Universita' degli studi di Napoli Federico II

Corso di Visione Computazionale

Prof. Francesco Isgrò

Development of a Denoising AutoEncoder, based on a FFML neural network, for the reconstruction of noisy images

---

Ciro Brandi, crbn@hotmail.it

Marco Urbano, urbamarc@gmail.com

Luigi Urbano, luigiurbano@outlook.com

A.A. 2019/2020

Indice

Introduzione **2**

1 Il caso di studio **2**
1.1 Dataset MNIST . . . 2

2 Autoencoder e Deinoser **3**
2.1 Basic autoencoder . . . 3
2.2 Denoising autoencoder . . . 4

3 Cenni teorici **5**
3.1 Rete neurale multistrato fully-connected . . . 5
3.2 Feed-forward . . . 5
3.3 Backpropagation . . . 6
3.4 Criterio di fermata: Generalization Loss . . . 7
3.5 Funzioni di errore: somma dei quadrati . . . 8
3.6 Segnali di Rumore . . . 8
3.6.1 Rumore Gaussiano . . . 8
3.6.2 Rumore sale e pepe . . . 9
3.6.3 Rumore Standard . . . 10

4 Implementazione del sistema **11**
4.1 Le classi che descrivono la rete . . . 11
4.2 Net . . . 12
4.3 Layer . . . 14
4.4 Utils . . . 14

5 Sperimentazione **15**
5.1 Iper-parametri utilizzati . . . 15
5.1.1 Basic Autoencoder . . . 15
5.1.2 Denoising Autoencoder . . . 15
5.1.3 Dimensione Dataset e Learning rate . . . 15
5.2 Denoiser con tecnica dei pesi legati . . . 16

6 Risultati **17**
6.1 Risultati senza rumore . . . 17
6.2 Rumore Standard . . . 17
6.2.1 Risultati su un training set di 1000 immagini . . . 17
6.2.2 Risultati su un training set di 10000 immagini . . . 19
6.3 Rumore Salt-and-pepper . . . 21
6.3.1 Risultati su un training set di 1000 immagini . . . 21
6.3.2 Risultati su un training set di 10000 immagini . . . 23
6.4 Caratteristiche dell'Hardware utilizzato . . . 25

7 Conclusioni **26**

8 Bibliografia **26**
8.1 Riferimenti . . . 26
8.2 Fonti foto . . . 26

Introduzione

L'intero lavoro è basato sull'articolo *"**Extracting and Composing Robust Features with Denoising Autoencoders**"*, il quale propone la realizzazione di una rete neurale con capacità di ricostruzione di immagini distorte, seguendo diverse specifiche.
L'obiettivo è stato la realizzazione della struttura descritta nell'articolo, allo scopo di effettuare denoising di un dataset di immagini, a cui è stato opportunamente applicato del rumore (di diverso tipo).
È stato utilizzato il dataset MNIST, così da disporre di un numero considerevole di immagini ridondanti, al fine di poter effettuare una buona fase di addestramento e, quindi, ottenere risultati migliori in fase di ricostruzione.
Seguendo l'articolo sono state implementate due reti neurali di tipo feed-forward fully connected le quali sono rispettivamente l'*autoencoder* ed il *denoiser*.
In questo documento sono descritte le specifiche tecniche utilizzate per l'implementazione delle reti, su cui sono stati eseguiti numerosi test al fine di riportare una corretta sperimentazione.

1 Il caso di studio

1.1 Dataset MNIST

Figura 1: Dataset MNIST

Per il progetto è stato utilizzato il dataset *MNIST*, che consiste in un ampio database di cifre scritte a mano, utile per la formazione di sistemi di elaborazione delle immagini.
Il database MNIST contiene 60000 immagini di addestramento e 10000 immagini di test.
L'insieme di immagini del database è una combinazione di due dei database NIST ed equivalgono a cifre scritte da studenti di scuole superiori e dipendenti dell'*United States Census Bureau*.
I numeri rappresentati sono nell'intervallo tra 0 e 9.

2 Autoencoder e Deinoser

Un autoencoder è un tipo di rete neurale artificiale utilizzata per apprendere codifiche di dati efficienti in modo non supervisionato. Lo scopo di un autoencoder è apprendere una rappresentazione (codifica) di un insieme di dati, al fine di ridurre il numero di variabili casuali introdotte durante al costruzione della rete neurale (*dimensionality reduction*) addestrandola poi a non considerare il "rumore" applicato al segnale. L'autoencoder tenta di generare dalla codifica ridotta una rappresentazione il più vicino possibile al suo input originale.

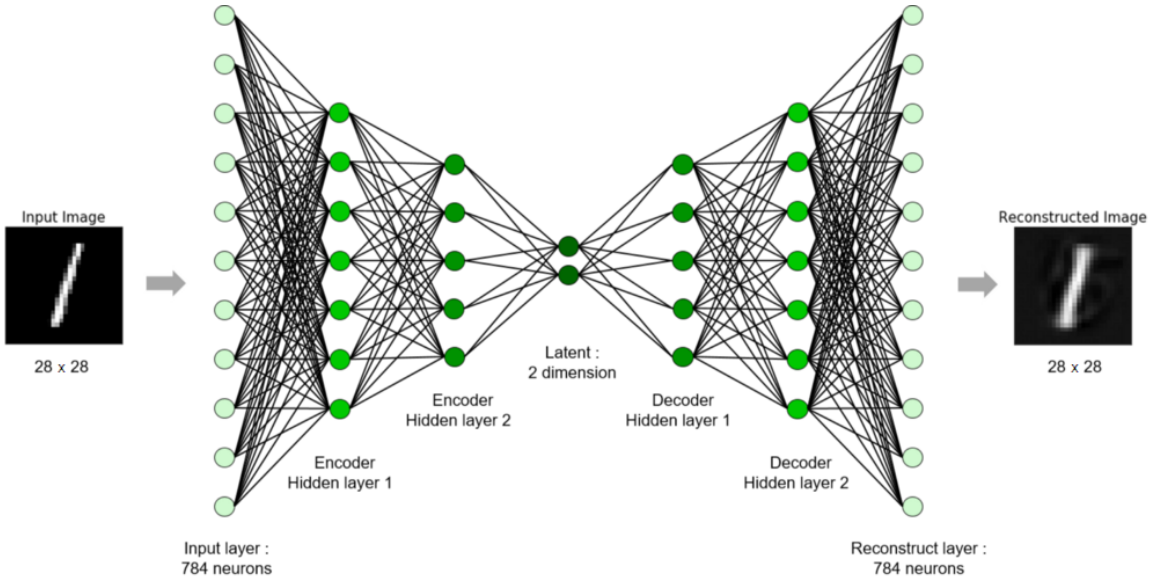


Figura 2: Esempio di Auto-encoder

2.1 Basic autoencoder

il basic autoencoder è una rete neurale feed-forward full connected costruita nel seguente modo:

- Strato di input
- Strato di nodi Hidden
- Strato di nodi Output

L'addestramento è avvenuto con l'inizializzazione casuale dei pesi. Al termine della fase di training l'output dell'basic autoencoder è stato utilizzato come input del denoiser, questa operazione ha dimostrato migliorie significative sulla generazione dell'errore rispetto all'inizializzazione casuale dei pesi del secondo autoencoder, ottenendo cosi una migliore generalizzazione nella fase di ricostruzione.

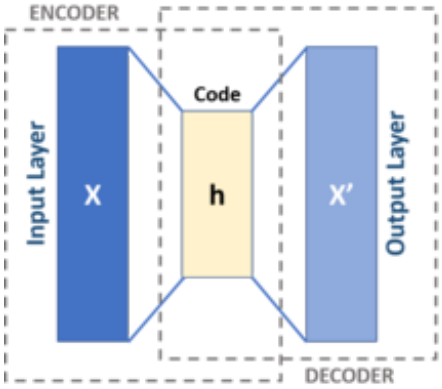


Figura 3: Esempio di Basic auto-encoder

Nel nostro caso l'autoencoder prende in input un vettore $x \in [0,1]^d$, a cui viene applicato un mapping deterministico producendo un vettore $y = f_\theta(x) = s(Wx + b); y \in [0,1]^2$ parametrizzato da $\theta = (W, b)$. W è una matrice di pesi $d^i \times d$ e b è un vettore di bias. La rappresentazione risultante y viene quindi mappata su un vettore "ricostruito" $z = g_{\theta'}(y) = s(W^{'}y + b^i)$ con $\theta^{'} = (W^{'}, b^{'})$ La matrice dei pesi $W^{'}$ del mapping di ricostruzione può eventualmente essere vincolata attraverso la tecnica dei pesi legati $W^{'} = W^T$.Ogni dato di input x^i è mappato su una corrispondente risultato intermedio y^i e una ricostruzione z^i. L'autoencoder è ottimizzato in modo tale da ridurre al minimo l'errore di ricostruzione:
$\theta^*, \theta^{'*} = \underset{\theta,\theta'}{\text{argmin}} = \frac{1}{n}\sum_{i=1}^{n} L(x^i, z^i)$, dove L è una qualsiasi funzione di errore. Nel nostro caso la funzione di errore utilizzata è la somma dei quadrati: $L(x, y) = |x - z|^2$. Al termine dell'addestramento il basic autoencoder avrà minimazzato la funzione di errore, modificando i pesi in maniera opportuna. L'output al livello k-esimo verra utilizzato come input per il livello (k+1)-esimo del denoiser.

2.2 Denoising autoencoder

Il precedente autoencoder, definito *"based"*, è stato opportunamente modificato, seguendo le direttive dell'articolo.
Da esso otteniamo un autoencoder, che chiameremo *denoising autoencoder*, il cui funzionamento è descritto di seguito.

L'addestramento dell'autoencoder ha il fine di renderlo capace di eliminare il rumore da immagini ricevute in input.
Il processo di ricostruzione di una immagine deriva da diverse fasi necessarie per rendere performante l'autoencoder.
Vi è una prima fase in cui è applicato rumore per ogni input x; il rumore applicato è di diversi tipi, descritti nel paragrafo dedicato.
L'input corrotto $\bar{x}$ è mappato, come per il *basic autoencoder*, da una funzione $y = f_\theta(\bar{x}) = s(W\bar{x} + b)$, da cui si ottiene $z = g_{\theta'}(y) = s(W^{'}y + b^{'})$. La funzione di errore utilizzata, anche in questo caso, è la somma dei quadrati.
Importante sottolineare che in questo caso **z** è una funzione deterministica di $\bar{x}$ e non di x, quindi il risultato della mappatura stocastica di x.

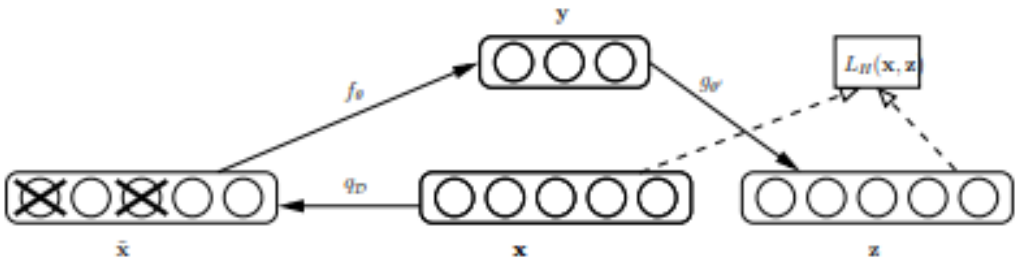


Figura 4: Esempio di denoising auto-encoder

3 Cenni teorici

3.1 Rete neurale multistrato fully-connected

Una *rete neurale multistrato fully-connected* è una rete neurale artificiale costituita da più strati computazionali **interconnessi** in modalità **fully-connected**. Questo significa che *ogni neurone* è direttamente connesso ai neuroni dello strato successivo e che non sono presenti cicli tra le varie connessioni.

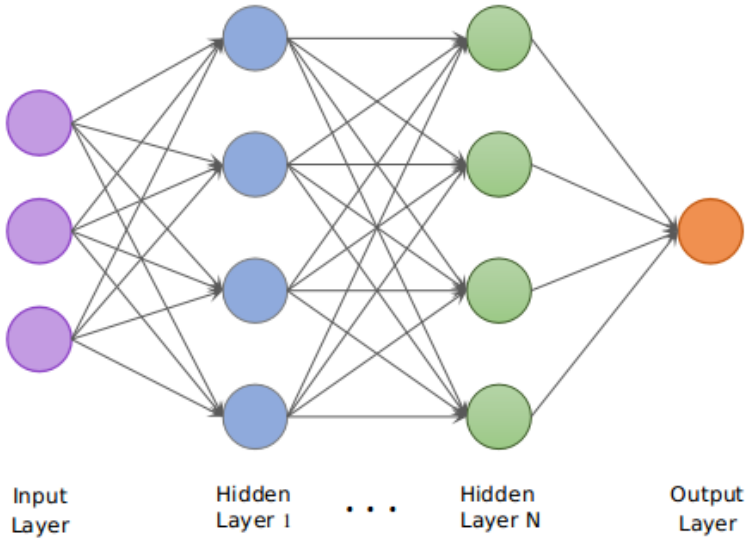


Figura 5: Rete neurale multistrato fully-connected

In sintesi, una rete neurale è fully-connected se è possibile associare numeri sequenziali crescenti ad ogni neurone, input compresi, tale che ciascun neurone riceva connessioni da tutti e soli i neuroni o gli input con numero associato più piccolo e non vi sono cicli.
La sequenza di attivazione è asincrona e segue *l'ordinamento topologico* stabilito dalle connessioni.

3.2 Feed-forward

I neuroni della rete, inclusi quelli di output, hanno associata una **funzione di attivazione**, solitamente la sigmoide o l'identità, che permette di calcolarne l'attivazione attraverso la funzione:

$$z_h = f_h * \sum_k (w_{hk} * z_k) + b_h$$

Richiamando la funzione su tutti i neuroni della è possibile effettuare la ***propagazione in avanti***.

3.3 Backpropagation

La ***backpropagation*** è un algoritmo per il calcolo delle derivate intermedie, ovvero della funzione di errore rispetto ai valodi di attivazione, necessarie al colcolo delle derivate rispetto ai pesi ed ai bias.
La backpropagation, anche chiamata *error backpropagation*, permette alla rete di modificare i pesi delle connessioni confrontando, in fase di training, il risultato ottenuto con il risultato atteso.
La backpropagation non serve per effettuare training, e va quindi utilizzata in sincronia con algoritmi di training come ad esempio la *discesa del gradiente* in cui i pesi vengono aggiornati, arrivando alla seguente formula di aggiornamento dei pesi:

$$w_{ij} = w_{ij} - \eta * \frac{\delta E}{\delta w_{ij}}$$

In generale, l'algoritmo utilizza la somma pesata di ciascun nodo, equivalente a:

$$a_i = \sum_j w_{ij} * z_j$$

con z_j output del neurone che invia una connessione al neurone i, calcolato con:

$$z_i = g(a_j)$$

con g uguale alla funzione di output scelta.

Le derivate della funzione di errore rispetto ai valori di attivazione, necessarie per calcolare le derivate della funzioni di errore rispetto ai pesi, prendono il nome di δ e sono definiti ricorsivamente come:

- $\delta_i^L = g'_L(a_i^L) * E'(z_i^L, t_i)$, con L strato di output della rete.
- $\delta_i^l = g'_l(a_i^l) * (\delta_{l+1} * W_{il+1})$, con l strato hidden.

Evidenziando che la backpropagation è un algoritmo per il solo calcolo delle derivate, l'aggiornamento dei pesi può essere effettuato in due modalità:

1. ***online learning***: i pesi sono aggiornati subito dopo aver presentato un pattern del training set alla rete e ne sono state calcolate le derivate.
2. ***batch learning***: dopo che l'intero training set è stato presentato alla rete e si ha un accumulamento dei pesi dato da: $\Delta w_{ji} = -\eta * \sum_n \delta_j^n * z_i^n$.

3.4 Criterio di fermata: Generalization Loss

L'idea che sta alla base del criterio detto *generalization loss* è quella di interrompere il processo di apprendimento non appena si raggiunge un determinato livello di errore sul *validation set*. Dunque a differenza del criterio di stopping più semplice, ovvero quello in cui ci si ferma non appena l'errore sul *validation set* sale invece di continuare nella sua discesa, *generalization loss* continua l'apprendimento fino ad un determinato valore della funzione che è specificato dal parametro *alpha*. E' facile allora immaginare che, visto che in fase di apprendimento l'errore sul validation set può ancora scendere anche dopo essere aumentato, questo criterio di stopping cerca di arginare quanto più possibile il limite imposto dal più semplice criterio di stop prima illustrato.

Possiamo descrivere formalmente questo criterio come segue:

" Sia E la funzione di errore scelta per l'algoritmo di training, ad esempio la Cross Entropy come nel nostro caso. Quindi si definisca con $\boldsymbol{E_{val}(t)}$ *l'errore sul validation set corrispondente all'epoca* $\boldsymbol{t}$ *e si ricordi che dato che nel caso reale l'errore di generalizzazione è sconosciuto, l'unico modo per effettuare una stima di tale errore è quello di basarsi proprio sull'errore sul validation set. Inoltre si indichi con* $\boldsymbol{E_{opt}(t)}$ *come il più basso errore ottenuto sul validation set sino all'epoca* $\boldsymbol{t}$ *e lo si definisca come:*

$$\boldsymbol{E_{opt}(t) := min_{t' \leq t}\ E_{va}(t')}$$

Date queste definizioni preliminari è possibile definire ***generalization loss*** *all'epoca t come l'incremento relativo dell'errore di validazione sul minimo fin ora raggiunto, in formule:*

$$\boldsymbol{GL(t) = 100 * (\frac{E_{\mathrm{va}}}{E_{\mathrm{opt}}} - 1)}$$

In generale un alto tasso di ***generalization loss*** *è una ovvia ragione di fermare il processo di training poichè ciò è un indicatore esplicito di* ***overfitting*** *e dunque di perdita di generalizzazione. Questa definizione dunque ci porta all'idea di questo criterio:*

Il processo di training viene interrotto non appena il valore di GL eccede una certa soglia α

Dunque si definisce GL_α *come:*

$$\boldsymbol{GL_\alpha}\textbf{: ci si ferma dopo la prima epoca t con } \boldsymbol{GL(t) > \alpha}$$

Questo criterio, come è facilmente osservabile, tiene conto solamente dell'andamento del validation set e non del training set: il prossimo criterio sarà incentrato sulle informazioni che provengono sia dall'andamento dell'errore sul validation set che sul training set."

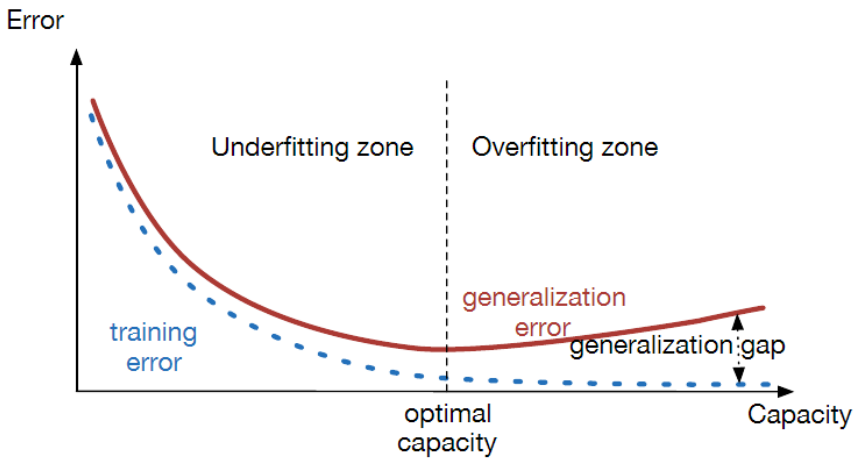


Figura 6: Andamento errori con Generalitazion Loss [4]

3.5 Funzioni di errore: somma dei quadrati

La somma dei quadrati è definita come:

$$E = \sum_{n=1}^{N} E^n$$

$$E^n = \sum_{k=1}^{c} \frac{1}{2} \cdot (y_k^n - t_k^n)^2$$

La funzione è utilizzata per lo più per problemi di regressione, cioè in cui la rete deve approssimare una funzione matematica.
La derivata della funzione rispetto al k-esimo nodo di output y_k equivale a:

$$\frac{\partial E^n}{\partial y_k} = y_k - t_k$$

3.6 Segnali di Rumore

Successivamente descriveremo brevemente i segnali di rumore trattati in questa sperimentazione, con la relativa distribuzione di probabilità.

3.6.1 Rumore Gaussiano

Il rumore gaussiano, è un rumore statistico avente una funzione di densità di probabilità (PDF) pari a quella della distribuzione normale, nota anche come distribuzione gaussiana. La funzione di densità di probabilità p di una variabile casuale gaussiana z è dato da:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

dove z rappresenta il livello di grigio, μ il valore grigio medio e σ la sua deviazione standard.

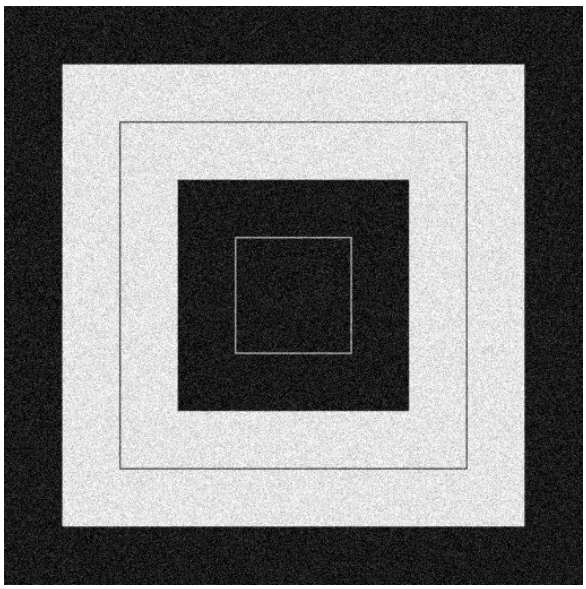


Figura 7: Immagine con rumore Gaussiano

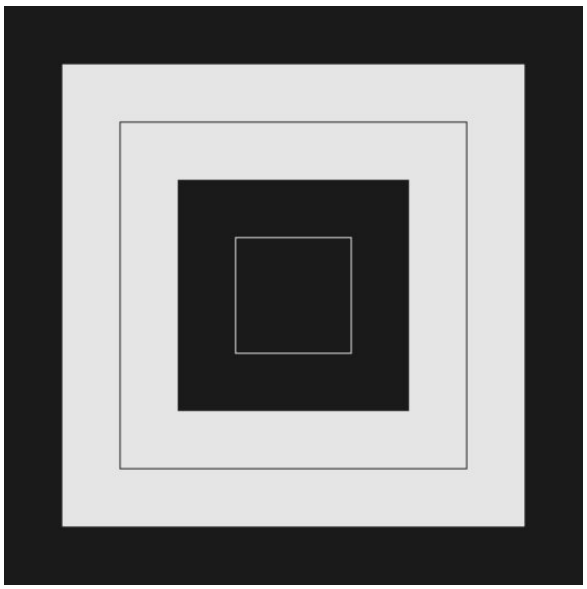


Figura 8: Immagine senza rumore Gaussiano

3.6.2 Rumore sale e pepe

La funzione di probabilità di densità del rumore ad impulsi (bipolare) è data da:

$$\mathrm{p(z)} = \begin{cases} P_a & \text{per z=a} \\ P_b & \text{per z=b} \\ 0 & \text{altrimenti} \end{cases}$$

Se $b > a$, l'intensità b apparirà come un punto chiaro nell'immagine, altrimenti a apparirà come un punto scuro. Se nessuno dei due valori di probabilità è nullo e sono approssimativamente uguali, i valori del rumore ad impulsi somiglieranno a granuli di sale e di pepe sparsi sull'immagine, da qui il nome con cui, comunemente, ci si riferisce a questo tipo di errore.

Figura 9: Immagine con e senza rumore sale e pepe

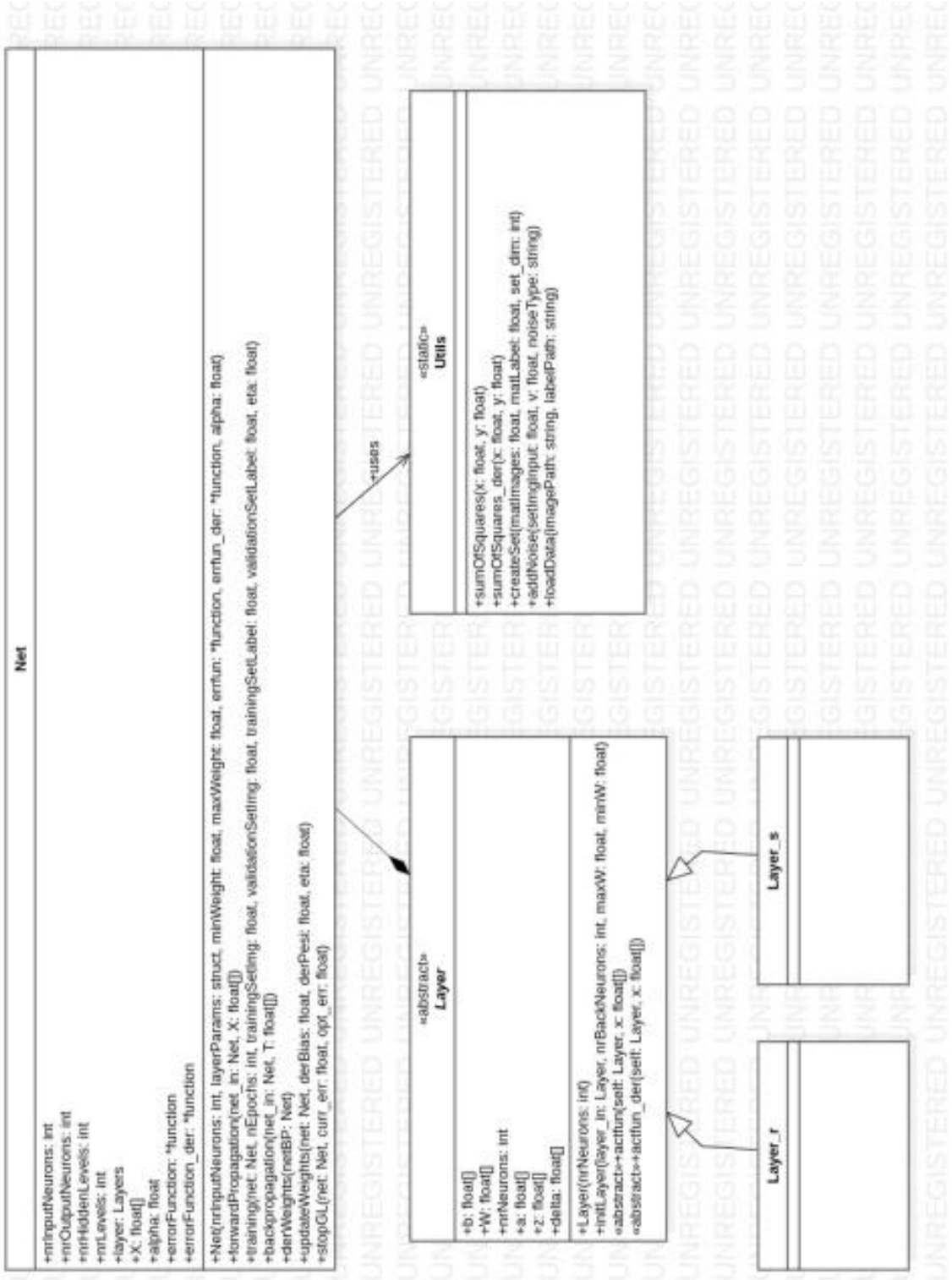
3.6.3 Rumore Standard

Per rumore *standard* ci si riferisce ad un rumore parametrizzato da una proporzione di distruzione desiderata sull'input, in dettaglio:
per ogni input x, un numero fisso di d componenti viene scelto a caso e il loro valore assume un valore 0, mentre le altre componenti rimangono inalterate. l'autoencoder verrà addestrato per cercare di "riempire" gli spazi vuoti lasciati dal processo di distruzione.

4 Implementazione del sistema

Il sistema è stato implementato mediante linguaggio *Matlab*, preferito ad altri data la rapidità con cui effettua le operazioni matematiche richieste dalla discesa del gradiente impiegata nell'autoencoder e nel denoiser essendo entrambi reti neurali, con l'ausilio delle librerie *"Toolbox Image Processing"*, impiegate per la generazione di dataset soggetti a diverso tipo di rumore. Questo capitolo vuole essere una rapida introduzione alle classi che compongono il sistema, mostrate nel successivo class diagram e descritte testualmente nei sottoparagrafi che seguono.

4.1 Le classi che descrivono la rete



4.2 Net

La classe Net rappresenta una generica rete neurale che può essere dotata di diverse funzioni di errore, a seconda che la si voglia usare per effettuare classificazione o, come nel nostro caso, regressione, che possono essere fornite mediante il costruttore di seguito mostrato.

```
function net = Net(nrInputNeurons, layerParams, minWeight,
    maxWeight, errfun, errfun_der, alpha)
```

Ogni rete neurale è composta da un livello di livelli (o layers) che si dividono in livelli di *input, hidden* e di *output* che sono forniti al costruttore attraverso il parametro *layerParams*, il quale altro non è che un array di strutture che contengono rispettivamente:

- n_nodes: numero di neuroni dello strato i-esimo
- act_fun: funzione di attivazione dello strato i-esimo

Di seguito è possibile trovare l'esempio di istanziazione di queste strutture sia per la prima rete, l'autoencoder, che per la seconda, il denoiser.

```
% Defining autoencoder's layers details
layers_Autoencoder(1) = struct('n_nodes', hiddenNodes_AutoEncoder,
    'act_function', "relu");
layers_Autoencoder(2) = struct('n_nodes', n_features_AutoEncoder,
    'act_function', "sigmoid");

% Defining Denoiser's layers details
layers_Denoiser(1) = struct('n_nodes', hiddenNodes_AutoEncoder,
    'act_function', "relu");
layers_Denoiser(2) = struct('n_nodes', hiddenNodes_Denoiser,
    'act_function', "relu");
layers_Denoiser(3) = struct('n_nodes', hiddenNodes_AutoEncoder,
    'act_function', "relu");
layers_Denoiser(4) = struct('n_nodes', n_features_Denoiser,
    'act_function', "sigmoid");
```

Il criterio di stop utilizzato in questa specifica implementazione è il cosiddetto "Generalization Loss", di cui si è discusso nei precedenti capitoli. L'iperparametro *alpha*, utilizzato per la fermata da GL, è fornito anch'esso al momento dell'istanziazione.
Il metodo deputato ad effettuare il processo di apprendimento è *"training"*.

```
function [bestnet, errorTS_arr, errorVS_arr, totEpochs] =
    training(net, nEpochs, trainingSetImg, validationSetImg,
    trainingSetLabel, validationSetLabel, eta)
```

I valori restituiti sono i seguenti:

- best_net: rete con il livello di apprendimento migliore.
- errorTS_arr: vettore contenente il livello di errore durante la fase di training per ogni epoca.
- errorVS_arr: vettore contenente il livello di errore durante la fase di validation per ogni epoca.

- totEpochs: valore che indica il totale di epoche raggiunte durante la fase di training da stampare a video.

I parametri sono invece:

- net: rete con il livello di apprendimento migliore.
- nEpochs: limite di epoche in cui effettuare il training.
- trainingSetImg: matrice contenente le immagini del training set soggette a rumore rappresentate come vettori di float di dimensione 784 (dato che le immagini sono 28x28).
- validationSetImg: matrice contenente le immagini del validation set soggette a rumore rappresentate come vettori di float di dimensione 784 (dato che le immagini sono 28x28).
- trainingSetLabel: matrice contenente le immagini del training set senza rumore rappresentate come vettori di float di dimensione 784 (dato che le immagini sono 28x28).
- validationSetLabel: matrice contenente le immagini del validation set senza rumore rappresentate come vettori di float di dimensione 784 (dato che le immagini sono 28x28).

Questo metodo restituisce la rete con il miglior livello di apprendimento a partire dall'oggetto sul quale viene invocato, ovvero una rete con tutti gli strati aventi come valore dei pesi di ogni collegamento tra i neuroni un numero casuale. L'apprendimento viene effettuato tramite la tecnica della discesa del gradiente, la quale è implementata attraverso la seguente coppia di metodi, che sono richiamati all'interno di *training*.

```
1 function net = forwardPropagation(net_in, X)
2 function net = backPropagation(net_in, T)
```

4.3 Layer

La classe Layer rappresenta un generico livello di una rete neurale. Tale classe è stata pensata come una classe astratta proprio per dare la possibilità al programmatore di implementare il proprio Layer, dotato di una funzione di attivazione ad-hoc, e non dover scegliere tra un insieme di Layer ristretti. I soli metodi da implementare per definire una sottoclasse di Layer sono solo quelli inerenti la funzione di attivazione e la sua derivata.

```
1    methods (Abstract)
2        actfun(self, x)
3        actfun_der(self, x)
4    end
```

Sono state già implementate le sottoclassi *Layer_s* e *Layer_r* dotate rispettivamente di funzione di attivazione e derivata di Sigmoide e RELU (REctifier Linear Unit), utili per la realizzazione del denoising autoencoder teorizzato nel paper[2] di cui questo documento vuole proporre una implementazione correlata di testing.

4.4 Utils

Si è scelto infine di introdurre una particolare classe statica, denominata *Utils*, deputata al contenimento di tutte quelle funzioni, da adesso metodi della medesima classe, che sarebbero servite ad effettuare operazioni di contorno, la quale responsabilità non sarebbe stata assimilabile ne alla classe Net ne a Layer. I metodi di questa classe sono illustrati mediante la loro firma nel seguente frammento di codice.

```
1 function z = sumOfSquare(x,y)
2 function z = sumOfSquare_der(x, y)
3 function [image_set,label_set] = createSet(matImages, matLabel,
      set_dim)
4 function noisyImage = addNoise(inputSet, v, noiseType)
5 function [images, labels] = loadData(imagePath, labelPath)
```

Le prime due funzioni sono banalmente la funzione di errore "somma dei quadrati" e la sua derivata.
Il metodo *createSet* si occupa di restituire un dataset di immagini con le rispettive etichette: è importante prestare particolare attenzione a questo metodo, in quanto prende in input immagini rappresentate come matrici 28 pixel per 28 e le trasforma in array di 784 elementi.
Il metodo addNoise è deputato ad aggiungere rumore alle immagini fornite in input.

5 Sperimentazione

Il Denoiser è stato sottoposto a diverse sperimentazione utilizzando il dataset MNIST e le funzioni descritte nei paragrafi precedenti.
In particolare la configurazione utilizzata per le sperimentazioni è la seguente: Denoiser addestrato con *discesa del gradiente* e tecnica dei pesi legati su un partizionamento del dataset MNIST, utilizzando un approccio *batch*; Nella configurazione del denoiser sono state applicate diverse tipologie di rumore definite precedentemente:

- Rumore Standard
- Rumore Sale e pepe

Il rumore è stato applicato alla partizione del dataset utilizzato per addestrare la rete al fine di valutare il processo di ricostruzione dell'immagine originale.

5.1 Iper-parametri utilizzati

Di seguito gli iper-parametri utilizzati in tutte le sperimentazioni effettuate.

5.1.1 Basic Autoencoder

Iper-Parametro	***Valore***
Numero di features di input	784
Numero di strati hidden	1
Numero di neuroni degli strati hidden	128
Numero di output	784
Intervallo dei pesi di partenza generati randomicamente	[-0.09, 0.09]
Funzione di attivazione hidden layer	ReLu
Funzione di attivazione output layer	Sigmoide
Funzione di errore	Somma dei quadrati

5.1.2 Denoising Autoencoder

Iper-Parametro	***Valore***
Numero di features di input	784
Numero di strati hidden	3
Numero di neuroni degli strati hidden	128-32-128
Numero di output	784
Intervallo dei pesi di partenza generati randomicamente	[-0.09, 0.09]
Funzione di attivazione hidden layer	ReLu
Funzione di attivazione output layer	Sigmoide
Funzione di errore	Somma dei quadrati

5.1.3 Dimensione Dataset e Learning rate

Iper-Parametro	***Configurazione_1***	***Configurazione_2***
Training set	1000	10000
Validation set	330	3300
Test set	330	3300
Learning rate	0.00002	0.000002
Epoche	1000	1000
Percentuale di rumore	0/25/50/75	0/25/50/75

5.2 Denoiser con tecnica dei pesi legati

Al fine di mostrare la robustezza del denoiser, al termine della fase di addestramento, sono state scelte le seguenti tipologie di rumore e di intensità da applicare ad i dataset:

- Nessun rumore
- ***Standard*** con intensità del 25%/50%/75%
- ***Salt-and-pepper*** con intensità del 25%/50%/75%

Eccezion fatta per la configurazione "senza rumore", ogni configurazione è stata eseguita su dataset di 1000 e 10000 immagini, cambiando opportunamente gli iperparametri, come descritto nel paragrafo 5.1.3.

6 Risultati

Per ogni configurazione è presente un grafico che illustra l'andamento della funzione di errore, del training set rispetto al validation set, della rete in fase di apprendimento. Inoltre, vi è un sottoinsieme di immagini a cui è applicato il relativo rumore e la ricostruzione restituita dalla rete per ognuna di esse.

6.1 Risultati senza rumore

Figura 10: Ricostruzione di immagini con rumore al 0% (epoche autoencoder: 335; epoche denoiser: 356)

6.2 Rumore Standard

Di seguito sono mostrati i risultati del Denoiser con un training set di dimensioni 1000/10000, con una percentuale di input distrutto dello 25%,50%,75% e le relative funzioni di errore.

6.2.1 Risultati su un training set di 1000 immagini

In questa sperimentazione sono stati utilizzati i seguenti parametri per l'early stopping sia per l'autoencoder che per il denoiser:
$\alpha_{\text{autoencoder}} = 2.3$
$\alpha_{\text{denoiser}} = 2.1$.

Intensità 25%

Figura 11: Ricostruzione di immagini con rumore standard 25% (epoche autoencoder: 335; epoche denoiser: 594)

Intensità 50%

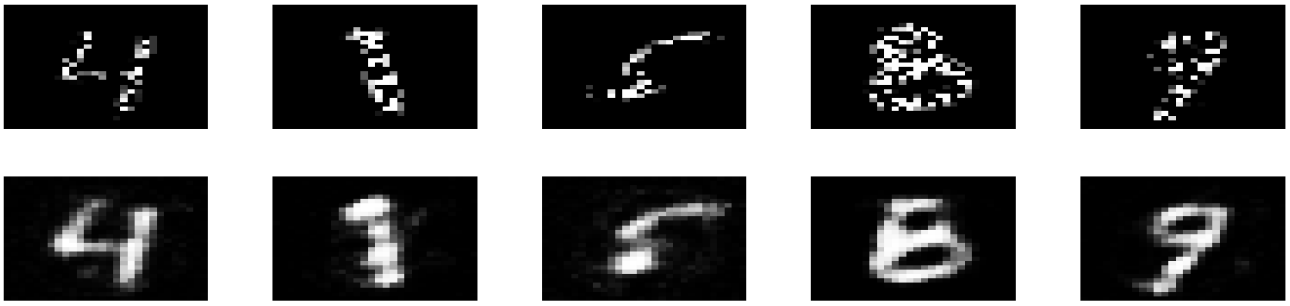


Figura 12: Ricostruzione di immagini con rumore standard 50% (epoche autoencoder: 786; epoche denoiser: 1000)

Intensità 75%

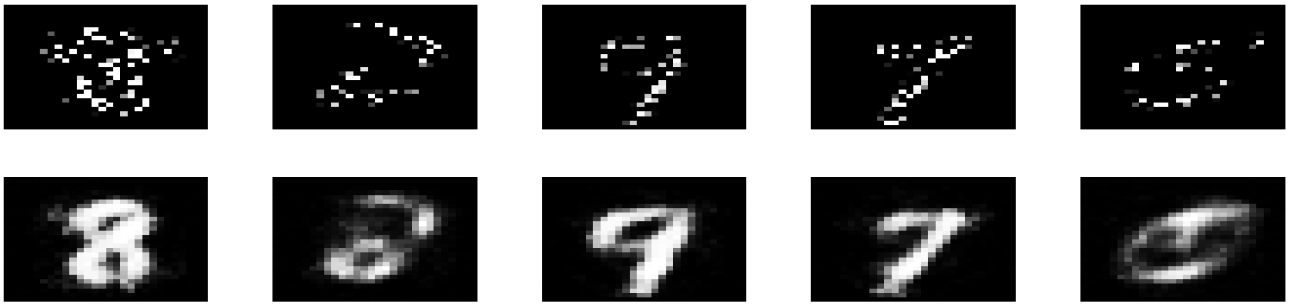
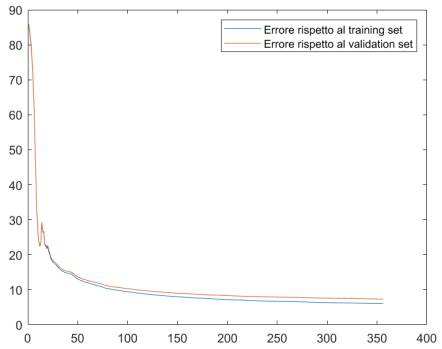
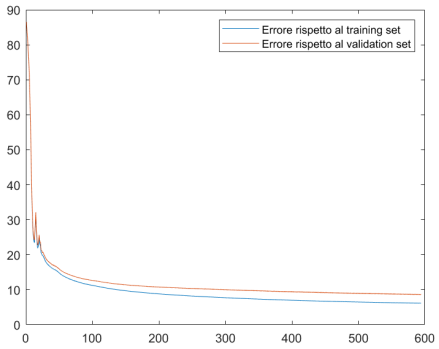


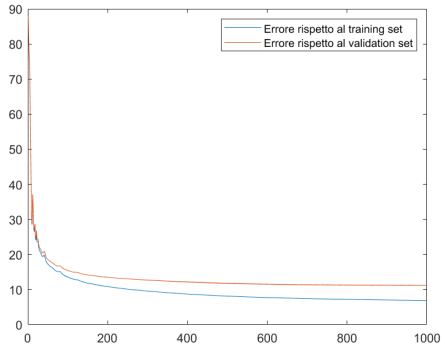
Figura 13: Ricostruzione di immagini con rumore standard 75% (epoche autoencoder: 729; epoche denoiser: 1000)



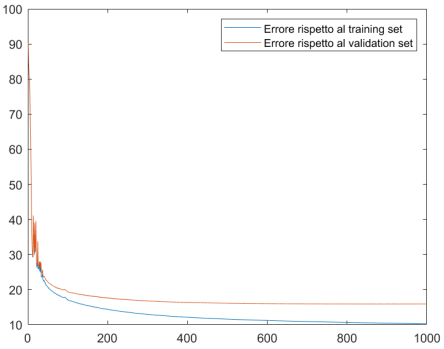
(a) Andamento dell'errore con 0% di distruzione



(b) Andamento dell'errore con 25% di distruzione



(c) Andamento dell'errore con 50% di distruzione



(d) Andamento dell'errore con 75% di distruzione

6.2.2 Risultati su un training set di 10000 immagini

In questa sperimentazione sono stati utilizzati i seguenti parametri per l'early stopping sia per l'autoencoder che per il denoiser:
$\alpha_{\text{autoencoder}} = 0.2$
$\alpha_{\text{denoiser}} = 0.2$.

Intensità 25%

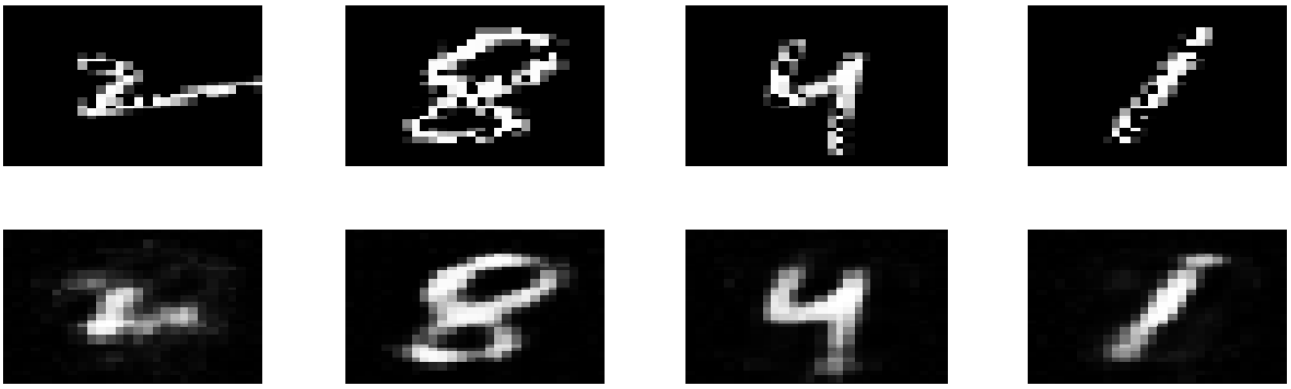


Figura 15: Ricostruzione di immagini con rumore standard 25%.(epoche autoencoder: 1000; epoche denoiser: 590)

Intensità 50%

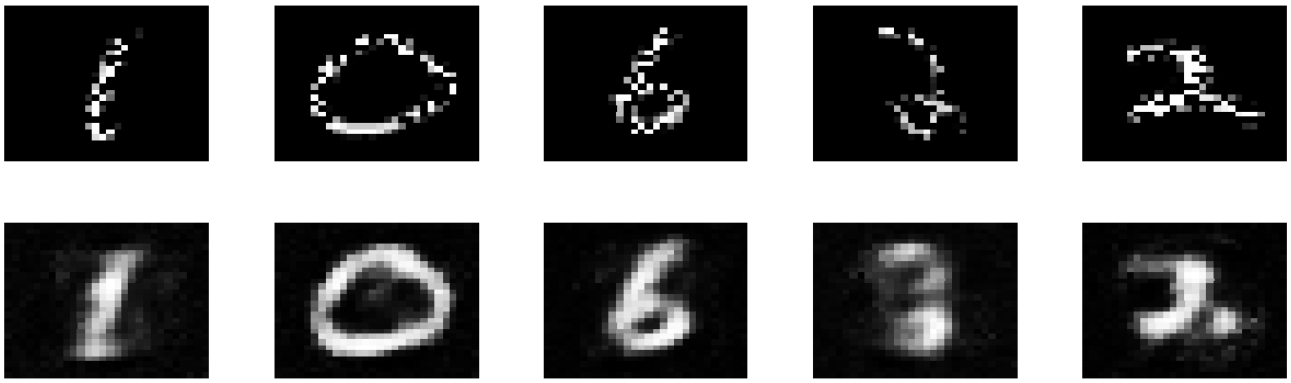


Figura 16: Ricostruzione di immagini con rumore standard 50%(epoche autoencoder: 1000; epoche denoiser: 334)

Intensità 75%

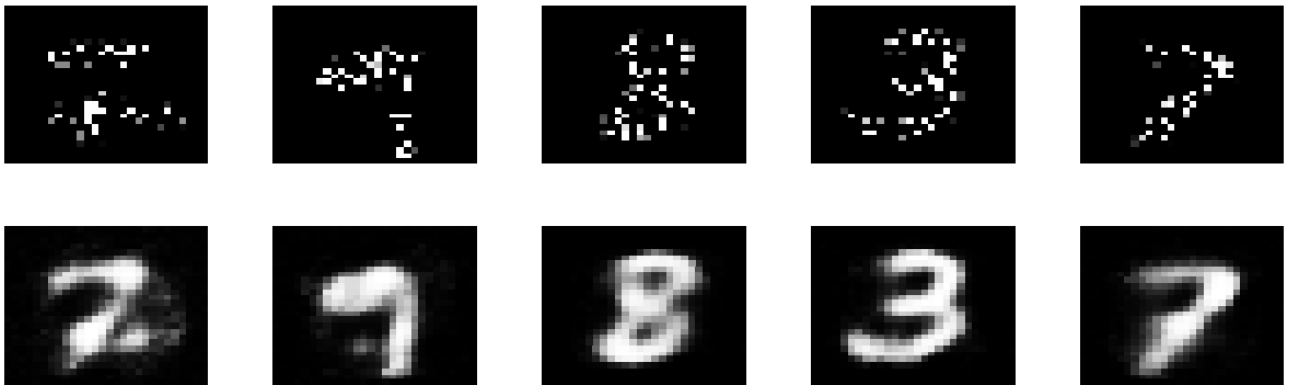
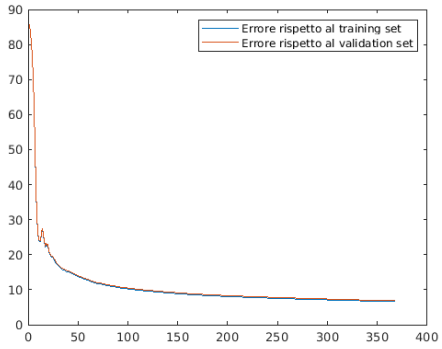
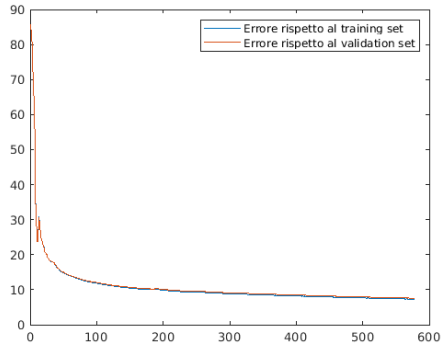


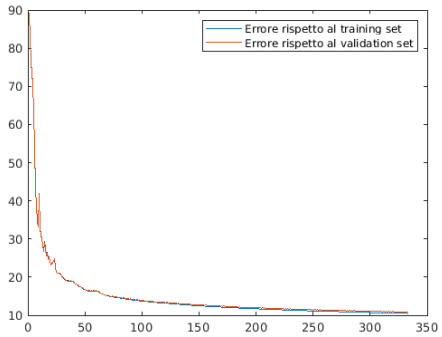
Figura 17: Ricostruzione di immagini con rumore standard 75% (epoche autoencoder: 1000; epoche denoiser: 1000)



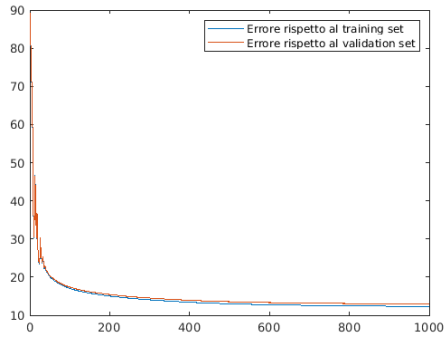
(a) Andamento dell'errore con 0% di distruzione



(b) Andamento dell'errore con 25% di distruzione



(c) Andamento dell'errore con 50% di distruzione



(d) Andamento dell'errore con 75% di distruzione

6.3 Rumore Salt-and-pepper

Di seguito sono mostrati i risultati del Denoiser con un training set di dimensioni 1000/10000, con una percentuale di input distrutto dello 25%,50%,75% e le relative funzioni di errore.

6.3.1 Risultati su un training set di 1000 immagini

In questa sperimentazione sono stati utilizzati i seguenti parametri per l'early stopping sia per l'autoencoder che per il denoiser:
$\alpha_{\text{autoencoder}} = 2.3$
$\alpha_{\text{denoiser}} = 2.1$.

Intensità 25%

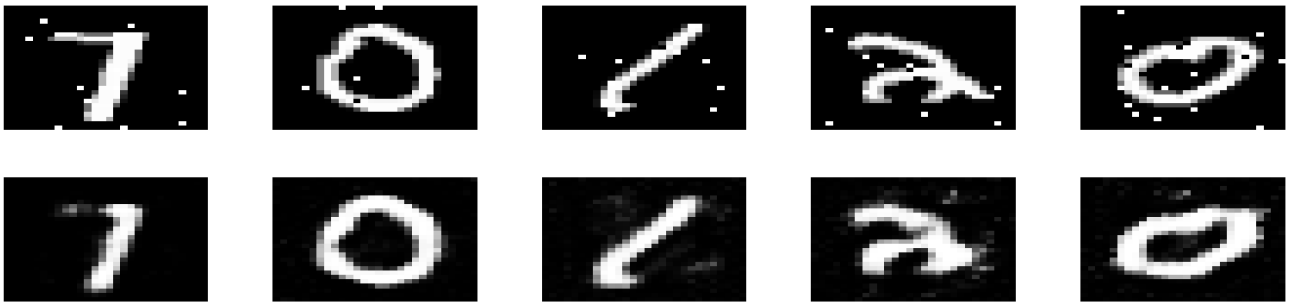


Figura 19: Ricostruzione di immagini con rumore salt-and-pepper 25% (epoche autoencoder: 1000; epoche denoiser: 1000)

Intensità 50%

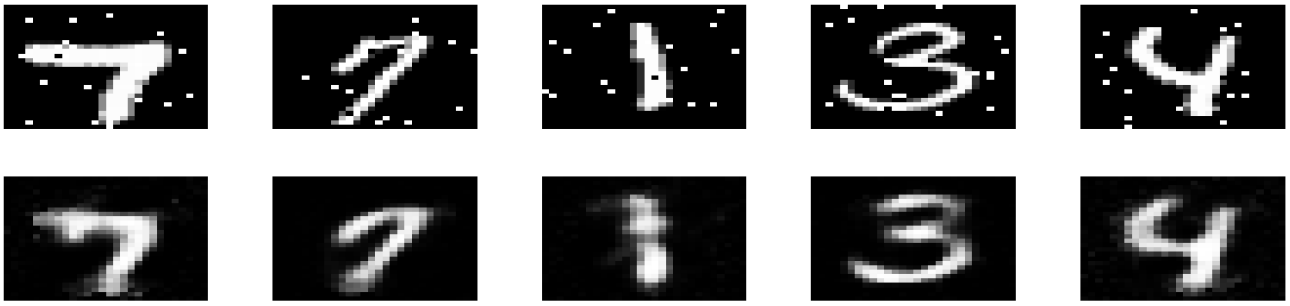


Figura 20: Ricostruzione di immagini con rumore salt-and-pepper 50%(epoche autoencoder: 334; epoche denoiser: 1000)

Intensità 75%

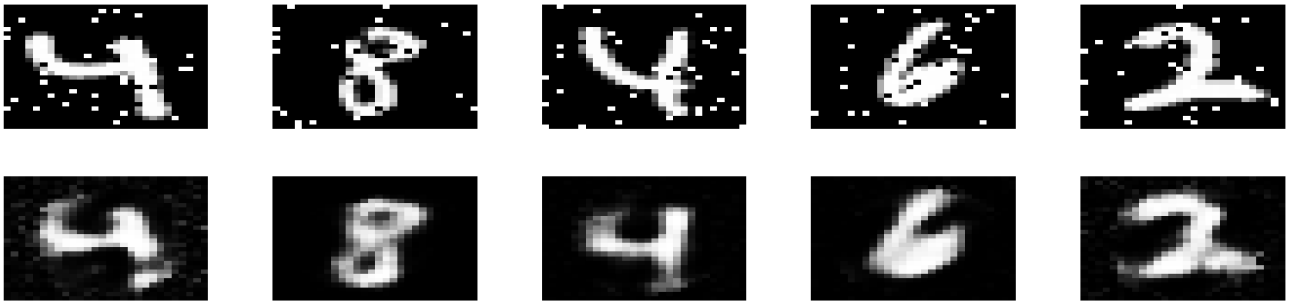
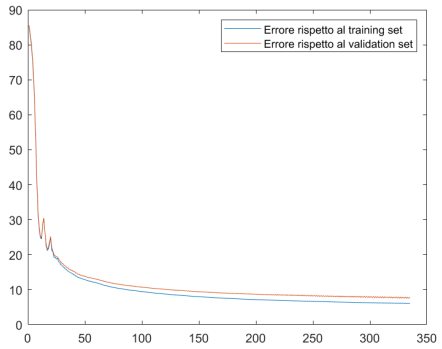
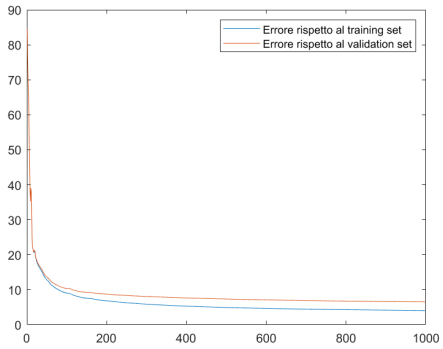


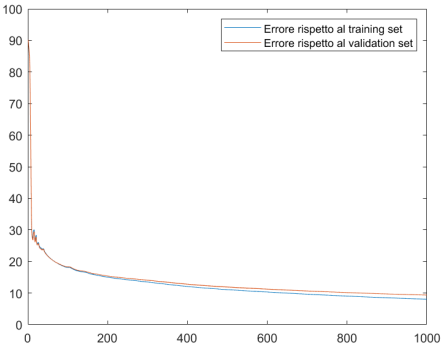
Figura 21: Ricostruzione di immagini con rumore salt-and-pepper 75%(epoche autoencoder: 339; epoche denoiser: 1000)



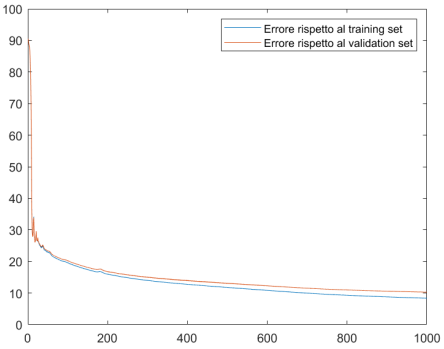
(a) Andamento dell'errore con 0% di distruzione



(b) Andamento dell'errore con 25% di distruzione



(c) Andamento dell'errore con 50% di distruzione



(d) Andamento dell'errore con 75% di distruzione

6.3.2 Risultati su un training set di 10000 immagini

Intensità 25%

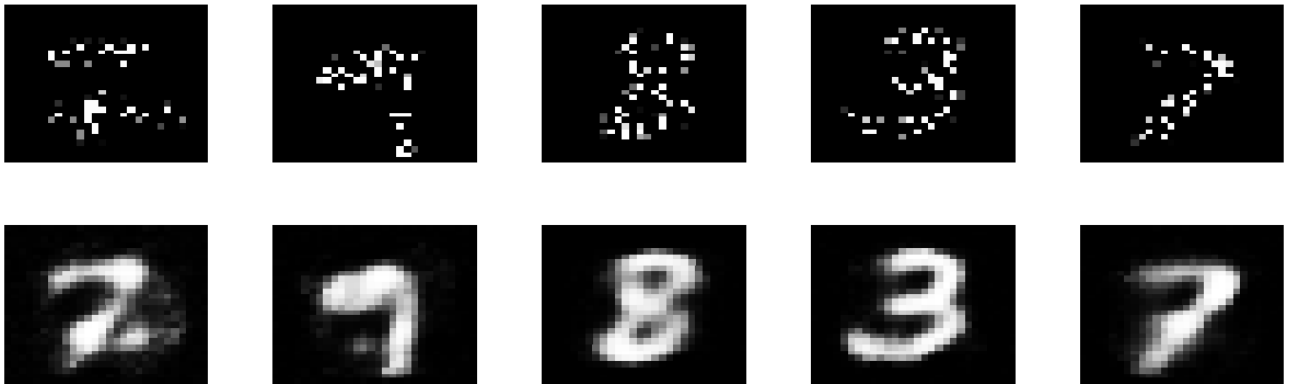


Figura 23: Ricostruzione di immagini con rumore salt-and-pepper 25% (epoche autoencoder: 334; epoche denoiser: 1000)

Intensità 50%

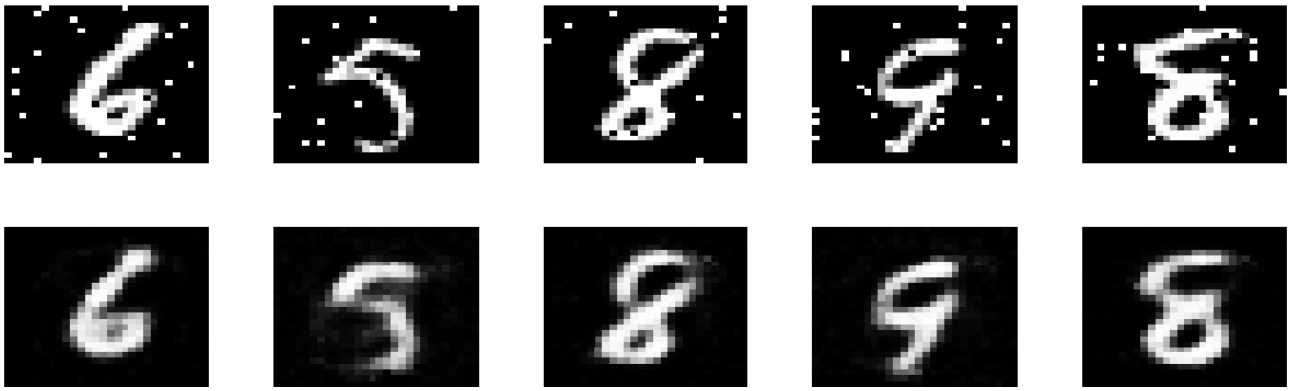


Figura 24: Ricostruzione di immagini con rumore salt-and-pepper 50%(epoche autoencoder: 334; epoche denoiser: 334)

Intensità 75%

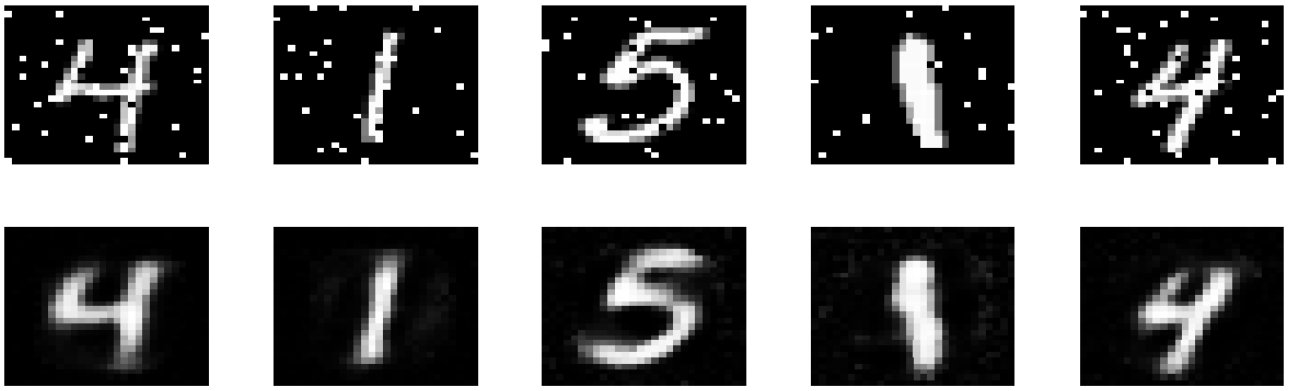
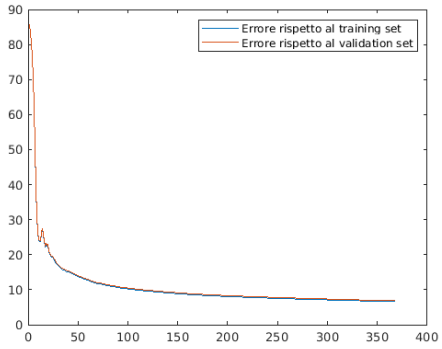
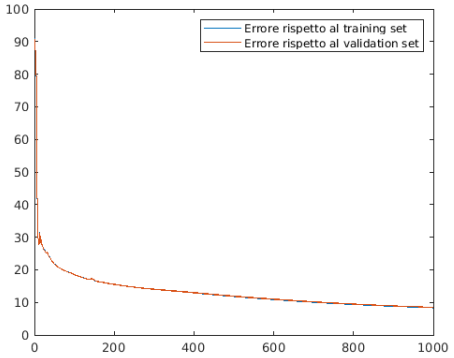


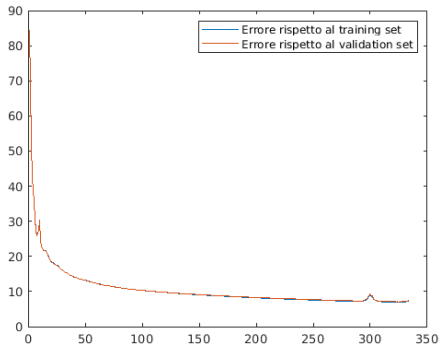
Figura 25: Ricostruzione di immagini con rumore salt-and-pepper 75%(epoche autoencoder: 334; epoche denoiser: 1000)



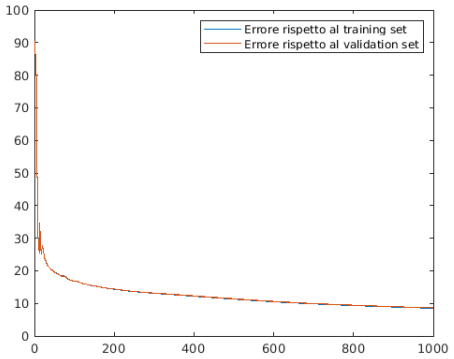
(a) Andamento dell'errore con 0% di distruzione



(b) Andamento dell'errore con 25% di distruzione



(c) Andamento dell'errore con 50% di distruzione



(d) Andamento dell'errore con 75% di distruzione

6.4 Caratteristiche dell'Hardware utilizzato

Di seguito le caratteristiche dell'hardware utilizzato per effettuare la sperimentazione.

Componente	***Caratteristiche***
Dell	G5 15
Processore	Intel(R) Core(TM) i7-9750H (12 MB Cache, fino a 4,5 GHz, hexa-core)
Sistema Operativo	Windows 10 Home
Memoria RAM	32 GB DDR4
Hard Disk	1TB + 128GB SSD

7 Conclusioni

Da quanto visto in questa serie di esperimenti possiamo affermare che un Denoising Autoencoder permette di ottenere buoni risultati con diversi tipi di rumore. Se paragoniamo questa tecnica di miglioramento delle immagini corrotte con altre tecniche, quali i vari filtri di riduzione del rumore, è possibile affermare che la qui presente tecnica è dotata di grande robustezza: è infatti dimostrato che si ottengono risultati discreti rispettivamente con rumore di tipo impulsivo (cosiddetto rumore sale e pepe) e con rumore di tipo additivo (il rumore gaussiano precedentemente illustrato). E' bene ricordare che i filtri di riduzione del rumore non sono in genere adatti per tutti i tipi di rumore: ad esempio è inutile, in pratica, applicare un filtro media su immagini soggette a rumore impulsivo, dato che comporterebbe una perdita di dettaglio considerevole sulle patch in cui sono presenti pixel distorti. Inoltre, come ulteriore esempio, se applicassimo un filtro di tipo mediana su immagini soggette a rumore di tipo gaussiano otterremmo scarsi risultati, in quanto si tratta di un filtro adatto a rimuovere i cosiddetti outliers. E' possibile concludere che quanto affermato finora rispetto alla capacità di un Denoising Autoencoder è strettamente legato ad una fase iniziale di addestramento, cosa richiesta ad esempio dai filtri precedentemente illustrati. Per ovviare ai tempi di addestramento è possibile impiegare le librerie che fanno uso della scheda grafica per effettuare le operazioni più onerose di addestramento, nello specifico i prodotti matriciali necessari per la discesa del gradiente nel nostro caso.

8 Bibliografia

8.1 Riferimenti

[1] Bishop Christopher M., Neural Networks for pattern recognition, Oxford University Press, Oxford University Press, 2005

[2] Extracting and Composing Robust Features with Denoising Autoencoders
`https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf`

[3] Steven H. Strogatz, Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering, 2000

[4] Early Stopping — but when?, Lutz Prechelt
`https://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf`

8.2 Fonti foto

[1] Libreria data MNIST
`https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png`

[2] Rete neurale
`https://cdn-images-1.medium.com/max/1600/1*Gh5PS4R_A5drl5ebd_gNrg@2x.png/`

[3] Generalization loss
`https://srdas.github.io/DLBook/DL_images/HP06.png`

[4] Autoencoder 1 Hidden Layer
`https://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Autoencoder_schema.png/220px-Autoencoder_schema.png`