naturale per definirle è impiegare la tonalità e la saturazione del colore.

Tonalità e Saturazione

La tonalità è un attributo associato alla lunghezza d'onda dominante in una miscela di onde luminose. Quindi, la tonalità rappresenta il colore dominante come percepito da un osservatore: quando chiamiamo un oggetto rosso, arancione o giallo, ci riferiamo alla sua tonalità. La saturazione si riferisce alla purezza relativa di un colore, o alla quantità di luce bianca mescolata con una tonalità. I colori dello spettro puro (ad esempio, rosso, verde, blu) sono completamente (100%) saturati. Colori come il rosa (rosso e bianco) e il lavanda (viola e bianco) sono meno saturati, con il grado di saturazione inversamente proporzionale alla quantità di luce bianca aggiunta. La luce bianca ha saturazione zero.

1.5 Modelli di Colore

Scopo dei Modelli di Colore

Lo scopo dei modelli di colore (detti anche spazi di colore o sistemi di colore) è di facilitare la specificazione quantitativa dei colori in qualche standard. Un modello di colore è definito da un sistema di coordinate (di tre o più assi) e uno spazio all'interno di quel sistema, dove ogni colore è rappresentato da un singolo punto.

Spazio Colore RGB

Ogni colore è definito dalle intensità dei suoi componenti primari rosso, verde e blu. Il modello è basato su un sistema di coordinate cartesiane, con lo spazio di interesse che è un cubo di lato unitario. Tutti i valori RGB sono normalizzati nell'intervallo [0,1]. Il numero di bit usati per rappresentare ogni pixel nello spazio RGB è chiamato profondità del pixel. Il bianco è (255, 255, 255) mentre il nero è (0, 0, 0)

Spazio Colore HSI

Per applicazioni di miglioramento delle immagini, e più in generale per descrizioni dei colori percettivamente più naturali, l'uso dei modelli RGB può introdurre distorsioni cromatiche nelle immagini finali. In questi casi, è conveniente usare spazi di colore che separano la componente di intensità (spesso chiamata luminanza) dalle componenti cromatiche. Il modello HSI rappresenta i colori in base alla loro tonalità, saturazione e intensità.

Spazio Colore YCbCr

Il modello YCbCr è largamente impiegato da standard internazionali di codifica digitale per la compressione immagini, come JPEG e MPEG, e dalla TV digitale. Y codifica l'informazione di luminanza (intensità) mentre Cb e Cr rappresentano l'informazione cromatica. È facile mappare un colore nello spazio RGB sullo spazio YCbCr:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Similitudini tra l'occhio umano e le telecamere

L'acquisizione di un'immagine a colori da parte di una telecamera o di uno scanner è simile al meccanismo della visione umana. La retina dell'occhio umano è paragonabile al sensore di una telecamera. I modelli di colore RGB sono usati per codificare e rappresentare le immagini acquisite. **Ci sono delle severe similarità tra il funzionamento degli occhi umani e le telecamere**

Image Formation in short

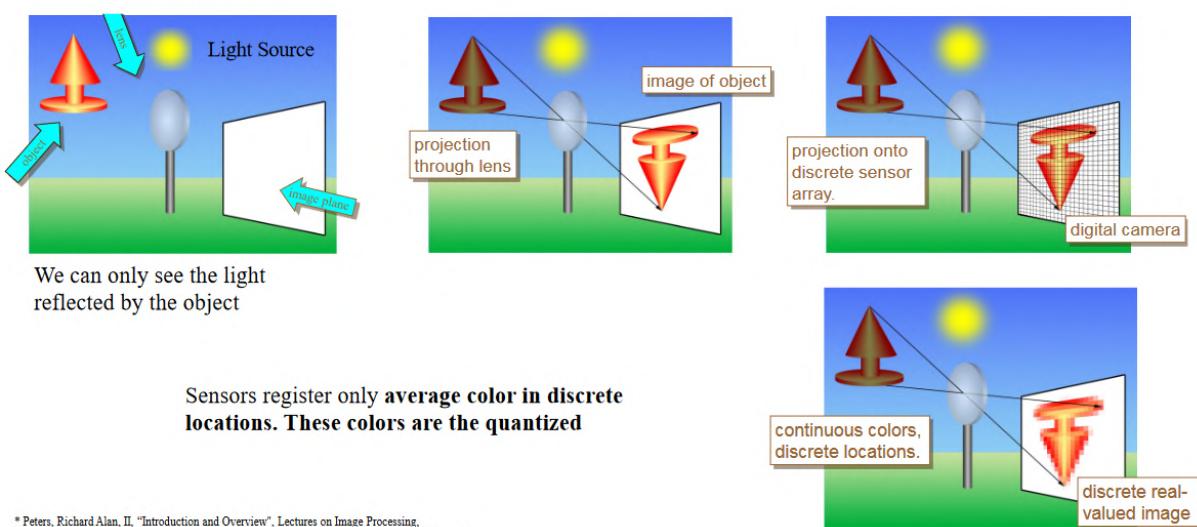


Figura 1.1: Spettro della luce visibile a confronto

1.6 Immagini Digitali

Le immagini digitali sono memorizzate come matrici 2D dove ogni cella memorizza i valori dei pixel.

Immagini in Scala di Grigi

Nelle immagini in scala di grigi, il pixel memorizza il livello di radiazione prodotto da una sorgente luminosa, chiamata intensità. Le immagini monocromatiche possono essere caratterizzate come distribuzioni spaziali della radianza della scena. Un'immagine in scala di grigi è una matrice $M \times N$.

Immagini a Colori

Nelle immagini a colori, il pixel memorizza i valori di intensità basati sulle lunghezze d'onda della radiazione, come Rosso, Blu e Verde. Un'immagine a colori è una matrice $M \times N \times 3$, dove ogni canale è una matrice $M \times N$ che memorizza i valori di intensità di una specifica lunghezza d'onda. Queste immagini specifiche sono anche chiamate immagini RGB poiché esistono altri spazi colore.

Quantizzazione

I valori di intensità sono quantizzati nell'intervallo $[0, L - 1]$ dove L è generalmente 256. Pertanto, un byte è sufficiente per memorizzare un valore di intensità.

1.7 Video Digitali

Caratteristiche

Un video è una sequenza di immagini, tutte della stessa dimensione ($H \times W$), chiamate frame. Un flusso video è caratterizzato da:

Frame Rate

Il frame rate è il numero di frame per secondo (fps). Due standard sono utilizzati per i dispositivi consumer:

- ▶ **PAL:** Usato in UE, Asia, Australia, specificando 25 fps.
- ▶ **NTSC:** Usato in USA, Canada, Giappone, specificando quasi 30 fps.

Aspect Ratio

Il rapporto d'aspetto è il rapporto tra larghezza e altezza. I formati tipici sono 4:3 e 16:9.

Risoluzione Video

Le dimensioni tipiche dei frame includono:

- ▶ 1280 x 720 (HD o 720p)
- ▶ 1920 x 1080 (Full HD o 1080p)
- ▶ 3840 x 2160 (4K o Ultra HD o 2160p)

Memorizzazione

Memorizzare un video a colori di 60 secondi in HD a 25 fps richiede di salvare 1500 frame su HDD. Ogni frame occupa $1280 \times 720 \times 3$ byte (circa 2,8 MB). Memorizzare questo breve video richiederebbe 4,2 GB.

Compressione

Molti frame sono simili, quindi è possibile utilizzare tecniche di compressione per risparmiare spazio. La compressione è anche importante per lo streaming poiché riduce la larghezza di banda richiesta. I video compressi devono essere codificati utilizzando standard adeguati per migliorare la compatibilità tra dispositivi e piattaforme.

Codec Video

Un codec video si riferisce agli standard di compressione, coinvolgendo:

- ▶ Un passaggio di codifica (eseguito tramite l'encoder) per comprimere i dati.
- ▶ Un passaggio di decodifica (tramite il decoder) per approssimare il video originale (quando la compressione determina una perdita di informazioni).

I codec più utilizzati includono:

- ▶ **H.264**: Soprattutto per lo streaming, chiamato anche AVC. Utilizzato da Netflix.
- ▶ **H.265**: Chiamato anche HEVC, per lo streaming 4K.
- ▶ **VP9**: Di Google, utilizzato da YouTube.
- ▶ **Xvid**: Gratuito per MPEG4.
- ▶ **DivX**: Per MPEG4.

Il codec e il formato video non sono la stessa cosa:

- ▶ **Codec**: Algoritmo di compressione-decompressione.
- ▶ **Formato Video**: Indica il tipo di file, un contenitore per il video compresso e altri dati (come audio o sottotitoli).

I formati video comuni includono: MKV, AVI, MOV e MP4 (conosciuto anche come MPEG4).

A causa dell'algoritmo di compressione, è difficile leggere un frame casuale da un video poiché la compressione di questo frame può coinvolgere informazioni estratte dai frame precedenti.

1.8 Dispositivi di Acquisizione delle Immagini

Webcam e CCTV

Le webcam tradizionali sono più facili da usare per videoconferenze e flussi video standard e sono più economiche dei sistemi di protocollo Internet, ma non sono adatte a scopi di sorveglianza.

Il circuito chiuso (CCTV), noto anche come videosorveglianza, utilizza telecamere per trasmettere un segnale a un luogo specifico su un set limitato di monitor tramite collegamenti dedicati.

Telecamere IP

Le telecamere IP, o telecamere a protocollo Internet, sono l'evoluzione del CCTV e offrono più opzioni e una migliore qualità video. Le telecamere IP utilizzano il protocollo Internet (IP) utilizzato dalla maggior parte delle reti locali (LAN) per trasmettere video attraverso reti di dati in forma digitale. Il video IP può essere trasmesso su Internet pubblico, consentendo agli utenti di visualizzare le loro telecamere da remoto.

Tipi di Telecamere IP

I principali tipi includono:

- ▶ Telecamere fisse
- ▶ Telecamere pan-tilt-zoom (PTZ)
- ▶ Telecamere multi-sensore

Le telecamere PTZ possono essere controllate a distanza e hanno la capacità di zoom ottico.

Sistemi di Sorveglianza Industriale

I sistemi di videosorveglianza industriale utilizzano videoregistratori di rete per supportare le telecamere IP per la registrazione, l'archiviazione, l'elaborazione dei flussi video e la gestione degli allarmi. Dal 2008, i produttori di videosorveglianza IP possono utilizzare un'interfaccia di rete standardizzata (ONVIF) per supportare la compatibilità tra i sistemi.

Altri Dispositivi di Acquisizione

- ▶ Fotocamere Stereo
- ▶ Fotocamere a Distanza – Fotocamere di Profondità
- ▶ Sistemi Lidar (Light Detection And Ranging) (nube di punti 3D)
- ▶ Fotocamere a Infrarossi (visione notturna)
- ▶ Scanner
- ▶ Dispositivi Medici

1.9 Rendering

Sistemi di Colore Additivo

Televisioni, monitor per computer e schermi sono sistemi di colore additivo che utilizzano lo spazio colore RGB. I CRT (tubi a raggi catodici) utilizzati negli schermi TV a colori e nei monitor per computer usano la natura additiva dei colori della luce. Un display CRT è composto da una grande matrice di triadi di punti di fosforo sensibile agli elettroni. Quando eccitati, ogni punto in una triade produce luce rossa, verde o blu. L'intensità della luce emessa da ogni punto di fosforo è modulata dall'intensità della corrente elettronica corrispondente. I tre colori primari di ogni triade di fosfori sono "aggiunti" insieme per formare un "pixel" di colore corretto.

I CRT sono stati sostituiti da tecnologie digitali a pannello piatto, come i display a cristalli liquidi (LCD) e i dispositivi al plasma, che utilizzano lo stesso principio, richiedendo tre sub-pixel (rosso, verde e blu) per generare un singolo pixel di colore.

Sistemi di Colore Sottrattivo

I sistemi sottrattivi sono basati su coloranti ciano, magenta e giallo (CMY), e coinvolgono la miscelazione di materiali fisici come gli inchiostri di stampa. Più materiali coloranti sono mescolati, più scuro diventa il colore. Il sistema è chiamato "sottrattivo" a causa dell'assorbimento o sottrazione di certe lunghezze d'onda dalla luce bianca (stampa su carta bianca). Le stampanti usano lo spazio colore CMYK perché la combinazione dei colori CMY dà un nero imperfetto, quindi usano il nero come quarto colore.

1.10 OpenCV

Introduzione

In questa classe, utilizzeremo la libreria OpenCV per gestire immagini e video. La libreria OpenCV (<https://opencv.org>) viene fornita con una documentazione API disponibile online su <https://docs.opencv.org/4.7.0/>.

Moduli

I principali moduli della libreria OpenCV includono:

- ▶ **core**: Funzionalità di base
- ▶ **imgproc**: Elaborazione delle Immagini
- ▶ **videoio**: Input/Output Video
- ▶ **highgui**: GUI ad alto livello
- ▶ **calib3d**: Calibrazione della Fotocamera e Ricostruzione 3D
- ▶ **features2d**: Framework di Caratteristiche 2D
- ▶ **objdetect**: Rilevamento Oggetti

IMAGE FORMATION

2.1 Formazione dell'Immagine

Il processo di formazione dell'immagine produce un'immagine basata su:

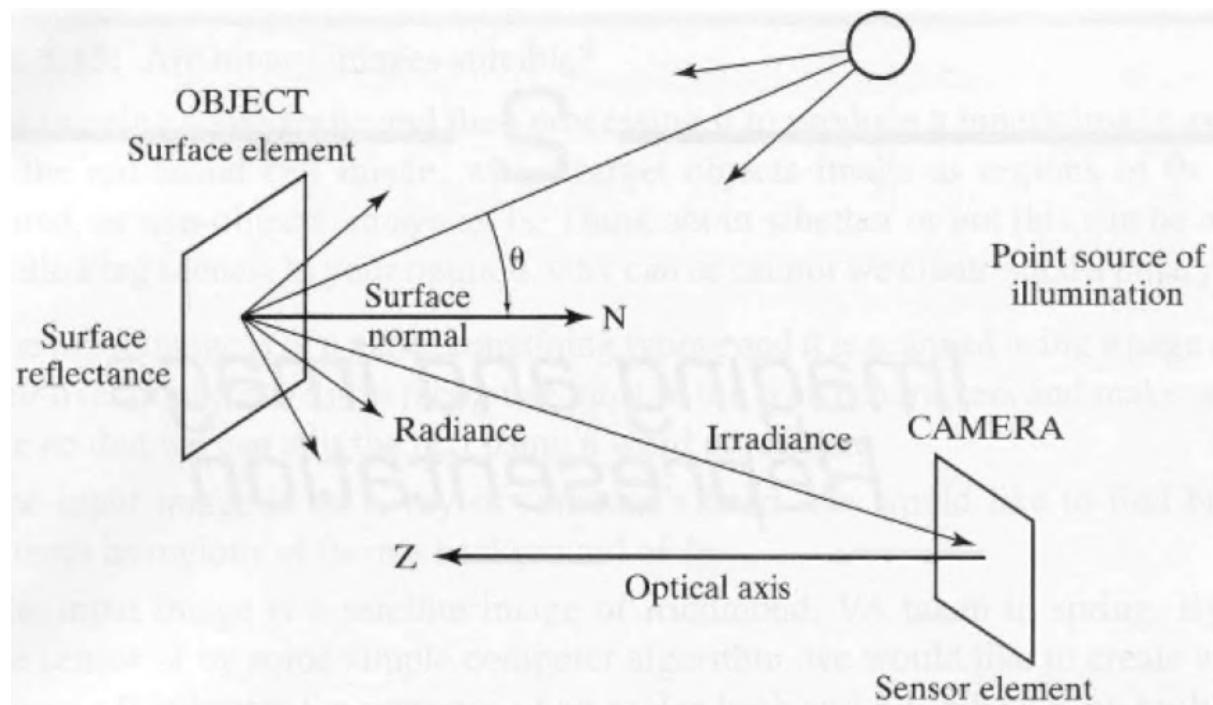
- ▶ un insieme di condizioni di illuminazione
- ▶ la geometria della scena
- ▶ le proprietà della superficie
- ▶ l'ottica della fotocamera

Per capire come si ottiene un'immagine, dobbiamo concentrarci su:

- ▶ La trasformazione geometrica che proietta punti 3D in punti 2D
- ▶ Come l'illuminazione, le proprietà della superficie e l'ottica interagiscono per produrre valori di colore

2.2 Imaging

Il processo di imaging è una mappatura di un oggetto su un piano dell'immagine: ogni punto 2D sull'immagine corrisponde a un punto 3D sull'oggetto. Durante questo processo, la lente raccoglierà e metterà a fuoco la luce diffusa dall'oggetto illuminato.



2.3 Primitivi Geometrici

Punti 2D

I punti 2D (coordinate dei pixel in un'immagine) possono essere denotati utilizzando una coppia di valori chiamati coordinate non omogenee,

$$\mathbf{x} = (x, y) \in \mathbb{R}^2$$

I punti 2D possono anche essere rappresentati utilizzando coordinate omogenee

$$\mathbf{x} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathbb{P}^2$$

dove i vettori che differiscono solo per scala sono considerati equivalenti.

Lo spazio \mathbb{P}^2 è chiamato lo spazio proiettivo 2D e

$$\mathbb{P}^2 = \mathbb{R}^3 - \{(0, 0, 0)\}$$

Un vettore omogeneo \mathbf{x} può essere convertito nuovamente in un vettore non omogeneo \mathbf{x} dividendo per l'ultimo elemento w

$$\mathbf{x} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(\tilde{x}, \tilde{y}, 1) = \tilde{w}\mathbf{x}$$

dove $\mathbf{x} = (x, y, 1)$ è il vettore aumentato.

I punti omogenei il cui ultimo elemento è $w = 0$ sono chiamati punti ideali o punti all'infinito e non hanno un'equivalente rappresentazione non omogenea (in quanto non è possibile dividere per 0).

Punti 3D

I punti 3D (ad esempio, punti nel mondo 3D) possono essere denotati utilizzando coordinate non omogenee,

$$\mathbf{x} = (x, y, z) \in \mathbb{R}^3$$

È a volte utile denotare un punto 3D utilizzando il vettore aumentato $\mathbf{x} = (x, y, z, 1)$

Altri primitivi sono linee, piani, coniche...

Point	Inhomogeneous coordinates	Homogeneous coordinates	Augmented vector
2D Points	$(x, y) \in \mathbb{R}^2$	$(\tilde{x}, \tilde{y}, \tilde{w}) \in \mathbb{P}^2$	$(x, y, 1) = (\tilde{x}/\tilde{w}, \tilde{y}/\tilde{w}, 1)$
3D Points	$(x, y, z) \in \mathbb{R}^3$	$(\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathbb{P}^3$	$(x, y, z, 1)$

2.4 Traslazione

Ogni coordinata del vettore \mathbf{x} è spostata di una quantità. La traslazione 2D produce il vettore $\mathbf{x}' = \mathbf{x} + \mathbf{t}$. La traslazione può essere espressa in forma matriciale come:

$$\bar{\mathbf{x}}' = \begin{pmatrix} I & \mathbf{t} \\ 0^T & 1 \end{pmatrix} \bar{\mathbf{x}}$$

Qui, I è una matrice identità 2×2 , \mathbf{t} è un vettore 2D che memorizza gli spostamenti delle coordinate. La matrice viene applicata a un vettore aumentato e produce un vettore aumentato.

2.5 Rotazione

Per ruotare un punto di un angolo θ , utilizziamo una matrice di rotazione R :

$$\mathbf{x}' = R\mathbf{x}$$

con:

$$R = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

R è una matrice di rotazione ortonormale poiché $RR^T = I$ e $|R| = 1$.

2.6 Roto-traslazione

Questa trasformazione è anche chiamata trasformazione rigida 2D e combina traslazione e rotazione

$$\mathbf{x}' = R\mathbf{x} + \mathbf{t}$$

In forma matriciale (quindi con vettori aumentati):

$$\bar{\mathbf{x}}' = \begin{pmatrix} R & \mathbf{t} \\ 0^T & 1 \end{pmatrix} \bar{\mathbf{x}}$$

Questa trasformazione preserva le distanze.

2.7 Scaling

Lo scaling può essere ottenuto moltiplicando ciascuna coordinata per uno scalare $s \in \mathbb{R}$

$$\mathbf{x}' = s\mathbf{x}$$

Lo scaling può essere combinato con la roto-traslazione (trasformazione di similarità):

$$\mathbf{x}' = sR\mathbf{x} + \mathbf{t}$$

In forma matriciale:

$$\bar{\mathbf{x}}' = \begin{pmatrix} s \cos \theta & -\sin \theta & t_x \\ \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{x}}$$

La trasformazione di similarità non preserva le distanze ma preserva gli angoli tra le linee.

Transformation	Inhomogeneous coordinates	Augmented vector
Translation	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$	$\bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{\mathbf{x}},$
Rotation	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$	$\bar{\mathbf{x}}' = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{\mathbf{x}}$
Scaling	$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} x \\ y \end{bmatrix}$	$\bar{\mathbf{x}}' = \begin{bmatrix} sI & 0 \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{\mathbf{x}}$
Similarity	$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$	$\bar{\mathbf{x}}' = \begin{bmatrix} s \cos \theta & -\sin \theta & t_x \\ \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{x}}$

2.8 Trasformazione Affine

Sotto una trasformazione affine, le linee parallele rimangono parallele. Si scrive come

$$\mathbf{x}' = A\mathbf{x}$$

Dove A è una matrice 2×3

In forma matriciale:

$$\bar{\mathbf{x}}' = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{x}}$$

2.9 Trasformazione Proiettiva

Nota anche come omografia o trasformazione prospettica, preserva le linee rette
Opera su coordinate omogenee ed è definita da una matrice 3×3

$$\tilde{\mathbf{x}}' = \tilde{H}\tilde{\mathbf{x}},$$

Due matrici che differiscono solo per una scala sono equivalenti.

Per ottenere il risultato in omogenee, le coordinate devono essere normalizzate:

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$

Transformation	Matrix	# DoF	Preserves	Icon
translation	$[\mathbf{I} \quad \mathbf{t}]_{3 \times 4}$	3	orientation	
rigid (Euclidean)	$[\mathbf{R} \quad \mathbf{t}]_{3 \times 4}$	6	lengths	
similarity	$[s\mathbf{R} \quad \mathbf{t}]_{3 \times 4}$	7	angles	
affine	$[\mathbf{A}]_{3 \times 4}$	12	parallelism	
projective	$[\tilde{\mathbf{H}}]_{4 \times 4}$	15	straight lines	

2.10 Rotazione in 3D

Una rotazione di base (chiamata anche rotazione elementare) è una rotazione attorno a uno degli assi di un sistema di coordinate.

Il pedice indica l'asse di rotazione

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix}$$

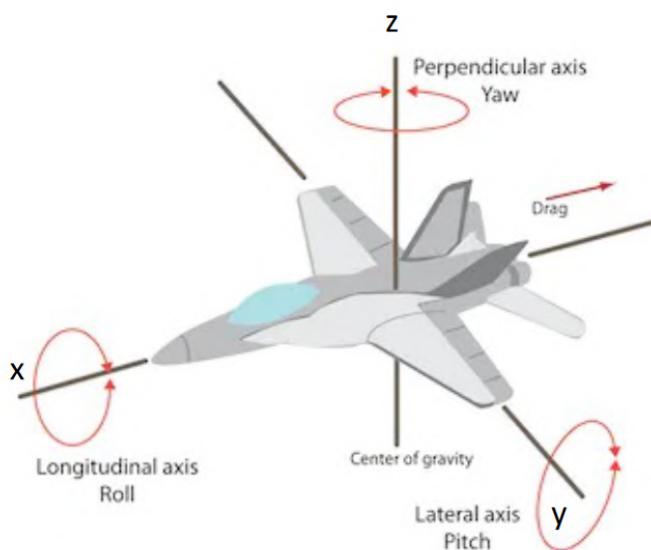
$$R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.11 Rotazione Generica

Altre matrici di rotazione possono essere ottenute da quelle elementari usando la moltiplicazione delle matrici. L'ordine delle moltiplicazioni influenza il risultato.

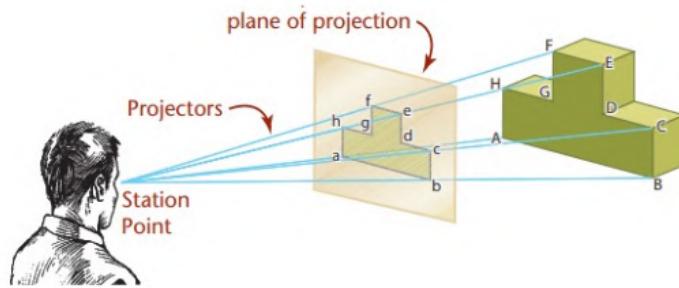
Una rotazione i cui angoli di imbardata (yaw), beccheggio (pitch) e rollio (roll) sono α , β e γ rispettivamente

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{pmatrix}$$



2.12 Proiezioni 3D su 2D

Le primitive 3D possono essere proiettate su un piano 2D (chiamato piano immagine) attraverso matrici di proiezione da 3D a 2D. Il processo di proiezione dipende dal modello di proiezione adottato. Il modello più comunemente usato è quello prospettico, che può rappresentare il comportamento delle fotocamere reali.



Proiezione Ortografica

La proiezione ortografica elimina la coordinata z del punto 3D p , risultando nel punto 2D x . Essa è definita da una matrice composta da una matrice identità $I_{2 \times 2}$ e una colonna di zeri. In forma matriciale (utilizzando coordinate omogenee), questa proiezione può approssimare le lenti a lunga focale (teleobiettivo). Una proiezione ortografica elimina semplicemente la componente z del punto p per ottenere il punto 2D x . Questo può essere scritto come:

$$x = [I_{2 \times 2} \mid 0] p$$

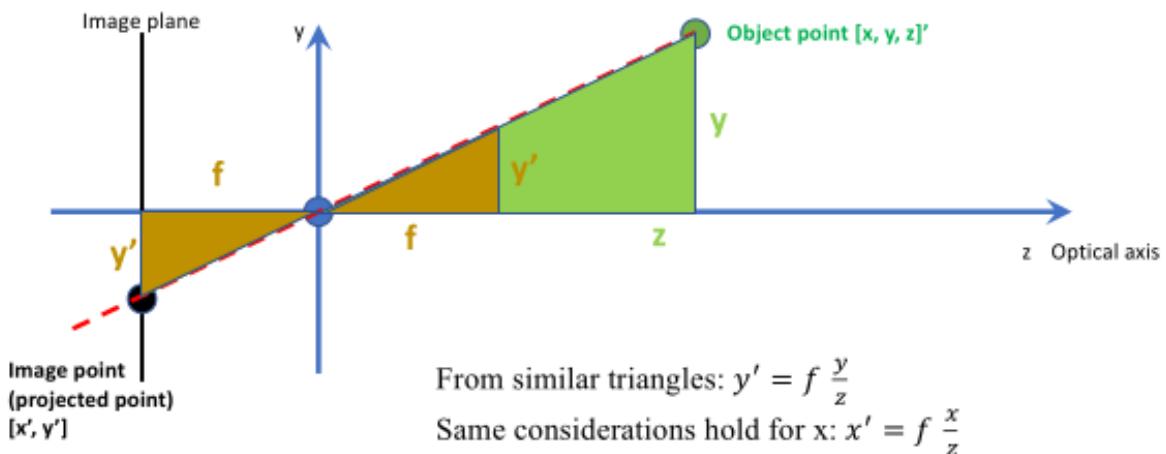
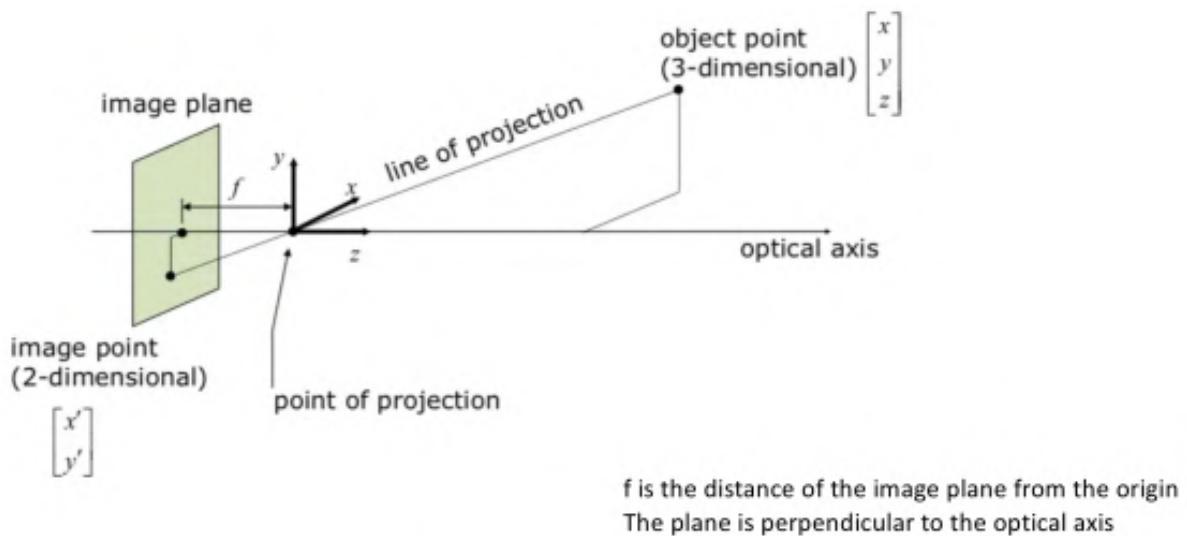
Utilizzando le coordinate omogenee (proiettive), possiamo scrivere:

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{p}$$

cioè, eliminiamo la componente z ma manteniamo la componente w .

Proiezione Prospettica

Una proiezione prospettica può essere definita attraverso un centro di proiezione e un piano di proiezione a una distanza finita dal centro di proiezione. Dato un punto 3D P , l'obiettivo è trovare il punto immagine corrispondente sul piano di proiezione. In questa proiezione, gli oggetti distanti appaiono più piccoli rispetto a quelli più vicini. Le linee parallele (che si incontrano nel punto all'infinito) sembrano intersecarsi nell'immagine proiettata (in un punto chiamato punto di fuga).



Se $f=1$, i punti vengono proiettati dividendoli per la coordinata Z

Il vettore aumentato $(x, y, 1)$ del punto proiettato sarà:

$$\bar{x} = P_z(p) = \left(\begin{array}{c} x/z \\ y/z \\ 1 \end{array} \right)$$

Dopo la proiezione, non è possibile recuperare la distanza tra gli oggetti poiché le informazioni sulla profondità vengono perse! In coordinate omogenee:

$$\tilde{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{p},$$

In generale con $f \neq 1$, in coordinate in omogenee / vettore aumentato

$$\vec{x} = \begin{pmatrix} \frac{fx}{z} \\ \frac{fy}{z} \\ 1 \end{pmatrix}$$

In coordinate omogenee

$$\vec{x} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \vec{p}$$

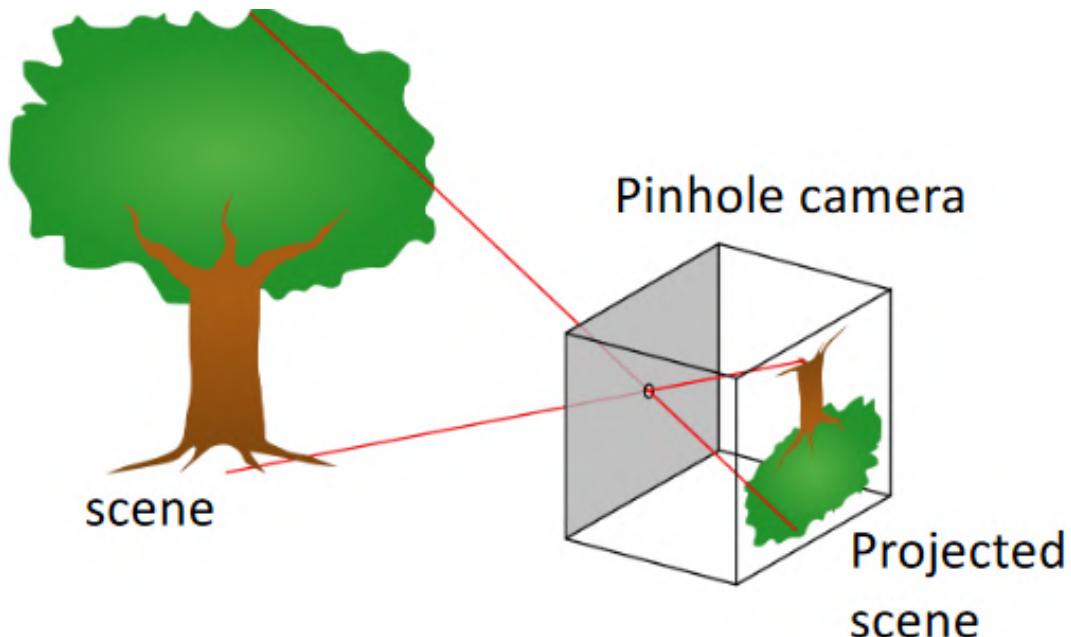
2.13 Modello della Fotocamera Stenopeica

Una fotocamera stenopeica è una fotocamera semplice senza lente ma con un piccolo foro chiamato stenopeico.

Può essere pensata come una scatola a prova di luce con un piccolo foro su un lato.

La luce di una scena passa attraverso il foro stenopeico e proietta un'immagine invertita sul lato opposto della scatola (effetto camera oscura).

La dimensione delle immagini dipende dalla distanza tra l'oggetto e il foro stenopeico.



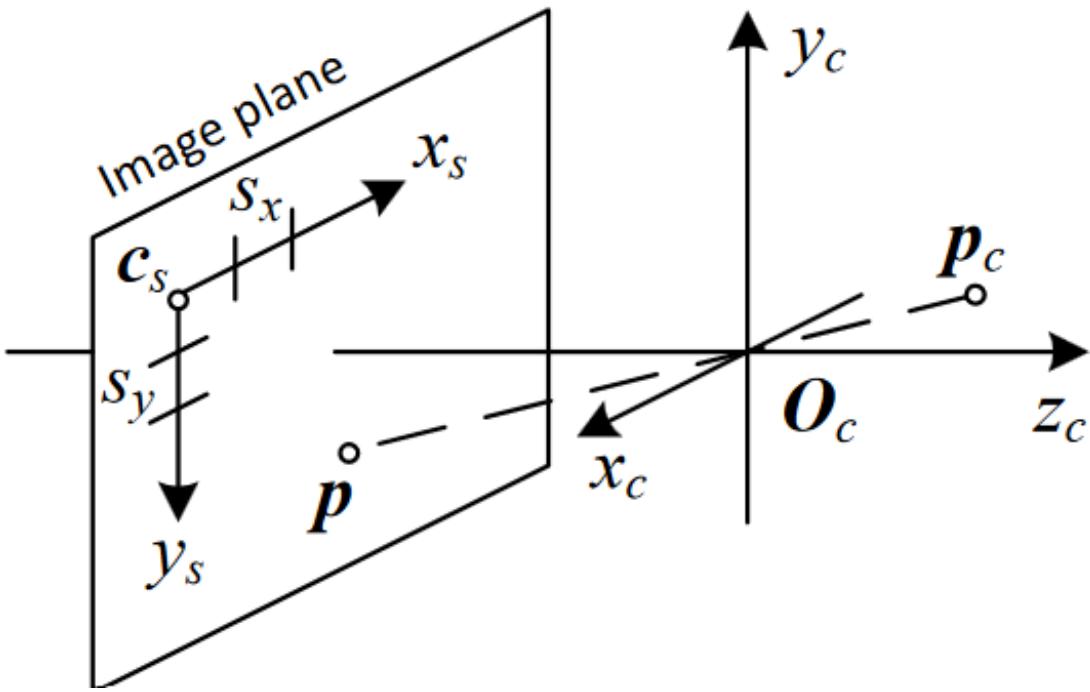
Intrinseci della Fotocamera

Supponendo di utilizzare un modello di fotocamera stenopeica e una certa matrice di proiezione, dobbiamo trasformare le coordinate risultanti (sul piano dell'immagine nel mondo 3D) in coordinate di pixel.

Le coordinate dei pixel sono valori discreti.

Abbiamo bisogno di una corrispondenza tra le coordinate dei pixel 2D e i punti 2D sul piano di proiezione (piano del sensore).

Per utilizzare una trasformazione geometrica adeguata, dobbiamo definire sistemi di coordinate appropriati.



Il sistema di riferimento 3D (x_c, y_c, z_c) centrato in O_c è utilizzato per rappresentare i punti nella scena 3D. L'origine è anche il centro di proiezione (è il foro stenopeico nel modello della fotocamera stenopeica).

Il sistema di riferimento 2D (x_s, y_s) centrato in c_s è posto sull'immagine, dove l'origine è generalmente nell'angolo in alto a sinistra dell'immagine. I parametri s_x e s_y rappresentano la spaziatura dei pixel poiché le coordinate dei pixel sono valori discreti.

Il piano del sensore (il piano dell'immagine) può avere una certa orientazione rispetto a O_c . p_c è un punto nel mondo 3D, p è la sua proiezione sul piano dell'immagine 3D. La linea tratteggiata è il raggio di proiezione.

Corrispondenza tra Pixel dell'Immagine e Punti Proiettati

Per prima cosa, dobbiamo trovare una corrispondenza tra ciascun pixel dell'immagine (x_s, y_s) e i punti proiettati p tenendo conto di:

- ▶ L'orientazione (3D) dell'array del sensore, cioè una matrice di rotazione R_s .
- ▶ La traslazione (3D) dovuta ai diversi sistemi di riferimento considerati, quindi la traslazione c_s .
- ▶ Un fattore di scala dovuto alla spaziatura dei pixel s_x, s_y .

Il vettore aumentato del pixel dell'immagine è denotato con \vec{x}_s .

Questa convenzione non è rispettata da tutte le librerie di imaging, ma l'aggiustamento per altri sistemi di coordinate è semplice. Per mappare i centri dei pixel alle coordinate 3D, moltiplichiamo prima i valori (x_s, y_s) per le spaziature dei pixel (s_x, s_y) (a volte espresse in micron per i sensori a

stato solido) e poi descriviamo l'orientazione dell'array del sensore rispetto al centro di proiezione della fotocamera O_c con un'origine c_s e una rotazione 3D R_s .

La proiezione combinata 2D a 3D può essere quindi scritta come:

$$p = [R_s \ c_s] \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix} = M_s \tilde{x}_s = \omega M_s \tilde{x}_s$$

Le prime due colonne della matrice $3 \times 3 M_s$ sono i vettori 3D corrispondenti ai passi unitari nell'array dei pixel dell'immagine lungo le direzioni x_s e y_s , mentre la terza colonna è il vettore di traslazione in 3D.

\tilde{x}_s è il pixel in coordinate omogenee.

2.14 Matrice di Proiezione M_s

La matrice M_s dovrebbe essere parametrizzata da 8 incognite:

- ▶ 3 per la rotazione R_s (3 angoli);
- ▶ 3 per la traslazione c_s ;
- ▶ 2 fattori di scala s_x, s_y .

I vincoli possono essere inclusi considerando che l'array del sensore si trova a una distanza fissa (lunghezza focale) dall'origine O_c . Quindi, le incognite effettive sono 7 (in teoria).

Abbiamo trovato una relazione tra p e il pixel \vec{p} :

$$\vec{p} = \omega M_s \tilde{x}_s$$

A causa del processo di proiezione, $\vec{p} = s \vec{p}_c$ (dove s è un fattore di scala). Quindi:

$$\tilde{x}_s = \alpha M_s^{-1} p_c = \mathbf{K} p_c$$

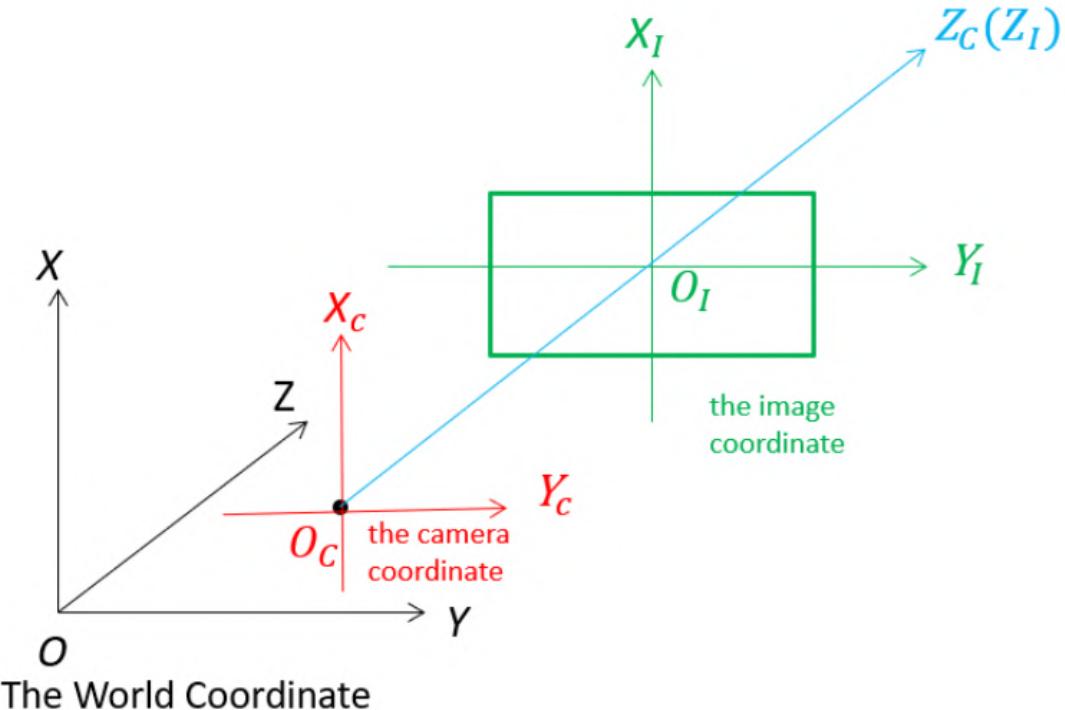
La matrice $3 \times 3 \mathbf{K}$ è chiamata matrice di calibrazione e descrive gli intrinseci della fotocamera (contrapposti agli estrinseci della fotocamera che definiscono la posa della fotocamera nello spazio).

2.15 Stima di \mathbf{K}

In teoria, K ha 7 gradi di libertà e 8 in pratica (non conosciamo la lunghezza focale). Dovremmo stimare K utilizzando misurazioni di punti 3D conosciuti nella scena e punti corrispondenti nell'immagine. Solitamente vengono utilizzati pattern come scacchiere per misurare tali punti. Tuttavia, non possiamo recuperare completamente la matrice K basandoci solo su misurazioni esterne, poiché è difficile misurare punti nel sistema di riferimento 3D centrato in O_c . Abbiamo bisogno di stimare sia i parametri intrinseci (K) che quelli estrinseci (R, t) della fotocamera con una serie di misurazioni introducendo un sistema di riferimento 3D nel mondo 3D.

2.16 Aggiunta di un nuovo sistema di coordinate

Con il sistema di coordinate del mondo possiamo rappresentare la posa della fotocamera nel mondo. Rispetto all'origine del sistema di coordinate del mondo, l'origine del sistema di coordinate della fotocamera è caratterizzata da una matrice di rotazione R e un vettore di traslazione t . Diciamo che la posa della fotocamera è $[R, t]$.



2.17 Stima dei parametri della fotocamera

Nel mondo 3D, la fotocamera ha una posizione e un'orientazione che possono essere rappresentate da una matrice di rotazione R e un vettore di traslazione t . Pertanto, un punto p_w nel mondo 3D può essere proiettato considerando che la matrice P , detta anche matrice della fotocamera. Dato che le rotazioni sono matrici ortogonali, possiamo stimare K e $[R \ t]$ fino a una matrice di rotazione R_1 .

$$P = KR, \quad t = KR \cdot R^\top$$

In pratica, è impossibile stimare K e $[R \ t]$ basandosi solo sulla corrispondenza dei punti.

Dobbiamo semplificare e fare delle ipotesi sul processo di proiezione. Le primitive e le trasformazioni.

- La dimensione dell'immagine è $W \times H$.
- Il piano dell'immagine è perpendicolare all'asse ottico z_c .

- Il centro dell'immagine è (cx, cy) .
- La distanza tra il piano dell'immagine e il centro ottico (origine) è la lunghezza focale f .

Vincolo sulla forma di K

Con queste ipotesi, vincoliamo la matrice K imponendo che sia una matrice triangolare superiore con 5 gradi di libertà. La matrice K può essere scritta come:

$$K = \begin{pmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

dove f è la lunghezza focale, a è il rapporto di aspetto, s codifica la deviazione tra gli assi del sensore, c_x e c_y sono le coordinate del centro dell'immagine (in pixel), anche chiamato punto principale.

In pratica, è impossibile stimare K e $[R \ t]$ basandosi solo su corrispondenze di punti. Esempio:

Se $s = 0, a = 1, R = I, t = [0, 0, 0]$, e $\vec{p}_w = [x_w, y_w, z_w, 1]$:

$$K = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\vec{x} = K[R, t]p_w = K[I0] \cdot p_w = [K \ 0]p_w = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} fx_w + c_x z_w \\ fy_w + c_y z_w \\ z_w \end{pmatrix}$$

Quando si divide per z_w :

$$\bar{x} = \begin{pmatrix} \frac{x_w}{z_w}f + c_x \\ \frac{y_w}{z_w}f + c_y \\ 1 \end{pmatrix}$$

che è una proiezione prospettica con una traslazione.

2.18 Stima dei parametri della fotocamera

$$K = \begin{pmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

dove f è la lunghezza focale, a è il rapporto di aspetto, s codifica la deviazione tra gli assi del sensore, c_x e c_y sono le coordinate del centro dell'immagine (in pixel), anche chiamato punto principale.

- ▶ Utilizziamo il set di corrispondenze misurate per stimare la matrice della fotocamera P di dimensioni 3×4 .
- ▶ Stimiamo K utilizzando una factorization della matrice RQ. K sarà la matrice triangolare superiore (R) trovata dalla fattorizzazione.

Per stimare K e $[R, t]$, dobbiamo prima stimare P e quindi scomporre la matrice.

L'algoritmo più semplice per stimare i parametri della fotocamera utilizza soluzioni a forma chiusa di problemi ben noti ed è chiamato "Direct Linear Transform" (DLT).

I principali passaggi di DLT sono:

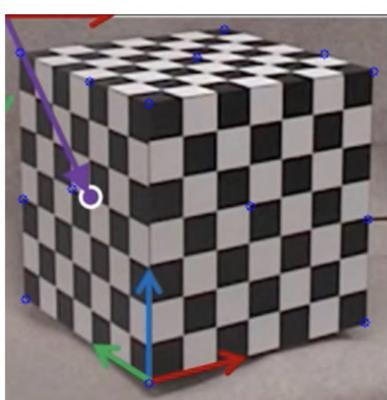
1. Trovare le corrispondenze punto 2D-punto 3D; (5 parametri per K + 6 parametri per $[R, t]$ => almeno 6 punti)
2. Derivare la matrice della fotocamera P
3. Utilizzare la fattorizzazione RQ (una variante di QR) per derivare K ($= R$) e R ($= Q$)
4. Infine, trovare t

2.19 Pattern di Calibrazione

Per semplificare il processo, spesso vengono utilizzate scacchiere per derivare due insiemi di punti:

- ▶ I punti (u, v) sono sull'immagine (in pixel).
- ▶ I punti (x, y, z) sono nel mondo 3D (in mm o metri).

Di seguito viene descritto come calibrare una fotocamera da corrispondenze trovate su un'immagine.



```
#first face (fake real world coordinates)
objp[1,:] =[7, 0, 0]
objp[2,:] =[7, 0, 7]
objp[3,:] =[0, 0, 7]
objp[4,:] =[3, 0, 4]

#corresponding points (on the image, manually measured)
imgp[1,:] = [252, 212]
imgp[2,:] = [258, 39]
imgp[3,:] = [103, 61]
imgp[4,:] = [173, 132]
```

2.20 Stima della Matrice della Fotocamera

I due insiemi vengono utilizzati per stimare la matrice della fotocamera. Per ogni coppia di punti 2D-3D, deve valere:

$$\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix} \quad (2.1)$$

La relazione sopra fornisce un sistema di equazioni in cui le incognite sono gli elementi della matrice della fotocamera.

$$\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix}$$

$$a = p_{11}x_w^{(i)} + p_{12}y_w^{(i)} + p_{13}z_w^{(i)} + p_{14}$$

$$b = p_{21}x_w^{(i)} + p_{22}y_w^{(i)} + p_{23}z_w^{(i)} + p_{24}$$

$$c = p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}$$

$$u^{(1)} = \frac{a}{c} = \frac{p_{11}x_w^1 + p_{12}y_w^1 + p_{13}z_w^1 + p_{14}}{p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}}$$

$$v^{(1)} = \frac{b}{c} = \frac{p_{21}x_w^1 + p_{22}y_w^1 + p_{23}z_w^1 + p_{24}}{p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}}$$

$$u^{(1)}(p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}) = p_{11}x_w^1 + p_{12}y_w^1 + p_{13}z_w^1 + p_{14}$$

$$v^{(1)}(p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}) = p_{21}x_w^1 + p_{22}y_w^1 + p_{23}z_w^1 + p_{24}$$

$$p_{11}x_w^1 + p_{12}y_w^1 + p_{13}z_w^1 + p_{14} - u^{(1)}(p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}) = 0$$

$$p_{21}x_w^1 + p_{22}y_w^1 + p_{23}z_w^1 + p_{24} - v^{(1)}(p_{31}x_w^1 + p_{32}y_w^1 + p_{33}z_w^1 + p_{34}) = 0$$

$$[x_w^1, y_w^1, z_w^1, 1, 0, 0, 0, 0, -u^{(1)}x_w^1, -u^{(1)}y_w^1, -u^{(1)}z_w^1, -u^{(1)}] [p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34}]^T = 0$$

$$[0, 0, 0, 0, x_w^1, y_w^1, z_w^1, 1, -v^{(1)}x_w^1, -v^{(1)}y_w^1, -v^{(1)}z_w^1, -v^{(1)}] [p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34}]^T = 0$$

Le equazioni possono essere riscritte come una matrice di valori costanti moltiplicata per il vettore delle incognite. Il nuovo sistema può essere risolto mediante minimizzazione degli errori quadratici minimi.

$$\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\ \vdots & \vdots \\ x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_ix_w^{(i)} & -u_iy_w^{(i)} & -u_iz_w^{(i)} & -u_i \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_ix_w^{(i)} & -v_iy_w^{(i)} & -v_iz_w^{(i)} & -v_i \\ \vdots & \vdots \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n \end{bmatrix} = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}$$

Poiché la matrice di proiezione può essere calcolata fino a una scala, possiamo dividere per $p_{3,4}$ tutti i parametri o possiamo vincolare p in modo che $\|p\|_2 = 1$.

Nel secondo caso, vogliamo trovare p in modo che Ap sia il più possibile vicino a 0 e la norma di p sia il più possibile vicina a 1.

In pratica, vogliamo risolvere il seguente problema di minimizzazione (con un valore molto piccolo di λ):

$$\min_p ||Ap||^2 - \lambda ||p||^2$$

che è equivalente a

$$\min_p p^T A^T A p - \lambda p^T p$$

Calcolando il gradiente:

$$2A^T A p - 2\lambda p = 0 \quad \Rightarrow \quad A^T A p = \lambda p$$

Questo problema si riduce quindi a un problema di autovalori.

Pertanto, troviamo p calcolando, attraverso la SVD, l'autovettore corrispondente all'autovalore più piccolo.

Stima di $K, (R, t)$

Una volta che abbiamo p (e lo abbiamo ridimensionato), dobbiamo ricordare che, per le sue prime tre colonne:

- ▶ Sono una matrice triangolare superiore per una matrice ortogonale, quindi una fattorizzazione RQ.
- ▶ Possiamo provare a fattorizzare l'inverso di $P_{1:3}$ o lavorare sulla matrice $(TP_{1:3})^T$, dove T è la matrice:

$$T = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Una volta trovato K , possiamo calcolare $t = K^{-1}P_4$:

Alcuni dettagli implementativi

Per determinare la matrice di proiezione, calcoliamo la matrice $A^T A$ e calcoliamo i suoi autovalori e autovettori. Questo può essere facilmente fatto dalla decomposizione SVD.

In effetti: Se $A = USV^T$, allora $A^T A = VS^2 V^T$. Così $A^T A V = VSU^T USV^T = VS^2 V^T$ dove V è la matrice degli autovettori e S^2 contiene sulla diagonale gli autovalori.

La SVD può essere applicata tramite `numpy.linalg.svd()`.

Sappiamo tutti della fattorizzazione QR. Fattorizza una matrice in una matrice ortogonale Q e una matrice triangolare superiore R . Può essere calcolato tramite `np.linalg.qr`.

Nel nostro problema, dobbiamo fattorizzare $P_{1:3}$ usando una fattorizzazione RQ. Generalmente, le librerie di algebra lineare non forniscono una tecnica del genere.

Quindi, trasformiamo la matrice originale $P_{1:3}$ in modo che possiamo applicare la fattorizzazione QR.

Nel nostro caso, è sufficiente pre-moltiplicare $P_{1:3}$ per una matrice T il cui effetto è quello di invertire le righe dall'alto verso il basso, e quindi trasporre la matrice risultante. Notare che $T = T^T$:

$$P_1 = (TP)^T \quad \text{applicando la fattorizzazione QR :} \quad P_1 = Q_1 R_1$$

$$TP = P_1^T \quad \Rightarrow \quad P = T^T P_1^T = T(Q_1 R_1)^T = T R_1^T Q_1^T = T R_1^T T T Q_1^T = (T R_1^T T)(T Q_1^T)$$

Quindi:

$$P = (T R_1^T T)(T Q_1^T) = K(T Q_1^T)$$

L'unico problema che abbiamo ora è che $(T Q_1^T)$ non è una matrice di rotazione ma una riflessione a causa della moltiplicazione con T (il determinante è -1). Senza cambiare la forma della matrice, lo risolviamo pre-moltiplicando per la matrice C , che è una correzione per l'orientamento dell'asse z (spostiamo il piano di proiezione):

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

$$Q = CTQ_1^T \quad \text{e} \quad P = KQ$$

La pre-moltiplicazione per T può essere fatta con `np.flipud()`
la post-moltiplicazione per T può essere fatta con `np.fliplr()`.

In OpenCV, possiamo utilizzare il metodo `cv2.calibrateCamera()` per stimare i parametri della telecamera. Questo metodo è generale e tiene conto anche delle deformazioni dell'obiettivo, come vedremo più avanti.

2.21 Calibrazione della fotocamera di Zhang

È una tecnica standard de facto (formati le cui specifiche sono di pubblico dominio, ma non sono mai state normate da un ente preposto) per calibrare una fotocamera utilizzando un bersaglio planare (di solito una scacchiera).

Calcoliamo le omografie delle viste dei punti del bersaglio e recuperiamo i parametri intrinseci usando una soluzione lineare in forma chiusa.

Procedura di calibrazione

Assumiamo di utilizzare un bersaglio di calibrazione planare composto da N punti. Vengono scattate M immagini differenti (viste) muovendo il bersaglio di calibrazione di fronte alla fotocamera.

Per ogni immagine, si calcola l'omografia H_i che mappa i punti dallo spazio proiettivo 2D del bersaglio al piano immagine:

$$\tilde{x}_s = K \begin{bmatrix} R & t \end{bmatrix} p_w = P p_w$$

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Assumiamo che $z = 0$ sia il piano su cui si trova il pattern di calibrazione nell'immagine. Per ogni immagine, assumendo $z = 0$ per tutti i punti del mondo reale sul pattern:

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = k \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

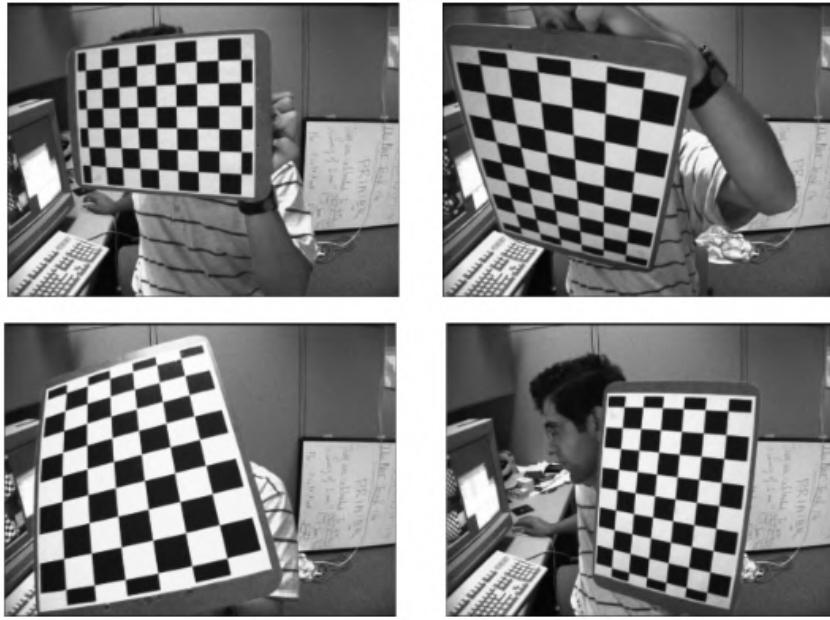
Usando un diverso sistema di coordinate del mondo in ogni immagine si semplifica l'equazione e la matrice da calcolare è una matrice di omografia 3×3 :

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Con $H = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$, a causa della presenza di w , H può essere calcolata fino a un fattore di scala. Per stimare H , procediamo in modo simile a quanto fatto per P e risolviamo un problema di minimizzazione dei minimi quadrati. Abbiamo quindi le omografie H_i con $i = 1, \dots, M$.

Vogliamo fattorizzare K fuori da queste omografie:

$$H = [h_1 \ h_2 \ h_3] = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$$



r_1 e r_2 sono colonne di R , che è una matrice di rotazione. Quindi le due colonne devono essere ortonormali:

$$\begin{aligned} r_1^T r_1 &= r_2^T r_2 = 1 \\ r_1^T r_2 &= r_2^T r_1 = 0 \end{aligned}$$

$$H = [h_1 \ h_2 \ h_3] = \lambda K [r_1 \ r_2 \ t]$$

Quindi:

$$\begin{aligned} h_1 &= \lambda K r_1 \Rightarrow K^{-1} h_1 = \lambda r_1 \\ h_2 &= \lambda K r_2 \Rightarrow K^{-1} h_2 = \lambda r_2 \end{aligned}$$

Mettendo tutto insieme:

$$\begin{aligned} r_1^T r_2 &= 0 \Rightarrow h_1^T K^{-T} K^{-1} h_2 = 0 \Rightarrow h_1^T B h_2 = 0 \\ r_1^T r_1 &= r_2^T r_2 \Rightarrow h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \Rightarrow h_1^T B h_1 = h_2^T B h_2 \end{aligned}$$

La matrice $B = K^{-T} K^{-1}$ è simmetrica e dobbiamo solo stimare il vettore 6D dalle omografie M .

Notiamo che B è simmetrica, definita da un vettore 6D $b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$.

Sia $h_i = [h_{i1}, h_{i2}, h_{i3}]^T$ la i -esima colonna di H . Abbiamo:

$$h_i^T B h_j = [h_{i1} \ h_{i2} \ h_{i3}] \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \begin{bmatrix} h_{j1} \\ h_{j2} \\ h_{j3} \end{bmatrix}$$

$$\begin{aligned}
&= [h_{i1} \ h_{i2} \ h_{i3}] \begin{bmatrix} B_{11}h_{j1} + B_{12}h_{j2} + B_{13}h_{j3} \\ B_{12}h_{j1} + B_{22}h_{j2} + B_{23}h_{j3} \\ B_{13}h_{j1} + B_{23}h_{j2} + B_{33}h_{j3} \end{bmatrix} \\
&= \mathbf{B}_{11}h_{i1}h_{j1} + \mathbf{B}_{12}h_{i1}h_{j2} + \mathbf{B}_{13}h_{i1}h_{j3} + \mathbf{B}_{12}h_{i2}h_{j1} + \mathbf{B}_{22}h_{i2}h_{j2} \\
&\quad + \mathbf{B}_{23}h_{i2}h_{j3} + \mathbf{B}_{13}h_{i3}h_{j1} + \mathbf{B}_{23}h_{i3}h_{j2} + \mathbf{B}_{33}h_{i3}h_{j3} \\
&= \mathbf{B}_{11}(h_{i1}h_{j1})\mathbf{B}_{12}(h_{i1}h_{j2} + h_{i2}h_{j1}) + \mathbf{B}_{13}(h_{i1}h_{j3} + h_{i3}h_{j1}) + \\
&\quad \mathbf{B}_{22}(h_{i2}h_{j2}) + \mathbf{B}_{23}(h_{i2}h_{j3} + h_{i3}h_{j2}) + \mathbf{B}_{33}h_{i3}h_{j3} = v_{ij}^T b
\end{aligned}$$

dal quale ottengo:

$$h_i^T B h_j = v_{ij}^T b$$

Ponendo:

$$\begin{aligned}
v_{ij} &= [h_{i1}h_{j1}, \ h_{i1}h_{j2} + h_{i2}h_{j1}, \ h_{i2}h_{j2}, \ h_{i3}h_{j1} + h_{i1}h_{j3}, \ h_{i3}h_{j2} + h_{i2}h_{j3}, \ h_{i3}h_{j3}]^T \\
b &= [B_{11}, \ B_{12}, \ B_{22}, \ B_{13}, \ B_{23}, \ B_{33}]^T
\end{aligned}$$

I due vincoli:

$$h_1^T B h_2 = 0$$

$$h_1^T B h_1 = h_2^T B h_2$$

diventano

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0$$

e, con \mathbf{M} immagini:

$$\mathbf{V}\mathbf{b} = 0$$

Dove \mathbf{V} è una matrice $2M \times 6$. La soluzione è l'autovettore di $\mathbf{V}^T \mathbf{V}$ associato al più piccolo autovalore.

Avendo denotato:

$$B = K^{-T} K^{-1}$$

Avremo quindi:

$$h_1^T B h_2 = 0$$

$$h_1^T B h_1 = h_2^T B h_2$$

La matrice B è simmetrica e può essere calcolata conoscendo la forma di K :

$$K = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad K^{-1} = \begin{bmatrix} \frac{1}{f} & -\frac{s}{af^2} & \frac{c_x}{f} + \frac{c_y s}{af^2} \\ 0 & \frac{1}{af} & -\frac{c_y}{af} \\ 0 & 0 & 1 \end{bmatrix}$$

La matrice $B = K^{-T}K^{-1}$ è quindi:

$$B = \lambda \begin{bmatrix} \frac{1}{f^2} & -\frac{s}{af^3} & \frac{c_x s}{af^3} - \frac{c_x}{f^2} \\ -\frac{s}{af^3} & \frac{s^2}{a^2 f^4} + \frac{1}{a^2 f^2} & \frac{c_x s}{af^3} - \frac{c_y s^2}{a^2 f^4} \frac{c_y}{a^2 f^2} \\ \frac{c_y s}{af^3} - \frac{c_x}{f^2} & \frac{c_x s}{af^3} - \frac{c_y s^2}{a^2 f^4} - \frac{c_y}{a^2 f^2} & \left(\frac{c_y s}{af^2} - \frac{c_x}{f}\right)^2 + \frac{c_y^2}{a^2 f^2} + 1 \end{bmatrix}$$

Da B a K

$$K = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad B = K^{-T}K^{-1}$$

$$w = B_{11}B_{22}B_{33} - B_{12}^2B_{33} - B_{23}^2B_{11} + 2B_{12}B_{13}B_{23} - B_{22}B_{13}^2d = B_{11}B_{22} - B_{12}^2$$

$$f = \sqrt{\frac{w}{dB_{11}}} \quad s = \sqrt{\frac{w}{d^2B_{11}}}B_{12} \quad af = \sqrt{\frac{wB_{11}}{d^2}}$$

$$c_x = \frac{B_{12}B_{23} - B_{13}B_{22}}{d} \quad c_y = \frac{B_{12}B_{13} - B_{11}B_{23}}{d}$$

2.22 Distorsione della Lente

Distorsione Radiale

Molti obiettivi grandangolari introducono distorsione radiale, che si manifesta come una curvatura visibile nella proiezione delle linee rette. Tra i tipi di distorsione radiale, si distinguono:



(a)

Barrel
distortion

(b)

Pincushion
distortion

(c)

Fisheye distortion

2.23 Distorsione Radiale

La distorsione radiale può essere modellata matematicamente. Basandosi sul modello di distorsione radiale, le coordinate nelle immagini osservate sono spostate verso il centro dell'immagine (distorsione a barilotto) o lontano da esso (distorsione a cuscino) proporzionalmente alla loro distanza radiale. Pertanto, i modelli di distorsione radiale sono definiti da polinomi di basso ordine della distanza r di ciascun punto dal centro dell'immagine.

Modello Brown-Conrady

Ad esempio (possono essere aggiunti più elementi):

$$\begin{aligned}\hat{x}_c &= x_c(1 + k_1 r_c^2 + k_2 r_c^4) \\ \hat{y}_c &= y_c(1 + k_1 r_c^2 + k_2 r_c^4)\end{aligned}$$

Dove:

- ▶ (x_c, y_c) è il punto da correggere (dopo la rotazione e la traslazione ma prima di applicare K)
- ▶ (\hat{x}_c, \hat{y}_c) è il punto corretto
- ▶ k_1 e k_2 sono i parametri della trasformazione
- ▶ $r_c^2 = x_c^2 + y_c^2$

2.24 Distorsione Tangenziale

Si verifica perché la lente dell'immagine non è perfettamente parallela al piano di imaging e alcune aree appaiono più vicine del previsto. Un possibile modello per la distorsione radiale e tangenziale è:

$$\begin{aligned} X_d &= (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)x + 2(k_4 y + k_5 x)x + k_5 r^2 \\ Y_d &= (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)y + 2(k_4 y + k_5 x)y + k_4 r^2 \end{aligned}$$

Il modello sopra considera entrambe le distorsioni, radiale e tangenziale.

2.25 Distorsione Radiale e Tangenziale

Data una stima iniziale di K , R e t , il modello più generale di distorsione della lente funziona come segue:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [R \ t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Rototraslare i punti del mondo basandosi sulla posizione stimata della fotocamera:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}$$

Calcolare:

$$\begin{aligned} r^2 &= x'^2 + y'^2 \\ \gamma &= \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \end{aligned}$$

Questo è il modello di distorsione radiale usato da OpenCV.

Dati r e γ :

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'\gamma + 2p_1x'y' + p_2(r^2 + 2x'^2) \\ y'\gamma + 2p_2x'y' + p_1(r^2 + 2y'^2) \end{bmatrix}$$

Una volta corretta la distorsione, possiamo usare il modello a foro stenopeico per proiettare sul piano dell'immagine:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix}$$

Stima dei Parametri

I parametri per il modello di distorsione sono calcolati con i seguenti passaggi:

- ▶ Calcolare un valore iniziale per K, R, t ignorando la deformazione della lente
- ▶ Usare questi valori come soluzione iniziale e stimare i parametri del modello di distorsione e i valori affinati di K e R minimizzando l'errore quadratico medio (questo è fatto mediante minimizzazione non lineare dell'errore quadratico medio)

In alternativa, i due set di parametri possono essere stimati alternando la soluzione di due problemi fino alla convergenza:

- ▶ Parametri di deformazione congelati, stimare K, R e t
- ▶ K, R e t congelati e stimare i parametri di deformazione

2.26 Lente Fisheye

Una lente fisheye è una lente grandangolare ultra larga che produce una forte distorsione visiva intesa a creare un'immagine emisferica. Le lenti fisheye usano una mappatura speciale, che dà alle immagini un aspetto convesso non rettilineo caratteristico. Le lenti fisheye possono utilizzare una delle seguenti proiezioni:



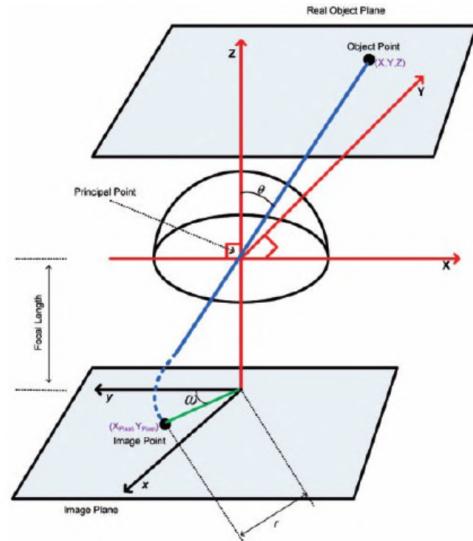
$$\begin{aligned} r &= f\theta \\ r &= 2f \tan\left(\frac{\theta}{2}\right) \\ r &= f \sin(\theta) \\ r &= 2f \sin\left(\frac{\theta}{2}\right) \end{aligned}$$

Tuttavia, le lenti fisheye possono non obbedire esattamente a queste formule, a causa della distorsione.

Modello di Proiezione

Proiezione di un Punto 3D sul Piano dell'Immagine

- ▶ Un punto nel mondo 3D $P = (X, Y, Z)$ è proiettato sul piano dell'immagine nel punto (X_{image}, Y_{image}) .
- ▶ Il piano dell'immagine è a una distanza dal punto principale di f (lunghezza focale).
- ▶ Il punto P forma un angolo θ rispetto all'asse z .
- ▶ In coordinate polari, il punto immagine ha coordinate (r, ω) .
- ▶ La coordinata r dipende dal modello di proiezione adottato.
- ▶ L'angolo ω è lo stesso angolo che la proiezione di P avrebbe sul piano XY (se consideriamo le coordinate sferiche di P , è l'angolo azimutale φ nella figura).



2.27 Formazione dell'Immagine Fotometrica

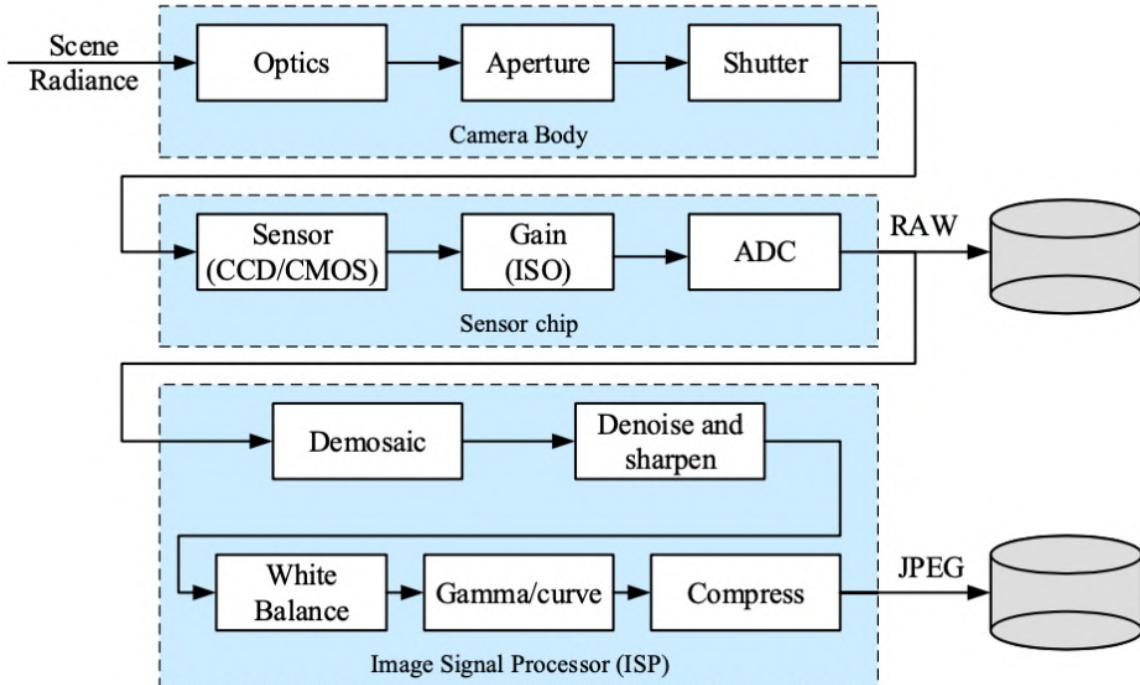
I metodi di proiezione ci dicono dove sono proiettate le caratteristiche 3D sull'immagine. Tuttavia, le immagini sono costituite da valori discreti di colore o intensità. Questi valori dipendono da diversi fattori, tra cui:

- ▶ illuminazione ambientale
- ▶ proprietà e geometria della superficie illuminata, dipendenti dalle proprietà del materiale che possono influenzare la direzione della luce riflessa
- ▶ ottica della fotocamera e proprietà del sensore (ad esempio, la forma della lente)

2.28 Fotocamere Digitali

Dopo essere partiti da una o più sorgenti luminose, riflettendosi su una o più superfici nel mondo, e passando attraverso l'ottica della fotocamera (lenti), la luce raggiunge finalmente il sensore di imaging. I fotoni che arrivano su questo sensore vengono convertiti nei valori digitali (R, G, B). Considereremo un modello di fotocamera digitale che tiene conto di:

- ▶ esposizione (guadagno e velocità dell'otturatore),
- ▶ mappature non lineari,
- ▶ campionamento e aliasing,
- ▶ rumore.



2.29 Chip Sensore

La luce che cade su un sensore di imaging viene solitamente raccolta da un'area di rilevamento attiva, integrata per la durata dell'esposizione, e poi passata a un set di amplificatori di senso. I due principali tipi di sensore utilizzati nelle fotocamere digitali e nei video sono il dispositivo a trasferimento di carica (CCD) e l'ossido di metallo complementare su silicio (CMOS).

CCD

In un CCD, i fotoni vengono accumulati in ciascun pozzo attivo durante il tempo di esposizione. Le cariche vengono trasferite da un pozzo all'altro fino a quando non vengono depositate nell'amplificatore di segnale e quindi convertite in un segnale digitale. Il CCD offre alta qualità dell'immagine con meno rumore, ma è più costoso.

CMOS

Nei sensori CMOS, i fotoni che colpiscono il sensore influenzano direttamente la condutività (o guadagno) di un fotodiodo, e sono localmente amplificati prima di essere letti utilizzando uno schema di multiplexing. La qualità dell'immagine è inferiore con più rumore, ma i sensori CMOS sono più economici.

2.30 Fattori che Influenzano le Prestazioni dei Sensori di Immagine

- ▶ **Velocità dell'otturatore:** tempo di esposizione, controlla la quantità di luce che raggiunge il sensore.
- ▶ **Passo di campionamento:** spazio tra le celle adiacenti sul chip. Influisce sulla risoluzione dell'immagine (numero di pixel) e sulla sensibilità alla luce/rumore (se troppo piccolo).
- ▶ **Fattore di riempimento:** area effettiva di rilevamento. Influisce sull'aliasing (meno con un valore più alto) e sulla sensibilità alla luce.
- ▶ **Dimensione del chip:** spesso misurata in pollici (es. 0.5 pollici). Chip più grandi sono preferibili (più fotosensibili) ma più costosi.
- ▶ **Guadagno analogico:** il segnale è amplificato da un amplificatore. L'impostazione ISO nelle fotocamere aiuta a controllare tale guadagno. Un guadagno più alto migliora la qualità dell'immagine in condizioni di scarsa illuminazione ma può anche amplificare il rumore.
- ▶ **Rumore del sensore:** rumore additivo da varie fonti tra cui rumore di fotoni, quantizzazione, il processo mediante il quale i fotoni vengono contati.
- ▶ **Risoluzione del convertitore AD:** numero di bit prodotti dal processo di conversione da analogico a digitale (bit).

2.31 Campionamento e Quantizzazione (ADC)

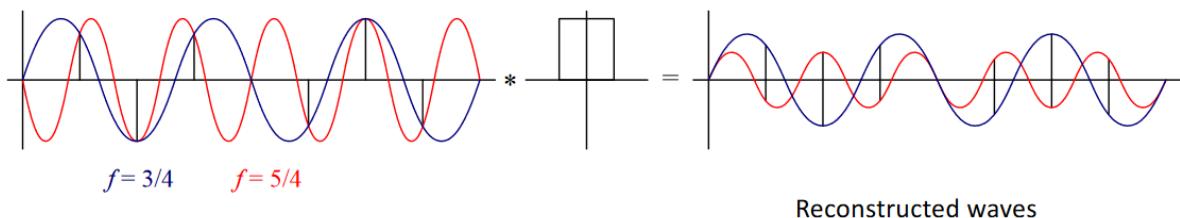
- ▶ Il segnale (analogico) acquisito attraverso il sensore viene digitalizzato sia spazialmente che in ampiezza per ottenere un segnale digitale.
- ▶ Il campionamento si riferisce al processo di digitalizzazione spaziale del segnale. La frequenza di campionamento determina la risoluzione spaziale dell'immagine digitalizzata.
- ▶ La quantizzazione si riferisce al processo di digitalizzazione dell'ampiezza del segnale. Il livello di quantizzazione determina il numero di livelli di intensità nell'immagine digitalizzata.
- ▶ Il numero di livelli di quantizzazione dovrebbe essere sufficientemente alto per la percezione umana dei dettagli di ombreggiatura nell'immagine. L'occorrenza di falsi contorni è il problema principale nelle immagini che sono state quantizzate con livelli di luminosità insufficienti.

2.32 Aliasing

Un segnale a banda limitata può essere rappresentato nello spazio delle frequenze come la somma di onde sinusoidali e coseno. L'aliasing è una distorsione che si verifica quando due elementi del segnale diventano indistinguibili a causa del sottocampionamento. Per evitare l'aliasing, il teorema di campionamento di Shannon stabilisce che la frequenza di campionamento minima deve essere almeno il doppio della frequenza massima del segnale. Tuttavia, l'aliasing è inevitabile in una funzione campionata in un intervallo limitato. Gli effetti dell'aliasing possono essere ridotti attenuando le frequenze più alte prima del campionamento, ma non possono essere eliminati in post-elaborazione.

Aliasing e Anti-Aliasing

Consideriamo due onde sinusoidali di frequenze diverse (blu e rosso). Le due onde vengono sottocampionate con una frequenza pari a 2 (vedi linee verticali) e producono gli stessi campioni, perdendo così le informazioni sulle frequenze originali. Le onde ricostruite possono essere ottenute convolvendo le versioni sottocampionate con un filtro a scatola. I due segnali ricostruiti appaiono come l'inverso l'uno dell'altro e non possiamo distinguere tra le frequenze originali.



Il teorema di campionamento di Shannon afferma che la frequenza di campionamento minima f_s per ricostruire un segnale dai suoi campioni istantanei deve essere almeno il doppio della massima frequenza f_{max} del segnale (nota come frequenza di Nyquist): $f_s \geq 2f_{max}$. Sfortunatamente, l'aliasing sarà sempre presente in una funzione campionata, poiché in pratica una funzione può essere campionata solo in un intervallo limitato. Questo è il caso di un'immagine, che è un segnale di estensione spaziale limitata. In queste condizioni, i segnali vengono scomposti in onde sinusoidali infinite (in altre parole: nessuna funzione di durata finita può essere a banda limitata). L'aliasing è una conseguenza inevitabile del lavoro con sequenze discrete di lunghezza finita.

Gli effetti dell'aliasing possono essere ridotti attenuando le alte frequenze (ad esempio sfocando l'immagine) prima che la funzione venga campionata. Questo processo di anti-aliasing non può essere applicato dopo l'acquisizione dell'immagine, poiché l'aliasing non può essere "annullato dopo il fatto" utilizzando strumenti computazionali. In post-elaborazione, l'immagine può essere migliorata con un po' di sfocatura, ma l'aliasing sarà comunque presente.

2.33 Percezione del Colore

Quando un raggio di luce solare passa attraverso un prisma, si scomponete in uno spettro continuo di colori visibili. La luce visibile è composta da una banda relativamente stretta di frequenze e i colori percepiti dagli esseri umani sono determinati dalla natura della luce riflessa dall'oggetto. Gli oggetti che riflettono tutte le lunghezze d'onda visibili appaiono bianchi, mentre quelli che favoriscono la riflettanza di un intervallo limitato dello spettro appaiono colorati. La percezione del colore nell'occhio umano è mediata dai coni, sensori che assorbono luce in tre intervalli di frequenza corrispondenti a blu, verde e rosso. I colori percepiti sono una combinazione di questi colori primari.

La Commission Internationale de l'Éclairage (CIE) ha standardizzato i colori primari rosso, verde e blu con lunghezze d'onda specifiche di 700 nm, 546.1 nm e 435.8 nm rispettivamente. La combinazione additiva di questi colori primari può produrre altri colori, e la combinazione additiva di due colori primari produce un colore secondario (magenta, ciano, giallo).

2.34 Sistema di Colori Additivi

I display CRT e molte tecnologie di schermi digitali utilizzano la combinazione additiva dei colori della luce per produrre immagini a colori. Ogni pixel è composto da sub-pixel rosso, verde e blu che, modulati in intensità, formano il colore desiderato. Anche le moderne tecnologie a schermo piatto come LCD e display al plasma seguono lo stesso principio.

2.35 Sistema di Colori Sottrattivi

I sistemi di colori sottrattivi, utilizzati principalmente nella stampa, si basano su coloranti ciano, magenta e giallo (CMY) che assorbono rispettivamente la luce nelle regioni spettrali rosso, verde e blu. La combinazione dei coloranti produce colori variando la quantità di luce assorbita e riflessa. Poiché la combinazione di CMY dà un nero imperfetto, viene utilizzato anche un quarto colore, il nero (CMYK), per migliorare la qualità della stampa.

Descrizione dei Colori

Per la luce acromatica, l'unico attributo necessario per descrivere un'immagine è la sua intensità nella posizione (x, y) , che è chiamata livello di grigio dell'immagine alle coordinate (x, y) . Per la luce cromatica, ogni pixel dell'immagine deve essere descritto tenendo conto dei tre colori primari la cui combinazione determina il colore del pixel.

In pratica, un colore è caratterizzato da due componenti:

- ▶ Luminosità (ovvero l'intensità del colore)
- ▶ Cromaticità (che caratterizza le lunghezze d'onda che determinano il colore)

Un modo naturale per definirle è usare la tonalità (hue) e la saturazione del colore. La tonalità è un attributo associato alla lunghezza d'onda dominante in una miscela di onde luminose. Pertanto, la tonalità rappresenta il colore dominante come percepito da un osservatore: quando chiamiamo un oggetto rosso, arancione o giallo, ci riferiamo alla sua tonalità.

La saturazione si riferisce alla purezza relativa di un colore, o alla quantità di luce bianca mescolata con una tonalità. I colori dello spettro puro (cioè rosso, verde, blu) sono completamente (100%) saturati. Colori come il rosa (rosso e bianco) e il lavanda (violetto e bianco) sono meno saturati, con il grado di saturazione inversamente proporzionale alla quantità di luce bianca aggiunta. La luce bianca ha saturazione zero.

Lo scopo dei modelli di colore (chiamati anche spazi di colore o sistemi di colore) è facilitare la specificazione quantitativa dei colori in qualche standard. Un modello di colore è definito da un sistema di coordinate (di tre o più coordinate) e da uno spazio sottostante a tale sistema, dove ogni colore è rappresentato da un singolo punto.

Spazio di Colore RGB

Ogni colore è definito dalle intensità dei suoi componenti primari rosso, verde e blu. Il modello si basa su un sistema di coordinate cartesiane, con il sottospazio di interesse che è un cubo di lato unitario. Tutti i valori RGB sono normalizzati nell'intervallo [0,1]. Rosso, verde e blu sono rappresentati dai vertici sugli assi, mentre ciano, magenta e giallo si trovano su altri tre vertici. Il nero si trova all'origine e il bianco al vertice più lontano dall'origine. La scala di grigi (punti di valori RGB uguali) si estende dal nero al bianco lungo la linea che unisce questi due punti. Il numero di bit usati per rappresentare ogni pixel nello spazio RGB è chiamato profondità del pixel.

Spazio di Colore HSI

Per applicazioni di miglioramento delle immagini e, più in generale, per descrizioni del colore percettivamente più naturali, l'uso dei modelli RGB può introdurre distorsioni cromatiche nelle immagini finali. In questi casi, è conveniente usare spazi di colore che disaccoppiano la componente di intensità (spesso chiamata luminanza) dalle componenti di cromaticità. Il modello HSI rappresenta i colori tramite tonalità, saturazione e intensità (quindi un'immagine a 3 canali).

Spazio di Colore YCbCr

Il modello YCbCr è un altro modello ampiamente impiegato dagli standard internazionali di codifica digitale, come JPEG e MPEG, e dalla TV digitale. Y codifica le informazioni di luminanza (intensità) mentre Cb e Cr rappresentano le informazioni cromatiche. È facile mappare un colore nello spazio RGB su YCbCr:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Acquisizione di Immagini a Colori

La maggior parte delle fotocamere digitali utilizza un array di filtri di colore (CFA), dove i sensori alternati sono coperti da filtri di colore diversi basati su uno schema. Lo schema più usato nelle fotocamere a colori è lo schema Bayer, che posiziona filtri verdi sopra metà dei sensori (in uno schema a scacchiera) e filtri rossi e blu sopra i restanti. In effetti, il segnale di luminanza è principalmente determinato dai valori verdi e il sistema visivo è molto più sensibile ai dettagli ad alta frequenza nella luminanza che nella crominanza. Il processo di interpolazione dei valori di colore mancanti in modo da avere valori RGB validi per tutti i pixel è noto come demosaicizzazione.

Prima di codificare i valori RGB rilevati, la maggior parte delle fotocamere esegue una sorta di operazione di bilanciamento del colore nel tentativo di spostare il punto di bianco di un'immagine data più vicino al bianco puro (valori RGB uguali). Un modo semplice per eseguire la correzione del colore è moltiplicare ciascuno dei valori RGB per un fattore diverso.

I fosfori nel CRT utilizzato per visualizzare il segnale TV rispondevano in modo non lineare alla tensione di ingresso. La relazione tra la tensione V e la luminosità risultante B era caratterizzata da un numero, chiamato gamma. Per compensare questo effetto, prima della trasmissione del segnale, la luminanza rilevata Y veniva rimappata tramite una gamma inversa. Questa mappatura non solo

compensava la correzione gamma, ma poteva anche ridurre il rumore aggiunto a Y' durante la trasmissione del segnale. Nella televisione a colori, la correzione gamma viene eseguita su ciascun canale di colore. Oggi, anche se lavoriamo direttamente con segnali digitali, applichiamo ancora la correzione gamma.

L'ultima fase nella pipeline di elaborazione di una fotocamera è solitamente una forma di compressione dell'immagine (a meno che non si utilizzi uno schema di compressione senza perdita come RAW o PNG della fotocamera). Tutti gli algoritmi di compressione video e immagine a colori iniziano convertendo il segnale nello spazio colore YCbCr (o in qualche variante strettamente correlata), in modo da poter comprimere il segnale di luminanza Y con maggiore fedeltà rispetto al segnale di crominanza CbCr. Una volta che le immagini di luminanza e crominanza sono state sottocampionate e separate in immagini individuali, vengono quindi passate a una fase di trasformazione a blocchi, che estrae e quantizza i coefficienti rappresentanti le immagini. Con i video, è anche consuetudine eseguire una compensazione del moto basata su blocchi, cioè codificare la differenza tra ogni blocco e un set previsto di valori di pixel ottenuti da un blocco spostato nel frame precedente.

IMAGE PROCESSING

3.1 Elaborazione delle Immagini

Notazione

- ▶ Indichiamo con $\mathbf{x} = (i, j)$ la posizione del pixel, con $0 \leq i < H$ e $0 \leq j < W$.
- ▶ Un'immagine di dimensioni $H \times W$, f è una funzione discreta tale che $f(\mathbf{x})$ (ossia $f(i, j)$) restituisce il valore del pixel.
- ▶ Il valore del pixel appartiene all'intervallo $[0, L - 1]$ per le immagini in scala di grigi. Se l'immagine è a 8 bit, $L = 256 (= 2^8)$.
- ▶ In pratica, un'immagine in scala di grigi è memorizzata come una matrice.
- ▶ Il valore del pixel è un tripletto di valori per le immagini a colori. Se utilizziamo lo spazio colore RGB, $f(i, j) = [r, g, b]$ e ciascuna di queste intensità di colore appartiene a $[0, L - 1]$.
- ▶ In pratica, un'immagine a colori è memorizzata come una matrice tridimensionale ($H \times W \times 3$).

Operatori su Immagini e Operatori di Punto

- ▶ Un operatore di elaborazione delle immagini generale è una funzione $h()$ che prende un'immagine in ingresso f e produce un'immagine in uscita g

$$g(x) = h(f(x)) \quad \text{per ogni pixel } x$$

- ▶ Gli operatori di punto (chiamati anche trasformazioni di intensità o processi di punto omogenei) sono tali che il valore del pixel in uscita dipende solo dal valore del pixel in ingresso.
- ▶ Queste trasformazioni si applicano generalmente a immagini in scala di grigi o al canale di luminanza (nelle immagini a colori).
- ▶ Possono essere implementate in modo rapido utilizzando tabelle di consultazione che memorizzano il valore di uscita corrispondente a ciascun valore di ingresso.

Operatori di Punto per Immagini a Colori

- ▶ Aggiungere lo stesso valore a ciascun canale di colore in un'immagine RGB non solo aumenta l'intensità apparente di ogni pixel, ma può anche influenzare la tonalità e la saturazione del pixel.
- ▶ Per evitare questo effetto, è meglio:
 - rappresentare l'immagine in un altro spazio colore (YCbCr o HSI),

- migliorare la luminanza (Y o I),
- recuperare l'immagine RGB.



Operatori Puntuali

- Una trasformazione non lineare molto utilizzata, che viene spesso applicata alle immagini prima di ulteriori elaborazioni, è la **correzione gamma**, la quale viene utilizzata per rimuovere la mappatura non lineare tra la radianza in ingresso e i valori quantizzati dei pixel. Per invertire la mappatura gamma applicata dal sensore, possiamo usare

$$g(x) = [f(x)]^{\frac{1}{\gamma}}$$

- Spesso si utilizza un valore di gamma pari a 2.2.

Istogrammi

- L'istogramma (normalizzato) delle intensità di un'immagine digitale con livelli di grigio nell'intervallo $[0, L - 1]$ è una funzione discreta:

$$p(r_k) = \frac{n_k}{n} \quad \text{per } k = 0, 1, \dots, L - 1$$

- dove n_k è il numero di pixel nell'immagine con intensità r_k , e $n = MN$ è il numero totale di pixel.
- Il valore $p(r_k)$ è una stima a posteriori della probabilità di occorrenza dell'intensità r_k nell'immagine, utile per fornirci una descrizione globale dell'aspetto dell'immagine.
- Ad esempio, i picchi dell'istogramma indicano normalmente la presenza nell'immagine di alte concentrazioni di pixel con lo stesso valore di intensità o con valori simili, talvolta corrispondenti a regioni dell'immagine.

Come estrarre un'immagine monocromatica da un'immagine a colori

- Se I è un'immagine a colori con canali RGB, allora la sua immagine di valore V , che codifica il valore di intensità dell'immagine e in qualche modo l'informazione di luminanza, può essere calcolata mediante una media pixel-per-pixel dei canali R, G, B:

$$V(i, j) = \frac{R(i, j) + G(i, j) + B(i, j)}{3}$$

- Un modo per stimare l'immagine di luminanza L è tramite una somma ponderata:

$$L(i, j) = 0,299 R(i, j) + 0,587 G(i, j) + 0,114 B(i, j)$$

Equalizzazione dell'Iistogramma

- L'equalizzazione dell'istogramma è una trasformazione di intensità volta a produrre idealmente un'immagine con un istogramma "piatto", ovvero con pixel distribuiti uniformemente su tutti i possibili livelli di grigio (ogni livello di intensità caratterizza lo stesso numero di pixel nell'immagine).
- L'immagine finale avrà un aspetto ad alto contrasto e un'ampia gamma dinamica, anche se, in pratica, una distribuzione davvero uniforme delle intensità non può essere ottenuta, principalmente perché le intensità sono quantità discrete.
- Il trucco per implementare l'equalizzazione dell'istogramma è lo stesso che si usa per generare campioni casuali da una funzione di densità di probabilità:
 1. Prima, calcolare la funzione di distribuzione cumulativa del livello di intensità.
 2. Riscalare la funzione di distribuzione cumulativa tra 0 e $L - 1$, che fornisce la mappatura.
 3. Per qualsiasi valore di intensità in ingresso, determinare il valore di intensità in uscita che il pixel dovrebbe assumere in base alla mappatura trovata.

Relazioni di base tra i pixel: Vicinato

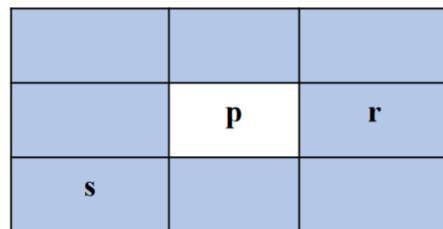
- ▶ Consideriamo un pixel p con coordinate (r, c) [r = riga, c = colonna].
- ▶ I 4-vicini di p sono i pixel $(r + 1, c), (r - 1, c), (r, c + 1), (r, c - 1)$.
- ▶ Gli 8-vicini di p sono i pixel $(r + 1, c), (r - 1, c), (r, c + 1), (r, c - 1), (r + 1, c + 1), (r - 1, c - 1), (r - 1, c + 1), (r + 1, c - 1)$.

	$(r-1, c)$		$N_4(p)$
$(r, c-1)$	(r, c)	$(r, c+1)$	
	$(r+1, c)$		

$(r-1, c-1)$	$(r-1, c)$	$(r-1, c+1)$	$N_8(p)$
$(r, c-1)$	(r, c)	$(r, c+1)$	
$(r+1, c-1)$	$(r+1, c)$	$(r+1, c+1)$	

Relazioni di base tra i pixel: Adiacenza

- ▶ Sulla base della relazione spaziale, consideriamo due tipi di adiacenza:
 - 4-adiacenza: due pixel p e q con valori da V sono 4-adiacenti se q appartiene all'insieme $N_4(p)$.
 - 8-adiacenza: due pixel p e q con valori da V sono 8-adiacenti se q appartiene all'insieme $N_8(p)$.
- ▶ Parliamo di adiacenza perché un'immagine può anche essere considerata come un grafo in cui i pixel sono nodi e i vicini sono nodi adiacenti.



Componenti Connesse

- ▶ Le componenti connesse sono definite come regioni di pixel adiacenti che hanno lo stesso valore di ingresso o etichetta.

- Una volta che un'immagine binaria o multivalore è stata segmentata nelle sue componenti connesse, è spesso utile calcolare le statistiche di area per ciascuna regione individuale R . Tali statistiche includono:
 - l'area (numero di pixel),
 - il perimetro (numero di pixel di confine),
 - il baricentro (valori medi di x e y),
 - e i momenti del secondo ordine.
- Per trovare le componenti connesse è necessario specificare il tipo di adiacenza da utilizzare (4 o 8).

Relazioni di base tra i pixel

- Per qualsiasi pixel p in S , l'insieme di pixel in S connessi a p in S è chiamato componente连通的 di S . Se ha solo una componente连通的, allora l'insieme S è chiamato insieme连通的.
- Affinché queste definizioni abbiano senso, è necessario specificare il tipo di adiacenza utilizzata: ad esempio, se si utilizza la 4-adiacenza, l'immagine binaria mostrata nella figura seguente contiene due componenti connesse nere, mentre c'è solo una componente连通的 nera se si considera l'8-adiacenza.
- Lo stesso tipo di ragionamento è valido per le porzioni bianche dell'immagine binaria, in modo che abbiamo in totale due componenti connesse (una bianca e una nera) che si intrecciano se si utilizza l'8-adiacenza, o quattro componenti connesse distinte (due bianche e due nere) se si utilizza la 4-adiacenza.

Filtraggio Lineare

- Un operatore di vicinato o operatore locale utilizza una collezione di valori di pixel nelle vicinanze di un dato pixel per determinare il suo valore di output finale.
- Gli operatori di vicinato possono essere utilizzati per filtrare immagini, aggiungendo sfocatura morbida, affinando i dettagli, accentuando i bordi, o rimuovendo il rumore.
- In particolare, gli operatori di filtraggio lineare coinvolgono combinazioni pesate fisse di pixel in piccoli vicinati.

Dato un'immagine f e un filtro lineare h , il processo di filtraggio produce l'immagine g .

Ogni valore di pixel in uscita $g(i, j)$ è una somma pesata dei valori di pixel in f all'interno di un piccolo vicinato N centrato in (i, j) .

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l)$$

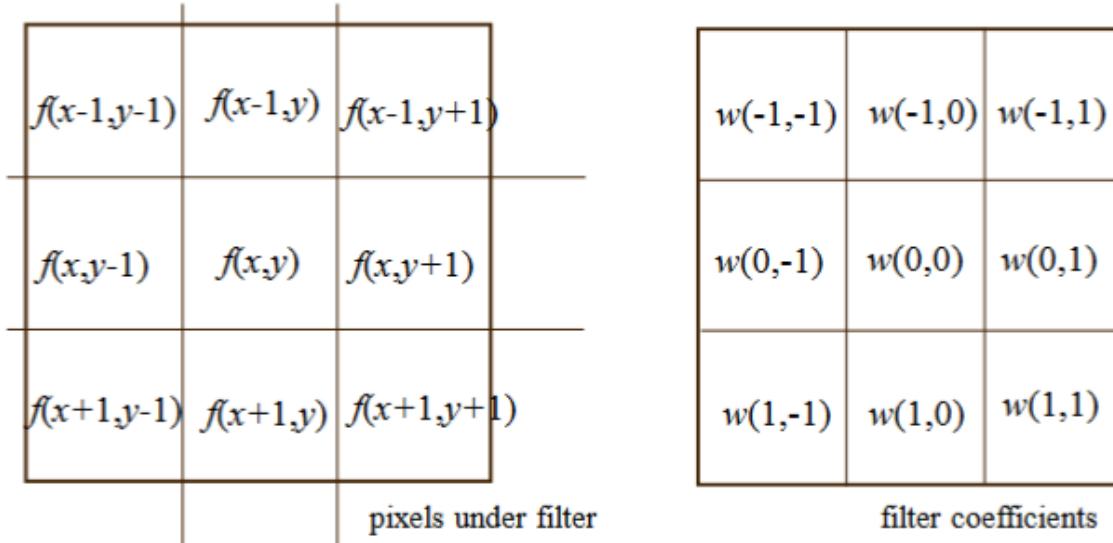
NB: This is the fundamental operator used in NN

Linear Filtering

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline
 45 & 60 & 98 & 127 & 132 & 133 & 137 & 133 \\ \hline
 46 & 65 & 98 & 123 & 126 & 128 & 131 & 133 \\ \hline
 47 & 65 & 96 & 115 & 119 & 123 & 135 & 137 \\ \hline
 47 & 63 & 91 & 107 & 113 & 122 & 138 & 134 \\ \hline
 50 & 59 & 80 & 97 & 110 & 123 & 133 & 134 \\ \hline
 49 & 53 & 68 & 83 & 97 & 113 & 128 & 133 \\ \hline
 50 & 50 & 58 & 70 & 84 & 102 & 116 & 126 \\ \hline
 50 & 50 & 52 & 58 & 69 & 86 & 101 & 120 \\ \hline
 \end{array} * \begin{array}{|c|c|c|} \hline
 0.1 & 0.1 & 0.1 \\ \hline
 0.1 & 0.2 & 0.1 \\ \hline
 0.1 & 0.1 & 0.1 \\ \hline
 \end{array} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline
 69 & 95 & 116 & 125 & 129 & 132 \\ \hline
 68 & 92 & 110 & 120 & 126 & 132 \\ \hline
 66 & 86 & 104 & 114 & 124 & 132 \\ \hline
 62 & 78 & 94 & 108 & 120 & 129 \\ \hline
 57 & 69 & 83 & 98 & 112 & 124 \\ \hline
 53 & 60 & 71 & 85 & 100 & 114 \\ \hline
 \end{array}$$

image $f(x,y)$ kernel $h(x,y)$ $g(x,y)$ Filtered image

Il filtraggio spaziale lineare di un'immagine f di dimensione $M \times N$ viene generalmente applicato con filtri w di dimensione $m \times n$, dove m e n sono valori dispari (poiché semplificano l'indicizzazione dei pixel dell'immagine) e 3×3 è la dimensione minima.



- Gli elementi nel kernel o maschera di peso $h(k, l)$ sono spesso chiamati coefficienti del filtro, pesi del filtro o parametri del filtro. L'operatore di correlazione sopra può essere notato in modo più compatto come:

$$g = f \otimes h$$

- Una variante comune di questa formula è:

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l) = \sum_{k,l} f(k, l)h(i - k, j - l)$$

- dove il segno degli spostamenti in f è stato invertito. Questo è chiamato operatore di convoluzione e h è quindi chiamata funzione di risposta all'impulso:

$$g = f * h$$

Padding

- ▶ Data una maschera, il filtraggio spaziale viene applicato facendo scorrere una maschera spaziale sull'immagine e, per ogni spostamento, calcolando la somma dei prodotti.
- ▶ Un problema sorge nel calcolo dell'output del filtro spaziale al bordo dell'immagine.
- ▶ L'immagine filtrata avrà una dimensione inferiore e le informazioni al bordo potrebbero non essere accurate.
- ▶ La soluzione per ottenere immagini filtrate della stessa dimensione dell'input richiede l'estensione dell'immagine di un numero sufficiente di elementi per garantire che l'immagine e la maschera si sovrappongano completamente anche ai bordi dell'immagine. Questo è chiamato padding.

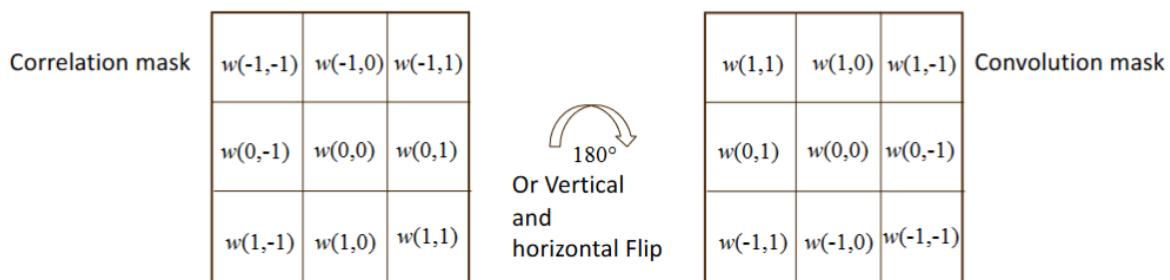
Modalità di Padding

Un certo numero di diverse modalità di padding o estensione sono state sviluppate per operazioni di vicinato:

- ▶ **Zero padding:** tutti i pixel al di fuori dell'immagine sorgente sono impostati a 0;
- ▶ **Wrap padding:** si esegue un loop “intorno” all'immagine in una configurazione “toroidale”;
- ▶ **Clamp padding:** si ripetono indefinitamente i pixel ai bordi;
- ▶ **Mirror padding:** si riflettono i pixel rispetto al bordo dell'immagine.

Correlazione e Convoluzione vs Filtraggio Spaziale Lineare

- ▶ La correlazione è il processo di spostamento di una maschera sull'immagine e di calcolo della somma dei prodotti in ogni posizione.
- ▶ La convezione è lo stesso processo, tranne per il fatto che il filtro viene prima ruotato di 180°.
- ▶ Convezione e correlazione sono uguali quando la maschera è simmetrica rispetto al suo centro, come nel caso della maggior parte dei filtri di miglioramento.



Differenza tra Correlazione e Convoluzione rispetto al Filtraggio Lineare

- ▶ Nonostante la correlazione e la convoluzione implementino entrambe il filtraggio spaziale lineare, sono concettualmente differenti.
- ▶ La correlazione è un'operazione matematica semplice per confrontare due segnali ed è una misura di quanto un segnale somiglia all'altro. Infatti, la correlazione è utilizzata nelle tecniche di matching dei template.
- ▶ La convoluzione è anch'essa uno strumento matematico che viene utilizzato per combinare due segnali al fine di produrre un nuovo segnale. Pertanto, la convoluzione misura l'effetto di un segnale sull'altro.
- ▶ La convoluzione è comunemente usata per implementare operazioni lineari principalmente finalizzate alla levigatura e al miglioramento dei bordi.

Processo di Correlazione

Input:

- ▶ f è l'immagine, in questo caso un impulso.
- ▶ w è il kernel.

Processo di Convoluzione

La convoluzione di un filtro con un impulso unitario produce una copia del filtro nella posizione dell'impulso, come ben noto dalla teoria dei sistemi lineari.

- ▶ Sia la correlazione che la convoluzione sono operatori *linear shift-invariant* (LSI):

$$h \circ (f_0 + f_1) = h \circ f_0 + h \circ f_1$$

- ▶ e obbediscono al principio di invarianza rispetto agli spostamenti:

$$g(i, j) = f(i + k, j + l) \iff (h \circ g)(i, j) = (h \circ f)(i + k, j + l)$$

Filtraggio Lineare Reso Più Veloce

- ▶ Il *Linear Filtering* può essere espresso come una moltiplicazione matrice-vettore, il che significa che può essere calcolato più velocemente. Questo è il modo in cui la maggior parte delle librerie implementa il *Linear Filtering*. Ovviamente, l'aumento delle dimensioni delle maschere aumenterà la complessità.
- ▶ Per kernel molto grandi, a seconda delle dimensioni dell'immagine, potrebbe essere più conveniente filtrare nel *Frequency Domain* (Fourier), dove la convoluzione è un prodotto elementare (teorema della convoluzione).

Correlation and Convolution

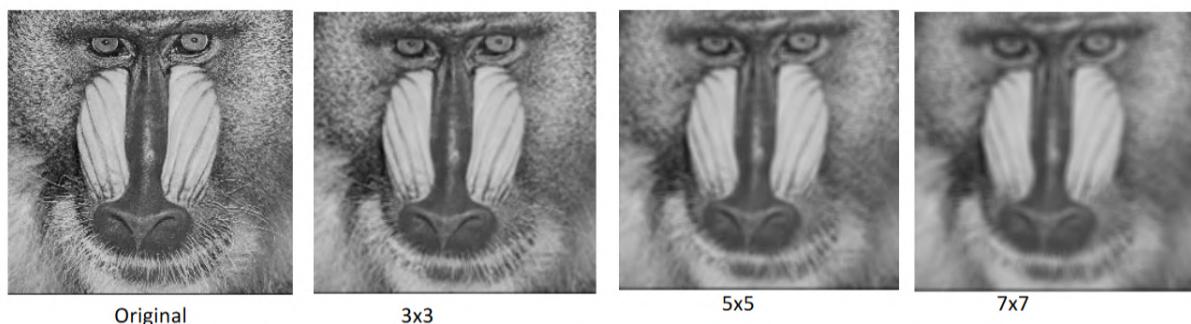
- ▶ Correlazione e convoluzione usano lo stesso algoritmo. La maggior parte degli algoritmi di visione artificiale forniscono funzioni per calcolare la convoluzione. In questo caso particolare, la correlazione può essere calcolata con questa funzione invertendo il kernel.
- ▶ **Spoiling...** Nelle *Convolutional Neural Network* (CNN), l'operazione principale è il *linear spatial filtering*.
- ▶ A differenza di quanto accade nell'elaborazione delle immagini, dove le maschere sono definite manualmente in base al problema da risolvere e, talvolta, all'esperienza, nelle CNN i pesi delle maschere vengono appresi dai dati attraverso un algoritmo che mira a minimizzare l'errore del modello sui dati.
- ▶ Poiché le maschere sono apprese, l'operazione di *spatial filtering* può essere considerata sia come una correlazione che come una convoluzione (impariamo direttamente i kernel ruotati).
- ▶ Questo è interessante quando vogliamo interpretare il modello addestrato:
 - Se consideriamo i kernel appresi come maschere di correlazione, il modello funziona semplicemente come una memoria, immagazzinando pezzi importanti di informazioni.
 - Se consideriamo i kernel appresi come maschere convolutive, il modello sta semplicemente estraendo nuove caratteristiche importanti dai dati e quindi elaborando i dati.
 - Forse, i filtri appresi giocano entrambi i ruoli? Forse alcuni dovrebbero essere considerati come correlazione e altri come convoluzione?

A cosa servono i filtri lineari?

- ▶ Un filtro spaziale lineare è completamente definito dai valori dei suoi mn coefficienti, che devono essere selezionati in base a ciò che il filtro è destinato a fare:
 - Il *Blurring* è spesso utilizzato in compiti di pre-elaborazione, come la rimozione di dettagli irrilevanti da un'immagine, ad esempio dettagli di piccole dimensioni (rispetto alla dimensione della maschera), prima dell'estrazione di regioni (grandi), e per compensare piccole imperfezioni, come piccoli spazi in linee o curve derivanti da errori o imprecisioni degli algoritmi di rilevamento dei contorni.
 - La riduzione del rumore può essere ottenuta tramite il *blurring* con un filtro lineare e anche tramite filtraggio non lineare.
 - Il *Sharpening* è utilizzato per evidenziare variazioni di intensità al fine di aumentare il contrasto locale dell'immagine e migliorare i piccoli dettagli, inclusi i bordi degli oggetti (*edge crispening*).

Smoothing Immagine

- ▶ I filtri di *Smoothing* sono utilizzati per il *image blurring* e per la riduzione del rumore.
- ▶ Un filtro lineare di *smoothing* è semplicemente un filtro di media: il valore di ogni pixel in un'immagine viene sostituito dalla media dei livelli di intensità nel vicinato definito dalla maschera del filtro, riducendo così le transizioni di intensità "brusche".
- ▶ Il filtro di media più semplice è una maschera in cui tutti i coefficienti sono uguali e di valore tale da permettere risultati compatibili con l'intervallo di intensità adottato. La somma di tutti i coefficienti nella maschera è sempre 1.
- ▶ Poiché i dettagli e il rumore sono legati alle alte frequenze, questi filtri sono anche chiamati *low pass filters* (che mantengono le basse frequenze e attenuano le alte).
- ▶ Lo *Smoothing* è più pronunciato con una dimensione del vicinato/maschera più grande.
- ▶ La media dei valori dei pixel in un vicinato può produrre livelli di intensità non necessariamente presenti nell'immagine originale. L'effetto è più evidente nelle aree dove ci sono transizioni di intensità (bordo). Questo è ciò che produce l'effetto di *blurring*. Lo *Smoothing* riduce il contrasto dell'immagine.
- ▶ Poiché il rumore casuale consiste tipicamente di transizioni brusche nei livelli di grigio, la riduzione del rumore è una delle applicazioni più ovvie dello *smoothing*.



Cosa succede se applichiamo due volte un filtro di media?

- ▶ L'applicazione consecutiva di un filtro di media è (quasi) equivalente all'applicazione di filtri più grandi all'immagine.
- ▶ Ad esempio, l'applicazione di un filtro 5×5 è equivalente a due applicazioni consecutive di una maschera 3×3 .

Filtri Gaussiani

- ▶ La maschera spaziale può essere ottenuta mediante una formulazione matematica dell'operazione di filtro

- ▶ Una funzione gaussiana di due variabili è una funzione continua a forma di campana, con la seguente forma di base:

$$h(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

- ▶ Dove σ , la deviazione standard, controlla la larghezza della gaussiana, cioè la stretta della campana, che aumenta con σ . **Filtro Gaussiano**
- ▶ Una funzione gaussiana può essere utilizzata per generare una maschera spaziale valutando la funzione intorno al suo centro, che rappresenta il punto in cui viene applicato il filtro.
- ▶ La sigma (σ) svolge un ruolo importante nella definizione della maschera poiché influenza il vicinato del punto immagine corrispondente al centro della maschera che sarà utilizzato nell'operazione di filtraggio.
- ▶ I filtri 3×3 e 5×5 sono generati da una funzione gaussiana con $\sigma = 0.5$. Successivamente, i coefficienti vengono normalizzati in base alla loro somma in modo che i pesi della maschera sommino a 1.
- ▶ L'applicazione di questi filtri ha l'effetto di levigare l'immagine ed è particolarmente utilizzata per la riduzione del rumore.

$$\begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$

$$\begin{bmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix}$$

- ▶ Applicando questi filtri si ottiene un effetto di smoothing dell'immagine e sono particolarmente utilizzati per la riduzione del rumore.

Affilatura (Sharpening) delle immagini

- ▶ I filtri di affilatura operano aumentando le differenze di intensità tra i pixel adiacenti. In questo senso, possono essere considerati complementari ai filtri di levigatura.
- ▶ L'aumento delle transizioni di intensità può anche aumentare il rumore dell'immagine.
- ▶ Evidenziare le variazioni di intensità significa anche far passare le componenti ad alta frequenza nell'immagine e rifiutare le basse frequenze, per cui i filtri di questo tipo sono anche chiamati filtri passa-alto.
- ▶ Intuitivamente, per aumentare le differenze di intensità tra i pixel adiacenti, il peso del centro della maschera deve essere di segno opposto rispetto ai pesi dei vicini.
- ▶ Si noti che la somma dei pesi è 1 per evitare bias nell'intensità dell'immagine. Vedremo in alcune diapositive come questi filtri vengono derivati.

- Se l'obiettivo del filtraggio spaziale è l'estrazione dei bordi, è auspicabile che la somma dei pesi della maschera sia 0.
 - In questo caso, il risultato del filtraggio di un'area grigia uniformemente (o lentamente) variabile è circa zero, come ci si aspetta da un filtro passa-alto, che idealmente deve eliminare la componente a zero frequenza, nota anche come componente DC (che è il valore medio di intensità nell'immagine).
- Un filtraggio passa-alto severo rende l'intensità media uguale a 0 (l'intensità media di un'immagine è proporzionale alla componente DC), quindi l'immagine risultante dovrebbe contenere pixel negativi. Pertanto, l'intensità dell'immagine visualizzata è stata certamente normalizzata prima della visualizzazione.

Mascheratura non nitida

- I kernel di levigatura possono essere utilizzati anche per affilare le immagini utilizzando un processo chiamato mascheratura non nitida.
- Sfocare l'immagine riduce le alte frequenze (idealmente dovrebbe eliminarle), quindi la differenza tra l'immagine originale e quella sfocata ha un contenuto più elevato in alte frequenze.
- Possiamo affilare l'immagine sommando una frazione di tale differenza

$$g_{\text{sharp}} = f + \gamma (f - h_{\text{blur}} * f)$$

Filtri Derivati

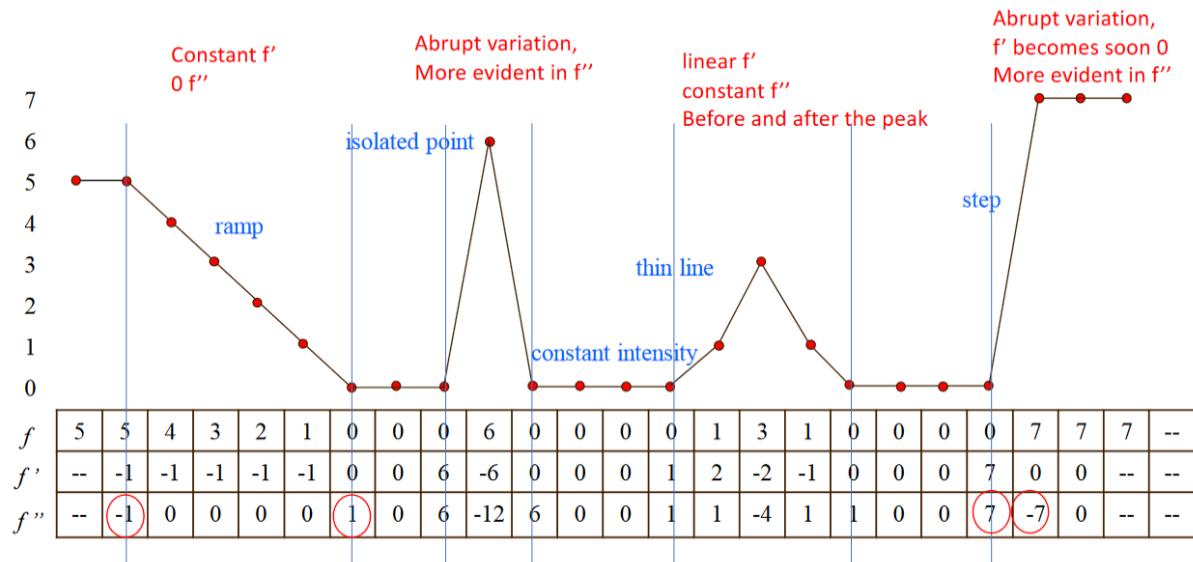
- Il filtro passa-alto (con somma dei pesi pari a 0) si comporta come un operatore derivato, poiché realizza la differenziazione delle funzioni discrete. Le derivate di una funzione discreta possono essere definite in termini di differenze locali.
- Una definizione comune della derivata prima di una funzione discreta 1-D, $f(x)$, in un punto x è la differenza:

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

- La derivata seconda di $f(x)$ è la differenza:

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

- I bordi nelle immagini digitali sono spesso transizioni di intensità a forma di gradino.
- La prima derivata dell'immagine produce bordi "spessi" (poiché la prima derivata è diversa da zero lungo un gradino), con uno spessore pari alla larghezza del gradino.
- La seconda derivata produce un bordo più sottile ma "doppio", con valori (spesso) separati da zeri e di segno diverso.



- In presenza di rumore isolato, la risposta della seconda derivata è più forte rispetto alla prima, a causa del suo comportamento più aggressivo nell'accentuare le transizioni di intensità nette.
- Il segno della seconda derivata cambia da negativo a positivo quando la transizione di intensità va da chiaro a scuro, o da positivo a negativo quando la transizione va da scuro a chiaro.

Filtri Derivati - Laplaciano

- La seconda derivata di un'immagine, $f(x, y)$, di due variabili nella posizione (x, y) può essere calcolata tramite l'operatore Laplaciano, definito come:

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

- È un operatore lineare e *isotropo* (la sua risposta è indipendente dalla direzione delle discontinuità nell'immagine). In altre parole, è invariante alla rotazione (per funzioni continue).
- Il modo più semplice per approssimare il Laplaciano in forma discreta consiste nel calcolare le seconde derivate sia in direzione x che y e sommare i due risultati.
- Per calcolare il Laplaciano, dobbiamo usare i pixel di un intorno:

$$L(x, y) = \nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

- Questo filtro è isotropo per rotazioni in incrementi di 90° :

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

- Possiamo renderlo isotropo per rotazioni in incrementi di 45° includendo i valori diagonali:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

oppure sono usate anche versioni:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{o} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Nitidezza tramite Laplaciano

- Le immagini filtrate con il Laplaciano perdono completamente l'informazione di sfondo e l'output del Laplaciano non è nullo solo nei dettagli accentuati. Possiamo aggiungerlo all'immagine originale $f(x, y)$:

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & \text{se il peso centrale è } < 0 \\ f(x, y) + \nabla^2 f(x, y) & \text{se il peso centrale è } > 0 \end{cases}$$

- Il che significa:

$$g(x, y) = f(x, y) - [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] + 4f(x, y)$$

- Oppure, tenendo conto anche dei valori diagonali:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{o} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Esempio - Laplaciano

- Esempio di utilizzo del Laplaciano con OpenCV:

```
lap = cv2.Laplacian(im_low, ddepth=-1) # 3x3
```

- Per migliorare l'immagine è possibile utilizzare un filtro:

```
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
```

```
im_sharp = cv2.filter2D(src=im_low, ddepth=-1, kernel=kernel)
```

Gradiente di Immagine

- Una approssimazione molto comune della derivata prima di una funzione digitale è il gradiente della funzione. Il gradiente di una funzione è:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- La cui magnitudine è:

$$M(x, y) = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

- E l'orientamento del gradiente è:

$$\alpha(x, y) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

- Il gradiente ha la proprietà geometrica importante che punta nella direzione del massimo cambiamento di f nella posizione (x, y) e la magnitudine del gradiente misura la velocità di questo cambiamento.
- I componenti del gradiente possono essere calcolati utilizzando differenze locali nelle due direzioni o approssimati mediante filtraggio lineare.

Operatori di Sobel

- I metodi derivativi sono **suscettibili anche al rumore dell'immagine** e ad altre variazioni spurie di intensità, come piccole fluttuazioni di luce.
- È più conveniente usare operatori di gradiente che agiscono sia come operatori derivativi in una direzione sia come operatori medianti nell'altra.

Operatori di Sobel: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

- Il gradiente G_x è dato da:

$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

- Il gradiente G_y è dato da:

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

Esempio

- Esempio di utilizzo degli operatori Sobel con OpenCV:

```
gray = cv2.cvtColor(im_low, cv2.COLOR_BGR2GRAY)

gx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
gy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)

Mag = (np.absolute(gx) + np.absolute(gy)/4).astype(np.uint8)

gx = (np.absolute(gx)/2).astype(np.uint8)
gy = (np.absolute(gy)/2).astype(np.uint8)
```

Filtri Separabili

- Il processo di esecuzione di una convoluzione richiede K^2 (moltiplicazioni e addizioni) operazioni per pixel, dove K è la dimensione (larghezza o altezza) del kernel di convoluzione.
- Questa operazione può essere velocizzata eseguendo prima una convoluzione orizzontale 1D seguita da una convoluzione verticale 1D, che richiede un totale di $2K$ operazioni per pixel.
- In pratica, un filtro separabile può essere scritto come prodotto di due filtri più semplici. Tipicamente, un'operazione di convoluzione 2D viene separata in due filtri 1D.
- **Non tutti i filtri sono separabili.**

Filtri Separabili

- Un esempio di filtro media è dato da:

$$\frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Un esempio di filtro di Sobel è dato da:

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

- I filtri 2D si ottengono come il prodotto esterno di due kernel 1D. Dato un kernel 2D K , possiamo sapere se è separabile utilizzando la SVD su di esso. Se solo il primo valore singolare è diverso da zero, è separabile:

$$K = \sum_i \sigma_i u_i v_i^T$$

Filtro LoG

- ▶ Per filtrare sia le alte che le basse frequenze, utilizziamo filtri passa-banda.
- ▶ Un modo è quello di sfocare l'immagine con un filtro gaussiano e poi prendere la Laplaciana. A causa delle proprietà dell'operatore di convoluzione, questo è equivalente a convolare direttamente con la Laplaciana di Gauss (LoG)

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma)$$

La forma di un cappello messicano indica che convolare il LoG con un'immagine restituirà una versione sfocata dell'immagine, riducendo così il rumore.

Inoltre, il valore medio della funzione è 0 e quindi sarà presente nel risultato della convoluzione.

- ▶ Maschere di dimensioni arbitrarie possono essere ottenute campionando l'espressione LoG e scalando i coefficienti in modo che sommino a zero.
- ▶ Una possibile maschera LoG è:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Osserva come è fatta la maschera: un termine positivo centrale circondato da una regione negativa adiacente, i cui valori aumentano in funzione della distanza dall'origine, e una regione esterna a zero.

- ▶ Maschere più grandi possono essere utilizzate per rilevare bordi sfocati e maschere più piccole per rilevare dettagli finemente focalizzati.

3.2 Filtri Non Lineari

- ▶ Per alcuni tipi di rumore casuale, in particolare il rumore impulsivo, noto anche come rumore sale e pepe, i filtri mediani offrono eccellenti capacità di riduzione del rumore, con un notevole minore sfocatura rispetto ai filtri di smussamento lineari di dimensioni simili.
- ▶ Un filtro mediano è un filtro non lineare che sostituisce il valore di un pixel con la mediana dei valori di intensità nel vicino di quel pixel. Ricorda che la mediana, M , di un insieme di valori è tale che metà dei valori nell'insieme sono minori o uguali a M , e metà sono maggiori di M .
- ▶ Il filtro mediano appartiene alla categoria di filtri non lineari noti come filtri per statistiche di ordine: la loro risposta si basa sull'ordinamento (classifica) dei pixel contenuti nell'area dell'immagine compresa dal filtro, e poi sulla sostituzione del pixel centrale con il valore determinato dal risultato della classifica.
- ▶ Altri esempi noti di questa categoria di filtri sono il filtro massimo e il filtro minimo.



Image with salt&pepper noise



Image after Gaussian smoothing



Image after median filtering

Elaborazione di Immagini Binari

Le immagini binarie sono spesso il risultato di un'operazione di soglia:

$$\theta(f, t) = \begin{cases} 1, & \text{se } f \geq t \\ 0, & \text{altrimenti} \end{cases}$$

- ▶ Le operazioni più comuni su immagini binarie sono chiamate operazioni morfologiche.
- ▶ Esse applicano una convoluzione con un elemento strutturante binario seguita da un'operazione di soglia.
- ▶ L'elemento strutturante può essere di qualsiasi forma, da un semplice filtro a scatola 3×3 , a strutture a disco più complicate.

Operatori Morfologici

- ▶ La dilatazione fa crescere (ispessire) gli oggetti costituiti da 1, l'erosione li restringe (assottiglia). Le operazioni di apertura e chiusura tendono a lasciare le grandi regioni e i confini lisci non modificati, mentre rimuovono piccoli oggetti o buchi e lisciamo i confini.
- ▶ **dilatazione:** $\text{dilate}(f, s) = \Theta(c, 1)$

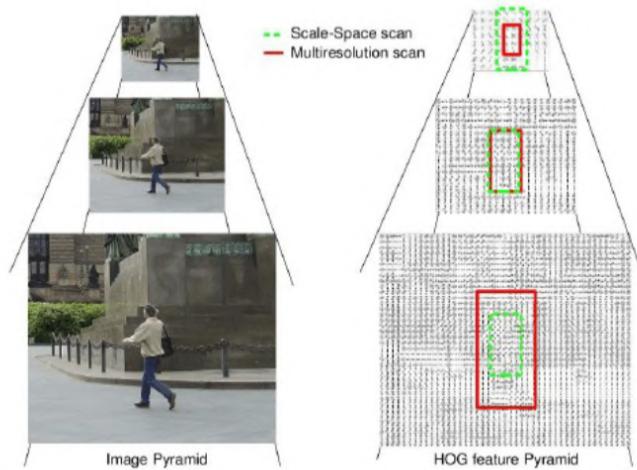
- ▶ **erosione:** $\text{erode}(f, s) = \Theta(c, S)$
- ▶ **maggioranza:** $\text{maj}(f, s) = \Theta(c, S/2)$
- ▶ **apertura:** $\text{open}(f, s) = \text{dilate}(\text{erode}(f, s), s)$
- ▶ **chiusura:** $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$

Multi-risoluzione

- ▶ A volte vogliamo cambiare la risoluzione di un'immagine, ad esempio aumentando o diminuendo la sua dimensione.
- ▶ Aumentare la dimensione di un'immagine richiede un processo di interpolazione.
- ▶ Diminuire la dimensione può essere fatto mediante decimazione dei pixel. Per limitare l'aliasing, idealmente convolviamo l'immagine con un filtro passa basso e poi decimiamo.
- ▶ In alcune applicazioni, come quelle relative alla ricerca di oggetti, non conosciamo la scala alla quale cercare l'oggetto. In questo caso, si utilizzano le piramidi di immagini. In una piramide, si considerano immagini con risoluzioni diverse.
- ▶ Nella piramide dell'immagine, ogni livello può evidenziare diversi livelli di dettaglio, dal fine (in basso) al grosso (in alto).
- ▶ In pratica, ogni livello è ottenuto convolvendo con un filtro gaussiano e poi decimando l'immagine.

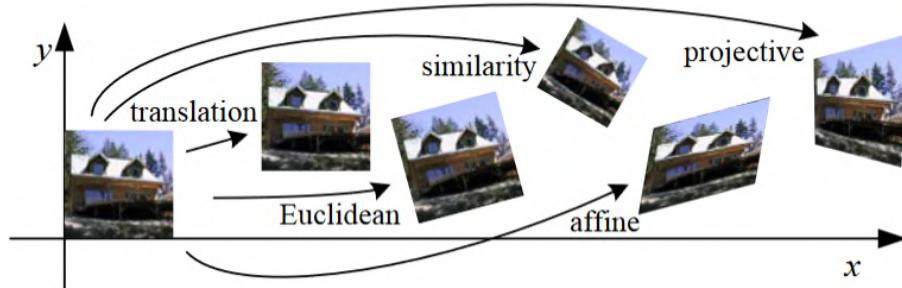
Esempio - Rilevamento Pedoni

- HOG = Istogramma dei Gradienti Orientati
- Per ogni immagine nella piramide, le caratteristiche HOG vengono estratte e utilizzate per rilevare un pedone.
- Anziché cercare all'interno di riquadri di diverse scale, la dimensione del riquadro (quello verde) è fissa e l'immagine viene ridimensionata.

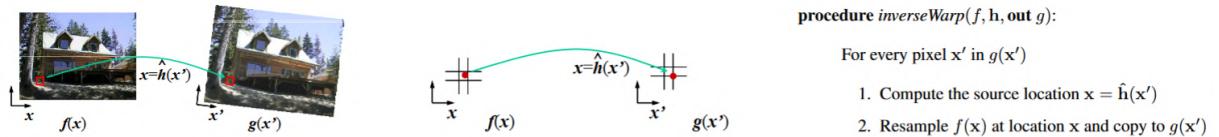


Trasformazioni Geometriche

Le trasformazioni geometriche definiscono mappature sui pixel dell'immagine. Le trasformazioni più comunemente adottate sono mostrate qui sotto: Esistono due modi per definire questa mappatura:



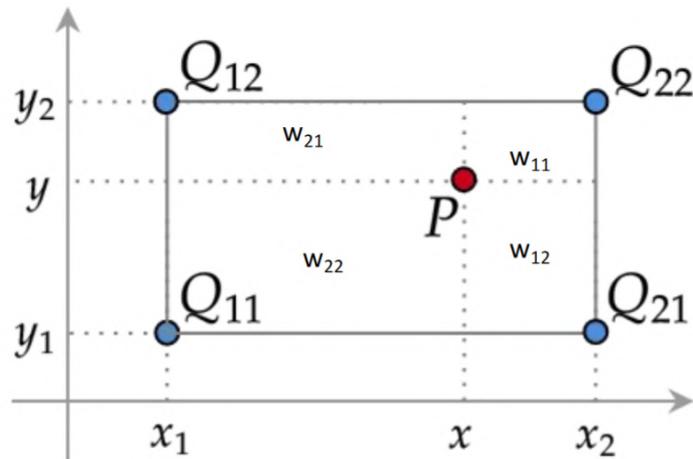
- Mappatura Diretta
- Mappatura Inversa
- Questo approccio soffre di diverse limitazioni. Il processo di copia di un pixel $f(x)$ in una posizione x' in g non è ben definito quando x' ha un valore non intero. Il secondo grande problema con la deformazione diretta è l'apparizione di crepe e buchi, specialmente quando si ingrandisce un'immagine. Infatti, non siamo certi che tutti i pixel dell'immagine di output riceveranno un valore.



- ▶ Applichiamo una trasformazione inversa. Per ogni pixel dell'immagine di output, calcoliamo il valore corrispondente nell'immagine di input.
- ▶ Poiché il risultato potrebbe non essere un intero, applichiamo l'interpolazione del livello di grigio:
 - Interpolazione del vicino più vicino (arrotonda i pixel e prende il livello di quello più vicino).
 - Interpolazione bilineare (usa una somma pesata dei quattro vicini basata sulla loro distanza dal pixel risultante).

Interpolazione Bilineare

- ▶ Dopo la trasformazione, il pixel nell'immagine di input è P .
- ▶ È incluso tra i 4 pixel Q :



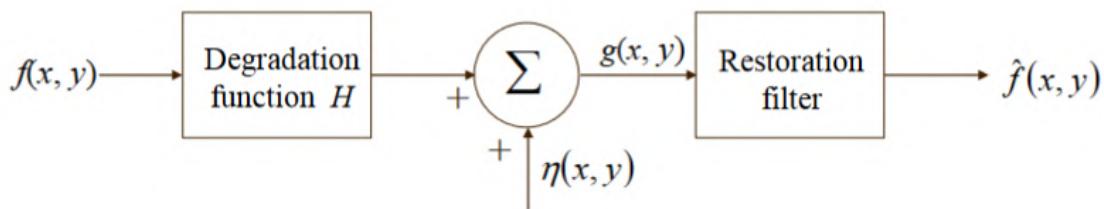
- ▶ Nell'immagine di output, il pixel corrispondente a P viene calcolato come somma pesata dei 4 vicini.
- ▶ I pesi sono le aree indicate nella figura.

3.3 Ripristino delle Immagini

- ▶ Il ripristino delle immagini mira a migliorare un'immagine in un senso predefinito.
- ▶ Il ripristino cerca di recuperare un'immagine degradata utilizzando una conoscenza a priori del fenomeno di degrado.
- ▶ Le tecniche di ripristino sono normalmente basate sulla costruzione di un modello del degrado e sull'applicazione del processo inverso per recuperare l'immagine non degradata.
- ▶ Mentre l'allungamento del contrasto è considerato una tecnica di miglioramento, poiché si basa principalmente sugli aspetti piacevoli che potrebbe offrire alla vista, la rimozione della sfocatura dell'immagine, ad esempio causata da turbolenza atmosferica o movimento della fotocamera, applicando una funzione di deblurring, è considerata una tecnica di ripristino.

Il processo di degradazione/ripristino

Il processo di degradazione può essere modellato come una funzione di degradazione H che, insieme a un termine di rumore additivo h , opera su un'immagine di input $f(x, y)$ per produrre un'immagine degradata $g(x, y)$:



Dato $g(x, y)$: e alcune conoscenze sulla funzione di degradazione H e il termine di rumore additivo h , l'obiettivo del restauro è ottenere una stima (la migliore possibile) dell'immagine originale. Se H è un processo lineare e invariante di posizione, allora l'immagine degradata è data da:

$$g(x, y) = h(x, y) * f(x, y) + \text{rumore}$$

dove $h(x)$ è la rappresentazione spaziale della funzione di degrado ed è chiamata funzione di diffusione dei punti.

Poiché è coinvolta una convoluzione, a volte ci si riferisce al ripristino come deconvoluzione

3.4 Modelli di rumore

Il rumore nelle immagini digitali sorge principalmente durante l'acquisizione e/o la trasmissione delle immagini. Il rumore può essere caratterizzato mediante le sue caratteristiche spaziali o attraverso il suo contenuto in frequenza nel senso di Fourier (ad esempio, il rumore bianco ha uno spettro di Fourier costante).

Un altro aspetto importante è se il rumore è non correlato con l'immagine, cioè non c'è correlazione tra i valori dei pixel e i valori delle componenti di rumore. Alcuni esempi di rumore correlato sono:

- ▶ distorsione a mezzitoni (effetto moiré)
- ▶ corruzione dovuta a interferenze elettriche
- ▶ rumore derivante dall'interazione tra sensore e sorgente del segnale in alcune modalità di imaging medico

Al contrario, esempi di rumore non correlato sono:

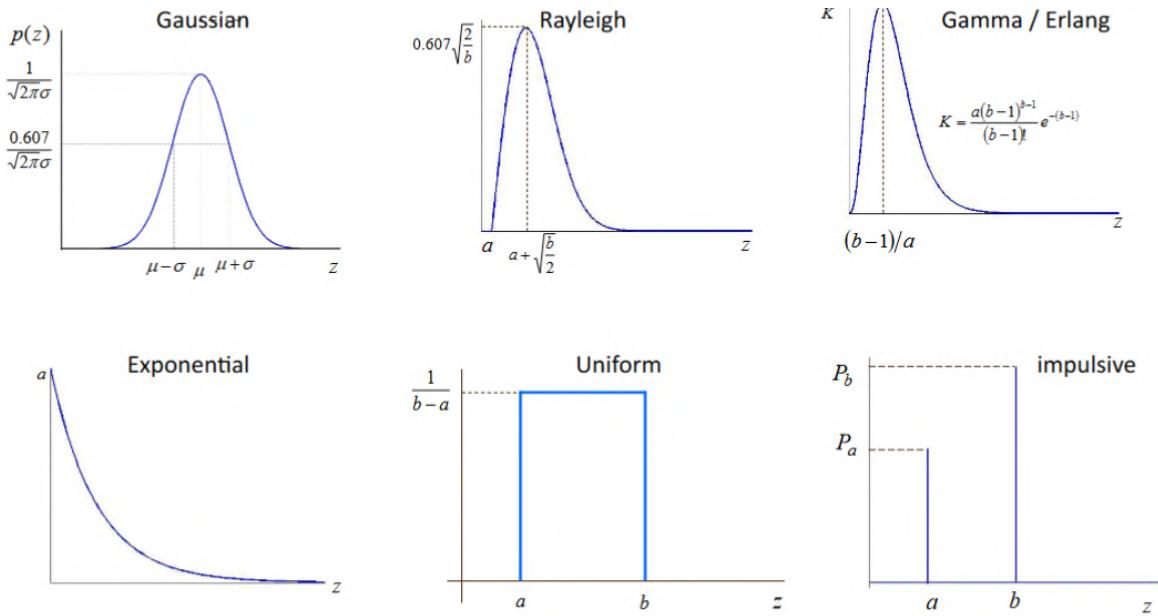
- ▶ rumore quantico nei sensori CCD
- ▶ rumore di quantizzazione durante l'acquisizione dell'immagine

Assumeremo d'ora in poi che il rumore sia indipendente dalle coordinate spaziali (con l'eccezione del rumore spazialmente periodico) e non correlato con l'immagine. Un descrittore spaziale adatto per il rumore è il comportamento statistico dei valori di intensità nella componente di rumore $n(x, y)$ del modello di degradazione/ristorazione.

- ▶ Questa intensità può essere considerata una variabile casuale, caratterizzata da una funzione di densità di probabilità (PDF).
- ▶ Alcuni tra i modelli di rumore più comuni nelle applicazioni di elaborazione delle immagini, che sono anche tra i meno difficili da trattare matematicamente, sono:
 - Gaussiano (o normale)
 - Rayleigh
 - Erlang (o gamma)
 - Esponenziale
 - Uniforme
 - Impulso (o sale-e-pepe)

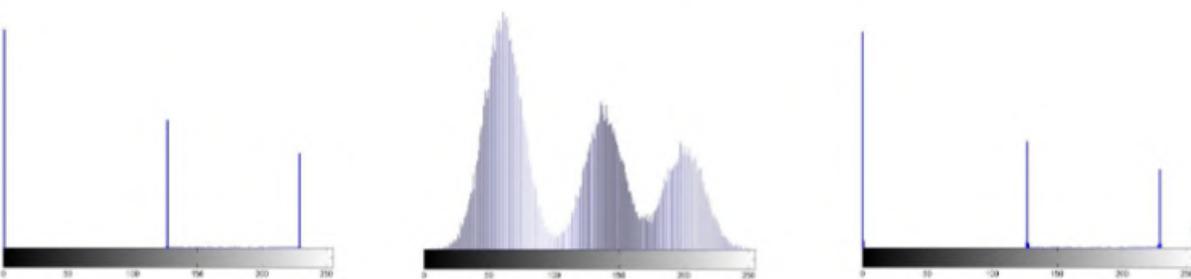
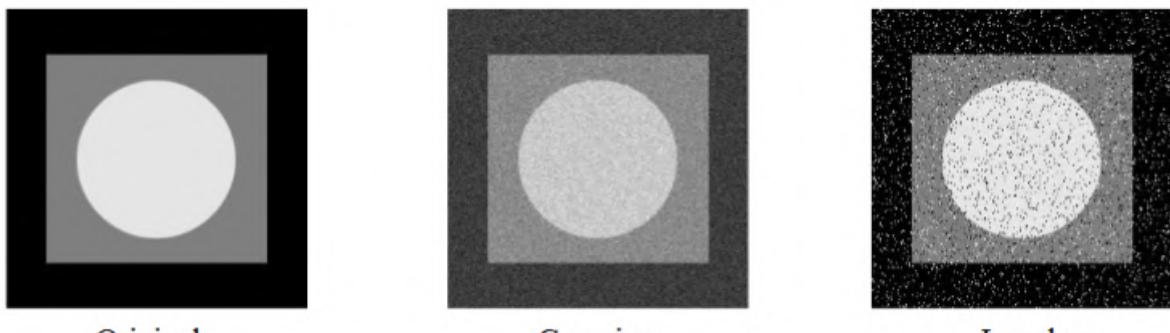
Alcuni tra i modelli di rumore più comuni nelle applicazioni di elaborazione delle immagini sono:

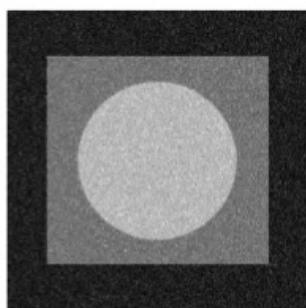
- ▶ Il rumore Gaussiano può rappresentare la corruzione dovuta al rumore dei circuiti elettronici e al rumore dei sensori che si verifica in presenza di scarsa illuminazione e/o alta temperatura.
- ▶ La densità di Rayleigh può essere utilizzata per caratterizzare fenomeni di rumore nelle immagini di profondità.
- ▶ Le densità esponenziale e gamma trovano applicazione nell'imaging laser.
- ▶ Il rumore a impulso è solitamente causato durante l'acquisizione delle immagini, quando possono verificarsi transitori rapidi, come interruttori difettosi.
- ▶ La densità uniforme è forse meno comunemente adottata per descrivere situazioni pratiche, ma trova ampia applicazione nelle simulazioni.



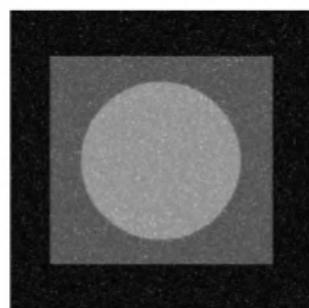
Un pattern di test ben adatto per illustrare i modelli di rumore precedenti è composto da aree semplici e costanti che coprono la scala dei grigi dal nero al quasi bianco in soli tre incrementi, per facilitare l'analisi visiva, alle quali vengono aggiunte componenti di rumore.

Quasi tutte le immagini rumorose non mostrano differenze visive importanti, sebbene i loro istogrammi siano significativamente diversi: solo l'immagine corrotta dal rumore a impulso è visivamente indicativa del tipo di rumore che causa la degradazione.

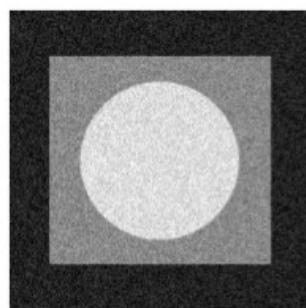




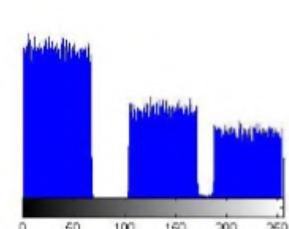
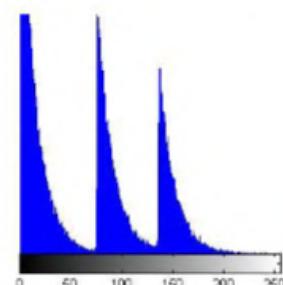
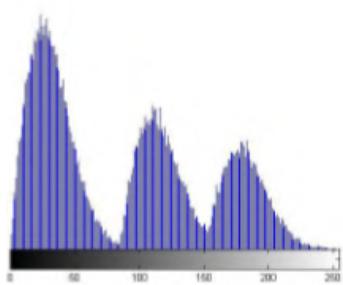
Rayleigh



Exponential

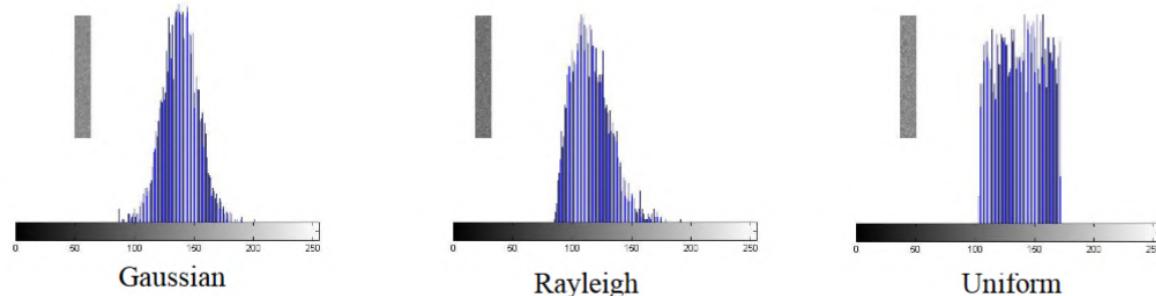


Uniform



Stima dei parametri del rumore

- ▶ Quando sono disponibili solo immagini degradate generate da un sensore (che è il caso più frequente), spesso è possibile stimare i parametri del rumore da piccole aree di intensità ragionevolmente costante (una striscia verticale o orizzontale può essere sufficiente).
- ▶ Negli esempi seguenti, è stata utilizzata una striscia verticale (di 150 x 20 pixel). È stata ritagliata nell'area di intensità intermedia, rispettivamente dalle immagini Gaussiana, Rayleigh e uniforme. Sono mostrati anche gli istogrammi ottenuti dai dati nelle strisce dell'immagine.



Se il modello di rumore selezionato è Gaussiano, la media e la varianza permettono di specificare completamente la PDF(Funzione di Densità di Probabilità); per le altre distribuzioni discusse in precedenza, la media e la varianza possono essere utilizzate per risolvere i parametri a e b .

- ▶ Nel caso del rumore a impulso, dobbiamo stimare P_a e P_b , le probabilità effettive di occorrenza dei pixel neri e bianchi, rispettivamente.
- ▶ A tal fine, possiamo considerare un'area di grigio medio, relativamente costante, sufficientemente grande da permettere un'analisi quantitativa dell'istogramma corrispondente, in modo da poter misurare le altezze dei picchi corrispondenti ai pixel neri e bianchi.

Restaurazione in caso di rumore additivo

- ▶ Quando l'unica degradazione è il rumore additivo, le equazioni del modello di degradazione/restaurazione diventano

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

- ▶ La stima esatta dell'immagine non degradata dovrebbe essere possibile solo se i termini di rumore $n(u, v)$ fossero noti, in modo da poterli sottrarre da $g(x, y)$.
- ▶ Questo è irrealistico, quindi questo approccio potrebbe essere l'eccezione e non la regola.
- ▶ In queste situazioni, è consigliabile adottare il filtraggio spaziale, ad esempio il filtraggio medio (per il rumore Gaussiano) o il filtro mediano (per il rumore impulsivo).

Caratterizzazione delle prestazioni degli algoritmi di restaurazione

- ▶ Tra le metriche per valutare un algoritmo di restaurazione c'è il Rapporto Segnale/Rumore (SNR):
 - Questo rapporto fornisce una misura del livello di potenza del segnale informativo (cioè, dell'immagine non degradata) rispetto al livello di potenza del rumore (differenza tra le immagini degradata e non degradata).

- Le immagini con poco rumore tendono ad avere un alto SNR e, viceversa, la stessa immagine con un livello di rumore più alto ha un SNR più basso.
- Ricorda che le metriche quantitative non necessariamente si correlano bene con la qualità percepita dell'immagine.

MODEL FITTING AND OPTIMIZATION

Model fitting and Optimizazion

4.1 Interpolazione di Dati Sparsi

L'obiettivo dell'interpolazione di dati sparsi è produrre una funzione $f(x)$ che passi attraverso un insieme di punti dati d_k posti nelle posizioni x_k tali che

$$f(x_k) = d_k$$

4.2 Approssimazione di Dati Sparsi

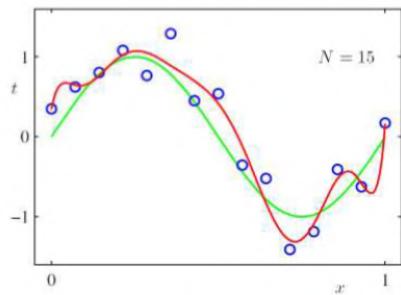
L'obiettivo dell'approssimazione di dati sparsi è produrre una funzione $f(x)$ che passi vicino a un insieme di punti dati d_k posti nelle posizioni x_k tali che

$$f(x_k) \approx d_k$$

In questo caso, vogliamo minimizzare l'errore medio che commettiamo utilizzando f :

$$E_D = \sum_k \|f(x_k) - d_k\|^2$$

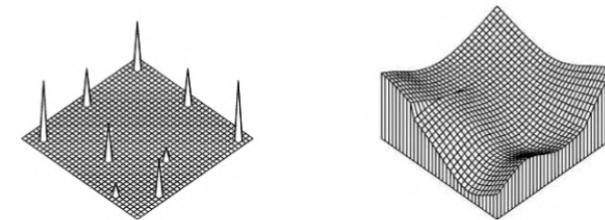
Questo problema prende diversi nomi. Viene chiamato **Regressione** in statistica e machine learning. x è l'input, t è il target



1D example: x is the input, t is the target

N is the number of data points

The 2 functions are possible approximations of the data



2D example: 9 2D samples, the function approximate a surface

N è il numero di punti dati

Le 2 funzioni sono possibili approssimazioni dei dati

4.3 Cosa è f

Nell'approssimazione di dati sparsi, f rappresenta il nostro modello sconosciuto. Un modello è caratterizzato da parametri: cambiando i parametri abbiamo un modello diverso. Quando si risolve un problema di regressione dobbiamo:

1. Identificare l'input e l'output del problema.
2. Definire la formulazione matematica del nostro modello per mappare l'input all'output. La formulazione mette in evidenza i parametri del modello e la forma della funzione.
3. Trovare i parametri del modello.

Come trovare i parametri del modello - Intuizioni

- ▶ Le coppie (input, output) possono essere potenzialmente infinite.
- ▶ Poiché non abbiamo la possibilità di gestire coppie infinite, consideriamo una popolazione di campioni, ossia un insieme limitato di dati di input per cui conosciamo il corrispondente output.
- ▶ Utilizziamo questo insieme limitato di coppie per stimare i parametri del modello risolvendo un problema di ottimizzazione, che tenta di minimizzare l'errore del modello sui dati noti.

Questo significa fondamentalmente che:

- ▶ Il nostro modello sarà un'approssimazione del vero modello che regola i fenomeni che stiamo modellando (infatti, minimizziamo l'errore, probabilmente non avremo mai un errore pari a 0).
- ▶ Il nostro modello può funzionare in modo eccellente sui dati che conosciamo ma, molto probabilmente, commetterà errori sui dati che non appartengono alla popolazione utilizzata per stimare i suoi parametri perché non possiamo usare tutte le possibili coppie input-output.
- ▶ Cambiando la popolazione, probabilmente otterremo parametri diversi (e quindi un modello diverso).
- ▶ I dati spesso provengono da misurazioni (ad esempio, le immagini provengono da un dispositivo che rileva la luce) e sono affetti da rumore. Quindi, anche i dati non sono "perfetti".

Approssimazione di Dati Sparsi

Esistono tecniche che utilizzano campioni distribuiti in una griglia (interpolazione basata su mesh). In alcuni casi, tramite triangolazione, è possibile stimare l'interpolante lineare a tratti (il valore regredito, ossia l'output, dipende dai valori ai vertici del triangolo a cui appartiene il punto).

In dimensioni superiori, è comune usare approcci senza mesh che definiscono l'interpolante desiderato come una somma pesata di funzioni base (chiamate anche funzioni kernel o kernel in breve).

In questi casi, il modello f è definito come:

$$f(x) = \sum_k w_k \phi(\|x - x_k\|)$$

dove i x_k sono le posizioni dei punti dati sparsi, le ϕ sono le funzioni base radiali (o kernel), e i w_k sono i pesi locali associati a ciascun kernel. Le funzioni base ϕ sono chiamate radiali perché sono applicate alla distanza radiale tra un campione di dati x_k e un punto di valutazione x .

La scelta di ϕ determina le proprietà di levigatezza dell'interpolante, mentre la scelta dei pesi w_k determina quanto strettamente la funzione approssima l'input.

Alcune funzioni base comunemente utilizzate includono:

- ▶ Gaussiana $\phi(r) = \exp(-r^2/c^2)$
- ▶ Multiquadric di Hardy $\phi(r) = \sqrt{r^2 + c^2}$
- ▶ Multiquadric inverso $\phi(r) = \frac{1}{\sqrt{r^2 + c^2}}$
- ▶ Spline a piastre sottili $\phi(r) = r^2 \log r$

In queste equazioni, $r = \|x - x_k\|$ è la distanza radiale e c è un iperparametro di scala che controlla la dimensione (falloff radiale) delle funzioni base e quindi la sua levigatezza (basi più compatte portano a funzioni più levigate).

esempio:

Consideriamo 4 punti $\{x_1, x_2, x_3, x_4\}$ e scegliamo un kernel gaussiano con $c = 2$ (che è il valore di sigma). Il modello sarà:

$$f(x) = \sum_{i=1}^4 w_i e^{-\frac{\|x-x_i\|^2}{2c^2}}$$

Sapendo i 4 punti e i pesi, possiamo tracciare la funzione f .

Funzioni a Base Radiale

In generale, conosciamo i campioni di dati x_k e i valori attesi d_k , che sono i valori che ci aspettiamo il modello restituiscia quando i campioni x_k vengono forniti in input (in pratica, vogliamo che $f(x_k) = d_k$). Vogliamo stimare i pesi w_k in modo che il modello sia in grado di restituire valori accurati per qualsiasi campione, anche se questo campione non è incluso nel dataset fornito. Per fare ciò, risolveremo un sistema lineare di equazioni:

$$f(x_k) = \sum_l w_l \phi(\|x_k - x_l\|) = d_k$$

Per grandi quantità di sovrapposizione delle funzioni di base (grandi valori di c), queste equazioni possono essere mal condizionate (piccole variazioni nei valori dei dati o nelle posizioni possono causare grandi cambiamenti nella funzione interpolata)

Un approccio migliore è risolvere il problema di approssimazione dei dati regolarizzati, che comporta la minimizzazione del problema dei minimi quadrati regolarizzati

$$E(\{w_k\}) = \sum_k \left| \sum_l \mathbf{w}_l \phi(\|\mathbf{x}_k - \mathbf{x}_l\|) - \mathbf{d}_k \right|^2 + \lambda \sum_k |\mathbf{w}_k|^p$$

For each data point \mathbf{x}_k Distance from the regressed values and the desired output Constraint on parameters

- ▶ Nel **problema dei minimi quadrati regolarizzati**, la penalità (l'ultimo termine) aggiunta per vincolare i parametri cambia in base a p e assume significati diversi
- ▶ Se $p = 2$, usiamo la norma L2 e parliamo di **ridge regression** (regolarizzazione di Tikhonov), che ammette una soluzione in forma chiusa in cui aggiungiamo una frazione della matrice identità a una matrice di covarianza per essere sicuri che abbia un rango completo
- ▶ Se $p = 1$, usiamo la norma L1 e parliamo della **LASSO Regression** (LASSO sta per Least Absolute Selection and Shrinkage). Non ha una soluzione comoda in forma chiusa. La soluzione è tipicamente trovata usando la programmazione quadratica o metodi di ottimizzazione convessa più generali.
- ▶ La regressione LASSO forza più voci di w a essere uguali a 0 (cioè, la soluzione è sparsa). La regolarizzazione di Tikhonov forza le voci di w a essere piccole ma non necessariamente 0.

Esempio

Consideriamo il seguente dataset:

$$(1, 1)d_1 = 5.03$$

$$(-1, -3)d_2 = -1.28$$

$$(7, 3)d_3 = 6.46$$

$$(-10, 2)d_4 = 7.40$$

Vogliamo utilizzare il kernel gaussiano con $c = 2$. Dobbiamo quindi stimare il kernel gaussiano sulle distanze euclidiene quadrate tra ciascuna coppia di punti:

$$\begin{bmatrix} 0 & 8 & 40 & 122 \\ 8 & 0 & 80 & 90 \\ 40 & 80 & 0 & 290 \\ 122 & 90 & 290 & 0 \end{bmatrix}$$

$$K = \begin{bmatrix} 1.000000 & 0.135335 & 0.000045 & 0.00 \\ 0.135335 & 1.000000 & 0.000000 & 0.00 \\ 0.000045 & 0.000000 & 1.000000 & 0.00 \\ 0.000000 & 1.000000 & 0.000000 & 1.00 \end{bmatrix}$$

Così, se \mathbf{d} è il vettore $\mathbf{d} = [5.03, -1.28, 6.46, 7.40]^T$, usando la matrice K abbiamo che:

$$K\mathbf{w} = \mathbf{d}$$

Vogliamo minimizzare il seguente problema:

$$E(w) = \|K\mathbf{w} - \mathbf{d}\|^2 + \lambda\|\mathbf{w}\|_2^2$$

Questo problema può essere scritto come:

$$\begin{aligned} E(w) &= (Kw - d)^T(Kw - d) + \lambda w^T w \\ &= w^T K^T Kw - 2w^T K^T d + d^T d + \lambda w^T w \\ &= w^T (K^T K + \lambda I)w - 2w^T K^T d + d^T d \end{aligned}$$

Quindi, l'errore è:

$$E(w) = w^T (K^T K + \lambda I)w - 2K^T d w + d^T d$$

Il gradiente è:

$$\nabla E = 2(K^T K + \lambda I)w - 2K^T d$$

- E, per trovare l'ottimo, dovrebbe essere uguale a 0. Risolvendo per w :

$$\begin{aligned} (K^T K + \lambda I)w &= K^T d \\ w &= (K^T K + \lambda I)^{-1} K^T d \end{aligned}$$

La quale è una soluzione in forma chiusa

$$w = (K^T K + \lambda I)^{-1} K^T d$$

Nel nostro esempio:

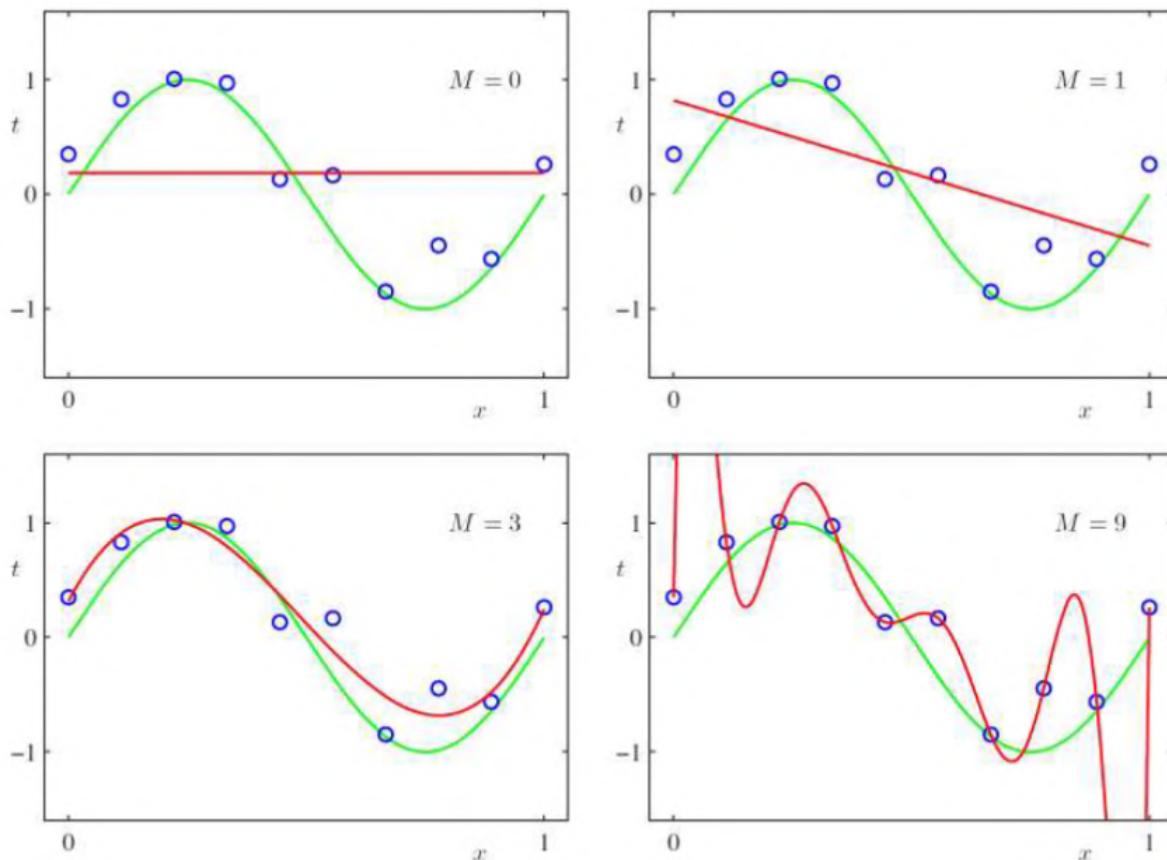
$$K = \begin{bmatrix} 1.000000 & 0.135335 & 0.000045 & 0.00 \\ 0.135335 & 1.000000 & 0.000000 & 0.00 \\ 0.000045 & 0.000000 & 1.000000 & 0.001 \\ 0.000000 & 10.00000 & 0.000000 & 1.00 \end{bmatrix}$$

$$\lambda = 0.0000001$$

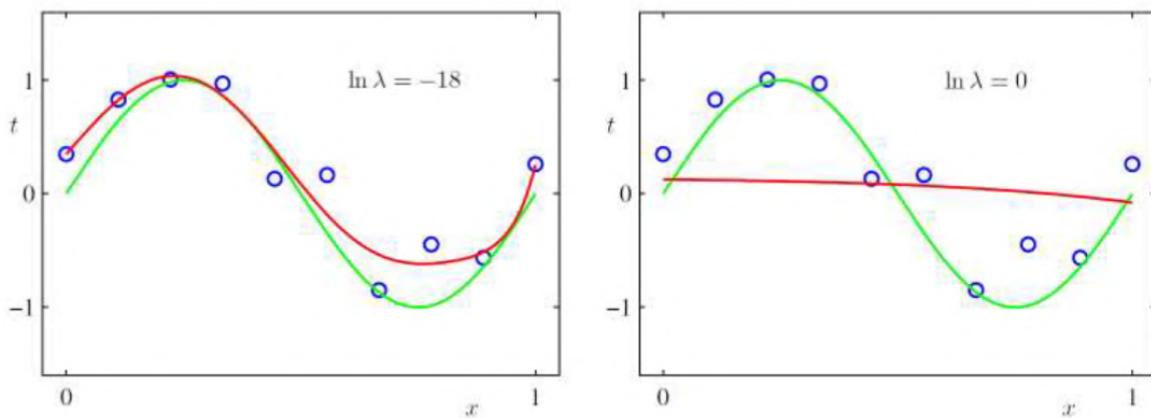
$$w = [5.30000833, -1.99727787, 6.45975874, 7.39999926]$$

4.4 Overfitting e Underfitting

Nella maggior parte dei problemi di fitting dei dati, i campioni d_k (e talvolta anche le loro posizioni x_k) sono rumorosi, quindi adattarli esattamente non ha senso. Consideriamo un semplice esempio di fitting polinomiale. Viene fornito un insieme di campioni rumorosi (mostrati come cerchi). Adattiamo curve polinomiali con diversi ordini M . Aumentare M significa aumentare la complessità del modello (più parametri).



- ▶ $M = 0$ e $M = 1$: underfitting, le curve sono troppo piatte e non rappresentano la tendenza reale dei dati.
- ▶ $M = 3$: sembra rappresentare bene la tendenza reale dei dati.
- ▶ $M = 9$: overfitting, si adatta esattamente ai dati ma non rappresenta il fenomeno reale.

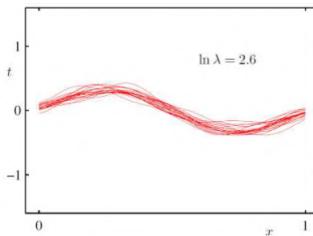


Il grafico a sinistra usa $\lambda = 10^{-18}$, quello a destra usa $\lambda = 1$ (quindi, una regolarizzazione troppo forte).

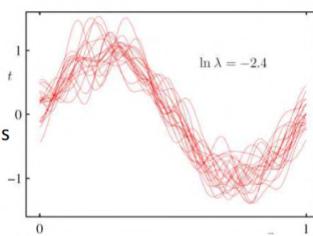
Compromesso Bias-Varianza

- ▶ Il bias nella stima del modello misura l'errore tra le soluzioni fornite dal nostro modello e quelle reali. Un modello con alto bias potrebbe non rappresentare bene le relazioni tra input e target (underfitting).
- ▶ La varianza nella stima del modello è dovuta alla sensibilità a piccole fluttuazioni nel set di addestramento. In pratica, piccole fluttuazioni nel set di addestramento possono originare modelli completamente diversi (overfitting).
- ▶ Con una regolarizzazione troppo forte, abbiamo un bias maggiore e una varianza minore.
- ▶ Con una regolarizzazione troppo debole, abbiamo una varianza maggiore e un bias minore.
- ▶ Quindi, dobbiamo trovare un buon compromesso tra bias e varianza selezionando la giusta quantità di regolarizzazione.
- ▶ A sinistra, rappresentano le curve rosse diversi modelli stimati dopo piccole modifiche al set di allenamento
- ▶ A destra, la curva verde è la realtà di base. La curva rossa è la media di quelli tracciati a sinistra.

Low variance among the estimated models but very high bias

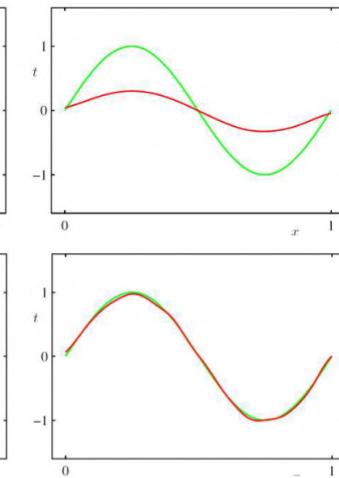


High variance among the estimated models but low bias



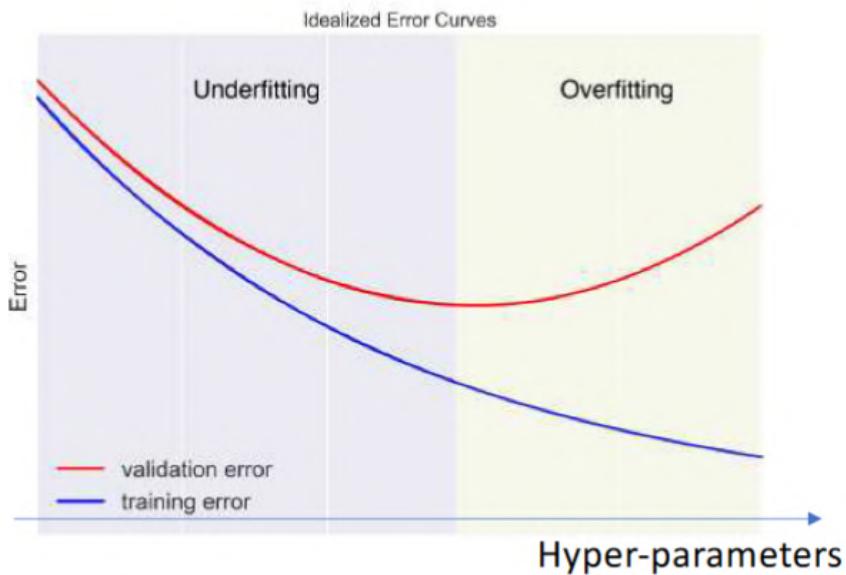
This is a case of underfitting and too strong regularization

This is a case of overfitting and too weak regularization



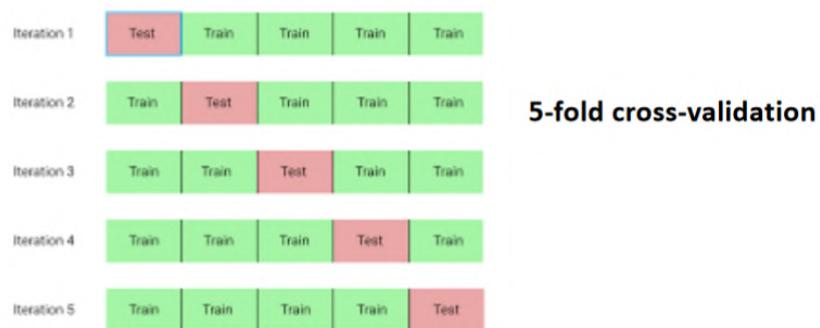
Selezione del Modello

- Nel nostro semplice esempio, ci troviamo nella situazione in cui:
 - I dati disponibili sono limitati
 - Dobbiamo trovare un valore appropriato per λ e per l'ordine del polinomio. Queste variabili sono generalmente chiamate iperparametri perché non possono essere stimati risolvendo il problema di minimizzazione, ma devono essere forniti
 - In base agli iperparametri selezionati, otteniamo modelli diversi e vogliamo trovare un buon compromesso bias-varianza
 - In pratica, dobbiamo selezionare il modello che funziona meglio per il nostro problema
 - Le tecniche di selezione del modello stimano gli iperparametri in un algoritmo di addestramento per ottenere buone prestazioni
- Se ci viene fornito un set di campioni, possiamo salvarne alcuni in un set di validazione per vedere se il modello stimato è sottostimato o sovrastimato.
- Quando variamo un parametro come λ (o qualsiasi altro iperparametro), otteniamo una curva che rappresenta l'errore del nostro modello rispetto al valore assegnato a λ .
 - La curva blu denota l'errore del modello sui dati di addestramento (quelli usati per stimare i parametri del modello)
 - La curva rossa è l'errore del modello sui dati di validazione (che non sono mai stati usati per stimare i parametri del modello)
 - Si dovrebbe selezionare il valore di λ che non causa evidenti sottostima o sovrastima sul set di validazione



Selezione del Modello con la Validazione Incrociata (Cross-Validation)

- Possiamo ripetere il processo di divisione dei nostri dati di campionamento in set di addestramento e di validazione più volte. Una tecnica ben nota, chiamata validazione incrociata, divide i dati di addestramento in K fold (pezzi di uguali dimensioni). A turno, un fold gioca il ruolo del set di validazione e la parte rimanente del set di addestramento.
- Il miglior parametro di regolarizzazione viene trovato facendo la media su tutte le K iterazioni di addestramento. Questa tecnica è considerata troppo costosa quando si addestrano grandi reti neurali a causa dei lunghi tempi di addestramento coinvolti.
- Quando il numero di divisioni è uguale al numero di campioni nel dataset, parliamo di validazione incrociata leave-one-out.



Altre Tecniche di Selezione del Modello

- Il criterio di informazione bayesiano (BIC) e il criterio di informazione di Akaike (AIC) forniscono misure delle prestazioni del modello che tengono conto della complessità del modello.
- Combinano un termine che riflette quanto bene il modello si adatta ai dati con un termine che penalizza il modello in proporzione al numero di parametri (che misura in qualche modo la complessità del modello).

$$AIC = 2k - 2 \ln(L)$$

$$BIC = k \ln(n) - 2 \ln(L)$$

- Dove k è il numero di parametri, n è il numero di campioni, L è una funzione che misura la bontà dell'adattamento, cioè quanto bene il modello spiega i dati forniti (per essere più precisi, è la verosimiglianza del modello...)
- In base a questi criteri, dovremmo favorire il modello con il valore minimo di AIC o BIC.

4.5 Campo Aleatorio di Markov per la riduzione del rumore

Metodi Variazionali

- ▶ Questo tipo di metodologia è anche utilizzato in una famiglia di metodi chiamati Metodi Variazionali (VM) dove le variabili sono, tuttavia, valori continui (quindi avremo integrali nella funzione di perdita).
- I Metodi Variazionali sono tra le tecniche più classiche per l'ottimizzazione di funzioni di costo in dimensioni superiori.
- ▶ Molte sfide nella Visione Artificiale e in altri domini di ricerca possono essere formulate come metodi variazionali.
- ▶ Esempi includono la rimozione del rumore, la sfocatura, la segmentazione delle immagini, il tracciamento, la stima del flusso ottico, la stima della profondità da immagini stereo o la ricostruzione 3D da più viste.
- ▶ Il primo metodo varazionale fu dimostrato da Bernoulli nel 1697, Eulero e Lagrange fornirono successivamente soluzioni più generali.

Questi problemi implicano l'ottimizzazione di funzioni non convesse.

4.6 Metodi Variazionali e Minimizzazione dell'Energia

- ▶ Diverse superfici possono adattarsi ai punti dati forniti.
- ▶ Tuttavia, vogliamo controllare la levigatezza di queste superfici.
- ▶ Per fare ciò, introduciamo norme (misure) sui derivati delle funzioni per trovare soluzioni a energia minima per queste norme.
- ▶ L'apprendimento consiste nel minimizzare una funzione di energia che associa basse energie ai valori corretti e energie più alte ai valori errati.

Metodi Variativi

Molti compiti di visione sono naturalmente posti come problemi di minimizzazione dell'energia su una griglia rettangolare di pixel, dove l'energia comprende un termine di dati e un termine di regolarizzazione:

$$E(u) = E_{\text{dati}}(u) + E_{\text{smoothness}}(u)$$

Il termine dei dati $E_{\text{dati}}(u)$ esprime il nostro obiettivo che il modello ottimale u sia coerente con le misurazioni. L'energia di regolarizzazione $E_{\text{smoothness}}(u)$ deriva dalla nostra conoscenza a priori sulle soluzioni plausibili. Ad esempio:

- ▶ Nel campo della denoising, data un'immagine rumorosa $I'(x, y)$, dove alcune misurazioni possono mancare o essere corrotte, vogliamo recuperare l'immagine originale $I(x, y)$, che tipicamente si assume essere liscia.

- ▶ Da un lato, ci aspettiamo che l'immagine originale I non differisca troppo da I' , il che può essere espresso scegliendo una forma appropriata di E_{dati} .
- ▶ Dall'altro lato, ci si aspetta una liscezza locale dei livelli di grigio che dovrebbe essere applicata scegliendo una forma appropriata di $E_{\text{smoothness}}$.

Molti compiti di visione artificiale sono naturalmente posti come problemi di minimizzazione dell'energia su una griglia rettangolare di pixel, dove l'energia comprende un termine dati e un termine di levigatezza:

$$E(u) = E_{\text{dati}}(u) + E_{\text{smoothness}}(u)$$

Il termine dati $E_{\text{dati}}(u)$ esprime il nostro obiettivo che il modello ottimale u sia coerente con le misurazioni. L'energia di levigatezza $E_{\text{levigatezza}}(u)$ è derivata dalla nostra conoscenza a priori su soluzioni plausibili.

Ad esempio, nel campo della denoising, data un'immagine rumorosa $I'(x, y)$, dove alcune misurazioni possono essere mancanti o corrotte, vogliamo recuperare l'immagine originale $I(x, y)$, che si suppone tipicamente sia levigata. Da un lato, ci aspettiamo che l'immagine originale I non differisca troppo da I' , il che può essere espresso scegliendo una forma appropriata di E_{dati} . D'altra parte, ci si aspetta una levigatezza locale dei livelli di grigio e dovrebbe essere applicata scegliendo una forma appropriata di $E_{\text{levigatezza}}$.

4.7 Metodi Variativi per la smoothness

Supponiamo di voler adattare un modello da un insieme di campioni con alcune proprietà di levigatezza. Possiamo farlo formulando il problema come un problema di minimizzazione dell'energia. Per prima cosa, dobbiamo misurare la levigatezza di un modello, cosa che può essere fatta utilizzando norme sulle derivate delle funzioni. In due dimensioni:

$$\mathcal{E}_1 = \int \left[f_x^2(x, y) + f_y^2(x, y) \right] dx dy = \int \|\nabla f(x, y)\|^2 dx dy$$

Questo funzionale integra la magnitudine del gradiente sulle due dimensioni:

$$\mathcal{E}_2 = \int \left[f_{xx}^2(x, y) + 2f_{xy}^2(x, y) + f_{yy}^2(x, y) \right] dx dy$$

che integra le derivate di secondo ordine. È anche chiamato thin-plate spline.

Intuitivamente, questi funzionali misurano la variazione (non levigatezza) in una funzione e possono essere utilizzati come regolarizzatori nei problemi di fitting, forzando la soluzione a essere levigata.

4.8 Minimizzazione dell'Energia Discreta

Dobbiamo anche discretizzare i funzionali di levigatezza. Considerando come vengono calcolate le derivate delle funzioni discrete, la più semplice espressione di questi funzionali è (nel libro, trovi

versioni più complete):

$$E_1 = \sum_{i,j} [f(i+1,j) - f(i,j)]^2 + [f(i,j+1) - f(i,j)]^2 \quad (4.1)$$

$$\begin{aligned} E_2 = \sum_{i,j} & [f(i+1,j) - 2f(i,j) + f(i-1,j)]^2 + \sum_{i,j} 2 [f(i+1,j+1) - f(i+1,j) - f(i,j+1) + f(i,j)]^2 \\ & + \sum_{i,j} [f(i,j+1) - 2f(i,j) + f(i,j-1)]^2 \end{aligned} \quad (4.2)$$

4.9 Metodi Variativi per Immagini

Nella maggior parte delle applicazioni di visione artificiale, le funzioni che stiamo cercando di modellare sono solo a tratti continue. Ad esempio, nelle immagini a colori, i cambiamenti nell'aspetto degli oggetti (il colore della superficie o l'albedo) introducono discontinuità. Questo è vero anche per le mappe di profondità o i campi ottici.

Per tener conto di queste discontinuità, possiamo controllare localmente la continuità della funzione

$$\mathcal{E}_{CC} = \int \rho(x,y) [1 - \tau(x,y)] [f_x^2(x,y) + f_y^2(x,y)] + \tau(x,y) [f_{xx}^2(x,y) + 2f_{xy}^2(x,y) + f_{yy}^2(x,y)] dx dy$$

dove le funzioni $\rho(x,y) \in [0, 1]$ e $\tau(x,y) \in [0, 1]$ controllano la continuità e la tensione (quanto è piatta) della superficie.

- ▶ In aggiunta al termine di regolarizzazione (smoothness), i problemi variazionali richiedono anche un termine di dati (data penalty).
- ▶ Per la regressione, questo termine misura l'errore del modello sui dati del dataset:

$$E_D = \sum_i [f(x_i; y_i) - d_i]^2$$

- ▶ Pertanto, la funzione da minimizzare per stimare i parametri del modello è:

$$E = E_D + \lambda E_S$$

dove E_S non è definito in questo estratto, ma si riferisce a una qualche forma di misura di smoothness.

- ▶ Infine, l'ultimo termine rappresenta uno dei due funzionali precedenti (o varianti), che impone la smoothness del modello e rappresenta una penalità di smoothness.

4.10 Prospettiva Bayesiana

- ▶ La regolarizzazione implica la minimizzazione di funzionali energetici definiti su funzioni (a tratti) continue.

- ▶ Una tecnica alternativa è formulare un modello bayesiano o generativo, che modella separatamente il processo di formazione dell'immagine rumorosa (misurazione), oltre ad assumere un modello statistico a priori sullo spazio delle soluzioni.
- ▶ Vedremo prima che anche usando un approccio bayesiano finiremo con il minimizzare una funzione di energia.
- ▶ Successivamente, vedremo i Campi Aleatori di Markov (MRF), un modello generativo utilizzato nella visione artificiale. La verosimiglianza logaritmica di un MRF può essere descritta usando termini di interazione (o penalità) di vicinato locale.

4.11 Massima Verosimiglianza a Posteriori

- ▶ Supponiamo di avere una variabile sconosciuta x (x può avere diversi significati qui, può rappresentare i parametri del modello) e possiamo osservare le misurazioni y . Le variabili x e y sono generalmente multidimensionali.
- ▶ Definiamo la distribuzione a posteriori $p(x|y)$ come una funzione di densità (misura quanto è probabile avere il valore x dato i dati osservati).
- ▶ Chiamiamo $p(y|x)$ la verosimiglianza sui dati disponibili (misura quanto bene la conoscenza di x spiega le misurazioni osservate).
- ▶ Denotiamo $p(x)$ il priore (codifica la nostra conoscenza di x senza considerare alcuna osservazione).

Basato sulla regola di Bayes:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

E prendendo il logaritmo negativo:

$$-\log p(x|y) = -\log p(y|x) - \log p(x) + C$$

Che è la verosimiglianza logaritmica negativa a posteriori.

- ▶ Per trovare il valore più probabile di x (MAP) dato y , vogliamo minimizzare questo logaritmo negativo della verosimiglianza a posteriori:

$$-\log p(x|y) = -\log p(y|x) - \log p(x) + C$$

- ▶ Questa espressione ha un termine che dipende dai dati y e un termine che dipende dalla prior su x . Assomiglia a un problema di minimizzazione dell'energia:

$$E(x, y) = E_d(x, y) + E_p(x)$$

- ▶ Pertanto, minimizzare l'energia o trovare le soluzioni MAP sono equivalenti.

4.12 Campi Aleatori di Markov

I Campi Aleatori di Markov sono modelli probabilistici definiti su griglie di pixel o voxel bidimensionali o tridimensionali. I MRF sono un caso speciale della più generale famiglia di modelli grafici. Quando si usano i MRF per applicazioni di elaborazione delle immagini, le variabili sconosciute x sono l'insieme dei pixel di output (ciò che vogliamo stimare), mentre y sono i pixel di input (ciò che osserviamo). I MRF sono ampiamente usati per la rimozione del rumore e la segmentazione. Quindi, il significato di x dipende dal problema che vogliamo risolvere. Infatti, x può essere il valore del pixel denoised o l'etichetta da assegnare a ciascun pixel.

$$x = f(0, 0) \dots f(m - 1, n - 1) \quad (4.3)$$

$$y = a(0, 0) \dots a(m - 1, n - 1) \quad (4.4)$$

Come già detto, dobbiamo formulare il problema in modo da ottenere una funzione nella forma

$$E(x, y) = E_D(x, y) + E_P(x)$$

In MRF, la penalità sui dati E_D rappresenta il legame tra x e y e dipende dal problema da risolvere.

L'energia a priori E_P esprime i vincoli sui pixel nello stesso vicinato. È generalmente espressa come

$$E_P(x) = \sum_{\{(i,j),(k,l)\} \in \mathcal{N}(i,j)} V_{i,j,k,l}(f(i, j), f(k, l)),$$

A seconda di V , questa penalità a volte corrisponde all'uso di una distribuzione di Boltzmann come prior $p(x)$

$$p_i = \frac{1}{Q} e^{-\frac{\epsilon_i}{kT}}$$

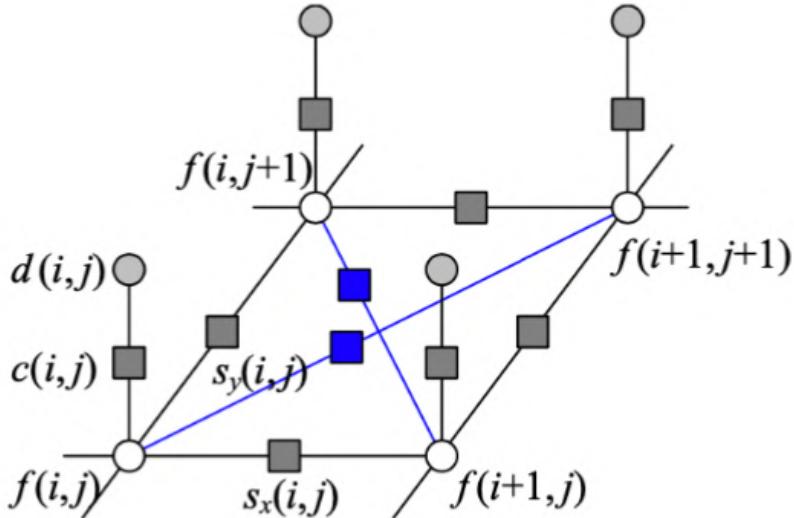
(La distribuzione di Boltzmann è una distribuzione di probabilità che fornisce la probabilità di uno stato particolare in funzione dell'energia di quello stato e della temperatura del sistema a cui è applicata)

K è la costante di Boltzmann, T è la temperatura, ϵ è l'energia dello stato i , Q è utilizzato per normalizzare e ottenere una distribuzione valida.

4.13 Campo Aleatorio di Markov come Modello Grafico

- ▶ Variabili Casuali: $f(i, j)$ sono le variabili casuali sconosciute che vogliamo stimare, $d(i, j)$ sono le variabili osservate.
- ▶ Collegamenti (fattori): $s(i, j)$ rappresentano le interazioni tra le variabili sconosciute e sono determinati dai Potenziali di interazione. Il loro numero dipende dal tipo di vicinato che consideriamo (N4 o N8). $c(i, j)$ denotano le funzioni di penalità dei dati e collegano le variabili osservate e sconosciute, sono chiamati Potenziali unari. I fattori sono funzioni di probabilità (improprie) il cui prodotto è la distribuzione a posteriori (non normalizzata).

- ▶ I vicini definiscono cliques all'interno del modello grafico. I cliques sono unità base che catturano tutte le dipendenze possibili che non devono essere trascurate. Nei modelli grafici non orientati, la distribuzione congiunta può essere fattorizzata in un prodotto di potenziali di clique (unari e di interazione). Questi potenziali di clique rappresentano una nozione di "bontà" o "compatibilità" delle variabili.



4.14 Campi Aleatori di Markov Binari

- ▶ In un campo aleatorio di Markov binario, la variabile x è binaria. Può essere usata per denoisare immagini scansionate in bianco e nero così come nella segmentazione immagine sfondo/oggetto.
- ▶ Per scopi di denoising, la penalità dei dati di un pixel (i, j) riflette l'accordo tra le immagini scansionate e finali

$$E_D(i, j) = w\delta(f(i, j), d(i, j))$$

- ▶ La penalità di levigatezza esprime il livello di accordo tra pixel vicini

$$E_P(i, j) = s\delta(f(i, j), f(i + 1, j)) + s\delta(f(i, j), f(i, j + 1)).$$

- ▶ Una volta formulata l'energia, come la minimizziamo? L'approccio più semplice è eseguire la discesa del gradiente, invertendo uno stato alla volta se produce una energia più bassa. Questo approccio è conosciuto come classificazione contestuale, modelli condizionali iterati (ICM), o primo con la massima confidence (HCF) se il pixel con la maggiore diminuzione di energia è selezionato per primo.
- ▶ Sfortunatamente, questi metodi discendenti tendono a rimanere facilmente bloccati nei minimi locali. Un approccio alternativo è aggiungere un po' di casualità al processo, che è conosciuto come discesa del gradiente stocastica. Quando la quantità di rumore è diminuita nel tempo, questa tecnica è conosciuta come ricottura simulata ed è stata prima popolarizzata nella

visione artificiale da German e German e successivamente applicata alla corrispondenza stereo da Barnard, tra gli altri.

- ▶ Anche questa tecnica, tuttavia, non performa così bene. Per immagini binarie, una tecnica molto migliore, introdotta alla comunità della visione artificiale da Boykov, Veksler, e Zabih è riformulare la minimizzazione dell'energia come un problema di ottimizzazione max-flow/min-cut. Questa tecnica è conosciuta informalmente come graph cuts nella comunità della visione artificiale. Per funzioni di energia semplici, ad esempio quelle in cui la penalità per pixel vicini non identici è una costante, questo algoritmo è garantito per produrre la soluzione ottimale.

$$E_D(i, j) = w\delta(f(i, j), d(i, j))$$

Example

At iteration k, assume w=s=-1:

0	0	1	0	0
0	1	1	1	0
1	1	1	1	0
0	1	0	1	0
0	0	0	0	1

Noisy Image d

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	0	1	0
0	0	0	0	0

Restored image f

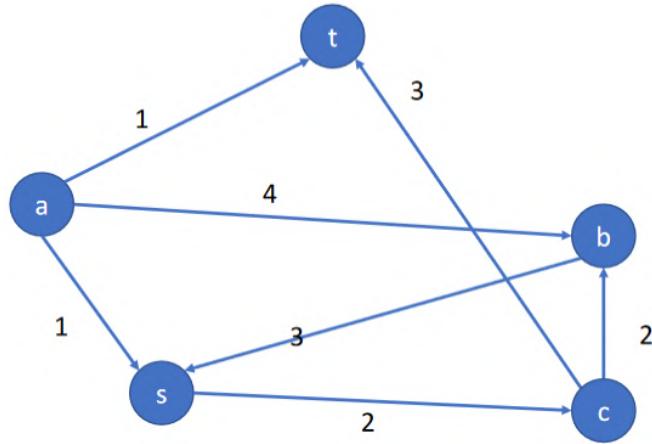
And so on

-1	-1	0	-1	-1
-1	-1	-1	-1	-1
0	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	0

$$E_P(i, j) = s\delta(f(i, j), f(i + 1, j)) + s\delta(f(i, j), f(i, j + 1)) + \dots$$

4.15 Binary Markov Random Fields and Graph-Cut

Consideriamo un grafo $G(V, E)$ dove V sono i nodi ed E sono gli archi, e ad ogni arco è stato associato un nodo c .



Concentriamoci su due vertici, s e t .

Un taglio di G rispetto a s e t è una partizione dei vertici del grafo in due gruppi S e T tale che

$$V = S \cup T, \quad S \cap T = \emptyset$$

$$s \in S, \quad t \in T$$

La somma totale dei pesi degli archi che vanno da S a T è chiamata *costo* del taglio.

$$\text{Costo} = 1 + 4 + 3 + 2 \quad (\text{l'arco da } b \text{ a } s \text{ non è incluso})$$

$$S = \{s, a, c\}, \quad T = \{t, b\}$$

Un taglio è quindi una partizione del grafo in due insiemi.

Ci sono molti tagli possibili rispetto a s e t .

Ogni taglio ha il proprio costo.

Un problema generale nella teoria dei grafi è calcolare il taglio con il costo minimo (*min-cut*).

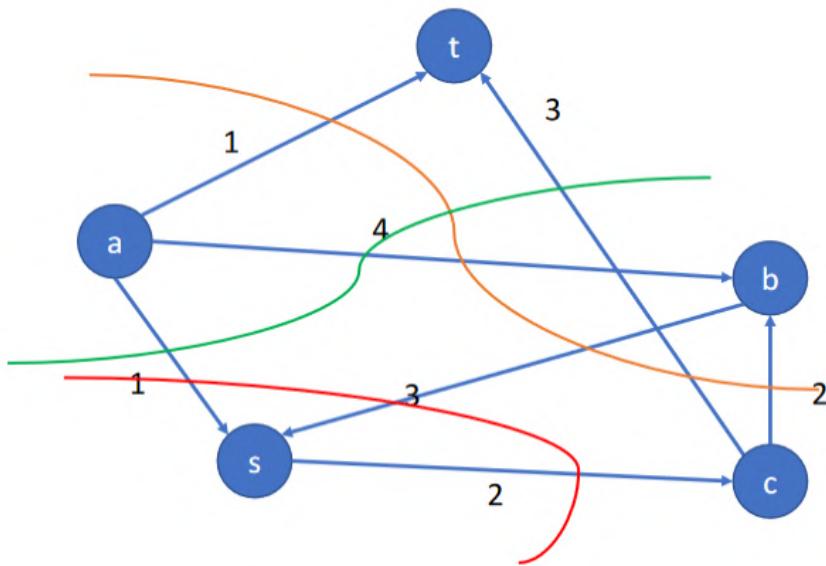
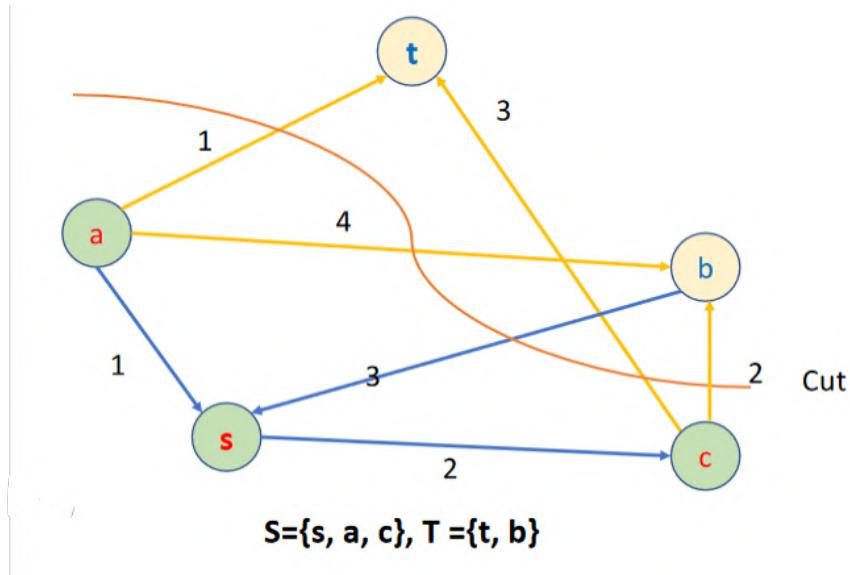
L'algoritmo che risolve il problema è chiamato *graph-cut()*.

Il problema del min cut è il duale del problema del max flow.

Nel max flow, assumiamo che ci sia un nodo sorgente s e un nodo bersaglio t .

Il nostro obiettivo è massimizzare il flusso che può attraversare gli archi del grafo considerando che ogni arco ha una capacità limitata (il costo dell'arco).

L'algoritmo per calcolare il max flow (ad esempio l'algoritmo di Ford-Fulkerson ma esistono altri metodi) determina iterativamente un percorso nel grafo alla volta, per muovere il massimo flusso possibile.



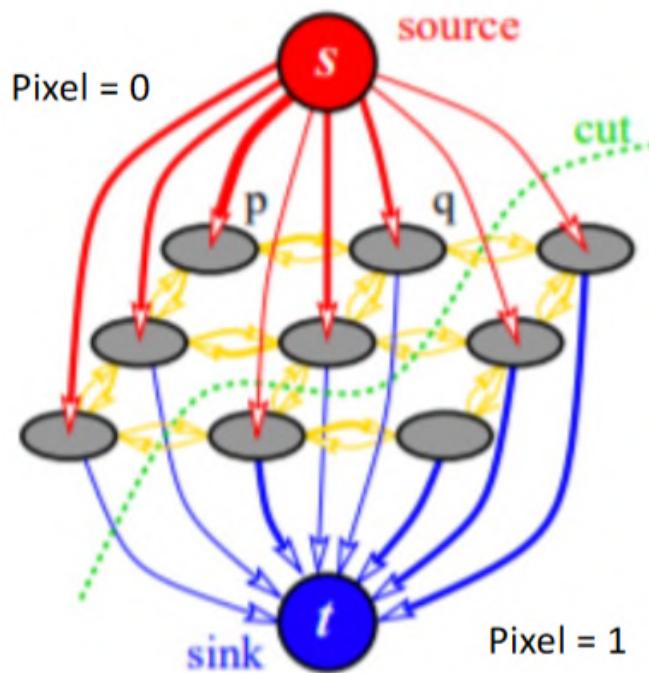
Una volta che non ci sono più percorsi da trovare nel grafo perché gli archi si saturano, abbiamo trovato il max flow.

L'insieme degli archi saturi è il taglio minimo corrispondente.

Quando i pesi sono tutti non negativi, il max flow può essere trovato in tempo polinomiale.

Min Cut per Immagini Binari

- ▶ Tutti i pixel sono nodi del grafo.
- ▶ Includiamo due ulteriori vertici: il nodo sorgente s e il nodo sink t .
- ▶ Gli archi tra i pixel sono quelli necessari per garantire l'adiacenza N_4 .
- ▶ s e t sono collegati con tutti i pixel della nostra immagine.
- ▶ Gli archi rappresentano:
 - Il costo di appartenere a una classe o all'altra (foreground/background o 0-1 nelle immagini in bianco e nero rumorose) se incidente sulla sorgente/sink.
 - Il potenziale di coppia tra i pixel (accordo nel vicinato).

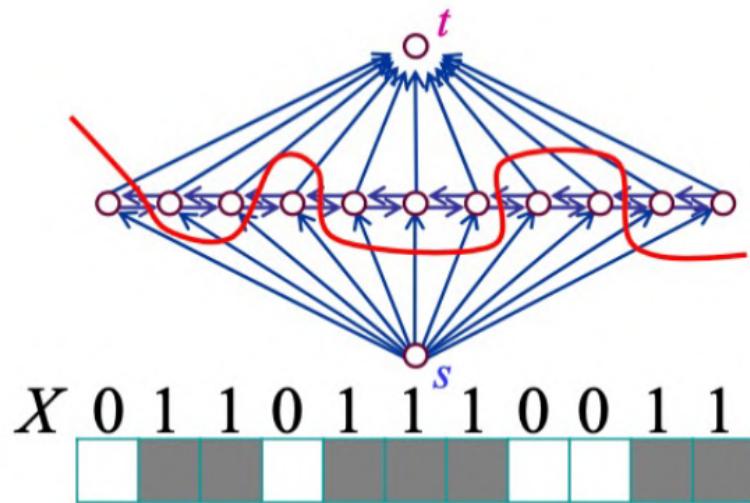


(b) A cut on \mathcal{G}

Interpretazione 1-D

Per semplificare l'interpretazione, consideriamo il seguente grafo:

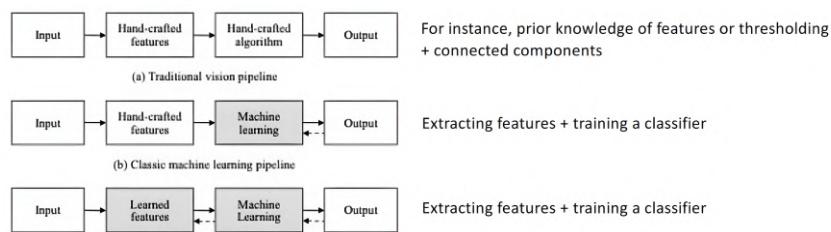
Una volta trovato il min-cut, il pixel nell'insieme S assume il valore 0 e i pixel nell'insieme T assumono il valore 1.



LEARNING

5.1 Evoluzione della Visione Artificiale

Le tecniche di machine learning hanno sempre avuto un ruolo importante e spesso centrale nello sviluppo degli algoritmi di visione artificiale. Attualmente, le reti neurali profonde sono i modelli di machine learning più popolari e ampiamente utilizzati nella visione artificiale.



5.2 Problemi da Risolvere

I principali problemi da risolvere includono la classificazione e la regressione.

- ▶ **Classificazione:** ci viene data una collezione di input $\{x_i\}$, che possono essere caratteristiche derivate dalle immagini di input, abbinate alle rispettive etichette di classe (o target) $\{t_i\}$, che provengono da un insieme di classi $\{C_k\}$.
- ▶ **Regressione:** ci viene data una collezione di input $\{x_i\}$, che possono essere caratteristiche derivate dalle immagini di input, abbinate a output scalari o vettoriali con valori reali $\{y_i\}$, che è l'output atteso del nostro modello.

5.3 Apprendimento

L'apprendimento può essere supervisionato, non supervisionato o semi-supervisionato.

- ▶ **Apprendimento supervisionato:** abbiamo input e output attesi o etichette
- ▶ **Apprendimento non supervisionato:** abbiamo input ma non vengono fornite etichette o output
- ▶ **Apprendimento semi-supervisionato:** le etichette o gli output sono forniti solo per un sottoinsieme dei campioni di input

5.4 Apprendimento Supervisionato

L'algoritmo di apprendimento regola i parametri del modello in modo da massimizzare l'accordo tra le previsioni del modello e gli output target (etichette o valori reali/vettoriali).

Nella fase di addestramento, tutti i dati di addestramento (coppie input-output etichettate) sono processati (spesso iterando su di essi molte volte).

Il modello addestrato può quindi essere utilizzato per prevedere nuovi valori di output per input precedentemente non visti. Questa fase è spesso chiamata fase di test.

Informalmente, il processo di apprendimento appare così: raccogliere una grande collezione di esempi per i quali le covariate sono note e selezionarne un sottoinsieme casuale, acquisendo le etichette di verità a terra per ciascuno. A volte queste etichette potrebbero essere dati già raccolti (ad esempio, un paziente è morto nell'anno successivo?) e altre volte potremmo dover impiegare annotatori umani per etichettare i dati (ad esempio, assegnare immagini a categorie).

Insieme, questi input e le corrispondenti etichette costituiscono il set di addestramento. Alimentiamo il dataset di addestramento in un algoritmo di apprendimento supervisionato, una funzione che prende come input un dataset e restituisce un'altra funzione, il modello addestrato. Infine, possiamo alimentare input precedentemente non visti al modello addestrato, usando i suoi output come previsioni dell'etichetta corrispondente.

Regressione

Forse il compito di apprendimento supervisionato più semplice da capire è la regressione. Considera, ad esempio, un set di dati raccolti da un database di vendite di case. Potremmo costruire una tabella, dove ogni riga corrisponde a una casa diversa e ogni colonna corrisponde a qualche attributo rilevante, come la metratura della casa, il numero di camere da letto, il numero di bagni e il numero di minuti (a piedi) dal centro della città. In questo dataset ogni esempio sarebbe una casa specifica e il corrispondente vettore di caratteristiche sarebbe una riga nella tabella.

Se vivi a New York o San Francisco e non sei il CEO di Amazon, Google, Microsoft o Facebook, il vettore di caratteristiche (metratura, numero di camere da letto, numero di bagni, distanza a piedi) per la tua casa potrebbe essere qualcosa come: [100, 0, .5, 60]. Tuttavia, se vivi a Pittsburgh, potrebbe essere più simile a [3000, 4, 3, 10]. Vettori di caratteristiche come questo sono essenziali per la maggior parte degli algoritmi classici di machine learning. Continueremo a denotare il vettore di caratteristiche corrispondente a qualsiasi esempio i come x_i e possiamo riferirci compatta alla tabella completa contenente tutti i vettori di caratteristiche come X .

Ciò che rende un problema di regressione sono effettivamente gli output. Supponi di essere sul mercato per una nuova casa. Potresti voler stimare il valore di mercato equo di una casa, date alcune caratteristiche come queste. Il valore target, il prezzo di vendita, è un numero reale. Se ricordi la definizione formale dei numeri reali, potresti grattarti la testa ora. Le case probabilmente non vendono mai per frazioni di centesimo, per non parlare di prezzi espressi come numeri irrazionali. In casi come questo, quando il target è effettivamente discreto, ma dove l'arrotondamento avviene su una scala sufficientemente fine, abuseremo un po' del linguaggio e continueremo a descrivere i nostri output e target come numeri reali.

Denotiamo qualsiasi target individuale y_i (corrispondente all'esempio x_i) e l'insieme di tutti i target y (corrispondenti a tutti gli esempi X). Quando i nostri target assumono valori arbitrari in un certo intervallo, chiamiamo questo un problema di regressione. Il nostro obiettivo è produrre un modello le cui previsioni approssimino strettamente i valori reali.

5.5 Cosa significa “massimizzare l'accordo”?

Nella classificazione, il modello addestrato occasionalmente commetterà errori sotto dati incerti, rumorosi e/o incompleti. Il nostro obiettivo è massimizzare l'utilità attesa del modello, o al contrario, minimizzare la sua perdita o rischio atteso. Per fare ciò, dovremmo conoscere la vera distribuzione di probabilità sugli input. Purtroppo, non la conosciamo! Tuttavia, possiamo conoscere (o stimare) la distribuzione dei dati di addestramento e speriamo che la vera distribuzione di probabilità sugli input sia vicina a quella.

Utilizzando la distribuzione dei dati di addestramento eseguiamo una minimizzazione del rischio empirico, dove il rischio atteso (perdita) può essere stimato con

$$E_{\text{Risk}}(w) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i; w))$$

dove:

- ▶ $f()$ è il modello da addestrare, che restituisce l'output previsto
- ▶ w sono i parametri del modello
- ▶ x_i è il campione di input, y_i è l'etichetta attesa corrispondente
- ▶ N è il numero di campioni nel set di addestramento
- ▶ E è il valore atteso del rischio
- ▶ L è la funzione di perdita e misura il “costo” di prevedere un output $f(x_i; w)$ per l'input x_i e i parametri del modello w quando il target corrispondente è y_i

La funzione di perdita $L()$ prende in input l'output vero e l'output previsto. Generalmente, utilizziamo diverse funzioni di perdita a seconda del problema da risolvere, regressione o classificazione. Ad esempio, spesso nei problemi di regressione utilizziamo il MSE o le sue varianti. Nei compiti di classificazione, è comune minimizzare il tasso di errore di classificazione.

5.6 Pre-elaborazione dei Dati

Prima di addestrare un modello con alcune tecniche di machine learning, i dati sono generalmente pre-elaborati. La pre-elaborazione in generale include:

- ▶ Centralizzazione dei vettori di caratteristiche, cioè sottraendo il loro valore medio
- ▶ Standardizzazione dei vettori di caratteristiche, cioè, dopo la centralizzazione dei dati, ridimensionamento di ogni componente in modo che la sua varianza sia 1

- Whitening dei dati, se possibile. Lo Whitening comporta il calcolo della matrice di covarianza degli input, prendendo la sua SVD e poi ruotando il sistema di coordinate in modo che le dimensioni finali siano non correlate e abbiano varianza unitaria. È generalmente fatto per input a bassa dimensione poiché è una pre-elaborazione costosa per grandi set di immagini.

Ciascuno di questi passaggi di pre-elaborazione richiede il calcolo di valori come media e deviazione standard. È importante ricordare che questi valori devono essere calcolati solo sul set di addestramento. Successivamente, il set di test deve essere processato utilizzando la stessa procedura e i valori stimati per pre-elaborare il set di addestramento.

5.7 Approcci di Apprendimento Supervisionato Popolari

Gli approcci di apprendimento supervisionato popolari includono Nearest Neighbor (NN), che assegna l'etichetta del campione più vicino, e K-Nearest Neighbor (K-NN), che assegna l'etichetta più frequente tra i K vicini. La Fast Library for Approximate Nearest Neighbors (FLANN) è incorporata come parte di OpenCV e include diverse varianti come k-d trees randomizzati e hashing sensibile alla località (LSH).

Nearest Neighbor (NN) e KNN

Il **Nearest Neighbor (NN)** e il **K-Nearest Neighbors (KNN)** sono algoritmi di apprendimento supervisionato utilizzati principalmente per problemi di classificazione e regressione.

Nearest Neighbor (NN)

Il Nearest Neighbor è un metodo molto semplice per classificare o stimare un nuovo punto dati basato sul punto dati più vicino nel set di addestramento.

Funzionamento di base:

1. **Memorizzazione dei dati di addestramento:** NN memorizza semplicemente tutti i punti dati del set di addestramento.
2. **Classificazione o Predizione:** Per un nuovo punto dati, l'algoritmo trova il punto dati più vicino (il "nearest neighbor") nel set di addestramento. La classe o il valore di questo punto vicino viene assegnato al nuovo punto dati.

K-Nearest Neighbors (KNN)

Il K-Nearest Neighbors è una generalizzazione del Nearest Neighbor, dove invece di considerare solo il punto più vicino, si considerano i K punti più vicini.

Funzionamento di base:

1. **Memorizzazione dei dati di addestramento:** KNN memorizza tutti i punti dati del set di addestramento.
2. **Scelta di K :** Si sceglie un valore di K , che è il numero di vicini più vicini da considerare.

3. **Classificazione:** Per un nuovo punto dati, si trovano i K punti più vicini nel set di addestramento.

- ▶ **Votazione:** Si assegna al nuovo punto la classe che è più comune tra i K vicini (per classificazione).
- ▶ **Media:** Si calcola la media dei valori dei K vicini e si assegna questo valore al nuovo punto (per regressione).

Distanze

Gli algoritmi NN e KNN utilizzano una misura di distanza per determinare la vicinanza tra i punti dati. Le misure di distanza più comuni sono:

- ▶ **Distanza Euclidea:** La radice quadrata della somma dei quadrati delle differenze tra le coordinate dei punti.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- ▶ **Distanza di Manhattan:** La somma delle differenze assolute tra le coordinate dei punti.

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$$

Vantaggi

- ▶ **Semplicità:** Entrambi gli algoritmi sono facili da comprendere e implementare.
- ▶ **Efficienza su piccoli set di dati:** Funzionano bene su set di dati di piccole dimensioni.

Svantaggi

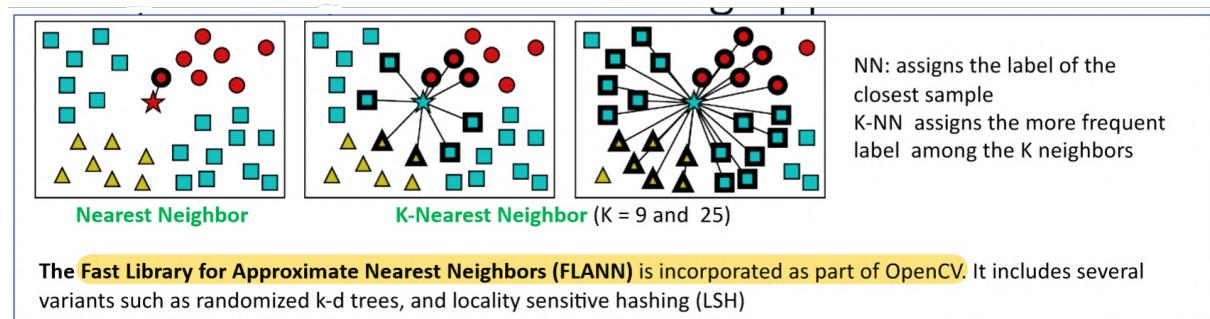
- ▶ **Lentezza su grandi set di dati:** La ricerca dei vicini può essere computazionalmente costosa su set di dati molto grandi.
- ▶ **Sensibilità ai dati rumorosi:** Possono essere influenzati da outlier o dati rumorosi.
- ▶ **Scelta di K :** La prestazione di KNN dipende fortemente dalla scelta del parametro K . Un valore troppo piccolo di K può rendere il modello sensibile al rumore, mentre un valore troppo grande può diluire il modello e considerare punti irrilevanti.

Ottimizzazioni

- ▶ **Riduzione della dimensione:** Tecniche come PCA (Principal Component Analysis) possono essere utilizzate per ridurre la dimensione dei dati e migliorare l'efficienza.
- ▶ **Algoritmi di ricerca avanzata:** Strutture dati come KD-trees o Ball-trees possono essere utilizzate per accelerare la ricerca dei vicini più vicini.

Applicazioni

- **Classificazione di immagini:** Ad esempio, riconoscimento di cifre scritte a mano.
- **Sistemi di raccomandazione:** Ad esempio, suggerire prodotti simili basati sui gusti dell'utente.
- **Riconoscimento vocale:** Classificare frammenti audio in categorie predefinite.



Classificazione Bayesiana

Per alcuni semplici problemi di machine learning, se abbiamo un modello analitico di costruzione e rumore delle caratteristiche, o se possiamo raccogliere abbastanza campioni, possiamo determinare le distribuzioni di probabilità dei vettori di caratteristiche per ciascuna classe $p(x|C_k)$ così come le probabilità a priori delle classi $p(C_k)$. Secondo la regola di Bayes, la probabilità della classe C_k dato un vettore di caratteristiche x è data da

$$p_k = p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{e^{l_k}}{\sum_j e^{l_j}}$$

dove:

- $p(C_k|x)$ è la probabilità a posteriori della classe C_k dato il vettore di caratteristiche x .
- $p(x|C_k)$ è la verosimiglianza, la probabilità del vettore di caratteristiche x dato che appartiene alla classe C_k .
- $p(C_k)$ è la probabilità a priori della classe C_k .
- $p(x)$ è la probabilità marginale del vettore di caratteristiche x , calcolata come:

$$p(x) = \sum_j p(x|C_j)p(C_j)$$

dove la seconda forma (usando le funzioni exp) è conosciuta come esponenziale normalizzato o funzione softmax. La quantità

$$l_k = \log p(x|C_k) + \log p(C_k)$$

è la log-verosimiglianza del campione x appartenente alla classe C_k . È talvolta conveniente denotare la funzione softmax come una funzione da vettore a vettore,

$$p = \text{softmax}(l).$$

La funzione softmax è una funzione che trasforma un vettore di valori in un vettore di probabilità:

$$\text{softmax}(l)_i = \frac{e^{l_i}}{\sum_j e^{l_j}}$$

dove:

- l_i è l'elemento i -esimo del vettore l .
- e^{l_i} è l'esponenziale dell'elemento i -esimo del vettore l .

Nel caso in cui le componenti del vettore delle caratteristiche siano generate indipendentemente, cioè

$$p(\mathbf{x}|C_k) = \prod_i p(x_i|C_k)$$

la classificazione bayesiana diventa una tecnica chiamata *naive Bayes classifier*.

$$\hat{y} = \arg \max_{k \in \{1, \dots, K\}} p(C_k) \prod_{i=1}^n p(x_i|C_k).$$

Regressione Logistica

La regressione logistica applica una proiezione lineare l_i su un vettore di pesi, seguita da una funzione logistica:

$$l_i = w \cdot x_i + b$$

$$p_i = p(C_0|x_i) = \sigma(l_i) = \sigma(w^T x_i + b)$$

dove σ è la funzione logistica sigmoidea. I pesi sono trovati massimizzando le log-verosimiglianze a posteriori delle etichette corrette. Questo si trasforma in minimizzare la funzione di perdita di entropia incrociata:

$$E_{CE}(w, b) = - \sum_i \{t_i \log p_i + (1 - t_i) \log(1 - p_i)\}.$$

classificatore lineare binario

Un classificatore lineare binario cerca un iperpiano (confine decisionale) per separare i campioni delle due classi. Questo confine decisionale può esistere (dati linearmente separabili) o meno. Possono esserci diversi confini decisionali.

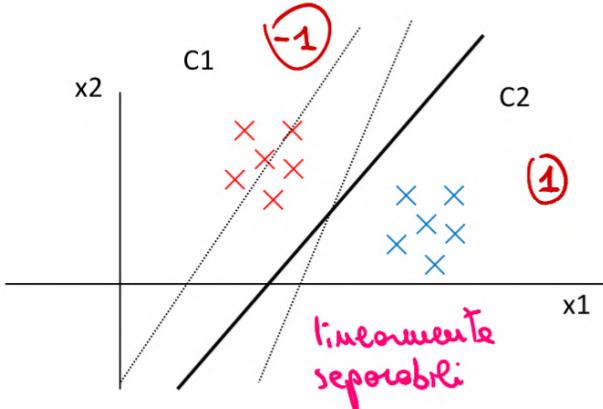
Consideriamo generalmente vettori aumentati, in modo che il confine passi attraverso l'origine ed è rappresentato dal vettore di parametri w .

Per classificare un nuovo campione, osserviamo il lato su cui il campione si trova rispetto al

confine decisionale. Matematicamente, questo viene fatto considerando il segno dell'equazione $\mathbf{w}^T \mathbf{x}$. In pratica:

$$t = \text{sign}(\mathbf{w}^T \mathbf{x})$$

Per problemi di classificazione multi-classe, possiamo usare diversi classificatori lineari (1 vs 1 o 1 vs tutti). Per dati non linearmente separabili, possiamo fare la classificazione proiettando i dati in uno spazio dimensionale superiore.



Support Vector Machines

Le SVM trovano la superficie decisionale lineare (iperpiano) che massimizza il margine con i campioni di addestramento più vicini, che sono chiamati vettori di supporto. Le SVM sono classificatori a margine massimo. Il margine massimo è la più grande distanza perpendicolare tra le due linee. La linea continua, che rappresenta l'iperpiano a metà strada tra gli iperpiani tratteggiati, è il classificatore a margine massimo.

Per massimizzare il margine, troviamo semplicemente il vettore di pesi w di norma minima che soddisfa:

$$\arg \min_{w,b} \|w\|^2 \quad \text{tale che} \quad \hat{t}_i(w \cdot x_i + b) \geq 1$$

dove $\hat{t}_i \in \{-1, 1\}$. Risolvendo quindi un problema di ottimizzazione.

Possiamo anche usare la funzione kernel invece della linearità, che è l'effetto della proiezione in uno spazio dimensionale più alto.

$$l_i = f(x_i; w, b) = \sum_k w_k \phi(\|x_i - x_k\|) + b$$

Alberi Decisionali

Gli alberi decisionali eseguono una sequenza di operazioni più semplici, spesso guardando solo singoli elementi delle caratteristiche prima di decidere quale elemento guardare successivamente. Gli alberi decisionali, come le SVM, sono classificatori discriminativi (o regressori), poiché non formano mai esplicitamente un modello probabilistico (generativo) dei dati che stanno classificando. Una foresta casuale è creata costruendo un insieme di alberi decisionali, ciascuno dei quali prende decisioni leggermente diverse. Al momento del test (classificazione), un nuovo campione è classificato da ciascuno degli alberi nella foresta casuale e le distribuzioni di classe nei nodi finali sono mediate per fornire una risposta più accurata di quanto potrebbe essere ottenuto con un singolo albero (con una data profondità).

5.8 Approcci di Apprendimento Non Supervisionato Popolari

Una delle cose più semplici che puoi fare con i tuoi dati di esempio è raggrupparli in insiemi basati su somiglianze (ad esempio, distanze vettoriali). In statistica, questo problema è noto come analisi cluster. Algoritmi di clustering popolari includono clustering spettrale, Laplaciano del grafo, clustering gerarchico, agglomerativo e divisivo. Un algoritmo famoso (usato per inizializzare la mistura di Gaussiane) è K-means.

5.9 Riconoscimento Visivo

Diversi tipi di problemi:

- **Riconoscimento di istanze:** stiamo cercando di trovare esemplari di un particolare oggetto prodotto
 - Gli approcci più riusciti tendono a utilizzare caratteristiche 2D invariante al punto di vista (angoli o punti di interesse).
 - Le caratteristiche delle immagini dei test vengono confrontate con il database degli oggetti, utilizzando una strategia di corrispondenza delle caratteristiche.
 - Quando viene trovato un numero sufficiente di corrispondenze, queste vengono verificate trovando una trasformazione geometrica che allinea i due set di caratteristiche.
- **Classificazione delle immagini** (o riconoscimento delle categorie di immagini): stiamo riconoscendo la classe a cui appartiene l'immagine.
 - **Approcci bag-of-features**, che rappresentano le immagini come collezioni disordinate di descrittori di caratteristiche e usano le proprietà statistiche di queste collezioni per prendere decisioni sull'etichetta da assegnare.
 - **Modelli basati su parti**, che trovano le parti costituenti di un oggetto e misurano le loro relazioni geometriche. Molto utilizzati per il rilevamento della posa del corpo.
 - **Reti neurali profonde**: i primissimi modelli convoluzionali (AlexNet e VGG) risolvevano questo problema.

- ▶ **Riconoscimento di categoria o classe:** stiamo riconoscendo qualsiasi istanza di una particolare classe altamente variabile, come gatti, cani o motociclette.
- ▶ **Il riconoscimento di classe** è generalmente eseguito insieme al rilevamento degli oggetti, il che significa che non solo riconosciamo la classe ma delineiamo anche (con riquadri di delimitazione) vari oggetti in un'immagine. Il rilevamento degli oggetti include varianti più specializzate come il rilevamento del viso e il rilevamento dei pedoni.

- La maggior parte degli approcci classici utilizzava una *finestra mobile* per estrarre le caratteristiche e un classificatore per assegnare l'etichetta.

Al giorno d'oggi, le tecniche di *deep learning* sono le più riuscite.

Riconosciamo qualsiasi istanza di una classe e rileviamo anche l'istanza (*object detection*).

- ▶ **Segmentazione semantica:** delineiamo vari oggetti e materiali in modo preciso a livello di pixel, cioè per etichettare ogni pixel con un'identità e una classe di oggetto. Le varianti su questo includono la segmentazione delle istanze, dove ogni oggetto separato ottiene un'etichetta unica, la segmentazione panottica, dove sia gli oggetti che i materiali (ad esempio, erba, cielo) vengono etichettati, e la stima della posa, dove i pixel vengono etichettati con le parti del corpo e le orientazioni delle persone.

- Per molti anni, l'approccio preminente è stato l'utilizzo del *Conditional Random Field* (per semplificare, pensalo come una variante del *Markov Random Field*).

Riconoscimento simultaneo della classe e segmentazione (a livello di pixel).

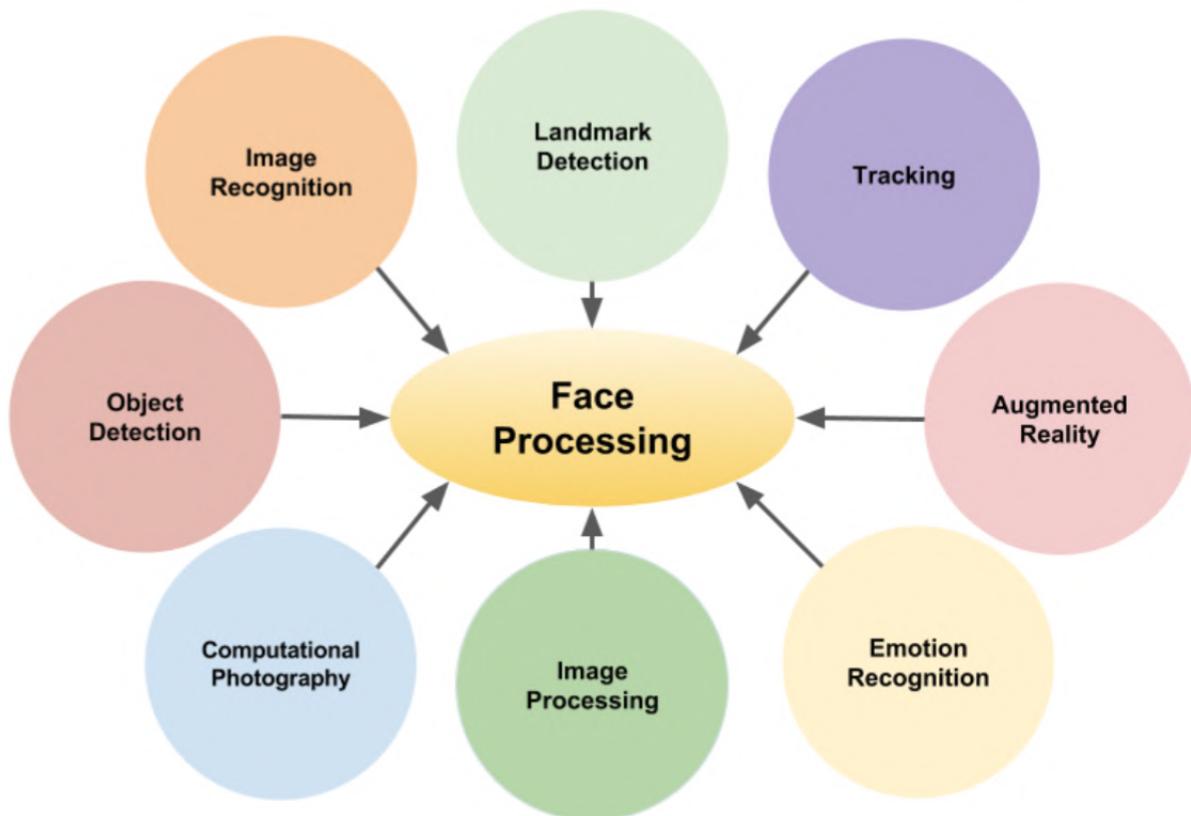
FACES

6.1 Face Processing

Le facce sono ovunque: la maggior parte delle foto e dei video contiene facce.

Il face processing include:

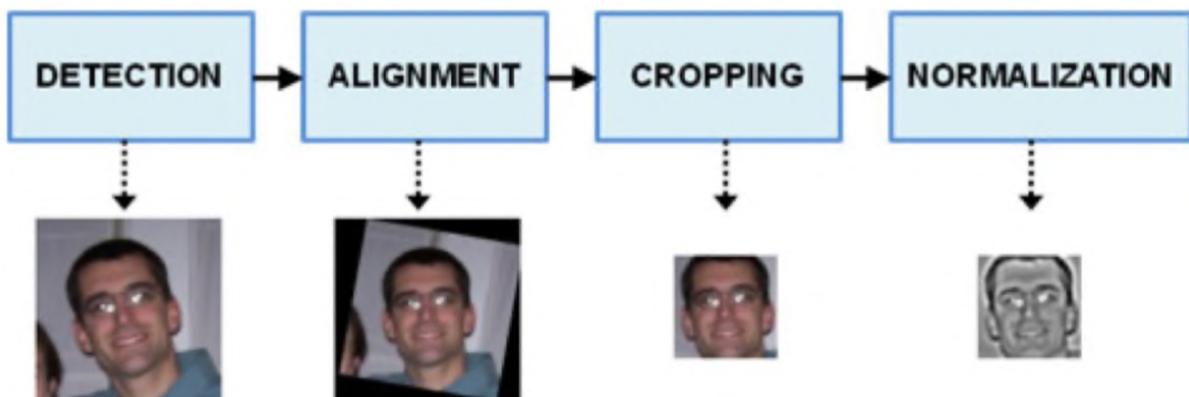
- ▶ rilevamento di oggetti,
- ▶ riconoscimento facciale (riconoscimento dell'identità),
- ▶ riconoscimento delle emozioni,
- ▶ rilevamento dei landmark (punti di riferimento),
- ▶ fotografia computazionale,
- ▶ realtà aumentata e molto altro.



Face Processing

Nei metodi CV classici vengono generalmente utilizzate diverse fasi (non necessariamente tutte).

- ▶ **Rilevamento della Faccia:** la faccia può essere frontale, laterale o in altre pose.
- ▶ **Allineamento:** per facilitare altri passi di elaborazione, generalmente le facce (specialmente le viste frontali) vengono allineate. Esistono metodi di co-allineamento (congealing-funneling) o talvolta vengono utilizzati i landmark facciali (ad esempio, la posizione degli occhi). L'allineamento delle facce stima una trasformazione dell'immagine che riduce la variabilità all'interno del set di immagini facciali.
- ▶ **Ritaglio:** per focalizzarsi sui tratti del volto, si conserva solo la regione in cui è più probabile trovare informazioni sul volto, eliminando aree rumorose come capelli e sfondo.
- ▶ **Allineamento/Normalizzazione dell'Intensità:** per ridurre gli effetti delle diverse condizioni di illuminazione, talvolta vengono utilizzati passaggi di pre-elaborazione come la correzione gamma, l'accentuazione e l'eguaglianza del contrasto.
- ▶ **Descrizione della Faccia:** vengono estratte caratteristiche per rappresentare i tratti del volto, ad esempio, eigenfaces e Local Binary Patterns.

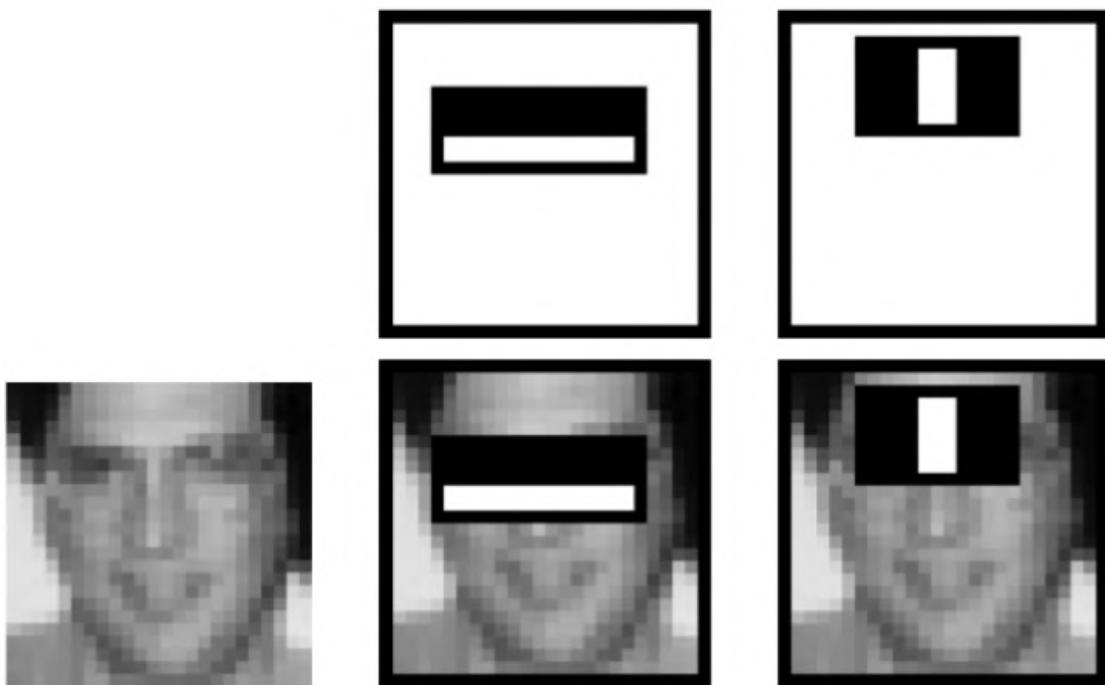


Face Detection – Approccio Classico

- ▶ Il rilevatore di facce introdotto da Viola e Jones (2004) è probabilmente il più noto.
- ▶ È relativamente semplice e molto intuitivo.
- ▶ Utilizza un insieme di semplici caratteristiche facciali chiamate Haar-like features, e una tecnica di apprendimento chiamata boosting.
- ▶ Il boosting comporta l'addestramento di una serie di classificatori semplici sempre più discriminanti e poi la combinazione dei loro output.
- ▶ È interessante notare che studi iniziali sulle CNN hanno mostrato che le caratteristiche apprese automaticamente dalla rete nei primi strati somigliano alle Haar-like features.

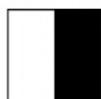
Haar-like Features

- ▶ Ogni caratteristica è un singolo valore ottenuto sottraendo la somma dei pixel sotto il rettangolo bianco dalla somma dei pixel sotto il rettangolo nero.
- ▶ Le caratteristiche vengono estratte in ogni posizione dell'immagine a tutte le dimensioni possibili.
- ▶ Per ogni calcolo delle caratteristiche, è necessario trovare la somma dei pixel sotto i rettangoli bianchi e neri.
- ▶ La complessità cresce con la dimensione dell'immagine.
- ▶ Per velocizzare il calcolo delle caratteristiche, si adotta la tecnica delle immagini integrali (integral images).



120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151

Let us apply an edge feature 4x4 on the image above
Features are computed in a sliding window approach



We will get (without padding) 9 feature values

120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151

At each time, we need to sum pixel values in white areas and subtract the pixel values in black area

116		

120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151

At each time, we need to sum pixel values in white areas and subtract the pixel values in black area

-116	-232	

120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151
120	121	120	150	149	151

At each time, we need to sum pixel values in white areas and subtract the pixel values in black area

-116	-232	-30
-116		

Integral Image (summed area table)

- Se un'immagine viene ripetutamente convoluta con diversi filtri a scatola (soprattutto filtri di dimensioni diverse in posizioni diverse), è possibile pre-calcolare la tabella delle somme (Crow 1984), che è semplicemente la somma corrente di tutti i valori dei pixel dall'origine.
- La somma S dei valori dei pixel in una regione $[i_0, i_1] \times [j_0, j_1]$ è calcolata rapidamente usando l'immagine integrale.

$$s(i, j) = \sum_{k=0}^i \sum_{l=0}^j f(k, l)$$

$$S(i_0 \dots i_1, j_0 \dots j_1) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1)$$

3	2	7	2	3
1	5	1	3	4
5	1	3	5	1
4	3	2	1	6
2	4	1	4	8

(a) $S = 24$

Initial image
The sum of pixel values in the green region is $S=24$

3	5	12	14	17
4	11	19	24	31
9	17	28	38	46
13	24	37	48	62
15	30	44	59	81

(b) $s = 28$

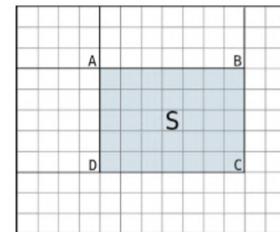
This is the integral image
Computed as cumulative sum along rows and columns
The cell $[i,j]$ is the sum of pixel values in the region $[0,i] \times [0,j]$

$$19 + 17 + 3 - 11 = 28$$

3	5	12	14	17
4	11	19	24	31
9	17	28	38	46
13	24	37	48	62
15	30	44	59	81

(c) $S = 24$

$$S = C - B - D + A$$



6.2 Face Detection - Viola-Jones

- ▶ Le immagini integrali permettono di velocizzare il calcolo delle diverse Haar-like features a differenti scale.
- ▶ L'immagine integrale viene calcolata una sola volta e utilizzata per calcolare tutte le caratteristiche.
- ▶ Supponiamo di avere un'immagine $N \times M$. Vogliamo calcolare 5 Haar-like features di dimensioni 7×7 , 13×13 e 21×21 .
- ▶ Per ciascuna delle 5 caratteristiche dovremo calcolare $(N - 7 + 1) \times (M - 7 + 1) + (N - 13 + 1) \times (M - 13 + 1) + (N - 21 + 1) \times (M - 21 + 1)$ valori.
- ▶ Se $N = M = 100$ abbiamo circa 23000×5 valori (circa 115K. Nel documento originale vengono calcolate 160K caratteristiche per 4 kernel a diverse scale).
- ▶ I valori sono impilati in un vettore, generalmente chiamato feature vector.
- ▶ Ora dobbiamo gestire un vettore di caratteristiche ad alta dimensione per classificare l'immagine come viso/non viso.
- ▶ Molto probabilmente, non tutte le caratteristiche sono importanti e vorremmo concentrarci su alcune caratteristiche significative.
- ▶ Ad esempio, probabilmente le caratteristiche intorno agli occhi o al naso sono utili, ma forse non sempre.

La forza del metodo Viola-Jones sta nell'uso di AdaBoost per costruire un classificatore più forte basato su un insieme di classificatori deboli.

AdaBoost viene utilizzato sia per selezionare le caratteristiche sia per addestrare un classificatore.

AdaBoost

- ▶ Il boosting consiste nel costruire un classificatore $h(x)$ come somma di m deboli apprenditori (weak learners):

$$h(x) = \text{sign} \left(\sum_{j=0}^{m-1} \alpha_j h_j(x) \right)$$

- ▶ $h(x)$ prende in input un vettore di caratteristiche. Restituisce $+1$ o -1 , quindi è un classificatore binario.
- ▶ I classificatori deboli possono essere qualsiasi funzione che restituisce una decisione sul vettore x (ad esempio $+1$ o -1). Di per sé, il classificatore debole può avere un alto tasso di errore. Tuttavia, insieme, diversi classificatori deboli possono contribuire a prendere la decisione corretta.
- ▶ Il boosting consiste nel costruire un classificatore $h(x)$ come somma di m deboli apprenditori (weak learners):

$$h(x) = \text{sign} \left(\sum_{j=0}^{m-1} \alpha_j h_j(x) \right)$$

- Ad esempio, un classificatore debole può essere semplicemente una funzione di soglia

$$h_j(x \mid p, \theta_j) = \begin{cases} -1 & px[j] < p\theta_j \\ 1 & \text{altrimenti} \end{cases}$$

Dove $p = -1$ o $+1$ e θ è un valore di soglia.

- Il classificatore debole (learner) è progettato per selezionare la caratteristica che meglio separa gli esempi positivi (viso) dagli esempi negativi (non viso).
- Per ciascuna caratteristica, il learner debole determina la funzione di classificazione soglia ottimale, in modo che il numero minimo di esempi sia classificato erroneamente. Nessuna singola caratteristica può svolgere il compito di classificazione con basso errore.
- I weak learners vengono trovati iterativamente. A ogni passo, i weak learners cercheranno di classificare correttamente i campioni più difficili. Per fare ciò, i campioni nel set di addestramento sono associati a un peso. Più alto è il peso, maggiore è la necessità di trovare un classificatore debole in grado di gestire il campione corrispondente.

3	2	7	2	3
1	5	1	3	4
5	1	3	5	1
4	3	2	1	6
2	4	1	4	8

(a) $S = 24$

Initial image
The sum of pixel values in the green region is $S=24$

3	5	12	14	17
4	11	19	24	31
9	17	28	38	46
13	24	37	48	62
15	30	44	59	81

(b) $s = 28$

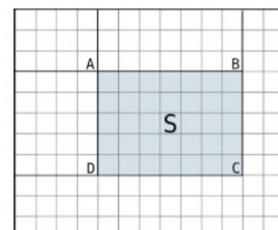
This is the integral image
Computed as cumulative sum along rows and columns
The cell $[i,j]$ is the sum of pixel values in the region $[0,i] \times [0,j]$

$$19 + 17 + 3 - 11 = 28$$

3	5	12	14	17
4	11	19	24	31
9	17	28	38	46
13	24	37	48	62
15	30	44	59	81

(c) $S = 24$

$$S = C - B - D + A$$

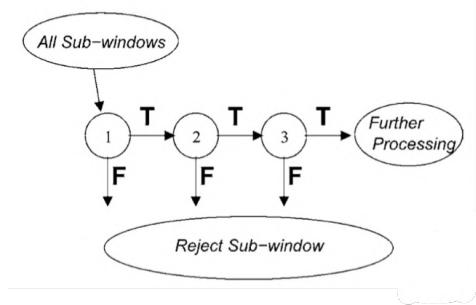


- In fase di test, abbiamo una cascata di classificatori deboli:

- Un risultato positivo dal primo classificatore innesca la valutazione di un secondo classificatore, anch'esso regolato per ottenere tassi di rilevamento molto elevati. Un risultato positivo dal secondo classificatore innesca un terzo classificatore, e così via. Un risultato negativo in qualsiasi punto porta al rifiuto immediato della sottofinestra.
- La struttura della cascata riflette il fatto che all'interno di una singola immagine, la stragrande maggioranza delle sottofinestre è negativa. Di conseguenza, la cascata tenta di rifiutare il maggior numero possibile di negativi al primo stadio possibile.

- Una serie di classificatori viene applicata a ogni sottofinestra.

- ▶ Il classificatore iniziale elimina un gran numero di esempi negativi con un'elaborazione minima.
- ▶ Gli strati successivi eliminano ulteriori negativi, ma richiedono una maggiore elaborazione. Dopo diversi stadi di elaborazione, il numero di sottofinestre viene ridotto radicalmente.
- ▶ La cascata viene ottenuta tramite un approccio euristico che modifica le soglie per massimizzare il tasso di rilevamento, minimizzando al contempo l'errore.



Viola Jones in openCV

```

faces_xml = 'haarcascade_frontalface_alt.xml'
eyes_xml = 'haarcascade_eye_tree_eyeglasses.xml'

face_cascade = cv2.CascadeClassifier()
eyes_cascade = cv2.CascadeClassifier()
# -- 1. Load the cascades
if not face_cascade.load(faces_xml):
    print('(!)Error loading face cascade')
    exit(0)

if not eyes_cascade.load(eyes_xml):
    print('(!)Error loading eyes cascade')
    exit(0)

# -- 2. Read the video stream
camera = cv2.VideoCapture(0)
if not camera.isOpened():
    print('(!)Error opening video capture')
    exit(0)

while True:
    ret, frame = camera.read()
    if frame is None:
        print('(!) No captured frame -- Break!')
        break

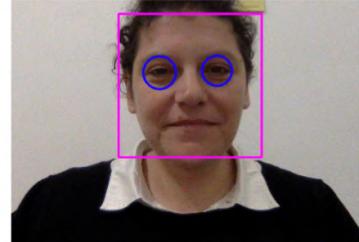
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #frame_gray = cv2.equalizeHist(frame_gray)
    # -- Detect faces
    faces = face_cascade.detectMultiScale(frame_gray)
    for (x, y, w, h) in faces:
        center = (x + w // 2, y + h // 2)

        faceROI = frame_gray[y:y + h, x:x + w]

        # -- In each face, detect eyes
        eyes = eyes_cascade.detectMultiScale(faceROI)
        for (x2, y2, w2, h2) in eyes:
            eye_center = (x + x2 + w2 // 2, y + y2 + h2 // 2)
            radius = int(round((w2 + h2) * 0.25))
            frame = cv2.circle(frame, eye_center, radius, (255, 0, 0), 4)

    frame = cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 255), 4)

    cv2.imshow('Capture - Face detection', frame)
    if cv2.waitKey(10) == 27:
        break
  
```



6.3 Riconoscimento Facciale - Approcci Classici

- Una volta che i volti sono stati rilevati, possiamo estrarre altre caratteristiche per riconoscere l'identità.
- Le caratteristiche per il rilevamento e il riconoscimento sono generalmente diverse! Il riconoscimento richiede dettagli fini, mentre il rilevamento richiede dettagli grossolani.
- I sistemi di riconoscimento facciale sono oggi categorizzati in:
 - riconoscimento facciale a mondo chiuso: i soggetti sono formalmente iscritti nel sistema utilizzando immagini del volto di esempio. I soggetti sono in numero limitato.
 - riconoscimento facciale a mondo aperto: i soggetti sconosciuti sono automaticamente iscritti nel sistema alla prima visualizzazione. Il numero di soggetti è potenzialmente illimitato (si pensi alle persone che si muovono in un aeroporto).
- Un approccio popolare e semplice per il riconoscimento facciale a mondo chiuso è quello delle *eigenfaces*.
- Le *eigenfaces* si basano sull'osservazione fatta per la prima volta da Kirby e Sirovich (1990) che un'immagine facciale arbitraria x può essere compressa e ricostruita partendo da un'immagine media m e aggiungendo un numero ridotto di immagini firmate scalate u_i ,

$$\hat{x} = m + \sum_{i=0}^{m-1} a_i u_i$$

dove le immagini di base firmate u_i possono essere derivate da un insieme di immagini di addestramento utilizzando l'analisi delle componenti principali (PCA) (nota anche come analisi degli autovalori o trasformata di Karhunen–Loëve).

- Turk e Pentland (1991) hanno riconosciuto che i coefficienti a_i nell'espansione delle *eigenfaces* potevano essere usati per costruire un algoritmo di confronto di immagini rapido.
- **Turk e Pentland (1991)** hanno riconosciuto che i **coefficienti** a_i nell'espansione delle eigenface potevano essere utilizzati per costruire un algoritmo di confronto delle immagini rapido.

6.4 EigenFaces - Turk Pentland

- Più in dettaglio, si inizia con una raccolta di immagini facciali di addestramento $\{x_j\}$, da cui si calcola l'immagine media m e una matrice di covarianza:

$$C = \frac{1}{N} \sum_{j=0}^{N-1} (x_j - m)(x_j - m)^T$$

- Possiamo applicare la decomposizione agli autovalori (sì, di nuovo SVD!), che si scrive come:

$$C = U \Lambda U^T = \sum_{i=0}^{N-1} \lambda_i u_i u_i^T$$

- Dove i λ_i sono gli autovalori di C e i u_i sono gli autovettori.

6.5 Face Space

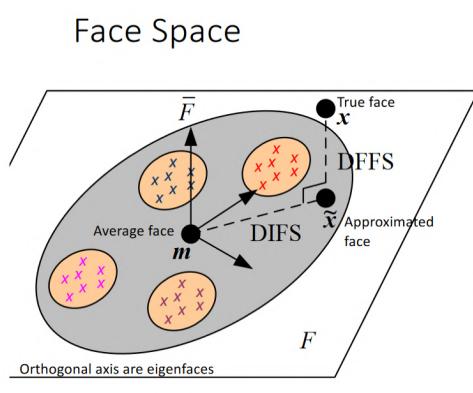
- ▶ Due proprietà importanti della decomposizione agli autovalori sono che i coefficienti a_i ottimali (migliore approssimazione) per qualsiasi nuova immagine x (ad es. immagine di test) possono essere calcolati come:

$$a_i = (x - m) \cdot u_i,$$

- ▶ Supponendo che gli autovalori $\{\lambda_i\}$ siano ordinati in ordine decrescente, troncare l'approssimazione di x in qualsiasi punto M fornisce la migliore approssimazione possibile (minimo errore) tra \hat{x} e x . In altre parole, usiamo solo i primi M vettori base.
- ▶ Troncare la decomposizione delle eigenface di un'immagine facciale dopo M componenti equivale a proiettare l'immagine in uno spazio lineare F , che possiamo chiamare **spazio delle facce (face space)**.
- ▶ Poiché gli autovettori (eigenface) sono ortogonali e di norma unitaria, la distanza di un volto proiettato \hat{x} rispetto all'immagine media m può essere scritta come:

$$DIFS = \|x - m\| = \sqrt{\sum_{i=0}^{M-1} a_i^2}$$

- ▶ È possibile anche misurare la distanza tra due volti diversi nello spazio delle facce prendendo la norma della differenza dei coefficienti dell'eigenface. Una migliore distanza, che tiene conto della covarianza misurata, è la **distanza di Mahalanobis**.



6.6 EigenFaces and Face Recognition

- ▶ Le **EigenFaces** sono state introdotte per descrivere i volti.
- ▶ Il riconoscimento facciale, cioè associare a un nuovo volto un'identità tra quelle conosciute, richiede qualche altra tecnica.
- ▶ Qui, sono stati utilizzati diversi approcci:
 - Classificazione con Nearest Neighbor;

- Classificatore K-NN;
- SVMs;
- Qualsiasi altra tecnica di classificazione.

6.7 Forma del Viso e Orientamento della Testa

- ▶ La posa della testa di una persona dovrebbe essere presa in considerazione durante il riconoscimento.
- ▶ I primi sistemi di riconoscimento facciale identificavano punti caratteristici distintivi sulle immagini facciali e svolgevano il riconoscimento in base alle loro posizioni relative o distanze.
- ▶ I contorni delle caratteristiche facciali possono essere utilizzati per normalizzare (deformare) ogni immagine in una forma canonica. La deformazione può anche essere eseguita rilevando punti di riferimento sul volto (detti anche punti chiave), come occhi e naso.

6.8 Modelli Statistici di Forma

- ▶ Dato un insieme di esempi di una forma, possiamo costruire un modello statistico di forma.
- ▶ Ogni forma nel set di addestramento è rappresentata da un insieme di n punti di riferimento etichettati, che devono essere coerenti da una forma all'altra (cioè, i punti di riferimento sono sempre forniti nello stesso ordine).
- ▶ Dato un insieme di tali esempi etichettati, li allineiamo in un sistema di coordinate comune utilizzando l'Analisi di Procrustes. Questo processo traduce, ruota e scala ogni forma di addestramento in modo da minimizzare la somma delle distanze quadratiche rispetto alla media dell'insieme.
- ▶ Ogni forma può essere rappresentata da un vettore $2n$:

$$\mathbf{x} = (x_1, \dots, x_n, y_1, \dots, y_n).$$

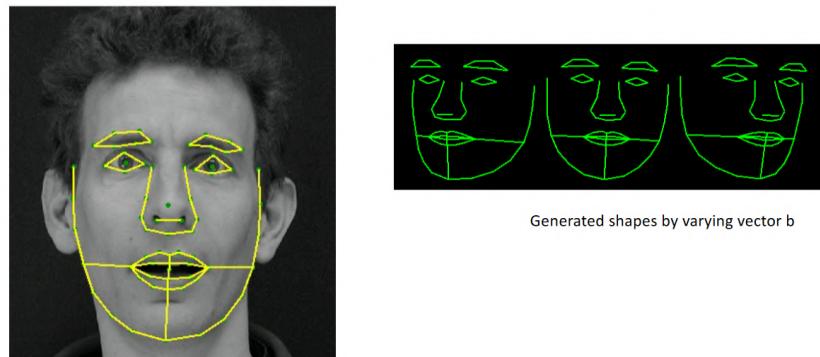
- ▶ Il set di addestramento allineato forma una nube nello spazio dimensionale $2n$ e può essere considerato come un campione da una funzione di densità di probabilità gaussiana.
- ▶ Utilizziamo l'Analisi delle Componenti Principali (PCA) per individuare gli assi principali della nube, e modelliamo solo i primi pochi assi, che rappresentano la maggior parte della variazione.
- ▶ Il modello di forma è quindi:

$$\mathbf{x} = \mathbf{x}_{\text{mean}} + \mathbf{P}\mathbf{b},$$

dove \mathbf{x}_{mean} è la media degli esempi di addestramento allineati, \mathbf{P} è una matrice $2n \times t$ le cui colonne sono vettori unitari lungo gli assi principali della nube, e \mathbf{b} è un vettore di t elementi dei parametri di forma.

- ▶ Variando i parametri di forma entro limiti appresi dal set di addestramento, possiamo generare nuovi esempi.

Statistical Shape Models



6.9 Active Shape Models

- ▶ I Modelli Statistici di Forma vengono utilizzati nei Modelli di Forma Attivi (Active Shape Models, ASM).
- ▶ I Modelli di Forma Attivi sono modelli statistici delle forme di oggetti che si deformano iterativamente per adattarsi a un esempio dell'oggetto in una nuova immagine. La forma può essere deformata modificando il vettore b nel modello statistico di forma (SSM).
- ▶ Le deformazioni della forma si fermano quando i vincoli sull'aspetto dell'immagine intorno a ciascun punto di riferimento sono soddisfatti. Ad esempio, i punti di riferimento si trovano su bordi forti.
- ▶ Si assume di avere una stima iniziale dei parametri di posa e di forma (ad esempio, la forma media). Questa viene aggiornata iterativamente.
- ▶ Gli approcci multi-risoluzione che utilizzano la piramide di immagini gaussiane sono spesso utilizzati per migliorare i risultati e accelerare il processo.

6.10 Modelli di Aspetto Attivi (AAM)

- ▶ I Modelli di Aspetto Attivi (Active Appearance Models, AAM) di Cootes, Edwards e Taylor (2001) estendono gli ASM modellando sia la variazione nella:
 - Forma di un'immagine s , che è normalmente codificata dalla posizione dei punti caratteristici chiave sull'immagine.
 - Variazione nella texture t , che viene normalizzata a una forma canonica prima di essere analizzata (la texture rappresenta qui la pelle e le condizioni di illuminazione).

- ▶ Sia la forma che la texture sono rappresentate come deviazioni da una forma media \bar{s} e una texture media \bar{t} :

$$s = \bar{s} + U_s a$$

$$t = \bar{t} + U_t a$$

- ▶ Dove gli autovettori in U_s e U_t sono stati pre-scalati (whitened) e i vettori a rappresentano una deviazione standard osservata nei dati di addestramento.
- ▶ I parametri di forma sono anche trasformati da una similarità globale per adattarsi alla posizione, dimensione e orientamento di un dato volto.

6.11 Applicazioni dei Modelli di Aspetto Attivi (AAM)

- ▶ Anche se gli AAM sono stati talvolta utilizzati direttamente per il riconoscimento, il loro principale utilizzo nel contesto del riconoscimento è quello di allineare i volti in una posa canonica in modo che possano essere utilizzati metodi più tradizionali di riconoscimento facciale.
- ▶ Il modello addestrato viene utilizzato per rilevare, dai punti di riferimento iniziali (mostrati nella figura), i punti di riferimento sul volto di test.

Riconoscimento delle Espressioni Facciali

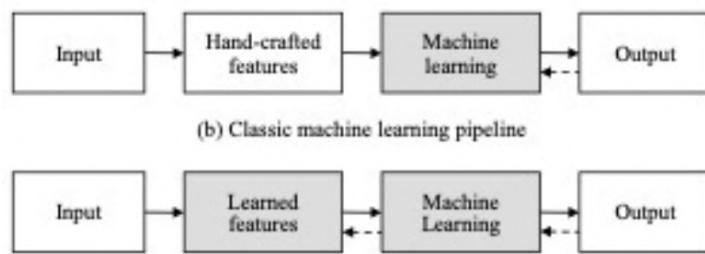
- ▶ I punti di riferimento del volto sono utilizzati anche per riconoscere le espressioni facciali.
- ▶ Questo viene fatto classificando le caratteristiche estratte dai punti di riferimento. In alcune applicazioni, vengono analizzate sequenze di immagini del volto.

TECNICHE DI COMPUTER VISION MODERNE

Tecniche di Computer Vision moderne

7.1 Reti Neurali Profonde (Deep Neural Network)

Le pipeline di deep learning adottano un approccio end-to-end per l'apprendimento automatico, ottimizzando ogni fase del processo stimando i parametri che minimizzano la perdita di addestramento. Di conseguenza, è utile se la perdita è una funzione differenziabile di tutti i parametri. Le reti neurali profonde forniscono un'architettura di calcolo uniforme e differenziabile, scoprendo automaticamente rappresentazioni interne utili.

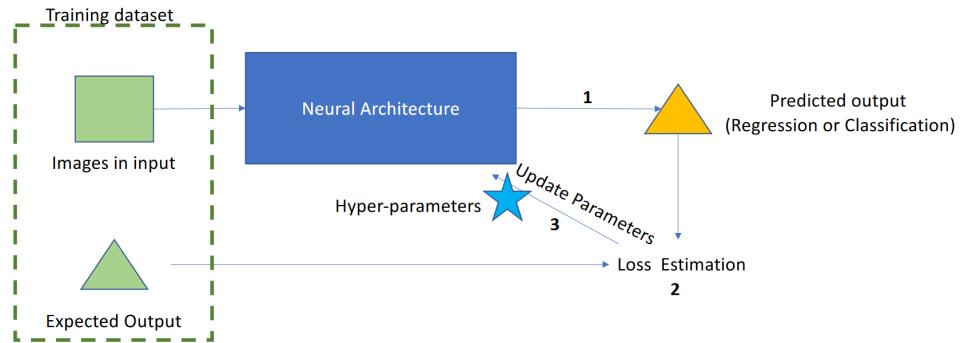


Panoramica Storica delle Reti Neurali

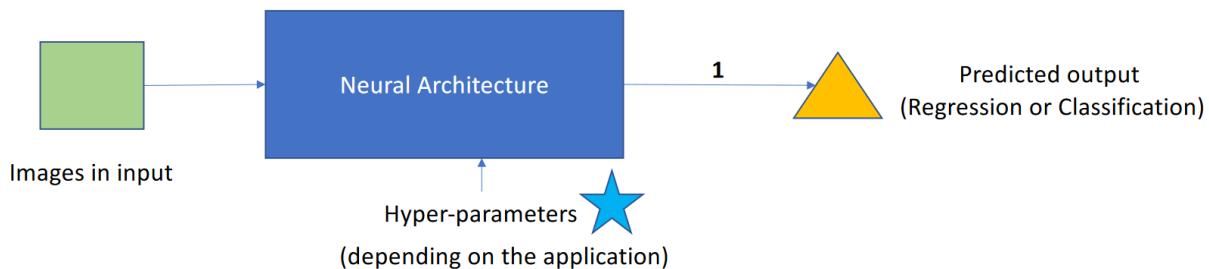
- ▶ 1958: Rosenblatt propone l'algoritmo del percettrone.
- ▶ 1960: Widrow-Hoff propongono la regola delta, modello Adaline.
- ▶ 1969: Minsky critica fortemente il percettrone; si inizia a studiare la capacità dei classificatori.
- ▶ Anni 1970: Era del Connessionismo; le reti neurali artificiali (ANN) sono usate per spiegare fenomeni mentali.
- ▶ 1980: Fukushima propone il Neocognitron, il primo tipo di rete neurale convoluzionale.
- ▶ 1986: Hinton et al. propongono l'algoritmo di retropropagazione.
- ▶ Anni 1990: LeCun propone la Rete Neurale Convoluzionale addestrata mediante retropropagazione (LeNet).
- ▶ 2012: Hinton et al. propongono reti neurali multi-strato profonde per il riconoscimento visivo (AlexNet).
- ▶ 2014: Zisserman et al. propongono VGG.

7.2 Addestramento End-to-End (Apprendimento Supervisionato)

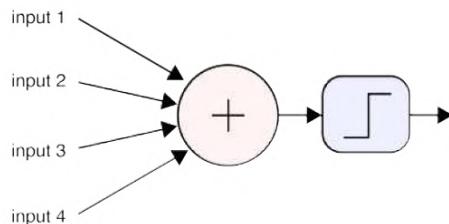
Rispetto ad altre tecniche di apprendimento automatico, che normalmente si basano su diverse fasi di pre-elaborazione per estrarre caratteristiche su cui costruire classificatori, gli approcci di deep learning vengono solitamente addestrati end-to-end, andando direttamente dai pixel grezzi agli output finali desiderati.



Dopo l'apprendimento (Test)



7.3 Neurone



I neuroni eseguono somme pesate dei loro input seguite da una funzione di attivazione non lineare. Matematicamente, questo è rappresentato come:

$$s = \sum_i w_i x_i + b = w^T x + b$$

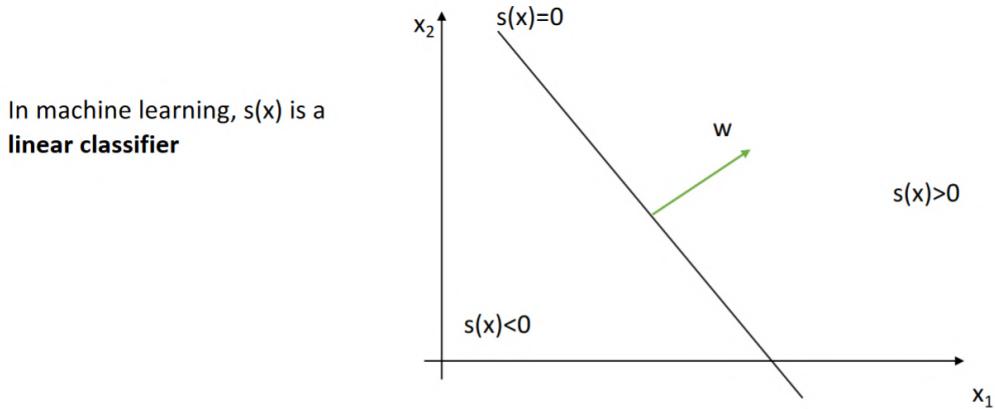
dove w è il vettore dei parametri del neurone e b è il bias. L'output del neurone è quindi:

$$y = h(s)$$

con y output del neurone, calcolato mediante la funzione di attivazione non lineare $h()$, Quando h è una funzione a gradino, il bias b è sostanzialmente una soglia.

Significato Geometrico del Neurone

- ▶ Prima di tutto, concentriamoci sul significato di $s(x) = w^T \cdot x + b$.
- ▶ Da un punto di vista geometrico, $s(x) = 0$ è un iperpiano.
- ▶ In 2D, $s(x)$ divide il piano su cui si trova x in due sottospazi:



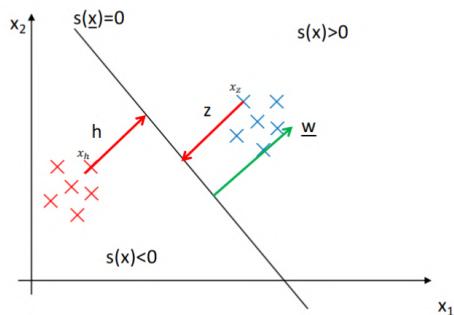
- ▶ Il valore di $s(x)$ è correlato alla **distanza di un punto dall'iperpiano**.

$$h = \frac{|w^T x_h + w_0|}{\|w\|} = \frac{|g(x_h)|}{\|w\|}$$

$$z = \frac{|w^T x_z + w_0|}{\|w\|} = \frac{|g(x_z)|}{\|w\|}$$

$$h = \frac{|w^T x_h + w_0|}{\|w\|} = \frac{|g(x_h)|}{\|w\|}$$

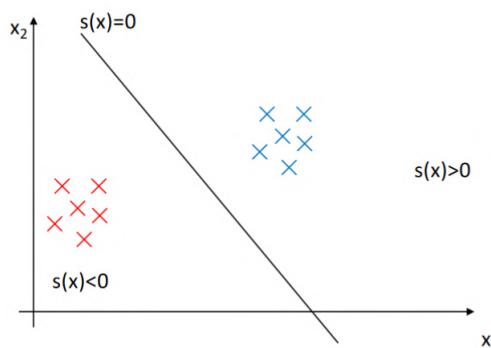
$$z = \frac{|w^T x_z + w_0|}{\|w\|} = \frac{|g(x_z)|}{\|w\|}$$



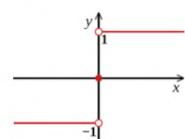
Thus, with a proper choice of w an hyper-plane can be used to implement a (binary) linear classifier

$$y = h(s(x)) = \begin{cases} 1 & s(x) > 0 \\ -1 & \text{else} \end{cases}$$

This is a linear classifier



$h()$ is a non-linear function, in this case a sign function



Per semplificare la formulazione...

Incorporiamo il bias b nel vettore dei parametri:

$$s(x) = w^T x + b$$

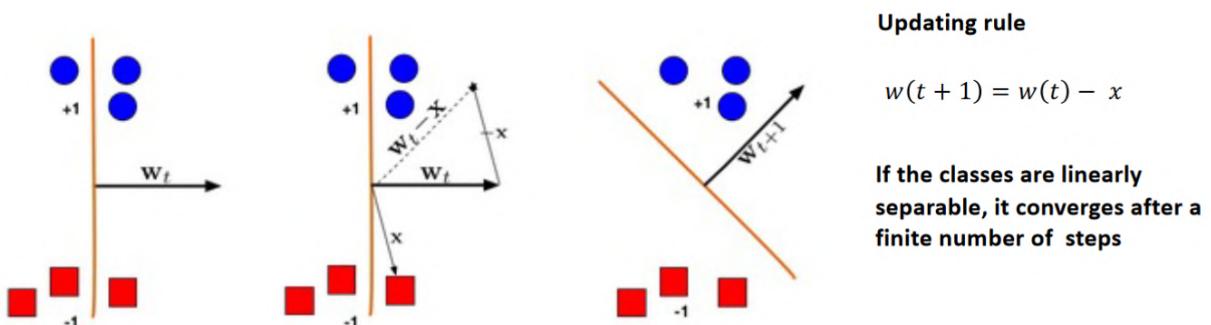
$$w' = \begin{bmatrix} w \\ b \end{bmatrix}, \quad x' = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

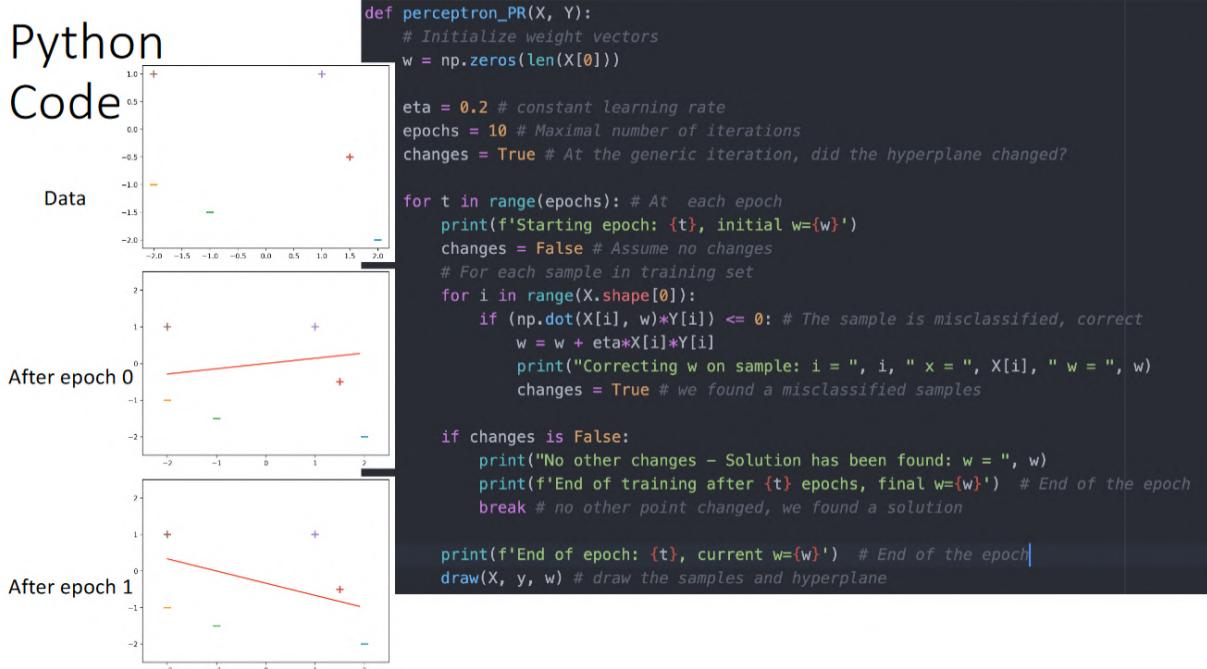
$$s(x) = w'^T x'$$

- x' è spesso chiamato **vettore aumentato**.
- Questo iperpiano passa per l'origine.
- Da ora in poi, ci riferiremo a w' e x' come w e x .

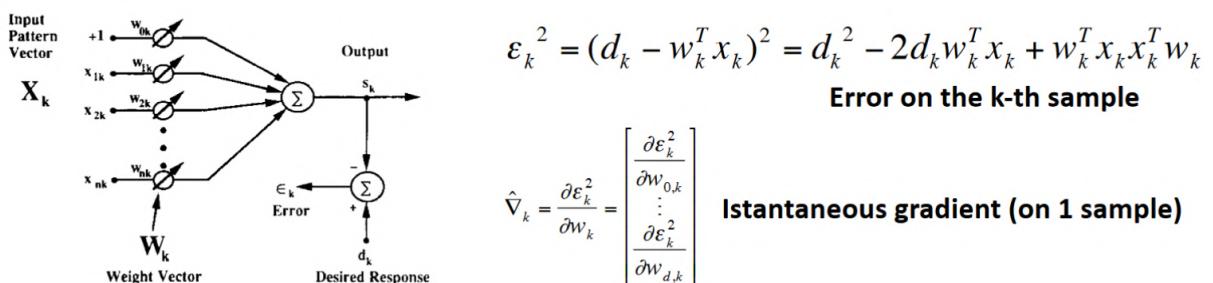
Apprendimento dei parametri del Neurone

- Due algoritmi popolari per apprendere i pesi del neurone sono:
 - L'algoritmo del perceptrone
 - La regola delta
- Nell'algoritmo del perceptrone, la funzione $h()$ è la funzione segno.
- L'algoritmo minimizza il numero di errori commessi dal classificatore ruotando l'iperpiano.





- Due algoritmi popolari per apprendere i pesi del neurone sono:
 - L'algoritmo del percettrone
 - La regola delta
- La regola delta è stata introdotta per addestrare il modello Adeline, che esegue la regressione
(Il problema può anche essere risolto in forma chiusa! La regola delta è la versione iterativa dell'errore quadratico medio. Ricorda cosa abbiamo detto sulla regressione)



Minimizzazione

- Per minimizzare una funzione $L(\mathbf{w})$, dobbiamo applicare un algoritmo, come l'algoritmo di discesa del gradiente.
- Ricorda che, per ottimizzare una funzione, generalmente calcoliamo il gradiente poiché esso fornisce informazioni sulla direzione e il tasso di cambiamento più rapido di una funzione.
- Se il gradiente di una funzione in un punto \mathbf{p} è diverso da zero, la sua direzione è quella in cui la funzione aumenta più rapidamente a partire da \mathbf{p} .
- La magnitudo del gradiente è il tasso di incremento in quella direzione.

- ▶ In pratica, la direzione del gradiente punta verso valori più alti della funzione. Seguendo la direzione del gradiente, potremmo "incontrare" un punto di massimo.
- ▶ Al contrario, seguendo la direzione inversa del gradiente, potremmo incontrare un punto di minimo.

La regola delta (Delta Rule) è stata introdotta per apprendere un regressore iterativamente.

$$\varepsilon_k^2 = (d_k - w_k^T x_k)^2 = d_k^2 - 2d_k w_k^T x_k + w_k^T x_k x_k^T w_k \quad \text{Error on the k-th sample}$$

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial w_k} = \begin{bmatrix} \frac{\partial \varepsilon_k^2}{\partial w_{0,k}} \\ \vdots \\ \frac{\partial \varepsilon_k^2}{\partial w_{d,k}} \end{bmatrix} = -2d_k x_k + 2x_k (x_k^T w_k) = -2(d_k - x_k^T w_k)x_k = -2\varepsilon_k x_k$$

This is a scalar!

$$w_{k+1} = w_k + \mu \cdot (-\hat{\nabla}_k) = w_k + 2\mu \varepsilon_k x_k$$

We update the weights by a fraction of the gradient and, thus by the error multiplied by the input (**Gradient descent**)

7.4 Neurone come Rilevatore di Caratteristiche

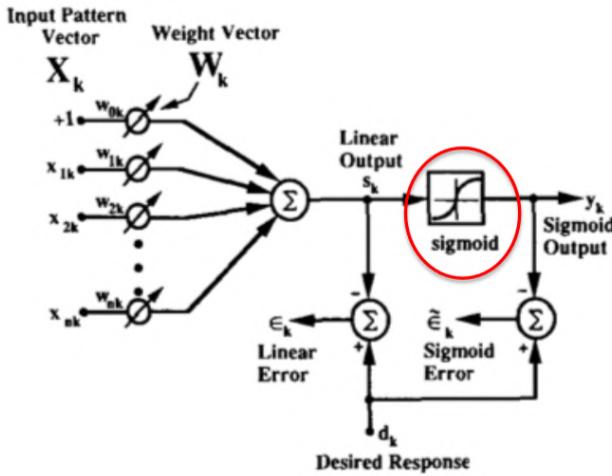
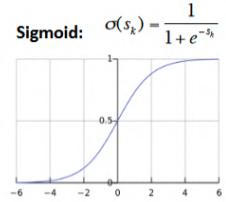
$$s(x) = w^T x$$

Se ci pensiamo, questo può essere interpretato come un filtro 1D (quindi una convoluzione/correlazione 1D). Sotto questo punto di vista, un neurone estrae una caratteristica dai dati in ingresso. La successiva funzione non lineare $h()$, chiamata anche funzione di attivazione, ci dice se la caratteristica estratta è importante o meno.

Ad esempio, un'altra funzione di attivazione è la sigmoide, il cui output può essere interpretato come una probabilità.

Vedremo più avanti altre funzioni di attivazione.

Neuron learning



The **Error** is now expressed as:

$$\epsilon_k^2 = (d_k - \sigma(s_k))^2 \quad s_k = \mathbf{w}_k^T \mathbf{x}_k$$

When activation functions are used,
we need to differentiate a composite
function (chain rule)

Quando si usano funzioni di attivazione, è necessario differenziare una funzione composta
(regola della catena)

$$\epsilon_k^2 = (d_k - \sigma(s_k))^2 \quad s_k = \mathbf{w}_k^T \mathbf{x}_k$$

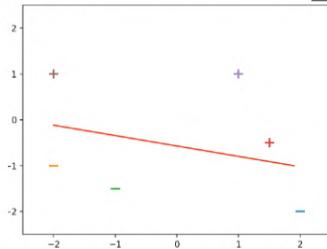
$$\hat{\nabla}_k = \frac{\partial \epsilon_k^2}{\partial \mathbf{w}_k} = \frac{\partial \epsilon_k^2}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial s_k} \cdot \frac{\partial s_k}{\partial \mathbf{w}_k}$$

$$\hat{\nabla}_k = \frac{\partial \epsilon_k^2}{\partial \mathbf{w}_k} = -2\epsilon_k \cdot \frac{\partial \sigma(s_k)}{\partial s_k} \cdot x_k$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

Python Code

```
Press to start
End of epoch: 1, w=[0.044185  0.47733188  0.08247562]
End of epoch: 2, w=[0.18393214  0.62418462  0.21532507]
End of epoch: 3, w=[0.13542149  0.69318748  0.30685166]
End of epoch: 4, w=[0.15296819  0.73091247  0.36359951]
End of epoch: 5, w=[0.16315511  0.7527018  0.39779331]
End of epoch: 6, w=[0.16916287  0.7655307  0.41821251]
End of epoch: 7, w=[0.17272581  0.7731325  0.43837028]
End of epoch: 8, w=[0.17484184  0.77764841  0.43768288]
End of epoch: 9, w=[0.17609871  0.7803319  0.44190239]
End of epoch: 10, w=[0.17684637  0.78192717  0.44445925]
End of epoch: 11, w=[0.17729887  0.78287558  0.44597944]
End of epoch: 12, w=[0.17755515  0.78343945  0.44688326]
End of epoch: 13, w=[0.17771227  0.78377469  0.44742063]
End of epoch: 14, w=[0.17780568  0.78397401  0.44774012]
End of epoch: 15, w=[0.17786122  0.78409251  0.44793006]
End of epoch: 16, w=[0.17789424  0.78416297  0.4480431]
End of epoch: 17, w=[0.17791388  0.78420485  0.44811014]
Solution found in 17 epochs
```

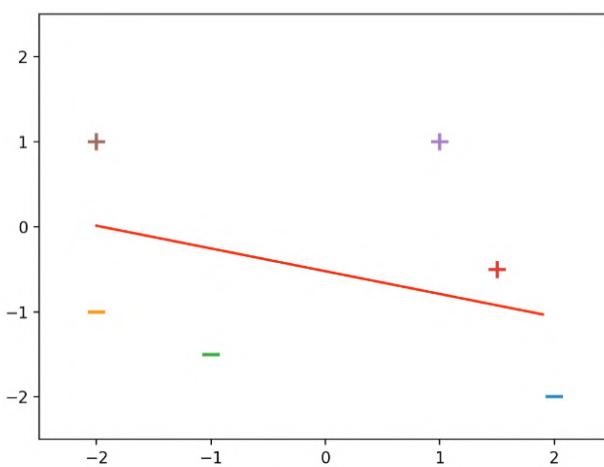


```
def adaline_delta(X, Y):
    # Initialize weight vectors
    w = np.zeros(len(X[0]))

    eta = 0.1 # constant learning rate
    epochs = 100 # Maximal number of iterations
    changes = True # At the generic iteration, did the hyperplane changed?

    t = 0
    w_old = np.ones(len(X[0]))
    tol = 1e-4
    while(t<epochs and np.linalg.norm(w - w_old)>tol):
        w_old = w.copy()
        # For each sample in training set, modify w even if
        # the sample is correctly classified
        for i in range(X.shape[0]):
            error = Y[i] - np.dot(X[i], w)
            w = w + eta*error*X[i]

        t = t + 1
        print(f'End of epoch: {t}, w={w}') # End of the epoch
        draw(X, y, w) # draw the samples and hyperplane
    plt.show()
    if (np.linalg.norm(w - w_old)<=tol):
        print("Solution found in ", t, "epochs")
    else:
        print("Maximal number of epochs reached")
```



The iterative version is useful for datasets that don't fit in memory!

$$J(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{x}_i^T \mathbf{w})^2 \equiv \sum_{i=1}^N e_i^2$$

$$\sum_{i=1}^N \mathbf{x}_i (y_i - \mathbf{x}_i^T \hat{\mathbf{w}}) = 0 \quad \text{gradient}$$

$$\left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right) \hat{\mathbf{w}} = \sum_{i=1}^N (\mathbf{x}_i y_i)$$

$$(\mathbf{X}^T \mathbf{X}) \hat{\mathbf{w}} = \mathbf{X}^T \mathbf{y}$$

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

```
xTx = np.matmul(X.T, X)
xTy = np.matmul(X.T, y)
w = np.matmul(np.linalg.inv(xTx), xTy)
```

7.5 Perceptron e Regola Delta

- ▶ Abbiamo chiaramente usato l'algoritmo di discesa del gradiente per addestrare il neurone con la regola delta.
- ▶ Abbiamo utilizzato il framework punizione-ricompensa per addestrare il perceptron.
- ▶ Si può dimostrare che anche il perceptron utilizza una tecnica di discesa del gradiente, e infatti gli aggiornamenti dei pesi corrispondono al gradiente della seguente funzione costo:

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in Y} (\delta_x, \mathbf{w}^T \mathbf{x})$$

dove Y è l'insieme dei punti classificati erroneamente.

- ▶ $\mathbf{w}^T \mathbf{x}$ porta informazioni sulla distanza del punto \mathbf{x} dall'iperpiano (dovrebbe essere positiva o negativa a seconda della classe predetta).
- ▶ $\delta_x = \begin{cases} -1 & \text{se } \mathbf{x} \in Y \text{ e } \mathbf{x} \in \omega_1 \\ +1 & \text{se } \mathbf{x} \in Y \text{ e } \mathbf{x} \in \omega_2 \end{cases}$
- ▶ Stiamo minimizzando le distanze dall'iperpiano dei punti classificati erroneamente:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \sum_{\mathbf{x} \in Y} \delta_x \mathbf{x}$$

Regola del Perceptron

- ▶ Trova sempre una soluzione se le classi sono separabili, ma non converge se non lo sono.

Minimizzazione dell'Errore – Regola Delta

- ▶ La regola delta consiste sostanzialmente nel trovare una minimizzazione dell'errore quadratico medio (LMS).
- ▶ La soluzione LMS garantisce la convergenza, ma potrebbe non trovare un iperpiano separatore se le classi sono separabili.
- ▶ Minimizza la somma delle distanze dall'iperpiano separatore.

7.6 Capacità di un Neurone

- ▶ Minsky ha evidenziato che con un solo neurone non è possibile classificare qualsiasi insieme di punti.
- ▶ Come esempio, si consideri la funzione XOR, che non è linearmente separabile.
- ▶ La capacità di un classificatore si riferisce al numero massimo di insiemi di punti che il classificatore è in grado di separare perfettamente.

Consideriamo il caso in cui \mathbf{x} ha dimensionalità 2 e ogni coordinata può assumere solo valori +1 e -1. Ci sono quindi solo 2^2 punti possibili (mostrati in figura). Ciascuno di questi punti può appartenere alla classe positiva o negativa. Pertanto, ci sono 2^4 partizioni possibili dei punti dati. In questo semplice problema, 2 delle 16 partizioni non sono linearmente separabili. Una partizione è mostrata in figura, l'altra è quella speculare.

Aumentare la Capacità del Classificatore

Prima Strategia

- ▶ Rimappare i punti in uno spazio a dimensionalità superiore dove i dati sono linearmente separabili.

Ad esempio: trasformare il punto (x_1, x_2) in $(x_1^2, x_1, x_1x_2, x_2, x_2^2)$. Il neurone rimane un iperpiano nello spazio 5D. Quando rimappato di nuovo in 2D, diventa una funzione ellittica. Funziona perfettamente con l'XOR, ma come posso trovare la trasformazione dei dati corretta?

7.7 Rete Multi-Neurone

Seconda Strategia

- ▶ Utilizzare più di un neurone.

Un "timido" tentativo: Madaline

Come addestrare la rete?

Sono state proposte diverse euristiche, come selezionare casualmente un neurone alla volta o fissarne uno alla volta. Con questi metodi, nei problemi più grandi, i neuroni non imparano a collaborare!

7.8 Strati

- ▶ Invece di essere connessi in un grafo di calcolo irregolare, le reti neurali sono solitamente organizzate in strati consecutivi.

Possiamo ora pensare a tutte le unità all'interno di uno strato come a un vettore, con le corrispondenti combinazioni lineari scritte come:

$$\mathbf{s}_l = \mathbf{W}_l \mathbf{x}_l,$$

dove \mathbf{x}_l sono gli input dello strato l , \mathbf{W}_l è una matrice di pesi, e \mathbf{s}_l è un vettore che memorizza la somma pesata di ciascun elemento.

Di conseguenza, uno strato esegue una trasformazione lineare dei dati di input!

I neuroni nello stesso strato elaborano lo stesso input \mathbf{x}_l e lo strato complessivamente restituisce un vettore \mathbf{s}_l :

$$\mathbf{s}_l = \mathbf{W}_l \mathbf{x}_l$$

Al vettore \mathbf{s}_l applichiamo una non-linearietà elemento per elemento tramite un insieme di funzioni di attivazione:

$$\mathbf{y}_l = h(\mathbf{s}_l)$$

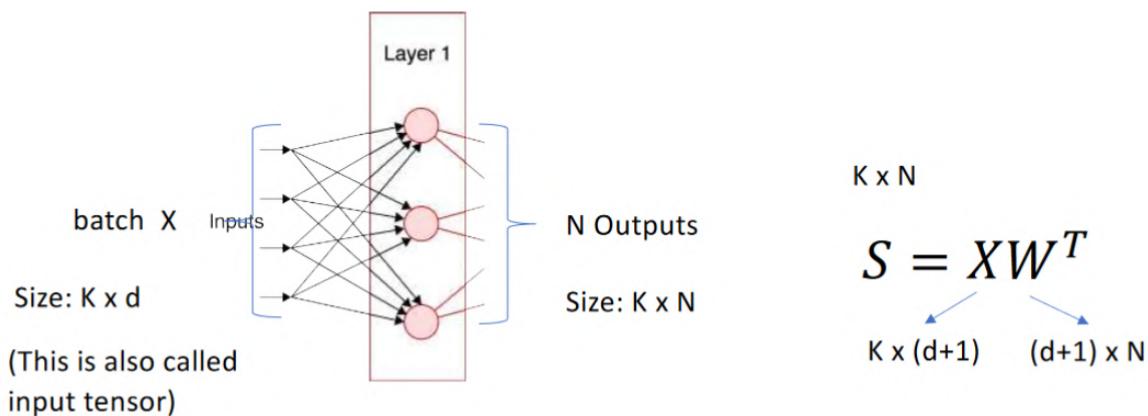
Uno strato in cui viene utilizzata una matrice di pesi completa (densa) per la combinazione lineare è chiamato strato completamente connesso (FC), poiché ogni uscita dello strato dipende da tutti i suoi input. Questi strati sono anche chiamati strati densi.

Esempio

- ▶ Supponiamo di avere uno strato denso con N neuroni.
- ▶ Lo strato prende in input vettori in \mathbb{R}^d .
- ▶ Quanti parametri avrà lo strato?
- ▶ Qual è la dimensione del vettore di output?

7.9 Lavorare con vettori di input multipli - il batch

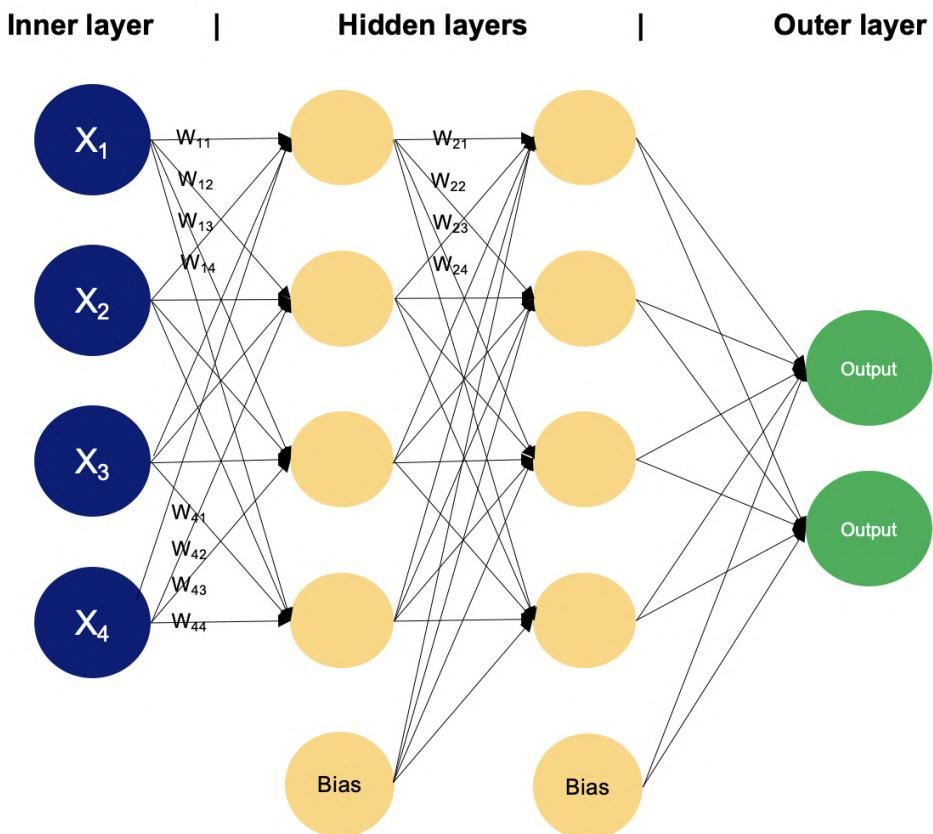
- ▶ Potremmo voler processare più vettori contemporaneamente:
- ▶ Quando usiamo la GPU, caricare e scartare un singolo campione nella/dalla memoria della GPU richiede tempo. Preferiamo spostare batch di dati (vettori impilati).
- ▶ Durante l'addestramento, invece di utilizzare il gradiente istantaneo (su un solo campione) per aggiornare i parametri (una tecnica chiamata discesa del gradiente stocastico), usiamo la media del gradiente di più campioni (quelli nel batch, discesa del gradiente su mini-batch).



Singolo Strato vs Multi-Strato vs Rete Neurale Profonda

- ▶ Le reti neurali sono solitamente organizzate in strati consecutivi.
- ▶ Usare un singolo strato non consente la collaborazione dei neuroni, che verrebbero addestrati in modo indipendente senza sapere cosa stanno imparando gli altri neuroni riguardo al problema. Inoltre, un neurone ha una capacità molto bassa (ricorda Minsky) e non è sufficiente per catturare la complessa struttura dei dati. Una rete neurale a singolo strato può essere utilizzata solo per rappresentare funzioni linearmente separabili.
- ▶ Pertanto, gli strati densi sono utilizzati in cascata (strati densi impilati). Una rete che consiste solo di strati completamente connessi (e non convoluzionali) è ora spesso chiamata perceptron multistrato (MLP). Il Perceptron Multistrato può essere utilizzato per rappresentare regioni convesse.
- ▶ Il numero di strati in cascata in una rete neurale è chiamato profondità. Quando la profondità è maggiore di 3 o 4, parliamo generalmente di rete neurale profonda. Oggi, la maggior parte delle reti neurali profonde nella Computer Vision (CV) ha più di 50 strati.

7.10 Perceptron Multistrato - Rete Feed Forward



Questo è il modo in cui un MLP è generalmente rappresentato. Supponiamo di utilizzare solo sigmoid come funzioni di attivazione.

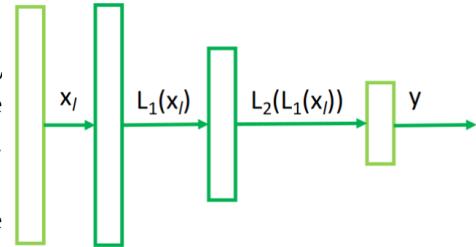
Supponiamo di ragionare su un singolo input (che può essere generalizzato a un batch).

Il campione x_l viene fornito in input.

Primo strato è caratterizzato dal parametro \mathbf{W}_1 , prende in input x_l , calcola l'output di ciascun neurone e applica una funzione sigmoide. Così, produce il vettore \mathbf{L}_1 .

Secondo strato è caratterizzato dal parametro \mathbf{W}_2 , prende in input \mathbf{L}_1 , calcola l'output di ciascun neurone e applica una funzione sigmoide. Così, produce il vettore \mathbf{L}_2 .

Strato di output è caratterizzato dal parametro \mathbf{W}_3 , prende in input \mathbf{L}_2 , calcola l'output di ciascun neurone e applica una funzione sigmoide. Così, produce l'output y .

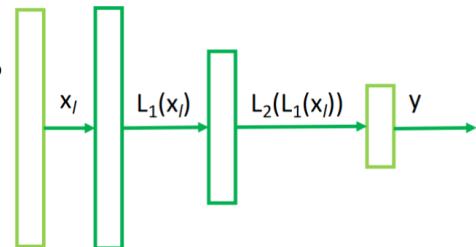


Primo Strato Nascosto L'output del primo strato nascosto è:

$$L_1(x_l) = \mathbf{W}_1 x_l$$

Secondo Strato Nascosto L'output del secondo strato nascosto è:

$$L_2(L_1(x_l)) = \mathbf{W}_2 L_1(x_l) = \mathbf{W}_2 \mathbf{W}_1 x_l$$



Output Finale L'output y è:

$$y = \mathbf{W}_3 L_2(L_1(x_l)) = \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 x_l = \mathbf{W} x_l$$

Questo è lineare. L'MLP collassa in un unico strato rete.

I parametri del primo livello nascosto sono:

$$W_1 \rightarrow N_1 \times (d + 1) \quad \text{dove } N_1 \text{ è il numero di neuroni nel primo livello}$$

I parametri del secondo livello nascosto sono:

$$W_2 \rightarrow N_2 \times (N_1 + 1) \quad \text{dove } N_2 \text{ è il numero di neuroni nel secondo livello}$$

I parametri del livello di output sono:

$$W_3 \rightarrow N_3 \times (N_2 + 1) \quad \text{dove } N_3 \text{ è il numero di output}$$

Il numero totale di parametri del modello è:

$$\text{Numero di parametri del modello} = N_1 \times (d + 1) + N_2 \times (N_1 + 1) + N_3 \times (N_2 + 1)$$

7.11 Propagazione in Avanti (Forward Propagation)

- ▶ Una rete neurale è una funzione altamente non lineare e composta.
- ▶ Per calcolare l'output del modello, è necessario propagare il vettore di input attraverso la rete, da uno strato all'altro.
- ▶ In ogni strato, trasformiamo linearmente i dati (usando i parametri W) e poi applichiamo una funzione di attivazione non lineare.
- ▶ Il processo di far fluire i dati nella rete si chiama propagazione in avanti (Forward Propagation).
- ▶ La propagazione in avanti viene utilizzata in due casi:
 - Durante il test, per ottenere l'output del modello addestrato (usando i parametri ottimali trovati).
 - Durante l'addestramento, per ottenere l'output del modello e misurare l'errore commesso con la stima corrente dei parametri.

Propagazione in Avanti – Esempio Numerico

Supponiamo quanto segue:

$$\mathbf{X} \text{ è un punto 3D: } \mathbf{x} = \begin{bmatrix} 1 \\ 0.2 \\ -0.5 \end{bmatrix}$$

- ▶ $N_1 = 3$
- ▶ $N_2 = 2$
- ▶ $N_3 = 1$
- ▶ Usiamo le sigmoidi e i pesi sono già stati appresi.

$$L_1(\mathbf{x}_l) = \sigma(W_1 \mathbf{x}_l) = \sigma \left(\begin{bmatrix} 0.2 & -0.1 & 0.4 & 0.3 \\ 0.1 & 0.2 & -0.3 & 0.1 \\ 1.2 & -2 & 0.03 & -0.1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.2 \\ -0.5 \\ 1 \end{bmatrix} \right) = \sigma \left(\begin{bmatrix} 0.28 \\ 0.39 \\ 0.885 \end{bmatrix} \right) = \begin{bmatrix} 0.5695 \\ 0.5963 \\ 0.7079 \end{bmatrix}$$

$$L_2(L_1(\mathbf{x}_l)) = \sigma(W_2 L_1(\mathbf{x}_l)) = \sigma \left(\begin{bmatrix} 1 & -2 & 3.2 & 0.8 \\ 0.3 & 2 & 4.2 & -1 \end{bmatrix} \begin{bmatrix} 0.5695 \\ 0.5963 \\ 0.7079 \\ 1 \end{bmatrix} \right) = \sigma \left(\begin{bmatrix} 2.44 \\ 3.34 \end{bmatrix} \right) = \begin{bmatrix} 0.92 \\ 0.97 \end{bmatrix}$$

$$y = \sigma(W_3 L_2(L_1(\mathbf{x}_l))) = \sigma \left(\begin{bmatrix} 0.3 & 1 & -3 \end{bmatrix} \begin{bmatrix} 0.92 \\ 0.97 \\ 1 \end{bmatrix} \right) = \sigma(-1.7583) = 0.147$$

7.12 Funzioni di Attivazione

- Le prime reti neurali utilizzavano principalmente funzioni sigmoidi.
- Le reti neurali più recenti (dal 2010 in poi) utilizzano le Unità Lineari Rettificate (ReLU) o varianti di queste.
- L'uso della funzione di attivazione errata può causare problemi nella fase di addestramento, specialmente nel calcolo del gradiente.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

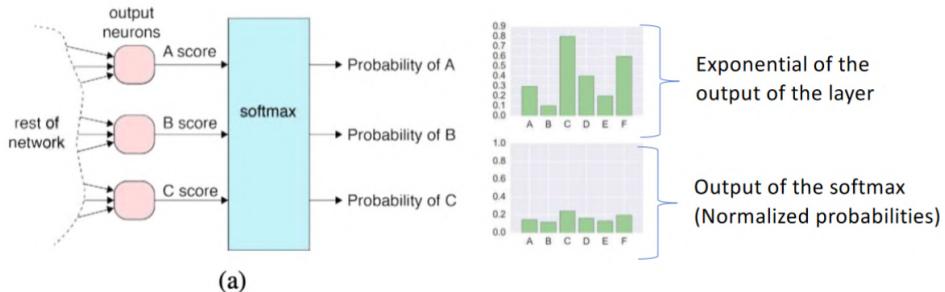
L'ultimo strato determina l'output del modello. Vogliamo scegliere una funzione che ci permetta di ottenere l'output desiderato. Nella classificazione, ci aspettiamo che l'output sia un valore di probabilità o una distribuzione di probabilità.

Problem Type	Output Type	Final Activation Function
Regression	Numerical value	Linear
Classification	Binary outcome	Sigmoid
Classification	Single label, multiple classes	Softmax
Classification	Multiple labels, multiple classes	Sigmoid

7.13 Funzione Softmax

- ▶ L'output l_k -esimo dell'ultimo strato viene trasformato per fornire un valore di probabilità valido.

$$p_k = p(C_k | \mathbf{x}) = \frac{\exp l_k}{\sum_j \exp l_j},$$



Attenzione!

- ▶ È comune interpretare gli output finali di una rete come una distribuzione di probabilità.
- ▶ Se una rete è ben addestrata e predice risposte con buona accuratezza, è facile pensare che queste probabilità siano una misura della fiducia in una particolare risposta. Tuttavia, generalmente non lo sono!
- ▶ Come vedremo, la minimizzazione della perdita durante l'addestramento incoraggia la rete a massimizzare le risposte corrette ponderate per probabilità, e non incoraggia, di fatto, gli output della rete ad essere correttamente calibrati in termini di confidenza.
- ▶ In generale, sono necessarie tecniche di calibrazione per allineare le probabilità del classificatore con la vera affidabilità (Guo - 2017, Platt 2000, Bell - 2020).

7.14 Libreria TensorFlow

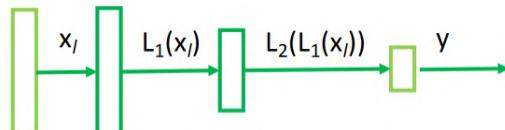
- ▶ TensorFlow è una piattaforma open source completa per il machine learning.
- ▶ TensorFlow rende facile per principianti ed esperti creare modelli di machine learning.
- ▶ TensorFlow include Keras (da molti anni).
- ▶ Keras è un'API progettata per ridurre il carico cognitivo: offre API consistenti e semplici, minimizza il numero di azioni richieste all'utente per casi d'uso comuni, e fornisce messaggi di errore chiari e utili.

Classi Principali di TensorFlow-Keras

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

- ▶ `Sequential` è la classe utilizzata per creare una rete come sequenza di strati sovrapposti. Infatti, si può pensare a un modello `Sequential` come a una pila (*stack*) a cui aggiungere o da cui rimuovere strati.
- ▶ È utilizzata quando il modello riceve in input un solo ingresso e produce un solo output.
- ▶ `Dense` è la classe utilizzata per implementare uno strato denso con un numero specifico di neuroni.
- ▶ La funzione di attivazione può essere integrata nello strato oppure è possibile includere strati specifici che implementano la funzione di attivazione.

Example of FFN



x is a 3D point, $N_1 = 3$, $N_2 = 2$, $N_3 = 1$, we use only sigmoids

```

1 import tensorflow as tf
2 from tensorflow.keras import Sequential
3 from tensorflow.keras.layers import Input, Dense
4
5 model = Sequential()
6 model.add(Input(shape=(3,)))
7 model.add(Dense(3, activation="sigmoid"))
8 model.add(Dense(2, activation="sigmoid"))
9 model.add(Dense(1, activation="sigmoid"))
10
11 model.summary()
  
```

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	12
dense_1 (Dense)	(None, 2)	8
dense_2 (Dense)	(None, 1)	3

Total params: 23
Trainable params: 23
Non-trainable params: 0

7.15 Functional API

- ▶ La Functional API di Keras è un modo per creare modelli più flessibili rispetto all'API `tf.keras.Sequential`.
- ▶ La Functional API può gestire modelli con topologia non lineare (ad esempio, con più rami nella rete), strati condivisi, e persino con input o output multipli.
- ▶ L'idea principale è che un modello di deep learning è solitamente un grafo aciclico diretto (DAG) di strati. Quindi, la Functional API è un modo per costruire grafi di strati.
- ▶ Per costruire un modello utilizzando la Functional API, si inizia creando un nodo di input.

- Un nuovo nodo nel grafo degli strati sarà creato facendo una "chiamata di strato" su questo input. Consiste nel disegnare una freccia da "inputs" a questo strato che hai creato. Stai "passando" gli input allo strato e ottieni x come output.

7.16 Verso l'apprendimento...

- Ora siamo in grado di impostare una rete (decidendo il numero di strati, le loro larghezze e profondità).
- Sappiamo anche che, nel test, la rete viene utilizzata per prevedere l'output.
- Durante questo passaggio, gli input vengono propagati in avanti.
- Per la fase di apprendimento, i parametri sono inizializzati casualmente. Poi, utilizziamo la discesa del gradiente o una delle sue varianti per modificare iterativamente i pesi fino a quando la rete non converge a un buon set di valori, cioè un livello accettabile di prestazioni sui dati di addestramento e di validazione.

7.17 Funzione di Perdita

- La funzione di perdita misura la bontà del modello stimato sul set di addestramento.
- Prende in input l'output predetto e quello vero (etichette/probabilità o valori reali a seconda del tipo di problema) e li confronta.
- Selezioniamo la funzione di perdita in base al problema da risolvere e alla funzione di attivazione dell'ultimo strato del modello.

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

7.18 Binary Cross-Entropy

- ▶ Per la classificazione binaria, la maggior parte delle reti neurali utilizza una funzione sigmoid.
- ▶ Poiché gli output dovrebbero rappresentare probabilità di classe, sono positivi e inferiori a 1. In questo caso, utilizziamo la perdita binary cross-entropy:

$$E_{CE}(w, b) = - \sum_i t_i \log p_i + (1 - t_i) \log(1 - p_i)$$

- ▶ Dove:

- t_i appartiene a $\{0, 1\}$ ed è l'etichetta di classe vera per ogni campione di addestramento x_i (la classe annotata).
- $p_i = p(C_1|x)$ è la probabilità, stimata dal modello, che il campione x_i abbia etichetta 1 (in altre parole, se questa probabilità è alta, il modello sta prevedendo l'etichetta 1).

7.19 Cross-Entropy

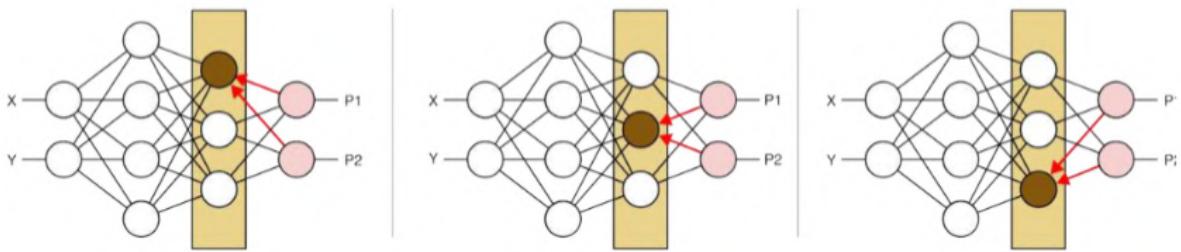
- ▶ Per la classificazione multi-classe con singola etichetta, la maggior parte delle reti neurali utilizza una funzione softmax.
- ▶ Poiché gli output dovrebbero rappresentare probabilità di classe che sommano a 1, è naturale utilizzare la perdita cross-entropy multi-classe.

- ▶ Dove:

- Utilizziamo la codifica one-hot e $t_{ik} = 1$ se il campione i appartiene alla classe k (e 0 altrimenti).
- $p_{ik} = p(C_k|x)$ è la probabilità, stimata dal modello, che il campione x_i abbia etichetta k .

7.20 Backpropagation (Hinton – 1986)

- ▶ Per addestrare il modello, calcoliamo le derivate (gradienti) della funzione di perdita per un campione di addestramento n rispetto ai pesi w utilizzando la regola della catena, partendo dagli output e risalendo attraverso la rete fino agli input.
- ▶ Nel passaggio in avanti (valutazione) di una rete neurale, le attivazioni (output degli strati) vengono calcolate strato per strato, iniziando con il primo strato e terminando con l'ultimo.
- ▶ Durante il backpropagation, eseguiamo una traversata simile del grafo inverso. Tuttavia, invece di calcolare le attivazioni, calcoliamo le derivate della perdita rispetto ai pesi e agli input, che chiamiamo errori.
- ▶ Propagando all'indietro le derivate (errori) attraverso uno strato intermedio della rete profonda. Le derivate della funzione di perdita applicate a un singolo esempio di addestramento rispetto a ciascuno degli input delle unità rosa vengono sommate insieme e il processo viene ripetuto concatenandosi all'indietro attraverso la rete.



- ▶ Supponiamo di utilizzare una perdita L2 (errore quadratico) tra l'output e l'output desiderato sul campione n .

$$\frac{\partial E_n}{\partial y_{nk}} = y_{nk} - t_{nk}$$

- ▶ Per calcolare le derivate parziali del termine di perdita rispetto ai pesi e alle attivazioni precedenti, lavoriamo all'indietro attraverso la rete.

Per calcolare la derivata della perdita E_n rispetto ai pesi, bias e attivazioni degli input, utilizziamo la regola della catena:

$$\frac{\partial E_n}{\partial y_{nk}} = y_{nk} - t_{nk} \quad \text{con} \quad y_i = h(s_i) \quad \text{ed} \quad s_i = \mathbf{w}_i^T \mathbf{x}_i + b_i = \sum_j w_{ij} x_{ij} + b_i$$

- ▶ Con riferimento all'output s_i del neurone i -esimo:

$$e_i = \frac{\partial E_n}{\partial s_i} = h'(s_i) \frac{\partial E_n}{\partial y_i}$$

- ▶ Con riferimento al peso j -esimo del neurone i -esimo:

$$\frac{\partial E_n}{\partial w_{ij}} = x_{ij} \frac{\partial E_n}{\partial s_i} = x_{ij} e_i$$

- ▶ Con riferimento al bias del neurone i -esimo:

$$\frac{\partial E_n}{\partial b_i} = \frac{\partial E_n}{\partial s_i} = e_i$$

- ▶ Con riferimento all'input j -esimo del neurone i -esimo:

$$\frac{\partial E_n}{\partial x_{ij}} = w_{ij} \frac{\partial E_n}{\partial s_i} = w_{ij} e_i$$

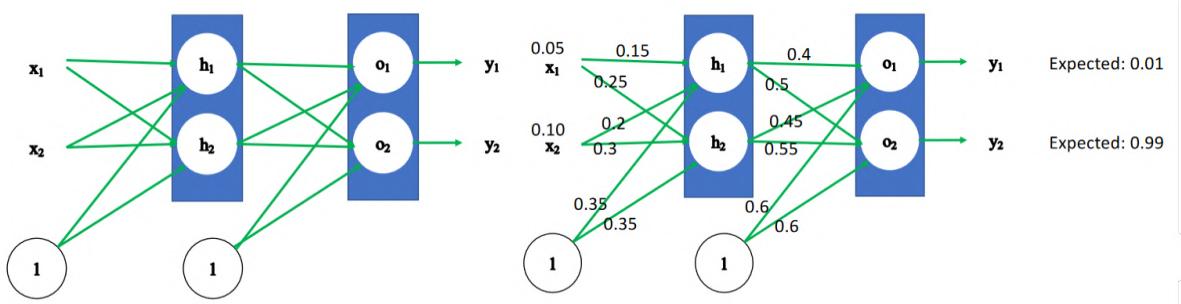
Gli output di una unità o di un livello diventano gli input per il livello successivo. Pertanto, per calcolare la derivata rispetto a y_i , dobbiamo sommare i contributi rispetto a tutti gli input x_{ij} :

$$\frac{\partial E_n}{\partial x_{ij}} = w_{ij} \frac{\partial E_n}{\partial s_i} = w_{ij} e_i \quad \Rightarrow \quad \frac{\partial E_n}{\partial y_i} = \sum_{k>i} \frac{\partial E_n}{\partial x_{ki}} = \sum_{k>i} w_{ki} e_k$$

$$e_i = h'(s_i) \frac{\partial E_n}{\partial y_i} = h'(s_i) \sum_{k>i} w_{ki} e_k$$

Per calcolare l'errore di una unità (*backpropagation*), sommiamo pesantemente gli errori provenienti dalle unità successive e poi moltiplichiamo questo valore per la derivata della funzione di attivazione corrente $h'(s_i)$.

- ▶ Questa regola di backpropagation ha una spiegazione molto intuitiva. L'errore (derivata della funzione di perdita) per una determinata unità dipende dagli errori delle unità a cui essa trasmette, moltiplicati per i pesi che le collegano.
- ▶ La pendenza della funzione di attivazione $h'(s_i)$ modula questa interazione.
- ▶ Le derivate e la propagazione dell'errore attraverso le altre unità seguono la stessa procedura:
 - applicare ricorsivamente la regola della catena,
 - calcolare le derivate analitiche delle funzioni applicate,
 - fino a ottenere le derivate della funzione di perdita rispetto a tutti i parametri da ottimizzare, ovvero, il gradiente della perdita.
- ▶ Una tipica implementazione dell'addestramento di reti neurali memorizza le attivazioni per un dato campione e le utilizza durante la fase di backpropagation (propagazione dell'errore all'indietro) per calcolare le derivate dei pesi.



$$h_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775, \\ h_2 = 0.25 \times 0.05 + 0.3 \times 0.1 + 0.35 = 0.3925,$$

$$\text{out}(h_1) = \sigma(h_1) = 0.5933, \\ \text{out}(h_2) = \sigma(h_2) = 0.5969.$$

$$o_1 = 0.4 \times 0.5933 + 0.45 \times 0.5969 + 0.6 = 1.106, \\ o_2 = 0.5 \times 0.5933 + 0.55 \times 0.5969 + 0.6 = 1.2249,$$

$$\text{out}(o_1) = \sigma(o_1) = 0.7514, \\ \text{out}(o_2) = \sigma(o_2) = 0.7729.$$

$$y_1 = \text{out}(o_1) = 0.7514,$$

$$y_2 = \text{out}(o_2) = 0.7729.$$

Errore Totale

$$\begin{aligned} e_1^2 &= (t_1 - y_1)^2 = (0.01 - 0.7514)^2 = 0.5497, \\ e_2^2 &= (t_2 - y_2)^2 = (0.99 - 0.7729)^2 = 0.0471, \\ E &= \frac{e_1^2 + e_2^2}{2} = \frac{0.5497 + 0.0471}{2} = 0.2984. \end{aligned}$$

Backpropagation

Gradiente rispetto alle uscite o_1 e o_2

$$\begin{aligned} \frac{\partial E}{\partial o_1} &= e_1 \cdot y_1(1 - y_1) = -0.7414 \times 0.7514 \times (1 - 0.7514) = -0.1385, \\ \frac{\partial E}{\partial o_2} &= e_2 \cdot y_2(1 - y_2) = 0.2171 \times 0.7729 \times (1 - 0.7729) = 0.0381. \end{aligned}$$

Gradiente rispetto ai pesi $w_7, w_8, w_9, w_{10}, w_{11}, w_{12}$

$$\begin{aligned} \frac{\partial E}{\partial w_7} &= \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_7} = -0.1385 \times 0.5933 = -0.0822, \\ \frac{\partial E}{\partial w_8} &= \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_8} = -0.1385 \times 0.5969 = -0.0827, \\ \frac{\partial E}{\partial w_9} &= \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_9} = -0.1385 \times 1 = -0.1385, \\ \frac{\partial E}{\partial w_{10}} &= \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_{10}} = 0.0381 \times 0.5933 = 0.0226, \\ \frac{\partial E}{\partial w_{11}} &= \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_{11}} = 0.0381 \times 0.5969 = 0.0227, \\ \frac{\partial E}{\partial w_{12}} &= \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_{12}} = 0.0381 \times 1 = 0.0381. \end{aligned}$$

Gradiente rispetto agli output di h_1 e h_2

$$\begin{aligned} \frac{\partial E}{\partial \text{out}(h_1)} &= \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial \text{out}(h_1)} + \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial \text{out}(h_1)} \\ &= -0.1385 \times 0.4 + 0.0381 \times 0.5 = -0.03635, \\ \frac{\partial E}{\partial \text{out}(h_2)} &= \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial \text{out}(h_2)} + \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial \text{out}(h_2)} \\ &= -0.1385 \times 0.45 + 0.0381 \times 0.55 = -0.04137. \end{aligned}$$

Gradiente rispetto agli input di h_1 e h_2

$$\begin{aligned}\frac{\partial E}{\partial h_1} &= \frac{\partial E}{\partial \text{out}(h_1)} \cdot \text{out}(h_1)(1 - \text{out}(h_1)) \\ &= -0.03635 \times 0.5933 \times (1 - 0.5933) = -0.00877,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial h_2} &= \frac{\partial E}{\partial \text{out}(h_2)} \cdot \text{out}(h_2)(1 - \text{out}(h_2)) \\ &= -0.04137 \times 0.5969 \times (1 - 0.5969) = -0.00995.\end{aligned}$$

Gradiente rispetto ai pesi $w_1, w_2, w_3, w_4, w_5, w_6$

$$\begin{aligned}\frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} = -0.00877 \times 0.05 = -0.0004385,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_2} &= \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_2} = -0.00877 \times 0.1 = -0.000877,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_3} &= \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_3} = -0.00877 \times 1 = -0.00877,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_4} &= \frac{\partial E}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_4} = -0.00995 \times 0.05 = -0.0004975,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_5} &= \frac{\partial E}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_5} = -0.00995 \times 0.1 = -0.000995,\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_6} &= \frac{\partial E}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_6} = -0.00995 \times 1 = -0.00995.\end{aligned}$$

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])

model.fit(
    X,
    y,
    batch_size=64,
    epochs=50)
```

Build the **computational graph**. We must specify loss, optimizer, and metric

Train the network:
Pass data, specify batch_size and epochs

Epoch 1/50
16/16 [=====] - 1s 1ms/step - loss: 0.6933 - accuracy: 0.4910
Epoch 2/50
16/16 [=====] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.5080
Epoch 3/50
16/16 [=====] - 0s 1ms/step - loss: 0.6929 - accuracy: 0.5200
...
16/16 [=====] - 0s 1ms/step - loss: 0.6924 - accuracy: 0.5170
Epoch 49/50
16/16 [=====] - 0s 1ms/step - loss: 0.6924 - accuracy: 0.5170
Epoch 50/50
16/16 [=====] - 0s 1ms/step - loss: 0.6925 - accuracy: 0.5170

This is just an example **trained with random points and labels...**
Indeed, it is not learning!

7.21 Grafico Computazionale

- ▶ Il pannello Grafici di TensorBoard è uno strumento potente per esaminare il modello di TensorFlow.
- ▶ Puoi visualizzare rapidamente un grafico concettuale della struttura del tuo modello e assicurarti che corrisponda al design previsto.
- ▶ Puoi anche visualizzare un grafico a livello di operazioni per capire come TensorFlow interpreta il tuo programma.
- ▶ Esaminare il grafico a livello di operazioni può darti intuizioni su come modificare il tuo modello. Ad esempio, puoi riprogettare il modello se l'allenamento procede più lentamente del previsto.

7.22 Grafico Operativo (da TensorBoard)

- ▶ Dal terminale, esegui `tensorboard --logdir logs`.
- ▶ TensorBoard consente di scomporre la struttura dell'intero flusso di esecuzione in forma di grafico a livello operativo, mostrando i dettagli di ogni operazione.
- ▶ Possiamo visualizzare le operazioni a livello più basso, che possono fornire intuizioni importanti riguardo al design e all'ottimizzazione di ciascun componente del nostro flusso di lavoro.

7.23 Ispezione dei Risultati da fit

- ▶ La funzione `fit` restituisce un oggetto con diversi attributi.
- ▶ Tra questi, c'è `history`, che è un dizionario che memorizza informazioni sull'allenamento, come:
 - `Loss`: un array con il valore della perdita (loss) nel corso delle epoche, utile per la creazione di grafici.
 - `Accuracy`: un array con i valori dell'accuracy nel corso delle epoche.

7.24 Regolarizzazione tramite p-norma

- ▶ La regolarizzazione e altre tecniche possono essere utilizzate per prevenire l'overfitting delle reti neurali, migliorando così la loro capacità di generalizzare su dati non visti.
- ▶ Le penalità di p -norma sui pesi possono essere utilizzate per migliorare il condizionamento del sistema e ridurre l'overfitting.
- ▶ Impostare $p = 2$ risulta nella solita regolarizzazione $L2$ e riduce i pesi elevati.
- ▶ Usare $p = 1$ è chiamato *lasso* e può portare alcuni pesi fino a zero.

- ▶ Man mano che i pesi vengono ottimizzati all'interno di una rete neurale, questi termini rendono i pesi più piccoli.

Regolarizzazione tramite p-norma in TensorFlow

Ogni layer in TensorFlow ha un parametro per specificare se utilizzare o meno un regolarizzatore:

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_regularizer=None, # Regolarizzatore per applicare una penalità sul kernel del layer
    bias_regularizer=None,   # Regolarizzatore per applicare una penalità sul bias del layer
    activity_regularizer=None # Regolarizzatore per applicare una penalità sull'output del layer
)

tf.keras.regularizers.L1(l1=0.01, **kwargs) # l1 * reduce_sum(abs(x))
tf.keras.regularizers.L2(l2=0.01, **kwargs) # l2 * reduce_sum(square(x))
tf.keras.regularizers.L1L2(l1=0.0, l2=0.0) # Combina i due
dense = tf.keras.layers.Dense(3, kernel_regularizer='l2')
```

7.25 Regolarizzazione tramite Data Augmentation

- ▶ Un'altra potente tecnica per ridurre l'overfitting è aggiungere più campioni di addestramento perturbando gli input e/o gli output dei campioni già raccolti.
- ▶ L'augmentation del dataset è particolarmente efficace nei compiti di classificazione delle immagini, dove le classi di immagini non dovrebbero cambiare sotto piccole perturbazioni locali.
- ▶ Per le immagini, le distorsioni possono consistere in:
 - Aggiungere rumore ai pixel degli input;
 - Spostare i pixel applicando trasformazioni geometriche alle immagini.

Regolarizzazione tramite Data Augmentation in TensorFlow

- ▶ TensorFlow fornisce funzioni per aiutare con il data augmentation.
- ▶ Queste funzioni sono a volte specializzate per il dataset in questione.
- ▶ In generale, è importante considerare attentamente il problema e implementare la tecnica di data augmentation che meglio si adatta all'obiettivo.
- ▶ Per le immagini, una tecnica tipica è ruotare l'immagine. Altre tecniche includono rafforzare il contrasto, equalizzare, ecc.

7.26 Regolarizzazione tramite Dropout

- ▶ Il dropout è una tecnica di regolarizzazione introdotta da Srivastava, Hinton et al. (2014).
- ▶ A ogni mini-batch durante l’allenamento, una certa percentuale p (ad esempio, 50%) delle unità in ogni layer viene fissata a zero.
- ▶ Impostare casualmente le unità a zero introduce rumore nel processo di allenamento e previene la specializzazione eccessiva delle unità su campioni o compiti particolari.
- ▶ Poiché eliminare p delle unità riduce il valore atteso di qualsiasi somma a cui l’unità contribuisce di una frazione $(1 - p)$, le somme pesate s_i in ciascun layer vengono moltiplicate (durante l’allenamento) per $(1 - p)^{-1}$.

Regolarizzazione tramite Dropout in TensorFlow

```
tf.keras.layers.Dropout(
    rate, noise_shape=None, seed=None, **kwargs
)
```

7.27 Regolarizzazione tramite Batch Normalization

- ▶ Ottimizzare i pesi in una rete neurale profonda può essere lento nel convergere.
- ▶ Uno dei problemi classici con le tecniche di ottimizzazione iterativa è il cattivo condizionamento, in cui le componenti del gradiente variano notevolmente in magnitudine.
- ▶ La normalizzazione del batch (Ioffe e Szegedy 2015) riscalava (e ricentra) le attivazioni in una determinata unità in modo che abbiano varianza unitaria e media zero.
- ▶ Effettuiamo questa normalizzazione considerando tutti i campioni di addestramento n in un determinato mini-batch B e calcolando le statistiche di media e varianza per l’unità i .
- ▶ Questa normalizzazione può ostacolare l’allenamento se non necessaria. Pertanto, un ulteriore guadagno e bias (addestrabili) vengono aggiunti a ciascuna unità i .

Regolarizzazione tramite Batch Normalization in TensorFlow

```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer='zeros',    # è un fattore di offset appreso
    gamma_initializer='ones',   # è un fattore di scaling appreso
    moving_mean_initializer='zeros',
    moving_variance_initializer='ones',
```

```

        beta_regularizer=None,
        gamma_regularizer=None,
        beta_constraint=None,
        gamma_constraint=None,
        synchronized=False,
        **kwargs
)

```

7.28 Ottimizzatori

- ▶ I praticanti hanno sviluppato una serie di algoritmi di ottimizzazione basati su estensioni della discesa del gradiente stocastico (SGD).
- ▶ Nella SGD, invece di valutare la funzione di perdita sommando su tutti i campioni di addestramento, valutiamo un singolo campione di addestramento n e calcoliamo le derivate della perdita associata $E_n(w)$.
- ▶ Poi facciamo un piccolo passo discendente lungo la direzione di questo gradiente.
- ▶ Nella discesa del gradiente batch, tutti i dati di addestramento vengono utilizzati in una singola iterazione. Il gradiente della funzione di perdita viene calcolato per ciascun campione, mediato, e poi utilizzato per aggiornare i parametri.
- ▶ Nella discesa del gradiente stocastico, si utilizza un singolo campione di addestramento per iterazione. Questo fornisce più frequenti aggiornamenti, ma introduce rumore nella stima del gradiente.
- ▶ In un certo senso, si potrebbe pensare alla discesa del gradiente stocastico come a una regolarizzazione che stabilizza gli effetti della discesa del gradiente batch.

Ottimizzatori in TensorFlow

```

tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    name='Adam',
    **kwargs
)

```

Esempio Completo

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()

train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

train_images = np.expand_dims(train_images, axis=-1)
test_images = np.expand_dims(test_images, axis=-1)

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential([
    Dense(128, activation='relu', input_shape=(28 * 28,)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(
    train_images,
    train_labels,
    epochs=5,
    batch_size=32
)

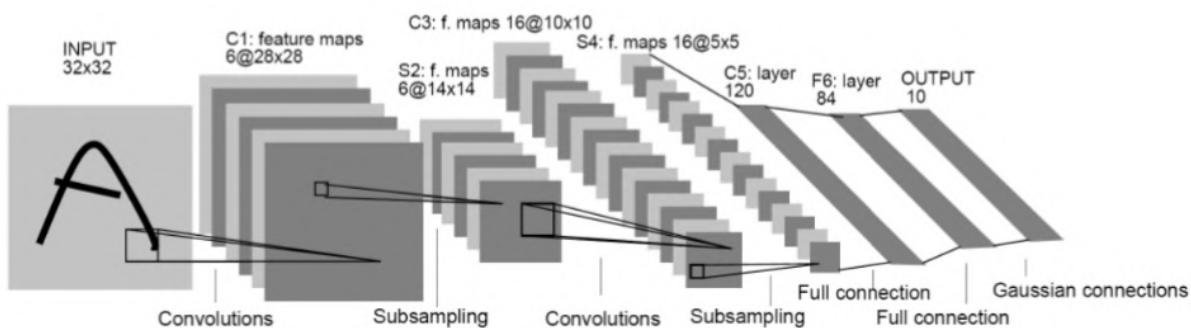
model.evaluate(
    test_images,
```

```
    test_labels  
)
```

RETI CONVOLUZIONALI

8.1 Convolutional Neural Network

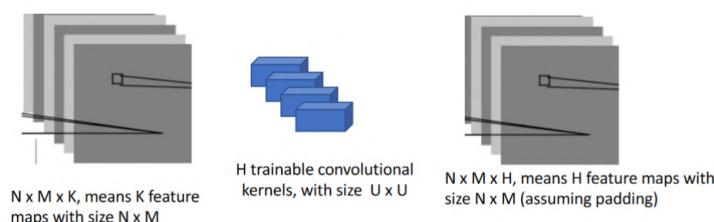
- ▶ Il componente più cruciale delle reti profonde per l'elaborazione delle immagini e la visione artificiale è l'uso di convoluzioni multistrato addestrabili.
- ▶ L'idea delle reti neurali convoluzionali è stata popolarizzata da LeCun nel 1998, quando ha introdotto la rete LeNet-5 per il riconoscimento delle cifre.



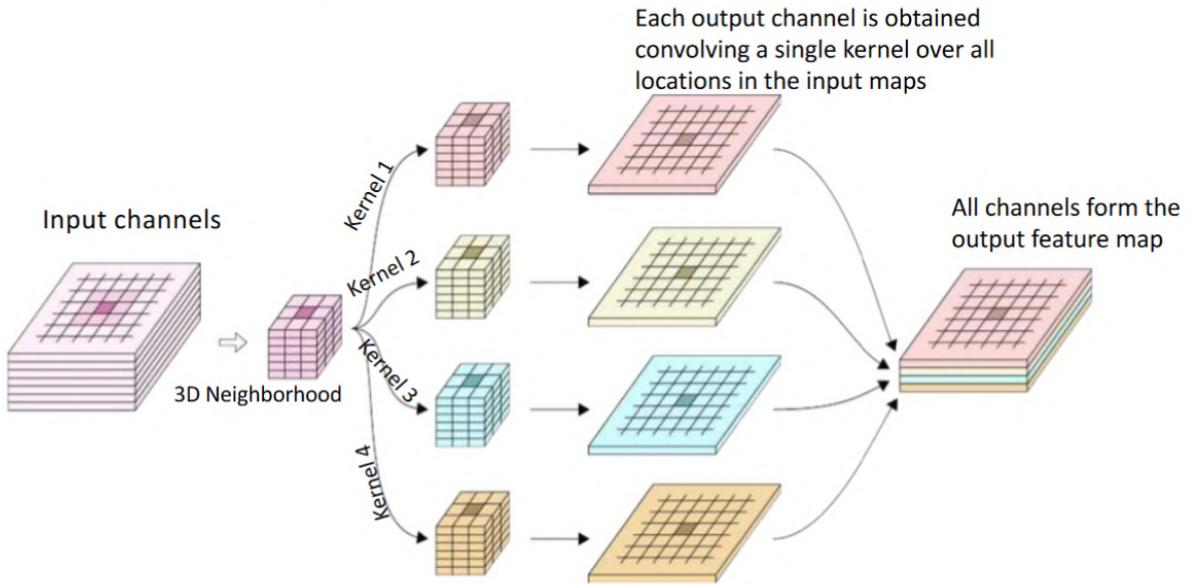
- ▶ Questa rete utilizza più canali in ogni layer e alterna convoluzioni multicanale con operazioni di downsampling, seguite da alcuni strati completamente connessi che producono un'attivazione per ciascuna delle 10 cifre classificate.

8.2 Layer Convoluzionali

- ▶ Le reti convoluzionali organizzano ogni layer in mappe di caratteristiche, che possono essere pensate come piani paralleli o canali.
- ▶ In un layer convoluzionale, le somme ponderate vengono eseguite solo all'interno di una piccola finestra locale, e i pesi sono identici per tutti i pixel, proprio come nella convoluzione e correlazione delle immagini regolari invarianti per traslazione.
- ▶ A differenza della convoluzione delle immagini, dove lo stesso filtro viene applicato a ciascun canale (colore), le convoluzioni combinano linearmente le attivazioni da ciascuno dei C_1 canali di input in un layer precedente e utilizzano diversi kernel di convoluzione per ciascuno dei C_2 canali di output.



- ▶ Ogni kernel di convoluzione 2D prende tutti i canali di input finestrati su una piccola area e produce i valori in uno dei canali di output.

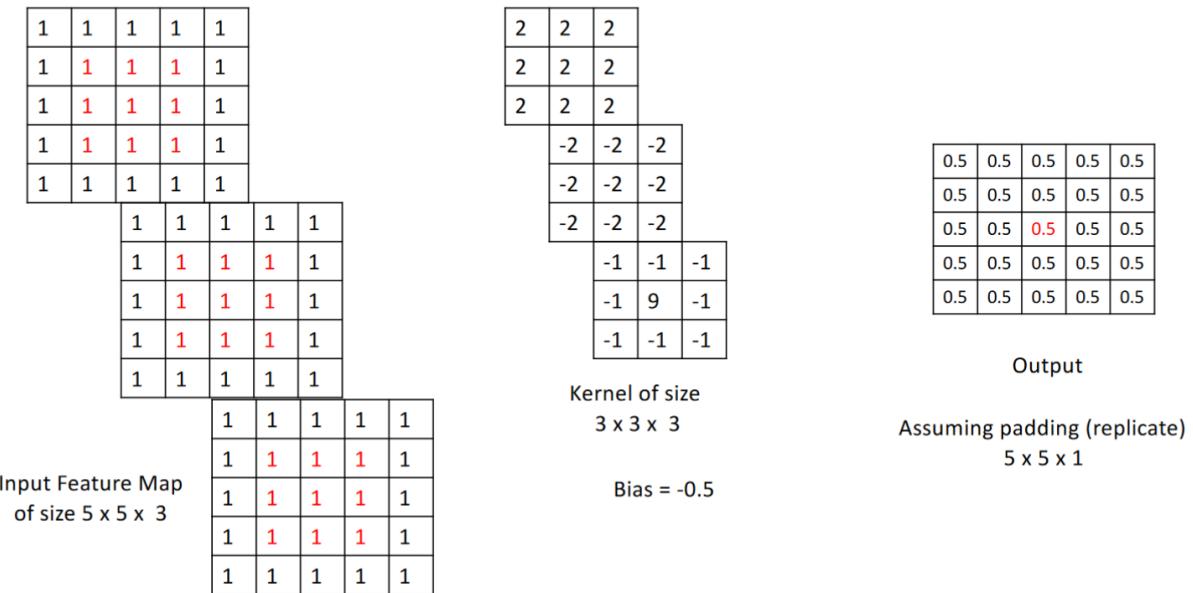


- ▶ I layer convoluzionali costruiscono caratteristiche locali e le combinano in modi diversi per produrre caratteristiche più discriminative e semanticamente significative.
- ▶ Data una mappa di caratteristiche x con C_1 canali, il valore di output nella posizione (i, j, c_2) è calcolato come

$$s(i, j, c_2) = \sum_{c_1 \in C_1} \sum_{(k, l) \in N} w(k, l, c_1, c_2) x(i + k, j + l, c_1) + b(c_2)$$

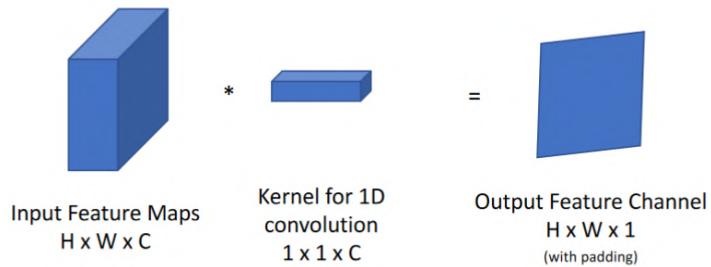
- ▶ L'intero layer convoluzionale ha quindi un numero di parametri pari a $(S \times S \times C_1 + 1) \times C_2$.
- ▶ C_2 è il numero di kernel (di dimensione $S \times S$) nel layer.
- ▶ Considerando che l'operazione viene eseguita per ogni elemento nella mappa di caratteristiche di output, assumendo una dimensione $W \times H$, il numero totale di operazioni (moltiplicazioni-somme) è $O(WHS^2C_1C_2)$.

Example



8.3 Convoluzione 1D

- ▶ Alcune reti neurali, come il modulo Inception in GoogLeNet (Szegedy 2015), utilizzano convoluzioni 1×1 , che non eseguono effettivamente convoluzioni ma combinano vari canali su base per-pixel, spesso con l'obiettivo di ridurre la dimensionalità dello spazio delle caratteristiche.



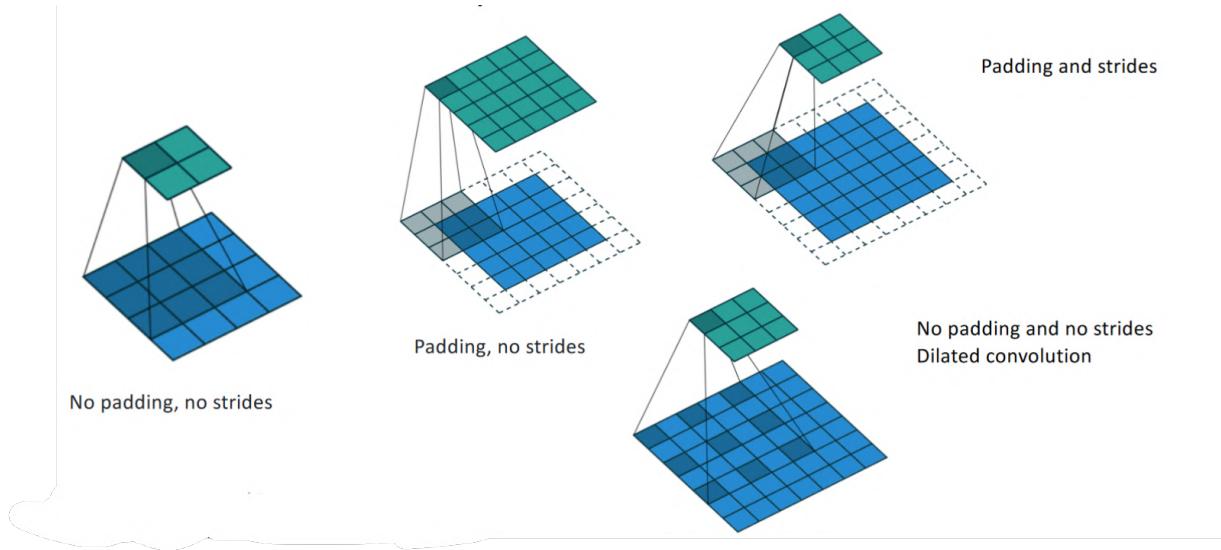
8.4 Layer Convoluzionale vs Layer Denso

- ▶ Poiché i pesi in un kernel di convoluzione sono gli stessi per tutti i pixel all'interno di un dato layer e canale, questi pesi sono effettivamente condivisi.
- ▶ Questo significa che ci sono molti meno pesi da apprendere rispetto ai layer completamente connessi. Significa anche che durante la retropropagazione, gli aggiornamenti dei pesi del kernel sono sommati su tutti i pixel in un dato layer/canale.
- ▶ In un ConvLayer, il numero di parametri è $(S \times S \times C_1 + 1) \times C_2$.
- ▶ In un layer denso con C_2 neuroni, avremmo $(W \times H \times C_1 + 1) \times C_2$.

- ▶ In generale, i kernel hanno dimensioni piccole rispetto alle mappe di caratteristiche.
- ▶ Diversamente dai layer densi, nei ConvLayer manteniamo l'informazione spaziale e teniamo conto delle proprietà locali nell'immagine.

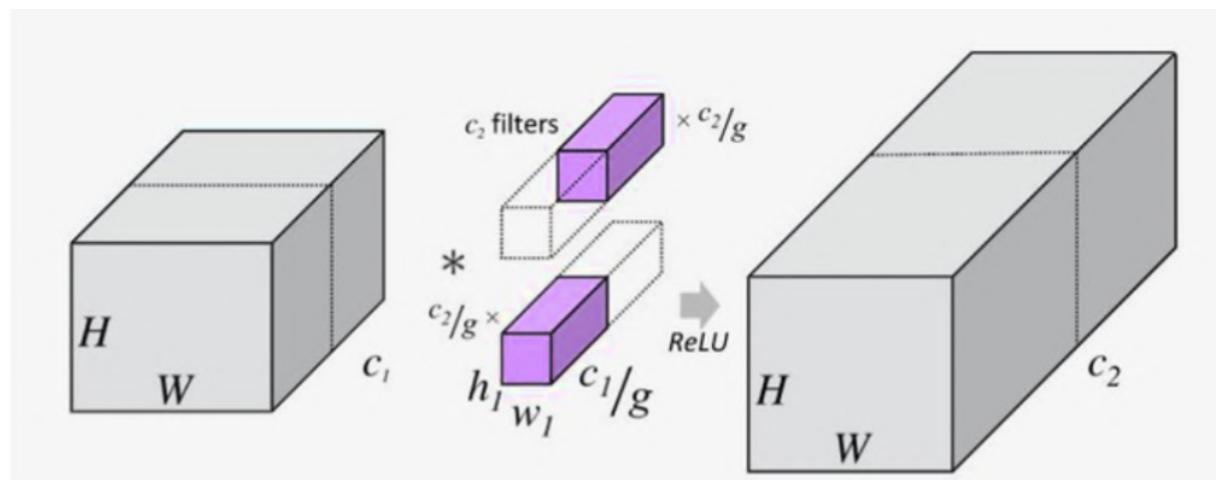
8.5 Layer Convolzionale – Altri Parametri

- ▶ **Padding:**
 - Le reti precoci come LeNet-5 non eseguivano padding sull'immagine, che quindi si riduceva dopo ogni convoluzione.
 - Le reti moderne possono specificare facoltativamente una larghezza e modalità di padding, utilizzando una delle scelte tradizionali nell'elaborazione delle immagini, come lo zero padding o la replicazione dei pixel.
- ▶ **Stride:**
 - Lo stride predefinito per la convoluzione è 1 pixel, ma è anche possibile valutare la convoluzione solo ad ogni n -esima colonna e riga.
- ▶ **Dilation:**
 - Righe e colonne saltate possono essere inserite tra i campioni di pixel durante la convoluzione. Sebbene in linea di principio ciò possa portare a aliasing, può anche essere efficace nel fare pooling su una regione più ampia utilizzando meno operazioni e parametri apprendibili.
- ▶ **Grouping:**
 - Per impostazione predefinita, tutti i canali di input vengono utilizzati per produrre ogni canale di output. Possiamo anche raggruppare i layer di input e output in G gruppi separati, ciascuno dei quali viene convoluto separatamente. $G = 1$ corrisponde alla convoluzione regolare, mentre $G = C_1$ significa che ciascun canale di input corrispondente viene convoluto indipendentemente dagli altri, il che è noto come convoluzione depthwise o separata per canale.



8.6 Grouping

- ▶ Il raggruppamento è utile nelle grandi reti neurali, dove il numero di canali cresce con la profondità.
- ▶ Aiuta a ridurre il numero di parametri e fornisce caratteristiche migliori.
- ▶ Normalmente, in una convoluzione regolare, abbiamo filtri che scorrono su tutto l'input.
- ▶ Utilizzando convoluzioni raggruppate, possiamo separare i filtri in gruppi disgiunti, ovvero gruppi di filtri che convolvono su mappe di caratteristiche distinte.
- ▶ Ad esempio, se vogliamo applicare 32 filtri, possiamo dividere i 32 in 2 gruppi di 16 filtri, il primo gruppo convolverà sui primi 50% dei canali di input, mentre il secondo convolverà sui secondi 50%.



8.7 Layer Convoluzionale in Tensorflow.keras

```
tf.keras.layers.Conv2D(filters, kernel_size,
strides=(1, 1),
padding="valid", # "valid" significa nessun padding. "same" risulta in padding con
#zeri
data_format=None, # channels_last (default) o channels_first.
dilation_rate=(1, 1),
groups=1,
activation=None,
use_bias=True,
kernel_initializer="glorot_uniform",
bias_initializer="zeros",
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None, **kwargs )
```

8.8 CNN in Tensorflow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense
```

- **Parte Convoluzionale:** Impara quali caratteristiche estrarre per risolvere il problema:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten()) ( W x H x C tensors are reshaped into W*H*C x 1 vectors)
```

- **Subnet Feed-Forward:** Impara come utilizzare le caratteristiche estratte per risolvere il problema. Ad esempio, impara come classificare.

```
model.add(Dense(64, activation='relu'))
model.add(Dense(10))
```

8.9 Pooling

- ▶ Stride maggiori di 1 possono essere utilizzati per ridurre la risoluzione di un determinato strato.
- ▶ Un'altra tecnica è l'utilizzo del pooling.
- ▶ Dopo aver diviso le mappe delle caratteristiche in finestre (scorrevoli), è possibile prendere il valore medio o massimo dei valori nella finestra in modo canale-per-canale.
- ▶ Questi sono chiamati rispettivamente average pooling e max pooling e possono anche utilizzare stride maggiori di 1.
- ▶ Stride e dimensioni delle finestre comuni per il max pooling sono uno stride di 2 e finestre non sovrapposte 2×2 o finestre sovrapposte 3×3 .
- ▶ Il pooling dovrebbe fornire una certa invarianza agli spostamenti negli input. Tuttavia, la maggior parte delle reti profonde non è così invariante agli spostamenti, il che degrada le loro prestazioni.

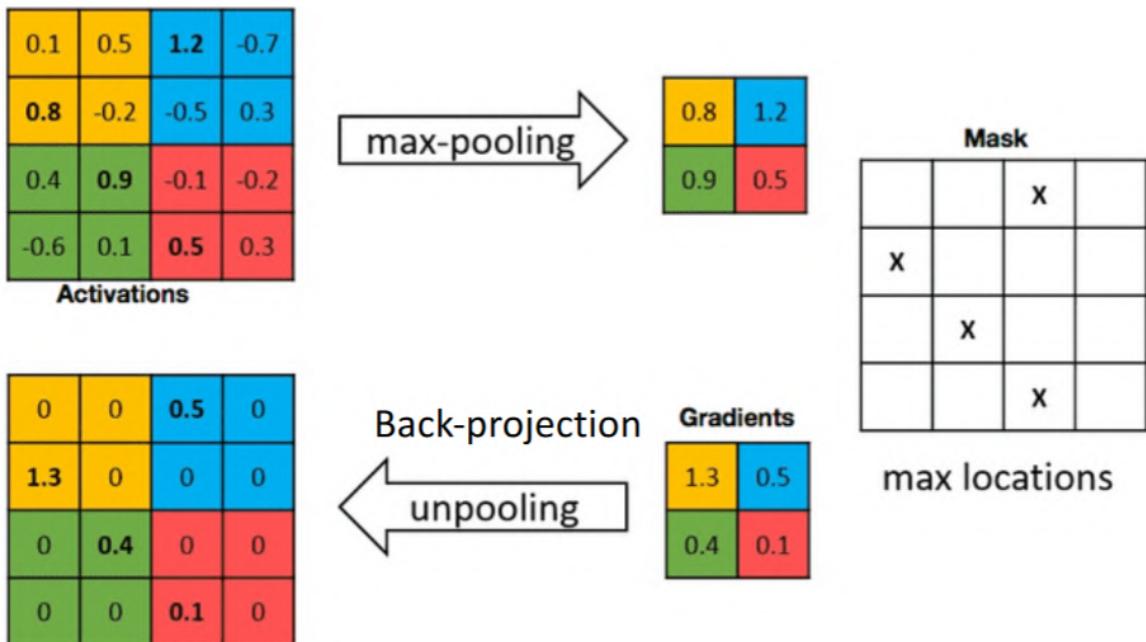


8.10 Pooling in TensorFlow

```
tf.keras.layers.MaxPooling2D
pool_size=(2, 2), # dimensione della finestra su cui prendere il massimo
strides=None, # specifica di quanto si sposta la finestra di pooling per ogni passo
padding="valid", # "valid" significa senza padding, "same" con padding
data_format=None,
**kwargs
tf.keras.layers.AveragePooling2D
pool_size=(2, 2),
strides=None,
padding="valid",
data_format=None,
**kwargs
```

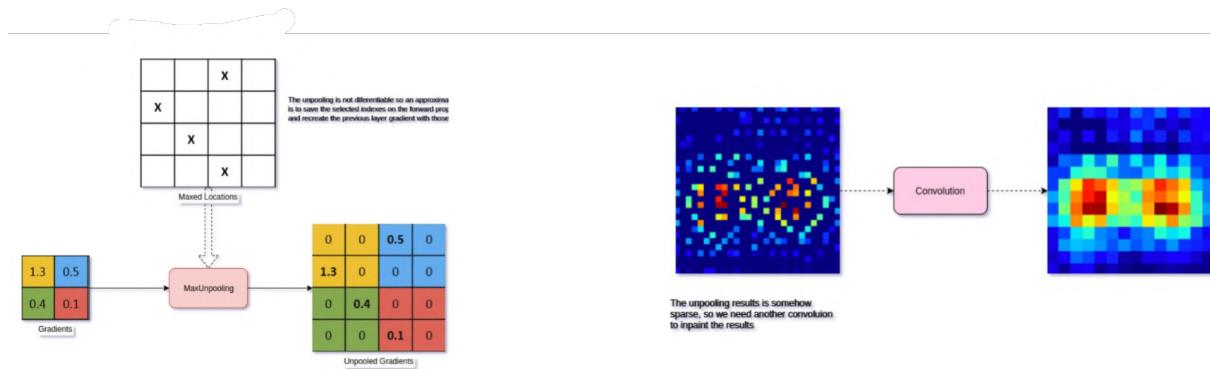
8.11 Max-Pooling e Backpropagation

- ▶ L'operatore di max-pooling agisce come un "interruttore" che collega una delle unità di input all'unità di output.
- ▶ Durante la backpropagation, dobbiamo solo trasmettere l'errore e le derivate all'unità massimamente attiva, purché si ricordi quale unità ha generato questa risposta.



8.12 Unpooling

- ▶ Questo stesso meccanismo di max-unpooling può essere utilizzato per creare una "rete di deconvoluzione" quando si cerca lo stimolo che attiva più fortemente una particolare unità.
- ▶ L'unpooling può essere utilizzato per (approssimativamente) invertire l'effetto dell'operazione di max pooling. Funziona come un upsampler e necessita di un ConvLayer per "inpaintare" (filtro passa-basso) i risultati dell'upsampling.



Example

```
import numpy as np
import tensorflow as tf
import tensorflow_addons as tfa

x = tf.keras.Input(shape=(6,6,1))
p, indices = tf.nn.max_pool_with_argmax(x, 2, 2, padding="SAME")
up = tfa.layers.MaxUnpooling2D()(p, indices)
model = tf.keras.Model(inputs=x, outputs=[p, up])
model.summary()

#Let us evaluate the model
ones_matrix = np.ones((1, 6, 6, 1))
ones_matrix[0,0,0,0] = 3
ones_matrix[0,-1,-1,0] = 5
res = model(ones_matrix)

print("MaxPooling")
print(res[0][0, :, :, 0].numpy())
print("MaxUnPooling2D")
print(res[1][0, :, :, 0].numpy())
```

```
Total params: 0
Trainable params: 0
Non-trainable params: 0
-----
MaxPooling
[[3. 1. 1.]
 [1. 1. 1.]
 [1. 1. 5.]]
MaxUnPooling2D
[[3. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 5.]]
```

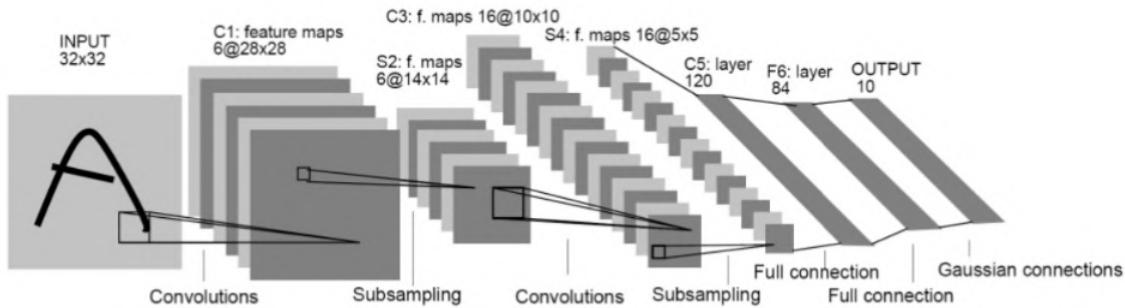
8.13 Convoluzione Trasposta - Deconvoluzione

- ▶ Se vogliamo invertire uno strato convoluzionale, possiamo aggiungere righe e colonne extra di zeri tra i pixel nello strato di input e quindi eseguire una normale convoluzione.
- ▶ Questa operazione è chiamata convoluzione inversa, anche se è più comunemente nota come convoluzione trasposta, perché quando le convoluzioni sono scritte in forma matriciale, questa operazione è una moltiplicazione con una matrice dei pesi trasposta.

- ▶ Come con la convoluzione regolare, si possono specificare parametri di padding, stride, dilatazione e raggruppamento. Tuttavia, in questo caso, lo stride specifica il fattore con cui l'immagine sarà upsampleata anziché downsampleata.

8.14 LeNet-5

- ▶ Una delle prime applicazioni commerciali delle CNN è stata LeNet-5 (LeCun et al. 1998).
- ▶ Contiene la maggior parte degli elementi delle moderne CNN, anche se utilizzava non-linearietà sigmoide, average pooling e unità Gaussian RBF invece del softmax nel suo output.
- ▶ La rete è stata inizialmente impiegata per leggere automaticamente gli assegni depositati negli sportelli bancomat per verificare che gli importi scritti e digitati fossero gli stessi.
- ▶ Dataset utilizzati per l'addestramento e il test: MNIST (LeCun et al. 1998), CIFAR-10 (Krizhevsky 2009) o Fashion MNIST (Xiao et al. 2017).



8.15 CIFAR-10

- ▶ Il dataset CIFAR-10 consiste di 60.000 immagini a colori 32x32 in 10 classi, con 6.000 immagini per classe. Ci sono 50.000 immagini di addestramento e 10.000 immagini di test.

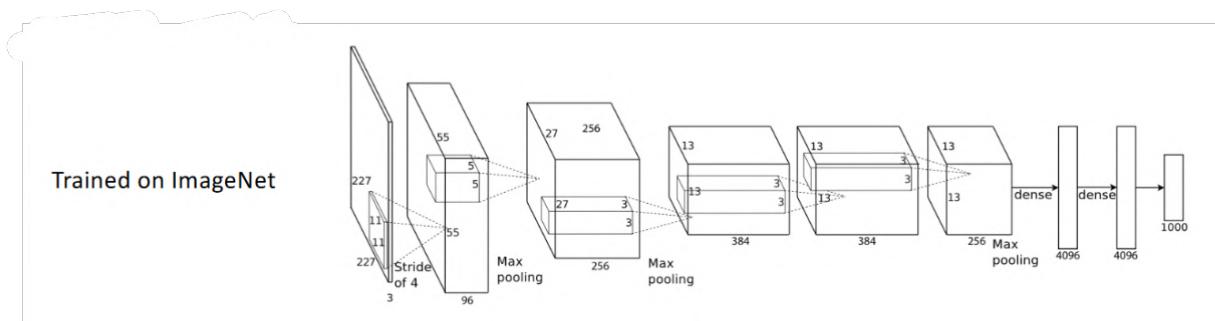
8.16 Fashion-MNIST

- ▶ Fashion-MNIST è un dataset di immagini di articoli Zalando—consistente in un set di addestramento di 60.000 esempi e un set di test di 10.000 esempi. Ogni esempio è un'immagine in scala di grigi 28x28, associata a un'etichetta di una delle 10 classi.

8.17 AlexNet

- ▶ Il successo delle CNN è iniziato nel 2012 con l'articolo di Krizhevsky, Sutskever e Hinton (2012).

- ▶ L'AlexNet (anche chiamata rete SuperVision) ha portato a una drastica riduzione dei tassi di errore dal 25,8
- ▶ Contiene una serie di strati convoluzionali con non-linearietà ReLU (rettificate lineari), max pooling, alcuni strati completamente connessi e uno strato finale softmax, che viene alimentato in una perdita di entropia incrociata multi-classe. La rete include anche dropout e utilizza piccole traduzioni e manipolazioni del colore per l'augmentation dei dati, momentum e regolarizzazione del peso (penalità L2 sul peso).

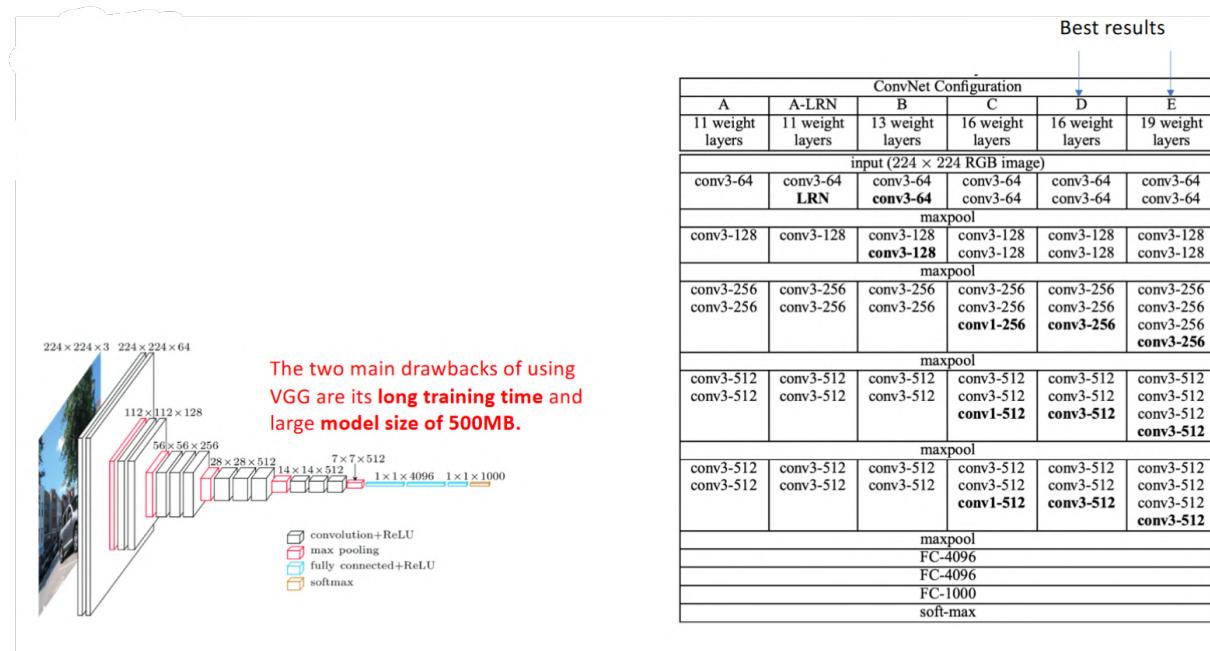


8.18 ImageNet

- ▶ ImageNet è organizzato secondo i nomi nella gerarchia di WordNet, in cui ogni nodo della gerarchia è rappresentato da centinaia e migliaia di immagini. I dati sono disponibili gratuitamente per i ricercatori per uso non commerciale.
- ▶ Il sottoinsieme più utilizzato di ImageNet è l'ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012-2017 dataset di classificazione e localizzazione delle immagini.
- ▶ Questo dataset copre 1000 classi di oggetti e contiene 1.281.167 immagini di addestramento, 50.000 immagini di validazione e 100.000 immagini di test. Questo sottoinsieme è disponibile su Kaggle.

8.19 VGG

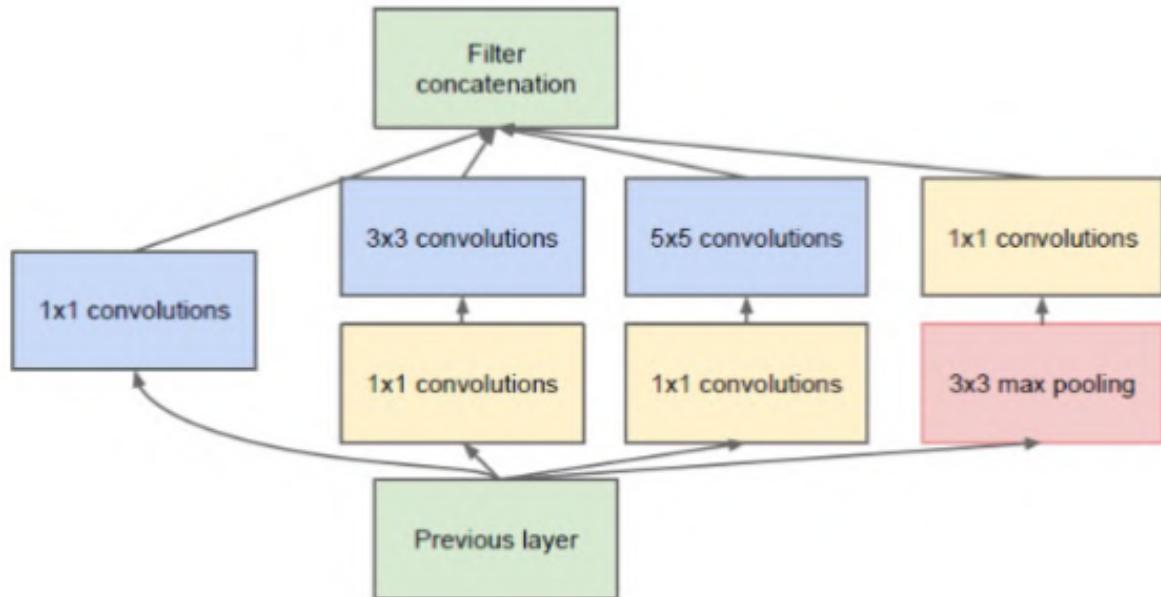
- ▶ Simonyan e Zisserman nel 2014 hanno proposto la rete Oxford Visual Geometry Group (VGG).
- ▶ VGG utilizza blocchi di convoluzione/ReLU 3×3 ripetuti, intervallati da max-pooling 2×2 e raddoppiamento dei canali, seguiti da alcuni strati completamente connessi, per produrre reti di 16–19 strati che riducono ulteriormente l'errore del 40%
- ▶ VGG-16 ha 16 strati, VGG-19 ne ha 19. Paper: <https://arxiv.org/abs/1409.1556>
- ▶ **Risultati migliori:** I due principali svantaggi dell'uso di VGG sono i tempi di addestramento lunghi e la grande dimensione del modello di 500MB.



8.20 GoogLeNet e Inception

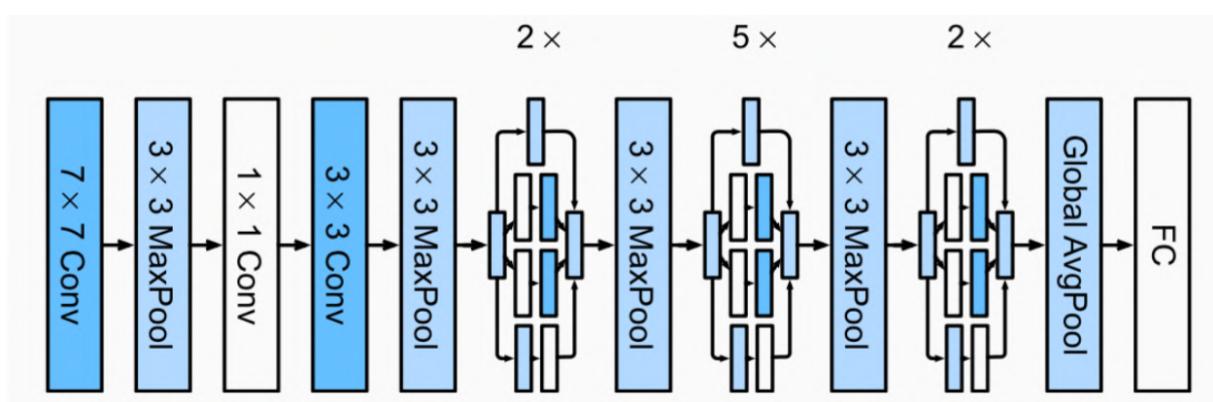
- ▶ GoogLeNet di Szegedy et al. 2015 si è concentrata invece sull'efficienza.
- ▶ Inizia con una rete aggressiva che utilizza una serie di convoluzioni regolari e con stride e strati di max-pooling per ridurre rapidamente la risoluzione delle immagini da 224×224 a 28×28 .
- ▶ Utilizza quindi un numero di moduli Inception, ciascuno dei quali è una piccola rete neurale ramificata le cui caratteristiche vengono concatenate alla fine.
- ▶ Una delle caratteristiche importanti di questo modulo è che utilizza convoluzioni "bottleneck" 1×1 per ridurre il numero di canali prima di eseguire convoluzioni più grandi 3×3 e 5×5 , risparmiando così una quantità significativa di calcolo.
- ▶ Questo tipo di proiezione seguita da un'ulteriore convoluzione è simile nello spirito all'approssimazione dei filtri come somma di convoluzioni separabili proposta da Perona (1995).

- ▶ Rimuove anche gli strati completamente connessi (MLP) alla fine, affidandosi invece al pooling medio globale seguito da uno strato lineare prima del softmax.
- ▶ Le sue prestazioni sono simili a quelle di VGG, ma con costi di calcolo e dimensioni del modello notevolmente inferiori.



8.21 GoogLeNet

- ▶ GoogLeNet utilizza uno stack di un totale di 9 blocchi Inception, disposti in 3 gruppi con max-pooling tra di essi, e pooling medio globale nella sua testa per generare le sue stime. Il max-pooling tra i blocchi Inception riduce la dimensionalità. Il primo modulo è simile ad AlexNet e LeNet.
- ▶ Codice Inception – utile nel modello sequenziale.

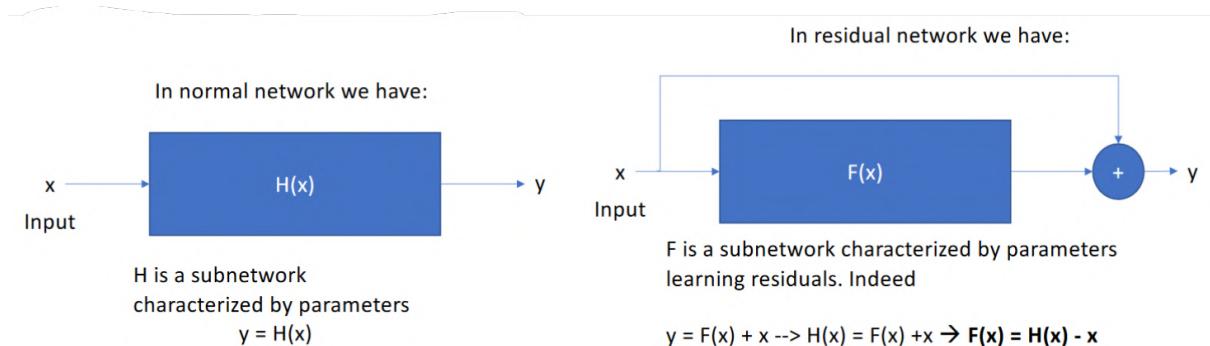


8.22 ResNet

- ▶ Le Reti Residuali (Residual Networks, ResNet) (He et al. 2016) hanno ampliato in modo drammatico il numero di strati (profondità) che potevano essere addestrati con successo (152 strati).
- ▶ Introduce connessioni di salto (skip connections), che permettono all'informazione (e ai gradienti) di fluire attorno a un insieme di strati convoluzionali. La rete impara i residui (differenze) tra un insieme di attivazioni in ingresso e in uscita.
- ▶ Una variante è il "blocco bottleneck" che riduce il numero di canali prima di eseguire lo strato convoluzionale 3×3 . Questa è un'identità (basta passare l'input così com'è).
- ▶ Per costruire una ResNet, vari blocchi residuali sono intervallati con convoluzioni a strisciamento e raddoppio dei canali per ottenere il numero desiderato di strati.
- ▶ Combinando vari numeri di blocchi residuali, sono state costruite e valutate ResNet di 18, 34, 50, 101 e 152 strati. Le reti più profonde hanno una maggiore accuratezza ma un costo computazionale più elevato.

8.23 Più sui Blocchi Residuali

- ▶ In una Rete Neurale Residuale, gli strati di peso apprendono funzioni residue rispetto agli input dello strato. Le connessioni di salto eseguono mapping di identità, unendosi agli output degli strati tramite somma.
- ▶ Le connessioni di salto di identità, spesso chiamate "connessioni residue", sono utilizzate nei modelli Transformer (ad es., BERT, GPT, ChatGPT), nel sistema AlphaGo Zero, nel sistema AlphaStar e nel sistema AlphaFold.



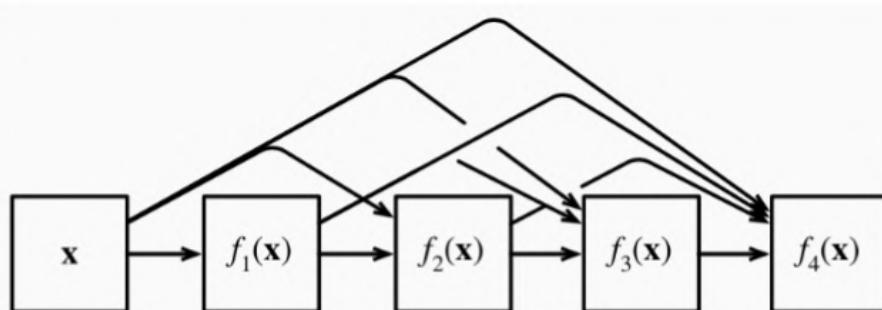
8.24 Blocchi Residuali e Backpropagation

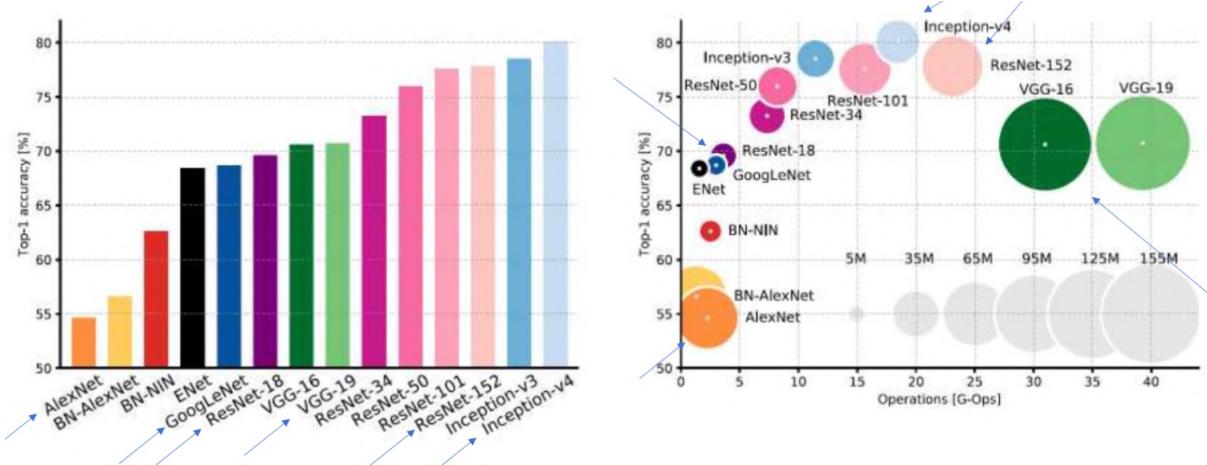
- ▶ La formulazione del Residual Learning fornisce il beneficio aggiunto di affrontare in una certa misura il problema del gradiente che svanisce.
- ▶ Infatti, quando si calcolano le derivate rispetto all'input x , sommiamo 1.

8.25 DenseNet

- ▶ DenseNet (rete convoluzionale densa), (Huang et al., 2017), è in una certa misura l'estensione logica delle connessioni residue.
- ▶ DenseNet è caratterizzata sia dal pattern di connettività in cui ogni strato si collega a tutti gli strati precedenti sia dall'operazione di concatenazione (anziché l'operatore di somma in ResNet) per preservare e riutilizzare le caratteristiche degli strati precedenti.
- ▶ Eseguiamo una mappatura da x ai suoi valori dopo aver applicato una sequenza sempre più complessa di funzioni.
- ▶ La mappatura è tale che alla fine tutte queste funzioni sono combinate in un MLP per ridurre nuovamente il numero di caratteristiche.
- ▶ I componenti principali che compongono un DenseNet sono i blocchi densi e gli strati di transizione. I primi definiscono come gli input e gli output vengono concatenati, mentre i secondi controllano il numero di canali affinché non sia troppo grande, poiché l'espansione.
- ▶ Un blocco denso consiste di più blocchi convoluzionali, ciascuno utilizzando lo stesso numero di canali di output. Nella propagazione in avanti, tuttavia, concateniamo l'input e l'output di ciascun blocco convoluzionale sulla dimensione del canale.
- ▶ Poiché ogni blocco denso aumenterà il numero di canali, aggiungerne troppi porterà a un modello eccessivamente complesso. Uno strato di transizione è utilizzato per controllare la complessità del modello. Riduce il numero di canali utilizzando una convoluzione 1×1 . Inoltre, dimezza l'altezza e la larghezza tramite pooling medio con uno stride di 2.
- ▶ Sebbene queste operazioni di concatenazione riutilizzino le caratteristiche per ottenere efficienza computazionale, purtroppo portano a un elevato consumo di memoria GPU. Di conseguenza, l'applicazione di DenseNet potrebbe richiedere implementazioni più efficienti in termini di memoria che potrebbero aumentare i tempi di addestramento.

$$\mathbf{x} \rightarrow [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \dots]$$





8.26 MobileNet

- ▶ È emersa una tendenza nella costruzione di reti più piccole e meno costose dal punto di vista computazionale, che potrebbero essere utilizzate in applicazioni mobili e incorporate.
- ▶ Una delle prime reti progettate per un'esecuzione più leggera è stata MobileNet (Howard, Zhu et al. 2017), che ha utilizzato convoluzioni depthwise, un caso speciale di convoluzioni raggruppate in cui il numero di gruppi è uguale al numero di canali.
- ▶ Sono emerse diverse altre varianti con l'obiettivo di costruire reti più piccole ed efficienti.
- ▶ Un'altra tendenza è la compressione/distillazione delle reti che cerca di ridurre le dimensioni di una rete addestrata per ridurre la complessità.

8.27 Migliorare l'Efficienza

- ▶ Poiché le reti neurali profonde possono essere così intensive in termini di memoria e calcolo, un certo numero di ricercatori ha indagato metodi per ridurre entrambi, utilizzando aritmetica a precisione inferiore (ad esempio, fixed-point) e compressione dei pesi.
- ▶ Il paper XNOR-Net di Rastegari, Ordonez et al. (2016) indaga sull'uso di pesi binari (connessioni on-off) e optionalmente attivazioni binarie.
- ▶ Una delle tendenze più recenti nel design delle reti neurali è l'uso di algoritmi di Neural Architecture Search (NAS) per provare diverse topologie e parametrizzazioni delle reti.
- ▶ Questo processo è più efficiente (in termini di tempo del ricercatore) rispetto all'approccio trial-and-error che ha caratterizzato la progettazione delle reti precedenti.
- ▶ Pubblicazioni più recenti includono FBNet (Wu, Dai et al. 2019), RandomNets (Xie, Kirillov et al. 2019), EfficientNet (Tan e Le 2019), RegNet (Radosavovic, Kosaraju et al. 2020), FBNetV2 (Wan, Dai et al. 2020) ed EfficientNetV2 (Tan e Le 2021).

8.28 Uso di Modelli Pre-addestrati – Fine Tuning

- ▶ Possiamo utilizzare reti neurali pre-addestrate per le nostre applicazioni.
- ▶ Tuttavia, è comune affinare tali reti su dati più caratteristici dell'applicazione.
- ▶ Con il fine-tuning, una rete addestrata su un dataset viene riaddestrata su un altro dataset. I pesi trovati precedentemente vengono utilizzati come inizializzazione. L'addestramento da modelli pre-addestrati è generalmente più veloce e richiede anche meno dati.
- ▶ Tuttavia, spesso l'ultimo strato (o gli ultimi strati), chiamato head, deve essere sostituito per adattarsi al diverso problema da risolvere (ad esempio, il numero di classi). La parte rimanente della rete non viene modificata (in termini di struttura) ed è chiamata backbone.
- ▶ Il fine-tuning di alcuni strati del backbone è anche un'opzione, ma richiede sufficienti dati e un tasso di apprendimento più lento in modo che i benefici del pre-addestramento non vadano persi.

8.29 Manifold Learning

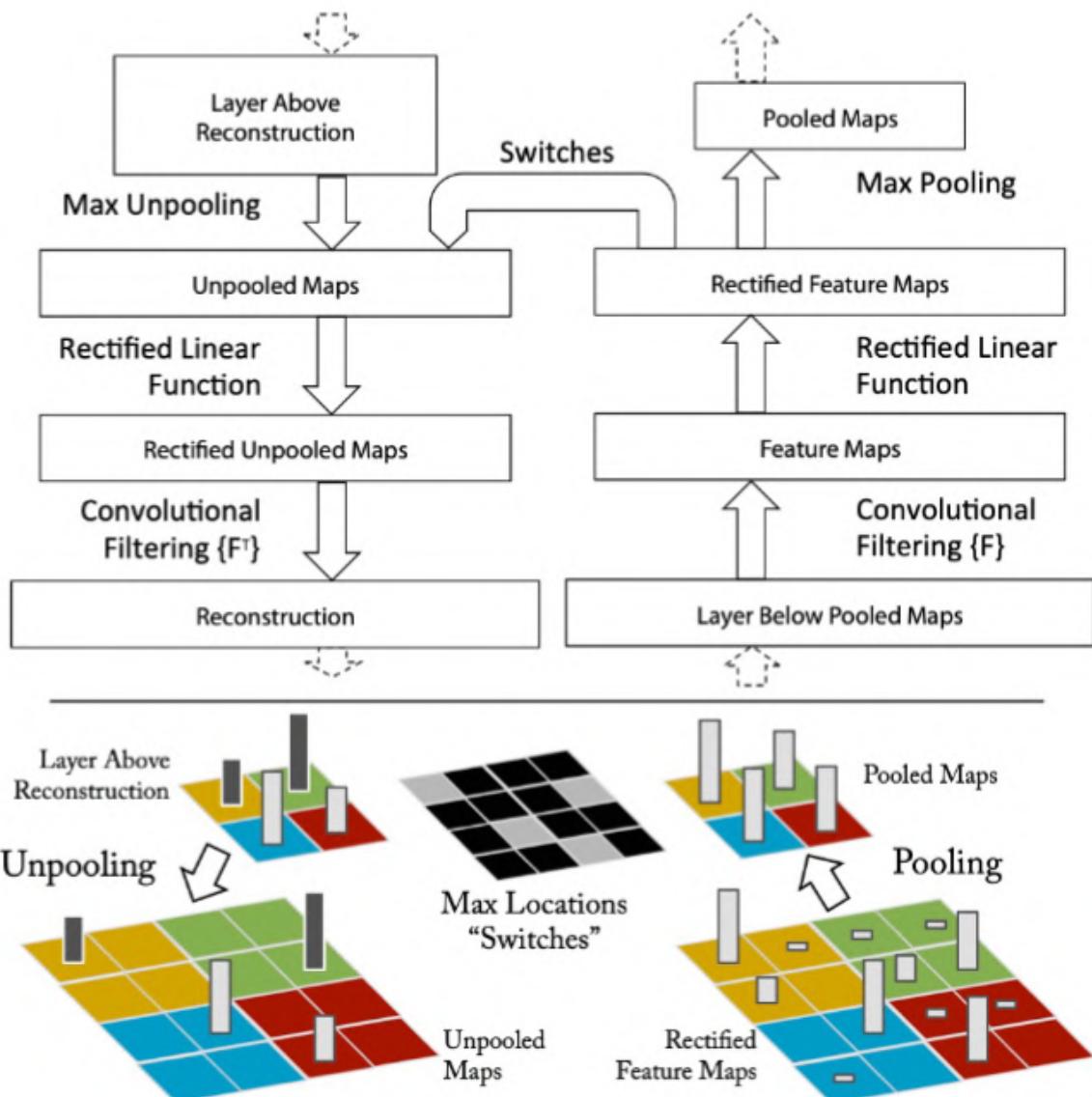
- ▶ Le tecniche di riduzione della dimensionalità non lineare, note anche come tecniche di manifold learning, possono essere utilizzate per estrarre rappresentazioni a bassa dimensionalità in uno spazio ad alta dimensionalità. Questo è utile nella visualizzazione dei dati.
- ▶ Un gran numero di algoritmi è stato sviluppato, tra cui: multidimensional scaling, Local Linear Embedding, t-distributed Stochastic Neighbor Embedding (t-SNE) e UMAP.
- ▶ Il t-SNE modella ciascun campione ad alta dimensione come un punto bidimensionale o tridimensionale in due passaggi:
 - Innanzitutto, costruisce una distribuzione di probabilità sulle coppie di campioni ad alta dimensione: gli oggetti simili sono assegnati a una probabilità più alta mentre i punti dissimili sono assegnati a una probabilità più bassa.
 - In secondo luogo, definisce una distribuzione di probabilità sulle coppie di proiezioni a bassa dimensione e minimizza la divergenza di Kullback–Leibler (divergenza KL) tra le due distribuzioni.
- ▶ L'approssimazione e proiezione uniforme del manifold (UMAP) è simile al t-SNE, ma assume che i dati siano distribuiti uniformemente su un manifold di Riemann localmente connesso e che la metrica riemanniana sia localmente costante o approssimativamente localmente costante.
- ▶ Oltre alla riduzione della dimensionalità, che può essere utile per regolarizzare i dati e accelerare la ricerca di similarità, gli algoritmi di manifold learning possono essere utilizzati per visualizzare le distribuzioni dei dati di input o le attivazioni degli strati della rete neurale.

8.30 Visualizzazione di Pesi e Attivazioni

- ▶ Visualizzare i pesi della rete e le funzioni di risposta delle diverse unità è un modo per sviluppare intuizioni e fare debugging o affinare i risultati. Può essere fatto utilizzando t-SNE o UMAP.
- ▶ Come possiamo visualizzare a cosa rispondono le singole unità ("neuroni") in una rete profonda?
- ▶ Per il primo strato in una rete, la risposta può essere letta direttamente dai pesi per un dato canale.
- ▶ Possiamo anche trovare le patch nel set di validazione che producono le risposte più grandi nelle unità in un dato canale.
- ▶ Per gli strati più profondi in una rete, possiamo trovare patch massimamente attivanti nelle immagini di input. Una volta trovate queste, Zeiler e Fergus (2014) accoppiano una rete di deconvoluzione con la rete originale per retropropagare le attivazioni delle caratteristiche fino alla patch dell'immagine.

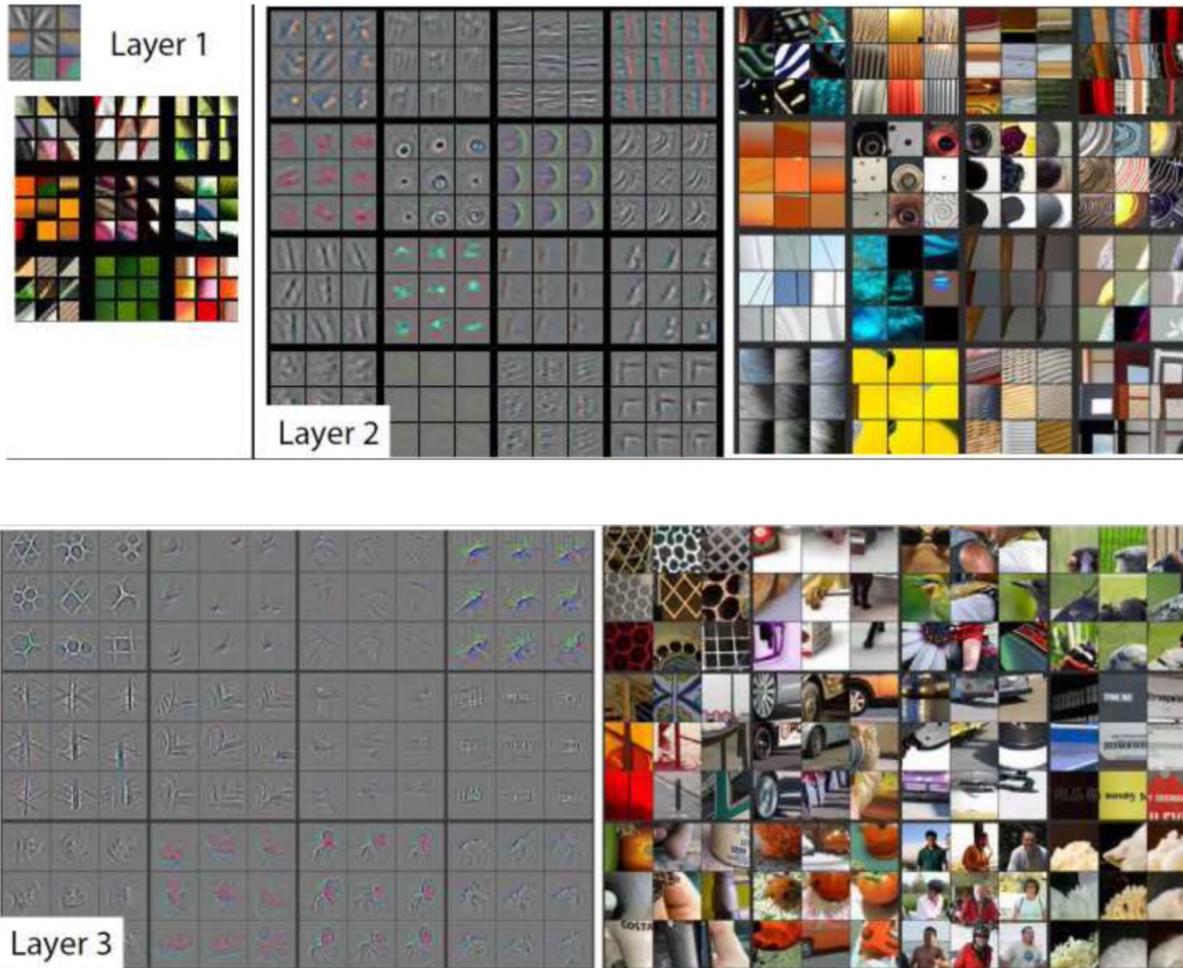
8.31 Zeiler e Fergus (2014)

- ▶ Per esaminare una convnet, una deconvnet è collegata a ciascuno dei suoi strati.
- ▶ Un'immagine di input viene presentata alla convnet e le caratteristiche vengono calcolate attraverso gli strati. Per esaminare un'attivazione data della convnet, impostiamo tutte le altre attivazioni nello strato a zero e passiamo le mappe delle caratteristiche come input allo strato di deconvnet collegato. Poi eseguiamo successivamente le seguenti operazioni per ricostruire l'attività nello strato:
 - Unpooling
 - Rettificazione: Per ottenere ricostruzioni valide delle caratteristiche a ciascuno strato (che dovrebbero essere positive), passiamo il segnale ricostruito attraverso una non-linearità ReLU.
 - Filtraggio: per invertire questo, la deconvnet utilizza versioni trasposte degli stessi filtri, ma applicati alle mappe rettificate. In pratica, questo significa capovolgere ogni filtro verticalmente e orizzontalmente.



8.32 Qual è l'effetto dell'apprendimento dei kernel convoluzionali?

- ▶ Le sotto-immagini 3×3 rappresentano le nove risposte migliori in una mappa delle caratteristiche (canale in un dato strato) proiettate indietro nello spazio dei pixel. Le immagini a colori sulla destra mostrano le patch di immagine più reattive dal set di validazione (da AlexNet).



- Layers close to the output of the network capture more complex concepts and features



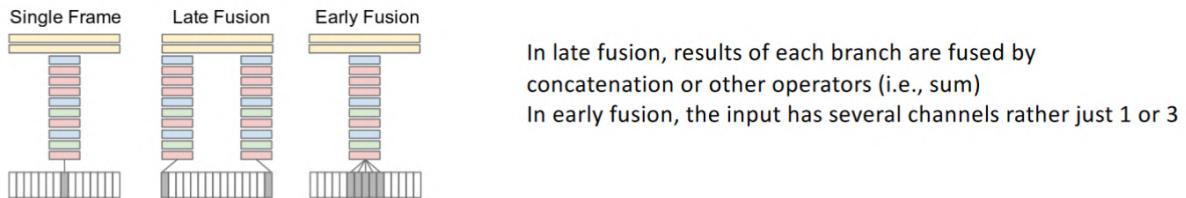
8.33 GradCam

- Selvaraju, Cogswell et al. (2017) chiamano la loro tecnica di visualizzazione mappatura di attivazione di classe ponderata dal gradiente (Grad-CAM).
- Grad-CAM utilizza i gradienti di qualsiasi concetto target (ad esempio "cane" in una rete di classificazione) che fluiscono nello strato convoluzionale finale per produrre una mappa di localizzazione grossolana che evidenzia le regioni importanti nell'immagine per prevedere il concetto.
- Calcoliamo prima il gradiente del punteggio per la classe c rispetto alle attivazioni della mappa delle caratteristiche A_k di uno strato convoluzionale. Questi gradienti che fluiscono all'indietro vengono mediati globalmente sulla larghezza e sull'altezza per ottenere i pesi di importanza dei neuroni. Eseguiamo una combinazione ponderata delle mappe di attivazione in avanti e la seguiamo con una ReLU.
- Utilizzato su VGG-16 e ResNet.

8.34 Deep Learning per Volumi e Video

- Le reti neurali profonde sono ampiamente utilizzate anche per dati 3D come immagini mediche e sequenze video. Una delle prime applicazioni è stata il riconoscimento delle azioni umane.
- Karpathy, Toderici et al. (2014) descrivono una serie di architetture alternative per fondere informazioni temporali.
- L'approccio a singolo frame classifica ciascun frame indipendentemente, basandosi esclusivamente sul contenuto di quel frame.
- La fusione tardiva prende le caratteristiche generate da ciascun frame e produce una classificazione per clip.

- ▶ La fusione precoce raggruppa piccoli set di frame adiacenti in più canali in una CNN 2D. Le interazioni attraverso il tempo non hanno gli aspetti convoluzionali della condivisione del peso e dell'invarianza allo spostamento temporale.
- ▶ Nella fusione tardiva, i risultati di ciascun ramo vengono fusi per concatenazione o altri operatori (ad es. somma).
- ▶ Nella fusione precoce, l'input ha più canali piuttosto che solo 1 o 3.



8.35 TimeDistributed in TensorFlow

- ▶ Questo wrapper consente di applicare un layer a ogni fetta temporale di un input.
- ▶ Ogni input deve essere almeno 3D e la dimensione dell'indice uno del primo input sarà considerata la dimensione temporale.
- ▶ `tf.keras.layers.TimeDistributed(layer, **kwargs)`

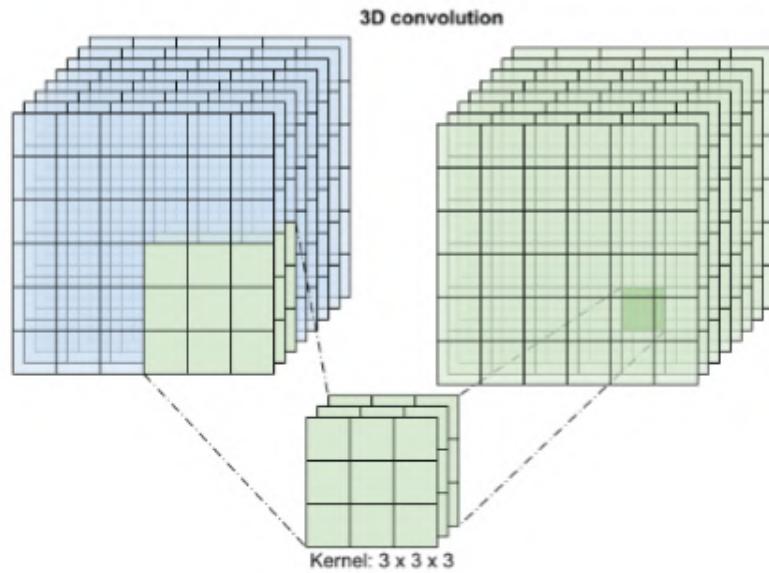
Esempio:

```
inputs = tf.keras.Input(shape=(10, 128, 128, 3))
conv_2d_layer = tf.keras.layers.Conv2D(64, (3, 3))
outputs = tf.keras.layers.TimeDistributed(conv_2d_layer)(inputs)
print(outputs.shape)
TensorShape([None, 10, 128, 128, 64])
```

- ▶ Poiché TimeDistributed applica la stessa istanza di Conv2D a ciascun timestamp, lo stesso set di pesi viene utilizzato a ciascun timestamp.

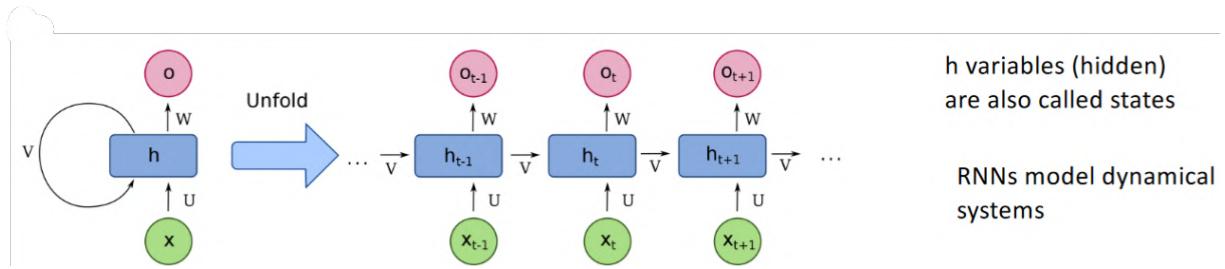
8.36 3D-Conv

- ▶ Le CNN 3D (Ji, Xu et al. 2013) apprendono kernel di spazio-tempo invarianti 3D che vengono eseguiti su finestre spaziotemporali e fusi in un punteggio finale.
- ▶ Le architetture CNN 3D possono sfruttare diverse risoluzioni spaziali e temporali, stride e profondità dei canali, ma possono essere molto intensive in termini di calcolo e memoria.
- ▶ Una CNN 3D utilizza un filtro tridimensionale per eseguire convoluzioni. Il kernel può scorrere in tre direzioni, mentre in una CNN 2D può scorrere in due dimensioni.



8.37 Reti Neurali Ricorrenti (RNN)

- ▶ Sebbene le reti convoluzionali 2D e 3D siano adatte per immagini e volumi, a volte desideriamo elaborare una serie temporale.
- ▶ Un buon modo per sfruttare le informazioni viste in precedenza è passare le caratteristiche rilevate in un istante temporale (ad esempio, un frame video) come input alla successiva elaborazione del frame.
- ▶ Tali architetture sono chiamate Reti Neurali Ricorrenti (RNN).
- ▶ Le RNN funzionano come una "memoria" che memorizza informazioni dai precedenti input per influenzare l'output corrente. Pertanto, l'output delle RNN dipende dagli elementi precedenti all'interno della sequenza.
- ▶ L'addestramento delle RNN è difficile.
- ▶ La retropropagazione richiede il calcolo delle derivate per tutte le unità "srotolate" (istanze temporali) e la somma di queste derivate per ottenere aggiornamenti dei pesi.
- ▶ Poiché i gradienti possono propagarsi per una lunga distanza all'indietro nel tempo, e quindi possono svanire o esplodere (proprio come nelle reti profonde prima dell'avvento delle reti residue), è anche possibile aggiungere unità di gating extra per modulare il flusso delle informazioni tra i frame.
- ▶ Tali architetture sono chiamate Gated Recurrent Units (GRU) e Long Short-Term Memory (LSTM).
- ▶ Le RNN e le LSTM sono spesso utilizzate per l'elaborazione video, poiché possono fondere informazioni nel tempo e modellare le dipendenze temporali.
- ▶ Per propagare le informazioni in avanti nel tempo, le RNN, GRU e LSTM devono codificare tutte le informazioni precedentemente utili nello stato nascosto che viene passato tra i passi temporali.

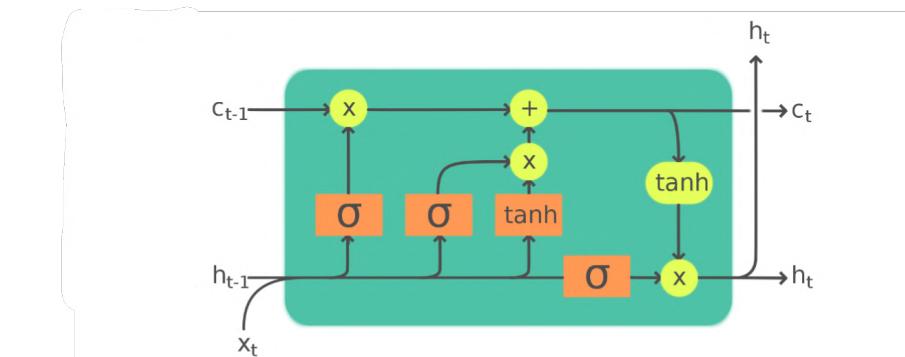


8.38 Long Short-Term Memory (LSTM)

- ▶ Un compito mostra dipendenze a lungo termine se il calcolo dell'output desiderato al tempo t dipende da un input presentato in un tempo precedente $k \ll t$.
- ▶ L'architettura LSTM mira a fornire una memoria a breve termine per le RNN che può durare migliaia di passi temporali, quindi "memoria a breve termine lunga".
- ▶ Un'unità LSTM comune è composta da una cella, una porta di ingresso, una porta di uscita e una porta di cancellazione.
- ▶ La cella memorizza valori su intervalli di tempo arbitrari. Le porte regolano il flusso di informazioni dentro e fuori la cella.
- ▶ Le porte di cancellazione decidono quali informazioni scartare da uno stato precedente assegnando a uno stato precedente un valore compreso tra 0 (scarta) e 1 (mantieni).
- ▶ Le porte di ingresso decidono quali pezzi di nuove informazioni memorizzare nello stato corrente.
- ▶ Le porte di uscita controllano quali pezzi di informazioni nello stato corrente emettere assegnando un valore da 0 a 1 alle informazioni.

8.39 Memoria a lungo breve termine (LSTM)

La *Long Short-Term Memory* (LSTM) è un tipo di *rete neurale ricorrente* (RNN) progettata per affrontare il problema del *vanishing gradient* presente nelle RNN tradizionali. Un'unità LSTM è composta da una cella, una porta di ingresso, una porta di uscita e una porta di dimenticanza. La cella conserva valori per intervalli di tempo arbitrari e le tre porte regolano il flusso di informazioni dentro e fuori dalla cella. Le equazioni per il passaggio in avanti di una cella LSTM con una porta di dimenticanza



sono:

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \sigma_h(c_t) \end{aligned}$$

dove $c_0 = 0$ e $h_0 = 0$, e l'operatore \odot denota il prodotto di Hadamard (prodotto elemento per elemento).

Variabili

Facendo riferimento ai pedici d e h per indicare rispettivamente il numero di caratteristiche di input e il numero di unità nascoste:

- ▶ $x_t \in \mathbb{R}^d$: vettore di input per l'unità LSTM
- ▶ $f_t \in (0, 1)^h$: vettore di attivazione della porta di dimenticanza
- ▶ $i_t \in (0, 1)^h$: vettore di attivazione della porta di input/aggiornamento
- ▶ $o_t \in (0, 1)^h$: vettore di attivazione della porta di output
- ▶ $h_t \in (-1, 1)^h$: vettore di stato nascosto, noto anche come vettore di output dell'unità LSTM
- ▶ $\tilde{c}_t \in (-1, 1)^h$: vettore di attivazione dell'input della cella
- ▶ $c_t \in \mathbb{R}^h$: vettore di stato della cella
- ▶ $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ e $b \in \mathbb{R}^h$: matrici di pesi e parametri del vettore di bias, che devono essere appresi durante l'addestramento

Funzioni di attivazione

- ▶ σ_g : funzione sigmoide.
- ▶ σ_c : funzione tangente iperbolica.
- ▶ σ_h : funzione tangente iperbolica oppure, come suggerisce il paper Peephole LSTM, $\sigma_h(x) = x$.

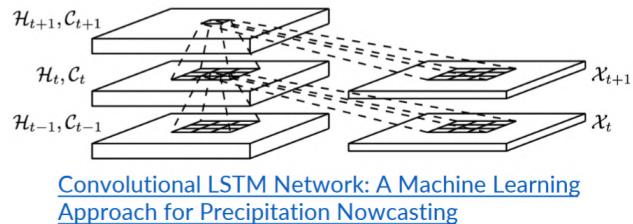
Allenamento

Le reti LSTM possono essere addestrate in modo supervisionato usando algoritmi di ottimizzazione come la discesa del gradiente combinata con il backpropagation attraverso il tempo.

ConvLSTM

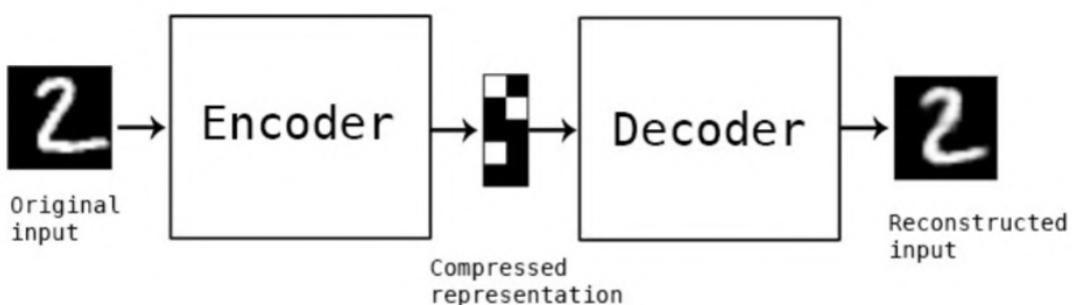
- ▶ Il ConvLSTM è un tipo di rete neurale ricorrente per la previsione spazio-temporale che ha strutture convoluzionali sia nelle transizioni da input a stato, sia da stato a stato.
- ▶ Il ConvLSTM determina lo stato futuro di una certa cella nella griglia in base agli input e agli stati passati dei suoi vicini locali. Questo può essere facilmente ottenuto utilizzando un operatore di convoluzione nelle transizioni da stato a stato e da input a stato.

$$\begin{aligned} i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot C_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot C_{t-1} + b_f) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\ o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \odot C_t + b_o) \\ H_t &= o_t \odot \tanh(C_t) \end{aligned}$$



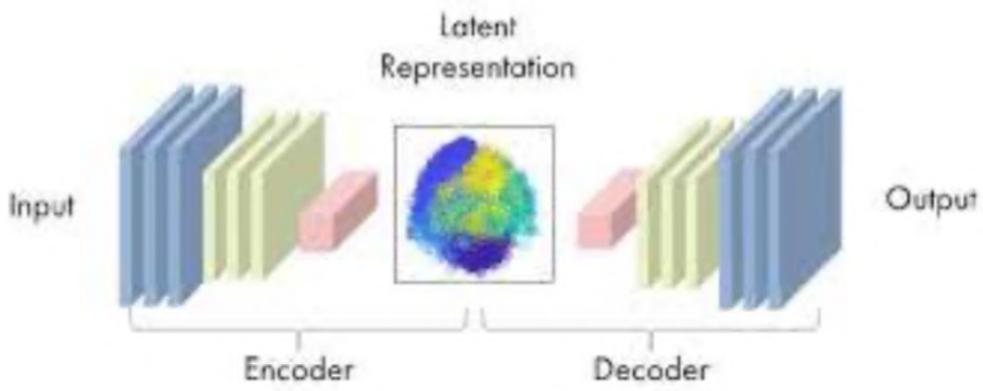
Autoencoder

- ▶ Gli autoencoder sono nati come uno strumento per l'inizializzazione dei pesi (o nell'addestramento auto-supervisionato delle reti) e poi sono stati utilizzati in diverse applicazioni, tra cui:
 - Denoising delle immagini
 - Compressione dei dati per la visualizzazione
- ▶ Negli autoencoder, l'idea principale è di ridurre la risoluzione di una rete e poi espanderla nuovamente.



- ▶ L'autoencoder può essere suddiviso in due sottoreti: l'encoder e il decoder.
- ▶ Quando l'obiettivo è la ricostruzione dell'input, l'output dell'encoder è una versione compressa dell'input e si trova in uno spazio latente.
- ▶ In alcune applicazioni, l'encoder viene poi utilizzato come estrattore di caratteristiche.

- ▶ Nella visualizzazione dei dati, prima usiamo gli autoencoder e poi il t-SNE per ridurre ulteriormente le dimensioni.



Autoencoder - Addestramento

- ▶ Gli autoencoder non sono una vera tecnica di apprendimento non supervisionato, ma una tecnica auto-supervisionata, dove gli obiettivi vengono generati dai dati di input.
- ▶ Il problema è definire una buona funzione di perdita, anziché concentrarsi sull'errore di ricostruzione pixel per pixel.
- ▶ Una possibilità è utilizzare la cross-entropia binaria per addestrare il modello. In qualche modo, lo spazio latente è simile a quello ottenuto applicando PCA.
- ▶ Un altro modo per vincolare le rappresentazioni a essere compatte è aggiungere un vincolo di scarsità sull'attività delle rappresentazioni nascoste, in modo che meno unità "si attivino" in un dato momento.

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# This is the size of our encoded representations
encoding_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# This is our input image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

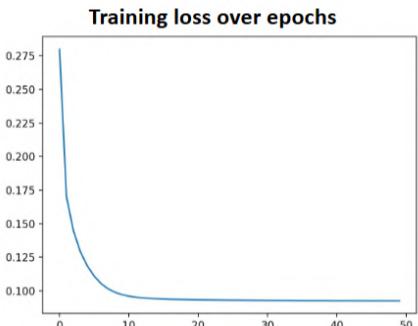
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
print(x_test.shape)

result = autoencoder.fit(x_train, x_train,
                        epochs=50,
                        batch_size=256,
                        shuffle=True,
                        validation_data=(x_test, x_test))
```

Example

```
(60000, 784)
(10000, 784)
Epoch 1/50
235/235 [=====] - 2s 5ms/step - loss: 0.2789 - val_loss: 0.1933
Epoch 2/50
235/235 [=====] - 1s 5ms/step - loss: 0.1716 - val_loss: 0.1531
...
Epoch 48/50
235/235 [=====] - 1s 6ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 49/50
235/235 [=====] - 1s 5ms/step - loss: 0.0926 - val_loss: 0.0914
Epoch 50/50
235/235 [=====] - 2s 6ms/step - loss: 0.0926 - val_loss: 0.0914
```

Example



```

decoded_imgs = autoencoder.predict(x_test)
n = 10 # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

```



Esempio: Image Denoising

```

import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import TensorBoard
import matplotlib.pyplot as plt

# DATASET
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# MODEL
input_img = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

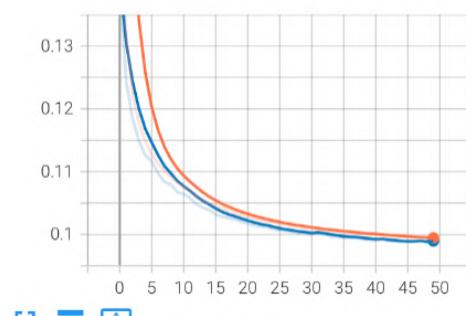
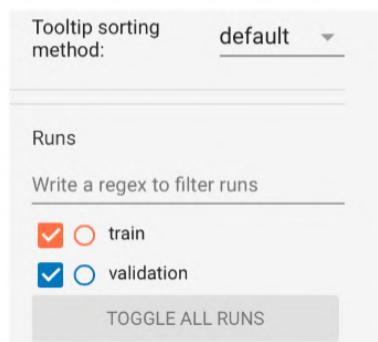
# At this point the representation is (7, 7, 16)

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

results = autoencoder.fit(x_train_noisy, x_train,
    epochs=50,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test_noisy, x_test),
    callbacks=[TensorBoard(log_dir='/tmp/autoencoder', histogram_freq=0,
    write_graph=False)])

```



```

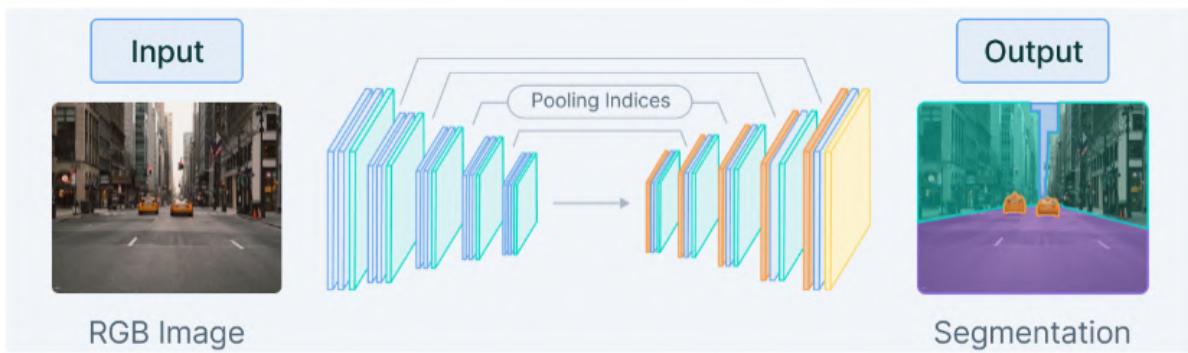
decoded_imgs = autoencoder.predict(x_test_noisy)
n = 10 # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()

```

Encoder-Decoder

- ▶ L'idea degli autoencoder è stata ampliata per affrontare altre applicazioni, tra cui:
 - Etichettatura semantica pixel per pixel
 - Inferenza della profondità monoculare
 - Trasferimento di stile neurale
- ▶ I modelli vengono addestrati con schemi adatti in cui la specificazione dell'output è estremamente importante.



U-Net

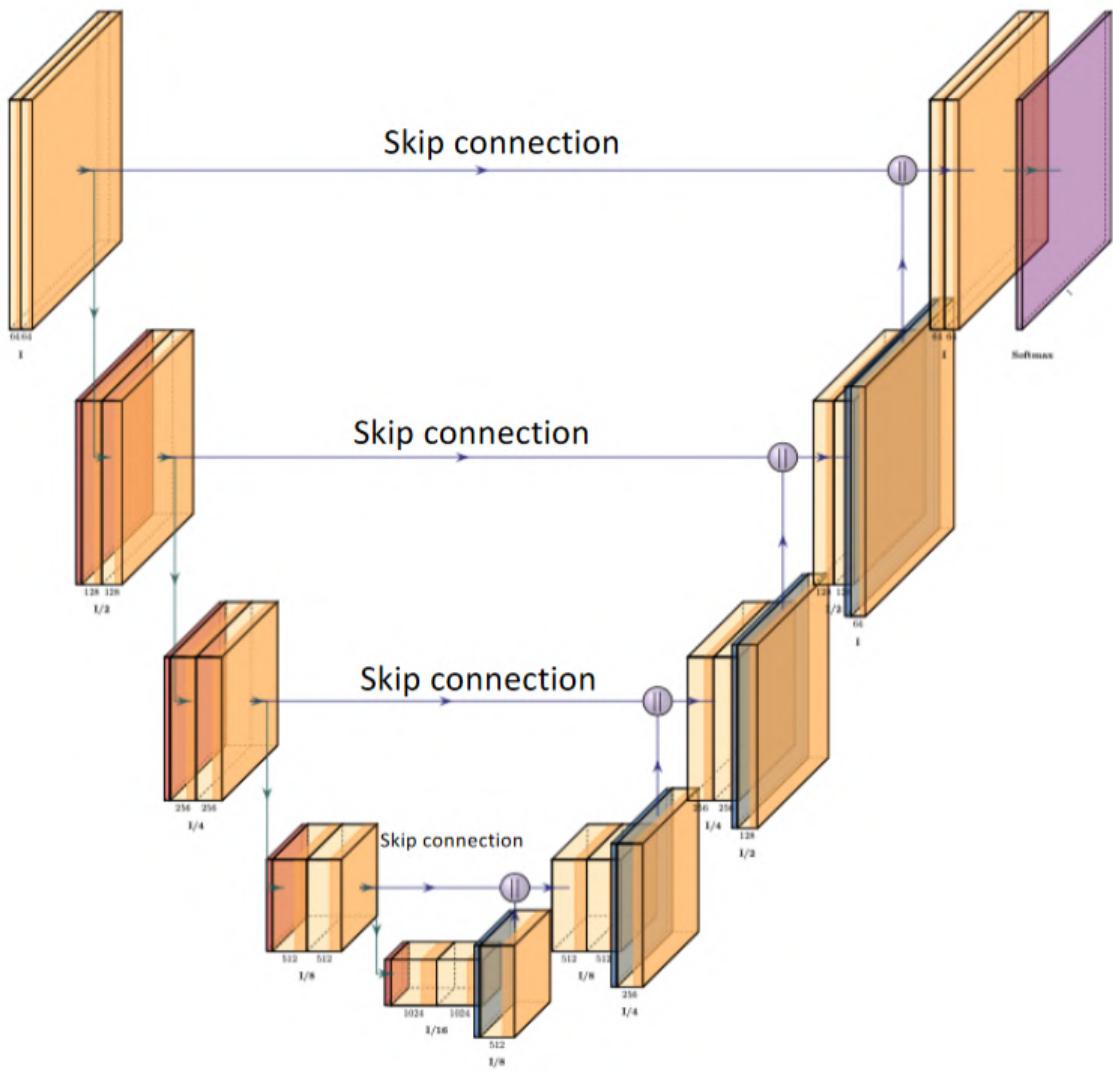
- ▶ Nelle reti encoder-decoder, il downsampling e l'upsampling delle mappe di caratteristiche possono essere combinati per raggiungere una varietà di compiti di elaborazione delle immagini a multi-risoluzione.
- ▶ Un approccio popolare (molto utilizzato nelle immagini mediche) è l'U-Net, che ha anche connessioni di skip tra i livelli di codifica e decodifica alla stessa risoluzione.

Reti Multi-Rami

- ▶ In diverse applicazioni, è necessario elaborare più input in parallelo. Questo viene fatto, ad esempio, nel riconoscimento delle azioni, dove vengono utilizzate immagini RGB e informazioni sul movimento, oppure nel tracciamento visivo, dove un modello di un oggetto e un'area di ricerca vengono analizzati.

Rete Siamese

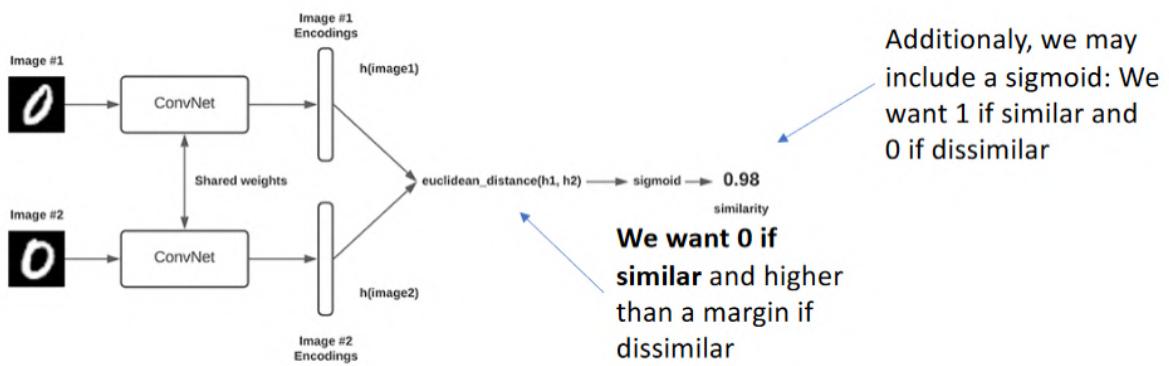
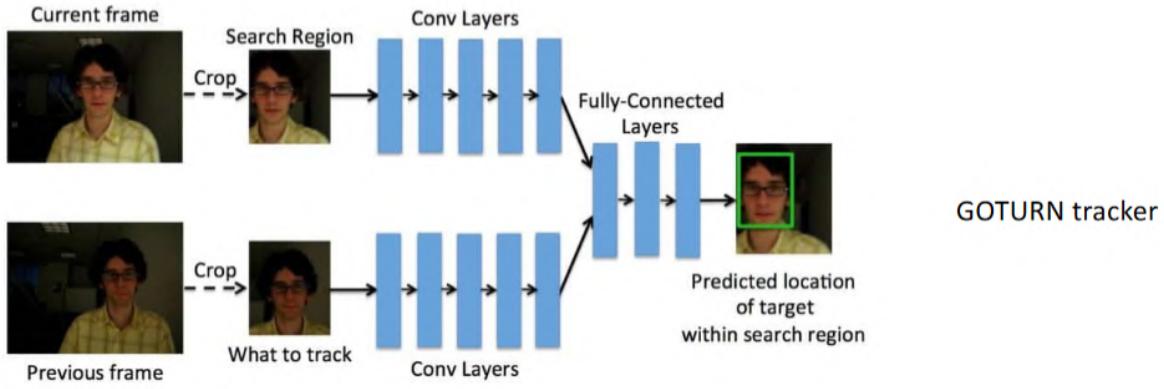
- ▶ Una rete neurale siamese (a volte chiamata rete neurale gemella) è una rete a due rami in cui i rami condividono i pesi.
- ▶ In altre parole, gli stessi pesi vengono utilizzati su due input diversi per calcolare vettori di output comparabili.



- ▶ Una rete siamese viene generalmente utilizzata per apprendere funzioni di similarità e, infatti, viene utilizzata per confrontare immagini in applicazioni di recupero (ad esempio nel riconoscimento facciale) o per la verifica delle immagini, ovvero per decidere se due immagini rappresentano lo stesso oggetto (ad esempio, la verifica facciale).

Rete Siamese - Funzioni di perdita

- ▶ L'apprendimento nelle reti gemelle può essere effettuato con triplet loss o contrastive loss.
- ▶ Nella triplet loss, un campione A viene confrontato con un campione positivo P (immagine veritiera) e un campione negativo N (immagine falsa).
- ▶ Il campione negativo forzerà l'apprendimento nella rete, mentre il vettore positivo agirà come un regolarizzatore.
- ▶ Ad esempio, se vogliamo addestrare un modello per classificare un'immagine come faccia o non faccia, potremmo utilizzare campioni come i seguenti.



- ▶ Nella contrastive loss, vengono forniti coppie di immagini (x_i, x_j) . Per ciascuna coppia, vogliamo un valore 0 se le immagini sono simili e 1 se sono dissimili.
- ▶ Così, una perdita utile è:

$$L = (1 - y) \frac{1}{2} (D_w)^2 + (y) \frac{1}{2} \max(0, m - D_w)^2$$

dove D_w è la distanza tra i vettori di output e m è un iperparametro (margin), che definisce la distanza minima tra i campioni dissimili.

- ▶ Queste due funzioni di perdita sono alla base dell'apprendimento contrastivo, il cui obiettivo è far apprendere al modello nuove cose contrastando tra cose simili e dissimili.

Generatore di Dati

- ▶ Quando il dataset è enorme e non può essere caricato interamente in memoria, è più conveniente suddividerlo in diverse parti e processarle.
- ▶ Non è conveniente addestrare il modello in modo indipendente su diverse parti del dataset perché il modello può overfitare su una parte del dataset e dimenticare il resto.
- ▶ Invece, durante l'addestramento, possiamo caricare o generare dati in lotti creando batch.
- ▶ Questo può essere fatto in diversi modi, il più semplice è usare un generatore di dati Python, che sfrutta un ciclo e la parola chiave `yield`.

```

def get_branch(shape_input):
    input_img = keras.Input(shape=shape_input)
    x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    y = layers.Flatten()(x)
    model = keras.Model(input_img, y)
    return model

def get_siamese(shape_input=(28, 28, 1)):
    branch = get_branch(shape_input)
    input_img_1 = keras.Input(shape=shape_input)
    input_img_2 = keras.Input(shape=shape_input)
    des_1 = branch(input_img_1)
    des_2 = branch(input_img_2)
    output = layers.Lambda(euclidean_distance, name='output_layer')([des_1, des_2])
    model = keras.Model([input_img_1, input_img_2], output)
    model.summary()
    return model

import tensorflow.keras.backend as K
def euclidean_distance(d):
    return K.sum(K.square(d[0] - d[1]), axis=-1)

```

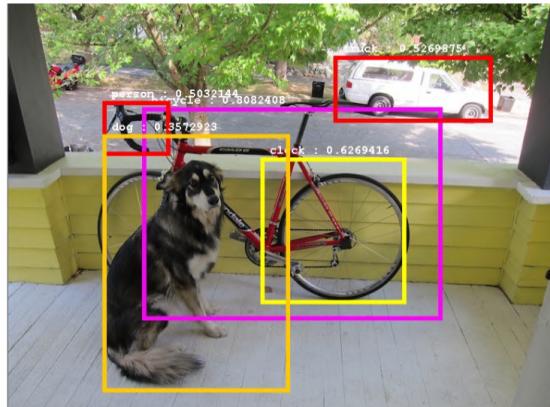
Early Stopping

- ▶ L'early stopping è una forma di regolarizzazione utilizzata per evitare l'overfitting durante l'addestramento di un modello con un metodo iterativo, come la discesa del gradiente.
- ▶ Gli algoritmi di addestramento aggiornano i parametri per migliorare l'adeguatezza del modello ai dati di addestramento con ogni iterazione.
- ▶ Fino a un certo punto, questo migliora le prestazioni del modello sui dati fuori dal set di addestramento. Oltre quel punto, migliorare l'adattamento del modello ai dati di addestramento comporta un aumento dell'errore di generalizzazione.
- ▶ Le regole di early stopping forniscono indicazioni su quante iterazioni possono essere eseguite prima che il modello inizi a sovradattarsi.
- ▶ Le regole di early stopping si basano sull'analisi dei limiti superiori dell'errore di generalizzazione in funzione del numero di iterazioni.
- ▶ L'errore sul set di validazione viene utilizzato come proxy per l'errore di generalizzazione nel determinare quando inizia l'overfitting. L'addestramento si interrompe non appena l'errore sul set di validazione è maggiore rispetto all'ultima volta che è stato controllato e si utilizzano i pesi che la rete aveva in quel passaggio precedente come risultato dell'addestramento.

OBJECT DETECTION VIA DEEP LEARNING

Object Detection via Deep Learning

9.1 Rilevamento degli Oggetti



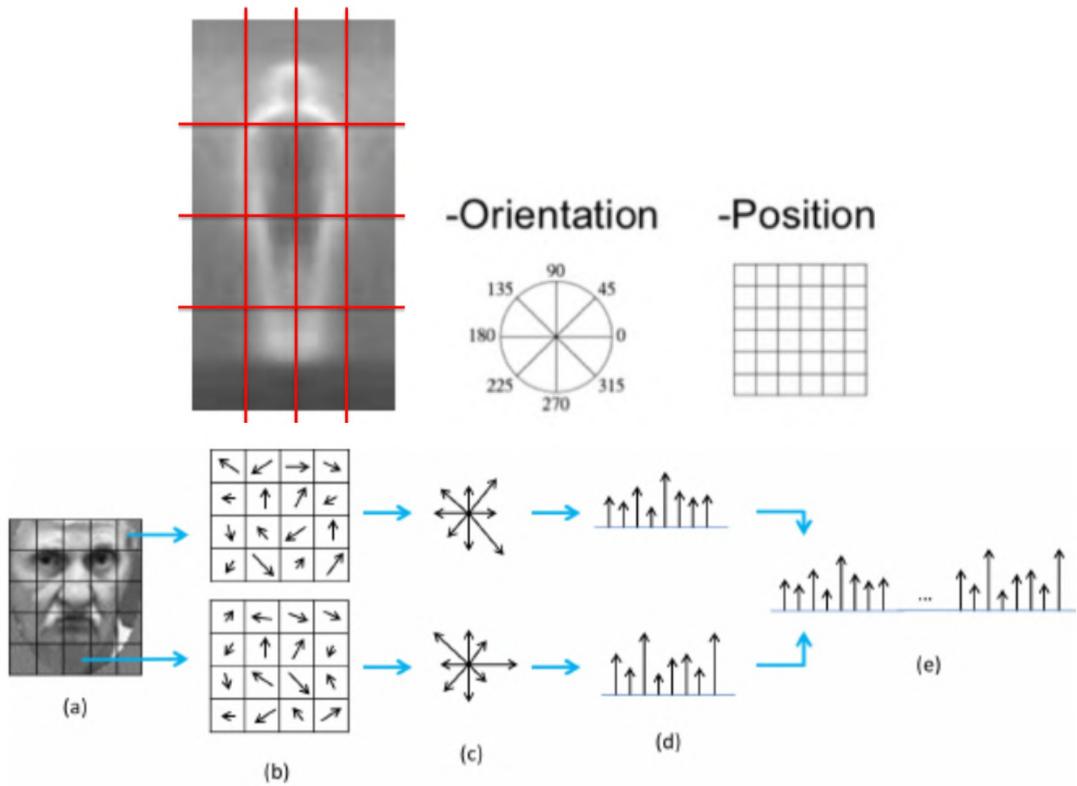
Where the object is
What the object is
Confidence

Rilevamento dei Pedoni – Un Caso Speciale

Quali caratteristiche sono utili per riconoscere che questa bounding box rappresenta un essere umano in piedi?

- ▶ In media, possiamo riconoscere un essere umano in piedi basandoci sul contorno del corpo.
- ▶ Pertanto, possiamo considerare il Gradiente dell'Immagine.
- ▶ La Magnitudine del Gradiente è maggiore sui pixel dove c'è una forte variazione di intensità.
- ▶ Inoltre, il Gradiente è ortogonale al contorno e fornisce informazioni sulla forma.
- ▶ Tuttavia, calcolare un descrittore globale basato sul Gradiente su tutta l'immagine è inefficace.
- ▶ Dobbiamo concentrarci sulle proprietà locali del Gradiente.
- ▶ Dividiamo l'immagine in celle.
- ▶ Per ogni cella, calcoliamo l'istogramma dei gradienti orientati (istogramma delle orientazioni pesato dalla magnitudine).
- ▶ Gli istogrammi devono essere normalizzati.
- ▶ Tuttavia, la normalizzazione per cella può appiattire le informazioni e si perde il "contesto".
- ▶ Pertanto, eseguiamo la normalizzazione a blocchi (tramite norma $L1$ o $L2$).
- ▶ Un blocco è un gruppo di celle adiacenti.
- ▶ Gli istogrammi vengono quindi concatenati per formare il popolare descrittore HOG.
- ▶ Dotati di un set di descrittori HOG per campioni positivi e negativi, possiamo utilizzare un classificatore per decidere se le bounding box corrispondenti rappresentano o meno un pedone.

- Il classificatore più utilizzato era l'SVM.

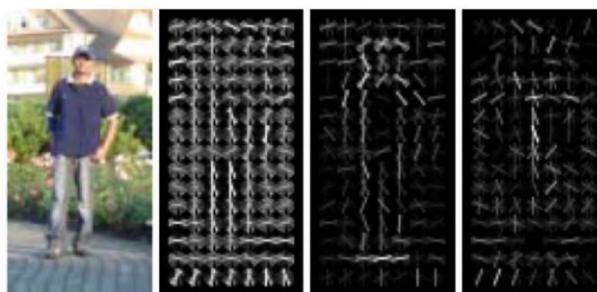


In un SVM Lineare

- Prevediamo la classe calcolando

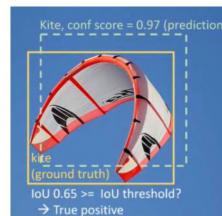
$$y = \text{sign}(w^T x + w_0)$$

- In questo esempio, ci sono un discreto numero di risposte positive attorno alla testa, al torso e ai piedi della persona, e relativamente poche risposte negative.



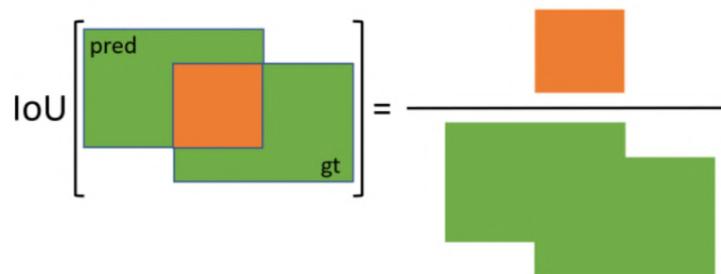
Il Mio Rilevatore è un Buon Rilevatore?

- ▶ La maggior parte degli algoritmi di rilevamento include piramide spaziale, modello deformabile a topologia a stella e SVM a finestra scorrevole.
- ▶ Tuttavia, come si misura quanto è buono un rilevatore?
- ▶ Quale dovrebbe essere il mio ground-truth?



Intersezione su Unione (IoU)

- ▶ L'IoU è in $[0, 1]$ con 1 come valore migliore!
- ▶ Conosciuto anche come indice di Jaccard o coefficiente di similarità di Jaccard.

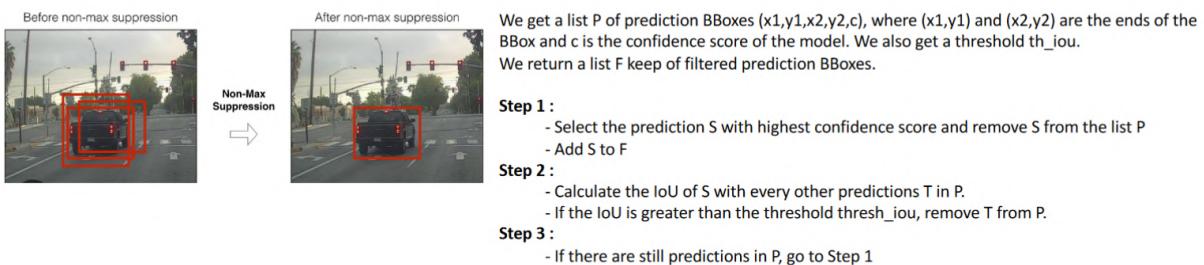


IoU is in $[0, 1]$ with 1 the best value!

$$IoU = \frac{B_{\text{pr}} \cap B_{\text{gt}}}{B_{\text{pr}} \cup B_{\text{gt}}}$$

Soppressione Non Massima

- ▶ I rilevatori di oggetti operano proponendo prima un numero di regioni rettangolari plausibili (rilevamenti) e poi classificando ogni rilevamento producendo anche un punteggio di confidenza.
- ▶ Queste regioni vengono quindi sottoposte a una fase di soppressione non massima (NMS), che rimuove i rilevamenti più deboli che hanno troppa sovrapposizione con i rilevamenti più forti, utilizzando un algoritmo avido che seleziona per primo il più sicuro.
- ▶ Selezioniamo le previsioni con la massima confidenza e sopprimiamo tutte le altre previsioni che hanno una sovrapposizione con le previsioni selezionate superiore a una soglia. In altre parole, prendiamo il massimo e sopprimiamo quelli non massimi.



Precisione - Richiamo

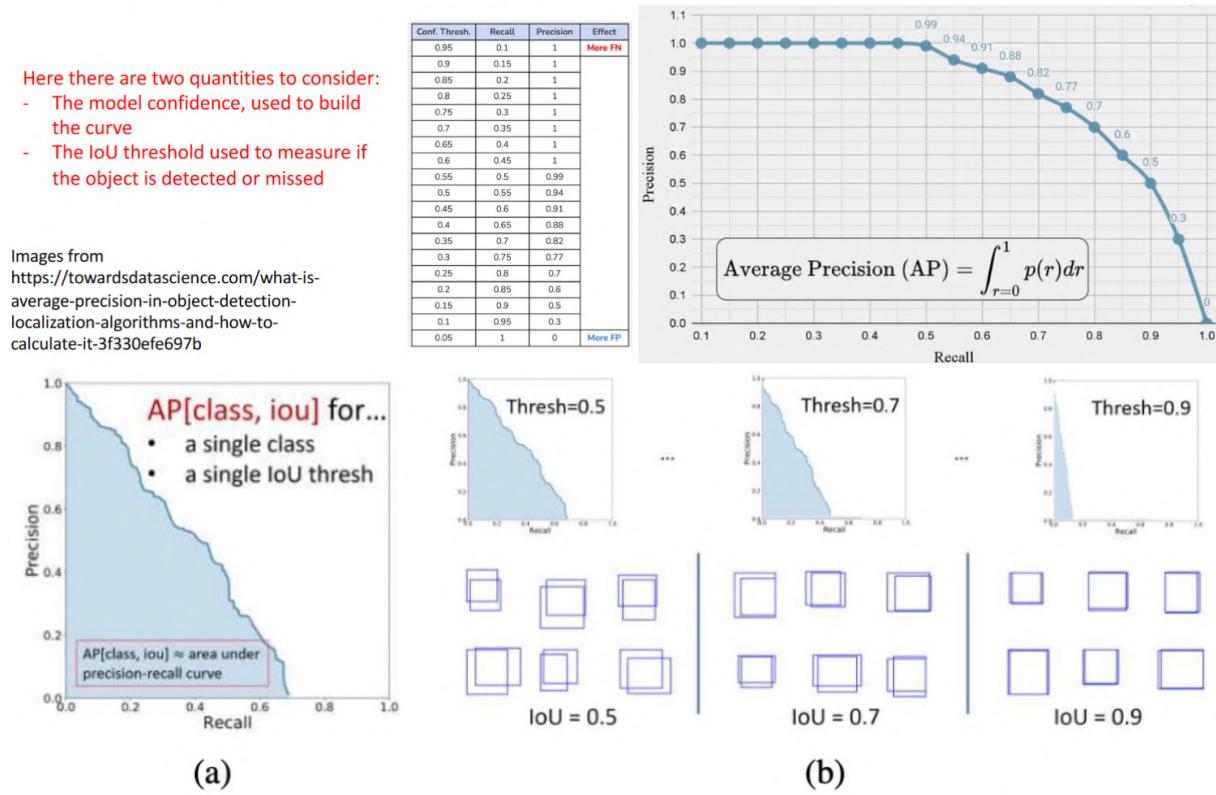
- ▶ Per valutare le prestazioni di un rilevatore di oggetti, iteriamo sui rilevamenti, dal più sicuro al meno, e li classifichiamo come veri positivi (TP) (etichetta corretta e IoU sufficientemente alto) o falsi positivi (FP) (etichetta errata o oggetto ground-truth già corrispondente).
- ▶ Per ogni nuova soglia di confidenza decrescente, possiamo calcolare la precisione e il richiamo, dove P è il numero di esempi positivi, cioè il numero di rilevamenti ground-truth etichettati nell'immagine di test.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \xleftarrow{\text{Classified as positive}}$$

$$\text{Accuracy} \xrightarrow{\hspace{1cm}} \text{recall} = \frac{\text{TP}}{P},$$

Curva Precisione-Richiamo e AP

- ▶ Calcolare la precisione e il richiamo a ogni soglia di confidenza ci permette di calcolare una curva precisione-richiamo.
- ▶ L'area sotto questa curva è chiamata precisione media (AP).
- ▶ Un punteggio AP separato può essere calcolato per ciascuna classe rilevata e i risultati mediati per produrre una precisione media media (mAP).

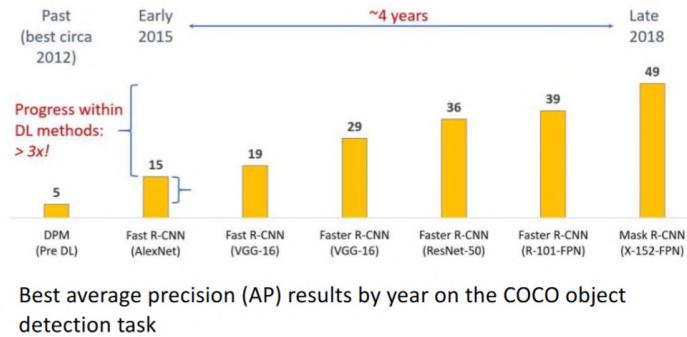


In some benchmark, a threshold on the IoU of 0.5 is used

In other benchmarks, multiple thresholds on the IoU are used and results are averaged

Rilevatori di Oggetti Moderni

- ▶ I rilevatori di oggetti moderni sono basati sul deep learning e hanno subito diverse trasformazioni nel tempo.
- ▶ I più famosi sono:
 - R-CNN
 - Fast R-CNN
 - Faster R-CNN
 - YOLO



Proposta di Regione

- ▶ La prima fase nel rilevamento degli oggetti in un'immagine è proporre un insieme di regioni rettangolari plausibili in cui eseguire un classificatore. Lo sviluppo di tali algoritmi di proposta di regione è stato un'area di ricerca attiva nei primi anni 2000.

► Algoritmo di Ricerca Selettiva:

- Invece di eseguire una ricerca esaustiva all'interno dell'immagine (cioè il metodo della finestra scorrevole), l'algoritmo di ricerca selettiva (Uijlings, 2013) sfrutta la struttura dell'immagine per generare possibili posizioni degli oggetti. Il metodo utilizza un raggruppamento gerarchico dei pixel per gestire tutte le possibili scale degli oggetti. Le regioni vengono raggruppate in base a colore, trama, dimensione e/o una misura di "insideness".

Selective Search in OpenCV

```
# initialize OpenCV's selective search implementation
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(image)
ss.switchToSelectiveSearchQuality() # or ss.switchToSelectiveSearchFast()
# , which is faster but less accurate
rects = ss.process()

for i in range(0, len(rects), 100):
    # clone the original image so we can draw on it
    output = image.copy()
    # loop over the current subset of region proposals
    for (x, y, w, h) in rects[i:i + 100]:
        # draw the region proposal bounding box on the image
        color = [random.randint(0, 255) for j in range(0, 3)]
        cv2.rectangle(output, (x, y), (x + w, y + h), color, 2)
    # show the output image
    cv2.imshow("Output", output)
    key = cv2.waitKey(0) & 0xFF
    # if the 'q' key was pressed, break from the loop
    if key == ord("q"):
        break
```

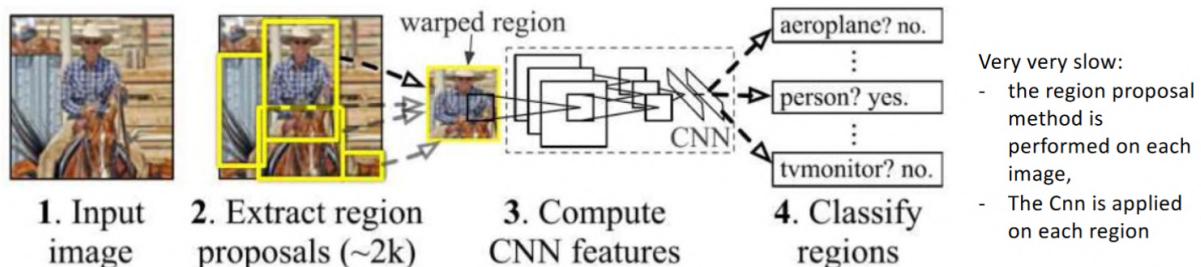
Use opencv-contrib-python
(uninstall opencv first)



Only 200 of the 11K regions found

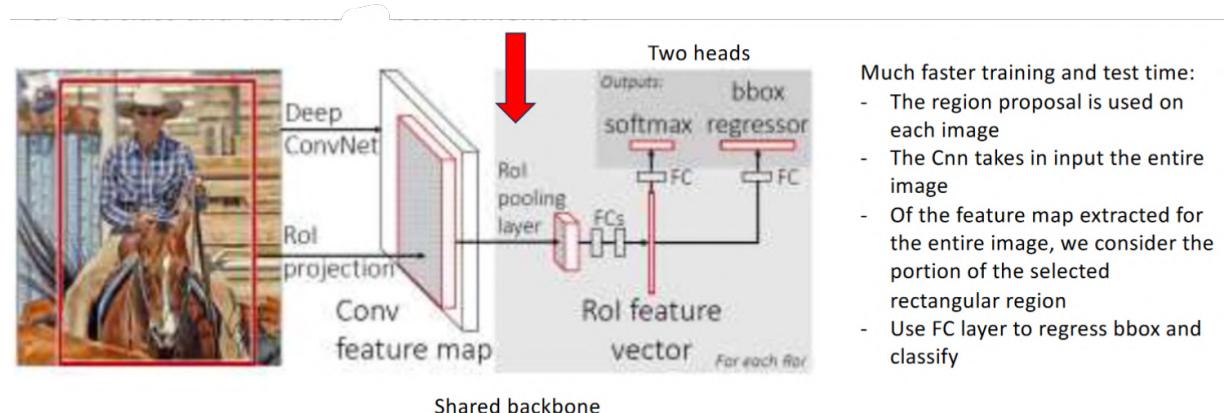
R-CNN Basato su Regione (R-CNN) 2014

- Uno dei primi rilevatori di oggetti basati su reti neurali è R-CNN, la Rete Convolutazionale Basata su Regione sviluppata da Girshick, Donahue et al. (2014).
- Questo rilevatore inizia estraendo circa 2.000 proposte di regione utilizzando l'algoritmo di ricerca selettiva. Ogni regione proposta viene quindi riscalata (deformata) in un'immagine quadrata 224×224 e passata attraverso una rete neurale AlexNet o VGG (per estrarre caratteristiche profonde) con un classificatore finale a macchina vettoriale di supporto (SVM).



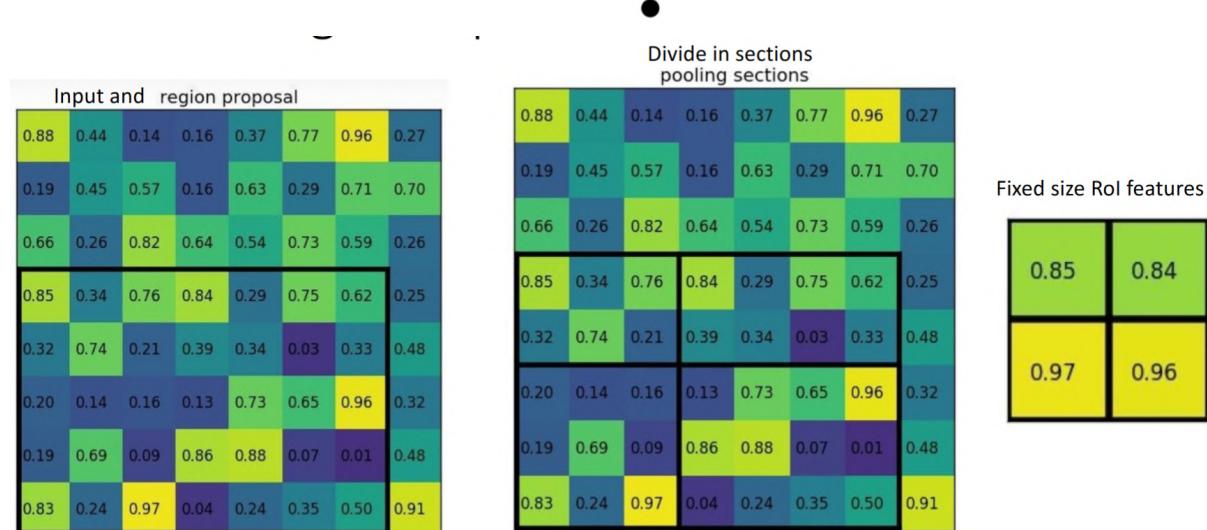
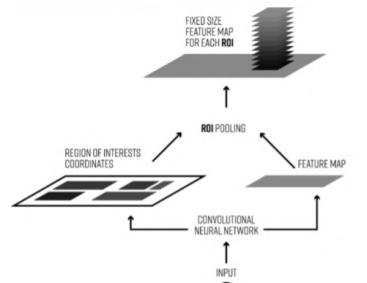
R-CNN Veloce (Fast R-CNN) 2015

- Fast R-CNN (Girshick) scambia le fasi di rete neurale convoluzionale e di estrazione della regione e sostituisce l'SVM con alcuni strati completamente connessi (FC), che calcolano sia una classe di oggetti che un raffinamento del bounding box.
- Tempo di allenamento e test molto più veloce:
 - La proposta di regione viene utilizzata su ogni immagine.
 - La CNN prende in input l'intera immagine.
 - Della mappa delle caratteristiche estratta per l'intera immagine, consideriamo la porzione della regione rettangolare selezionata.
 - Utilizziamo lo strato FC per regredire il bounding box e classificare.



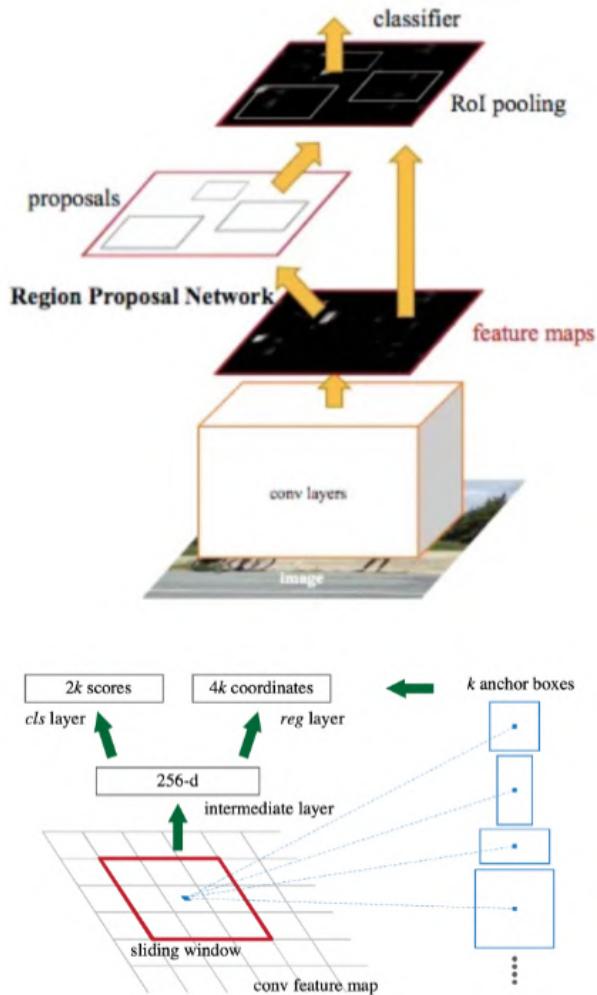
Pooling della Regione di Interesse (ROI Pooling)

- Il Region of Interest (RoI) pooling è un'operazione ampiamente utilizzata nei compiti di rilevamento degli oggetti utilizzando reti neurali convoluzionali.
- Il suo scopo è eseguire il max pooling su input di dimensioni non uniformi per ottenere mappe di caratteristiche di dimensioni fisse (ad es., 7×7). Il layer prende due input:
 - Una mappa di caratteristiche a dimensione fissa ottenuta da una rete convoluzionale profonda con diversi strati di convoluzione e max pooling.
 - Una matrice $N \times 5$ che rappresenta un elenco di regioni di interesse, dove N è il numero di ROI. La prima colonna rappresenta l'indice dell'immagine, e le rimanenti quattro sono le coordinate degli angoli in alto a sinistra e in basso a destra della regione.
- Per ogni regione di interesse dall'elenco di input, prende una sezione della mappa di caratteristiche di input che corrisponde ad essa e la ridimensiona a una dimensione predefinita (ad es., 7×7). Il ridimensionamento avviene mediante:
 - La suddivisione della proposta di regione in sezioni di dimensioni uguali (il cui numero è lo stesso della dimensione dell'output).
 - La ricerca del valore massimo in ogni sezione.
 - La copia di questi valori massimi nel buffer di output.



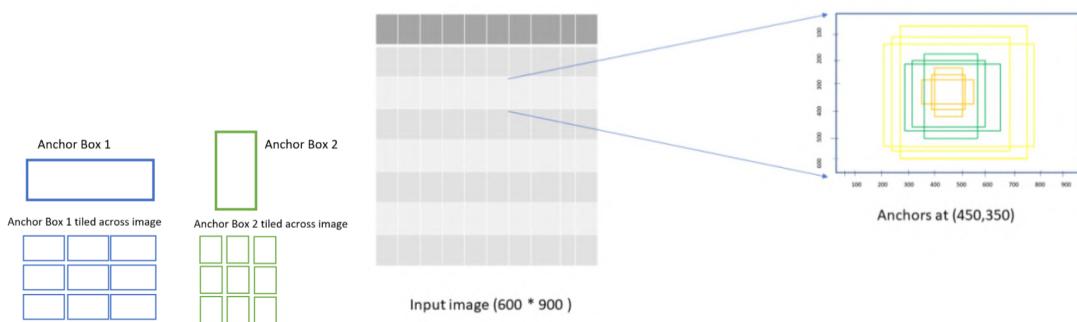
9.2 Faster R-CNN, 2015

- ▶ Il sistema Faster R-CNN sostituisce la fase relativamente lenta di ricerca selettiva con una rete di proposte di regioni convoluzionali (RPN), risultando in un'inferenza molto più rapida.
- ▶ Dopo aver calcolato le caratteristiche convoluzionali, l'RPN suggerisce in ogni posizione grezza un certo numero di potenziali finestre di ancoraggio (anchor boxes), che variano in forma e dimensione per accogliere diversi potenziali oggetti.
- ▶ Ogni proposta viene quindi classificata e raffinata da un'istanza delle testate Fast R-CNN, e le rilevazioni finali vengono classificate e unite utilizzando la soppressione non massima.



9.3 Ancoraggi e la loro Rappresentazione

- ▶ Una finestra di ancoraggio è una finestra che circonda gli oggetti (chiamata anche finestra di delimitazione).
- ▶ Questa finestra ha una certa altezza e larghezza, selezionata in base alle altezze e larghezze degli oggetti in un set di dati di addestramento.
- ▶ Le finestre di ancoraggio servono come riferimenti a più scale e rapporti d'aspetto.
- ▶ Posizioneremo le finestre di ancoraggio per ciascuna posizione della mappa di caratteristiche convoluzionale rispetto all'immagine di input. Restituisce finestre di ancoraggio affiancate sull'intera immagine.
- ▶ Ogni finestra di ancoraggio produce una previsione se contiene o meno il nostro oggetto desiderato. La previsione è sotto forma di classificazione binaria.
- ▶ In generale, prendiamo finestre di ancoraggio di dimensioni 64×64 , 128×128 e 256×256 .
- ▶ Si dovrebbero considerare dimensioni che rappresentano da vicino la scala e il rapporto d'aspetto degli oggetti nei propri dati di addestramento. Statisticamente, si può calcolare il K-means per raggruppare finestre di ancoraggio simili.
- ▶ I tre colori rappresentano tre scale o dimensioni: 128×128 , 256×256 , 512×512 .
- ▶ Per ciascun colore (scala), le tre finestre hanno rapporti altezza-larghezza di $1 : 1$, $1 : 2$, e $2 : 1$ rispettivamente.
- ▶ Abbiamo 9 finestre di ancoraggio per ciascuna posizione della mappa di caratteristiche. Tuttavia, molte finestre potrebbero non contenere alcun oggetto, quindi il modello deve imparare quale finestra di ancoraggio potrebbe contenere il nostro oggetto.
- ▶ Gli sviluppatori hanno scelto 3 scale e 3 rapporti d'aspetto, risultando in un totale di 9 proposte per ciascun pixel.
- ▶ Per l'intera immagine, il numero di ancoraggi è $W \times H \times K$.



9.4 RPN (Rete di Proposte di Regioni)

- ▶ L'obiettivo della RPN è classificare ciascuna finestra di ancoraggio per prevedere se potrebbe contenere un oggetto o meno.
- ▶ Per addestrare la RPN, è necessario preparare un set di dati in cui, per ciascuna finestra di ancoraggio precomputata, basata sulla metrica di distanza Intersection over Union (IoU) della finestra di ancoraggio rispetto alla finestra di delimitazione reale dell'oggetto, si assegna 1 alle finestre che sovrappongono gli oggetti e 0 altrimenti.
- ▶ In particolare, se una finestra di ancoraggio ha una IoU superiore a 0,5, verrà considerata come primo piano (foreground), mentre quelle con una IoU inferiore a 0,1 vengono considerate come sfondo (background).
- ▶ La RPN è una rete completamente convoluzionale che prevede contemporaneamente i limiti dell'oggetto e i punteggi di probabilità dell'oggetto in ciascuna posizione. La RPN è addestrata end-to-end.
- ▶ La RPN ha un classificatore e un regressore.
- ▶ L'ancora è il punto centrale della finestra di scorrimento. Per il modello ZF, che era un'estensione di AlexNet, le dimensioni sono 256-d e per VGG-16, erano 512-d.
- ▶ Il classificatore determina la probabilità che una proposta contenga l'oggetto target.
- ▶ La regressione regola le coordinate delle proposte. Impara gli offset (o differenze) per i valori di x, y, w, h rispetto alla finestra di riferimento (ground truth box) per la finestra di ancoraggio che è stata classificata come primo piano, dove (x, y) è il centro della finestra, e w e h sono la larghezza e l'altezza.
- ▶ È importante, per questa rete, definire la funzione di perdita appropriata.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Where,

i – Index of anchor

p – probability of being an object or not,

t – vector of 4 parameterized coordinates of predicted bounding box

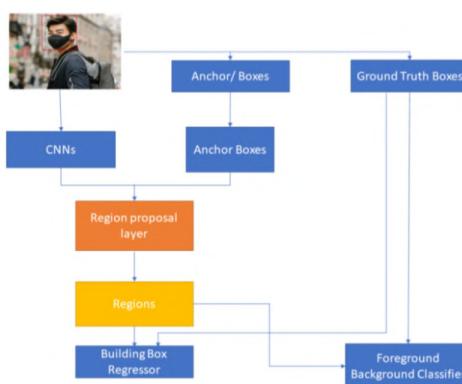
$*$ – represents ground truth box. L for cls represents Log Loss over two classes.

N_{cls} and N_{reg} – are the normalization

$\lambda = 10$ (default) and it is to scale classifier and regressor on the same level.

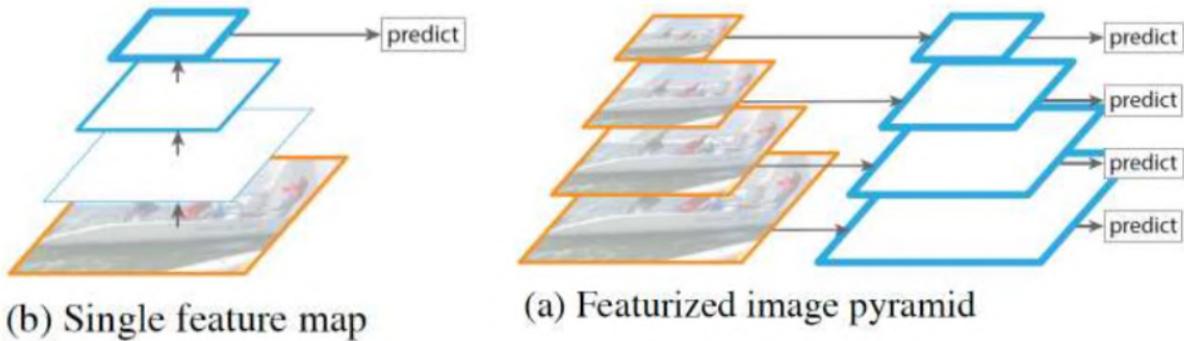
L_{cls} may be the binary cross-entropy

L_{reg} may be the smooth L1 norm (huber loss)



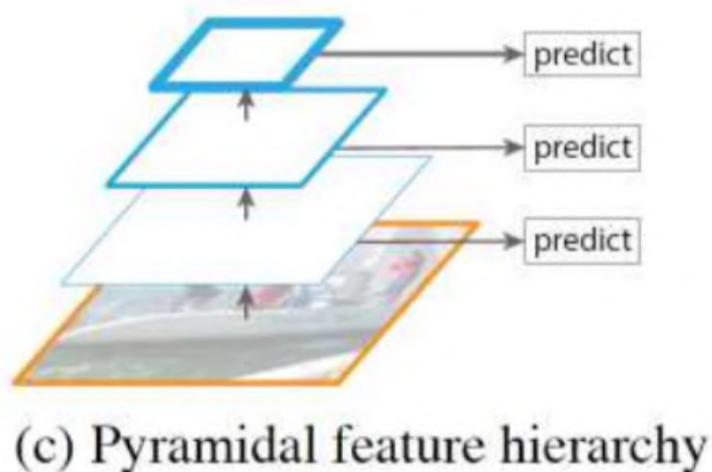
9.5 Mappa di Caratteristiche Singola vs. Più Mappe di Caratteristiche da una Piramide di Immagini

- ▶ R-CNN, Fast R-CNN e Faster R-CNN utilizzano una singola risoluzione dell'immagine.
- ▶ Per ottenere una migliore invarianza alla scala, sarebbe preferibile operare su una gamma di risoluzioni, ad es., calcolando una mappa di caratteristiche a ciascun livello della piramide di immagini, ma questo è computazionalmente costoso.



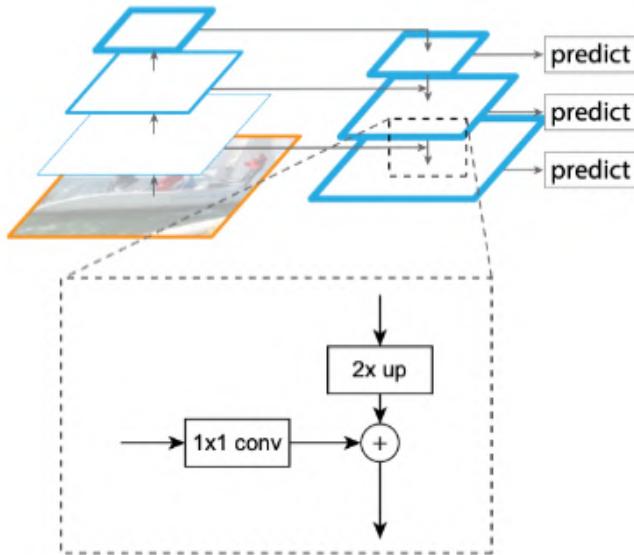
9.6 Piramide delle Mappe di Caratteristiche

- ▶ Potremmo invece semplicemente partire dai vari livelli all'interno della rete convoluzionale, ma questi livelli hanno diversi gradi di astrazione semantica, cioè i livelli più alti/piccoli sono sintonizzati su costrutti più astratti.



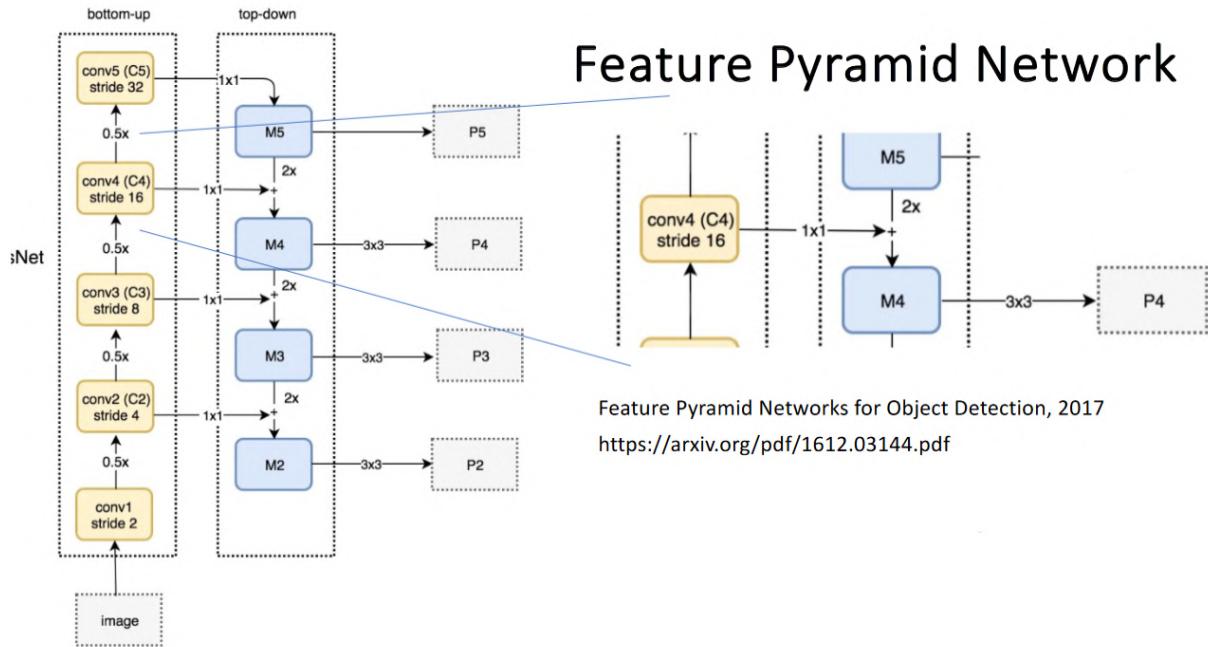
9.7 Rete a Piramide di Caratteristiche

- ▶ La soluzione migliore è costruire una Rete a Piramide di Caratteristiche (FPN) dove le connessioni dall'alto verso il basso vengono utilizzate per dotare i livelli più bassi della piramide (a risoluzione più alta) delle semantiche inferite a livelli più alti [lin2017feature].
- ▶ Queste informazioni aggiuntive migliorano significativamente le prestazioni dei rilevatori di oggetti (e altri compiti a valle) e rendono il loro comportamento molto meno sensibile alle dimensioni degli oggetti.
- ▶ La FPN sostituisce l'estrattore di caratteristiche dei rilevatori come Faster R-CNN e genera più strati di mappe di caratteristiche (mappe di caratteristiche multi-scala) con informazioni di qualità migliore rispetto alla normale piramide di caratteristiche per il rilevamento degli oggetti.
- ▶ Le connessioni laterali tra strati ricostruiti e le mappe di caratteristiche corrispondenti aiutano il rilevatore a prevedere meglio la posizione.
- ▶ Il percorso dall'alto verso il basso immagina caratteristiche a risoluzione più alta mediante upsampling di mappe di caratteristiche spazialmente più grossolane, ma semanticamente più forti, dai livelli più alti della piramide. Queste caratteristiche vengono poi migliorate con caratteristiche provenienti dal percorso dal basso verso l'alto tramite connessioni laterali. Ogni connessione laterale unisce mappe di caratteristiche della stessa dimensione spaziale dal percorso dal basso verso l'alto e dal percorso dall'alto verso il basso. La mappa di caratteristiche del percorso dal basso verso l'alto ha una semantica di livello inferiore, ma le sue attivazioni sono più accuratamente localizzate poiché è stata sottocampionata un numero inferiore di volte.



TensorflowHub

- ▶ TensorFlow Hub fornisce modelli di machine learning pre-addestrati e pronti per essere utilizzati per vari scopi. Si consiglia di scaricare e testare alcuni dei modelli disponibili!



- Ambiti di applicazione del testo:
 - * Embedding
 - * Modelli di linguaggio
 - * Preprocessing
- Ambiti di applicazione delle immagini:
 - * Classificazione
 - * Rilevamento di oggetti
 - * Segmentazione
 - * Super Risoluzione
 - * Rilevamento delle pose
- Ambiti di applicazione dei video:
 - * Classificazione
 - * Generazione
 - * Audio/testo

Faster RCNN in Tensorflow

- ▶ Esistono diversi modelli pre-addestrati per Faster RCNN (alcuni sviluppati per dispositivi mobili).
- ▶ Utilizzeremo il Faster RCNN con un ResNet50 v1 per immagini di dimensioni 800x1333.

```

class_id_to_label = {"1": "person", "2": "bicycle", "3": "car", "4": "motorcycle", "5": "airpl
color_dict = {"1": (255, 0, 0), "3": (255, 255, 0), "10": (0, 255, 0), "32": (0, 0, 255),
threshold_conf = 0.3
image_np = cv2.imread('img.jpg')
image_np = cv2.resize(image_np, (1333, 800)) # if I don't, it would do that the net

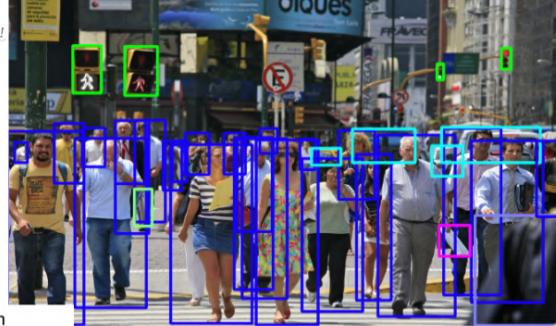
image = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB) #unclear from doc if needed!
image = np.reshape(image, (1, image.shape[0], image.shape[1], 3))

# Apply image detector on a single image.
model_handle = "faster_rcnn_resnet50_v1_800x1333_1"
detector = hub.load(model_handle)
detector_output = detector(image)

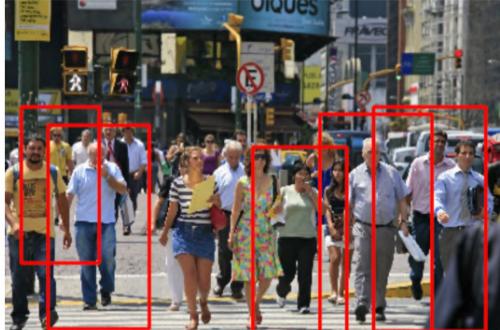
num_detections = detector_output["num_detections"]
class_ids = detector_output["detection_classes"].numpy()
bbox = detector_output["detection_boxes"].numpy()
conf = detector_output["detection_scores"].numpy()
bbox = np.squeeze(bbox)

+ code to draw results
conf 0.99838203 label person
conf 0.99770355 label person
conf 0.9968723 label person
conf 0.9966016 label traffic light .... conf 0.3481996 label traffic light
conf 0.33030114 label surfboard
conf 0.31558338 label person

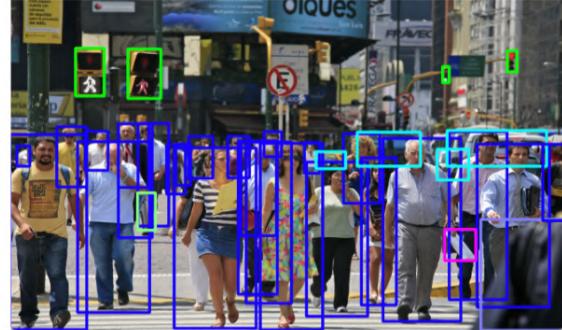
```



Confronto Qualitativo: HoG Detector vs Faster RCNN



Hog Pedestrian Detector

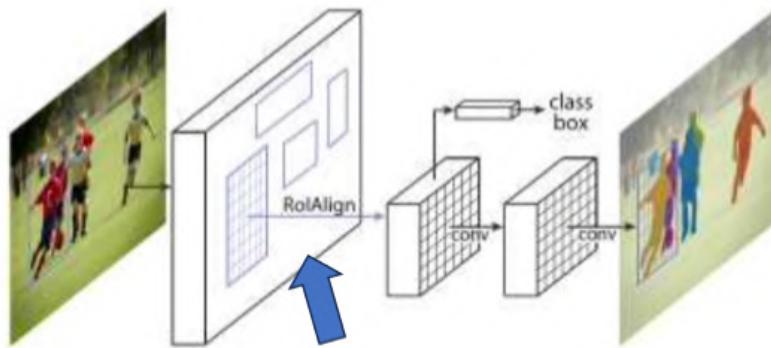


Faster RCNN

Segmentazione di Istanza e Mask RCNN

- ▶ La segmentazione di istanza è il compito di trovare tutti gli oggetti rilevanti in un'immagine e produrre maschere con precisione al pixel per le loro regioni visibili.
- ▶ Con l'avvento del deep learning, i ricercatori hanno iniziato a combinare proposte di regioni o pre-segmentazioni di immagini con fasi di convoluzione per inferire la segmentazione finale delle istanze.
- ▶ Un passo avanti nella segmentazione delle istanze è stato fatto con l'introduzione di Mask R-CNN (He, Gkioxari et al. 2017).
- ▶ Mask R-CNN utilizza la stessa rete di proposte di regioni (Region Proposal Network, RPN) di Faster R-CNN (Ren, He et al. 2015), ma aggiunge un ramo aggiuntivo per prevedere la maschera dell'oggetto, oltre al ramo esistente per la rifinitura della finestra di delimitazione e la classificazione.

- ▶ Come per altre reti che hanno più rami (o testate) e output, le perdite di addestramento corrispondenti a ciascun output supervisionato devono essere bilanciate con attenzione. È anche possibile aggiungere rami aggiuntivi, ad esempio rami addestrati per rilevare la posizione dei punti chiave umani.

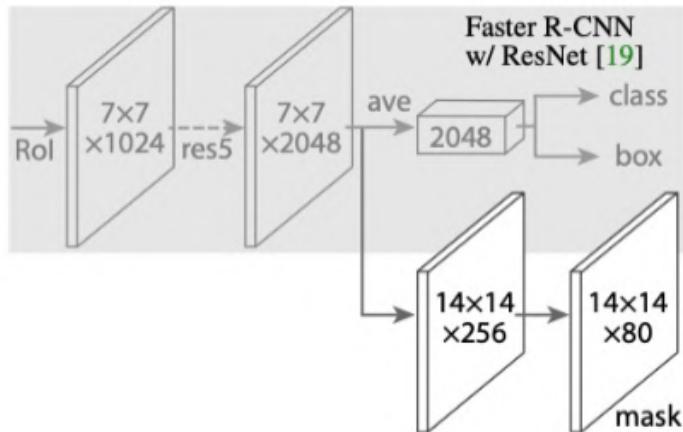


Definizione della Maschera

- ▶ Una maschera codifica la disposizione spaziale di un oggetto in input.
- ▶ A differenza delle etichette di classe o degli offset delle finestre che inevitabilmente vengono compressi in brevi vettori di output dai layer completamente connessi (fc), l'estrazione della struttura spaziale delle maschere può essere affrontata naturalmente attraverso la corrispondenza pixel-per-pixel fornita dalle convoluzioni.
- ▶ Pertanto, Mask-RCNN utilizza una Rete Completamente Convoluzionale (FCN), il che significa che, alla fine del ramo che prevede le maschere, non c'è alcun layer denso.
- ▶ Questo FCN è applicato a ciascun RoI, prevedendo una maschera di segmentazione in modo pixel-per-pixel.
- ▶ Il ramo della maschera prevede una maschera $m \times m$ da ciascun RoI mantenendo la disposizione spaziale dell'oggetto senza comprimerla in una rappresentazione vettoriale priva di dimensioni spaziali.

Output di Mask-RCNN

- ▶ Supponendo K classi e RoI di dimensioni $m \times m$, il ramo della maschera ha un output $K(m \times m)$ -dimensionale per ciascun RoI, che codifica K maschere binarie di risoluzione $m \times m$, una per ciascuna delle K classi.
- ▶ A questo output applichiamo una funzione *sigmoid* per pixel (anziché una *softmax*, che farebbe competere le maschere tra le classi).
- ▶ È essenziale separare la previsione della maschera e della classe: una maschera binaria è prevista per ciascuna classe indipendentemente, senza competizione tra le classi. La categoria viene invece prevista dal ramo di classificazione del RoI.

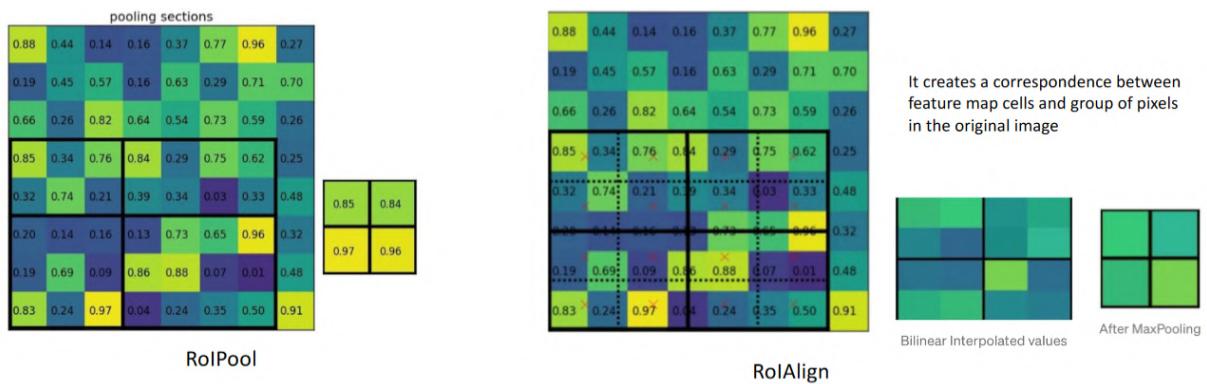


Testata di Faster R-CNN con un Backbone ResNet (res5) a cui è aggiunto un ramo di maschera.

- ▶ I numeri indicano la risoluzione spaziale e i canali.
- ▶ Le frecce indicano layer di tipo conv, deconv o fc come si può dedurre dal contesto (conv preserva la dimensione spaziale mentre deconv la aumenta).
- ▶ Tutti i conv sono 3×3 , eccetto il conv di output che è 1×1 , i deconv sono 2×2 con passo 2, e ReLU è usato nei layer nascosti.
- ▶ Il primo conv opera su un RoI 7×7 con passo 1. Segue un deconv e un conv 1×1 . Il numero di classi è 80.

RoIAlign – Mask RCNN

- ▶ In Faster R-CNN, il RoIPool esegue una quantizzazione spaziale grossolana per l'estrazione delle caratteristiche e introduce un disallineamento tra le caratteristiche e i pixel originali dell'immagine, causando una grande perdita di precisione nella localizzazione.
- ▶ Mask-RCNN sostituisce RoIPool con un nuovo layer privo di quantizzazione, chiamato RoIAlign, che preserva fedelmente le esatte posizioni spaziali.
- ▶ RoIAlign esegue il campionamento della mappa di caratteristiche. Il RoI è diviso in bin di dimensioni uguali (ad esempio, 3.5×2.5). L'interpolazione bilineare viene utilizzata per calcolare i valori esatti delle caratteristiche in input nei punti regolarmente campionati (indicati dalla x rossa) in ciascun bin del RoI, e aggregare il risultato (utilizzando max o average).
- ▶ Crea una corrispondenza tra le celle della mappa di caratteristiche e i gruppi di pixel nell'immagine originale.



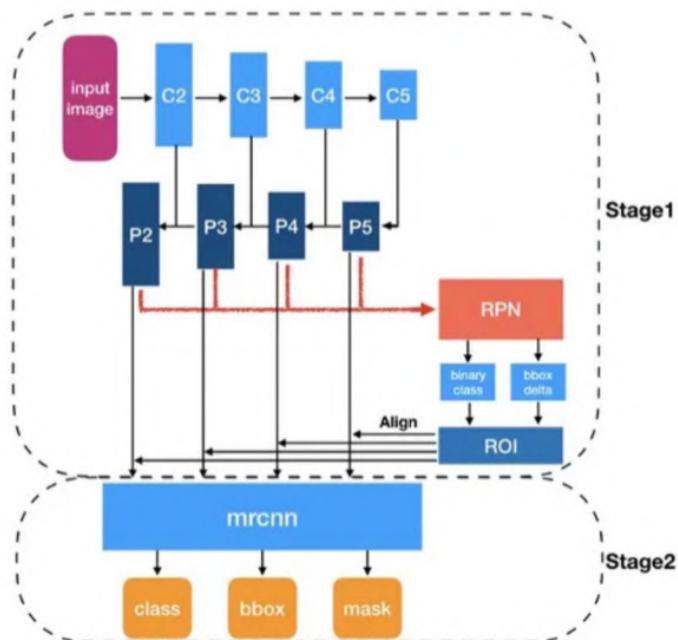
Architettura di Mask-RCNN

► Fase 1

- La rete prevede le classi e regredisce le finestre di delimitazione.
- Utilizza una CNN e una PFN. La PFN alimenta una RPN con due testate.
- Sulla base delle finestre di delimitazione, vengono estratti i RoI.

► Fase 2

- La rete utilizza i RoI per produrre le maschere binarie.



Addestramento di Mask-RCNN

- ▶ Durante l'addestramento, una perdita multi-task viene misurata su ciascun RoI:

$$L = L_{\text{cls}} + L_{\text{box}} + L_{\text{mask}}$$

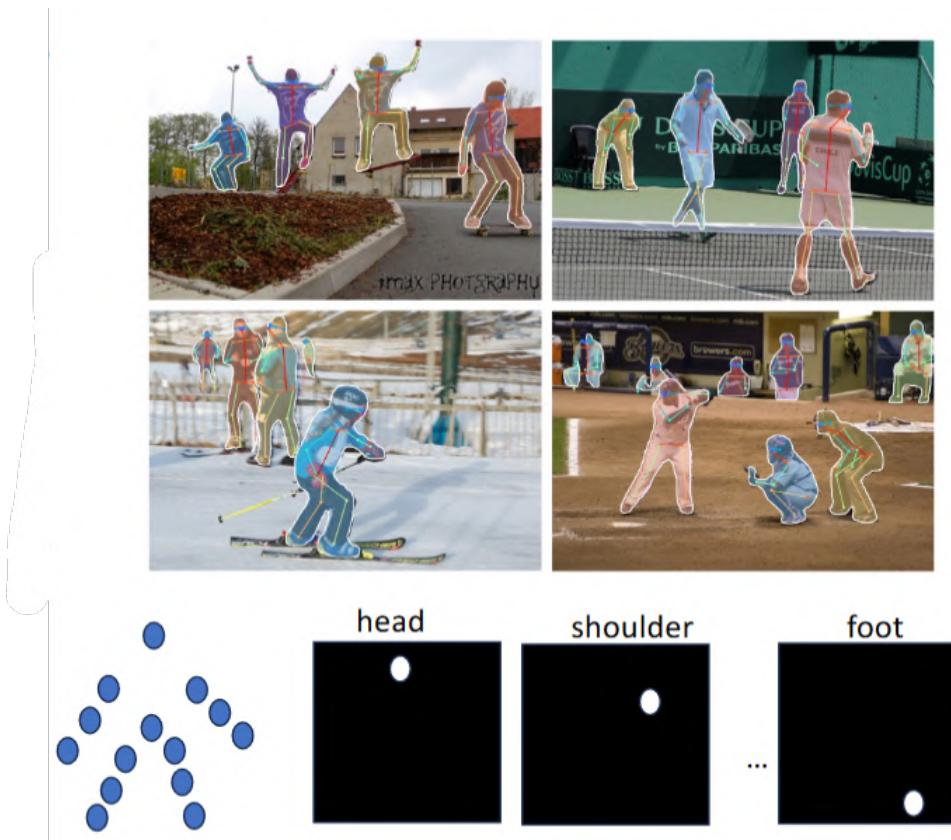
- ▶ La perdita di classificazione L_{cls} e la perdita della finestra di delimitazione L_{box} sono identiche a quelle definite in Faster RCNN.
- ▶ Per un RoI associato alla classe reale k , L_{mask} è definito solo sulla k -esima maschera (gli altri output delle maschere non contribuiscono alla perdita).
- ▶ L_{mask} è definito come la perdita media di cross-entropy binaria.

$$\mathcal{L}_{\text{mask}} = -\frac{1}{m^2} \sum_{1 \leq i, j \leq m} [y_{ij} \log \hat{y}_{ij}^k + (1 - y_{ij}) \log(1 - \hat{y}_{ij}^k)]$$

- ▶
- ▶ La definizione di L_{mask} permette alla rete di generare maschere per ogni classe senza competizione tra le classi. Questo separa la previsione della maschera e della classe ed è fondamentale per ottenere buoni risultati nella segmentazione delle istanze. È diverso dalla pratica comune nella segmentazione semantica, che tipicamente utilizza una softmax per pixel e una perdita di cross-entropy multinomiale.

9.8 Mask-RCNN per il Rilevamento della Posizione Umana

- ▶ Mask-RCNN può essere facilmente esteso alla stima della posa umana sostituendo il concetto di classe con quello di keypoint.
- ▶ La posa umana è rappresentata in termini di keypoint localizzati sulle articolazioni del corpo.
- ▶ La posizione del keypoint è codificata come una maschera one-hot, e Mask R-CNN continua a prevedere K maschere, una per ciascuno dei K tipi di keypoint (ad esempio, spalla sinistra, gomito destro).
- ▶ Per ciascuno dei K keypoint di un'istanza, l'obiettivo di addestramento è una maschera binaria one-hot $m \times m$ in cui solo un singolo pixel è etichettato come primo piano.
- ▶ Durante l'addestramento, per ciascun keypoint visibile di ground-truth, la perdita di entropia incrociata è minimizzata su un'uscita softmax m^2 -way (il che significa che c'è competizione tra i punti per lo stesso keypoint).
- ▶ I K keypoint sono ancora trattati indipendentemente.



9.9 Detectron

- ▶ Detectron è il sistema software di Facebook AI Research che implementa algoritmi di rilevamento di oggetti all'avanguardia, incluso Mask R-CNN.
- ▶ È scritto in Python e utilizza principalmente PyTorch. Esiste anche una versione scritta in TensorFlow. Ha una dipendenza da Caffe2 (una libreria C per le NN, il primo framework di deep learning).
- ▶ Detectron include implementazioni dei seguenti algoritmi di rilevamento di oggetti:
 - Mask R-CNN
 - Faster R-CNN
 - RPN
 - Fast R-CNN
 - R-FCN
- ▶ Utilizzando le seguenti architetture di rete backbone:
 - ResNeXt{50,101,152}
 - ResNet{50,101,152}
 - Feature Pyramid Networks (con ResNet/ResNeXt)
 - VGG16

9.10 Model Zoo – Mask RCNN

- Model Zoo cura e fornisce una piattaforma per i ricercatori di deep learning per trovare facilmente modelli pre-addestrati per una varietà di piattaforme e utilizzi.

Mask-RCNN su Model Zoo

```
def test_Mask_RCNN():
    #COCO LABELS:
    class_names = [...]

    image_np = cv2.imread('img.jpg')
    image = cv2.resize(image_np, (1024, 1024)) # if I don't, it would do that the net
    #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #unclear from doc if needed!

    # Create model object in inference mode.
    config = InferenceConfig()
    #config.display() #it prints info about the model
    model = modellib.MaskRCNN(mode="inference", model_dir=".logs", config=config)
    # Load weights trained on MS-COCO
    model.load_weights("mask_rcnn_coco.h5", by_name=True)
    # Run detection
    results = model.detect([image], verbose=1)
    # Visualize results
    r = results[0] #1 image only
    print(r['rois'].shape)      (29, 4)
    print(r['masks'].shape)     (1024, 1024, 29)
    visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                                class_names, r['scores'])
```



9.11 Two Stage vs Single Stage

- In un rilevatore a due stadi, una rete di proposte di regioni (RPN) seleziona le posizioni e le forme delle rilevazioni da considerare, e una seconda rete viene quindi utilizzata per classificare e regredire i pixel o le caratteristiche all'interno di ciascuna regione.
- Una rete a singolo stadio utilizza una singola rete neurale per produrre rilevazioni in una varietà di posizioni.
- Esempi di tali rilevatori sono:
 - SSD (Single Shot MultiBox Detector) di Liu, Anguelov et al. (2016)
 - YOLO (You Only Look Once) descritto in Redmon 2015, Redmon 2017 e Redmon 2018
 - RetinaNet (Lin, Goyal et al. 2017)

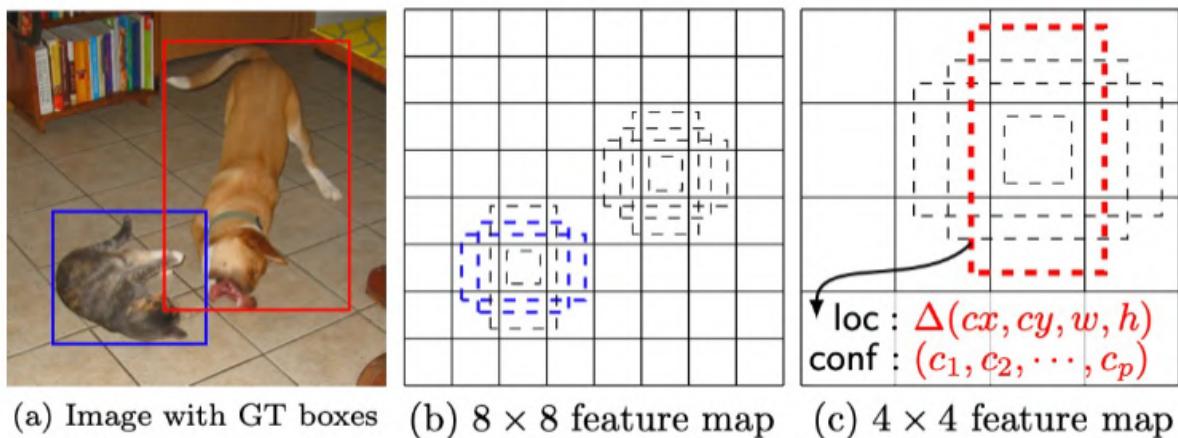
9.12 SSD - 2016

- SSD elimina completamente la generazione di proposte e i successivi stadi di ricampionamento dei pixel o delle caratteristiche, incapsulando tutto il calcolo in una singola rete.
- SSD discretizza lo spazio di output delle bounding box in un insieme di box predefiniti con diversi rapporti d'aspetto e scale per ogni posizione della mappa delle caratteristiche.

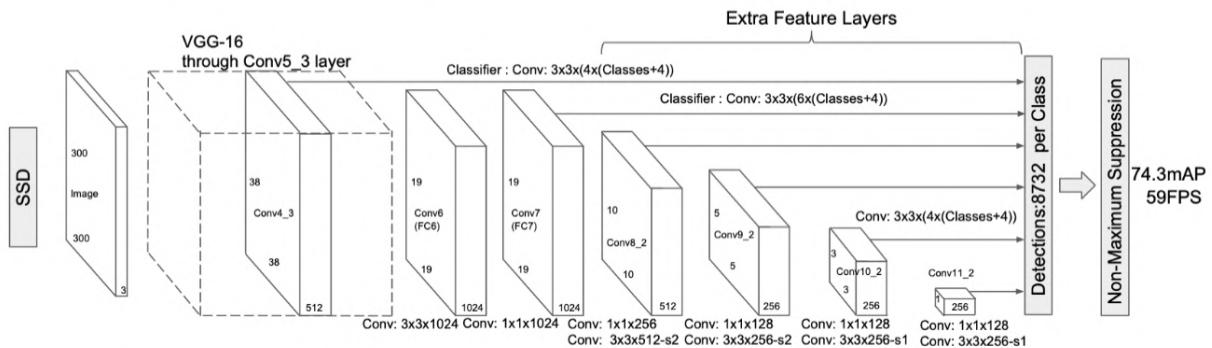
- ▶ Al momento della previsione, la rete genera punteggi per la presenza di ciascuna categoria di oggetti in ciascuna box predefinita e produce aggiustamenti alla box per meglio adattarsi alla forma dell'oggetto.
- ▶ Inoltre, la rete combina le previsioni da più mappe delle caratteristiche con diverse risoluzioni per gestire naturalmente oggetti di diverse dimensioni.
- ▶ Per uno strato di caratteristiche di dimensioni $m \times n$ con p canali, un kernel 3×3 produce un punteggio per una categoria o un offset di forma relativo alle coordinate della box predefinita.

9.13 SSD

- ▶ Il miglioramento fondamentale nella velocità deriva dall'eliminazione delle proposte di bounding box e della successiva fase di ricampionamento dei pixel o delle caratteristiche.
- ▶ I miglioramenti includono:
 - l'uso di un piccolo filtro convoluzionale per prevedere le categorie di oggetti e gli offset nelle posizioni delle bounding box, utilizzando predittori separati (filtri) per diverse rilevazioni con rapporti d'aspetto.
 - applicare questi filtri a più mappe delle caratteristiche dagli stadi successivi di una rete per eseguire il rilevamento a più scale.
- ▶ <https://arxiv.org/abs/1512.02325>
- ▶ SSD necessita solo di un'immagine di input e delle bounding box di ground truth per ogni oggetto durante l'addestramento. In modo convoluzionale, valuta un piccolo set (ad esempio 4) di box predefinite con diversi rapporti d'aspetto in ogni posizione in più mappe delle caratteristiche con diverse scale (ad esempio 8×8 e 4×4 in (b) e (c)).
- ▶ Per ciascuna box predefinita, prevede sia gli offset di forma sia le confidenze per tutte le categorie di oggetti ((c_1, c_2, \dots, c_p)).



SSD Architecture



For each of the k boxes at a given location, it computes c class scores and the 4 offsets relative to the original default box shape. This results in a total of $(c + 4) \times k$ filters that are applied around each location in the feature map, yielding $(c + 4)kmn$ outputs for a $m \times n$ feature map.

9.14 Architettura SSD

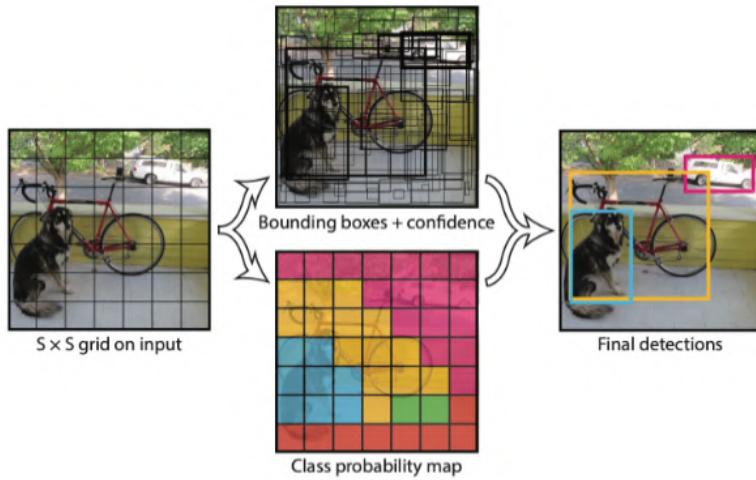
- Per ciascuna delle k box in una data posizione, calcola c punteggi di classe e i 4 offset relativi alla forma della box predefinita originale. Questo risulta in un totale di $(c + 4) \times k$ filtri che vengono applicati intorno a ciascuna posizione nella mappa delle caratteristiche, producendo $(c + 4)kmn$ output per una mappa delle caratteristiche $m \times n$.

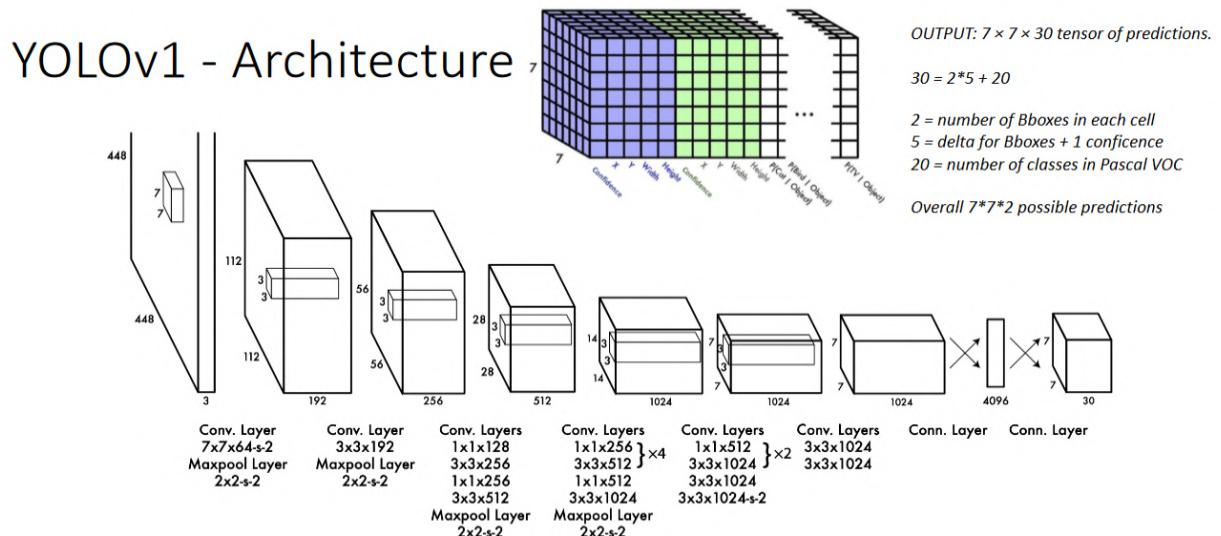
9.15 YOLO – You Only Look Once, dal 2015

- Il rilevatore YOLO è probabilmente uno dei rilevatori di oggetti a singolo stadio più famosi. È migliorato molto negli anni e fino ad oggi abbiamo YOLOv8.
- YOLOv1 è stato introdotto nel 2015. Successivamente, sono state rilasciate più versioni di YOLOv2, YOLOv3, YOLOv4 e YOLOv5, sebbene da persone diverse.
- In YOLO, definiamo:
 - Backbone:** una CNN che produce caratteristiche visive con diverse forme e dimensioni (ad esempio, ResNet, VGG e EfficientNet utilizzate come estrattori di caratteristiche).
 - Neck:** un insieme di strati che integrano e mescolano caratteristiche prima di passarle allo strato di previsione (ad esempio, FPN, Path Aggregation Network (PAN) e Bi-FPN).
 - Head:** prende le caratteristiche dal neck, esegue la classificazione insieme alla regressione sulle caratteristiche e sulle coordinate delle bounding box per completare il processo di rilevamento. Emette 4 valori, generalmente coordinate x, y insieme alla larghezza e all'altezza.

9.16 YOLOv1 (Redmon, Divvala, Girshick, Farhadi, 2016)

- ▶ Le immagini sono ridimensionate a 448×448 .
- ▶ Le caratteristiche dell'intera immagine sono utilizzate per prevedere ciascuna bounding box (implicitamente codifica informazioni contestuali sulle classi). Prevede tutte le bounding box per tutte le classi per un'immagine contemporaneamente.
- ▶ Il modello divide un'immagine in ingresso in celle $S \times S$ e calcola la probabilità che un oggetto si trovi all'interno di quella cella. Questo viene fatto per tutte le celle in cui l'immagine è divisa.
- ▶ Se il centro di un oggetto cade in una cella della griglia, quella cella è responsabile del rilevamento di quell'oggetto. Ogni cella della griglia prevede B bounding box e punteggi di confidenza per quelle box.
- ▶ La confidenza è definita come $\text{Pr}(\text{Object}) \times \text{IOU}$. Se non esiste alcun oggetto in quella cella, i punteggi di confidenza dovrebbero essere zero. Altrimenti, vogliamo che il punteggio di confidenza sia uguale all'intersezione sull'unione (IOU) tra la box prevista e la ground truth.
- ▶ L'algoritmo raggruppa celle ad alta probabilità vicine come un singolo oggetto. Le previsioni a bassa probabilità vengono scartate utilizzando la Non-Max Suppression (NMS).
- ▶ La backbone è addestrata a bassa risoluzione ma utilizzata su immagini ad alta risoluzione.
- ▶ La rete (ispirata al modello GoogLeNet) ha 24 strati convoluzionali seguiti da 2 strati completamente connessi. Strati di convoluzione 1×1 alternati riducono lo spazio delle caratteristiche dai precedenti strati.





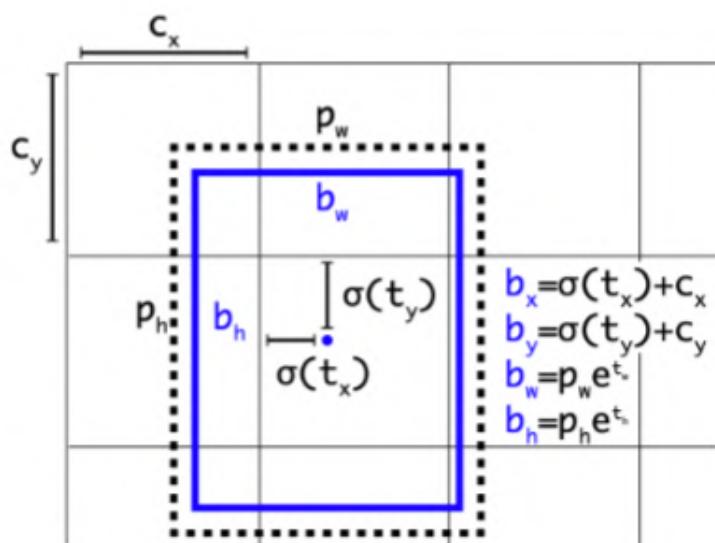
9.17 YOLOv2

- YOLOv2 è stato rilasciato da Joseph Redmon e Ali Farhadi nel 2016 nel loro paper intitolato "YOLO9000: Better, Faster, Stronger". Il 9000 significava che YOLOv2 era in grado di rilevare oltre 9000 categorie di oggetti.
- Rispetto a Fast R-CNN, YOLO commette un numero significativo di errori di localizzazione e ha un richiamo basso.
- YOLOv2 è più semplice e più facile da apprendere. Include:
 - Batch Normalization: miglioramento della convergenza della rete e regolarizzazione.
 - High Resolution Classifier: la rete di classificazione è addestrata alla risoluzione completa di 448×448 per 10 epoch su ImageNet. Quindi la rete risultante è rifinita sul rilevamento. Apporta modifiche (immagini 416×416 e eliminazione di uno strato di pooling) per rendere l'output degli strati convoluzionali ad alta risoluzione e con dimensioni dispari (13×13).
 - Convolutional With Anchor Boxes: YOLO prevede direttamente le coordinate delle bounding box utilizzando strati densi sopra l'estrattore di caratteristiche convoluzionali. YOLOv2 rimuove gli strati densi e utilizza anchor box per prevedere le bounding box. Questo comporta una piccola diminuzione dell'accuratezza ma consente di prevedere più di mille BBoxes.

9.18 YOLOv2 – lavorare con gli anchor

- In Faster RCNN, le dimensioni degli anchor sono scelte a mano.
- La rete può imparare ad aggiustare le box in modo appropriato, ma i priori sulle dimensioni delle box possono facilitare l'apprendimento della rete per prevedere buone rilevazioni.

- ▶ Pertanto, il clustering k-means sulle bounding box del set di addestramento viene utilizzato per trovare automaticamente buoni priori basati sull'IoU della box.
- ▶ 5 cluster risultano nei priori sulla destra.
- ▶ Invece di prevedere offset come in Faster RCNN, YOLOv2 prevede le coordinate della posizione relative alla posizione della cella della griglia (come YOLOv1). Questo vincola la ground truth a cadere tra 0 e 1. Viene utilizzata un'attivazione logistica per limitare le previsioni della rete a rientrare in questo intervallo.
- ▶ Infine, si basa su Darknet-19 (proposto nello stesso paper), che utilizza principalmente filtri 3×3 e raddoppia il numero di canali dopo ogni passaggio di pooling. Utilizzava il pooling medio globale per fare previsioni e filtri 1×1 per comprimere la rappresentazione delle caratteristiche tra le convoluzioni 3×3 .



9.19 YOLOv3

- ▶ Nel 2018, Joseph Redmon e Ali Farhadi hanno introdotto la terza versione di YOLOv3 nel loro paper "YOLOv3: An Incremental Improvement". Questo modello era un po' più grande rispetto ai precedenti ma più accurato e ancora abbastanza veloce.
- ▶ YOLOv3 consisteva in 75 strati convoluzionali senza utilizzare strati completamente connessi o di pooling, il che riduceva notevolmente le dimensioni e il peso del modello.
- ▶ Utilizza blocchi residui e Feature Pyramid Network (FPN) mantenendo tempi di inferenza minimi. Pertanto, prevede box a 3 diverse scale.
- ▶ La backbone prende il nome di Darknet53.

9.20 YOLOv4, 2020

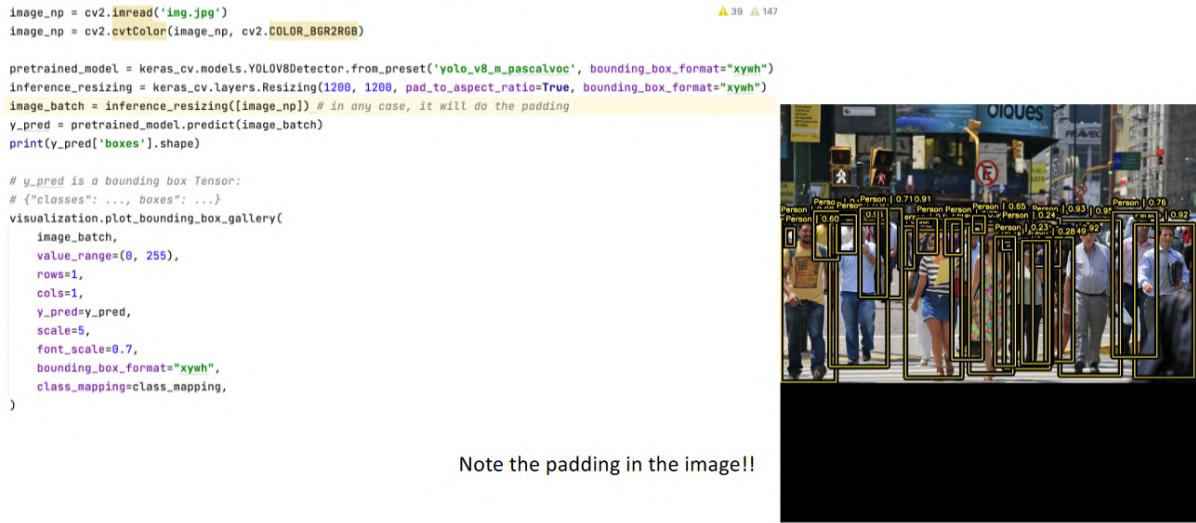
- ▶ YOLOv4 è stato rilasciato da Alexey Bochkovskiy et al. nel loro paper del 2020 "YOLOv4: Optimal Speed and Accuracy of Object Detection".
- ▶ YOLOv4 utilizza la backbone Darknet53 (la stessa di YOLOv3) e diverse ottimizzazioni per migliorare la velocità.
- ▶ "Bag of freebies" sono tecniche che influenzano la strategia di addestramento. Include data augmentation, random erase e CutOut per selezionare casualmente la regione rettangolare in un'immagine e riempirla con un valore casuale o complementare di zero, dropOut, MixUp (usa due immagini da moltiplicare e sovrapporre con diversi coefficienti), CutMix (copre l'immagine ritagliata alla regione rettangolare di altre immagini). Include anche varianti di tecniche di normalizzazione, regolarizzazione e funzioni di perdita.
- ▶ "Bag of specials" sono moduli plugin e metodi di post-elaborazione che migliorano la precisione del rilevamento degli oggetti. Includono metodi per ingrandire il campo ricettivo, introdurre meccanismi di attenzione o rafforzare la capacità di integrazione delle caratteristiche.
- ▶ YOLOv4 sperimenta queste tecniche per progettare un rilevatore a singolo stadio più veloce.

9.21 YOLOv5-YOLOv8 passando per YoloX

- ▶ YOLOv5 è stato rilasciato nel 2020 dalla compagnia Ultralytics. Non è stato rilasciato alcun paper, ma può essere trovato su <https://ultralytics.com/yolov5>.
- ▶ Utilizza una migliore data augmentation e calcoli di perdita, autoapprendimento degli anchor box e altri miglioramenti.
- ▶ Da YOLOv4, ci sono state diverse varianti, tra cui YOLOv6, YoloX e, più recentemente (nel 2022), YOLOv7, ma il loro confronto con lo stato dell'arte è poco chiaro.
- ▶ Sono disponibili molte implementazioni per PyTorch.
- ▶ YOLOv5 ha anche il codice per esportare da PyTorch a TensorFlow e anche personalizzare le dimensioni dell'immagine di input.
- ▶ YOLOv8 è anche sviluppato da Ultralytics (2023) e può essere trovato su <https://github.com/ultralytics/ultralytics/issues/189>.

pip install keras-cv

Yolov8 on Keras_CV

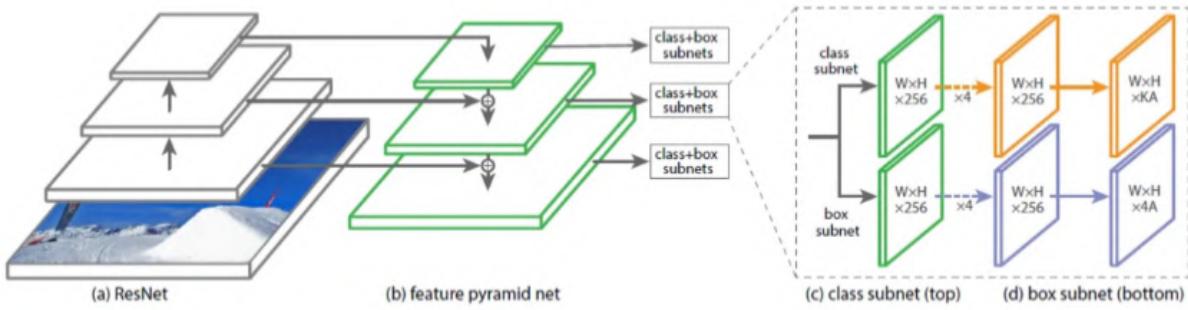


9.22 RetinaNet

- RetinaNet è un rilevatore a singolo stadio costruito sopra una FPN. Ha dimostrato di funzionare bene con oggetti densi e di piccola scala. Per questo motivo, è diventato un modello di rilevamento di oggetti popolare da utilizzare con immagini aeree e satellitari.
- Utilizza una focal loss per focalizzare l'addestramento su esempi difficili riducendo il peso della perdita su campioni ben classificati, prevenendo così che il gran numero di negativi facili sovraccarichi l'addestramento.
- La Focal Loss (FL) è un miglioramento rispetto alla Cross-Entropy Loss (CE) ed è introdotta per gestire il problema dello squilibrio di classe.
- I modelli a singolo stadio soffrono di un problema di squilibrio estremo di classe foreground-background a causa del campionamento denso delle anchor box (posizioni degli oggetti possibili).
- In RetinaNet, a ciascun livello della piramide possono esserci migliaia di anchor box. Solo poche saranno assegnate a un oggetto di ground-truth mentre la stragrande maggioranza sarà di classe background. Questi esempi facili (rilevamenti con alte probabilità) sebbene risultino in piccoli valori di perdita possono collettivamente sovraccaricare il modello. La Focal Loss riduce il contributo della perdita dagli esempi facili e aumenta l'importanza di correggere gli esempi classificati erroneamente.
- Ci sono quattro componenti principali dell'architettura del modello RetinaNet:
 - **Bottom-up Pathway:** la rete backbone (ad esempio ResNet) che calcola le mappe delle caratteristiche a diverse scale, indipendentemente dalle dimensioni dell'immagine di input o dalla backbone.
 - **Top-down pathway e Lateral connections:** (FPN) Il top-down pathway upsampling le mappe delle caratteristiche spazialmente più grossolane dai livelli più alti della piramide,

e le lateral connections fondono i livelli top-down e bottom-up con la stessa dimensione spaziale.

- **Classification subnetwork:** prevede la probabilità che un oggetto sia presente in ciascuna posizione spaziale per ciascuna anchor box e classe di oggetti.
- **Regression subnetwork:** regredisce l'offset per le bounding box dalle anchor box per ciascun oggetto di ground-truth.



RetinaNet in Keras_cv

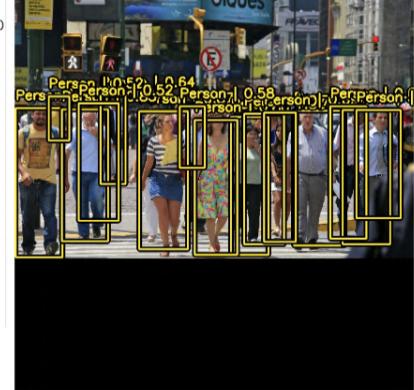
```

class_mapping = dict(zip(range(len(class_ids)), class_ids))

image_np = cv2.imread('img.jpg')
image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)

pretrained_model = keras_cv.models.RetinaNet.from_preset("retinanet_resnet50_pascalvoc", bounding_box_format="xywh")
inference_resizing = keras_cv.layers.Resizing(640, 640, pad_to_aspect_ratio=True, bounding_box_format="xywh")
image_batch = inference_resizing([image_np])
y_pred = pretrained_model.predict(image_batch)
# y_pred is a bounding box Tensor:
# {"classes": ..., "boxes": ...}
visualization.plot_bounding_box_gallery(
    image_batch,
    value_range=(0, 255),
    rows=1,
    cols=1,
    y_pred=y_pred,
    scale=5,
    font_scale=0.7,
    bounding_box_format="xywh",
    class_mapping=class_mapping,
)

```



TRACKING

10.1 Tracciamento Visivo

Il tracciamento visivo è un problema difficile poiché molte circostanze diverse e variabili devono essere conciliabili in un singolo algoritmo.

Ad esempio, un tracker può essere efficace nel gestire variazioni di illuminazione, ma avere difficoltà nel far fronte a cambiamenti di aspetto dell'oggetto dovuti a variazioni nei punti di vista dell'oggetto. Un tracker può prevedere il movimento e la velocità del bersaglio, ma può avere difficoltà nel tracciare oggetti rimbalzanti o cambiamenti improvvisi nella direzione del movimento.

Un tracker può fare una previsione dettagliata dell'aspetto previsto, ma potrebbe fallire su un oggetto articolato.

Come inizializzare un tracker?

Dato una sequenza di frame, il primo problema è capire cosa vogliamo tracciare e come possiamo ottenere la posizione del bersaglio nel primo frame. Se non facciamo nessuna assunzione su quale sia il bersaglio, possiamo cercare il movimento nelle immagini e tracciare qualsiasi cosa si muova nelle immagini. Vogliamo tracciare tutti i possibili cambiamenti nelle immagini? O forse solo quelli che hanno senso per la nostra applicazione?

Movimento per differenza tra frame



Previous frame



Current frame

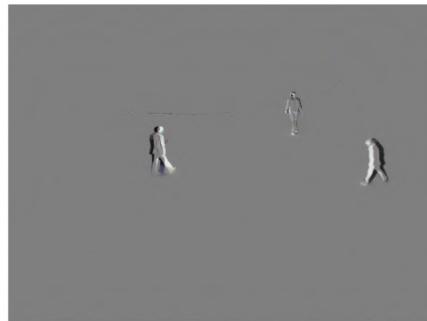
I risultati dipendono da come trattiamo i valori dei pixel, che saranno compresi tra -255 e 255.

Importante!! Questo approccio ha senso per telecamere statiche. Quando la telecamera si muove, dobbiamo anche considerare il movimento proprio della telecamera e compensarlo.



Absolute Differences after rescaling
in 0 255. Black means no motion

```
diff = frame_prev.astype(np.float32) - frame.astype(np.float32)
diff = np.absolute(diff)
diff = cv.normalize(diff, diff, 0, 255, cv.NORM_MINMAX, cv.CV_8UC3)
```

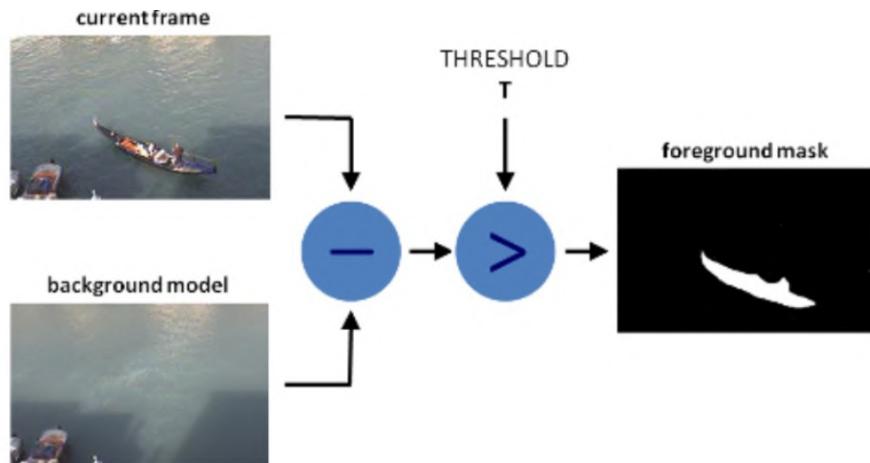


Differences after rescaling in 0 255
Gray means no motion

```
diff = frame_prev.astype(np.float32) - frame.astype(np.float32)
diff = cv.normalize(diff, diff, 0, 255, cv.NORM_MINMAX, cv.CV_8UC3)
```

Possiamo anche applicare una soglia (threshold) prima della riscalatura per filtrare il rumore. Come puoi vedere, i risultati dipendono dalla velocità dell'oggetto.
Abbiamo bisogno di operazioni morfologiche per trovare componenti connesse (es: dilatazione, soglia di isteresi, componenti connesse con 4-adjacency... tutti operatori che abbiamo visto nella prima parte del corso).

Modellazione di Primo Piano e Sfondo



Importante!! Puoi facilmente ottenere un modello di sfondo imparando incrementalmente dalle immagini solo se la telecamera è statica. Se la telecamera si muove, solo in alcuni casi è possibile utilizzare questa soluzione. In tutti i casi generali, questa non è la soluzione corretta!

Modellazione dello Sfondo

La modellazione dello sfondo consiste in due passaggi principali:

- ▶ Inizializzazione dello Sfondo;
- ▶ Aggiornamento dello Sfondo.

Nel primo passaggio, viene calcolato un modello iniziale dello sfondo.

Nel secondo passaggio, il modello viene aggiornato per adattarsi a possibili cambiamenti nella scena.

Dato un modello dello sfondo, il primo piano viene rilevato tramite sottrazione e soglia. Ogni frame viene utilizzato sia per calcolare la maschera del primo piano che per aggiornare lo sfondo. Il modello di sfondo più famoso è basato su una Miscela di Gaussiane.

Modellazione dello Sfondo con MoG

Ogni pixel X_t è caratterizzato dalla sua intensità nello spazio dei colori RGB. La probabilità di osservare il valore del pixel corrente è:

$$P(X_t) = \sum_{i=1}^K \omega_{i,t} \eta(X_t; \mu_i, \Sigma_{i,t})$$

Questa è una somma pesata di K Distribuzioni Gaussiane con una media e una matrice di Covarianza.

$$\eta(X_t, \mu, \Sigma) = \frac{e^{(-\frac{1}{2}(X_t - \mu)\Sigma^{-1}(X_t - \mu))}}{(2\pi)^{n/2}|\Sigma|^{1/2}} \quad (10.1)$$

Idealmente, la procedura di addestramento mira a stimare i pesi e le medie e le matrici di Covarianza. Questo è un algoritmo ben noto che utilizza il clustering K-means per inizializzare le modalità e l'algoritmo EM per affinare i parametri dei modelli.

Il modello è un modello pixel-wise. Si basa sull'assunzione che la distribuzione dello sfondo e quella del primo piano siano Gaussiane, il che non è sempre vero. Il primo piano viene rilevato confrontando il pixel della nuova immagine con ciascuna Gaussiana e sogliando la probabilità calcolata di appartenenza allo sfondo. Questo è approssimato dalla distanza di Mahalanobis.

$$\text{sqrt} \left((X_{t+1} - \mu_{i,t})^T \Sigma_{i,t}^{-1} (X_{t+1} - \mu_{i,t}) \right) \leq k\sigma_{i,t}$$

Poiché lo sfondo può cambiare dinamicamente (come all'aperto), i parametri del MoG vengono appresi incrementalmente. Se viene trovata una corrispondenza.

$$\mu_{i,t+1} = (1 - \rho)\mu_{i,t} + \rho X_{t+1}$$

$$\sigma_{i,t+1}^2 = (1 - \rho)\sigma_{i,t}^2 + \rho(X_{t+1} - \mu_{i,t+1})(X_{t+1} - \mu_{i,t+1})^T$$

dove:

$$\rho = \alpha \eta(X_{t+1}, \mu_i, \Sigma_i)$$

$$\omega_{i,t+1} = (1 - \alpha)\omega_{i,t} + \alpha$$

Nel documento originale di Friedman e Stauffer, la matrice di Covarianza è diagonale.

Se non viene trovata una corrispondenza, c'è comunque un aggiornamento della Gaussiana per tenere conto dei cambiamenti nella scena. Esistono altre varianti, questa è l'implementazione di base.

```
backSub = cv.createBackgroundSubtractorMOG2()
video = '/Users/liliana/Desktop/CORSI/CV/2023_2024/Slides/codice/vtest.avi'
capture = cv.VideoCapture(video)
if not capture.isOpened():
    print('Unable to open: ' + video)
    exit(0)

while True:
    ret, frame = capture.read()
    if frame is None:
        break

    #update the background model
    fgMask = backSub.apply(frame)

    #show the current frame and the fg masks
    cv.imshow('Frame', frame)
    cv.imshow('FG Mask', fgMask)

    keyboard = cv.waitKey(30)
    if keyboard == 'q' or keyboard == 27:
        break
```



Issue: when a person stop, it is absorbed in the background

Once we get the foreground mask, we must calculate the **connected components**. Issues are: shadows and people who walk too close

Problema: quando una persona si ferma, viene assorbita nello sfondo. Una volta ottenuta la maschera del primo piano, dobbiamo calcolare i componenti connessi. I problemi sono: ombre e persone che camminano troppo vicine.

Come inizializzare un tracker?

Poiché rilevare il movimento è difficile, per garantire il confronto tra diversi approcci, spesso i tracker venivano inizializzati con il primo rilevamento (una bounding box) del bersaglio nel primo frame. In qualche modo, questo ha aiutato a migliorare le strategie di addestramento ignorando il problema dell'inizializzazione.



Quali sono gli indizi importanti per eseguire il tracciamento?

Movimento: Direzione e velocità. Problema: abbiamo solo dati sul piano immagine 2D, ma i bersagli "vivono" nel mondo 3D. Qui il problema è che dobbiamo prevedere le traiettorie dei bersagli in un mondo 3D ma la trasformazione prospettica ci fa perdere le informazioni sulla profondità. Generalmente cerchiamo di tracciare oggetti che si muovono sul piano del suolo. Inoltre, i bersagli non si muovono necessariamente seguendo leggi di movimento predefinite e possono fermarsi o cambiare direzione in qualsiasi momento.



Approcci popolari per modellare il movimento dei bersagli sono:

- ▶ Filtro di Kalman, che modella la posizione del bersaglio e la sua velocità e/o accelerazione considerando il rumore Gaussiano. Utilizza un sistema LTI (come hai visto in Controlli) e, basato sullo stato attuale del sistema, prevede la nuova posizione. Ha bisogno di un altro algoritmo (rilevatore) per migliorare la localizzazione del bersaglio e i cui output vengono utilizzati per affinare i parametri del Processo Gaussiano.
- ▶ Filtro a Particelle, basato sul campionamento della posizione del bersaglio e sulla propagazione nel tempo della conoscenza precedente.



Quali sono gli spunti importanti per eseguire il monitoraggio?

L'aspetto del target è rappresentato da segnali visivi. La rappresentazione dell'aspetto implica una costanza di una certa proprietà per trasferire un frame al successivo. Senza nessuna tale ipotesi di costanza, il tracking non può funzionare.

Ci sono tre rappresentazioni essenzialmente diverse:

1. Un array 2D come i dati dell'immagine, cioè un array di valori di luminosità. La luminosità costante è realistica per oggetti lontani dalla fotocamera, ma in molti casi reali, l'assunzione viene rapidamente violata. Sono state utilizzate diverse rappresentazioni di colore: colore HSI, gradienti HSI. Per il tracking generale, potrebbe essere necessaria una grande e diversificata rappresentazione delle caratteristiche per catturare le proprietà distintive del target.
2. Un istogramma 1D di proprietà ordinate, ad esempio dei colori. Un istogramma rimuove qualsiasi ordine spaziale dando massimo spazio alla flessibilità del target durante il movimento. Le informazioni spaziali devono essere catturate altrove nell'algoritmo di tracking. L'approccio più efficace è l'uso di più di un istogramma in diversi spazi delle caratteristiche (utilizzando un approccio di selezione delle caratteristiche dettagliato in Collins et al. PAMI 2005).
3. Un vettore di caratteristiche che cattura proprietà rilevanti. Quando la forma dell'oggetto è importante, deve essere codificata successivamente da vincoli geometrici. Gradienti di Haar, pattern binari 2D, caratteristiche SIFT o SURF, caratteristiche di colore Lab, bag of features, sono stati tutti utilizzati nel tracking.

10.2 Tracking e telecamere

Quando si parla di tracking, è necessario distinguere tra:

- ▶ Tracking con una singola telecamera statica, che ci consente di modellare lo sfondo.
- ▶ Tracking con una singola telecamera mobile, che ci impedisce di modellare lo sfondo e dobbiamo fare affidamento su un rilevatore specializzato del target.
- ▶ Tracking con più telecamere statiche. Questo può essere ulteriormente classificato come:
 - Tracking con più telecamere con campi visivi (FoV) sovrapposti, in questo caso la rete di telecamere ci consente di raccogliere più visualizzazioni del target. È necessario stimare quali aree si sovrappongono e quali no.

- Tracking con più telecamere con FoV non sovrapposti, in questo caso dobbiamo capire dove le persone che escono da un FoV riappariranno nei FoV delle telecamere più vicine, considerando anche la distanza tra le telecamere (che può essere nota a priori o appresa dal sistema).
- ▶ Tracking con più telecamere mobili. Campo poco esplorato ma potenzialmente molto utile.

10.3 Tracking Visivo di Oggetti

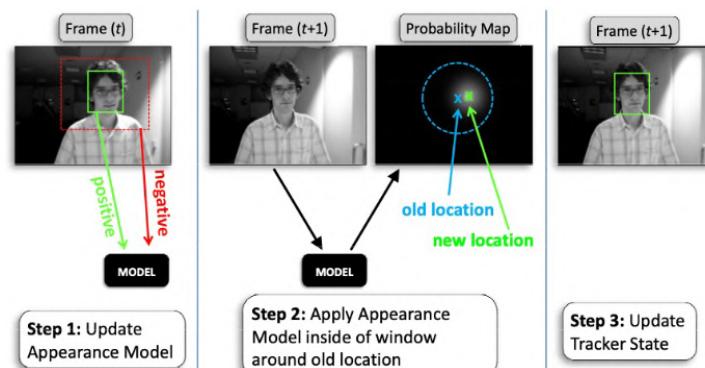
Il tracking di oggetti è un compito fondamentale della visione artificiale, che mira a prevedere la posizione di un dato oggetto target in ogni frame video. Questo compito è utilizzato in una vasta gamma di applicazioni nella robotica, videosorveglianza, auto autonome, interazione uomo-computer, realtà aumentata, tracking di animali e cellule, e altri campi.

I sottotask più popolari nel tracking visivo di oggetti sono:

- ▶ Tracking di Oggetto Singolo (SOT)
- ▶ Tracking di Oggetti Multipli (MOT)
- ▶ Tracking di Oggetti Multipli semi-supervisionato e agnostico alla classe
- ▶ Segmentazione degli Oggetti nel Video (VOS)

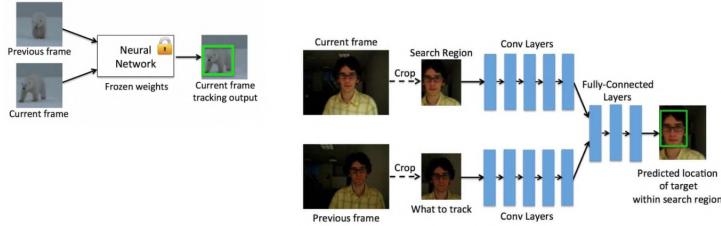
Esempio di SOT: Tracking-by-Detection

Una visione diversa del tracking è costruire il modello sulla distinzione del target dal background. Il tracking-by-detection costruisce un classificatore per distinguere i pixel del target dai pixel del background e aggiorna il classificatore con nuovi campioni in arrivo. Uno degli approcci più famosi si basa sull'apprendimento di istanze multiple (MI). Utilizza campioni da una regione di ricerca come campioni positivi e campioni dalla regione circostante come campioni negativi. Un classificatore viene addestrato incrementativamente utilizzando campioni raccolti nel tempo. Le caratteristiche di Haar sono utilizzate come caratteristiche. Il punteggio di classificazione più alto determina la nuova posizione nel MI Tracker. La novità del metodo risiede nell'uso di sacchetti di campioni. Un sacchetto è considerato positivo se contiene almeno un'istanza positiva. L'approccio di apprendimento è un esempio di boosting.



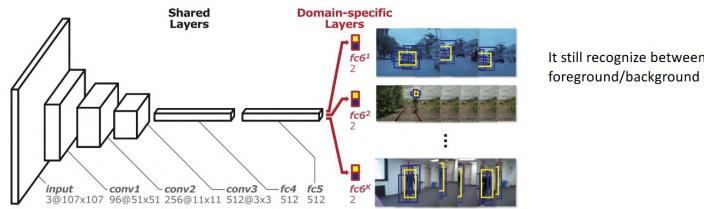
GOTURN

Il tracciamento generico di oggetti che utilizza reti di regressione, è un algoritmo di tracciamento basato sul Deep Learning. Utilizza il tracking siamese. È famoso perché è incredibilmente veloce e



l’addestramento è offline!

Esempio di SOT: MDNet



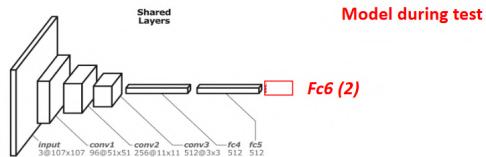
MDNet pre-addestra una CNN su dati su larga scala specializzati per il tracking visivo per ottenere una rappresentazione generica del target. La rete è composta da livelli condivisi e più rami di livelli specifici del dominio. I domini corrispondono a singole sequenze di addestramento e ciascun ramo è responsabile della classificazione binaria per identificare il target in ciascun dominio.

Example of SOT: MDNet

When a test sequence is given:

- all the existing branches of binary classification layers used in the training phase are **removed**
- **A new single branch** is constructed to compute target scores in the test sequence.
- The new classification layer and the fully connected layers within the shared layers are then **fine-tuned** online during tracking to adapt to the new domain.

The online update is conducted to model long-term and short-term appearance variations of a target for robustness and adaptiveness, respectively, and an effective and efficient hard negative mining technique is incorporated in the learning procedure.



Algorithm 1 Online tracking algorithm

```

Input : Pretrained CNN filters { $w_1, \dots, w_5$ }
Initial target state  $x_1$ 
Output: Estimated target states  $x_t^*$ 
1: Randomly initialize the last layer  $w_6$ .
2: Train a bounding box regression model.
3: Draw positive samples  $S_1^+$  and negative samples  $S_1^-$ .
4: Update { $w_4, w_5, w_6$ } using  $S_1^+$  and  $S_1^-$ ;
5:  $\mathcal{T}_s \leftarrow \{1\}$  and  $\mathcal{T}_t \leftarrow \{1\}$ .
6: repeat
7:   Draw target candidate samples  $x_t^i$ .
8:   Find the optimal target state  $x_t^*$  by Eq. (1).
9:   if  $f^+(x_t^*) > 0.5$  then
10:    Draw training samples  $S_t^+$  and  $S_t^-$ .
11:     $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{t\}$ ,  $\mathcal{T}_t \leftarrow \mathcal{T}_t \cup \{t\}$ .
12:    if  $|\mathcal{T}_s| > \tau_s$  then  $\mathcal{T}_s \leftarrow \mathcal{T}_s \setminus \{\min_{v \in \mathcal{T}_s} v\}$ .
13:    if  $|\mathcal{T}_t| > \tau_t$  then  $\mathcal{T}_t \leftarrow \mathcal{T}_t \setminus \{\min_{v \in \mathcal{T}_t} v\}$ .
14:    Adjust  $x_t^*$  using bounding box regression.
15:   if  $f^+(x_t^*) < 0.5$  then
16:     Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_s}^+$  and  $S_{v \in \mathcal{T}_s}^-$ .
17:   else if  $t \bmod 10 = 0$  then
18:     Update { $w_4, w_5, w_6$ } using  $S_{v \in \mathcal{T}_t}^+$  and  $S_{v \in \mathcal{T}_t}^-$ .
19: until end of sequence

```

MDNet e il Tracking di Oggetto Singolo (Sot)

MDNet ha influenzato molto la letteratura. Molte varianti sono state proposte, combinando anche l'apprendimento per rinforzo. Tutti questi approcci sono generalmente piuttosto lenti. Il SOT non è ancora risolto!

10.4 Tracking di Oggetti Multipli

Il tracking di oggetti multipli spesso ha poca o nessuna formazione preliminare riguardo l'apparenza e il numero di target.

Gli algoritmi MOT assegnano un ID target a ciascun riquadro di delimitazione (bbox). Questo ID target è noto come identità ed è importante perché consente al modello di distinguere tra oggetti all'interno di una classe.

La maggior parte degli algoritmi MOT incorpora un approccio chiamato tracking-by-detection. Il metodo tracking-by-detection coinvolge un rilevatore indipendente che viene applicato a tutti i frame dell'immagine per ottenere rilevamenti probabili, e quindi un tracker, che viene eseguito sul set di rilevamenti.

Il tracker tenta di eseguire l'associazione dei dati (ad esempio, collegando i rilevamenti con le identità conosciute per ottenere traiettorie complete).

Gli algoritmi di tracking in batch utilizzano informazioni da frame video futuri quando deducono l'identità di un oggetto in un certo frame. Questa metodologia risulta in una migliore qualità del tracking e può essere applicata a applicazioni di analisi video.

Gli algoritmi di tracking online utilizzano solo informazioni presenti e passate per trarre conclusioni riguardo un certo frame e generalmente performano peggio dei metodi batch. Tuttavia, questa metodologia è necessaria in applicazioni in tempo reale, come la navigazione o la guida autonoma.

- ▶ Il tracking di oggetti multipli spesso ha poca o nessuna formazione preliminare riguardo l'apparenza e il numero di target.
- ▶ Gli algoritmi MOT assegnano un ID target a ciascun riquadro di delimitazione. Questo ID target è noto come identità, ed è importante perché consente al modello di distinguere tra oggetti all'interno di una classe.
- ▶ La maggior parte degli algoritmi di tracking di oggetti multipli incorpora un approccio chiamato tracking-by-detection. Il metodo tracking-by-detection coinvolge un rilevatore indipendente che viene applicato a tutti i frame dell'immagine per ottenere rilevamenti probabili, e quindi un tracker, che viene eseguito sul set di rilevamenti.
- ▶ Il tracker tenta di eseguire l'associazione dei dati (ad esempio, collegando i rilevamenti con le identità conosciute per ottenere traiettorie complete).

Al giorno d'oggi, il rilevamento viene effettuato con rilevatori di oggetti pre-addestrati (Faster-RCNN, YOLO, DETR).

La predizione non è inclusa in tutti gli algoritmi MOT.

L'obiettivo della predizione è: data l'ultima posizione nota del target al frame k , trovare la probabile posizione al frame $k + 1$.

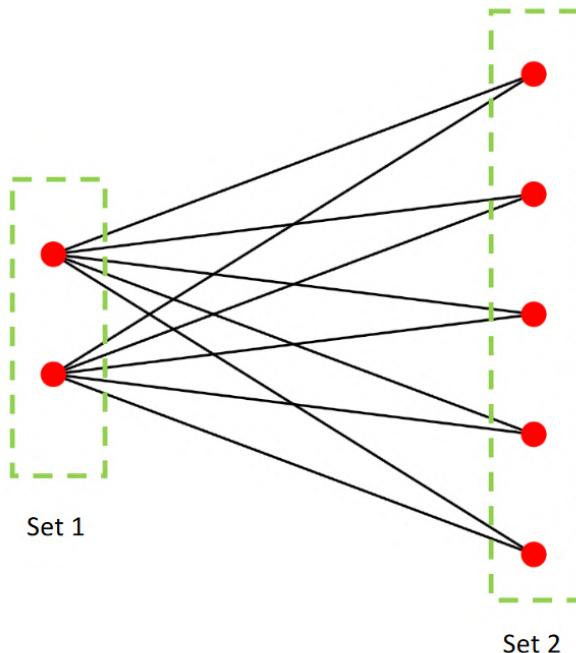
Nei metodi progettati per telecamere singole statiche, vengono utilizzati filtri di Kalman o filtri particellari.

10.5 Grafico Bipartito

Un grafico bipartito è un grafo in cui i nodi possono essere divisi in due insiemi disgiunti tali che tutti i bordi collegano un vertice in un insieme a un vertice in un altro insieme.

Non ci sono bordi tra i vertici negli insiemi disgiunti.

Nel grafo bipartito completo, ogni vertice di un insieme è connesso a ogni vertice di un altro insieme.



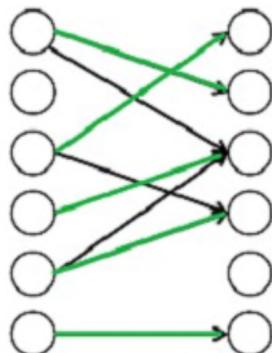
Matching in un Grafico Bipartito

Un matching in un grafico bipartito è un insieme di bordi scelti in modo tale che nessuno dei due bordi condividano un punto finale.

Un matching massimo è un matching di dimensione massima (numero massimo di bordi).

In un grafo bipartito pesato, un matching massimo pesato è un matching in cui la somma dei pesi è massimizzata.

L'algoritmo ungherese (algoritmo di Munkres) risolve questo problema di assegnazione ed è stato uno degli inizi degli algoritmi di ottimizzazione combinatoria. Una delle sue implementazioni utilizza una ricerca del percorso più breve modificata (modificato Bellman-Ford) e impiega $O(V^2E)$.



Only the green links
are in the matching

Problema di Assegnazione nel Tracking

- ▶ Conosciamo le identità (traiettorie) esistenti al tempo $t - 1$.
- ▶ Otteniamo nuovi rilevamenti sul frame al tempo t .
- ▶ Vogliamo trovare a quale traiettoria appendere ciascuno dei nuovi rilevamenti.

Costruiamo un grafo bipartito pesato:

I pesi generalmente rappresentano:

- ▶ Prossimità
- ▶ Somiglianza delle caratteristiche di apparenza
- ▶ Aggiungere nodi finti può anche tenere conto dell'ingresso/uscita.



10.6 SORT

SORT è un algoritmo di tracking online e in tempo reale per il tracking di oggetti multipli 2D in sequenze video. È una implementazione popolare di un framework MOT visivo basato su tecniche di associazione dei dati e stima dello stato. È progettato per applicazioni di tracking online dove solo i frame passati e presenti sono disponibili e il metodo produce identità degli oggetti al volo. Non gestisce l'occlusione o la riapparizione degli oggetti. La performance dipende dalla qualità del rilevamento. Utilizza fondamentalmente il filtro di Kalman e l'algoritmo ungherese.

DeepSort

SORT restituisce un numero relativamente alto di cambiamenti di identità e ha una carenza nel tracking attraverso le occlusioni. DeepSORT sostituisce la metrica di associazione con una metrica più informata che combina informazioni di movimento e apparenza. Una CNN è stata addestrata per discriminare i pedoni su un dataset di re-identificazione delle persone su larga scala. Utilizza ancora il filtro di Kalman per il passaggio di predizione. Per costruire il problema di associazione, DeepSORT combina 2 distanze: una sui riquadri di delimitazione ($d(1)$), l'altra sul descrittore di apparenza ($d(2)$). Per ogni identità, mantiene anche una memoria degli ultimi 100 descrittori. Con considerazioni statistiche, filtra le associazioni improbabili (matrice B).

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

- To build the association problem, DeepSort combines 2 distances: one on the bounding-boxes ($d^{(1)}$), the other on the appearance descriptor ($d^{(2)}$). For each identity, it also maintains a memory of the last 100 descriptors.

$$d^{(1)}(i, j) = (\mathbf{d}_j - \mathbf{y}_i)^T \mathbf{S}_i^{-1} (\mathbf{d}_j - \mathbf{y}_i),$$

$$d^{(2)}(i, j) = \min\{1 - \mathbf{r}_j^T \mathbf{r}_k^{(i)} \mid \mathbf{r}_k^{(i)} \in \mathcal{R}_i\}$$

$$c_{i,j} = \lambda d^{(1)}(i, j) + (1 - \lambda)d^{(2)}(i, j)$$

- With statistical considerations, it filters out unlikely associations (matrix B)

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \dots, N\}$, Detection indices $\mathcal{D} = \{1, \dots, M\}$, Maximum age A_{\max}

- 1: Compute cost matrix $\mathbf{C} = [c_{i,j}]$ using Eq. 5
- 2: Compute gate matrix $\mathbf{B} = [b_{i,j}]$ using Eq. 6
- 3: Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
- 4: Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
- 5: **for** $n \in \{1, \dots, A_{\max}\}$ **do**
- 6: Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
- 7: $[x_{i,j}] \leftarrow \text{min_cost_matching}(\mathbf{C}, \mathcal{T}_n, \mathcal{U})$
- 8: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
- 9: $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
- 10: **end for**
- 11: **return** \mathcal{M}, \mathcal{U}

10.7 Come valutare il nostro tracker?

Per misurare la performance del MOT dobbiamo prima considerare come vengono fornite le verità a terra. Le annotazioni sono fornite come una lista di riquadri di delimitazione dei target nella scena. Per ogni riquadro di delimitazione è assegnato un ID. Ovviamente, gli ID nelle annotazioni differiscono da quelli assegnati dal nostro tracker. Il nostro tracker può commettere errori! Il primo passo è quindi associare ogni bbox annotato alle posizioni dei nostri target... ancora una volta l'algoritmo ungherese!

Sono state proposte diverse metriche:

- ▶ Falsi negativi: Il numero totale di target mancati.
- ▶ Cambiamenti di ID: Numero di volte in cui l'identità segnalata di una traiettoria di verità a terra cambia.
- ▶ Maggiormente tracciati (MT): Percentuale di traiettorie di verità a terra che hanno la stessa etichetta per almeno l'80% della loro durata.
- ▶ Maggiormente persi (ML): Percentuale di traiettorie di verità a terra che vengono tracciate per al massimo il 20% della loro durata.
- ▶ Frammentazione (FM): Numero di volte in cui una traiettoria è interrotta da un rilevamento mancante.

Altre metriche proposte:

- ▶ Precisione del tracking multi-oggetto (MOTP): Riassunto della precisione complessiva del tracking in termini di sovrapposizione dei bounding box tra verità a terra e posizione segnalata.
- ▶ Accuratezza del tracking multi-oggetto (MOTA): Riassunto dell'accuratezza complessiva del tracking in termini di falsi positivi, falsi negativi e cambi di identità.
- ▶ ID F1 Score: Il rapporto tra rilevazioni correttamente identificate e il numero medio di rilevazioni di verità a terra e rilevazioni calcolate.
- ▶ HOTA: Higher Order Tracking Accuracy, media geometrica tra accuratezza di rilevamento e accuratezza di associazione. Media su soglie di localizzazione.

Esempio di tabella per riportare i risultati sperimentali

		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	71.0	80.2	46.9%	21.9%	434	587	7880	44564	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	406	642	5119	63352	3 Hz
EAMTT [20]	ONLINE	52.5	78.8	19.0%	34.9%	910	1321	4407	81223	12 Hz
POI [16]*	ONLINE	66.1	79.5	34.0%	20.8%	805	3093	5061	55914	10 Hz
SORT [12]*	ONLINE	59.8	79.6	25.4%	22.7%	1423	1835	8698	63245	60 Hz
Deep SORT (Ours)*	ONLINE	61.4	79.1	32.8%	18.2%	781	2008	12852	56668	40 Hz