UNIVERSITA' POLITECNICA DELLE MARCHE

Facolta' di Ingegneria

Dottorato in Ingegneria dell'Informazione
Curriculum Elettronica, Biomedica e dell'Automazione
XXXIV Ciclo

Sviluppo ed applicazione di tecniche di compressione per l'implementazione di algoritmi in Sistemi Embedded

Development and application of compression techniques for algorithms implementation in Embedded Systems

Dottorando
Manoni Lorenzo

Tutor
Prof. Turchetti Claudio

Anno Accademico 2020/2021

Abstract

Questa tesi di dottorato presenta lo sviluppo ed applicazione di tecniche di compressione per implementazione di algoritmi in sistemi embedded per i quali, viste le limitate risorse di memoria e di calcolo, è spesso problematica la trasmissione/ricezione di elevate quantità di dati, nonché implementazione di algoritmi e di modelli.

Come primo progetto di ricerca, è stato effettuato uno studio comparativo dei principali algoritmi di ricostruzione per il Compressed Sensing applicati al segnale elettromiografico, allo scopo di fornire linee guida nella scelta del miglior algoritmo nel contesto delle Wireless-Body-Area-Networks dove il Compressed Sensing può essere applicato per ridurre la quantità di dati trasmessi. E' stata ottenuta una migliore qualità di ricostruzione rispetto ai risultati in letteratura forzando la sparsità del segnale, nondimeno è stato proposto un solver per il Basis Pursuit con un ridotto costo computazionale ed un trascurabile numero di prodotti.

Come seconda tematica, in risposta al problema del "Curse of Dimensionality" nell'apprendimento non supervisionato, è stata proposta una tecnica di Manifold Learning per lo sviluppo di un modello causale per biosegnali. E' stato sviluppato un algoritmo di stima della dimensione intrinseca che supera le difficoltà delle tecniche in letteratura nella gestione di un alto numero di dimensioni, nonché un metodo per l'apprendimento del modello applicandolo alla ricostruzione e generazione del segnale ottenendo per quest'ultima risultati migliori rispetto alle Generative Adversarial Networks.

Come ultima area di ricerca, sono state proposte reti neurali convoluzionali per diverse applicazioni quali identificazione del mal dell'esca, Semantic Segmentation, Pedestrian Tracking. Esse sono state ottenute tramite tecniche di decomposizione tensoriale ed implementate su sistemi embedded, ottenendo ottimi livelli di accuratezza e tempi di inferenza inferiori a quelle delle reti disponibili nello stato dell'arte.

Parole chiave: Sistemi embedded, compressed sensing, manifold learning, convolutional neural networks, decomposizioni tensoriali.

Abstract(english)

This doctoral thesis presents the development and application of compression techniques in embedded systems whose limited resources in terms of memory and computational capacity often imply issues with transmission/reception of large data quantities and practical implementation of algorithms and models.

Concerning the first research project, a comparative study of Compressed Sensing reconstruction algorithms applied to elecromyographic signal was performed with the aim of giving some guidelines for the choice of the best algorithm with reference to Compressed Sensing application to Wireless-Body-Area-Networks in which Compressed Sensing turns out to be useful to reduce the quantity of transmitted data. A better reconstruction quality than results in literature was obtained by enforcing signal sparsity, moreover an optimized solver for Basis Pursuit was proposed which requires low computational cost and a neglegible number of product operations.

In the second research topic, in response to the Curse of Dimensionality problem which concerns unsupervised learning a novel Manifold Learning technique was proposed for causal modeling of biosignals. An Intrinsic Dimension algorithm was proposed which overcomes the difficulties of the techniques available in literature when dealing with a high number of dimensions, in addition a model learning technique was developed which was later applied to signal reconstruction and generation. The latter obtained better results than Generative Adversarial Networks.

In the last research area Convolutional Neural Networks were proposed for different applications as esca disease detection, Semantic Segmentation and Pedestrian Tracking. These networks which were designed by using tensor decomposition techniques, were implemented on embedded platforms achieving very good accuracy performances in addition to the lowest values for inference time if compared to the available state-of-art-networks.

Keywords: Embedded systems, compressed sensing, manifold learning, convolutional neural networks, tensor decompositions.

Indice

Introduzione **10**

1 Compressed Sensing **13**
1.1 Teoria 13
1.2 Applicazione Compressed Sensing al segnale EMG 15
1.2.1 Motivazione 15
1.2.2 Studio effettuato 16
1.2.3 Algoritmi di ricostruzione 18
1.2.4 Risultati sperimentali 26
1.3 CS e Mixed Integer Linear Programming 31

2 Manifold Learning per biosegnali **37**
2.1 Motivazione 37
2.2 Approccio utilizzato 39
2.3 Stima della dimensione intrinseca 41
2.3.1 Stima del gradiente con regressione di Nadaraya-Watson 41
2.3.2 Esperimenti su segnali sintetici 44
2.3.3 Esperimenti su segnali reali 45
2.4 Learning del modello 48
2.4.1 Ottimizzazione ordine del polinomio 49
2.4.2 Ricostruzione di segnali sinusoidali 50
2.4.3 Ricostruzione di segnali reali 52
2.5 Modello generativo 54

3 Compressione di CNN **59**
3.1 Tecniche di compressione delle Convolutional Neural Networks 59
3.2 Motivazione 60
3.3 Decomposizioni tensoriali 62
3.3.1 Decomposizione CANDECOMP/PARAFAC 62
3.3.2 Decomposizione di Tucker 63
3.3.3 Decomposizione tensoriali nelle CNN 64

3.4 Rete per identificazione esca 68
3.4.1 Descrizione del problema e stato dell'arte 68
3.4.2 Architettura proposta 70
3.4.3 Implementazione su OpenMV Cam 73
3.4.4 Risultati sperimentali 74
3.5 Semantic Segmentation 76
3.5.1 Descrizione del problema 76
3.5.2 CNN per la Semantic Segmentation dello stato dell'arte 77
3.5.3 Rete proposta: UNet-ResNet 79
3.5.4 Compressione tramite decomposizione CP 83
3.5.5 Implementazione su piattaforme embedded 86
3.6 Pedestrian Tracking 87
3.6.1 Pedestrian Tracking 87
3.6.2 CNN per la object detection dello stato dell'arte 88
3.6.3 Rete proposta basata sulla Tiny-YOLOv3-DarkNet . . 91
3.6.4 Risultati sperimentali 95

Lista delle figure

1.2.1 Esempio di frontend per applicazione del CS alla trasmissione dei dati tra sensori[1] 16
1.2.2 Enforcing della sparsità 17
1.2.3 SNR al variare dei frame ottenuti con il Basis Pursuit per le tre basi DCT,Haar,DB4[2] 26
1.2.4 SNR al variare delle iterazioni del solver l_1 per le tre basi DCT,Haar,DB4[2] 27
1.2.5 SNR al variare di K/M per vari valore del compression factor CF utilizzando la DB4 come base[2] 28
1.2.6 SNR al variare di CF per vari valore di K/M[2] 28
1.2.7 SNR al variare di K/M per diversi valori della potenza di rumore sovrapposto al segnale[2] 29
1.2.8 SNR al variare di K/M per il segnale EMG del dataset Physionet[2] 30
1.3.1 PERC & NMSE per coefficienti Binari[3] 34
1.3.2 PERC & NMSE per coefficienti CARS[3] 34
1.3.3 PERC & NMSE per coefficienti Gaussiani [3] 34
1.3.4 PERC & NMSE per coefficienti tratti da una distribuzione Uniforme[3] 35
1.3.5 PERC & NMSE per coefficienti Gaussiani applicando lo shift . 36
1.3.6 PERC & NMSE per coefficienti tratti da una distribuzione Uniforme applicando lo shift 36

2.1.1 Parametrizzazione di un Manifold[4] 38
2.2.1 Input e output di un sistema dinamico nonlineare h[4] 39
2.3.1 Alcuni frame del segnale vocale utilizzato[4] 46
2.4.1 Andamento del polinomio di Bernstein $C_\xi^m(x)$ per $m = 20$ e diversi valori di ξ[4] 49
2.4.2 Ricostruzione sinusoide con frequenza costante 51
2.4.3 Ricostruzione sinusoidi con frequenza variabile senza ottimizzazione di m 52
2.4.5 Sinusoidi ricostruite con ottimizzazione di m, $N_{\text{points}} = 2 * 10^5$ 53

2.5.1 Esempi di segnali generati per il TwoLeadECG classe 1 e warping della forma d'onda rispetto ai segnali nel training set [4] 55
2.5.3 Forme d'onda generate per il Two Lead ECG classe 2[4] 56
2.5.4 Forme d'onda generate per ECG200 classe 1[4] 57
2.5.5 Forme d'onda generate per ECG200 classe 2[4] 57

3.3.1 Decomposizione CP di un tensore di ordine 3[5] 62
3.3.2 Decomposizione di Tucker di un tensore di ordine 3[5] 64
3.3.3 Decomposizione CP per un layer convoluzionale 65
3.3.4 Decomposizione CP per il fully-connected layer 66
3.3.5 Decomposizione di Tucker per un layer convoluzionale 67
3.4.1 Alcune foglie di vite colpite dal mal dell'esca[6] 68
3.4.2 Sistema per la rilevazione dell'Esca[6] 69
3.4.3 Flowchart per il design del modello proposto[6] 70
3.4.4 Implementazione del modello su OpenMV Cam con il tool STM32Cube.AI[6] 73
3.5.1 Esempi di mappe ottenute tramite Semantic Segmentation[7] . 76
3.5.2 Architettura tradizionale U-Net 79
3.5.3 Residual block della ResNet con (a) e senza (b) downsampling. 80
3.5.4 Architettura Unet-ResNet-V1 81
3.5.5 Flowchart per la compressione della UNet-ResNet 84
3.6.1 Esempio di output della detection per il Pedestrian Tracking[8] 88
3.6.2 Schema a blocchi della Faster-R-CNN [9] 89
3.6.3 Architettura Single Shot Detector basato sulla Mobilenet . . . 89
3.6.4 Detection delle boxes per il Single Shot Detector 90
3.6.5 Architettura originale YOLO 91
3.6.6 Architettura YOLOv3-Darknet 92
3.6.7 Flowchart per il progetto della rete proposta 95

Lista delle tabelle

1.1 Confronto tra i risultati ottenuti e lo stato dell'arte 30
1.2 Confronto riassuntivo degli algoritmi 31

2.1 Valori singolari dello Jacobiano per il segnale generato dall'equazione di Duffin 44
2.2 Valori singolari dello Jacobiano stimato per i segnali generati dal sistema di Narendra 45
2.3 Valori singolari per il segnale vocale 47
2.4 Valori singolari per segnale ECG 47
2.5 Parametri per il modello generativo 55
2.6 Accuracy della classificazione del dataset ECG200 con la data augmentation 58

3.1 Architettura FR-Net 71
3.2 Dettagli sulla decomposizione dei layer della FR-Net 71
3.3 Architettura LR-Net 72
3.4 Finetuning LR-Net 73
3.5 ESCA dataset con la data augmentation 74
3.6 Confronto delle prestazioni dei modelli implementati su OpenMV Cam STM32H7 Plus 75
3.7 Confronto delle prestazioni delle reti su desktop 75
3.8 Architettura della Unet-Resnet V1 in dettaglio. Con c è indicato il numero di canali in output e con s lo stride di un blocco 80
3.9 Architettura Unet-Resnet-V2 in dettaglio 82
3.10 Architettura Unet-Resnet-V3 in dettaglio 83
3.11 Confronto tra i risultati sperimentali per le tre versioni V1,V2,V3 della UNet-ResNet 83
3.12 Parametri della compressione per la UNet-ResNet V1. 84
3.13 Parametri della compressione per la UNet-ResNet V2. 85
3.14 Parametri della compressione per la UNet-ResNet V3. 85

3.15 Confronto tra i risultati sperimentali per le tre reti compresse LR-UNet-ResNet V1,V2,V3. 85
3.16 Confronto tra la rete proposta e le reti dello stato dell'arte nella sperimentazione su Raspberry Pi 4 86
3.17 Confronto tra la rete proposta e le reti dello stato dell'arte nella sperimentazione su desktop e su GPU NVIDIA Tesla K80 87
3.18 Residual Block per la YOLOv3 92
3.19 Architettura Tiny-YOLOv3-Darknet 93
3.20 Architettura Tiny-YOLOv3-Darknet-Lite 94
3.21 Parametri della compressione della Tiny-YOLOv3-DarkNet-Lite 95
3.22 Confronto tra il modello proposto e i modelli dello stato dell'arte 97

Introduzione

I sistemi embedded sono ormai diffusissimi e utilizzati nei più svariati ambiti ed applicazioni: robotica[10], veicoli autonomi[11], monitoraggio in real-time di segnali biologici[12], riconoscimento delle attività e dei movimenti[13, 14].

E' chiaro come la tendenza degli ultimi anni sia di spostarsi dal *cloud computing* progressivamente all'*edge computing*, decentralizzando l'esecuzione dei modelli e degli algoritmi.

Spostare il calcolo in locale ed in prossimità della sorgente dei dati, approccio che trova una delle sue massime espressioni nell'Internet-of-Things(IoT), fornisce notevoli benefici in termini di privacy, congestionamento della rete, riduzione della latenza, risparmio di banda e potenza consumata.[15, 16, 17].

Trasferire i task affidati in precedenza al cloud, con enormi capacità di calcolo e storage di dati, ai nodi comporta tuttavia problemi viste le limitate risorse di questi dispositivi. In particolar modo se come nodi vengono scelti sistemi embedded i quali non possono gestire memoria, potenza consumata e carico computazionale troppo elevati.

In questo contesto le tecniche di compressione dei dati[18] e dei modelli[19, 20, 21] possono risultare fondamentali per la pratica applicabilità degli algoritmi e gestione dei dati su questi dispositivi.

Tematiche oggetto della ricerca

Nella seguente tesi di dottorato è stato effettuato lo sviluppo e implementazione di tecniche di compressione per la pratica applicazione di algoritmi su sistemi embedded in tre diversi ambiti di ricerca.

- **Compressed Sensing**

  Il primo ambito di ricerca è stato il Compressed Sensing, una tecnica di compressione che consente di ricostruire un segnale a partire da un numero di campioni inferiore a quello di partenza a patto che per il segnale esista una rappresentazione sparsa rispetto ad una opportuna base.

Lo scopo del lavoro è stato di sviluppare uno studio comparativo dei principali algoritmi di ricostruzione del Compressed Sensing applicato al segnale Elettromiografico (EMG). Il fine dello studio è stato di fornire delle linee guida per la scelta del miglior algoritmo nel contesto delle Wireless-Body-Area-Network. Il Compressed Sensing è molto utile in questo scenario per ridurre la potenza necessaria alla trasmissione dei dati.

Vi è infatti carenza di risultati in letteratura riguardo l'applicazione del Compressed Sensing al segnale EMG a causa della sua bassa comprimibilità.

Al tal proposito è stato utilizzato un approccio tramite controllo della sparsità per migliorare i risultati ottenuti dallo stato dell'arte.

- **Manifold Learning per biosegnali**

  La seconda tematica affrontata, più di natura teorica, è stata la realizzazione di un modello per biosegnali con una tecnica basata sul Manifold Learning. Tale modello è stato proposto come una possibile soluzione al cosiddetto *Curse of Dimensionality* (maledizione della dimensionalità) che affligge gli algoritmi di Unsupervised Learning (apprendimento non supervisionato).

  L'obiettivo del lavoro è stato come primo passo di sviluppare una tecnica di stima della dimensione intrinseca per un Manifold definito sulla base di un modello causale per i segnali oggetto di studio. Tale Manifold ha infatti una dimensione intrinseca troppo elevata per le tradizionali tecniche di stima della dimensionalità presenti in letteratura.

  Successivamente è stata proposta una tecnica di learning vera e propria per il Manifold, quest'ultima è stata applicata alla ricostruzione e generazione del segnale confrontando le prestazioni con quelle ottenute dallo stato dell'arte.

- **Decomposizioni tensoriali nelle reti neurali convoluzionali**

  La terza area di ricerca esplorata è stato lo sviluppo di Convolutional Neural Networks compresse tramite tecniche di decomposizione tensoriale, e loro implementazione su sistemi embedded.

  L'obiettivo della ricerca è stato di sviluppare reti dal basso dispendio computazionale che ottenessero tempi di inferenza inferiori alle reti più efficienti disponibili in letteratura una volta implementate su sistemi embedded. Nondimeno esse sono state sviluppate in modo da mantenere alti livelli di accuratezza.

Sono state sviluppate reti per la rilevazione del mal dell'esca su una OpenMV Cam, una rete per la Semantic Segmentation (segmentazione semantica) su Raspberry Pi 4 e su GPU, una rete per il Pedestrian Tracking (tracciamento dei pedoni) su Raspberry Pi 4.

Capitolo 1

Compressed Sensing

1.1 Teoria

Il Compressed Sensing(CS) è una emergente tecnica di compressione che consente di ridurre la frequenza di campionamento di un segnale sotto il limite di Nyquist mediante una trasformazione lineare applicata ad un certo numero di campioni.

Considerato un vettore $f \in \mathbb{R}^N$ contenente i campioni del segnale, esso viene compresso in un vettore $y \in \mathbb{R}^M$, $M < N$ mediante prodotto con una matrice Φ detta *matrice delle misure*:

$$y = \Phi f \tag{1.1}$$

Le condizioni teoriche per l'applicabilità del CS vengono fornite in [22] dove viene dimostrato che se il segnale ha una rappresentazione sparsa rispetto ad una opportuna base $\Psi = [\psi_1 \dots \psi_N]$:

$$f = \Psi x \tag{1.2}$$

dove x ha solo K elementi non nulli, risolvendo il seguente problema di ottimizzazione:

$$\arg\min_x ||x||_1 \qquad \text{s.t.} \quad \Phi\Psi x = y \tag{1.3}$$

è possibile ricostruire esattamente x con probabilità superiore a $1 - \delta$ se per una opportuna costante C:

$$M \geq C\mu^2(\Phi, \Psi) K \log\left(\frac{N}{\delta}\right) \tag{1.4}$$

dove $\mu^2(\Phi, \Psi) = \sqrt{N}\max_{i,k}|\phi_i^T \psi_k|$ è detta *coerenza mutua* ed è la massima correlazione tra le colonne di Φ e Ψ.

Questo risultato mostra un legame stretto tra la sparsità del segnale e la sua comprimibilità. Minore infatti è la frazione di coefficienti non nulli rispetto alla base, minore sarà il numero di campioni necessari per avere alta probabilità di esatta ricostruzione.

E' stata nondimeno dimostrata una condizione deterministica per l'esatta ricostruzione di x fondata sulla Restricted-Isometric-Property[23](RIP) della matrice $A = \Phi\Psi$. E' definita *costante isometrica* il più piccolo $\delta_K \geq 0$ tale che per qualunque vettore x K-sparso risulta:

$$(1 - \delta_K)||x||_2^2 \leq ||Ax||_2^2 \leq (1 + \delta_K)||x||_2^2 \tag{1.5}$$

In pratica la matrice A deve conservare le distanze rispetto allo spazio trasformato da una combinazione lineare delle sue colonne. Perché δ_K sia sufficientemente piccola le colonne di A devono avere una bassa correlazione tra loro. Essendo $A \in \mathbb{R}^{M \times N}$ una matrice rettangolare è ovviamente impossibile avere tutte le colonne ortogonali tra loro.

E' stato dimostrato sempre in [22] che considerato un vettore x stavolta non sparso, se $\delta_{2K} < \sqrt{2} - 1$ allora,detta $x^\star$ la soluzione del problema in 1.3.

$$||x - x^\star||_2 \leq C\frac{||x - \overline{x}_K||_2}{K} \tag{1.6}$$

dove $\overline{x}_K$ è la miglior approssimazione K-sparsa di x e C è una opportuna costante. In sostanza questo risultato mostra che l'errore di ricostruzione segue quello di approssimazione K-sparsa. Se poi $x = \overline{x}_K$ allora si ha una condizione sufficiente e deterministica per l'esatta ricostruzione.

Il problema di ottimizzazione in 1.3 è detto *Basis Pursuit* e costituisce un metodo rigoroso per la ricostruzione. Esso tuttavia richiede un costo computazionale estremamente elevato il che costituisce un limite in contesti dove le risorse di calcolo disponibili non sono elevate. Per questo motivo sono stati sviluppati algoritmi di ricostruzione meno rigorosi ma computazionalmente meno dispendiosi.

Per gli algoritmi si possono individuare le seguenti categorie principali:

- Ottimizzazione convessa. Algoritmi basati sul Basis Pursuit e sulle sue molteplici varianti. Sono basati su ottimizzazione mediante Interior-Point-Method(IPM). [24], richiedono elevato carico computazionale ma forniscono ricostruzione accurata.

- Algoritmi greedy. Cercano di identificare il set degli elementi non nulli iterativemente effettuando scelte secondo un certo criterio di ottimizzazione ad ogni iterazione. Hanno performances peggiori ma computazionalmente migliori. Un esempio è Orthogonal Matching Pursuit.

- Stima Bayesiana. Effettuano preliminarmente una stima della pdf del vettore x. I coefficienti vengono poi ricostruiti attraverso una minimizzazione di una likehood. Un esempio è il Block-Sparse-Bayesian-Learning (BSBL).

1.2 Studio comparativo del Compressed Sensing applicato al segnale EMG

1.2.1 Motivazione

Il principale scopo dell'attività di ricerca sul Compressed Sensing è stato di sviluppare uno studio comparativo dei principali algoritmi di ricostruzione per il CS applicato al segnale EMG. Il fine dello studio è stato di dare delle linee guida per la scelta del miglior algoritmo nel contesto di un'applicazione del CS nelle Wireless Body Area Networks(WBAN).

Le WBAN[25][26] sono reti di sensori per il monitoraggio di segnali biomedici utilizzati per le più svariate applicazioni: misura dello sforzo[27], controllo della postura[28], riconoscimento dei movimenti e dei gesti[29][30].

Questi dispositivi sono caratterizzati dal basso peso, costo e basso consumo di potenza dato che non possono incorporare batterie eccessivamente ingombranti.

Da questo punto di vista il CS può essere di enorme utilità in quanto consente di ridurre la quantità di dati da trasmettere da parte dei nodi della rete riducendo in questo modo la potenza consumata. Un esempio di sistema che utilizza il CS per comprimere il segnale acquisito da un sensore per poi ricostruirlo in ricezione è mostrato in Fig. 1.2.1

Il CS è stato applicato estensivamente a segnali biomedici quali Elettrocardiografico (ECG), Elettroencefalografico (EEG) ed EMG. Il segnale EMG ha mostrato di essere complessivamente meno adatto alla compressione tramite CS di ECG[31, 32, 33] ed EEG[34, 35, 36] al variare delle possibili scelte e/o ottimizzazioni della matrice delle misure e della base di rappresentazione.

In [37] viene suggerito che il CS non sia applicabile al segnale EMG dove per la matrice delle misure viene scelta una matrice di Bernoulli con $\phi_{i,j} \in \{0, 1\}$ e come base vengono utilizzate le B-spline[38].

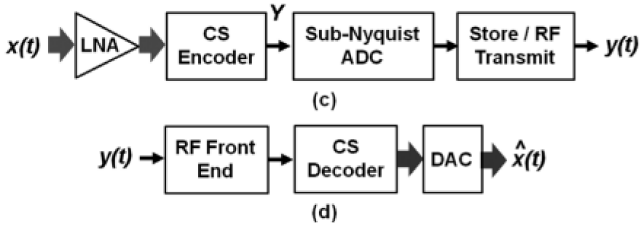


Figura 1.2.1: Esempio di frontend per applicazione del CS alla trasmissione dei dati tra sensori[1]

In [39] viene dimostrata l'efficacia del CS applicato al segnale EMG utilizzando per la ricostruzione l'algoritmo CosAmp, come base una combinazione tra la Discrete-Cosine-Transform(DCT) e le Wavelets DB4 ed una matrice di Bernoulli per l'encoding.

Un significativo miglioramento dell'errore di approssimazione è stato ottenuto da [40, 41] con una matrice di encoding deterministica, DCT come base ed un banale algoritmo dal carico computazionale molto ridotto.

E' stata sviluppata inoltre una tecnica di ottimizzazione della matrice delle misure[42] massimizzando la proiezione fatta dalle sue colonne nello spazio del segnale.

E' stato precedentemente proposto uno studio comparativo[43] del CS applicato al segnale EMG che forza la sparsità del segnale azzerando i campioni inferiori ad una determinata soglia nel dominio del tempo e della frequenza.

1.2.2 Studio effettuato

Nello studio effettuato in questo lavoro di tesi, è stato un confronto tra le basi DCT, Wavelet di Haar e DB4 selezionando quella che garantisce la ricostruzione più accurata utilizzando il Basis Pursuit.

Come matrice delle misure basandosi sulla letteratura è stata scelta una matrice di Bernoulli con elementi i.i.d. e $\phi_{i,j} \in \{1, -1\}$.

Sono stati considerati come algoritmi: Basis Pursuit, Orthogonal Matching Pursuit[44](OMP), Compressive Sensing Matching Pursuit[45](CosAmp), Normalized Iterative Hard Thresholding[46](NIHT). Per i tre algoritmi greedy considerati esistono formulazioni ed implementazioni ottimizzate che riducono al minimo la complessità computazionale.

Per quanto riguarda il Basis Pursuit è stato proposta una tecnica di soluzione che riduce la complessità rispetto ai tradizionali solver, in quanto richiede solamente somme ed un numero trascurabile di prodotti.

In maniera simile a quanto proposto in [43] è stata forzata la sparsità del segnale come preprocessing prima di eseguire la codifica nel vettore delle misure.

Ciò è stato effettuato nel dominio della base Ψ selezionando le K componenti principali come mostrato in Fig. 1.2.2.

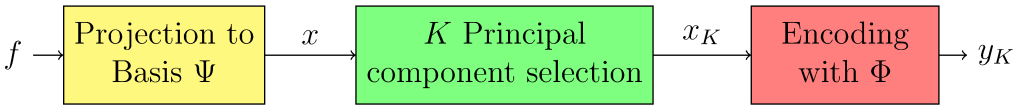


Figura 1.2.2: Enforcing della sparsità

dove il set delle componenti principali è definito dal seguente operatore:

$$\Lambda = \underset{K,\Psi}{\text{supp}}(x) \tag{1.7}$$
$$|\Lambda| = K,\ |x_i|||\Psi_i||_2 > |x_j|||\Psi_j||_2 \ \forall\ i \in \Lambda, j \notin \Lambda$$

In pratica vengono scelte le K componenti con il più alto $|x_i|||\Psi_i||_2$, esse infatti sono quelle che contribuiscono maggiormente all'energia del segnale.

Definiamo questa operazione per mezzo dell'operatore $F(\cdot, \Lambda)$:

$$x_K = F(x, \Lambda_K)$$
$$F(x,\Lambda)_\Lambda = x_\Lambda, \quad F(x,\Lambda)_{I_N \setminus \Lambda} = 0 \tag{1.8}$$

L'errore di ricostruzione tra f ed $f^\star$ può essere misurato tramite la PRD (Percentage Root-mean-squared Difference) così definito:

$$PRD = \frac{||f - f^\star||_2}{||f||_2} \tag{1.9}$$

A differenza di quanto fatto in [43], si è scelto di forzare la sparsità allo scopo di ottenere un significativo miglioramento delle performances di ricostruzione.

Detto infatti $f_K = \Psi x_K$ il segnale formato dalle sue K componenti principali, per la disuguaglianza triangolare si può scrivere:

$$PRD = \frac{||f - f_K + f_K - f^\star||_2}{||f||_2} \leq$$
$$\leq \frac{||f - f_K||_2}{||f||_2} + \frac{||f_K - f^\star||_2}{||f_K||_2}\frac{||f_K||_2}{||f^\star||_2} =$$
$$= PRD_K + PRD_{\text{rec}}\frac{||f_K||_2}{||f^\star||_2} \tag{1.10}$$

Il primo termine nella 1.10 rappresenta l'errore tra il segnale f e la sua approssimazione K-sparsa, il secondo l'errore tra l'approssimazione f_K e quello ricostruito.

Nella sperimentazione in [43] viene unicamente misurata PRD_{rec} ignorando invece la fedeltà col segnale originale f e scegliendo il livello di sparsità K/N in modo da avere esatta ricostruzione.

Mentre si avrà ovviamente una diminuzione di PRD_K all'aumentare di K, si otterrà invece un aumento di PRD_{rec} avendo gli algoritmi performances peggiori con numero più alto di componenti.

Globalmente vi è dunque quindi un minimo per PRD al variare di K, trovato il quale è possibile ottimizzare l'errore complessivo.

1.2.3 Algoritmi di ricostruzione

Basis Pursuit

Il problema di ottimizzazione in norma l_1 in 1.3 è stato ampiamente studiato e per la sua soluzione esistono numerosi tecniche disponibili:

- Tecniche basate sul Interior-Point-Method[24].
- Solver basati sul metodo del simplesso[47]. Esso si può infatti facilmente esprimere come un problema di programmazione lineare (LP).
- Solver ottimizzati specificatamente per questo problema quali SPGL1[48], YALL1[49],l1-magic[50].

Queste tecniche garantiscono un'alta accuratezza della soluzione trovata a prezzo di un alto costo computazionale.

E' stato proposto un solver che riduce la complessità a prezzo di una minor accuratezza della soluzione. In particolare esso richiede un numero trascurabile di prodotti rispetto alle somme, che in un sistema embedded può risultare un vantaggio.

Algorithm 1 Basis Pursuit solver

Input: $A = \Phi\Psi,\ y,\ K$

Inizialize:

$$\begin{cases} P = [p_1 \dots p_N],\ P = I - A^\dagger A, \\ x_0 = A^\dagger y \quad \textit{// Compute with QR factorization} \\ t = 0 \end{cases}$$

Output: K-sparse coefficient vector x

while $t <$ max_iter **do**

$s_t = \text{sign}(x_t),\ w_t = (s_t + 1)/2$ *// reduce s_t to binary vector w_t*

if $t > 0$ **then**

$v_t = w_t \oplus w_{t-1}$

$\Omega_t = \{j / v_t(j) = 1\}$ *// define the changing elements set Ω_t from s_{t-1} to s_t*

$q_t = q_{t-1} + 2 \sum_{j \in \Omega_t} p_j s_t(j)$

else

$q_t = P s_0$

$\mu = \dfrac{\epsilon_{tol} N}{\|q_0\|_2}$

end if

$x_{t+1} = x_t - \mu \dfrac{||x_t||_1}{N} q_t$

$t = t + 1$

end while

$\Lambda_K = \underset{K,\Psi}{\text{supp}}(x)$

$x \leftarrow F(x, \Lambda_K)$

Applicando i moltiplicatori di Lagrange al problema in 1.3 ed effettuando una discesa del gradiente abbiamo:

$$\begin{aligned} \mathcal{L}(x, \lambda) &= ||x||_1 + \lambda^T (Ax - y) \\ \frac{\partial \mathcal{L}}{\partial x} &= \text{sign}(x) + A^T \lambda \\ x_{t+1} &= x_t - \mu \left. \frac{\partial \mathcal{L}}{\partial x} \right|_{x_t} \end{aligned} \tag{1.11}$$

dove sign$(\cdot)$ è la funzione segno.

Imponendo il vincolo $Ax_t = y$ possiamo ricavare il moltiplicatore λ:

$$\lambda = -(AA^T)^{-1}A\text{sign}(x_t) \tag{1.12}$$

Sostituendo nella 1.11 otteniamo l'equazione di aggiornamento di x_t:

$$\begin{aligned} x_{t+1} &= x_t - \mu\left(I - A^T(AA^T)^{-1}A\right)\text{sign}(x_t) \\ x_{t+1} &= x_t - \mu P\text{sign}(x_t) \end{aligned} \tag{1.13}$$

dove detta $A^\dagger$ la pseudo-inversa di A, risulta $P = I - A^\dagger A$.

Dato che $s_t = \text{sign}(x_t)$ ha elementi $\in \{\pm 1\}$ per il calcolo di $q_t = Ps_t$ sono sufficienti somme e differenze delle colonne di P.

Inoltre l'aggiornamento di q_t può essere fatto sulla base del set di elementi s_t che variano rispetto a s_{t-1}. Si può facilmente verificare che:

$$\begin{aligned} w_t &= (s_t + 1)/2 \\ v_t &= w_t \oplus w_{t-1} \\ \Omega_t &= \{j \big| v_t(j) = 1\} \\ q_t = q_{t-1} &= 2\sum_{j\in\Omega_t} p_j v_t(j) \end{aligned} \tag{1.14}$$

L'aggiornamento di q_t ha un upper-bound di $\mathcal{O}(N^2)$ somme.

Per quanto riguarda l'inizializzazione x_0 un buon punto di partenza è la soluzione del problema in norma l_2:

$$\begin{aligned} x_0 &= \arg\min_x ||x||_2^2 \qquad \text{s.t.} \quad Ax = y \\ x_0 &= A^\dagger y \end{aligned} \tag{1.15}$$

Il calcolo della 1.15 è stato implementato in maniera ottimizzata tramite fattorizzazione QR[51] con complessità pari a $\mathcal{O}(MN)$.

Lo stepsize μ è stato normalizzato per la norma del segnale per uniformare la convergenza, l'equazione di aggiornamento diventa dunque:

$$x_{t+1} = x_t - \mu\frac{||x_t||_1}{N}q_t \tag{1.16}$$

Il valore di μ è stato scelto in modo da avere una variazione relativa sotto una certa soglia ϵ_{tol} per $t \to \infty$:

$$\frac{||x_{t+1} - x_t||_2}{||x_t||_2} < \epsilon_{tol} \quad t \to \infty$$
$$\mu \leq \frac{\epsilon_{tol} N}{||P\text{sign}(x_\infty)||_2} \frac{||x_\infty||_2}{||x_\infty||_1} \tag{1.17}$$

Visto che $||x_\infty||_{1,2} < ||x_0||_{1,2}$, la 1.17 si può ridurre ad una condizione più pratica:

$$\mu \leq \frac{\epsilon_{tol} N}{||P\text{sign}(x_0)||_2} \tag{1.18}$$

Per finire vengono mantenute solamente le K componenti principali $\Lambda_K = \text{supp}(x, K)$.

Lo pseudo-codice dell'algoritmo così proposto è riportato nell'Algorithm 1.

Orthogonal Matching Pursuit (OMP)

L'OMP[52, 44, 53] è uno dei principali algoritmi greedy e cerca di identificare il set Λ degli elementi non nulli del vettore K-sparso x con una procedura iterativa.

Partendo da un set vuoto, ad ogni iterazione $t \leq K$ viene aggiunta la colonna a_j che ha la correlazione più alta con il residuo $r_{t-1} = y - Ax_{t-1}$.

Il segnale x_t viene stimato minimizzando l'errore di approssimazione:

$$x_t = \arg\min_x ||y - Ax||_2^2$$
$$x_t = A_t^\dagger y \tag{1.19}$$

Esistono già implementazioni estremamente ottimizzate[54, 55] che riducono al minimo il carico computazionale per l'aggiornamento di x_t. Complessivamente il costo complessivo può essere ridotto a $\mathcal{O}(MNK)$ prodotti e somme. Lo pseudo-codice dell'OMP è riportato in Algorithm 2.

Algorithm 2 OMP

Input: $A = \Phi\Psi,\ y,\ K$

Initialize:

$$\begin{cases} r_0 = y & \textit{// residual} \\ A_0 = \emptyset & \textit{// columns} \\ t = 0 \end{cases}$$

Output: K-sparse coefficient vector x

while $t < K$ **do**

$\lambda_t = \arg\max_j |a_j^T r_{t-1}|$ *// find the column of A that is most correlated with the residual*

$A_t = [A_{t-1}\, a_{\lambda_t}]$ *// merge the new column*

$x_t = A_t^\dagger y$ *// find the best coefficients x_t from (1.19)*

$r_t = y - A_t x_t$ *// update the residual*

$t = t + 1$

end while

Compressive Sampling Matching Pursuit (CosAmp)

Il CosAmp[39, 45] si basa come l'OMP sull'identificazione delle colonne di A maggiormente correlate con il residuo.

Tuttavia in questo caso le K direzioni più correlate vengono aggiunte globalmente al set stimato Λ_t:

$$W_t = \arg\max_{|W|=K} ||A_W^T r_t||_1 \tag{1.20}$$
$$T_t = W_t \cup \Lambda_t$$

Viene poi risolto il problema di approssimazione ai minimi quadrati rispetto al set T_t:

$$z = A_{T_t}^\dagger y \tag{1.21}$$

Il set stimato viene infine aggiornato al solito selezionando le K componenti principali:

$$h_{T_t} = z \quad h_{I_N \setminus T_t} = 0 \tag{1.22}$$
$$\Lambda_{t+1} = \operatorname*{supp}_{K,\Psi}(h) \tag{1.23}$$

Anche per il CosAmp la soluzione del problema ai minimi quadrati è stata implementata in maniera ottimizzata. In particolare in[45] viene utilizzata l'iterazione di Richardson[56] riducendo il costo computazionale dell'algoritmo a $\mathcal{O}(N \log^2 N)$.

Lo pseudo-codice del CosAmp è riportato in Algorithm 3.

Algorithm 3 CoSaMP

Input: $A = \Phi\Psi,\ y,\ K$

Initialize:

$$\begin{cases} r_0 = y \quad \textit{// residual} \\ t = 0 \end{cases}$$

$\Lambda_0 = \underset{|\Lambda|=K}{\arg\max} \|A_\Lambda^T r_0\|_1$ *// find k columns of* A^T *that are most strongly correlated with residual* r_0

Output: K-sparse coefficient vector x

while $t <$ max_iter **do**

$W_t = \underset{|W|=K}{\arg\max} \|A_W^T r_t\|_1$ *// find K columns of* A^T *that are most correlated with residual* r_t

$T_t = \Lambda_t \cup W_t$ *// merge the new columns such that* $|T_t| = 2K$

$z = A_{T_t}^{\dagger} y$ *// find the best coefficients for residual approximation*

$h_{T_t} = z \quad h_{I_N \setminus T_t} = 0$

$\Lambda_{t+1} = \underset{k,\Psi}{\text{supp}}(h)$ *// find the set of sparsity* Λ_{t+1}

$x = F(h, \Lambda_{t+1})$ *// find sparse vector x*

$r_{t+1} = y - A_{\Lambda_{t+1}} x_{\Lambda_{t+1}}$ *// update the residual*

$t = t + 1$

end while

Normalized Iterative Hard Thresholding (NIHT)

L'idea di base del NIHT[46, 57] è di effettuare una discesa del gradiente del residuo $r_t = Ax_t - y$:

$$\begin{aligned} \widetilde{x}_{t+1} &= x_t - \mu \frac{\partial ||r_t||_2^2}{\partial x} \\ \widetilde{x}_{t+1} &= x_t - \mu A^T (Ax_t - y) \end{aligned} \tag{1.24}$$

Contemporaneamente ad ogni iterazione vengono mantenute al solito le K componenti principali:

$$\begin{aligned} \Lambda_{t+1} &= \underset{K,\Psi}{\text{supp}}(\widetilde{x}_{t+1}) \\ x_{t+1} &= F(\widetilde{x}_{t+1}, \Lambda_{t+1}) \end{aligned} \tag{1.25}$$

Nella versione tradizionale[46] dell'algoritmo x_t viene aggiornato semplicemente tramite 1.24 e 1.25 fino ad convergenza o a un numero massimo di iterazioni.

Sono state apportate tuttavia migliorie all'algoritmo in [57] con un'ottimizzazione dello stepsize μ_t minimizzando il modulo del residuo:

$$\mu_t = \arg\min_{\mu} ||Ax_{t+1} - y||_2^2 \tag{1.26}$$

L'incoveniente di quest'espressione è che non può essere risolta in forma chiusa in quanto x_{t+1} è ottenuto tramite l'hard-thresholding. In [57, 58] la minimizzazione viene fatta supponendo che il supporto non vari: $\Lambda^{\star}_{t+1} = \Lambda_t$.

Detto q_t il gradiente del residuo si ha:

$$\begin{aligned} \frac{\partial ||A_{\Lambda_t} x_{\Lambda_t} - y - \mu A_{\Lambda_t} q_t||_2^2}{\partial \mu} &= 0 \\ \mu_t = \frac{q^T_{\Lambda_{t+1}} A^T_{\Lambda_{t+1}} (A x_{\Lambda_{t+1}} - y)}{q^T_{\Lambda_{t+1}} A^T A q_{\Lambda_{t+1}}} &= \frac{q^T_{\Lambda_{t+1}} q_{\Lambda_{t+1}}}{w^T w} \end{aligned} \tag{1.27}$$

dove $w = A_{\Lambda_{t+1}} q_{\Lambda_{t+1}}$.

In questo lavoro è stato utilizzato un metodo iterativo per la stima del supporto tramite ottimizzazione di μ_t e riaggiornamento di Λ_{t+1} al valore effettivo:

$$\begin{aligned} &\text{Compute } \mu_t \text{ with 1.27} \\ &\Lambda_{t+1} \leftarrow \underset{K,\Psi}{\text{supp}}(x_t - \mu_t q_t) \end{aligned} \tag{1.28}$$

E' stata inoltre effettuata una normalizzazione del gradiente per ottenere stabilità dato che tra le basi utilizzate vi è la DB4 per la quale $||\Psi_1||_2 \neq ||\Psi_2||_2 \ldots \neq ||\Psi_N||_2$ e visto che la teoria del NIHT in letteratura è basata sull'ipotesi che $\Psi^T \Psi = I$.

Algorithm 4 NIHT

Input: $A = \Phi\Psi,\ y,\ k$

Initialize:

$$\begin{cases} \Lambda_0 = \underset{|\Lambda|=k}{\arg\max} \|A_\Lambda^T y\|_1 & \textit{// find the columns of } A^T \textit{ that are the} \\ & \textit{// most strongly correlated with residual} \\ x_0 = A_{\Lambda_0}^\dagger y & \textit{// find the best coefficients for residual approximation} \\ t = 0 \end{cases}$$

Output: k-sparse coefficient vector x

while $t < N_{iter}$ **do**

$r_t = A_{\Lambda_t} x_t - y$ *// update residual*

$\rho = \min_j \|a_j\|_2 \left(\frac{1}{\|a_1\|_2}, \dots, \frac{1}{\|a_N\|_2} \right)$ *// step size vector*

$q_t = \rho \odot (A_{\Lambda_t}^T\, r_t)$ *// normalized gradient vector*

$\Lambda_{t+1} = \Lambda_{t+1}^\star$ *// initialize the set of sparsity* Λ_{t+1}

if $t > 0$ **then**

while (stop criterion on Λ_{t+1}) **do**

$\tilde{x}_{t+1} = x_t - \mu_t(\Lambda_{t+1}) q_t$ *// update* x_t *with step size* μ_t *given by (1.27)*

$\Lambda_{t+1} = \underset{k,\Psi}{\operatorname{supp}}\,(\tilde{x}_{t+1})$ *// update set of sparsity* Λ_{t+1}

$x_{t+1} = F(\tilde{x}_{t+1}, \Lambda_{t+1})$ *// find sparse vector* x_{t+1}

end while

else

$\tilde{x}_{t+1} = x_t - \mu_t(I_N) q_t$

$[x_{t+1}]_k = F(\tilde{x}_{t+1}, \underset{k,\Psi}{\operatorname{supp}}(\tilde{x}_{t+1}))$ *// find sparse vector* x_{t+1}

end if

$t = t + 1$

end while

$$q_t = \rho \odot A_{\Lambda_t}^T (A_{\Lambda_t} x_t - y) \tag{1.29}$$
$$\rho = \min_j ||a_j||_2 \left(\frac{1}{||a_1||_2} \dots \frac{1}{||a_N||_2} \right)$$

Lo pseudo-codice riassuntivo del NIHT sperimentato è riportato in Algorithm 4.

1.2.4 Risultati sperimentali

Esperimento A

Gli algoritmi presentati sono stati applicati al dataset del segnale EMG acquisito ed utilizzato in [59, 60] per la classificazione delle attività motorie. I segnali sono stati prelevati dai muscoli Bicipite, Deltoide e Tricipite congiuntamente al segnale accelerometrico che non è stato utilizzato in questa sperimentazione. Il segnale è stato filtrato da un filtro passa-basso a 5 Hz e successivamente da un passa-basso a 500 Hz prima di essere a campionato con una frequenza di 2 KHz.

E' stato dapprima fatto un confronto tra le basi DCT,Haar e DB4 utilizzando il Basis Pursuit in norma l_1 per la ricostruzione, scegliendo una lunghezza del frame $N = 1024$.

E' stata utilizzata come metrica l'SNR (Signal-to-noise-ratio) in dB definito come l'inverso della PRD:

$$SNR = 20 \log_{10} \frac{||f||_2}{||f - f^\star||_2}$$

In Fig. 1.2.3 è riportato l'SNR in dB per il segnale prelevato dai 3 muscoli ottenuto con un fattore di compressione $M/N = 0.5$ con $K = 281$. E' evidente come la DB4 offra di gran lunga le migliori prestazioni rispetto a DCT ed Haar.

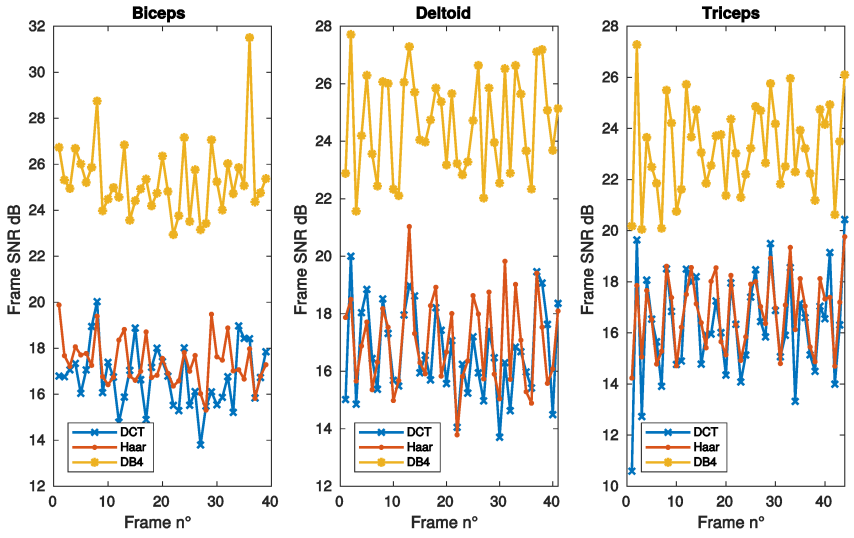


Figura 1.2.3: SNR al variare dei frame ottenuti con il Basis Pursuit per le tre basi DCT,Haar,DB4[2]

In Fig. 1.2.4 riporta l'SNR al variare delle iterazioni del solver l_1 proposto mostrando la sua convergenza.

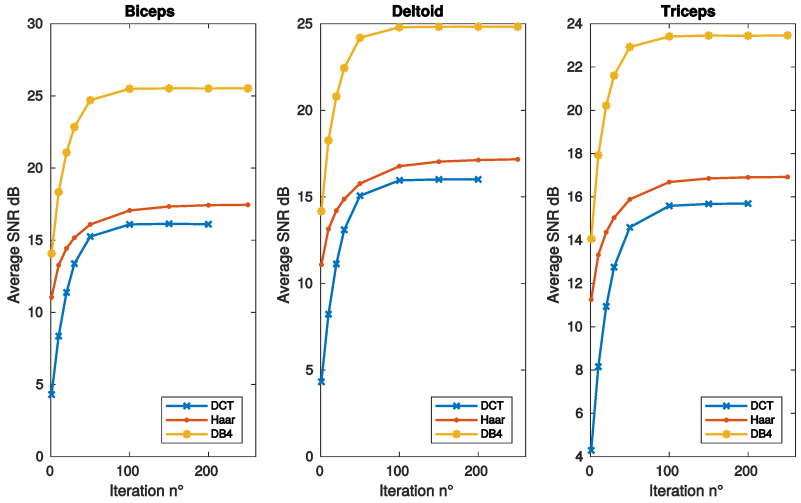


Figura 1.2.4: SNR al variare delle iterazioni del solver l_1 per le tre basi DCT,Haar,DB4[2]

Per quanto riguarda il confronto tra gli algoritmi la Fig. 1.2.5 riporta l'SNR dei quattro algoritmi L_1,OMP,CosAmp,NIHT al variare della sparsità espressa come $S_M = K/M$ per diversi fattori di compressione M/N.

Tutti gli algoritmi mostrano un picco dell'SNR intorno a $K/M = 0.4 \div 0.5$ che corrisponde ad un minimo globale della PRD già anticipato nella sottosez. 1.2.1.

L_1,OMP e CosAmp raggiungono elevati valori di SNR nel massimo, il quale può essere utilizzato per ottimizzare le prestazioni di ricostruzione.

Per quanto riguarda il CosAmp, pur ottenendo prestazioni molto vicine a L_1 e OMP nel picco, esso mostra un crollo delle prestazioni più marcato all'aumentare di K. Anche il NIHT ha un rapido degrado all'aumentare di K e le sue prestazioni sono in assoluto le peggiori.

La Fig. 1.2.6 mostra l'accuratezza della ricostruzione al variare del fattore di compressione CF per diversi valori di K/M.

E' stato inoltre condotta una sperimentazione considerando un rumore n sovrapposto al vettore delle misure:

$$y^{\text{noisy}} = \Phi x_K + n \tag{1.30}$$

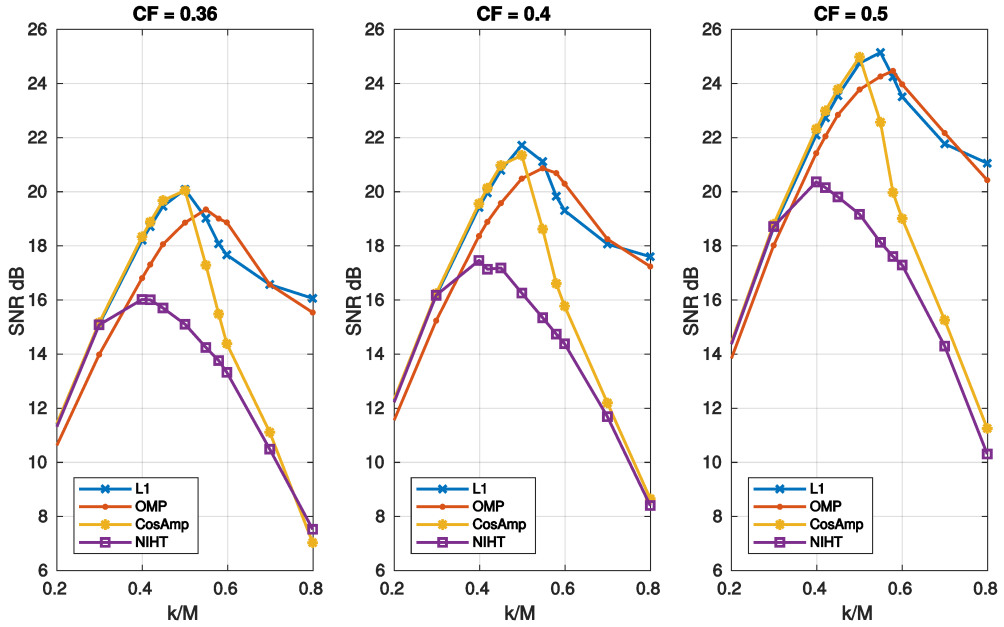


Figura 1.2.5: SNR al variare di K/M per vari valore del compression factor CF utilizzando la DB4 come base[2]

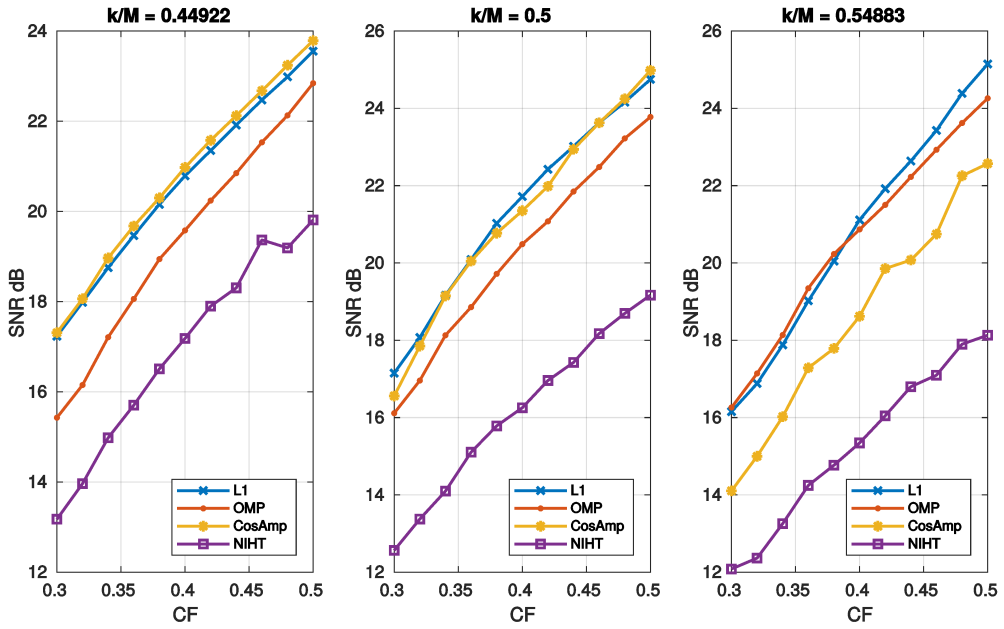


Figura 1.2.6: SNR al variare di CF per vari valore di K/M[2]

In Fig. 1.2.7 sono riportati i risultati ottenuti con $CF = 0.5$, per diversi valori della potenza di rumore così da avere un rapporto-segnale-rumore rispettivamente di 30, 25 e 20 dB.

Si può notare come L_1 per valori di K sufficientemente grandi, diventi pressoché costante all'aumentare della potenza di rumore. Si può evincere inoltre che all'aumentare della potenza di rumore l'accuratezza del NIHT degrada più lentamente rispetto a quella di OMP e CosAmp.

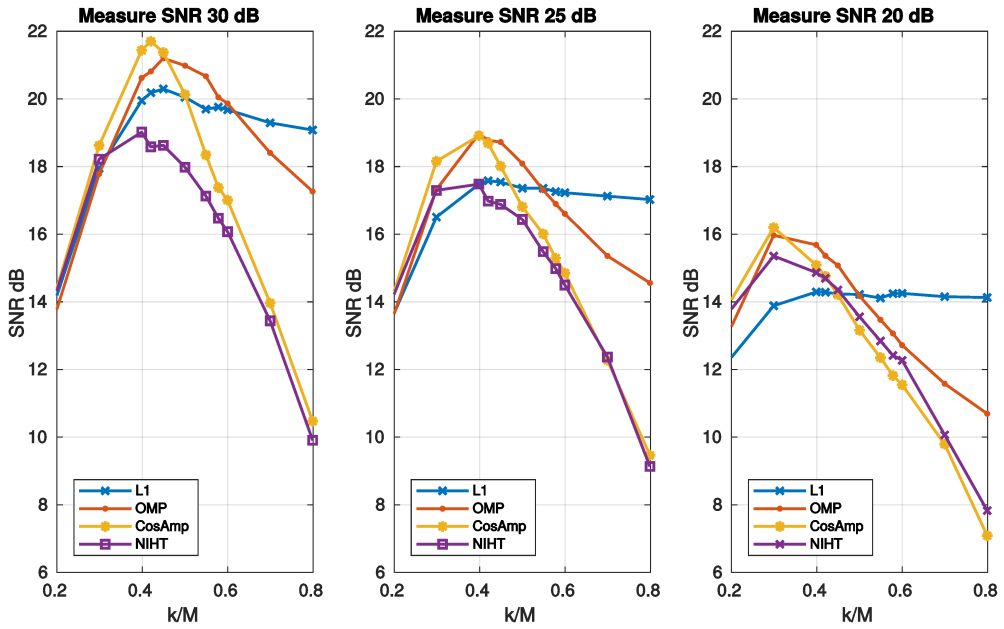


Figura 1.2.7: SNR al variare di K/M per diversi valori della potenza di rumore sovrapposto al segnale[2]

Esperimento B

E' stata eseguita una sperimentazione utilizzando il segnale EMG contenuto nel PhysioBank[61] che è un vasto repository di registrazioni di bio-segnali largamente utilizzato in letteratura.

In particolare di questo database è stata utilizzata la sezione 'Examples of Electromyograms'[62] contenente i segnali di tre soggetti rispettivamente sano, con una neuropatia e con una miopatia.

I segnali sono campionati a 4 KHz e non sono stati precedentemente filtrati. Contenendo tutte le loro componenti in frequenza la comprimibilità di questi segnali è inferiore rispetto al dataset scelto per l'esperimento A, il che mette questa sperimentazione nell'ottica worst-case.

Nondimeno il seguente dataset è stato il più utilizzato in letteratura per l'applicazione del compressed sensing al segnale EMG.

Esso pertanto è utile per confrontare i risultati dell'approccio presentato, basato sul controllo della sparsità, con lo stato dell'arte.

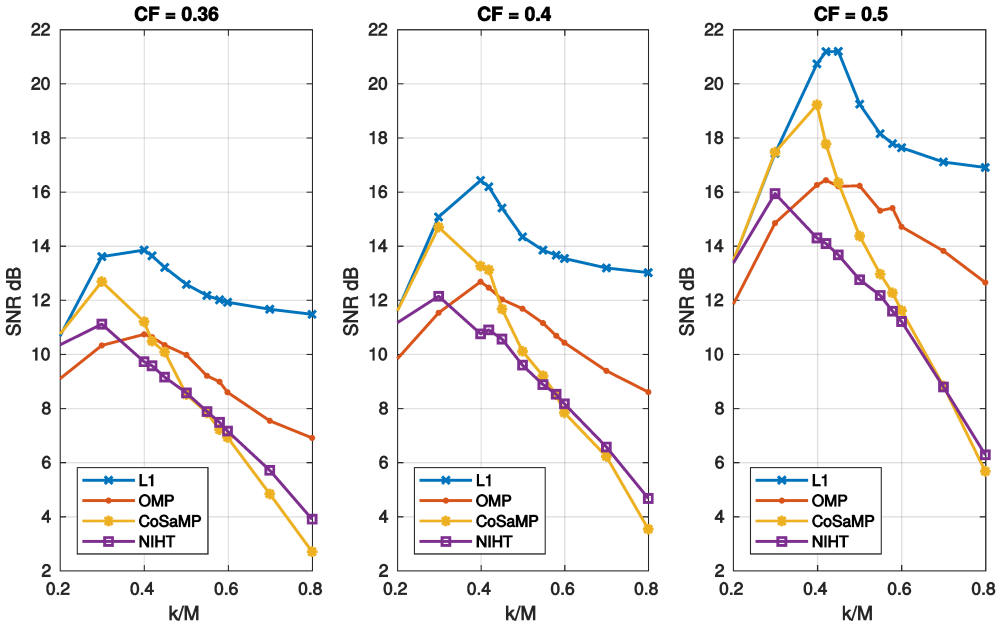


Figura 1.2.8: SNR al variare di K/M per il segnale EMG del dataset Physionet[2]

Come mostrato in Fig. 1.2.8 il CosAmp presenta sempre un degrado rapido al crescere di K pur avendo un picco superiore ad OMP e NIHT. Avendo il segnale sparsità ridotta la differenza tra le performaces di L_1 e degli algoritmi è decisamente più marcata.

	Basis Pursuit (Our work)	CosAmp (Our work)	Basis Pursuit [39]	CosAmp [39]	[40]
SNR (dB)	**16.4**	**14.7**	13.6	13.4	12.5
PRD (%)	**15.4**	**18.4**	20.3	21.3	23.7
Complexity	$\mathcal{O}(N^2K)$ sums $\mathcal{O}(MN)$ products	$\mathcal{O}(N\log^2 N)$ flops	$\mathcal{O}(N^{3.5})$ flops	$\mathcal{O}(N\log^2 N)$ flops	$\mathcal{O}(M^3)$ flops

Tabella 1.1: Confronto tra i risultati ottenuti e lo stato dell'arte

La Tab. 1.1 fornisce un confronto tra i risultati ottenuti e lo stato dell'arte del CS applicato al segnale EMG sul dataset PhysioBank.

I valori si riferiscono ad un $CF = 0.4$ e per gli algoritmi testati in questo lavoro è stato preso il massimo al variare di K/M. La tabella mostra l'approccio seguito in questo lavoro migliori l'accuratezza di ricostruzione rispetto allo stato dell'arte.

La Tab. 1.2 riporta un confronto generale tra i 4 algoritmi testati in termini di accuratezza, tolleranza al rumore e costo computazionale.

Algorithm	**Accuracy**	**Noise tolerance**	**Computational cost**
L_1	Excellent	Excellent	$\mathcal{O}(N^2K)$ sums $\mathcal{O}(MN)$ products
OMP	Good	Good	$\mathcal{O}(MNK)$ flops
CosAmp	Fair	Fair	$\mathcal{O}(N\log^2 N)$ flops
NIHT	Bad	Fair	$\mathcal{O}(MNK)$ flops

Tabella 1.2: Confronto riassuntivo degli algoritmi

1.3 Algoritmo di ricostruzione basato sulla Mixed-Integer Linear Programming

E' stata apportata una miglioria ad un algoritmo di ricostruzione per il CS proposto in [3]. L'algoritmo in questione è basato sulla Mixed-Integer-Linear-Programming(MILP) la quale si occupa della risoluzione di problemi di ottimizzazione lineari dove un sottoinsieme Ω delle variabili è vincolato ad essere intero:

$$\begin{cases} \min c^T x \\ Ax = b \\ x \geq 0 \\ x_i \in \mathbb{Z} \qquad i \in \Omega \end{cases} \tag{1.31}$$

Nell'approccio presentato in [3] viene effettuato l'encoding direttamente sul vettore K-sparso x: $y = \Phi x$.

Il problema viene formulato in maniera riconducibile alla 1.31 definendo un set di variabili binarie $z_i \in \{0, 1\},\ i = 1, \ldots N$ per identificare gli elementi non nulli.

L'algoritmo di ricostruzione dunque consiste nella soluzione del seguente problema di ottimizzazione:

$$\begin{cases} \min \sum_{i=1}^{N} z_i \\ \Phi x = y \\ c_l z_i \leq x_i \leq c_u z_i \quad i = 1, \dots N \\ \sum_{i=1}^{N} z_i \leq M/2 \\ z_i \in \{0, 1\} \qquad\quad i = 1, \dots N \end{cases} \tag{1.32}$$

dove c_l c_u sono un lower e un upper-bound per x.

E' stato aggiunto il vincolo $\sum_{i=1}^{N} z_i \leq M/2$ poiché si può dimostrare che in questo modo il problema ha un'unica soluzione che è la ricostruzione esatta di x.

Il problema in 1.32 può essere risolto tramite numerosi solver commercialmente disponibili quali AMPL[63], Gurobi[64], CPLEX.

Questi problemi vengono tipicamenti risolti con tecniche affini al Branch-and-Cut[65] che consiste nella suddivisione del problema $\mathcal{P}$ in sotto-problemi la cui soluzione è triviale oppure può essere trovata tramite il Cutting Plane. Viene creato un albero di ricerca i cui rami vengono scartati quando un sotto-problema $\mathcal{P}_j$ diventa non-feasible o la sua soluzione è subottima.

Risultati

L'algoritmo è stato applicato a 4 differenti distribuzioni random di x:

- Binario, gli elementi non nulli sono unitari

$$x_i = \begin{cases} 1 & i \in \Lambda \\ 0 & i \notin \Lambda \end{cases} \tag{1.33}$$

- Constant Amplitude Random Signs (CARS). Gli elementi hanno polarità positiva e negativa con uguale probabilità.

$$x_i = \begin{cases} \pm 1 & i \in \Lambda \\ 0 & i \notin \Lambda \end{cases} \tag{1.34}$$

- Gaussian x_i $i \in \Lambda$ sono tratti da una distribuzione Gaussiana $\mathcal{N}(0, 1)$.
- Uniform: x_i $i \in \Lambda$ sono uniformemente distribuiti in $[-1, 1]$.

I risultati sono stati confrontati con quelli ottenuti da algoritmi noti in letteratura:

- Basis Pursuit
- AstarOMP[66]. Variante dell'OMP dove le colonne di Φ vengono scelte attraverso un albero di ricerca ottimizzando un opportuno percorso.
- Iterative Hard Thresholding (IHT).
- Iterative Support Detection[67](ISD). Algoritmo iterativo che risolve il Basis Pursuit su un supporto Λ per poi riaggiornarlo.
- Smoothed-L0[68] risolve il problema NP-hard $\min||x||_0$ s.t. $\Phi x = y$ approssimando la norma l_0 con una funzione smooth: $||x||_0 \approx \sum_{i=1}^{N} \exp\left(-\frac{x_i^2}{2\sigma^2}\right)$

E' stata utilizzata una matrice Φ gaussiana con media nulla e varianza $1/N$, con $N = 256$ $M = 100$. I risultati sono stati mediati su $N_{sim} = 200$ segnali, per K tra $[10, 50]$. Per gli upper bound è stato scelto $c_u = -c_l = ||x||_\infty$.

Il problema è stato risolto con il solver IBM ILOG CPLEX Optimization Studio ed implementando il modello in linguaggio OPL.

E' stato valutato il $NMSE = ||x - x^\star||_2/||x||_2$ e la probabilità di esatta ricostruzione $PERC$ ottenuta considerando $NMSE < 10^{-2}$.

Se il solver non riesce a trovare la soluzione in $T_{max} = 100$ sec viene ritornato un vettore nullo $x^\star = 0$.

Il $NMSE$ e la $PERC$ per le 4 distribuzioni dei coefficienti sono riportate in Fig. 1.3.1, 1.3.2, 1.3.3, 1.3.4.

Per i coefficienti Binari l'algoritmo riesce sempre a trovare la soluzione $x^\star$. Per CARS e Uniform vengono ottenute le migliori prestazioni rispetto agli altri algoritmi, mentre nel caso Gaussiano ISD e SL0 hanno performances migliori.

Miglioramento proposto

Il principale svantaggio dell'algoritmo descritto è che se il solver non riesce a trovare la soluzione in un tempo accettabile non si ha disposizione alcuna stima di x.

E' essenziale pertanto avere il 100% di probabilità di ricostruzione per poter utilizzare in pratica l'algoritmo.

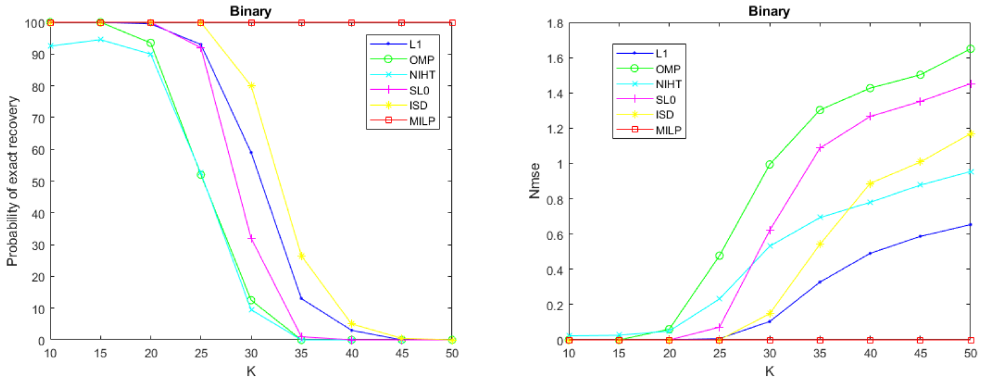


Figura 1.3.1: PERC & NMSE per coefficienti Binari[3]

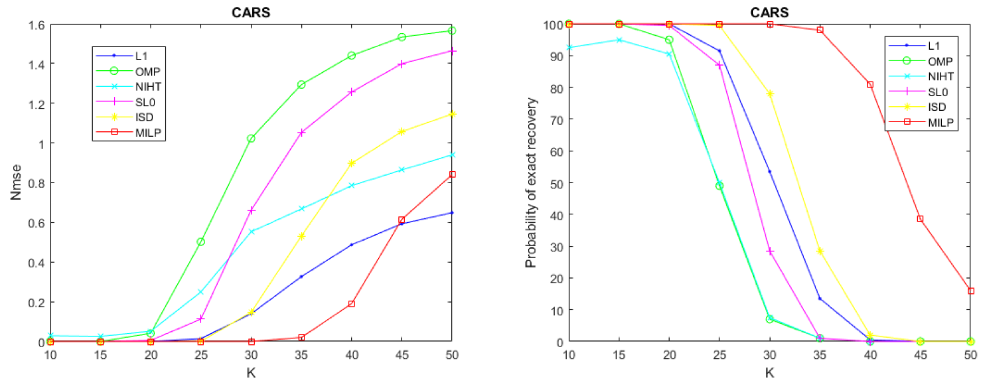


Figura 1.3.2: PERC & NMSE per coefficienti CARS[3]

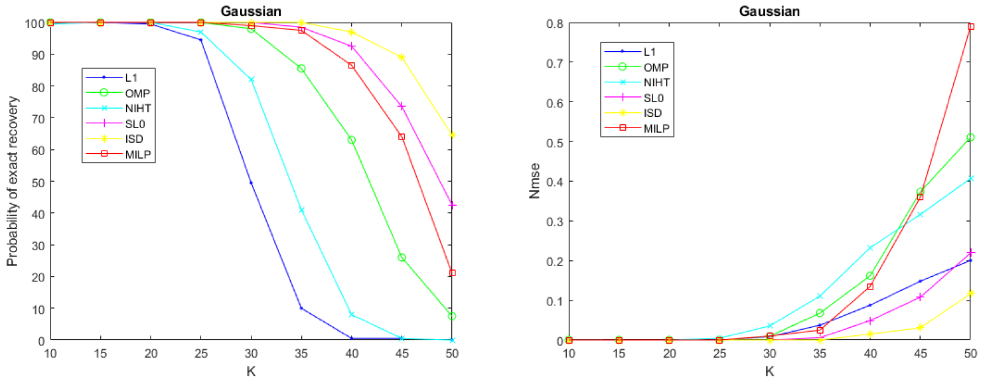


Figura 1.3.3: PERC & NMSE per coefficienti Gaussiani [3]

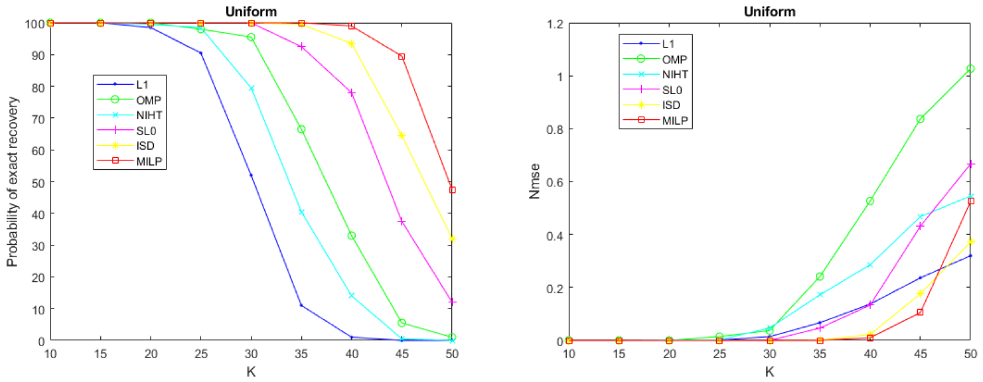


Figura 1.3.4: PERC & NMSE per coefficienti tratti da una distribuzione Uniforme[3]

Un metodo per ridurre lo spazio di ricerca della soluzione ottima può essere effettuare uno shift dei coefficienti prima della codifica di una quantità $s \geq 0$:

$$\begin{aligned} \overline{x} &= x + sv \\ v_\Lambda = 1 \; v_{I_N \setminus \Lambda} &= 0 \\ \overline{y} &= \Phi \overline{x} \\ \overline{c}_{l,u} &= c_{l,u} + s \end{aligned} \tag{1.35}$$

Il problema di ottimizzazione è stato risolto dunque rispetto alle quantità shiftate:

$$\begin{cases} \min \sum_{i=1}^{N} z_i \\ \Phi \overline{x} = \overline{y} \\ \overline{c}_l z_i \leq \overline{x}_i \leq \overline{c}_u z_i \quad i = 1, \dots N \\ \sum_{i=1}^{N} z_i \leq M/2 \\ z_i \in \{0, 1\} \qquad\quad i = 1, \dots N \end{cases} \tag{1.36}$$

In Fig. 1.3.5, 1.3.6 sono mostrati $PERC$ ed $NMSE$ ottenuti applicando uno shift s di 2,3,5,10. All'aumentare di s si ottiene il 100% di esatta ricostruzione per valori sempre maggiori di K.

Dal punto di vista pratico lo shift s non può essere arbitrariamente alto per non incorrere in problemi di overflow. Tuttavia questi risultati possono essere di notevole interesse per estendere l'applicabilità dell'algoritmo.

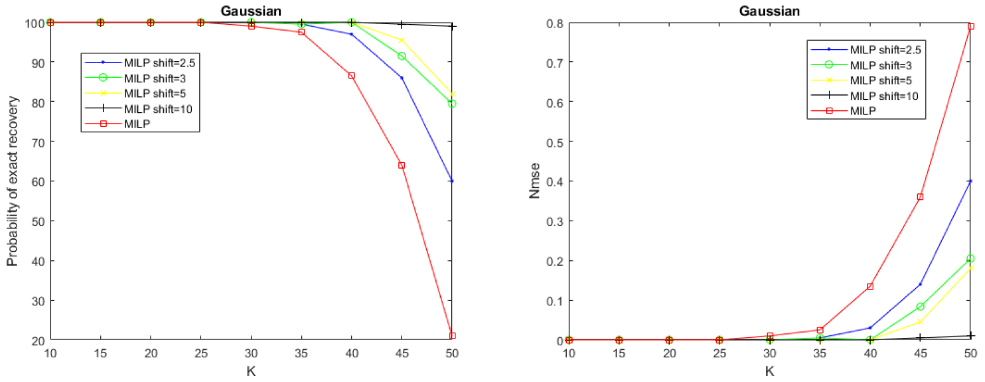


Figura 1.3.5: PERC & NMSE per coefficienti Gaussiani applicando lo shift

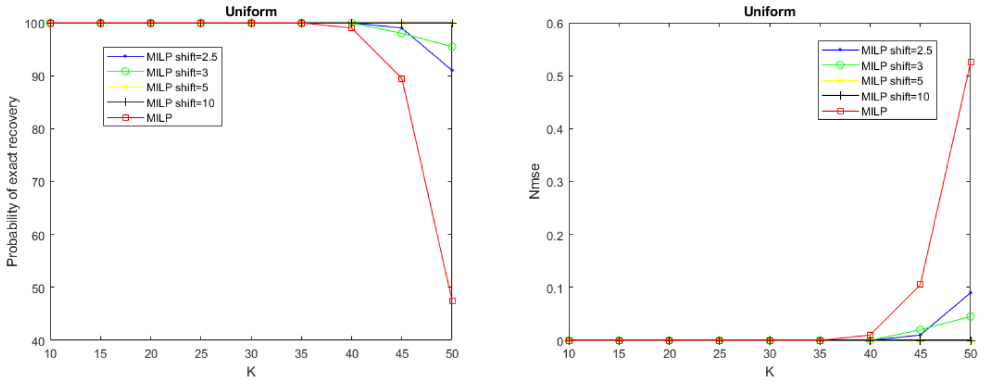


Figura 1.3.6: PERC & NMSE per coefficienti tratti da una distribuzione Uniforme applicando lo shift

Capitolo 2

Manifold Learning per il modeling di biosegnali

2.1 Motivazione

La riduzione della dimensionalità dei dati è fondamentale in moltissimi campi. Tra questi vi è il caso in cui essi sono raccolti sotto forma di serie temporale di valori come i biosegnali quali ad esempio ECG,EEG,PPG, segnale vocale,ecc.

In molte applicazioni come ricostruzione,classificazione e generazione del segnale, è infatti necessario un modello che descriva adeguatamente i dati.

Sviluppare un modello basato sulla fisica del problema in esame[69, 70] è estremamente complesso ed è possibile solamente in pochi casi particolari.

Per trovare un modello che rappresenti accuratamente i dati è possibile utilizzare tecniche di *unsupervised learning* le quali ricevono in input dati non etichettati [71] e non necessitano di una qualche misura di errore per produrre l'output desiderato.

Queste tecniche si basano essenzialmente sulla caratterizzazione dello spazio dei dati in ingresso, il quale soffre del cosiddetto *curse of dimensionality*. All'aumentare della dimensionalità dei dati infatti diventa troppo grande il numero di esempi necessari per ottenere una stima accurata.

Questo problema è ancor più rilevante nel contesto di segnali reali per i quali è necessario gestire un numero di dimensioni troppo grande per la maggior parte di questi algoritmi.

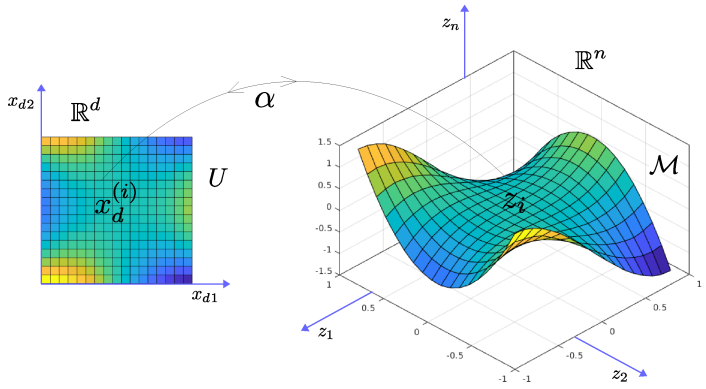


Figura 2.1.1: Parametrizzazione di un Manifold[4]

Per ridurre la dimensione dei dati, oltre alle tradizionali tecniche basate sull'analisi delle componenti principali [72, 73], vi è un'ampia letteratura sul *Manifold Learning* basato sul fatto che i dati siano embeddati in un manifold di dimensione molto minore di quella effettiva.

Uno step cruciale per il learning di un manifold è la stima della sua Dimensione Intrinseca (ID) ovvero della dimensione delle *variabili latenti* attraverso cui è possibile rappresentare il manifold.

A questo scopo esistono numerose tecniche disponibili in letteratura: locally linear embedding (LLE), isometric mapping (IM), locally multidimensional scaling (LMDS), maximum variance unfolding (MVU), local tangent space alignment (LTSA), Laplacian eigenmaps (LE), Riemannian manifold learning (RML), diffusion maps (DM), Hessian maps (HM)[74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84].

Esse tuttavia presentano problemi se la dimensione intrinseca del manifold è troppo elevata. E' stato dimostrato in [85] con segnali generati da sistemi dinamici non lineari (NLDS), anche con una dimensione intrinseca molto bassa gran parte degli algoritmi disponibili presenta errori di stima.

Lo scopo del lavoro è stato dunque di sviluppare una tecnica di manifold learning applicata ai biosegnali ed articolata nei seguenti step:

- Sviluppo di una nuova tecnica per la stima della dimensione intrinseca del manifold
- Learning del modello di parametrizzazione $\mathcal{M}(\cdot)$ tra lo spazio embeddato di dimensione d e quello a dimensione piena N.
- Applicazione del modello così ottenuto alla ricostruzione del segnale.

- Utilizzo del modello per la generazione di nuovi dati rispetto a quelli di training.

Per quanto concerne la generazione del segnale, le prestazioni del modello sono state confrontate con quelle ottenute dalle Generative Adversarial Networks (GAN)[86], reti basate su unità LSTM, estremamente efficaci ed utilizzate per la generazione e classificazione di segnali anche nell'ambito dei sistemi embedded.

2.2 Approccio utilizzato

Il modello proposto è stato sviluppato rappresentando il segnale in questione come l'uscita di un sistema dinamico nonlineare $z(t)$ sottoposto ad una eccitazione $e(t)$ non direttamente accessibile:

$$z(t) = h(e(t), e(t-1), \dots e(1)) \quad t = 1, \dots n \tag{2.1}$$

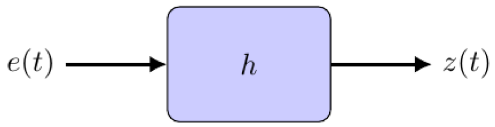


Figura 2.2.1: Input e output di un sistema dinamico nonlineare h[4]

E' stato considerato come ingresso $e(t)$ un processo stocastico stazionario a media nulla. E' ben noto per il Teorema di Wold[87] che un qualunque processo stocastico è scomponibile nella somma di una componente *lineare deterministica* $x(t)$ e in un rumore incorrelato totalmente non-deterministico $n(t)$:

$$e(t) = x(t) + n(t) \tag{2.2}$$

La componente deterministica è totalmente determinata da una combinazione lineare dei suoi d campioni passati $x(t) = L(x(t-1), x(t-2), \dots x(t-d))$. Applicando uno sviluppo in serie di Volterra per il sistema $h(\cdot)$ è possibile dimostrare dopo lunghe manipolazioni che $z(t)$ si può scrivere come la somma di tre contributi:

$$z(t) = y(t) + v(t) + \epsilon(t) \tag{2.3}$$

$$y(t) = h_x\left(x(d), \dots x(1), t\right) \tag{2.4}$$

$$v(t) = V\left(x(d), \dots x(1), n(t-1), \dots n(t-d), t\right) \tag{2.5}$$

dove il segnale $y(t)$ è classificabile come *lineare non deterministico* essendo determinato da una funzione nonlineare $h_x(\cdot, t)$ dei campioni passati di $x(t)$.

Il termine $\epsilon(t)$ dipende puramente dal rumore e non è predicibile.

Il termine $v(t)$ è invece ibrido ed è funzione sia della parte deterministica sia del rumore. Questo termine introduce aleatorietà nel segnale ed è importante in alcuni casi come le consonanti dello speech. Nei casi considerati tuttavia esso non ha un contributo significativo, per di più la sua stima è problematica e richiede tecniche di statistica piuttosto avanzate.

Per questi motivi in questo lavoro esso è stato trascurato ottenendo quindi:

$$z(t) = y(t) + \epsilon(t) \tag{2.6}$$

Ricompattando la 2.4 in forma vettoriale $z = (z(1), \dots z(n))$ si può scrivere:

$$z = \alpha(x_d) + \epsilon \tag{2.7}$$

dove $z \in \mathbb{R}^N$, $x_d \in \mathbb{R}^d$ e $d \ll N$. La 2.7 mostra che il vettore z appartiene ad un manifold di dimensione intrinseca d a meno di un rumore ϵ.

La 2.7 non è tuttavia utilizzabile in pratica dato che il vettore x_d contiene variabili latenti e non osservabili. Si può risolvere il problema esplicitando $y = (y_1(x_d), \dots y_N(x_d))$ rispetto ai campioni iniziali:

$$\begin{gathered} y = (y_1, \dots y_d, y_{d+1} \dots y_N) = (y^{''}(x_d), y^{'}(x_d)) \\ x_d = F^{-1}(y^{''}) \\ y^{'}(x_d) = y^{'}(F^{-1}(y^{''})) = G(y^{''}) \\ y = (y^{''}, G(y^{''})) \end{gathered} \tag{2.8}$$

dove si è supposto che la funzione $F(\cdot)$ sia invertibile. In questo modo è stato ottenuto un modello causale che opera direttamente sui campioni del segnale senza passare per le variabili latenti.

2.3 Algoritmo per la stima della dimensione intrinseca

Il metodo proposto per la stima della dimensione intrinseca è basato su un semplice risultato.

Considerata una partizione del vettore y:

$$y = (u, \beta(u)) = \psi(u) \tag{2.9}$$

con $u \in \mathbb{R}^p,\ p > d$, per i valori singolari dello Jacobiano $J\psi(u)$ risulta:

$$\lambda_{d+1} = \cdots = \lambda_p = 1 \tag{2.10}$$

In sostanza la dimensione intrinseca d è uguale al rango dello Jacobiano $J\psi(u)$ che sarà dato da:

$$J\psi(u) = \begin{pmatrix} I_{pp} \\ \dfrac{\partial \beta}{\partial u} \end{pmatrix} \tag{2.11}$$

2.3.1 Stima del gradiente con regressione di Nadaraya-Watson

Per la stima dello Jacobiano è stata utilizzata una tecnica basata sulla regressione di Nadaraya-Watson[88].

Consideriamo N osservazioni di un modello scalare:

$$w_k = f(u_k) + \epsilon_k \quad k = 1, \dots N \tag{2.12}$$

dove $u_k \in \mathbb{R}^p$ e dove ϵ_k è un rumore incorrelato con u con media nulla e varianza $E(\epsilon_k^2) = \sigma^2$.

La tecnica di Nadaraya-Watson è una regressione non-parametrica ed effettua la stima di $f(\cdot)$ in un punto u utilizzando un kernel $K(\cdot)$:

$$\widehat{f}(u) = \frac{\widehat{h}(u)}{\widehat{g}(u)} = \frac{\displaystyle\sum_{k=1}^{N} w_k K\left(H_k^{-1}(u - u_k)\right)}{\displaystyle\sum_{k=1}^{N} K\left(H_k^{-1}(u - u_k)\right)} \tag{2.13}$$

dove $g(u)$ denota la pdf di u, $h(u) = f(u)g(u)$ e $H_k = \mathrm{diag}(h_k^1 \dots h_k^p)$ è la larghezza di banda del kernel diversa per ogni componente.

Lo stimatore proposto in questo lavoro per il gradiente di f è stato ottenuto effettuando una derivazione della 2.13 :

$$\frac{\partial \widehat{f}(u)}{\partial u} = \frac{1}{\widehat{g}(u)} \frac{\partial \widehat{h}(u)}{\partial u} - \widehat{f}(u) \frac{\partial \widehat{g}(u)}{\partial u} \tag{2.14}$$

E' stata svolta una generalizzazione dell'analisi dell'errore della stima mostrata in [88] per lo stimatore di f. Similarmente allo stimatore di $f(\cdot)$ è facile provare che l'errore di stima $\mathcal{E}^2(u) = E\left(||\widehat{\nabla} f - \nabla f||_2^2\right)$ è formato da due contributi principali:

$$\begin{aligned} \mathcal{E}^2(u) &= \mathcal{E}_V^2(u) + \mathcal{E}_B^2(u) \\ \mathcal{E}_V^2(u) &= E\left(||\widehat{\nabla} f - E(\widehat{\nabla} f)||_2^2\right) \\ \mathcal{E}_B^2(u) &= ||E(\widehat{\nabla} f) - \nabla f||_2^2 \end{aligned} \tag{2.15}$$

Sostanzialmente $\mathcal{E}_V^2(u)$ rappresenta la varianza della stimatore mentre $\mathcal{E}_B^2(u)$ sta ad indicare il bias.

Estendendo l'analisi svolta in [88], dopo lunghi calcoli è stata trovata un'espressione analitica (vedi Appendice. 3.6.4) per $\mathcal{E}_{B,V}^2(u)$:

$$\mathcal{E}_V^2(u) = \frac{\sigma^2 \xi_2}{g(u) N^2} \sum_{k=1}^{N} \frac{\left\langle \mathrm{diag}(H_k^{-2}) \right\rangle}{D_k} \tag{2.16}$$

$$\mathcal{E}_B^2(u) = \frac{1}{N^2} \left\| \sum_{k=1}^{N} B_k(u) \right\|_2^2 \tag{2.17}$$

dove ξ_2 è una opportuna costante, $\langle \cdot \rangle$ denota la somma degli elementi di un vettore, $D_k = \det(H_k) = h_k^1 \cdots h_k^p$ e infine il termine $B_k(u) = O(H_k^2 Q_k(u))$ che ha un'andamento quadratico con la il che H_k.

Supponendo per semplicità di utilizzare un'unica larghezza di banda per tutte le direzioni e per tutti campioni $h_k^1 = \ldots h_k^p \equiv h$ l'errore $\mathcal{E}^2(u)$ diventa:

$$\mathcal{E}^2(u) = \frac{\sigma^2 \xi_2 p}{g(u) N h^{p+2}} + h^4 ||Q(u)||_2^2 \tag{2.18}$$

Il termine dovuto alla varianza $\mathcal{E}_V^2(u)$ cresce molto rapidamente per $h \to 0$ mentre il bias $\mathcal{E}_B^2(u)$ diventa dominanti all'aumentare di h.

Avendo i due termini andamento opposto $\mathcal{E}^2(u)$ avrà un minimo che può essere facilmente trovato azzerando la derivata della 2.18:

$$h^{\text{opt}}(u) = \left(\frac{\sigma^2 \xi_2 p(p+2)}{4g(u)||Q(u)||_2^2} \right)^{\frac{1}{p+6}} \tag{2.19}$$

E' stato pertanto trovato un metodo per ottimizzare la larghezza di banda h per la stima gradiente con la tecnica proposta.

Da un punto di vista implementativo la 2.19 richiede di avere una stima σ^2 e $||Q(u)||_2^2$.

La varianza di rumore si può stimare misurando l'errore della regressione fatta sui dati di training:

$$\widehat{\mathcal{E}}_V^2 = \sum_{i=1}^{N} \left(\widehat{f}_{-i}(u_i) - f(u_i) \right)^2 \widehat{g}(u_i) \tag{2.20}$$

Per quanto riguarda il bias, in maniera analoga a [88] si può scrivere:

$$E(\widehat{\nabla} f - \nabla f) \approx \frac{1}{\widehat{g}(u)} \left(\widehat{\nabla} h - \nabla h \right) - \frac{\widehat{f}(u)}{\widehat{g}(u)} \left(\widehat{\nabla} g - \nabla g \right) \tag{2.21}$$

Sfruttando il risultato mostrato in [89] si possono stimare i due termini di errore:

$$E\left(\widehat{\nabla} h - \nabla h \right) \approx \frac{1}{N} \sum_{k=1}^{N} f(u_k) L(u_k) - \widehat{\nabla} h \tag{2.22}$$

$$E\left(\widehat{\nabla} g - \nabla g \right) \approx \frac{1}{N} \sum_{k=1}^{N} L(u_k) - \widehat{\nabla} g \tag{2.23}$$

$$L(u_k) = \frac{1}{h^p} (K * \nabla K)_{h^{-1}(u-u_k)} \tag{2.24}$$

dove $*$ denota l'operatore di convoluzione. Supponendo di scegliere un kernel gaussiano si ha:

$$L(u_k) = \frac{1}{h^p} \nabla K \left(\frac{h^{-1}}{\sqrt{2}} (u - u_k) \right) \tag{2.25}$$

Scegliendo questo tipo di kernel 2.22, 2.23 si riconducono a regressioni con larghezza di banda pari a $\sqrt{2}h$.

Si può pertanto stimare la funzione $M(u) = Q(u)g(u)$ con la seguente:

$$\widehat{M}(u) = \frac{1}{s^2}\left[\left(\widehat{\nabla}h - \widehat{f}\widehat{\nabla}g\right)_{\sqrt{2}s} - \left(\widehat{\nabla}h - \widehat{f}\widehat{\nabla}g\right)_{s}\right] \tag{2.26}$$

dove s è una opportuna varianza. L'equazione per l'ottimizzazione della larghezza di banda h utilizzata in pratica diventa:

$$\widehat{h}^{\text{opt}}(u) = \left(\frac{\sigma^2 \xi_2 p(p+2)\widehat{g}(u)}{4||\widehat{M}(u)||_2^2}\right)^{\frac{1}{p+6}} \tag{2.27}$$

2.3.2 Esperimenti su segnali sintetici

Il metodo è stato inizialmente validato su segnali generati da sistemi dinamici nonlineari di dimensione intrinseca nota a priori.

Il primo esperimento che è stato condotto riguarda il segnale generato da un'equazione di Duffin:

$$y(t+1) = \frac{\Delta t^2\left[e(t) - \beta y^3(t)\right] - y(t-1) + \delta y(t)}{(1+k\Delta t)} \quad t = 1, \ldots, n \tag{2.28}$$
$$e(t) = \gamma\, x(2)\cos\left(x(1)\cdot\Delta t(t-1)\right)$$

con $\Delta t = 0.2$, $k = 0.3$, $\alpha = -4$, $\beta = 1$, $\gamma = 2$. E' stato scelto $n = 50$, un numero di punti $N = 1000$ ed una partizione con $3 \leq p \leq 12$.

In Tab. 2.1 sono riportati i valori singolari dello Jacobiano stimato dal metodo proposto.

E' evidente come solamente i primi 2 valori singolari siano significativamente diversi da 1. Il metodo pertanto stima correttamente la dimensione intrinseca $d = 2$ per il sistema.

p	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}
2	61.6595	6.2928	0	0	0	0	0	0	0	0	0	0
3	13.2458	1.7985	1.0000	0	0	0	0	0	0	0	0	0
4	2.5477	1.5152	1.0000	1.0000	0	0	0	0	0	0	0	0
5	1.6344	1.1899	1.0000	1.0000	1.0000	0	0	0	0	0	0	0
6	1.5323	1.0775	1.0000	1.0000	1.0000	1.0000	0	0	0	0	0	0
7	1.5483	1.0466	1.0000	1.0000	1.0000	1.0000	1.0000	0	0	0	0	0
8	1.4843	1.0292	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0	0	0	0
9	1.3838	1.0186	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0	0	0
10	1.3204	1.0152	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0	0
11	1.3083	1.0194	1.0002	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0
12	1.3202	1.0343	1.0003	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Tabella 2.1: Valori singolari dello Jacobiano per il segnale generato dall'equazione di Duffin

Come secondo esperimento con dati sintetici è stato preso un sistema di equazioni proposto da Narendra[90]:

$$y(t) = \frac{y(t-1)y(t-2)(y(t-1)-2.5)}{1+y^2(t-1)+y^2(t-2)} + e(t-1) \tag{2.29}$$
$$e(t) = ax(2)\sin(x(1)\omega t) + b\cos(\omega_1 x(3)) \quad t = 1, \ldots, n$$

dove $x = (x_1, x_2, x_3)$ è un vettore random uniformemente distribuito in $[-1, 1]$, $a = 2$, $b = 1.2$, $\omega = 1$, $\omega_1 = 1/3$.

E' stato utilizzato $n = 20$, un numero di punti $N = 10^6$ ed una partizione con $3 \leq p \leq 12$.

I valori singolari dello Jacobiano stimato per i segnali generati dal sistema di Narendra sono riportati in Tab. 2.2.

Anche in questo caso il metodo stima correttamente la dimensione intrinseca $d = 3$ dato che i valori singolari λ_i, $i \geq 4$ sono praticamente unitari.

p	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_10	λ_{11}	λ_{12}
3	133.5698	39.1274	6.3974	0	0	0	0	0	0	0	0	0
4	14.1856	8.8066	5.7307	1.0000	0	0	0	0	0	0	0	0
5	11.6586	3.6292	2.4125	1.0000	1.0000	0	0	0	0	0	0	0
6	9.8911	2.2837	2.1186	1.0000	1.0000	1.0000	0	0	0	0	0	0
7	7.8889	2.0632	1.5652	1.0000	1.0000	1.0000	1.0000	0	0	0	0	0
8	5.6862	1.8486	1.3928	1.0000	1.0000	1.0000	1.0000	1.0000	0	0	0	0
9	3.7562	1.6179	1.2492	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0	0	0
10	2.6113	1.5336	1.1504	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0	0
11	2.5292	1.4308	1.0843	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0
12	1.9266	1.4208	1.0590	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Tabella 2.2: Valori singolari dello Jacobiano stimato per i segnali generati dal sistema di Narendra

2.3.3 Esperimenti su segnali reali

La tecnica proposta è stata applicata a tre differenti dataset di segnali reali quali un dataset di vocali dello speech e uno di ECG.

Per lo speech è stato preso il segnale prodotto dalla vocale 'a' di uno speaker italiano, suddiviso in $N = 33567$ frame lunghi $n = 140$ e campionato a 16 KHz.

Al posto di una transizione netta dei valori singolari che si ha nei sistemi sintetici, per segnali reali si avrà una decadimento smooth.

Per stimare la dimensione d in ciascun punto u_i è stato contato il numero di valori singolari contenente una frazione trascurabile dell'energia complessiva:

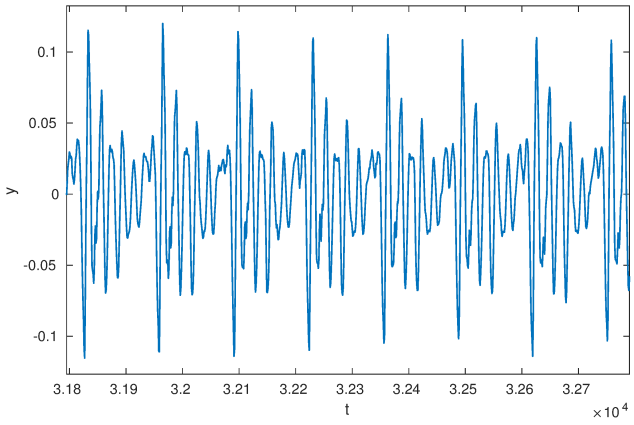


Figura 2.3.1: Alcuni frame del segnale vocale utilizzato[4]

$$\widehat{\mathrm{ID}}(u) = \min d : ||\lambda_{d+1:n}||_1 \leq \theta ||\lambda||_1 \tag{2.30}$$

dove $\theta = 10^{-5}$ nel caso delle vocali dello speech. I valori singolari per lo speech sono riportati in Tab. 2.3 dove la partizione è stata fatta variando p tra 16 e 26.

Per ottenere una stima globale è stata fatta una media pesata con la pdf:

$$\widehat{\mathrm{ID}} = \frac{\sum_{i=1}^{N} \widehat{\mathrm{ID}}(u_i)\widehat{g}(u_i)}{\sum_{i=1}^{N} \widehat{g}(u_i)} \tag{2.31}$$

In Fig. 2.3.2a, 2.3.2b sono riportati rispettivamente l'energia cumulativa residua dei valori singolari e la stima globale di d al variare della dimensione della partizione p.

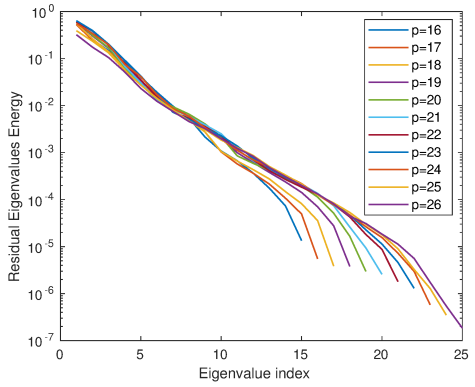
All'aumentare di p si ha una saturazione di $\widehat{\mathrm{ID}}$ verso $d = 16$ che quindi è il valore finale stimato dal metodo.

Come secondo esperimento è stato preso il segnale ECG dal UEA % UCR Time Series Classification Repository[91, 92] suddivisi in nei dataset distinti ECG200 e TwoLeadECG.

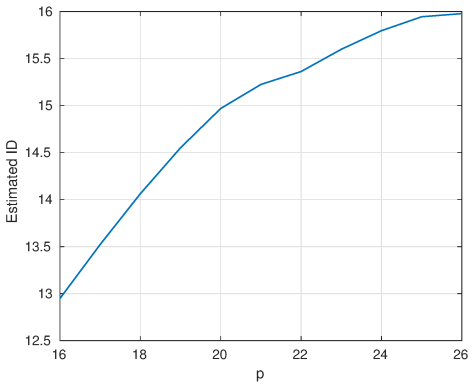
ECG200 è un dataset costituito da 200 frame lunghi 96 campioni di ECG, suddivisi nelle classi normale e infarto miocardico con 133 e 67 esempi rispettivamente. TwoLeadECG ha invece 1162 frame lunghi 82 campioni e suddivisi in due classi con 581 esempi ciascuna.

p	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	λ_{14}	λ_{15}	λ_{16}	λ_{17}	λ_{18}	λ_{19}	λ_{20}	λ_{21}	λ_{22}	λ_{23}	λ_{24}	λ_{25}	λ_{26}
16	5.0265	3.8932	2.4098	2.1121	1.4039	1.2308	1.2055	1.0425	1.0363	1.0138	1.0090	1.0043	1.0023	1.0018	1.0009	1.0005	0	0	0	0	0	0	0	0	0	0
17	5.2800	4.3764	2.5983	2.2906	1.4021	1.2540	1.1460	1.0464	1.0360	1.0289	1.0084	1.0060	1.0029	1.0017	1.0011	1.0008	1.0003	0	0	0	0	0	0	0	0	0
18	5.2192	4.7599	2.6268	2.2617	1.4672	1.2752	1.1011	1.0746	1.0397	1.0334	1.0207	1.0076	1.0056	1.0025	1.0017	1.0013	1.0008	1.0001	0	0	0	0	0	0	0	0
19	5.2595	4.5508	2.5780	2.5128	1.4871	1.3752	1.1091	1.1044	1.0402	1.0389	1.0255	1.0151	1.0056	1.0039	1.0035	1.0016	1.0011	1.0005	1.0001	0	0	0	0	0	0	0
20	5.5691	4.3462	2.8007	2.5817	1.5834	1.5598	1.1193	1.0944	1.0513	1.0407	1.0314	1.0190	1.0074	1.0060	1.0037	1.0028	1.0011	1.0006	1.0002	1.0001	0	0	0	0	0	0
21	6.0329	4.2077	3.0647	2.4333	1.8421	1.6343	1.1395	1.1148	1.0681	1.0444	1.0399	1.0168	1.0084	1.0063	1.0054	1.0038	1.0014	1.0009	1.0004	1.0002	1.0001	0	0	0	0	0
22	6.3710	4.0137	3.3214	2.3520	1.9466	1.5424	1.1858	1.0981	1.0885	1.0587	1.0361	1.0261	1.0181	1.0073	1.0055	1.0034	1.0020	1.0012	1.0004	1.0004	1.0001	1.0000	0	0	0	0
23	6.8395	3.9992	3.5059	2.3989	2.0078	1.3913	1.2162	1.1512	1.1179	1.0689	1.0422	1.0340	1.0159	1.0110	1.0048	1.0036	1.0020	1.0012	1.0005	1.0002	1.0002	1.0001	1.0000	0	0	0
24	7.4942	3.9982	3.7169	2.5576	2.1287	1.4306	1.2896	1.2027	1.1081	1.0930	1.0421	1.0356	1.0156	1.0085	1.0052	1.0041	1.0020	1.0013	1.0009	1.0005	1.0002	1.0001	1.0000	1.0000	0	0
25	7.9459	4.6569	3.7241	2.9325	2.1928	1.6229	1.2794	1.1728	1.1362	1.0622	1.0431	1.0378	1.0232	1.0165	1.0074	1.0032	1.0019	1.0014	1.0007	1.0006	1.0002	1.0002	1.0001	1.0000	1.0000	0
26	8.3517	6.1497	4.0213	3.1035	2.3893	1.6200	1.2544	1.1699	1.1357	1.0659	1.0426	1.0334	1.0151	1.0113	1.0072	1.0040	1.0016	1.0010	1.0008	1.0005	1.0004	1.0002	1.0001	1.0001	1.0000	1.0000

Tabella 2.3: Valori singolari per il segnale vocale



(a) Energia cumulativa valori singolari segnale vocale[4]



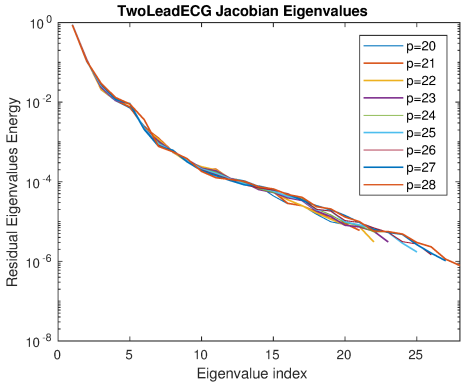
(b) Stima globale dimensione intrinseca del segnale vocale[4]

Essendo ECG200 troppo ridotto in termini di numero di punti esso non è stato utilizzato per la stima di d e si invece scelto il TwoLeadECG.

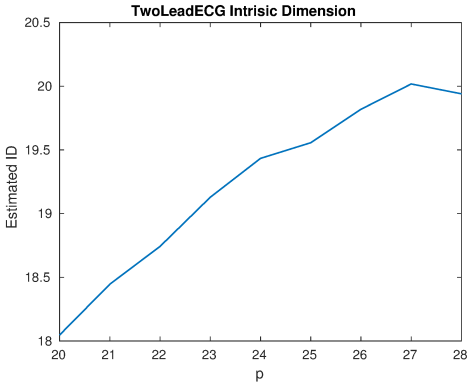
p	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	λ_{14}	λ_{15}	λ_{16}	λ_{17}	λ_{18}	λ_{19}	λ_{20}	λ_{21}	λ_{22}	λ_{23}	λ_{24}	λ_{25}	λ_{26}	λ_{27}	λ_{28}
20	8.0916	1.9855	1.1639	1.0876	1.0574	1.0205	1.0095	1.0043	1.0026	1.0018	1.0016	1.0010	1.0009	1.0006	1.0004	1.0002	1.0002	1.0001	1.0001	1.0001	0	0	0	0	0	0	0	0
21	8.0458	1.9404	1.1676	1.0901	1.0609	1.0189	1.0100	1.0046	1.0026	1.0017	1.0017	1.0009	1.0008	1.0006	1.0005	1.0002	1.0002	1.0001	1.0001	1.0001	1.0000	0	0	0	0	0	0	0
22	7.9458	1.9002	1.1670	1.0890	1.0706	1.0174	1.0093	1.0044	1.0025	1.0019	1.0016	1.0010	1.0008	1.0006	1.0004	1.0003	1.0002	1.0001	1.0001	1.0001	1.0001	1.0000	0	0	0	0	0	0
23	7.8564	1.8942	1.1841	1.0858	1.0729	1.0171	1.0081	1.0044	1.0028	1.0017	1.0015	1.0009	1.0008	1.0005	1.0004	1.0003	1.0003	1.0001	1.0001	1.0001	1.0001	1.0000	1.0000	0	0	0	0	0
24	7.7886	1.8561	1.1959	1.0896	1.0730	1.0166	1.0074	1.0044	1.0028	1.0017	1.0014	1.0009	1.0007	1.0005	1.0004	1.0004	1.0003	1.0002	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	0	0	0	0
25	7.7140	1.8390	1.2005	1.0882	1.0719	1.0169	1.0069	1.0045	1.0026	1.0016	1.0013	1.0009	1.0007	1.0005	1.0004	1.0003	1.0003	1.0002	1.0001	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	0	0	0
26	7.6306	1.8214	1.2089	1.0869	1.0702	1.0163	1.0063	1.0046	1.0023	1.0015	1.0011	1.0009	1.0006	1.0005	1.0005	1.0004	1.0003	1.0002	1.0001	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	1.0000	0	0
27	7.4641	1.7881	1.2153	1.0880	1.0676	1.0154	1.0058	1.0046	1.0023	1.0014	1.0011	1.0008	1.0006	1.0006	1.0005	1.0004	1.0003	1.0002	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0
28	7.0032	1.7502	1.2071	1.0915	1.0627	1.0255	1.0054	1.0040	1.0026	1.0013	1.0009	1.0008	1.0007	1.0005	1.0005	1.0003	1.0003	1.0002	1.0001	1.0001	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Tabella 2.4: Valori singolari per segnale ECG

Anche per il segnale ECG è stata scelta $\theta = 10^{-5}$ ottenendo una saturazione a $d = 20$ per la stima globale come mostrato in Fig. 2.3.3b.



(a) Energia cumulativa valori singolari segnale ECG[4]



(b) Stima globale dimensione intrinseca del segnale ECG[4]

2.4 Learning del modello

Una volta stimata la dimensione intrinseca è necessario effettuare il learning del modello di parametrizzazione $G(\cdot)$.

Per far ciò è stata utilizzata una tecnica di regressione simile in principio a quella di Nadaraya-Watson scelta in precedenza ma sostituendo il kernel gaussiano con i polinomi di Bernstein (PBP).

I polinomi di Bernstein[93] di ordine m in una dimensione sono definiti dalla seguente:

$$C_\xi^m(x) = \alpha_\xi^m x^\xi (1-x)^{m-\xi} \tag{2.32}$$

con $x \in [0,1]$, $\xi \in [0,m]$ e dove α_ξ^m è una opportuna costante per avere normalizzazione.

In Fig. 2.4.1 è riportato l'andamento della funzione $C_\xi^m(x)$ per alcuni valori di ξ. E' facile verificare che questi polinomi sono centrati in $x = \xi/m$.

La 2.32 viene facilmente generalizzata in d dimensioni dalla 2.33:

$$C_\xi^m(x) = \alpha_\xi^m \prod_{i=1}^{d} x_i^{\xi_i} (1-x_i)^{m-\xi_i} \tag{2.33}$$

dove $x, \xi \in \mathbb{R}^d$.

I polinomi di Bernstein sono già stati utilizzati nel Machine Learning con ottimi risultati per l'identificazione di sistemi nonlineari[94, 95].

Essendo questi polinomi sostanzialmente dei kernel centrati in $x = \xi/m$, il naturale metodo per approssimare una funzione $f(\cdot)$ in un punto $u = \xi/m$ dato il modello in un training set $[f(u_1), \ldots f(u_N)]$, è la seguente regressione:

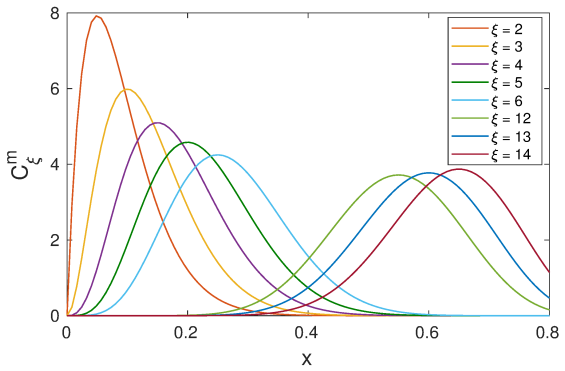


Figura 2.4.1: Andamento del polinomio di Bernstein $C_\xi^m(x)$ per $m = 20$ e diversi valori di ξ[4]

$$\widehat{f}\left(\frac{\xi}{m}\right) = \frac{\sum_{k=1}^{N} f(u_k) C_\xi^m(u_k)}{\sum_{k=1}^{N} C_\xi^m(u_k)} \tag{2.34}$$

L'ordine del polinomio m ha lo stesso ruolo della larghezza di banda H_k nella regressione di Nadaraya-Watson dato che esso controlla la varianza del kernel. Per $m \to \infty$ infatti il polinomio diventa una delta di Dirac attorno al punto $u = \xi/m$, al diminuire di m si ha invece un allargamento della campana.

2.4.1 Ottimizzazione ordine del polinomio

Analogamente alla regressione usata per la stima della dimensione intrinseca è stata proposta in questo lavoro una tecnica per l'ottimizzazione di m. Essendo in questo caso l'analisi teorica dell'errore di stima troppo complessa, è stata utilizzata una tecnica più empirica.

Dato un sottoinsieme del training set $U_{\text{opt}} = [u_1, \dots u_{\text{N}_{\text{opt}}}]$ è stato minimizzato l'errore di stima:

$$m^\star = \arg\min_m \mathcal{E}(m) \tag{2.35}$$
$$\mathcal{E}(m) = \left\| \widehat{F}(U_{\text{opt}}) - F(U_{\text{opt}}) \right\|_2^2$$

Detto $c(m) = [C_\xi^m(u_1), \dots C_\xi^m(u_N)]^T$, $f_T = [f(u_1), \dots f(u_N)]^T$, $X = [x_1^T, \dots x_N^T]$, la 2.34 può essere scritta in forma vettoriale:

$$\widehat{f}(u) = \frac{f_T^T c(m)}{\langle c(m) \rangle} \tag{2.36}$$

dove $\langle \cdot \rangle$ denota sempre la somma degli elementi di un vettore.
E' stato trovato lo zero della derivata con il metodo di Newton:

$$\frac{\partial \mathcal{E}(m)}{\partial m} = \left(\frac{\partial \widehat{F}_m}{\partial m} \right)^T (\widehat{F}_m - F) \tag{2.37}$$

$$\frac{\partial^2 \mathcal{E}(m)}{\partial m^2} = \left\| \frac{\partial \widehat{F}_m}{\partial m} \right\|_2^2 + \left(\frac{\partial^2 \widehat{F}_m}{\partial m^2} \right)^T (\widehat{F}_m - F)$$

$$m_{t+1} = m_t - \frac{\partial \mathcal{E}}{\partial m} \Big/ \frac{\partial^2 \mathcal{E}}{\partial m^2} \tag{2.38}$$

dove è stata omessa per semplicità la dipendenza da U_{opt}. E' stato utilizzato il metodo di Newton per la soluzione ottenendo per la derivata prima e seconda di $\widehat{f}(u)$ le seguenti espressioni compatte:

$$\frac{\partial^2 \widehat{f}(u)}{\partial m^2} = -2 \frac{\langle Sc(m) \rangle}{\langle c(m) \rangle} \frac{\widehat{f}(u)}{\partial m} + \tag{2.39}$$

$$+ \frac{1}{\langle c(m) \rangle} \left(f_T^T S^T Sc(m) - \widehat{f}(u) \left\langle S^T Sc(m) \right\rangle \right)$$

dove $S = \text{diag}\left(\log(X) u^T + \log(1 - X)(1 - u)^T \right)$.

2.4.2 Ricostruzione di segnali sinusoidali

Frequenza costante

Per la validazione del metodo è stata dapprima eseguita una sperimentazione su segnali sinusoidali:

$$y(t) = A \cos(2\pi f t + \phi) \quad t = 1, \dots n$$

Come primo esperimento si è scelto un set di sinusoidi mantenendo la frequenza $f = f_0$ costante, facendo variare l'ampiezza secondo una distribuzione gaussiana $A \sim \mathcal{N}(A_0, \sigma_A)$ e la fase in modo uniforme $\phi \in [-\pi, \pi]$. Abbiamo in questo caso ovviamente $d = 2$.

La Fig. 2.4.2 mostra la ricostruzione ottenuta con $N = 10^5$ sinusoidi con $A_0 = 0.5, f_0 = 0.004$, $\sigma_A = 0.1$, frames lunghi $n = 100$ e con un ordine del polinomio $m = 50000$.

Anche senza l'ottimizzazione dell'ordine del polinomio la ricostruzione è eccellente.

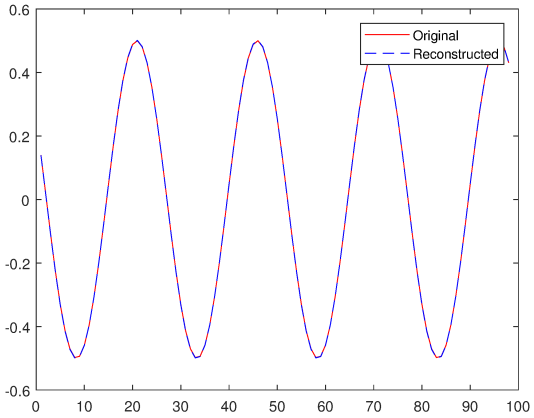


Figura 2.4.2: Ricostruzione sinusoide con frequenza costante

Frequenza variabile

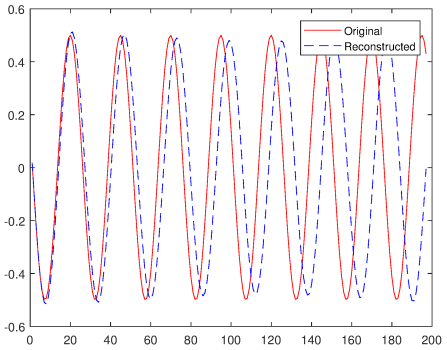
Come secondo esperimento è stata fatta variare la frequenza della sinusoide secondo una distribuzione gaussiana $f \sim \mathcal{N}(0, \sigma_f)$, $\sigma_f = 0.01$ e mantenendo le stesse distribuzioni per ampiezza e fase.

La Fig. 2.4.3 mostra la ricostruzione di sinusoidi con frequenza variabile con $m = 2000000$, $N = 5 * 10^5$ e $N = 10^6$ rispettivamente.

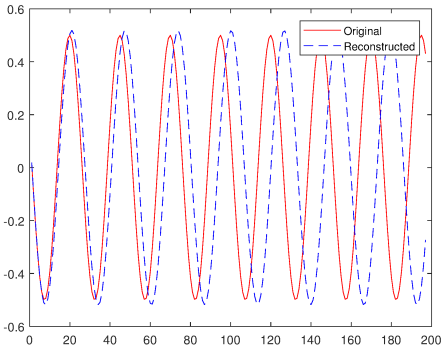
E' chiaro come al crescere di t l'errore di ricostruzione aumenta progressivamente. Anche se ancora presente, il problema viene mitigato dall'ottimizzazione dell'ordine del polinomio come mostrato in Fig. 2.4.5.

In Fig. 2.4.4a, 2.4.4b sono riportati il $NMSE(t) = \frac{\sum_{k=1}^{Ntest}(y_k(t) - \widehat{y}_k(t))^2}{\sum_{k=1}^{Ntest} y_k(t)^2}$ e l'ordine del polinomio ottimo rispettivamente.

La crescita progressiva dell'errore all'aumentare di t limita la lunghezza del frame necessaria per ottenere una qualità di ricostruzione totale del frame accettabile.

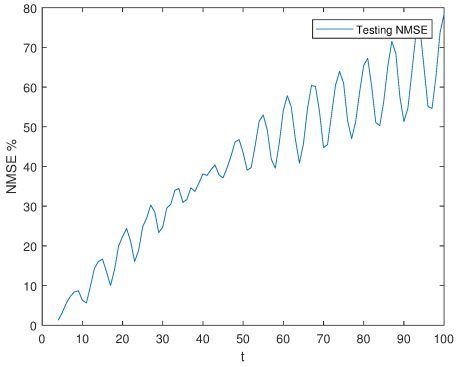


(a) $N_{\text{points}} = 5 * 10^5$

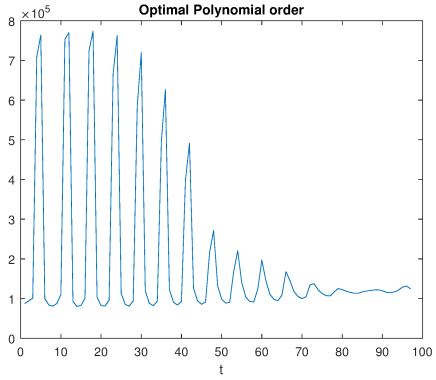


(b) $N_{\text{points}} = 10^6$

Figura 2.4.3: Ricostruzione sinusoidi con frequenza variabile senza ottimizzazione di m



(a) Errore di ricostruzione al variare dei campioni in un frame



(b) Valore ottimo di m al variare dei campioni in un frame

2.4.3 Ricostruzione di segnali reali

La ricostruzione è stata applicata ai due datasets di segnali reali sperimentati in precedenza.

Per il segnale ECG è stata utilizzata la matrice della classe 1 del dataset TwoLeadECG. Le Fig. 2.4.6a, 2.4.6b sono mostrati rispettivamente un frame del segnale ricostruito e il $NMSE$ ottenuto al variare di diversi punti di test. La qualità di ricostruzione è ottima con un $NMSE$ medio di 11.53%.

Per quanto riguarda il segnale vocale la lunghezza del frame è stata ridotta da $n = 140$ a $n = 67$ per ovviare al degrado della ricostruzione all'aumentare di t.

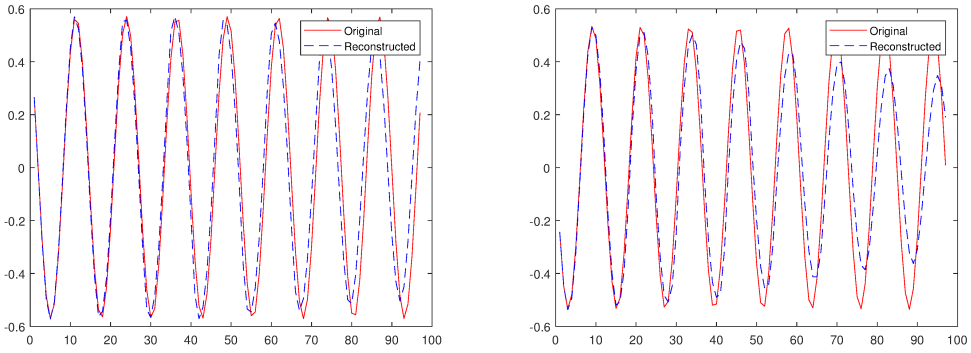
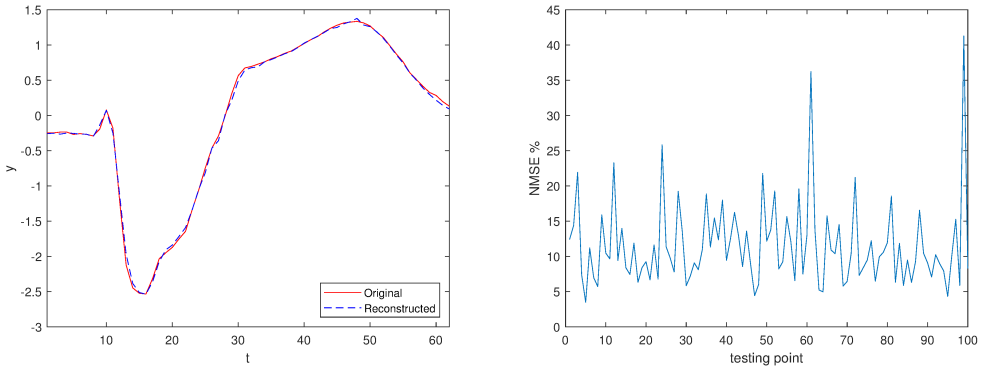


Figura 2.4.5: Sinusoidi ricostruite con ottimizzazione di m, $N_{\text{points}} = 2 * 10^5$

Il segnale di speech delle vocali infatti può essere ben approssimato da un segnale quasi-periodico con parametri delle sinusoidi variabili nel tempo:

$$y_{\text{vowel}}(t) \approx \sum_{k=1}^{N_h} A_k(t) \cos(2\pi f_k(t)t + \phi_k(t)) + n(t) \tag{2.40}$$

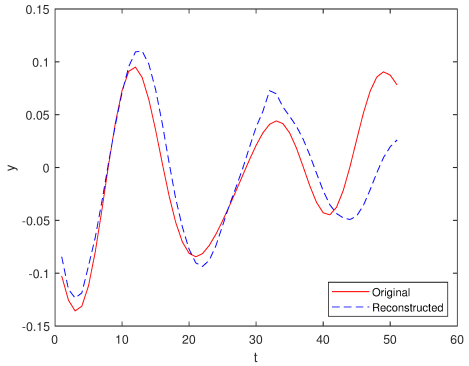


(a) Un frame del segnale ECG ricostruito

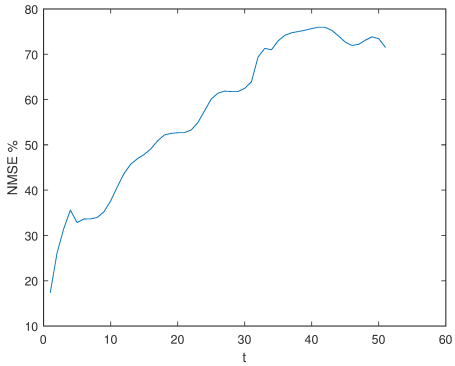
(b) NMSE per il segnale ECG al variare dei punti di test

Anche con questi accorgimenti tuttavia le performances di ricostruzione non sono ottimali con un $NMSE$ medio di 57.27%. Questo risultato è causato dall'alto numero di armoniche presenti nel segnale vocale e dal basso numero di punti di training del dataset.

Per ottenere prestazioni migliori in questo contesto è necessario un dataset molto ampio come visto per la ricostruzione delle sinusoidi.



(a) Un frame di segnale vocale ricostruito



(b) NMSE per il segnale vocale

2.5 Modello generativo

Il modello sviluppato è stato successivamente applicato con ottimi risultati alla generazione del segnale.

Per la generazione del segnale è stato utilizzato un approccio iterativo riapplicando il modello $\widehat{G}(\cdot)$ agli ultimi campioni di ciascun frame generato:

$$\begin{aligned} y_t &= (y_t^{(1)},\ y_t^{(2)}) \\ y_{t+1} &= \widehat{G}\left(y_t^{(2)} + \eta_t\right) \end{aligned} \tag{2.41}$$

dove $y_t^{(1)} \in \mathbb{R}^{N-d}$, $y_t^{(2)} \in \mathbb{R}^d$ e η_t è una componente di rumore gaussiano fittizio con deviazione standard σ_η, introdotta per evitare che il segnale generato sia perfettamente periodico.

Il modello è stato applicato ai due dataset, TwoLeadECG ed ECG200 utilizzati anche in [86] dove la generazione del segnale viene fatta con le Generative Adversarial Networks.

Analogamente alla sperimentazione eseguita in [86] le prestazioni del modello sono state misurate in termini di Dynamic-Time-Warping Distance (DTW) definita come la minima distanza tra il segnale generato e ciascun esempio nel training set al variare del warping delle forme d'onda:

$$DTW(x^i, x^j) = \sqrt{\min_{w_1,\dots w_R} \sum_{r=1, w_r=(k,l)}^{R} (x_l^i - x_k^j)^2} \tag{2.42}$$

La DTW è una misura della similarità dei segnali generati con il training set e deve essere sufficientemente bassa ma al contempo non eccessivamente per non cadere nell'overfitting.

La Tab. 2.5 riporta i parametri scelti per il modello generativo per le due classi di ciascun dataset.

Per l'ECG200 è stata utilizzata la stessa dimensione $d = 20$ del TwoLeadECG. Questa scelta è stata motivata dal numero di punti troppo basso per poter ottenere una stima di d accurata per l'ECG200.

	Manifold dimension d	Bernstein order m	Noise standard deviation $\sigma_\eta^\%$
ECG200 class 1	20	2	0.02
ECG200 class 2	20	2	0.03
Two Lead ECG class 1	20	25	0.01
Two Lead ECG class 2	20	20	0.07

Tabella 2.5: Parametri per il modello generativo

In Fig. 2.5.1, 2.5.3, 2.5.4, 2.5.5 si può notare come il modello generi traiettorie simili al training set per entrambi i dataset.

Le Fig. 2.5.2a, 2.5.2b riportano il confronto della la DTW in media e deviazione standard rispettivamente tra il modello e proposto e le GAN.

Il modello proposto ottiene valori migliori di DTW sia in media che deviazione standard rispetto alle GAN.

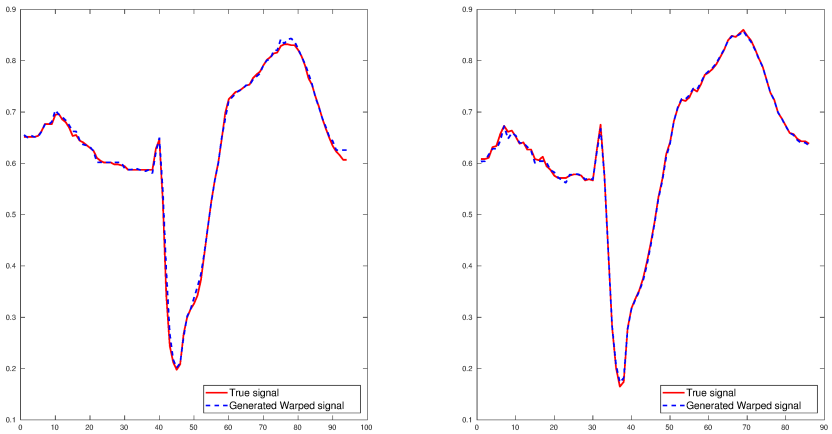
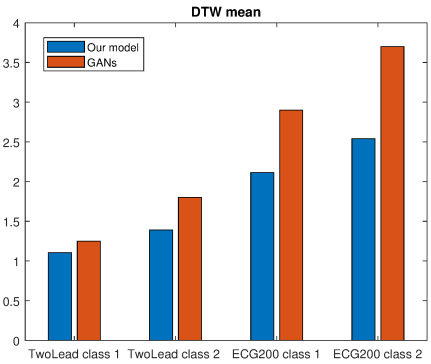
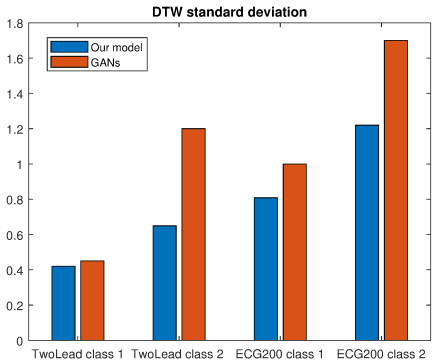


Figura 2.5.1: Esempi di segnali generati per il TwoLeadECG classe 1 e warping della forma d'onda rispetto ai segnali nel training set [4]



(a) DTW media per il modello proposto e per le GANs[4]



(b) Deviazione standard della DTW per il modello proposto e per le GANs[4]

Inoltre il nostro modello ha il vantaggio di non necessitare una lunga fase di training in quanto i parametri da ottimizzare sono solamente l'ordine del polinomio m e σ_η diversamente dalle GAN di cui devono essere allenati i pesi.

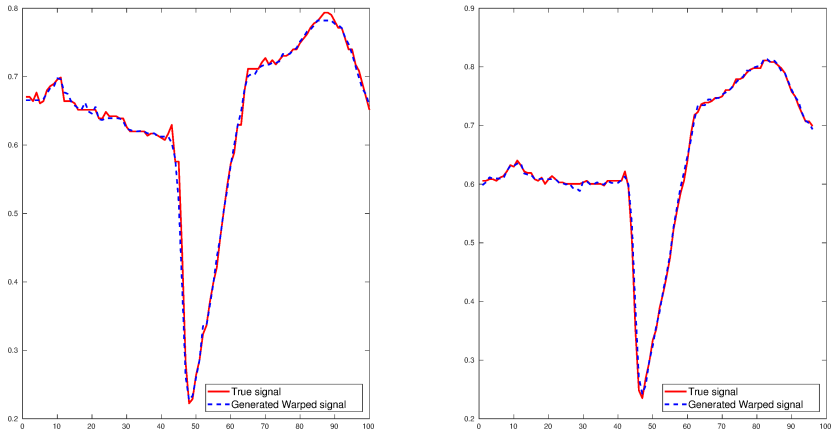


Figura 2.5.3: Forme d'onda generate per il Two Lead ECG classe 2[4]

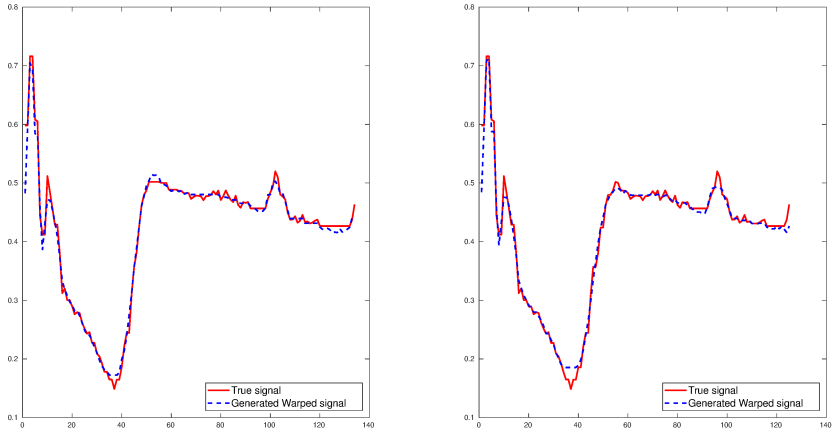


Figura 2.5.4: Forme d'onda generate per ECG200 classe 1[4]

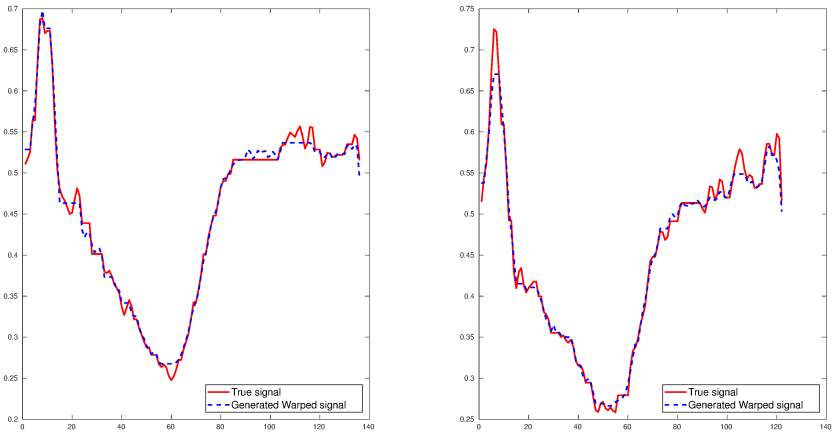


Figura 2.5.5: Forme d'onda generate per ECG200 classe 2[4]

Il modello generativo è stato poi applicato alla data augmentation dei dati utilizzata per migliorare le performances di classificazione[96, 97] con tecnica SVM.

In Tab. 2.6 è mostrata l'accuratezza ottenuta con la data augmentation effettuata con il modello generativo proposto e con le GAN applicata all'ECG200. A parità di incremento del numero di esempi il nostro modello garantisce un maggior aumento dell'accuratezza.

Il codice per la riproducibilità dei risultati complessivi ottenuti con la tecnica di Manifold Learning proposta si trova al seguente link `https://codeocean.com/capsule/6692152/tree/v3`.

Tabella 2.6: Accuracy della classificazione del dataset ECG200 con la data augmentation

n° of synthetic generated samples	Data augmentation with GANs SVM accuracy [%]	Data augmentation with our approach SVM accuracy [%]
0 (real data)	81.61	81.61
100	85.69	87.34
200	88.75	90.99
400	91.03	94.12
600	92.35	95.65
800	93.32	**96.49**

Capitolo 3

Compressione di CNN per sistemi embedded con decomposizioni tensoriali

3.1 Tecniche di compressione delle Convolutional Neural Networks

Lo sviluppo di Convolutional Neural Networks (CNN) per implementazione su sistemi embedded è un problema di grande interesse e molto esplorato negli ultimi anni per le più svariate applicazioni come classificazione, segmentazione, object detection..

Molto lavoro è stato svolto per il progetto di architetture a bassa di occupazione di memoria e basso costo computazionale che mantengano allo stesso tempo ottime prestazioni in termini di accuratezza.

Esiste una grandissima varietà di tecniche sviluppate per la compressione e la riduzione di complessità delle Deep Neural Networks. Tra le principali possiamo elencare:

- **Pruning**

  Consiste nell'eliminazione di pesi non indispensabili per il modello. Può essere basato sullo scarto dei pesi presi singolarmente ad uno ad uno[98], oppure riguardare interi filtri[99]. Nel primo caso si ha una riduzione dell'occupazione di memoria, tuttavia per avere una effettiva diminuzione della complessità computazionale è necessario implementarlo su hardware che supporta reti convoluzionali con pesi sparsi. Nel secondo invece si ha sempre un'effettiva riduzione del costo computazionale ma il pruning è meno ottimizzato.

Il pruning può essere basato sul modulo dei pesi o dei filtri, in particolare per i filtri è molto utilizzata la norma l_1, a tal scopo vengono spesso usate tecniche di regolarizzazione per compattare il più possibile il valore dei pesi stessi[100, 101]. A volte invece può basarsi sulle feature in output dai layer[102].

- **Quantizzazione**

  La quantizzazione delle reti può essere applicata sia ai pesi che alle feature ed è stata largamente implementata per sistemi embedded e mobili[103, 104, 105].

  Per quanto riguarda i sistemi embedded è stato sviluppato il formato TensorflowLite (*.tflite*) per la conversione di modelli implementati ad alto livello (Tensorflow,Keras,Pytorch,Caffe,ecc) in un formato per microcontrollori con processori ARM-Cortex-M-Series.

  Questo formato implementa la quantizzazione a 8 bit dei pesi ed eventualmente anche delle feature con un alto livello di ottimizzazione dell'occupazione di memoria, costo computazionale e tempo di inferenza per questa famiglia di processori.

- **Knowledge distillation**.

  Si basa sul training iniziale di un modello molto grande denominato "teacher" sui cui è basato il training di un modello più piccolo e lightweight che gioca il ruolo di "student".

  La supervisione da parte del teacher per lo student può essere effettuato in diverse maniere, ad esempio tramite riuso delle feature del "teacher" per lo student[106], tramite imitazione da parte dello student della funzione imparata dal teacher[107], oppure allineando le feature di layer corrispondenti dei due modelli[108].

3.2 Motivazione

Le tecniche di compressione per le Deep Neural Networks includono anche tecniche di decomposizione matriciale e tensoriale[5].

Per quanto riguarda le decomposizioni matriciali applicate alle CNN i tradizioni approcci in letteratura riguardano la Principal Component Analysis(PCA) e la Singular Value Decomposition(SVD).

In [109] la SVD viene applicata alla Resnet addestrata sul dataset CIFAR-10 e il training viene applicato direttamente alla decomposizione dei layer. In

questo caso tuttavia i tensori 4D dei layer convoluzionali vengono matricizzati e risultati sono subottimi rispetto all'utilizzo di fattorizzazioni tensoriali.

[110] decompone i layer con una PCA riducendo in questo modo l'occupazione di memoria nella RAM del dispositivo embedded per poi ricostruire i pesi in fase di inferenza. Questo approccio tuttavia non migliora i tempi di inferenza aggiungendo anzi un overhead richiesto dalla ricostruzione dei pesi a partire dalla base e dai coefficienti della PCA.

Le fattorizzazioni tensoriali consentono una migliore approssimazione del tensore dal momento che quelle matriciali necessitano inevitabilmente di una matricizzazione del tensore $\mathcal{W} \in \mathbb{R}^4$ di un layer.

Tra le molteplici tecniche disponibili quelle utilizzate tradizionalmente in letteratura sono essenzialmente la decomposizione CANDECOMP/PARAFAC la quale approssima un tensore con la somma di un numero R di tensori di rango 1, e la decomposizione di Tucker.

Contrariamente alle altre tecniche di compressione, le fattorizzazioni tensoriali sono state applicate alle DNN solamente dal punto di vista teorico. In [111] la decomposizione CP viene applicata alla Alexnet[112] trainandola per la classificazione sul dataset [113]. Tuttavia viene effettuata solo una misura del drop dell'accuracy ed una stima teorica della riduzione del numero di Floating-Point-Operations (FLOPs).

L'unico lavoro una rete compressa con decomposizioni tensoriali viene implementata su un sistema reale è in [114] dove la fattorizzazione di Tucker viene utilizzata su reti note in letteratura quali Alexnet, Googlenet[115], VGG16[116],VGG-S su una GPU Titan X in uno smartphone.

In questo lavoro sono state implementate le tecniche di decomposizione tensoriale per lo sviluppo di tre diverse reti le quali sono state implementate in sistemi embedded per tre diverse applicazioni.

Le tecniche di compressione implementate non sono da considerarsi come un'alternativa alle altre tecniche esistenti bensì possono essere utilizzate in combinazione con quest'ultime. Inoltre esse comportano una modifica dell'architettura del modello diversamente dal pruning dei pesi, quantizzazione e knowledge distillation.

3.3 Decomposizioni tensoriali

3.3.1 Decomposizione CANDECOMP/PARAFAC

La CANDECOMP/PARAFAC decomposition approssima un tensore di ordine n $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ con la somma di un numero R di tensori di rango 1:

$$\widehat{\mathcal{X}} = \sum_{r=1}^{R} a_r^{(1)} \circ \ldots \circ a_r^{(n)} \tag{3.1}$$

dove $\circ$ indica il prodotto esterno tra tensori. Le matrici $A^{(k)} = [a_1^{(k)}, \ldots, a_R^{(k)}] \in \mathbb{R}^{N_k \times R}$ vengono ottenute minimizzando l'errore di approssimazione:

$$A^{(1)}, \ldots, A^{(n)} = \underset{A^{(1)},\ldots,A^{(n)}}{\arg\min} \left\| \mathcal{X} - \widehat{\mathcal{X}} \right\|_2^2 \tag{3.2}$$

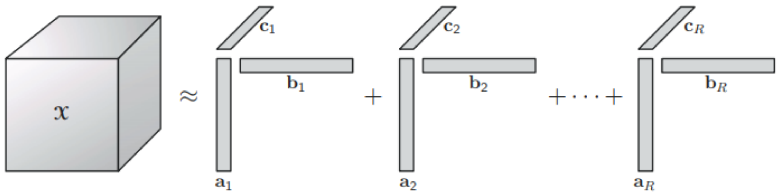


Figura 3.3.1: Decomposizione CP di un tensore di ordine 3[5]

Per la soluzione di questo problema di ottimizzazione sono state sviluppate tecniche denominate Alternated-Least-Squares[117](ALS).

Essenzialmente il problema di ottimizzazione in 3.2 viene risolto iterativamente rispetto alle singole matrici $A^{(k)}$. Detto infatti $\widehat{X}_{(k)}$ la matricizzazione modulo-k di $\widehat{\mathcal{X}}$ risulta:

$$\widehat{X}_{(k)} = A^{(k)} \left(A^{(n)} \odot \ldots \odot A^{(k+1)} \odot A^{(k-1)} \odot \ldots A^{(1)} \right)^T \tag{3.3}$$

dove $\odot$ è il prodotto di Khatri-Rao di due matrici. Esso viene calcolato attraverso il prodotto di Kronecker delle colonne di due matrici $A = [a_1, \ldots, a_N], B = [b_1, \ldots, b_N]$:

$$A \odot B = [a_1 \otimes b_1, \ldots, a_N \otimes b_N] \tag{3.4}$$

La minimizzazione dell'errore rispetto alla sola $A^{(k)}$ è infatti un problema ai minimi quadrati:

$$\begin{aligned} A_\star^{(k)} &= \arg\min_{A^{(k)}} \left\| X_{(k)} - \widehat{X}_{(k)} \right\|_2^2 \\ H_k &= \left(A^{(n)} \odot \ldots \odot A^{(k+1)} \odot A^{(k-1)} \odot \ldots A^{(1)} \right)^T \\ A_\star^{(k)} &= X_{(k)} H_k^\dagger \end{aligned} \tag{3.5}$$

Nonostante siano stati sviluppati algoritmi estremamente efficienti già inclusi in librerie Matlab, in Python nella libreria *Tensorly* [118],nella libreria *TenDeC++*[119] in C/C++, il calcolo della decomposizione è computazionalmente molto pesante ed è difficilmente implementabile direttamente su sistemi embedded.

Il fattore di compressione ottenuto dalla decomposizione CP sarà dato da:

$$c_f = R \frac{\sum_{k=1}^{n} N_k}{N_1 \cdots N_n} \tag{3.6}$$

3.3.2 Decomposizione di Tucker

La decomposizione di Tucker approssima un tensore $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ con un core-tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_n}$ moltiplicato lungo ogni direzione da matrici unitarie $U^{(k)} \in \mathbb{R}^{R_k \times N_k}$:

$$\widehat{\mathcal{X}} = \mathcal{G} \times_1 U^{(1)} \ldots \times_n U^{(n)} \tag{3.7}$$

dove $\times_k$ denota il prodotto modulo-k di un tensore con una matrice ottenuto moltiplicando le righe della matrice per le fibre del tensore lungo la direzione k:

$$\begin{aligned} \mathcal{Y} &= \mathcal{X} \times_k H \\ Y_{(k)} &= H X_{(k)} \end{aligned}$$

In maniera analoga alla decomposizione CP è utile effettuare la matricizzazione della 3.7 lungo la direzione k:

$$\widehat{X}_{(k)} = U^{(k)} G_{(n)} \left(U^{(n)} \otimes \ldots \otimes U^{(k+1)} \otimes U^{(k-1)} \otimes \ldots U^{(1)} \right)^T \tag{3.8}$$

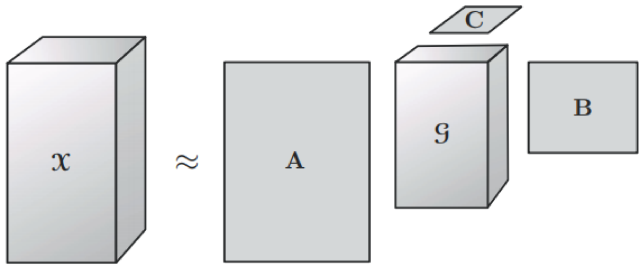


Figura 3.3.2: Decomposizione di Tucker di un tensore di ordine 3[5]

Si può provare inoltre che $U^{(k)}$ è ottenuto tramite la SVD troncata di $\widehat{X}_{(k)}$.

Per il calcolo della decomposizione è stato sviluppato il metodo Higher-Order-Orthogonal-Iteration (HOOI). Similarmente alla CP $U^{(k)}$ viene aggiornato iterativamente stavolta tramite SVD dell'approssimazione corrente di $X_{(k)}$.

Alla fine di ogni ciclo si può aggiornare $\mathcal{G}$ grazie alle proprietà del prodotto modulare:

$$\mathcal{G} = \widehat{X} \times_1 U^{(1)^T} \dots \times_n U^{(n)^T}$$

Il fattore di compressione per la decomposizione di Tucker sarà:

$$c_f = \frac{R_1 \cdots R_n + \sum_{k=1}^{n} R_k N_k}{N_1 \cdots N_n} \tag{3.9}$$

3.3.3 Applicazione delle decomposizioni tensoriali alle CNN

Decomposizione CP

Consideriamo un layer convoluzionale con un input $\mathcal{X}^{H \times W \times S}$ dove S è il numero di canali e H, W sono le dimensioni spaziali delle feature.

La convoluzione con il kernel $\mathcal{K} \in \mathbb{R}^{D \times D \times S \times T}$ può essere scritta come:

$$\mathcal{Y}_{h',w',t} = \sum_{s=1}^{S} \sum_{i,j} \mathcal{K}_{i,j,s,t} \mathcal{X}_{h_i,w_j,s} \tag{3.10}$$

Applicando la decomposizione CP a $\mathcal{K}$ avremo per ogni elemento:

$$\widehat{\mathcal{K}}_{i,j,s,t} = \sum_{r=1}^{R} a_{i,r} b_{j,r} c_{s,r} d_{t,r} \tag{3.11}$$

Sostituendo nella 3.10 si ottiene:

$$\begin{aligned}\widehat{\mathcal{Y}}_{h',w',t} &= \sum_{r=1}^{R}\sum_{s=1}^{S}\sum_{i,j} a_{i,r} b_{j,r} c_{s,r} d_{t,r} \mathcal{X}_{h_i,w_j,s} = \\ &= \sum_{r=1}^{R}\sum_{i,j}\sum_{s=1}^{S} \mathcal{Q}_{i,j,r} c_{s,r} d_{t,r} \mathcal{X}_{h_i,w_j,s}\end{aligned}$$

dove $\mathcal{Q}_{i,j,r} = a_{i,r}, b_{j,r}$. La convoluzione può essere scritta come la successione di tre convoluzioni:

$$\begin{aligned}\widehat{\mathcal{Y}}_{h',w',t} &= \sum_{r=1}^{R} \mathcal{Z}_{h',w',r} d_{t,r} \\ \mathcal{Z}_{h',w',r} &= \sum_{i,j} \mathcal{W}_{h_i,w_j,r} \mathcal{Q}_{i,j,r} \\ \mathcal{W}_{h_i,w_j,r} &= \sum_{s=1}^{S} \mathcal{X}_{h_i,w_j,s} c_{s,r}\end{aligned} \tag{3.12}$$

Le convoluzioni nella 3.12 sono equivalenti dal punto di vista pratico a sostituire il layer originale con 2 layer convoluzionali standard di dimensione 1×1 inframezzati da una convoluzione Depthwise che viene effettuata canale per canale. La decomposizione del layer così ottenuta è mostrata in Fig. 3.3.3

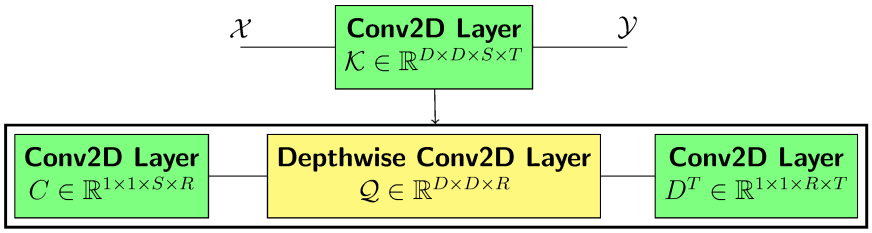


Figura 3.3.3: Decomposizione CP per un layer convoluzionale

Detto s lo strides della convoluzione si può facilmente stimare il fattore di compressione rispetto al numero di parametri c_{params} e il fattore di riduzione del costo computazionale c_{flops}:

$$c_{\text{params}} = \frac{R(D^2 + S + T)}{D^2 ST}$$
$$c_{\text{flops}} = \frac{RD^2/s^2 + R(S + T)}{D^2 ST/s^2} \tag{3.13}$$

Supponiamo ora di applicare la decomposizione ad un fully-connected-layer che è semplicemente caratterizzato da una trasformazione lineare:

$$y = Wx + b \tag{3.14}$$

con $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times n}$.

La decomposizione CP applicata ad una matrice 2D consiste in una semplice fattorizzazione low-rank: $W = BA^T$, $B \in \mathbb{R}^{m \times R}$ $A \in \mathbb{R}^{n \times R}$.

Per il fully-connected-layer avremo semplicemente una scomposizione in altri due fully-connected-layer come mostrato in Fig. 3.3.4.

$$z = A^T x, \quad y = Bz + b$$

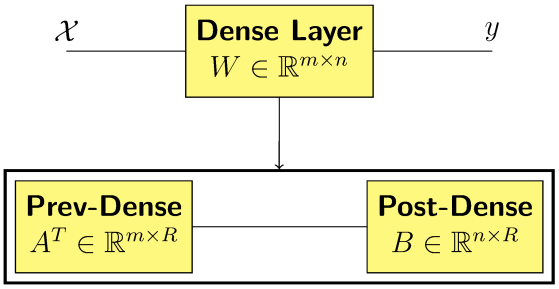


Figura 3.3.4: Decomposizione CP per il fully-connected layer

Nel caso del fully-connected-layer si ha molto semplicemente:

$$c_{\text{params}} = \frac{R(m + n)}{mn}, \quad c_{\text{ops}} = c_{\text{params}} \tag{3.15}$$

Decomposizione di Tucker

In generale un tensore del 4° ordine avrà la seguente decomposizione di Tucker:

$$\widehat{\mathcal{K}} = \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)} \tag{3.16}$$

Tuttavia nel caso del tensore dei pesi $\mathcal{K}$ non viene effettuata la compressione lungo le direzioni spaziali in quanto la dimensione $D \times D$ è tipicamente molto piccola.

Di conseguenza la decomposizione utilizzata diventa:

$$\begin{aligned} \widehat{\mathcal{K}} &= \mathcal{G} \times_3 U^{(3)} \times_4 U^{(4)} \\ \widehat{\mathcal{K}}_{i,j,s,t} &= \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} \mathcal{G}_{i,j,r_3,r_4} U^{(3)}_{r_3,s} U^{(4)}_{r_4,t} \end{aligned} \tag{3.17}$$

Sostituendo nella 3.10 abbiamo:

$$\widehat{\mathcal{Y}}_{h',w'',t} = \sum_{r_4=1}^{R_4} \sum_{i,j} \sum_{r_3=1}^{R_3} \sum_{s=1}^{S} U^{(4)}_{r_4,t} \mathcal{G}_{i,j,r_3,r_4} U^{(3)}_{r_3,s} \mathcal{X}_{h_i,w_j,s} \tag{3.18}$$

La 3.18 corrisponde a tre convoluzioni standard con filtri di dimensione 1×1, $D \times D$ e 1×1 come sintetizzato dalla Fig. 3.3.5.

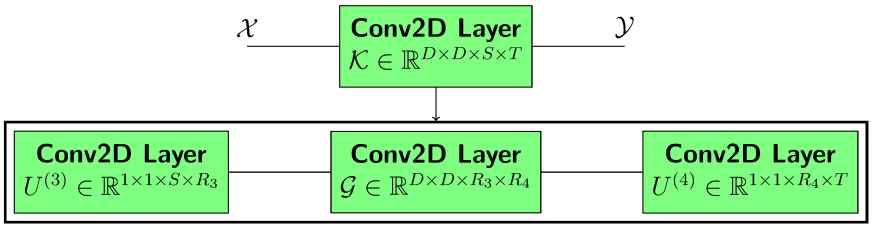


Figura 3.3.5: Decomposizione di Tucker per un layer convoluzionale

I fattori di compressione per il numero di parametri e per il costo computazionale sono dati da:

$$\begin{aligned} c_{\text{params}} &= \frac{D^2 R_3 R_4 + R_3 S + R_4 T}{D^2 S T} \\ c_{\text{ops}} &= \frac{D^2 R_3 R_4 / s^2 + R_3 S + R_4 T}{D^2 S T / s^2} \end{aligned} \tag{3.19}$$

Per quanto riguarda la decomposizione del fully-connected-layer anche in questo caso essa si riduce ad una fattorizzazione matriciale $W = UG$. La decomposizione del fully-connected-layer è pertanto identica a quella vista per la decomposizione CP.

3.4 Sviluppo di una rete per l'identificazione del mal dell'esca in real-time

3.4.1 Descrizione del problema e stato dell'arte

Il mal dell'esca è una malattia della vite che può colpire sia il tronco che il fogliame ed è causata da infezioni da parte di alcuni funghi specifici. Gli effetti includono necrosi delle foglie,essicazione della pianta e anche macchie violacee sull'uva.

Figura 3.4.1: Alcune foglie di vite colpite dal mal dell'esca[6]

Il trattamento per questa patologia consiste prevalentemente nell'identificazione e successiva rimozione delle piante malate nella stagione estiva. L'identificazione delle piante malate può essere un compito oneroso da svolgere a mano specialmente in vigneti di grandi dimensioni.

Per questa ragione può essere di grande interesse un sistema di identificazione automatica installato in un trattore che acquisendo le immagini con una videocamera riconosce e geolocalizza le piante malate semplificando di molto il lavoro di trattamento.

Negli ultimi anni molti lavori hanno utilizzato CNN per la classificazione delle patologie di piante [120, 121, 122]. Queste soluzioni ottengono elevate prestazioni di accuratezza a scapito tuttavia dell'utilizzo di reti molto pesanti quali Alexnet,VGG,ResNet,Inception, Faster R-CNN, dal costo computazionale troppo elevato per essere implementante in sistemi embedded dal basso costo e consumo di potenza.

In questo lavoro,svolto in collaborazione con *STMicroElectronics*, è stato proposto un sensore in grado di effettuare la detection del mal dell'esca grazie ad una CNN low-rank dal costo computazionale e dal tempo di inferenza estremamente ridotto.

Il sensore compie l'acquisizione dell'immagini grazie ad una OpenMV Cam che monta un microcontrollore STM32H7 Plus che ne effettua la classificazione con la rete importata nella board.

La CNN è stata progettata in modo da garantire alte performances di accuratezza ed un tempo di inferenza misurato sul microcontrollore, inferiore ad una determinata soglia.

Tale valore limite per il tempo di inferenza è stato stimato considerando il sistema rappresentato in Fig. 3.4.2:

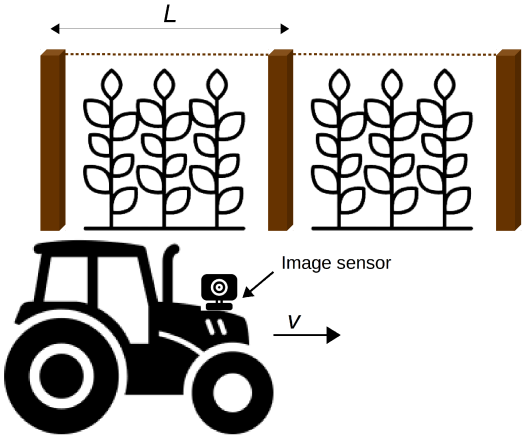


Figura 3.4.2: Sistema per la rilevazione dell'Esca[6]

Considerando che la videocamera montata su un trattore che procede ad una velocità v, ha una certa velocità di acquisizione delle immagini f, nel tempo che il trattore percorre la distanza L tra due filari la camera acquisisce un numero di immagini N_f:

$$N_f = \frac{L}{v \times f} \tag{3.20}$$

Realisticamente solo una frazione $N_{\text{inf}} = kN_f$ $k < 1$ delle immagini acquisite verrà utilizzata per l'inferenza, dunque il vincolo sul tempo di inferenza sarà scalato di un fattore $1/k$:

$$t_{\text{inf}} < \frac{1}{kf} = \frac{L}{N_{\text{inf}} \times v} \tag{3.21}$$

Supponendo $v = 8$ km/h, $L = 3$ m, $f = 50$ frame/sec abbiamo $N_f \approx 67$. Scegliendo $N_{\text{inf}} = 20$ abbiamo $t_{\text{inf}} < 68$ msec.

3.4.2 Architettura proposta

Il Design-flow della rete proposta è mostrato in Fig. 3.4.3. E' stata inizialmente sviluppata una rete full-rank denominata FR-Net ed è stata addestrata in modo da ottenere l'accuratezza richiesta con un discreto margine.

Essa è costituita da tre layer convoluzionali con attivazioni ReLu seguiti da downsampling effettuati con dei Maxpooling. in output dalle convoluzioni abbiamo due fully-connected-layer il primo dei quali con attivazione e ReLu ed infine il Softmax finale per la classificazione.

L'architettura dettagliata della FR-Net è riportata in Tab. 3.1.

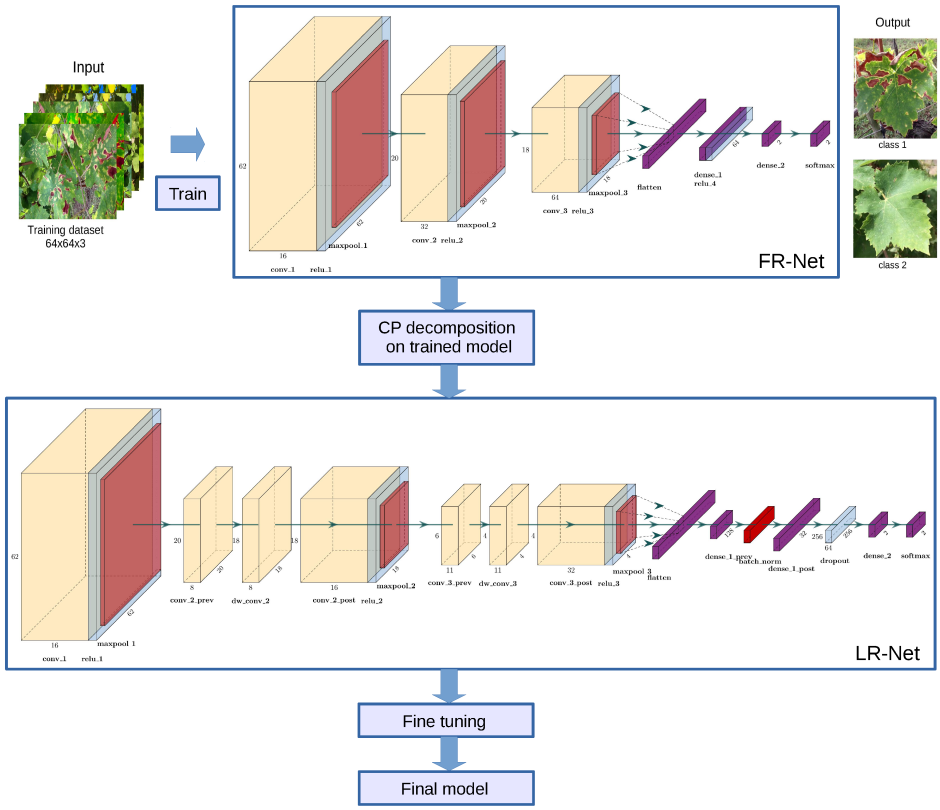


Figura 3.4.3: Flowchart per il design del modello proposto[6]

La rete FR-Net è stata allenata sull' ESCA-dataset descritto in dettaglio nella Sottosez. 3.4.4 con l'ottimizzatore Adadelta minimizzando una crossentropy loss, scegliendo un learning rate di 0.5 e un batch size pari a 64.

In base al design-flow in Fig. 3.4.3 dopo il training della FR-Net, è stata eseguita la compressione del modello tramite decomposizione CP.

Type	Filter shape	Input size	Number of parameters
conv_1	$3 \times 3 \times 3 \times 16$	$64 \times 64 \times 3$	448
relu_1	–	$62 \times 62 \times 16$	0
maxpool_1 (3×3)	–	$62 \times 62 \times 16$	0
conv_2	$3 \times 3 \times 16 \times 32$	$20 \times 20 \times 16$	4640
relu_2	–	$18 \times 18 \times 32$	0
maxpool_2 (3×3)	–	$18 \times 18 \times 32$	0
conv_3	$3 \times 3 \times 32 \times 64$	$6 \times 6 \times 32$	18496
relu_3	–	$4 \times 4 \times 64$	0
maxpool_3 (2×2)	–	$4 \times 4 \times 64$	0
flatten	–	$2 \times 2 \times 64$	0
dense_1	256×64	1×256	16448
relu_4	–	1×64	0
dropout (0.5)	–	1×64	0
dense_2	64×2	1×64	130
softmax	–	1×2	0

Tabella 3.1: Architettura FR-Net

Sono stati compressi tutti i layer convoluzionali fatta eccezione per il primo layer *conv_1* che ha dimensioni molto piccole ed ha un basso impatto sul costo computazionale della rete. Per di più esso estrae le feature principali utilizzate poi dal resto del modello, un'eventuale compressione pertanto risulterebbe in un degradamento troppo marcato dell'accuratezza. Analogo discorso può essere fatto per il fully-connected-layer finale *dense_2*.

La Tab. 3.2 riporta i dettagli della compressione dei layer della FR-Net.

Layer	Compression factor	Decomposition rank	Weights approximation error
conv_2	8	11	0.729
conv_3	8	23	0.751
dense_1	2	26	0.550

Tabella 3.2: Dettagli sulla decomposizione dei layer della FR-Net

La compressione non è stata effettuata in un singolo step, ma con un procedura iterativa tramite decomposizione di un singolo layer e successivo finetuning per compensare la perdita di accuracy dovuta alla compressione.

Questa procedura evita l'instabilità del modello dovuta a possibili minimi locali ottenuti comprimendo tutti i layer in unico step.

In Tab. 3.4 sono riportati il learning rate e il numero di epoche scelti per il finetuning di ciascun layer.

La rete low-rank così ottenuta è stata denominata LR-Net e la sua architettura è mostrata in Tab. 3.3.

Type	**Filter shape**	**Input size**	**Number of parameters**
conv_1	$3 \times 3 \times 3 \times 16$	$64 \times 64 \times 3$	448
relu_1	–	$62 \times 62 \times 16$	0
maxpool_1 (3×3)	–	$62 \times 62 \times 16$	0
conv_2_prev	$1 \times 1 \times 16 \times 11$	$20 \times 20 \times 16$	176
dw_conv_2	$3 \times 3 \times 11$	$20 \times 20 \times 11$	99
conv_2_post	$1 \times 1 \times 11 \times 32$	$18 \times 18 \times 11$	384
relu_2	–	$18 \times 18 \times 32$	0
maxpool_2 (3×3)	–	$18 \times 18 \times 32$	0
conv_3_prev	$1 \times 1 \times 32 \times 23$	$6 \times 6 \times 32$	736
dw_conv_3	$3 \times 3 \times 23$	$6 \times 6 \times 23$	207
conv_3_post	$1 \times 1 \times 23 \times 64$	$4 \times 4 \times 23$	1536
relu_3	–	$4 \times 4 \times 64$	0
maxpool_3 (2×2)	–	$4 \times 4 \times 64$	0
flatten	–	$2 \times 2 \times 64$	0
dense_1_prev	256×26	1×256	6656
batch_norm	–	1×26	104
dense_1_post	26×64	1×26	1728
relu_4	–	1×64	0
dropout (0.5)	–	1×64	0
dense_2	64×2	1×64	130
softmax	–	1×2	0

Tabella 3.3: Architettura LR-Net

Layer	Optimizer	Learning rate	Epochs
conv_2	Adadelta	0.01	20
conv_3	Adadelta	0.002	15
dense_1	Adadelta	0.01	30

Tabella 3.4: Finetuning LR-Net

3.4.3 Implementazione su OpenMV Cam

La rete sviluppata ad alto livello in Tensorflow/Keras v.2.4.1 in formato .h5, è stata implementata sul microcontrollore STM32H743II ARM Cortex-M7 incluso nella OpenMV Cam.

Il modello è stato dapprima convertito in formato .tflite grazie al convertitore TFliteConverter disponibile dalle API di Python con una quantizzazione a 8 bit che esegue una ulteriore compressione del modello.

Dopo la conversione in .tflite, è stato generato il codice C a basso livello per il microcontrollore grazie al tool STM32Cube.AI. Il file .cc così generato può essere caricato nella Flash del micro.

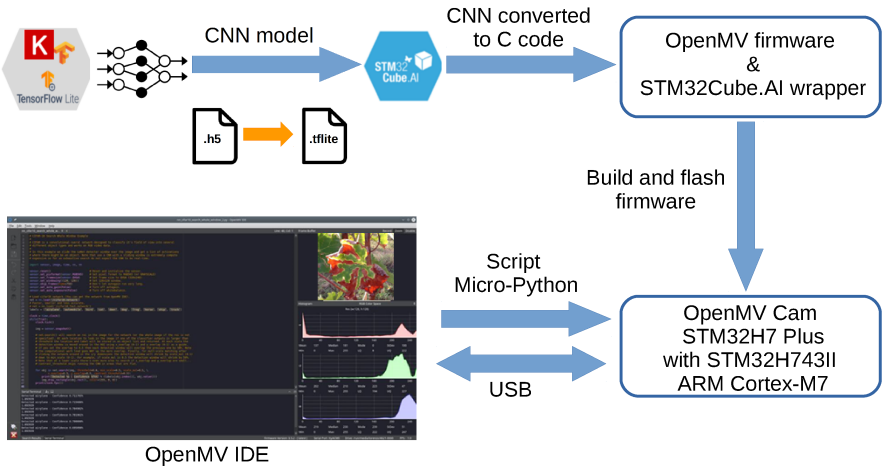


Figura 3.4.4: Implementazione del modello su OpenMV Cam con il tool STM32Cube.AI[6]

In alternativa a questa procedura, l'inferenza può essere fatta direttamente a partire dal modello .tflite grazie al MicroPython che consente di eseguire codice Python senza la conversione in C. La procedura per l'implementazione della rete su OpenMV Cam è riassunta dalla Fig. 3.4.4.

3.4.4 Risultati sperimentali

E' stato utilizzato per la sperimentazione il dataset ESCA [123] contenente 1770 immagini di foglie sane e infette dal mal dell'esca, di dimensione 1920×1080 e 1280×720. Il dataset è stato ampliato con una data augmentation eseguita con trasformazioni geometriche e variazioni di luminosità e contrasti. Il dataset ESCA dopo la data augmentation è descritto dalla Tab. 3.5.

Class name	Class ID	Number of original images	Number of images after all data augmentation transformations	Number of images after considered data augmentation transformations
esca	1	888	12432	8880
healthy	2	882	12348	8820
Total		1770	24780	17700

Tabella 3.5: ESCA dataset con la data augmentation

I risultati sono stati confrontati con quelli ottenuti da un gran numero di reti utilizzate in letteratura per la classificazione di patologie di piante quali MobileNet V1-V2-V3[124, 125, 126], ResNet[127], LeNet[128], SqueezeNet[129] ShuffleNet-V1-V2[130, 131], Improved ShuffleNet V1-V2[121], PeleeNet[132].

E' stata inizialmente eseguita una sperimentazione su PC con i modelli convertiti in .tflite per validare il modello proposto.

In Tab. 3.7 per i vari modelli sono riportati l'occupazione di memoria (KB), fattore di compressione, il numero di parametri, il tempo di inferenza (msec), l'accuracy nel testing set e la loss ottenuta.

Il modello proposto ottiene un'accuracy molto elevata, pari al 98.4%, ed al contempo i più bassi valori per l'occupazione di memoria, numero di parametri e tempo di inferenza. Vi è un fattore di compressione 4 di default applicato a tutti modelli derivante dalla quantizzazione nella conversione in .tflite.

L'implementazione su OpenMV Cam non è stata possibile per tutti i modelli dal momento che alcuni di essi non sono compatibili con il tool STMCube.AI.

La Tab. 3.6 riporta i risultati della sperimentazione sul micro. Sono stati misurati l'accuratezza, il tempo di inferenza, i frame al secondo ottenuti (FPS), il numero di immagini che è possibile processare tra filari della vite $N_{\text{inf}} = FPS \times N/v$, l'occupazione di memoria ed infine il parametro TP = accuracy $\times N_{\text{inf}}$ che misura il trade-off tra accuratezza e tempo di inferenza.

Il modello proposto ottiene i migliori valori per tutti i parametri rispetto alle altre reti, inoltre è l'unica che soddisfa il vincolo $t_{\text{inf}} < 68$ msec imposto inizialmente.

Il codice per la riproducibilità dei risultati ottenuti è raggiungibile al seguente link `https://codeocean.com/capsule/8031070/tree/v1`.

Model	**Test accuracy**	**Inference time [msec/img]**	**FPS**	$\boldsymbol{N_{inf}}$ **[FPS×(**$\boldsymbol{L/v}$**)]**	**TP [accuracy×**$\boldsymbol{N_{inf}}$**]**	**ROM bytes**
LR-Net	**0.980**	**64.213**	**15.574**	**21.180**	**20.756**	**13 015**
MobileNet V1 [124]	0.974	577.618	1.731	2.354	2.292	3237064
MobileNet V2 [125]	0.954	526.646	1.899	2.582	2.463	2268232
ResNet [127]	0.944	3687.71	0.271	0.368	0.347	23528194
LeNet [128]	0.942	85.726	11.665	15.864	14.944	338493
SqueezeNet [129]	0.976	448.866	2.228	3.030	2.957	745384
PeleeNet [132]	0.938	411.456	2.240	3.046	2.857	2108392

Tabella 3.6: Confronto delle prestazioni dei modelli implementati su OpenMV Cam STM32H7 Plus

Model	**Memory cost [KB]**	**Compression factor**	**Parameters number**	**Inference time [msec]**	**Test accuracy**	**Test loss**
LR-Net	**25.094**	**13**	**12204**	**12.044**	0.984	0.041
MobileNet V1 [124]	3525.992	4	3237058	132.881	0.968	0.328
MobileNet V2 [125]	2793.797	4	2268226	75.932	0.964	0.385
MobileNet V3 [126]	1905.031	4	1538162	25.988	0.920	0.603
ResNet [127]	23948.109	4	23581186	924.519	0.955	0.572
LeNet [128]	335.773	4	337806	15.795	0.943	0.368
SqueezeNet [129]	843.656	4	736450	152.698	0.984	0.052
ShuffleNet V1 [130]	1811.867	4	1359914	72.221	0.992	0.024
ShuffleNet V1 0.25x [130]	328.297	4	105404	14.841	0.980	0.067
Improved ShuffleNet V1 [121]	2354.695	4	5720606	70.311	**0.995**	**0.016**
ShuffleNet V2 [131]	1810.172	4	1329722	58.591	0.984	0.060
ShuffleNet V2 0.25x [131]	328.562	4	105404	14.553	0.962	0.228
Improved ShuffleNet V2 [121]	6369.516	4	5720606	147.065	0.992	0.033
PeleeNet [132]	2325.406	4	2113250	130.662	0.884	0.253

Tabella 3.7: Confronto delle prestazioni delle reti su desktop

3.5 CNN per la Semantic Segmentation su sistema embedded

3.5.1 Descrizione del problema

La Semantic Segmentation è un task della computer vision che consiste nella suddivisione dell'immagine in regioni e nell'etichettatura di queste sulla base di classi che identificano ciascuna uno o più tipi di oggetti.

Le applicazioni della Semantic Segmentation sono le più svariate come processing delle immagini in medicina[133], navigazione indoor[134], realtà aumentata [135].

Figura 3.5.1: Esempi di mappe ottenute tramite Semantic Segmentation[7]

In particolare combinando la Semantic Segmentation con i tradizionali algoritmi SLAM è possibile creare una mappa 3D dell'ambiente circostante permettendo la localizzazione e la rilevazione di ostacoli.

Ciò può essere utile in numerosi e variegati contesti come l'Autonomous Driving basato ad esempio sull'*edge computing* in cui l'esecuzione degli algoritmi e dei modelli di deep learning viene demandata ai nodi e non al cloud.

Se da un lato i nodi sono dispositivi dal basso costo e basso consumo di potenza, dall'altro nell' ambito della guida autonoma di veicoli elettrici i tempi di risposta necessari sono molto bassi date le velocità in gioco.

Un altro ambito di applicazione può invece essere la navigazione indoor di un più semplice robot[136], dove i tempi di esecuzione richiesti sono meno stringenti e quindi si può utilizzare anche hardware dalle minori performances computazionali.

In questo lavoro è stata proposta una CNN a basso costo computazionale per la Semantic Segmentation per sistema embedded. E' stata dimostrata l'applicabilità del modello sperimentazione su due piattaforme quali una Rasbperry Pi 4 adatta per applicazioni con tempi con tempi di inferenza più alti e una GPU NVIDIA Tesla K80 dalle performances più spinte.

3.5.2 CNN per la Semantic Segmentation dello stato dell'arte

In letteratura è stata sviluppata un gran varietà di reti per la Semantic Segmentation e sono denominate generalmente Fully-Convolutional-Networks (FCN). Queste reti diversamente da quelle utilizzate per la classificazione non hanno fully-connected-layer dal momento che esse devono fornire in output una mappa con una label per ogni pixel dell'immagine in input.

Anche per questo motivo possono risultare molto computazionalmente molto pesanti, ad esempio reti basate sulla VGG[116] e sulla AlexNet[137].

Le architetture disponibili in letteratura hanno una struttura tipicamente ad encoder-decoder, ovvero vi è un blocco di convoluzioni che riduce le dimensioni spaziali delle feature ed un blocco di decoding che riporta la dimensione a quella dell'immagine iniziale.

Le principali reti dello stato dell'arte sono qui elencate.

- **U-Net**

  La U-Net [138] è stata proposta per la segmentazione di immagini biomedicali e l'architettura ha una forma ad 'U' con pari numero di layer convoluzionali per l'encoder e decoder. Le feature in output dai layer dell'encoder vengono riutilizzate concatenandole a quelle del decoder.

- **DeepLab**

  La DeepLab[139] è in grado di effettuare la segmentazione con performances eccellenti anche a basse risoluzioni e con oggetti a scale differenti grazie alle 'dilated convolutions' e al 'Atrous Spatial Pyramidal Pooling' (ASPP), una tecnica di resampling delle feature per le diverse scale. Ha una occupazione di memoria e dispendio computazionale particolarmente elevati.

- **SegNet**

  La SegNet[140] è costituita da una backbone VGG-16 usata come encoder e da un decoder esegue l'upsampling utilizzando gli indici del maxpooling dell'encoder ottenendo feature sparse. Un banco di filtri trainato produce poi in output feature dense.

- **ENet**

  La ENet[141] (Efficient Neural Network) è riconosciuta in letteratura come la rete più efficiente per la Semantic Segmentation in termini di tempo di inferenza. L'encoder e il decoder sono costituiti da bottleneck implementate con residual-blocks, largamente utilizzati per reti profonde.

- **ICNet**

  La ICNet[142] è stata progettata per favorire l'efficienza mantenendo allo stesso momento la qualità della segmentazione. Vi sono tre path che fungono da encoder a diverse risoluzioni ottenute tramite downsampling, le feature alle diverse scale vengono combinate tramite delle fusion units e l'output viene fornito al decoder.

- **BiseNet**

  La BiseNet[143] è costituita da uno Spatial Path(SP) e un Context Path(CP). Lo Spatial Path contiene le informazioni spaziali dettagliate ed è ottenuto con tre convoluzioni che effettuano un downsampling a 1/8 della dimensione originale.

  Il Context Path contiene informazioni ad alto livello e viene fuso allo Spatial Path tramite un Feature-Fusion-Module. Vi è infine un upsampling che produce l'output a dimensione piena.

- **ErfNet**

  Nella ErfNet[144] encoder e decoder sono basati su residual-blocks denominati "Non-bottleneck-1D". Sono costituiti da 4 filtri 1D ciascuno ed eseguono dilated convolutions.

- **FastSCNN**

  La FastSCNN[145] è stata specificatamente progettata ed ottimizzata per la massima efficienza su sistemi embedded. In particolare le prestazioni ottimali si hanno se convertita in formato .tflite. E' costituita da due path di cui uno detto 'learning to downsample' calcola feature a basso livello le quali vengono poi combinati tramite un fusion module al secondo path che estrae informazioni a livello globale.

3.5.3 Rete proposta: UNet-ResNet

Architettura

Il design della rete proposta è partito dalla U-Net la cui architettura classica è mostrata in Fig. 3.5.2.

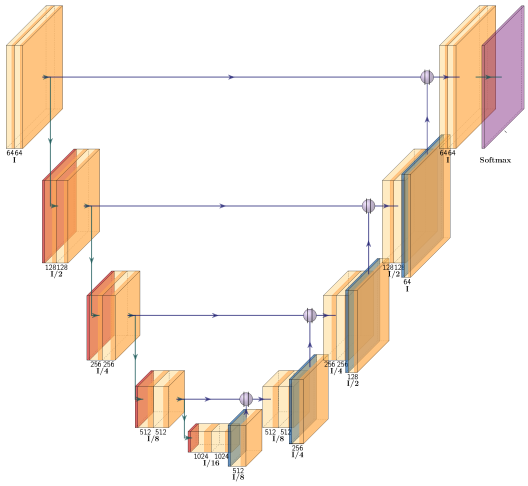


Figura 3.5.2: Architettura tradizionale U-Net

Per poter utilizzare questa rete tuttavia è necessario un numero troppo elevato di stadi per ottenere un'accuratezza accettabile il che risulterebbe poi in un modello non efficiente.

Pertanto i blocchi convoluzionali della U-Net implementati con semplici layer sono stati sostituiti con i residual-block della ResNet[127] riportati in Fig. 3.5.3.

Essi contengono 2 layer convoluzionali di dimensione 3×3 nel path principale e si dividono in due tipi: quelli che non effettuano un downsampling delle feature e quelli che invece hanno uno stride $s = 2$ nel primo layer convoluzionale. In questo caso vi è un layer di dimensione 1×1 con $s = 2$ nel residual path per aggiustare la dimensione.

Dell'architettura risultante denominata pertanto UNet-ResNet, sono state proposte tre versioni V1,V2,V3. Esse differiscono tra loro per il numero di stadi e alcuni fattori di downsampling. Questo è stato fatto per meglio esplorare il trade-off tra complessità del modello e performance di segmentazione.

L'architettura della UNet-ResNet-V1 è mostrata in Fig. 3.5.4 e riportata in dettaglio in Tab. 3.8.

L'encoder e il decoder hanno 2 stadi ciascuno dei quali è formato da 2 residual-block, il primo dei quali effettua un downsampling di un fattore 2.

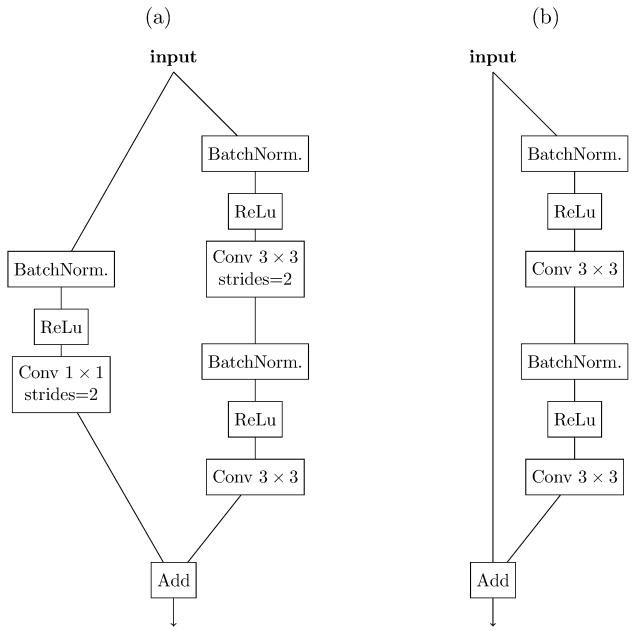


Figura 3.5.3: Residual block della ResNet con (a) e senza (b) downsampling.

Block	c	s	**Output name**	**Output shape**
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	2	X_0	$120 \times 90 \times 64$
residual block	64	2	-	$60 \times 45 \times 64$
residual block	64	1	X_1	$60 \times 45 \times 64$
residual block	128	2	-	$30 \times 23 \times 128$
residual block	128	1	-	$30 \times 23 \times 128$
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	1	Y_1	$60 \times 46 \times 64$
concat(Y_1, X_1)	-	-	-	
decoder residual block	64	2	-	$120 \times 90 \times 64$
decoder residual block	64	1	Y_2	$120 \times 90 \times 64$
concat(Y_2, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	4	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

Tabella 3.8: Architettura della Unet-Resnet V1 in dettaglio. Con c è indicato il numero di canali in output e con s lo stride di un blocco

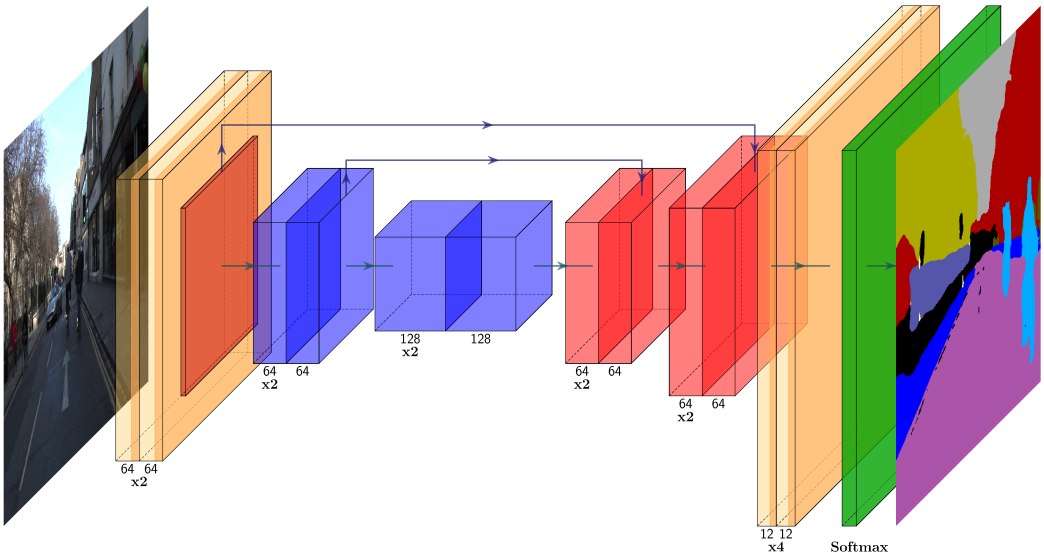


Figura 3.5.4: Architettura Unet-ResNet-V1

L'architettura in dettaglio delle versioni V2-V3 della UNet-ResNet è riportata rispettivamente in Tab. 3.9 e 3.10.

La UNet-ResNet-V2 ha uno stadio in più rispetto alla V1, al contempo il Maxpooling iniziale ha uno stride doppio. Riducendo la dimensione delle feature si ha un risparmio di costo computazionale ma un degradamento della qualità della segmentazione.

La versione V3 costituisce un intermedio tra V1 e V2, rispetto alla V2 lo stride aggiuntivo è stato "spostato" dal maxpooling ad un residual-block.

Per le tre reti è stato utilizzato un encoder pre-addestrato sul dataset Imagenet [146] in modo da migliorare l'accuratezza di segmentazione.

Training della UNet-ResNet

Le tre reti proposte sono state addestrate sul Camvid dataset[147, 148, 149] diffusamente utilizzato per la Semantic Segmentation.

Il dataset contiene 369 immagini di scene all'aperto per il training set, 100 per il validation set e 232 per il testing set. Hanno una risoluzione originale pari a 960×720 e sono segmentate in 32 classi di cui nella sperimentazione sono state selezionate solamente le 11 con il maggior contributo di label. La dimensione utilizzata per gli esperimenti è 480×360.

Le reti sono state trainate per 100 epoche con un ottimizzatore di Adam, un learning rate di 0.001 ed un batch size pari a 4.

La Tab. 3.11 riassume le prestazioni dei tre modelli misurati su desktop in termini di accuracy, occupazione di memoria (MB), numero di parametri,

Block	c	s	**Output name**	**Output shape**
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	4	X_0	$60 \times 45 \times 64$
residual block	64	2	-	$30 \times 23 \times 64$
residual block	64	1	X_1	$30 \times 23 \times 64$
residual block	128	2	-	$15 \times 12 \times 128$
residual block	128	1	X_2	$15 \times 12 \times 128$
residual block	256	2	-	$8 \times 6 \times 256$
residual block	256	1	-	$8 \times 6 \times 256$
decoder residual block	128	2	-	$16 \times 12 \times 128$
decoder residual block	128	1	Y_1	$16 \times 12 \times 128$
concat(Y_1, X_2)	-	-	-	
decoder residal block	64	2	-	$30 \times 24 \times 64$
decoder residal block	64	1	Y_2	$30 \times 24 \times 64$
concat(Y_2, X_1)	-	-	-	
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	1	Y_3	$60 \times 46 \times 64$
concat(Y_3, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	8	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

Tabella 3.9: Architettura Unet-Resnet-V2 in dettaglio

costo computazionale (MFLOPs) e mIoU che una metrica tradizionalmente utilizzata per la Semantic Segmentation che misura l'overlap tra le regioni dell'immagine e quelle predette.

La UNet-ResNet-V1 mostra i valori più alti per accuracy e mIoU, ha anche il minor numero di parametri e occupazione di memoria, tuttavia ha di gran lunga il costo computazionale più alto.

La V2 ha risultati peggiori in termini di accuratezza e perdendo comunque meno del 1% di accuracy ed il costo computazionale più basso.

La UNet-ResNet-V3 ha un costo computazionale molto vicino alla V2 e perde solamente lo 0.16% di accuracy rispetto alla V1. Essa rappresenta dunque un ottimo compromesso tra le prime due.

Block	c	s	**Output name**	**Output shape**
conv2d 7×7	64	2	-	$240 \times 180 \times 64$
maxpool	64	2	X_0	$120 \times 90 \times 64$
residual block	64	2	-	$60 \times 45 \times 64$
residual block	64	2	X_1	$30 \times 23 \times 64$
residual block	128	2	-	$15 \times 12 \times 128$
residual block	128	1	X_2	$15 \times 12 \times 128$
residual block	256	2	-	$8 \times 6 \times 256$
residual block	256	1	-	$8 \times 6 \times 256$
decoder residual block	128	2	-	$16 \times 12 \times 128$
decoder residual block	128	1	Y_1	$16 \times 12 \times 128$
concat(Y_1, X_2)	-	-	-	
decoder residual block	64	2	-	$30 \times 24 \times 64$
decoder residual block	64	1	Y_2	$30 \times 24 \times 64$
concat(Y_2, X_1)	-	-	-	
decoder residual block	64	2	-	$60 \times 46 \times 64$
decoder residual block	64	2	Y_3	$120 \times 92 \times 64$
concat(Y_3, X_0)	-	-	-	
conv2d 1×1	12	1	-	$120 \times 90 \times 12$
Upsample	12	4	-	$480 \times 360 \times 12$
Softmax	-	-	Y_{pred}	$480 \times 360 \times 12$

Tabella 3.10: Architettura Unet-Resnet-V3 in dettaglio

	Test accuracy [%]	**mIoU** [%]	**Storage cost** [MB]	**Parameters number**	**MFLOPs**
UNet-ResNet V1	86.19	56.89	12.663	1077k	5209
UNet-ResNet V2	85.69	54.17	46.620	4033k	2340
UNet-ResNet V3	86.03	54.81	46.726	4041k	2711

Tabella 3.11: Confronto tra i risultati sperimentali per le tre versioni V1,V2,V3 della UNet-ResNet

3.5.4 Compressione tramite decomposizione CP

Le reti mostrate non sono ancora in grado tuttavia di soddisfare i requisiti richiesti in termini di tempo di inferenza e costo computazionale una volta importate in un sistema embedded.

E' stata pertanto eseguita la compressione di questi modelli tramite la decomposizione CP la quale è stata applicata a tutti i layer convoluzionali di dimensione 3×3.

Diversamente da quanto fatto nella la rete per l'identificazione dell'esca è stata seguita una procedura iterativa "stage-wise" di compressione/finetuning.

Ad ogni iterazione è stata applicata la decomposizione a tutti i layer 3×3 di uno stadio dell'encoder e del corrispondente nel decoder per mantenere la simmetria della rete.

Per ultimo è stato compresso singolarmente il layer iniziale di dimensione 7×7 in quanto da esso dipende fortemente l'accuratezza della segmentazione. La procedura seguita è sintetizzata dalla Fig. 3.5.5.

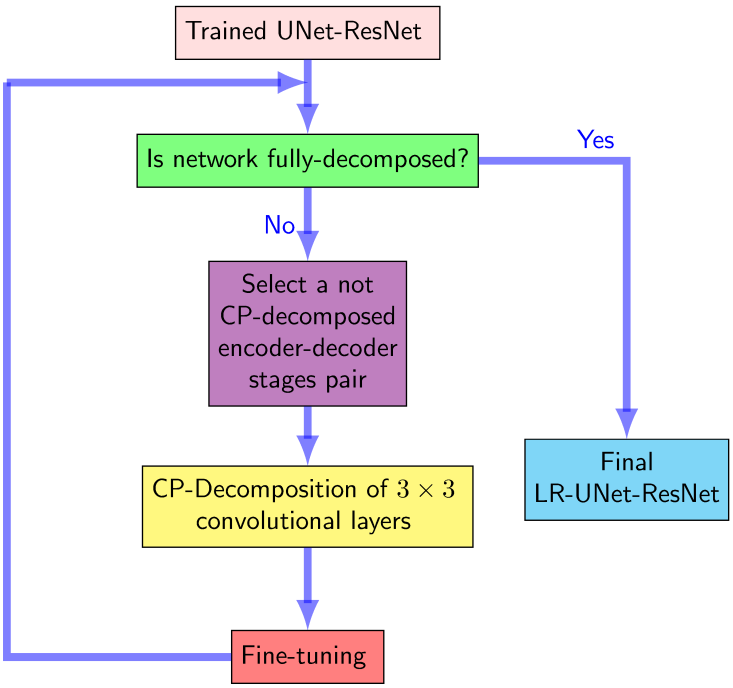


Figura 3.5.5: Flowchart per la compressione della UNet-ResNet

	Compression factor	**Optimizer**	**Learning rate**	**Epochs**
stage #2	0.1	Adam	0.0005	100
stage #1	0.05	Adam	0.001	160
conv2d 7×7	0.05	Adam	0.001	100

Tabella 3.12: Parametri della compressione per la UNet-ResNet V1.

I parametri della compressione quali fattore di compressione per i layer, learning rate e numero di epoche per il finetuning sono riportati per le versioni V1,V2,V3 rispettivamente in Tab. 3.12, 3.13, 3.14.

	Compression factor	**Optimizer**	**Learning rate**	**Epochs**
stage #3	0.1	Adam	0.0005	100
stage #2	0.1	Adam	0.001	100
stage #1	0.1	Adam	0.001	100
conv2d 7×7	0.1	Adam	0.001	100

Tabella 3.13: Parametri della compressione per la UNet-ResNet V2.

Le architetture prodotte dalla compressione sono state denominate LR-UNet-ResNet-V1-V2-V3. I risultati ottenuti dai modelli compressi sono mostrati in Tab. 3.15.

La V1 e la V3 dopo la compressione hanno prestazioni praticamente identiche come accuratezza e mIoU, la V3 tuttavia ha un leggero vantaggio in termini di MFLOPs.

La V2 ha sempre accuratezza e mIoU più bassi delle altre due ma allo stesso tempo ha un significativo vantaggio computazionale.

	Compression factor	**Optimizer**	**Learning rate**	**Epochs**
stage #3	0.1	Adam	0.0005	50
stage #2	0.1	Adam	0.0005	100
stage #1	0.1	Adam	0.001	160
conv2d 7×7	0.1	Adam	0.001	100

Tabella 3.14: Parametri della compressione per la UNet-ResNet V3.

	Test accuracy [%]	**mIoU** [%]	**Storage cost** [MB]	**Parameters number**	**MFLOPs**
LR-UNet-ResNet V1	85.24	54.17	1.966	124k	526.20
LR-UNet-ResNet V2	84.63	52.37	6.630	512k	366.12
LR-UNet-ResNet V3	85.25	54.82	6.751	520k	501.77

Tabella 3.15: Confronto tra i risultati sperimentali per le tre reti compresse LR-UNet-ResNet V1,V2,V3.

3.5.5 Implementazione su piattaforme embedded

Le reti così sviluppate ad alto livello in Tensorflow v.2.4.1, Tensorflow Keras 2.4.0 in Python 3.7.10, sono state testate come primo esperimento su una Raspberry Pi 4 basata su un Quad core Cortex-A72 (ARM v8).

E' stato utilizzato il formato .h5 per i modelli in quanto è quello più utilizzato nello stato dell'arte sulla Semantic Segmentation.

Le prestazioni sono state confrontate con quelle ottenute dalle reti disponibili in letteratura e citate nella sottosez. 3.5.2.

I risultati della sperimentazione su Raspberry Pi 4 sono riportati in Tab. 3.16 in termini di occupazione di memoria (MB), numero di parametri, MFLOPs, accuracy, tempo di inferenza e i frame al secondo (FPS).

Le tre architetture proposte ottengono ottimi livelli di accuratezza ed i migliori valori in assoluto per MFLOPs, tempo di inferenza ed FPS.

L'accuracy più alta è ottenuta dalla DeepLabv3+ per la quale però il tempo di inferenza è oltre 12 volte quello ottenuto dalle LR-UNet-ResNet.

La rete che più si avvicina a quelle proposte come efficienza è la Fast-SCNN che perde da 1.4% a 2% circa di accuracy al variare delle versioni della UNet-ResNet.

Model	**Storage cost** [**MB**]	**Compression factor**	**Parameters number**	**MFLOPs**	**Test accuracy** [%]	**Inference time** [**sec**]	**FPS**
LR-UNet-ResNet V1	**1.966**	10	**142 000**	**526.20**	85.24	**0.554**	**1.80**
LR-UNet-ResNet V2	6.630	10	512 670	**366.12**	84.62	**0.420**	**2.38**
LR-UNet-ResNet V3	6.751	10	520 000	**501.77**	85.24	**0.515**	**1.94**
U-Net [138]	**3.818**	1	**303 876**	7362.83	83.55	1.114	0.89
DeepLabv3+ [150]	472.909	1	41 255 900	68 780.63	**90.13**	6.544	0.15
SegNet [140]	32.840	1	2 848 748	97 646.51	84.53	6.385	0.15
ENet [141]	5.923	1	372 208	2630.39	83.93	1.148	0.87
ICNet [142]	78.044	1	6 741 900	5376.45	83.14	1.096	0.91
ERFNet [144]	24.645	1	2 070 328	17 739.99	86.91	2.087	0.47
BiSeNet [143]	203.008	1	26 533 848	36 342.10	87.95	3.783	0.26
Fast-SCNN [145]	21.080	1	1 797 212	1620.32	83.24	0.613	1.63

Tabella 3.16: Confronto tra la rete proposta e le reti dello stato dell'arte nella sperimentazione su Raspberry Pi 4

Model	Storage cost [MB]	Compression factor	Parameters number	MFLOPs	Test accuracy [%]	Inference time CPU [sec]	Inference time GPU [sec]	FPS with GPU
LR-UNet-ResNet V1	**1.966**	10	**142 000**	**526.20**	85.24	**0.047**	**0.011**	**90.90**
LR-UNet-ResNet V2	6.630	10	512 670	**366.12**	84.62	**0.037**	**0.011**	**90.90**
LR-UNet-ResNet V3	6.751	10	520 000	**501.77**	85.24	**0.043**	**0.012**	**83.33**
U-Net [138]	**3.818**	1	**303 876**	7362.83	83.55	0.092	0.021	47.61
DeepLabv3+ [150]	472.909	1	41 255 900	68 780.63	**90.13**	0.396	0.090	11.11
SegNet [**?**]	32.840	1	2 848 748	97 646.51	84.54	0.259	0.055	18.18
ENet [141]	5.923	1	372 208	2630.39	83.94	0.083	0.030	33.33
ICNet [142]	78.044	1	6 741 900	5376.45	83.14	0.101	0.021	47.61
ERFNet [144]	24.645	1	2 070 328	17 739.99	86.91	0.171	0.036	27.77
BiSeNet [143]	203.008	1	26 533 848	36 342.10	87.96	0.210	0.047	21.27
Fast-SCNN [145]	21.080	1	1 797 212	1620.32	83.24	0.049	0.014	71.42

Tabella 3.17: Confronto tra la rete proposta e le reti dello stato dell'arte nella sperimentazione su desktop e su GPU NVIDIA Tesla K80

La Tab. 3.17 riassume invece i risultati della sperimentazione eseguita su desktop con CPU con in aggiunta i tempi di inferenza e gli FPS ottenuti su una GPU NVIDA Tesla K80.

Quest'ultima è una GPU dalle prestazioni piuttosto elevate e consente di raggiungere ottenere per le UNet-ResNet valori di FPS che arrivano 90.9, compatibili con applicazioni più spinte come l'Autonomous Driving.

3.6 Rete per il Pedestrian-Tracking su sistema embedded

3.6.1 Pedestrian Tracking

Il Pedestrian Tracking è un task della computer vision applicato in moltissimi ambiti come videosorveglianza[151], assistenza ai disabili[152],guida autonoma[153].

Questo task comprende la risoluzione di due sotto-problemi quali la detection di Region-of-Interest (ROI) in cui è alta la probabilità di trovare un oggetto e successivamente la classificazione.

Alcuni algoritmi effettuano la detection delle ROI grazie a finestre dinamiche che vengono fatte scorrere sull'immagine [154, 155, 156]. Essi però richiedono un elevato numero di finestre (anche fino a 10^6), ragion per cui essi non sono adatti ad implementazione su sistemi embedded.

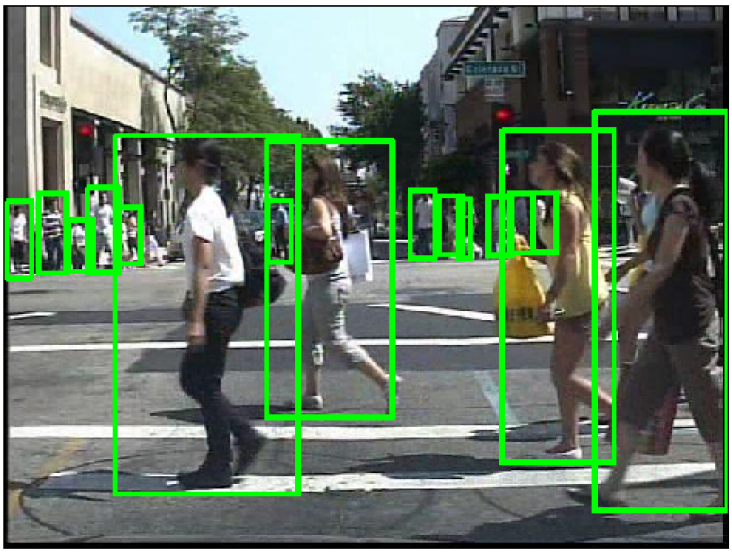


Figura 3.6.1: Esempio di output della detection per il Pedestrian Tracking[8]

Altri approcci individuano le regioni con tecniche di raggruppamento dei pixel, possono basarsi su segmentazione dell'immagine attraverso grafi[157], individuazione dei contorni[158], grouping dei superpixel[159].

3.6.2 CNN per la object detection dello stato dell'arte

Eccellenti risultati nel Pedestrian Tracking sono stati ottenuti grazie alle CNN progettate per la object detection.

Le CNN per object detection nello stato dell'arte si possono raggruppare in due categorie principali: le Region Proposal Networks e i Single-Shot-Detectors.

Region Proposal Networks

Nelle Region Proposal Networks la detection e la classificazione vengono effettuate da due blocchi distinti dell'architettura. Una delle più note ed utilizzate è la Faster-R-CNN[9] la cui architettura è sintetizzata in Fig. 3.6.2.

Vi è all'inizio una VGG pretrainata sul dataset Imagenet usata come backbone per l'estrazione delle feature. Un blocco della rete chiamato Region-Proposal-Network(RPN) si occupa dell'individuazione delle ROI, esse vengono poi utilizzate dal blocco Faster-R-CNN-detector per effettuare un pooling delle feature per poi fare la classificazione.

Il training di questa rete può essere fatto con una tecnica alternata dove la RPN ed il Faster-R-CNN-detector vengono iterativamente addestrate in

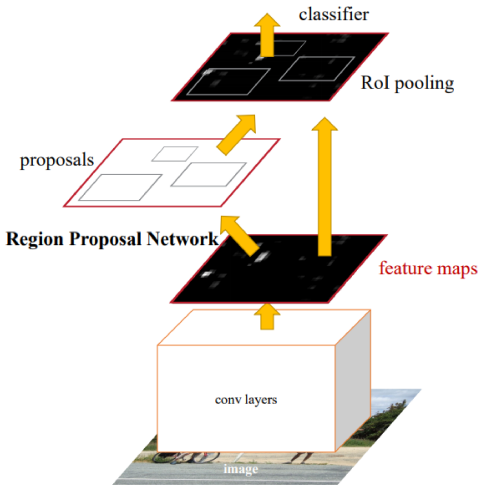


Figura 3.6.2: Schema a blocchi della Faster-R-CNN [9]

modo separato, oppure addestrando i due blocchi congiuntamente con una loss combinata.

Queste reti ottengono prestazioni particolarmente elevate ma sono troppo pesanti per applicazioni su sistemi embedded.

Single Shot Detectors

I Single-Shot-Detectors(SSD) diversamente dalle RPN eseguono contemporaneamente la localizzazione e classificazione condividendo sempre gli stessi pesi per i due task. Il Single-Shot-Detector più noto ed utilizzato è quello proposto da [160].

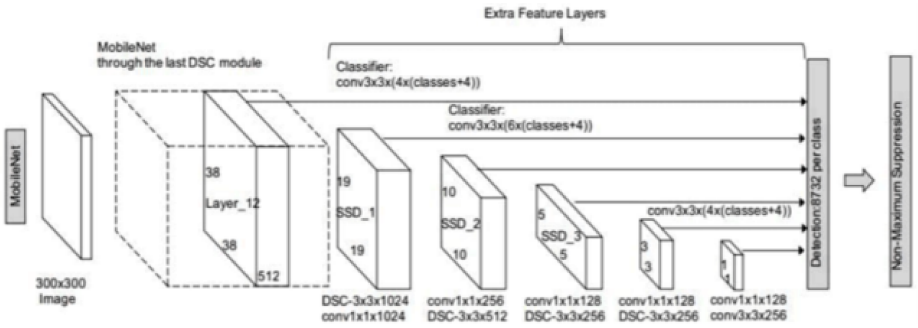


Figura 3.6.3: Architettura Single Shot Detector basato sulla Mobilenet

Come la Faster-R-CNN anche gli SSD sono basati su una backbone iniziale sviluppata per la classificazione e pre-addestrata, sono molto utilizzate la VGG, MobileNet-V1,V2,V3, ResNet.

Per questi detector vengono scelte delle griglie che suddividono l'immagine in celle. A ciascuna di queste celle viene associato un set di boxes dette 'default-boxes' che differiscono tra loro per l'aspect-ratio.

La rete produce in output le predizioni per la classificazione e la localizzazione delle boxes degli oggetti (misurata come offset rispetto alle default-boxes), per ciascuna delle default-boxes e per ogni cella. Questo approccio è schematizzato dalla Fig. 3.6.4.

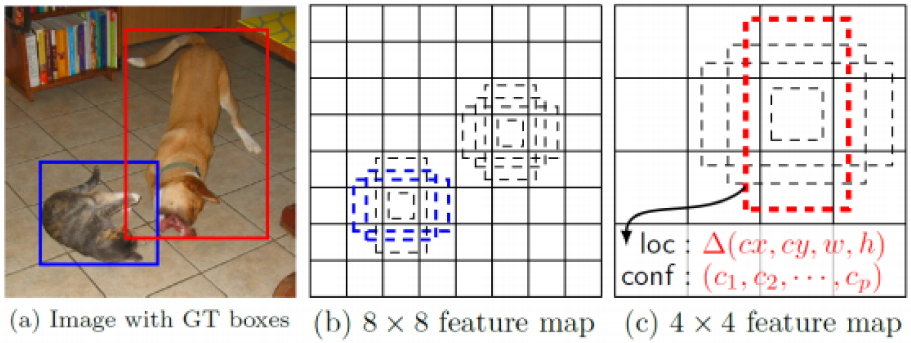


Figura 3.6.4: Detection delle boxes per il Single Shot Detector

Nel SSD in [160] vengono calcolate feature su più scale a cui corrispondo altrettante griglie. Esse vengono poi combinate per ottenere le predizioni che infine passano ad uno stadio di Non-Maximum-Suppression che elimina quelle con il grado di confidenza troppo basso.

YOLO

La YOLO[161] (You-Only-Look-Once) è un semplice tipo di SSD che estrae feature semplicemente attraverso convoluzioni e downsampling, senza combinare feature a scale diverse.

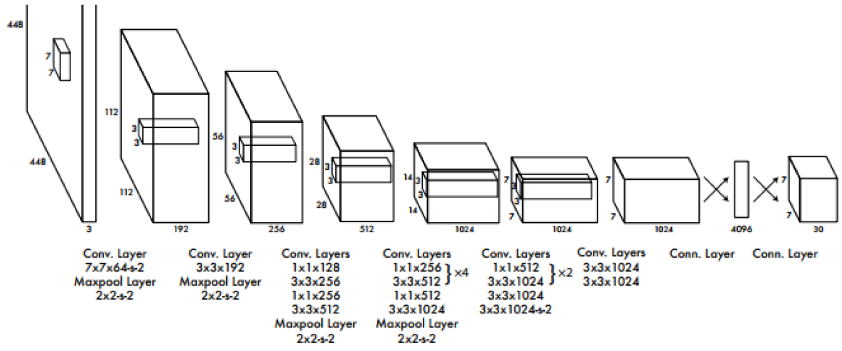


Figura 3.6.5: Architettura originale YOLO

L'immagine viene divisa in una griglia di dimensione $S \times S$, per ognuna delle default-boxes viene fornita in output la predizione delle 4 coordinate della box dell'oggetto e la confidenza in aggiunta a quelle per la classificazione.

La dimensione dell'output è dunque $S \times S \times B(C+5)$, dove B è il numero di default-boxes per cella e C è il numero di classi.

Al posto della classica ReLu per le attivazioni viene utilizzata la Leaky-ReLu che ha una pendenza non nulla anche per input negativi:

$$\phi_{\text{Leaky}}(x) = \begin{cases} x & x \geq 0 \\ 0.1x & x < 0 \end{cases} \tag{3.22}$$

Sono state proposte versioni migliorate YOLOv2[162],YOLOv3[163] che incorporano delle backbone quali DarkNet e ResNet e producono predizioni multiple a scale diverse.

3.6.3 Rete proposta basata sulla Tiny-YOLOv3-DarkNet

Tiny-YOLOv3-DarkNet

Il punto di partenza per il progetto della rete è stata la YOLOv3-DarkNet la cui architettura è stata progettata sulla base della DarkNet-53[164] ed è riportata in Fig. 3.6.6. Vi sono 5 blocchi convoluzionali basati su residual-block descritti in Tab. 3.18 e output su 3 scale diverse.

La YOLOv3-DarkNet è troppo pesante per i nostri scopi a causa dell'elevato numero di convoluzioni complessivo.

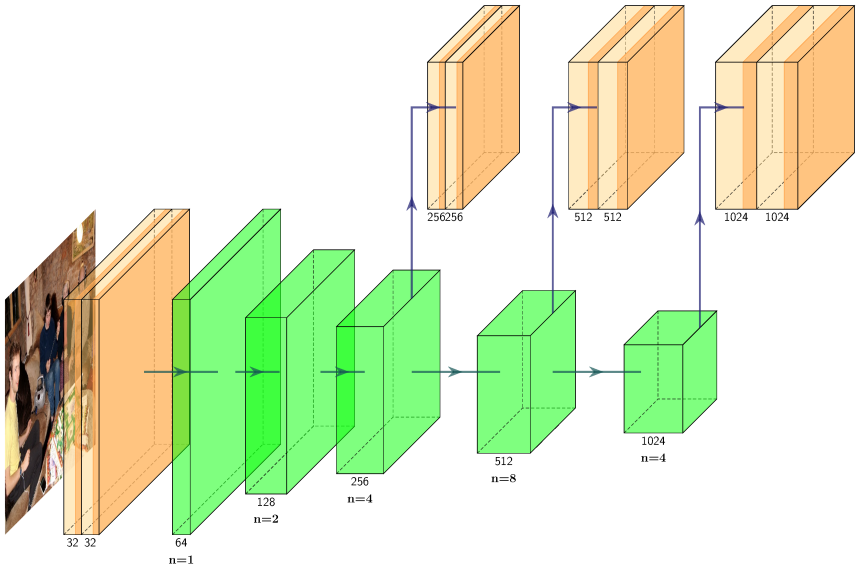


Figura 3.6.6: Architettura YOLOv3-Darknet

		filters	strides	output name
first block	Convolution (3×3)	f	2	-
	BatchNormalization	f	-	-
	Leaky-Relu	f	-	x
conv-block	Convolution (1×1)	$f/2$	1	-
	BatchNormalization	$f/2$	-	-
	Leaky-Relu	$f/2$	-	-
	Convolution (3×3)	f	1	-
	BatchNormalization	f	-	
	Leaky-Relu	f	-	y
	Add (x,y)	f	-	-
Repeat conv-block $n-1$ times				

Tabella 3.18: Residual Block per la YOLOv3

Una versione molto più leggera di questa rete è detta Tiny-YOLOv3-DarkNet. Essa utilizza 2 scale per le predizioni e sostituisce i blocchi descritti in Tab. 3.18 con semplici layer convoluzionali.

	filters	strides	output size	output name
Convolution (3×3)	16	1	$320 \times 320 \times 16$	-
BatchNormalization	16	-	$320 \times 320 \times 16$	-
Leaky-Relu	16	-	$320 \times 320 \times 16$	-
Maxpool	16	2	$160 \times 160 \times 16$	-
Convolution (3×3)	32	1	$160 \times 160 \times 32$	-
BatchNormalization	32	1	$160 \times 160 \times 32$	-
Leaky-Relu	32	-	$160 \times 160 \times 32$	-
Maxpool	32	2	$80 \times 80 \times 32$	-
Convolution (3×3)	64	1	$80 \times 80 \times 64$	-
BatchNormalization	64	-	$80 \times 80 \times 64$	-
Leaky-Relu	64	-	$80 \times 80 \times 64$	-
Maxpool	64	2	$40 \times 40 \times 64$	-
Convolution (3×3)	128	1	$40 \times 40 \times 128$	-
BatchNormalization	128	-	$40 \times 40 \times 128$	-
Leaky-Relu	128	-	$40 \times 40 \times 128$	-
Maxpool	128	2	$20 \times 20 \times 128$	-
Convolution (3×3)	256	1	$20 \times 20 \times 256$	-
BatchNormalization	256	-	$20 \times 20 \times 256$	-
Leaky-Relu	256	-	$20 \times 20 \times 256$	X
Maxpool	256	2	$10 \times 10 \times 256$	-
Convolution (3×3)	512	1	$10 \times 10 \times 512$	-
BatchNormalization	512	-	$10 \times 10 \times 512$	-
Leaky-Relu	512	-	$10 \times 10 \times 512$	-
Convolution (3×3)	1024	1	$10 \times 10 \times 1024$	-
BatchNormalization	1024	-	$10 \times 10 \times 1024$	-
Leaky-Relu	1024	-	$10 \times 10 \times 1024$	-
Convolution (1×1)	256	1	$10 \times 10 \times 256$	-
BatchNormalization	256	-	$10 \times 10 \times 256$	-
Leaky-Relu	256	-	$10 \times 10 \times 256$	Y
Convolution (3×3)	512	1	$10 \times 10 \times 512$	-
Leaky-Relu	512	-	$10 \times 10 \times 512$	-
Convolution (1×1)	75	1	$10 \times 10 \times 75$	out_1
Route12(X,Y)	512	-	$20 \times 20 \times 384$	-
Convolution (3×3)	512	1	$20 \times 20 \times 512$	-
Leaky-Relu	512	-	$20 \times 20 \times 512$	-
Convolution (1×1)	75	1	$20 \times 20 \times 75$	out_2

Tabella 3.19: Architettura Tiny-YOLOv3-Darknet

Troncamento del modello e compressione con decomposizione di Tucker

Per ottenere la rete proposta si è partiti facendo un primo training della Tiny-YOLOv3-DarkNet.

La rete ottenuta è stata successivamente troncata eliminando l'output a risoluzione più bassa sostituendo contemporaneamente le Leaky-ReLu con ReLu classiche. Quest'ultima scelta è stata motivata da ragioni di compatibilità sulla piattaforma embedded come verrà spiegato nella sottosez.3.6.4

L'architettura così ottenuta è stata denominata Tiny-YOLOv3-DarkNet-Lite ed è descritta dalla Tab. 3.20.

	filters	strides	output size
Convolution (3×3)	16	1	$320 \times 320 \times 16$
BatchNormalization	16	-	$320 \times 320 \times 16$
Relu	16	-	$320 \times 320 \times 16$
Maxpool	16	2	$160 \times 160 \times 16$
Convolution (3×3)	32	1	$160 \times 160 \times 32$
BatchNormalization	32	1	$160 \times 160 \times 32$
Relu	32	-	$160 \times 160 \times 32$
Maxpool	32	2	$80 \times 80 \times 32$
Convolution (3×3)	64	1	$80 \times 80 \times 64$
BatchNormalization	64	-	$80 \times 80 \times 64$
Relu	64	-	$80 \times 80 \times 64$
Maxpool	64	2	$40 \times 40 \times 64$
Convolution (3×3)	128	1	$40 \times 40 \times 128$
BatchNormalization	128	-	$40 \times 40 \times 128$
Relu	128	-	$40 \times 40 \times 128$
Maxpool	128	2	$20 \times 20 \times 128$
Convolution (3×3)	256	1	$20 \times 20 \times 256$
BatchNormalization	256	-	$20 \times 20 \times 256$
Relu	256	-	$20 \times 20 \times 256$
Maxpool	256	2	$10 \times 10 \times 256$
Convolution (3×3)	512	1	$10 \times 10 \times 512$
BatchNormalization	512	-	$10 \times 10 \times 512$
Relu	512	-	$10 \times 10 \times 512$
Convolution (3×3)	1024	1	$10 \times 10 \times 1024$
BatchNormalization	1024	-	$10 \times 10 \times 1024$
Relu	1024	-	$10 \times 10 \times 1024$
Convolution (1×1)	256	1	$10 \times 10 \times 256$
BatchNormalization	256	-	$10 \times 10 \times 256$
Relu	256	-	$10 \times 10 \times 256$
Convolution (3×3)	512	1	$10 \times 10 \times 512$
Relu	512	-	$10 \times 10 \times 512$
Convolution (1×1)	75	1	$10 \times 10 \times 75$

Tabella 3.20: Architettura Tiny-YOLOv3-Darknet-Lite

Per raggiungere le prestazioni volute in termini di efficienza il modello è stato poi compresso tramite la decomposizione di Tucker applicata ai layer con filtri di dimensione 3×3. Il rango $[R_3, R_4]$ per la decomposizione di Tucker di ciascun layer è riportato in Tab. 3.21.

E' stato infine effettuato il finetuning per ripristinare l'accuratezza persa a causa della compressione. Diversamente dalla decomposizione CP, è stato possibile fare il retraining in un unico step senza problemi di instabilità del modello.

Il flowchart per la derivazione del modello proposto è mostrato in Fig. 3.6.7.

Layer	Tucker ranks	Approximation error
Convolution 3x3 #1	[9, 19]	0.429
Convolution 3x3 #2	[19, 37]	0.410
Convolution 3x3 #3	[37, 74]	0.450
Convolution 3x3 #4	[74, 149]	0.459
Convolution 3x3 #5	[149, 298]	0.484
Convolution 3x3 #6	[298, 598]	0.460
Convolution 3x3 #7	[149, 298]	0.355

Tabella 3.21: Parametri della compressione della Tiny-YOLOv3-DarkNet-Lite

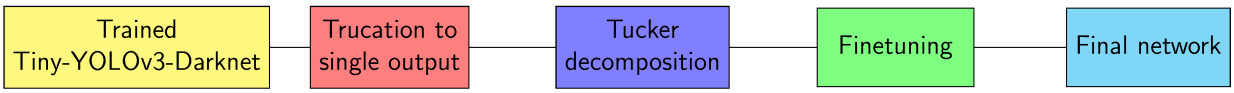


Figura 3.6.7: Flowchart per il progetto della rete proposta

3.6.4 Risultati sperimentali

Training e metriche utilizzate

Per la sperimentazione è stato utilizzato il dataset PASCAL VOC 2007/2012[165] contenente 17.125 immagini suddivise in 20 classi: 'aeroplane', 'bicycle', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'dining table', 'dog', 'horse', 'motorbike', 'person', 'potted plant', 'sheep', 'train', 'TV'.

La Tiny-YOLOv3-DarkNet è stata addestrata partendo da un modello pre-allenato sul COCO[166] che è un dataset molto ampio che include oltre 2.5 milioni di dati etichettati in oltre 328,000 immagini.

La loss utilizzata è data dalla combinazione di una crossentropy loss per la localizzazione e di una mean-square-error loss per la localizzazione delle boxes:

$$L = L_{\text{class}} + \alpha L_{\text{loc}} \tag{3.23}$$

$$L_{\text{class}} = -\sum_{i=1}^{S^2}\sum_{j=1}^{B}\theta_{ij}\left(\log y_{ij}^{\text{conf}} + \sum_{k=1}^{C} t_{ik}\log y_{ijk}\right)$$

$$L_{\text{loc}} = \sum_{c\in\text{coord}}\sum_{j=1}^{B}\sum_{i=1}^{S^2}\theta_{ij}\left(c_{ij} - \widehat{c}_{ij}\right)^2$$

$$\theta_{ij} = \begin{cases} \text{if a object appears in the cell } i \\ 1 \quad \text{and bounding box } j \text{ is responsible} \\ \text{for prediction} \\ 0 \quad \text{otherwise} \end{cases}$$

La Tiny-YOLOv3-DarkNet è stata allenata sul PASCAL VOC su immagini di dimensione 320×320, per 25 epoche con un ottimizzatore di Adam, un learning rate di 0.0001 e con un batch size pari a 32.

E' stato prima fatto un training mantenendo fissi tutti i layer tranne quelli finali di predizione ed un training successivo rilassando la rete e con gli stessi parametri utilizzati in precedenza.

Il finetuning del modello compresso con la decomposizione di Tucker è stato fatto sempre con un ottimizzatore Adam, per 15 epoche, con un learning rate di 0.00001 e un batch size pari a 32.

Le prestazioni di detection sono state misurate con la Average-Precision(AP), tradizionale metrica utilizzata nella object-detection.

Considerate la Precision e la Recall:

$$Prec = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN} \tag{3.24}$$

l'Average-Precision è definita come la media della Precision al variale della Recall:

$$AP = \int_0^1 p(r)dr \tag{3.25}$$

Per classificare una predizione come vera o false viene utilizzato l'indice di overlap misurato come il rapporto fra l'area di overlap tra la box predetta e quella effettiva e l'unione di suddette aree: $IoU = \frac{Overalp}{Union}$. Una soglia comunemente utilizzata per IoU è 0.5.

Test su Raspberry Pi 4 e Coral Edge TPU

Il modello proposto è stato implementato su Raspberry Pi 4 basata su un ARM-Cortex-A72 Quad core. In aggiunta per abbassare il tempo di inferenza del modello è stata connessa tramite USB un'unità Coral.

L'unità USB Coral aggiunge un coprocessore Edge TPU al sistema e consente di accelerare notevolmente l'hardware utilizzato.

Oltre al modello proposto, sono stati implementati sulla board altri tre modelli estremamente efficienti per la object detection quali SSD-Mobilenet-V1,V2 e EfficientDet[167].

I modelli implementati sono stati dapprima convertiti da formato .h5 a .tflite con quantizzazione ad 8 bit. E' stata poi applicata un'ulteriore conversione in .edge_tpu.tflite che è un formato ottimizzato in modo specifico per il coprocessore TPU.

E' stato osservato che le attivazioni Leaky-ReLu non sono ancora state ottimizzate in questo tipo di formato, ragion per cui esse aggiungono un overhead molto pesante al tempo di inferenza che non può essere gestito. Si è scelto pertanto di sostituire le Leaky-ReLu con ReLu tradizionali.

I risultati ottenuti su Raspberry Pi 4 con acceleratore Coral Edge TPU sono riportati in Tab. 3.22.

Model	$mAP_{IOU=0.5}$ [%]	Storage cost [MB]	Parameters number	CPU inference time (1/4 th.) [ms]	TPU inference time [ms]	CPU FPS	TPU FPS
Proposed Network	48.7	**4.0**	4.0 M	**177.92/64.79**	**13.73**	**5.5/16**	**71**
MobileNetv1-SSD	32.9	6.6	5.8 M	271.16/84.51	20.84	3.6/12	48
MobileNetv2-SSD	**72.7**	5.9	6.1 M	260.30/78.59	23.86	3.8/13	42
Tiny YOLO v3 Darknet	60.2	8.5	8.7 M	517.36/160.16	44.65	1.9/6.2	22
EfficientDet-Lite0	70.8	4.4	**3.2** M	409.98/163.52	118.94	2.4/6.1	8.4

Tabella 3.22: Confronto tra il modello proposto e i modelli dello stato dell'arte

Il modello proposto ottiene nettamente le migliori prestazioni in termini di tempo di inferenza e FPS sia su CPU che utilizzando l'acceleratore Coral Edge TPU. Ciò va scapito della mean-Average-Precision (mAP) che comunque si mantiene a un valore accettabile.

Conclusioni

In questa tesi di dottorato è stato presentato lo sviluppo e l'applicazione di tecniche di compressione implementazione di algoritmi in sistemi embedded in tre differenti aree di ricerca.

Il primo progetto di ricerca ha riguardato il Compressed Sensing, per il quale è stato proposto uno studio comparativo dei 4 principali algoritmi di ricostruzione quali Basis Pursuit,OMP,CosAmp,NIHT applicati al segnale EMG. E' stata effettuato un confronto tra tre basi di rappresentazioni DCT,Haar e DB4 ottenendo per quest'ultima di gran lunga i migliori risultati di ricostruzione.

E' stato utilizzato un approccio basato sul controllo della sparsità del segnale prima dell'encoding che ha consentito di ottenere miglioramenti nella qualità di ricostruzione rispetto ai risultati in letteratura.

E' stato ottenuto per i vari algoritmi un massimo nelle prestazioni al variare del parametro $S_M = K/M$ che consente di ottimizzare le prestazioni. Nondimeno sono stati testati gli algoritmi anche con la presenza di rumore.

A completamento dell'attività di ricerca sul Compressed Sensing è stato proposto un miglioramento di un algoritmo di ricostruzione basato sulla Mixed-Integer-Linear-Programming estendendo tramite un semplice shift dei coefficenti il range di valori di K per quali l'algoritmo è in grado di ricostruire il segnale con il 100% di probabilità.

Come secondo tema oggetto di ricerca è stato proposta una tecnica di Manifold Learning per la creazione di un modello causale di biosegnali.

E' stata dapprima sviluppato un nuovo algoritmo per la stima della dimensione intriseca per il Manifold considerato, basata sulla stima del gradiente con una regressione di Nadaraya-Watson ottimizzata mediante un'analisi teorica dell'errore di stima. Tale metodo supera le difficoltà delle tecniche di stima della dimensionalità dello stato dell'arte.

E' stato fatto il learning del modello tramite regressione con Polinomi di Bernstein. Il modello è stato applicato per la ricostruzione e soprattutto per la generazione di nuove traiettorie rispetto al training set. E' stata verifica-

ta la superiorità della tecnica proposta rispetto alle Generative Adversarial Networks in termini di simililarità dei dati generati rispetto a quelli utilizzati per il training.

Il modello generativo è stato poi applicato con successo alla data augmentation per la classificazione del segnale.

La terza tematica ha riguardato lo sviluppo di Convolutional Neural Networks compresse con tecniche di decomposizione tensoriali(quali decomposizione CP e di Tucker) per sistemi embedded.

E' stata sviluppata una rete denominata 'LR-Net' compressa tramite decomposizione CP, per il monitoraggio in real time del mal dell'esca su una OpenMV Cam con un microcontrollore STM32H743II ARM Cortex-M7.

E' stata proposta una rete denominata 'LR-UNet-ResNet' per la Semantic Segmentation in real-time compressa sempre mediante decomposizione CP ed implementata su Raspberry Pi 4 che su GPU.

E' stata infine sviluppata una rete a partire dalla Tiny-YOLOv3, compressa mediante la decomposizione di Tucker per il Pedestrian-Tracking su Raspberry Pi 4 con acceleratore Coral.

Tutte le reti proposte ottengono ottimi livelli di accuratezza ed i tempi di inferenza misurati su piattaforma embedded più bassi rispetto alle più efficienti reti disponibili nello stato dell'arte.

Nei primi due progetti di ricerca presentati in questa tesi sono state proposte e/o implementate tecniche di compressione basate sulla intrinseca comprimibilità dei dati.

Nel Compressed Sensing infatti essi hanno un legame esprimibile tramite una trasformazione lineare sia con lo spazio a dimensione ridotta dei coefficienti, sia con la effettiva rappresentazione compressa di essi.

Il Manifold Learning può essere riguardato da un certo punto di vista come una generalizzazione del Compressed Sensing, dove tra lo spazio a dimensione piena e quello a dimensione ridotta esiste un legame nonlineare.

Nella terza tematica affrontata, più di carattere applicativo, sono state utilizzate tecniche di approssimazione low-rank per tensori non direttamente legate alla rappresentazione in forma compatta da un punto di vista teorico.

Appendice A

In questa Appendice è fornita una dimostrazione per l'espressione dell'errore di stima del gradiente con la regressione di Nararaya Watson, vedi eq. 2.16.

E' stata condotta una generalizzazione a p dimensioni dell'analisi effettuata in [88] che verrà poi utilizzata per ottimizzare la larghezza di banda H_k.

In maniera analoga a [88], l'errore di stima del gradiente $\mathcal{E}_{H_k}(u)$ può essere scritto come:

$$\mathcal{E}_{H_k}(u) = \nabla\widehat{f} - \nabla f = \frac{1}{\widehat{g}(u)}\left(\nabla\widehat{h} - \nabla h\right) - \frac{\nabla f}{\widehat{g}(u)}\left(\widehat{g}(u) - g(u)\right) - \\ - \frac{\widehat{f}(u)}{\widehat{g}(u)}\left(\nabla\widehat{g} - \nabla g\right) - \frac{\nabla g}{\widehat{g}(u)}\left(\widehat{f}(u) - f(u)\right) \tag{26}$$

Calcoliamo ora media e varianza di ciascun termine con uno sviluppo in serie di Taylor di $f(t)$ e $g(t)$ attorno al punto di stima x:

$$f(t) \approx f(u) + \nabla f^T(t-u) + \frac{1}{2}(t-u)^T H_f(u)(t-u) \\ g(t) \approx g(u) + \nabla g^T(t-u) + \frac{1}{2}(t-u)^T H_g(u)(t-u) \tag{27}$$

Se ora sostituiamo $t = x - H_k y$ negli integrali $t = x - H_k y$, e indichiamo $z = H_k y$:

$$E(c_k(u)) = \int_{\mathbb{R}^p} \frac{1}{D_k} K\left(H_k^{-1}(t-u)\right) g(t)dt = \\ = \int_{\mathbb{R}^p} K(y)\left(g(u) - \nabla g^T H_k y + \frac{1}{2} y^T H_k^T H_g(u) H_k y\right) dt = \\ = g(u) + \int_{\mathbb{R}^p} \frac{1}{2} y^T H_k^2 H_g(u) y K(y) dy = \\ = g(u) + \frac{1}{2}\operatorname{Tr}\left(H_k^2 H_g(u)\right) = g(u) + \frac{1}{2}\sum_{j=1}^{p} h_j^2 \frac{\partial^2 g}{\partial u_j^2} \tag{28}$$

$$E(c_k^2(u)) = \int_{\mathbb{R}^p} \frac{1}{D_k^2} K^2\left(H_k^{-2}(t-u)\right) g(t)dt =$$
$$= \frac{1}{D_k^2} D_k \int_{\mathbb{R}^p} K^2(y) \left(g(u) - \nabla g^T H_k y + \frac{1}{2} y^T H_k^2 H_g(u) y \right) dy =$$
$$= g(u) \frac{\int_{\mathbb{R}^p} K^2(y)dy}{D_k} + \frac{1}{D_k} \int_{\mathbb{R}^p} y^T H_k^2 H_g(u) y K^2(y)dy \tag{29}$$

Per un generico kernel possiamo definire:

$$\xi_m = \int_{\mathbb{R}^p} y_i^m K^2(y)dy \ , \ i = 1, \ldots, p$$

Pertanto si può scrivere:

$$E(c_k^2(u)) = \frac{1}{D_k} \left(\xi_0 g(u) + \frac{\xi_2}{2} \operatorname{Tr}(H_k^2 H_g(u)) \right) \tag{30}$$

Ciascun termine differisce dalla quantità da stimare sia in media che in varianza; è molto difficile stimare il bias direttamente poiché esso dipende dalle derivate seconde di $g(u)$ ed è un polinomio del second'ordine di H_k: $E(\widehat{g}(u) - g(u)) = O(H_k^2)$. Allo stesso modo calcoliamo media e varianza degli altri termini:

$$E(d_k(u)) = -g(u) \int_{\mathbb{R}^p} H_k^{-1} y K(y)dy + \nabla g -$$
$$- \frac{H_k}{2} \int_{\mathbb{R}^p} y(y^T H_k^2 H_g(u) y) K(y)dy = \nabla g + O(H_k^2) \tag{31}$$
$$E(||d_k(u)||_2^2) = \frac{g(u)}{D_k} \int_{\mathbb{R}^p} ||H_k^{-1} y||_2^2 K^2(y)dy$$
$$- \frac{\nabla g^T}{D_k} \int_{\mathbb{R}^p} H_k y ||H_k^{-1} y||_2^2 K^2(y)dy +$$
$$+ \frac{1}{2D_k} \int_{\mathbb{R}^p} y^T H_k^2 H_g(u) y ||H_k^{-1} y||_2^2 K^2(y)dy =$$
$$= \frac{\xi_2 p \left\langle \frac{1}{H_k(j)^2} \right\rangle}{D_k} g(u) + O\left(\frac{R(H_k^2)}{D_k} \right) \tag{32}$$

dove $R(H_k^2)$ è una funzione razionale del second'ordine di $H_k(j)^2$.

$$R(H_k) = \frac{\sum_{j=1}^{p} a_j H_k^2(j)}{\sum_{j=1}^{p} b_j H_k^2(j)} \tag{33}$$

$$E(b_k(u)) = f(u)g(u) + O(H_k^2)$$
$$E(b_k(u)^2) = \xi_0 \frac{f^2(u)g(u)}{D_k} + O\left(\frac{R(H_k^2)}{D_k}\right) \tag{34}$$

$$E(a_k(u)) = \nabla(fg) + O(H_k^2)$$
$$E(||a_k(u)||_2^2) = \frac{\xi_2 p \left\langle \frac{1}{H_k(j)^2} \right\rangle}{D_k} f^2(u)g(u) + O\left(\frac{R(H_k^2)}{D_k}\right) \tag{35}$$

Valutando il modulo dell'errore dalla 26 otteniamo in aggiunta anche termini di cross-correlazione:

$$\begin{aligned} & g^2(u)E(||\nabla\widehat{f} - \nabla f||_2^2) = E(||\nabla\widehat{h} - \nabla h||_2^2) \\ & + f^2(u)E(||\nabla\widehat{g} - \nabla g||_2^2) + \\ & + ||\nabla f||_2^2 E(\widehat{g}(u) - g(u))^2 \\ & - 2f(u)E\left[(\nabla\widehat{h} - \nabla h)^T(\nabla\widehat{g} - \nabla g)\right] - \\ & - 2\nabla f^T E\left[(\nabla\widehat{h} - \nabla h)(\widehat{g}(u) - g(u)\right] \\ & + 2f(u)\nabla f^T E\left[(\nabla\widehat{g} - \nabla g)(\widehat{g}(u) - g(u))\right. \end{aligned} \tag{36}$$

Valutando questi termini abbiamo:

$$E(\nabla\widehat{h}^T\nabla\widehat{g}) = \frac{\xi_2 p \left\langle \frac{1}{H_k(j)^2} \right\rangle}{D_k} f(u)g(u) + O(\frac{R(H_k^2)}{D_k})$$
$$E(\widehat{g}(u)\nabla\widehat{h}) = O(\frac{R(H_k^2)}{D_k})$$
$$E(\widehat{g}(u)\nabla\widehat{g}) = O(\frac{R(H_k^2)}{D_k}) \tag{37}$$

Sostituendo quanto ottenuto nella 37 non è difficile verificare che analogamente al caso 1-dimensionale [88], il termine complessivo dovuto alla varianza e proporzionale a $\frac{R(H_k^2)}{D_k}$ dipende dalla potenza di rumore $\sigma^2 = E(\epsilon_k^2)$.

L'errore complessivo sarà infine dato da:

$$\mathcal{E}_{H_k}^2(u) = \frac{\sigma^2 \xi_2 p}{g(u)N^2} \sum_{k=1}^{N} \frac{\left\langle \frac{1}{H_k(j)^2} \right\rangle}{D_k} + \frac{1}{N^2} \left\| \sum_{k=1}^{N} B_k(u) \right\|_2^2 \tag{38}$$

dove $B_k(u)$ è il contributo del punto k-esimo al bias: $\text{Bias}(u) = E(\nabla\widehat{f} - \nabla f)$.

Pubblicazioni

[1] L. Manoni, C. Turchetti, L. Falaschetti, and P. Crippa, "A comparative study of computational methods for compressed sensing reconstruction of EMG signal," *Sensors*, vol. 19, no. 16, p. 3531, 2019.

[2] L. Manoni, C. Turchetti, and L. Falaschetti, "An effective manifold learning approach to parametrize data for generative modeling of biosignals," *IEEE Access*, vol. 8, pp. 207 112–207 133, 2020.

[3] C. Turchetti, L. Falaschetti, and L. Manoni, "Principal tensor embedding for unsupervised tensor learning," *IEEE Access*, vol. 8, pp. 225 240–225 257, 2020.

[4] M. Alessandrini, G. Biagetti, P. Crippa, L. Falaschetti, L. Manoni, and C. Turchetti, "Singular value decomposition in embedded systems based on arm cortex-m architecture," *Electronics*, vol. 10, no. 1, p. 34, Dec 2020. [Online]. Available: http://dx.doi.org/10.3390/electronics10010034

[5] L. Falaschetti, L. Manoni, R. C. F. Rivera, D. Pau, G. Romanazzi, O. Silvestroni, V. Tomaselli, and C. Turchetti, "A low-cost, low-power and real-time image detector for grape leaf esca disease based on a compressed cnn," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 3, pp. 468–481, 2021.

Bibliografia

[1] D. Gangopadhyay, E. G. Allstot, A. M. R. Dixon, K. Natarajan, S. Gupta, and D. J. Allstot, "Compressed sensing analog front-end for bio-sensor applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 2, pp. 426–438, 2014.

[2] L. Manoni, C. Turchetti, L. Falaschetti, and P. Crippa, "A comparative study of computational methods for compressed sensing reconstruction of EMG signal," *Sensors*, vol. 19, no. 16, p. 3531, 2019.

[3] N. B. Karahanoglu, H. Erdogan, and S. I. Birbil, "A mixed integer linear programming formulation for the sparse recovery problem in compressed sensing," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5870–5874, 2013.

[4] L. Manoni, C. Turchetti, and L. Falaschetti, "An effective manifold learning approach to parametrize data for generative modeling of biosignals," *IEEE Access*, vol. 8, pp. 207 112–207 133, 2020.

[5] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, pp. 455–500, 08 2009.

[6] L. Falaschetti, L. Manoni, R. C. F. Rivera, D. Pau, G. Romanazzi, O. Silvestroni, V. Tomaselli, and C. Turchetti, "A low-cost, low-power and real-time image detector for grape leaf esca disease based on a compressed cnn," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 3, pp. 468–481, 2021.

[7] D. S. Kim, Y. H. Kim, and K. R. Park, "Semantic segmentation by multi-scale feature extraction based on grouped dilated convolution module," *Mathematics*, vol. 9, no. 9, 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/9/947

[8] D. Park, D. Ramanan, and C. Fowlkes, "Multiresolution models for object detection," in *Computer Vision – ECCV 2010*, K. Daniilidis,

P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 241–254.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[10] C.-H. Kuo, H.-W. Yeh, C.-E. Wu, and K.-M. Hsiao, "Development of autonomous robotic wheelchair controller using embedded systems," in *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, 2007, pp. 3001–3006.

[11] S. Liang, C. Tang, X. Ning, S. Zeng, J. Yu, Y. Wang, K. Guo, D. Yang, T. Lu, and H. Yang, "Efficient computing platform design for autonomous driving systems," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 734–741. [Online]. Available: https://doi.org/10.1145/3394885.3431620

[12] K. Karunanithy and B. Velusamy, "Edge device based efficient data collection in smart health monitoring system using wireless body area network," *Biomedical Signal Processing and Control*, vol. 72, p. 103280, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809421008776

[13] Y. Xu and T. T. Qiu, "Human activity recognition and embedded application based on convolutional neural network," *Journal of Artificial Intelligence and Technology*, vol. 1, no. 1, p. 51–60, Dec. 2020. [Online]. Available: https://ojs.istp-press.com/jait/article/view/6

[14] S. Yue, "Image recognition of competitive aerobics movements based on embedded system and digital image processing," *Microprocessors and Microsystems*, vol. 82, p. 103925, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933121001046

[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[16] J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Edge-ai in lora-based health monitoring: Fall detection system with fog computing and lstm recurrent neural networks," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, 2019, pp. 601–604.

[17] M. Y. Mehmood, A. Oad, M. Abrar, H. M. Munir, S. F. Hasan, H. A. u. Muqeet, and N. A. Golilarz, "Edge computing for iot-enabled smart grid," *Security and Communication Networks*, vol. 2021, p. 5524025, Jul 2021. [Online]. Available: https://doi.org/10.1155/2021/5524025

[18] O. Ferraz, V. Silva, and G. Falcao, "Hyperspectral parallel image compression on edge gpus," *Remote Sensing*, vol. 13, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/6/1077

[19] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, "Real-time driver drowsiness detection for embedded system using model compression of deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[20] Y. W. Choi and J. W. Baek, "Edge camera system using dee p learning method with model compression on embedded applications," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 1–4.

[21] L. Brillet, N. Leclaire, S. Mancini, M. Nicolas, S. Cleyet-Merle, J.-P. Henriques, and C. Delnondedieu, "Compression and speed-up of convolutional neural networks through dimensionality reduction for efficient inference on embedded multiprocessor," *Journal of Signal Processing Systems*, 01 2021.

[22] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, March 2008.

[23] E. J. Candès, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9, pp. 589–592, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1631073X08000964

[24] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[25] R. Cavallari, F. Martelli, R. Rosini, C. Buratti, and R. Verdone, "A survey on wireless body area networks: Technologies and design challenges," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1635–1657, Third 2014.

[26] S. Movassaghi, M. Abolhasan, J. Lipman, D. Smith, and A. Jamalipour, "Wireless body area networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1658–1686, Third 2014.

[27] G. Biagetti, P. Crippa, A. Curzi, S. Orcioni, and C. Turchetti, "Analysis of the EMG signal during cyclic movements using multicomponent AM–FM decomposition," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 5, pp. 1672–1681, Sep. 2015.

[28] H. Ghasemzadeh, R. Jafari, and B. Prabhakaran, "A body sensor network with electromyogram and inertial sensors: Multimodal interpretation of muscular activities," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 2, pp. 198–206, March 2010.

[29] X. Zhang, X. Chen, Y. Li, V. Lantz, K. Wang, and J. Yang, "A framework for hand gesture recognition based on accelerometer and EMG sensors," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 6, pp. 1064–1076, Nov 2011.

[30] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct 2016, pp. 1–8.

[31] K. Luo, Z. Cai, K. Du, F. Zou, X. Zhang, and J. Li, "A digital compressed sensing-based energy-efficient single-spot bluetooth ecg node," *Journal of Healthcare Engineering*, vol. 2018, p. 2687389, Jan 2018. [Online]. Available: https://doi.org/10.1155/2018/2687389

[32] H. Mamaghanian, N. Khaled, D. Atienza, and P. Vandergheynst, "Compressed sensing for real-time energy-efficient ecg compression on wireless body sensor nodes," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 9, pp. 2456–2466, 2011.

[33] L. F. Polanía and R. I. Plaza, "Compressed sensing ecg using restricted boltzmann machines," *Biomedical Signal Processing and Control*, vol. 45, pp. 237–245, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809418301307

[34] S. Aviyente, "Compressed sensing framework for eeg compression," in *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, 2007, pp. 181–184.

[35] Z. Zhang, T.-P. Jung, S. Makeig, and B. D. Rao, "Compressed sensing of eeg for wireless telemonitoring with low energy consumption and inexpensive hardware," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 1, pp. 221–224, 2013.

[36] A. Majumdar and R. K. Ward, "Energy efficient eeg sensing and transmission for wireless body area networks: A blind compressed sensing approach," *Biomedical Signal Processing and Control*, vol. 20, pp. 1–9, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809415000385

[37] A. Casson and E. Rodriguez-Villegas, "Signal agnostic compressive sensing for body area networks: Comparison of signal reconstructions," *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2012, pp. 4497–500, 08 2012.

[38] M. Andrle and L. Rebollo-Neira, "Cardinal b-spline dictionaries on a compact interval," *Applied and Computational Harmonic Analysis*, vol. 18, pp. 336–346, 2005.

[39] F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 744–756, March 2012.

[40] A. Ravelomanantsoa, H. Rabah, and A. Rouane, "Compressed sensing: A simple deterministic measurement matrix and a fast recovery algorithm," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 12, pp. 3405–3413, Dec 2015.

[41] A. Ravelomanantsoa, A. Rouane, H. Rabah, N. Ferveur, and L. Collet, "Design and implementation of a compressed sensing encoder: Application to EMG and ECG Wireless biosensors," *Circuits, Systems, and Signal Processing*, vol. 36, no. 7, pp. 2875–2892, Jul 2017.

[42] A. Marchioni, M. Mangia, F. Pareschil, R. Rovatti, and G. Setti, "Rakeness-based compressed sensing of surface ElectroMyoGraphy for improved hand movement recognition in the compressed domain," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct 2018, pp. 1–4.

[43] A. M. R. Dixon, E. G. Allstot, D. Gangopadhyay, and D. J. Allstot, "Compressed sensing system considerations for ECG and EMG Wireless biosensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 2, pp. 156–166, April 2012.

[44] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, Dec 2007.

[45] D. Needell and J. Tropp, "CoSaMP: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, no. 3, pp. 301 – 321, 2009.

[46] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 265–274, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1063520309000384

[47] G. B. Dantzig, *Origins of the Simplex Method*. New York, NY, USA: Association for Computing Machinery, 1990, p. 141–151. [Online]. Available: https://doi.org/10.1145/87252.88081

[48] E. Berg and M. Friedlander, "Probing the pareto frontier for basis pursuit solutions," *SIAM J. Scientific Computing*, vol. 31, pp. 890–912, 11 2008.

[49] J. Yang and Y. Zhang, "Alternating direction algorithms for l1-problems in compressive sensing," *SIAM Journal on Scientific Computing*, vol. 33, no. 1, pp. 250–278, 2011. [Online]. Available: https://doi.org/10.1137/090777761

[50] E. C and J. Romberg, "l1-magic: Recovery of sparse signals via convex programming," 2005.

[51] S. Zhang and P. Wu, "High accuracy low precision qr factorization and least square solver on gpu with tensorcore," 2019.

[52] J. A. Tropp, "Greed is good: algorithmic results for sparse approximation," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2231–2242, Oct 2004.

[53] X. Cai, Z. Zhou, Y. Yang, and Y. Wang, "Improved sufficient conditions for support recovery of sparse signals via orthogonal matching pursuit," *IEEE Access*, vol. 6, pp. 30 437–30 443, 2018.

[54] F. Ren and D. Marković, "A configurable 12–237 ks/s 12.8 mw sparse-approximation engine for mobile data aggregation of compressively sampled physiological signals," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 68–78, 2016.

[55] H. Zhu, W. Chen, and Y. Wu, "Efficient implementations for orthogonal matching pursuit," *Electronics*, vol. 9, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/9/1507

[56] Å. Björck, *Numerical Methods for Least Squares Problems.* Society for Industrial and Applied Mathematics, 1996. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611971484

[57] T. Blumensath and M. E. Davies, "Normalized iterative hard thresholding: Guaranteed stability and performance," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 2, pp. 298–309, April 2010.

[58] E. Ollila, H.-J. Kim, and V. Koivunen, "Robust iterative hard thresholding for compressed sensing," in *2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, 2014, pp. 226–229.

[59] G. Biagetti, P. Crippa, L. Falaschetti, S. Orcioni, and C. Turchetti, "A portable wireless sEMG and inertial acquisition system for human activity monitoring," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10209 LNCS, pp. 608–620, 2017.

[60] G. Biagetti, P. Crippa, L. Falaschetti, and C. Turchetti, "Classifier level fusion of accelerometer and sEMG signals for automatic fitness activity diarization," *Sensors*, vol. 18, no. 9, 2018.

[61] "PhysioBank," https://physionet.org/physiobank/, accessed: 2019-03-19.

[62] "Neuroelectric and Myoelectric Databases - Examples of Electromyograms," https://physionet.org/physiobank/database/emgdb/, accessed: 2019-03-19.

[63] S. Takriti, *Interfaces*, vol. 24, no. 3, pp. 144–146, 1994. [Online]. Available: http://www.jstor.org/stable/25061891

[64] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[65] A. Rahmani and M. Yousefikhoshbakht, "An effective branch-and-cut algorithm in order to solve the mixed integer bi-level programming," *International Journal of Production Management and Engineering*, vol. 5, no. 1, pp. 1–10, 2017. [Online]. Available: https://polipapers.upv.es/index.php/IJPME/article/view/6512

[66] "AstarOMP software package," http://myweb.sabanciuniv.edu/karahanoglu/research/.

[67] "Threshold-ISD software package," http://www.caam.rice.edu/~optimization/L1/ISD/.

[68] "SL0 matlab code," http://ee.sharif.edu/SLzero/.

[69] D. Farina, A. Crosetti, and R. Merletti, "A model for the generation of synthetic intramuscular EMG signals to test decomposition algorithms," *IEEE Transactions on Biomedical Engineering*, vol. 48, no. 1, pp. 66–77, 2001.

[70] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith, "A dynamical model for generating synthetic electrocardiogram signals," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 3, pp. 289–294, 2003.

[71] G. Shi, H. Huang, and L. Wang, "Unsupervised dimensionality reduction for hyperspectral imagery via local geometric structure feature learning," *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 8, pp. 1425–1429, 2020.

[72] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[73] S. Løkse, F. M. Bianchi, and R. Jenssen, "Training echo state networks with regularization through dimensionality reduction," *Cognitive Computation*, vol. 9, no. 3, pp. 364–378, 2017.

[74] R. Talmon, S. Mallat, H. Zaveri, and R. R. Coifman, "Manifold learning for latent variable inference in dynamical systems," *IEEE Transactions on Signal Processing*, vol. 63, no. 15, pp. 3843–3856, Aug 2015.

[75] M. H. C. Law and A. K. Jain, "Incremental nonlinear dimensionality reduction by manifold learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 3, pp. 377–391, March 2006.

[76] T. Lin and H. Zha, "Riemannian manifold learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 5, pp. 796–809, May 2008.

[77] Z. Zhang, J. Wang, and H. Zha, "Adaptive manifold learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 253–265, Feb 2012.

[78] Y. Wang, J. Yang, and H. Liu, "Acoustic targets feature extraction method based on manifold learning," *Electronics Letters*, vol. 48, no. 3, pp. 139–140, February 2012.

[79] H. Qiao, P. Zhang, D. Wang, and B. Zhang, "An explicit nonlinear mapping for manifold learning," *IEEE Transactions on Cybernetics*, vol. 43, no. 1, pp. 51–63, Feb 2013.

[80] V. S. Tomar and R. C. Rose, "A family of discriminative manifold learning algorithms and their application to speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 1, pp. 161–171, Jan 2014.

[81] X. Xing, K. Wang, Z. Lv, Y. Zhou, and S. Du, "Fusion of local manifold learning methods," *IEEE Signal Processing Letters*, vol. 22, no. 4, pp. 395–399, April 2015.

[82] X. Xu, Z. Huang, L. Zuo, and H. He, "Manifold-based reinforcement learning via linear reconstruction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 934–947, April 2017.

[83] T. Shnitzer, R. Talmon, and J. J. Slotine, "Manifold learning with contracting observers for data-driven time-series analysis," *IEEE Transactions on Signal Processing*, vol. 65, no. 4, pp. 904–918, Feb 2017.

[84] H. Gu, X. Wang, X. Chen, S. Deng, and J. Shi, "Manifold learning by curved cosine mapping," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 10, pp. 2236–2248, Oct 2017.

[85] C. Turchetti and L. Falaschetti, "A manifold learning approach to dimensionality reduction for modeling data," *Information Sciences*, vol. 491, pp. 16 – 29, 2019.

[86] S. Harada, H. Hayashi, and S. Uchida, "Biosignal generation and latent variable analysis with recurrent generative adversarial networks," *IEEE Access*, vol. 7, pp. 144 292–144 302, 2019.

[87] H. Cramér and M. R. Leadbetter, *Stationary and related stochastic processes: Sample function properties and their applications.* Wiley, 1967.

[88] B. Bercu, S. Capderou, and G. Durrieu, "Nonparametric estimation of the derivative of the regression function: application to sea shores water quality," *arXiv: Statistics Theory*, 2016.

[89] A. V. Dobrovidov and I. Ruds'Ko, "Bandwidth selection in nonparametric estimator of density derivative by smoothed cross-validation method," *Automation and Remote Control*, vol. 71, no. 2, pp. 209–224, 2010.

[90] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, Mar 1990.

[91] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[92] A. Bagnall, J. Lines, W. Vickers, and E. Keogh, "The UEA & UCR time series classification repository," 2018. [Online]. Available: http://www.timeseriesclassification.com

[93] R. T. Farouki, "The Bernstein polynomial basis: A centennial retrospective," *Comput. Aided Geom. Des.*, vol. 29, no. 6, pp. 379–419, 2012.

[94] G. Biagetti, P. Crippa, L. Falaschetti, and C. Turchetti, "Machine learning regression based on particle Bernstein polynomials for nonlinear system identification," in *IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2017)*, 2017, in press.

[95] G. Biagetti, P. Crippa, and L. Falaschetti, "A machine learning approach to the identification of dynamical nonlinear systems," 09 2019, pp. 1–5.

[96] Q. Wen, L. Sun, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020.

[97] S. Harada, H. Hayashi, and S. Uchida, "Biosignal data augmentation based on generative adversarial networks," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2018, pp. 368–371.

[98] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[99] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[100] X. Chen, Y. Wang, Y. Zhang, P. Du, C. Xu, and C. Xu, "Multi-task pruning for semantic segmentation networks," 2020.

[101] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured pruning for efficient convnets via incremental regularization," 2018.

[102] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5068–5076, 2017.

[103] R. Goyal, J. Vanschoren, V. van Acht, and S. Nijssen, "Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms," 2021.

[104] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784–5789, 2018.

[105] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[106] W. Ahmed, A. Zunino, P. Morerio, and V. Murino, "Compact cnn structure learning by knowledge distillation," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 6554–6561.

[107] C. Yang, Z. An, L. Cai, and Y.-J. Xu, "Hierarchical self-supervised augmented knowledge distillation," 08 2021, pp. 1217–1223.

[108] H.-T. Li, S.-C. Lin, C.-Y. Chen, and C.-K. Chiang, "Layer-level knowledge distillation for deep neural network learning," *Applied Sciences*, vol. 9, no. 10, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/10/1966

[109] H. Yang, M. Tang, W. Wen, F. Yan, D. Hu, A. Li, H. Li, and Y. Chen, "Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification," 2020.

[110] L. F. Brillet, S. Mancini, S. Cleyet-Merle, and M. Nicolas, "Tunable cnn compression through dimensionality reduction," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3851–3855.

[111] M. Astrid and S.-I. Lee, "Cp-decomposition with tensor power method for convolutional neural networks compression," 2017.

[112] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[113] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[114] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2016.

[115] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[116] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[117] A.-H. Phan, P. Tichavský, and A. Cichocki, "Fast alternating ls algorithms for high order candecomp/parafac tensor factorizations," *IEEE Transactions on Signal Processing*, vol. 61, no. 19, pp. 4834–4846, 2013.

[118] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," 2018.

[119] J. Huang, L. Kong, X.-Y. Liu, W. Qu, and G. Chen, "A c++ library for tensor decomposition," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, 2019, pp. 1–2.

[120] J. Boulent, S. Foucher, J. Théau, and P.-L. St-Charles, "Convolutional neural networks for the automatic identification of plant diseases," *Frontiers in Plant Science*, vol. 10, p. 941, 2019.

[121] Z. Tang, J. Yang, Z. Li, and F. Qi, "Grape disease image classification based on lightweight convolution neural networks and channelwise attention," *Computers and Electronics in Agriculture*, vol. 178, p. 105735, 2020.

[122] M. Agarwal, S. K. Gupta, and K. K. Biswas, "Grape disease identification using convolution neural network," in *2019 23rd International Computer Science and Engineering Conference (ICSEC)*, 2019, pp. 224–229.

[123] M. Alessandrini, R. Rivera, L. Falaschetti, D. Pau, V. Tomaselli, and C. Turchetti, "A grapevine leaves dataset for early detection and classification of Esca disease in vineyards through machine learning," *Data in Brief*, p. 106809, 2021.

[124] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[125] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[126] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," *CoRR*, vol. abs/1905.02244, 2019.

[127] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[128] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[129] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *CoRR*, vol. abs/1602.07360, 2016.

[130] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[131] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet V2: Practical guidelines for efficient CNN architecture design," in *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018, pp. 122–138.

[132] R. J. Wang, X. Li, S. Ao, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," *CoRR*, vol. abs/1804.06882, 2018.

[133] Y. Fu, Y. Lei, T. Wang, W. J. Curran, T. Liu, and X. Yang, "A review of deep learning based methods for medical image multi-organ segmentation," *Physica Medica*, vol. 85, pp. 107–122, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1120179721001848

[134] Y. Yeboah, C. Yanguang, W. Wu, and Z. Farisi, "Semantic scene segmentation for indoor robot navigation via deep learning," in *Proceedings of the 3rd International Conference on Robotics, Control and Automation*, ser. ICRCA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 112–118. [Online]. Available: https://doi.org/10.1145/3265639.3265671

[135] Z.-W. Hong, Y.-M. Chen, H.-K. Yang, S.-Y. Su, T.-Y. Shann, Y.-H. Chang, B. H.-L. Ho, C.-C. Tu, T.-C. Hsiao, H.-W. Hsiao, S.-P. Lai, Y.-C. Chang, and C.-Y. Lee, "Virtual-to-real: Learning to control in visual semantic segmentation," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 4912–4920.

[136] X. Zhao, T. Zuo, and X. Hu, "Ofm-slam: A visual semantic slam for dynamic indoor environments," *Mathematical Problems in*

Engineering, vol. 2021, p. 5538840, Apr 2021. [Online]. Available: https://doi.org/10.1155/2021/5538840

[137] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[138] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Cham: Springer International Publishing, 2015, pp. 234–241.

[139] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[140] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[141] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," *CoRR*, vol. abs/1606.02147, 2016.

[142] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for real-time semantic segmentation on high-resolution images," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 405–420.

[143] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 325–341.

[144] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[145] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast semantic segmentation network," *CoRR*, vol. abs/1902.04502, 2019.

[146] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[147] "Motion-based Segmentation and Recognition Dataset," http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/, available online; Accessed: 2021-06-10.

[148] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Computer Vision – ECCV 2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 44–57.

[149] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, Jan 2009.

[150] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018, pp. 833–851.

[151] A. Leykin and R. Hammoud, "Pedestrian tracking by fusion of thermal-visible surveillance videos," *Mach. Vis. Appl.*, vol. 21, pp. 587–595, 06 2010.

[152] P.-J. Huang and D.-Y. Chen, "Robust wheelchair pedestrian detection using sparse representation," in *2012 Visual Communications and Image Processing*, 2012, pp. 1–5.

[153] Z. Chen and X. Huang, "Pedestrian detection for autonomous vehicle using multi-spectral cameras," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 211–219, 2019.

[154] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[155] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545, 2014.

[156] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

[157] J. Carreira and C. Sminchisescu, "Constrained parametric min-cuts for automatic object segmentation," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3241–3248.

[158] P. Krähenbühl and V. Koltun, "Geodesic object proposals," in *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014, pp. 725–739.

[159] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders, "Segmentation as selective search for object recognition," in *2011 International Conference on Computer Vision*, 2011, pp. 1879–1886.

[160] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[161] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[162] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016.

[163] ——, "Yolov3: An incremental improvement," 2018.

[164] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[165] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 06 2010.

[166] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar, "Microsoft COCO: Common objects in context," *arXiv:1405.0312v3 [cs.CV]*, pp. 1–15, 2015.

[167] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," 2020.