



UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI STUDIO TRIENNALE IN
INFORMATICA

Deep Denoising: benchmarking di architetture profonde

Relatore:

Prof. Donato Impedovo

Tesi di Laurea di:

Alessandro Congedo

Matricola 677315

Correlatore:

Dott. Davide Veneto

Anno accademico:

2022/2023

A Francesco e Giuseppe

Abstract

L'obiettivo di questa tesi è stato quello di esplorare l'impiego di varie metodologie di deep learning per la rimozione del rumore nelle immagini di impronte digitali. In particolare, il rumore "salt and pepper" e il blur rappresentano sfide significative che possono degradare la qualità delle immagini e, di conseguenza, compromettere l'efficacia dei sistemi di identificazione biometrica. Attraverso l'analisi di diversi modelli di deep learning, questa ricerca ha mirato a evidenziare le principali differenze e le potenzialità di ciascun approccio nell'affrontare le suddette tipologie di rumore. I risultati sperimentali hanno confermato che le tecniche di deep learning adottate sono state capaci di eliminare efficacemente il rumore dalle immagini di impronte digitali, migliorandone la qualità.

Indice

1	Introduzione	1
2	Stato dell'Arte	3
3	Dataset	11
4	Metodi	13
4.1	Architetture	13
4.1.1	SeConvNet	14
4.1.2	Denoising Autoencoder	15
4.1.3	Residual Dense Net	17
4.1.4	Denoising U-net	19
4.2	Metriche	21
4.3	Optimizer	24
5	Sperimentazione	25
5.1	Tecnologie Utilizzate	25
5.2	Data Preparation	26
5.2.1	Rumore	26
5.2.2	Divisione del Dataset	30
5.2.3	Normalizzazione	31
5.2.4	Data Generator	32
5.3	Addestramento	35
5.3.1	Callbacks	37
6	Risultati	39
6.1	Salt And Pepper, 5%	40
6.1.1	Tensorboard	41
6.1.2	Predictions	42
6.2	Salt And Pepper, 50%	43
6.2.1	Tensorboard	44

6.2.2	Predictions	46
6.3	Blur	47
6.3.1	Tensorboard	47
6.3.2	Predictions	49
6.4	Mixed (Salt and Pepper - 20% + Blur)	50
6.4.1	Tensorboard	51
6.4.2	Predictions	53
7	Conclusione	55
Bibliografia		57

Capitolo 1

Introduzione

L'attenuazione del rumore nelle immagini, o "*denoising*", rappresenta una sfida significativa nell'elaborazione delle immagini digitali, con impatti diretti sull'accuratezza dell'identificazione biometrica, in particolare per quanto riguarda le impronte digitali. La capacità di ottenere immagini chiare e prive di interferenze è di fondamentale importanza in numerosi ambiti, dalla sicurezza informatica all'identificazione personale. Il processo di rimozione del rumore da immagini di impronte digitali è intrinsecamente complesso, a causa delle diverse nature del rumore e alla complessità delle immagini stesse.

In questo contesto, il *deep learning* emerge come uno strumento eccezionalmente potente. Questa tesi si concentra sull'esplorazione di tecniche basate su tale strumento specificatamente progettate per affrontare la sfida del denoising di immagini di impronte digitali.

Saranno analizzate e confrontate quattro diverse architetture:

- ***SeConvNet***: evoluzione selettiva della classica CNN;
- ***U-net***: efficace nell'elaborazione di immagini biometriche;
- ***Denoising AutoEncoder (DAE)***: specificatamente utilizzato per trattare rumore;
- ***Residual Dense Network (RDN)***: per l'efficienza osservata in merito all'utilizzo di connessioni dense e residuali.

L'obiettivo principale di questo studio è di confrontare le varie architetture e valutare l'efficacia e le prestazioni di esse nel rimuovere i rumori di tipo "*Salt and Pepper*" (**SAP**) e ***blur*** da immagini di impronte digitali.

Attraverso l'introduzione di queste reti, miriamo a esplorare e comprendere in modo approfondito le caratteristiche uniche e i compromessi tra questi approc-

ci, ponendo le basi per future ricerche e sviluppi nell'ambito dell'elaborazione di immagini di impronte digitali con tecniche di deep learning.

Per questi esperimenti, verranno utilizzati dataset specifici di impronte digitali, con l'aggiunta controllata di rumore SAP e di blur per simulare condizioni di rumore realistiche.

Per valutare l'efficacia dei modelli di rete neurale nel denoising, saranno impiegate le seguenti metriche:

- *Mean Squared Error*;
- *Peak-to-Signal Noise Ratio*;
- *Structural Similarity Index Measure*.

I risultati dimostreranno l'efficacia delle tecniche proposte, evidenziando una capacità notevole di produrre immagini di impronte digitali con rumore significativamente ridotto.

La conclusione della tesi rifletterà sui limiti delle metodologie avanzate e delineerà possibili direzioni future di ricerca nel campo del denoising di immagini di impronte digitali attraverso il deep learning.

L'importanza di questa ricerca risiede nelle sue applicazioni pratiche in vari contesti, inclusi sistemi di sicurezza avanzati, autenticazione biometrica e analisi forense.

Capitolo 2

Stato dell'Arte

Le immagini vengono inquinate da vari tipi di rumore durante la fase di acquisizione, di compressione o trasmissione. La perdita di informazioni e la scarsa qualità delle immagini creano notevoli difficoltà per molte attività di elaborazione delle immagini[8][16].

Il denoising di impronte digitali rappresenta uno dei fronti più dinamici e sfidanti nell’ambito del riconoscimento biometrico. Le impronte digitali, utilizzate ampiamente per l’identificazione personale data la loro unicità e invariabilità nel tempo, possono subire degradazioni a causa di fattori ambientali oppure di limitazioni o errori nel processo di acquisizione. Il rumore di tipo "*salt and pepper*", caratterizzato da disturbi improvvisi e sparsi sui pixel dell’immagine, e il blur, che sfoca i dettagli rendendo le linee meno definite, sono due dei principali nemici della qualità delle impronte digitali: questi tipi di rumore non solo compromettono l’accuratezza dell’identificazione ma possono anche ridurre l’efficacia dei sistemi di sicurezza basati su riconoscimento biometrico.

La necessità di denoising efficace, pertanto, non può essere sottostimata. L’obiettivo è, quindi, rimuovere il rumore, mantenendo - o addirittura migliorando - i dettagli critici dell’impronta digitale. Le tecniche di denoising si confrontano con la sfida di bilanciare la rimozione del rumore con la preservazione della struttura dell’impronta. L’avvento delle reti neurali ha segnato una svolta nel trattamento delle immagini digitali, denoising incluso. L’approccio tradizionale, basato su metodi di filtraggio e tecniche di elaborazione delle immagini, spesso si scontra con limiti ben definiti nella capacità di discriminare tra rumore e dettagli fini.

Al contrario, le reti neurali, grazie alla loro capacità di apprendere rappresentazioni complesse dei dati, offrono un mezzo più sofisticato ed efficace

per affrontare il problema del denoising, come evidenziato in "*Comparison of Image Denoising using Traditional Filter and Deep Learning Methods*" di Limshuebchuey A. et al.[14]. Le reti neurali sono un sottoinsieme del machine learning e sono l'elemento centrale degli algoritmi di deep learning. La loro struttura è ispirata al cervello umano, emulando il modo in cui i neuroni si inviano segnali reciprocamente. Le reti neurali artificiali sono composte da livelli di nodi che contengono un livello di input, uno o più livelli nascosti e un livello di output.

Come analizzato in "*Deep learning on image denoising: An overview*" di Chunwei Tian et al.[18], in "*Brief review of image denoising techniques*" di Fan L. et al. [7] e altri paper [9][17], il campo del denoising di immagini ha visto notevoli progressi grazie all'adozione di tecniche basate, appunto, sul deep learning, in particolare le **Reti Neurali Convoluzionali** (CNN, *Convolutional Neural Network*), che hanno brillantemente superato le tecniche tradizionali nella distinzione tra rumore e dettagli fini delle immagini: i documenti evidenziano, infatti, l'esistenza di diversi tipi di rumore e le corrispondenti tecniche di deep learning adottate per affrontarli, ognuno con le proprie sfide uniche. Ilesanmi, A.E. e Ilesanmi, T.O. nel loro paper "*Methods for image denoising using convolutional neural network: a review.*" [10] hanno constatato che l'utilizzo delle CNN ha addirittura portato ad un miglioramento dei dettagli critici delle immagini.

In "*Hybrid Deep Learning Framework for Reduction of Mixed Noise via Low Rank Noise Estimation*" di Kim, Dai-Gyoung et al.[12] viene presentata una variante di rete neurale convoluzionale, la EnCNN, in grado di ridurre il rumore misto (cioè proveniente da varie fonti) dalle immagini. In questo lavoro è stato fondamentale l'approfondimento sul rumore utilizzato: una combinazione di **blur**, ossia rumore gaussiano, e **salt-and-pepper**, cioè rumore impulsivo. Questa sfida rappresenta un compito non convenzionale per i modelli di denoising, spesso addestrati a rimuovere solo un singolo rumore.

L'articolo "*Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising*" di Kai Zhang et al. [22] propone un modello di deep learning innovativo, il **DnCNN** (*Denoising Convolutional Neural Network*), che segna un notevole progresso nel campo del denoising delle immagini.

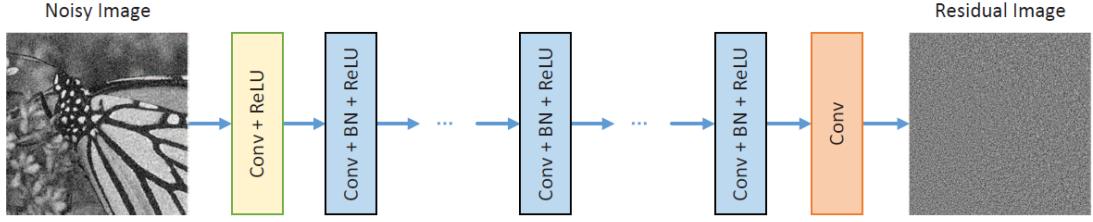


Figura 2.1: Denoising Convolutional Neural Network (DnCNN)[22]

Utilizzando strategie di apprendimento residuale e normalizzazione batch, questo modello non solo accelera il processo di addestramento ma migliora anche significativamente le prestazioni di denoising, gestendo diversi tipi di rumore e compiti di denoising con un unico modello.

Rispetto ai metodi tradizionali e allo stato dell’arte, come BM3D e WNNM, DnCNN dimostra superiorità, ottenendo risultati migliori in termini di *Peak Signal-to-Noise Ratio (PSNR)*[4] e *Structural Similarity Index Measure (SSIM)*[20] e qualitativamente più soddisfacenti, evidenziando la sua efficacia nel preservare dettagli e strutture dell’immagine pur eliminando il rumore. Questi avanzamenti indicano che le architetture di deep learning, adattabili a vari contesti di rumore senza addestramenti specifici, possono offrire soluzioni potenti e flessibili per il denoising delle immagini, rappresentando un passo significativo verso l’ottimizzazione delle tecniche di denoising nel dominio del riconoscimento biometrico e oltre.

Nel lavoro "*A deep convolutional neural network for salt-and-pepper noise removal using selective convolutional blocks*" di Ahmad Ali Rafiee e Mahmoud Farhang[2], viene proposto un modello innovativo, evoluzione della canonica CNN, denominato **SeConvNet**, per la rimozione del rumore di tipo salt-and-pepper (SAP) da immagini in scala di grigi e a colori. Questo approccio si distingue per l’introduzione di un nuovo blocco convoluzionale selettivo (**SeConv**), che permette a SeConvNet di superare i metodi esistenti per il denoising di SAP, specialmente a densità di rumore elevate e molto elevate.

Il modello SeConvNet si basa su un’architettura di rete profonda convoluzionale che sfrutta i blocchi SeConv per una stima iniziale dei pixel rumorosi, impiegando solo pixel non rumorosi per il calcolo. Questa strategia garantisce un’elevata efficacia nella preservazione dei dettagli e delle strutture delle im-

magini originarie, anche in presenza di forte rumore. La ricerca dimostra che SeConvNet ottiene prestazioni superiori rispetto ad altri approcci, offrendo risultati visivi notevolmente migliorati.

Il modello è stato addestrato e valutato su diversi dataset, inclusi immagini tradizionali di test e il Berkeley Segmentation Dataset (BSD68 per immagini in scala di grigi e CBSD68 per immagini a colori) e ha dimostrato una notevole capacità di restaurare immagini aggressivamente corrotte da rumore SAP, mantenendo al contempo dettagli fini ed elementi critici delle immagini.

"Enhanced Deep Residual Networks for Single Image Super-Resolution" di Bee Lim et al.[3] presenta un notevole avanzamento nel campo della super-risoluzione (SR) di immagini singole, proponendo una rete profonda di super-risoluzione (EDSR) che supera i metodi correnti di stato dell'arte in termini di efficacia. Questo avanzamento è ottenuto attraverso l'ottimizzazione dell'architettura di rete residuale convenzionale, eliminando moduli non necessari e ampliando le dimensioni del modello per migliorare ulteriormente le prestazioni senza compromettere l'efficienza computazionale. Uno dei contributi principali di questo studio è l'introduzione dell'apprendimento residuale per affrontare con successo la sfida della SR, dimostrando che l'eliminazione dei layer di normalizzazione batch dai blocchi residui porti a un significativo incremento delle prestazioni: si sottolinea come tali layer possano limitare la flessibilità della rete nel rappresentare le varie scale di rumore nelle immagini.

Il documento *"Residual Dense Network for Image Restoration"* di Zhang et al.[21] propone un approccio significativo all'elaborazione delle immagini attraverso l'uso di un modello di rete neurale chiamato *Residual Dense Network (RDN)*.

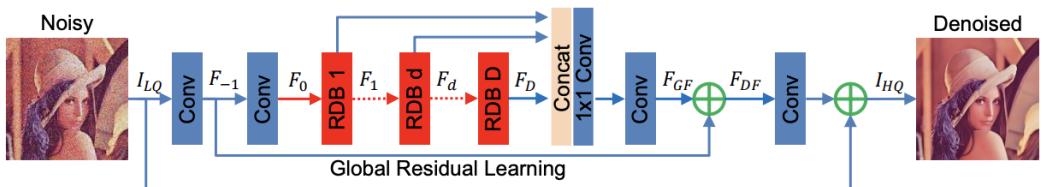


Figura 2.2: Residual Dense Net di Zhang et al.[21]

Questo approccio si distingue per la sua capacità di sfruttare efficacemente

le caratteristiche gerarchiche delle immagini, attingendo a tutte i layer convoluzionali per migliorare la qualità dell'immagine restaurata. Al cuore di questo modello vi è il *Residual Dense Block* (**RDB**), che permette un'intensa connessione tra i vari layer e promuove un meccanismo di memoria contigua, migliorando così l'efficienza nella cattura delle informazioni locali e globali.

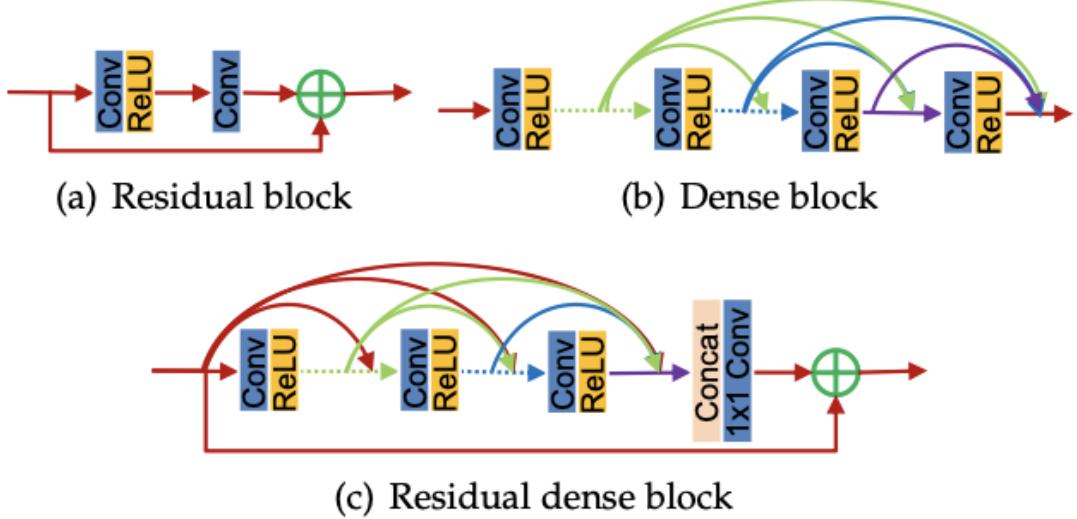


Figura 2.3: Residual Dense Block di Zhang et al.[21]

L'**RDN** si dimostra particolarmente efficace in diverse applicazioni di restauro dell'immagine, tra cui la super-risoluzione di singole immagini, la riduzione del rumore gaussiano, la diminuzione degli artefatti di compressione e il deblurring delle immagini: l'introduzione di strategie quali l'apprendimento residuale, la fusione delle caratteristiche locali e globali, e la considerazione delle caratteristiche gerarchiche dalle immagini originali a bassa qualità, consentono a RDN di ottenere un equilibrio ottimale tra l'efficienza computazionale e l'efficacia nel restauro delle immagini. Questi avanzamenti, insieme alla capacità del modello di gestire vari tipi di degradazione delle immagini, rendono RDN una soluzione promettente e versatile per il miglioramento della qualità delle immagini in un'ampia gamma di applicazioni pratiche. I test effettuati su set di dati standard e nel mondo reale mostrano che RDN supera i metodi precedenti e all'avanguardia, fornendo risultati qualitativamente e visivamente superiori.

Il report "Image Denoising Using a U-net" di Paavani Dua [6] presenta un

approccio innovativo al denoising delle immagini utilizzando una rete **U-net**, dimostrandosi superiore alle tecniche di denoising tradizionali come il filtraggio spaziale, il *thresholding wavelet* e il filtraggio nel dominio della trasformata. L'uso di reti U-net in questo contesto è significativo per la loro capacità di preservare i bordi delle immagini pur rimuovendo efficacemente il rumore, un aspetto cruciale nel miglioramento della qualità delle immagini, specialmente in applicazioni come l'elaborazione di immagini mediche.

Il progetto si basa sull'efficienza computazionale e sulla superiorità dei risultati prodotti dai CNN rispetto ai metodi tradizionali, utilizzando il dataset CIFAR-10 per addestramento e test, e sfruttando la potenza di calcolo delle GPU per ottimizzare ulteriormente il processo di *tuning* dei parametri. Attraverso la sperimentazione con vari tipi di rumore aggiunto alle immagini, come il rumore gaussiano e poissoniano, e l'utilizzo di diversi **ottimizzatori** come *Adam*, *RMSprop* e *SGD*, il modello U-net è stato raffinato per produrre risultati ottimali misurati in termini di **PSNR**. I risultati mostrano che l'ottimizzatore Adam ha portato alle prestazioni migliori, sottolineando l'efficacia dell'approccio basato su U-net nel denoising di immagini rispetto alle tecniche tradizionali.

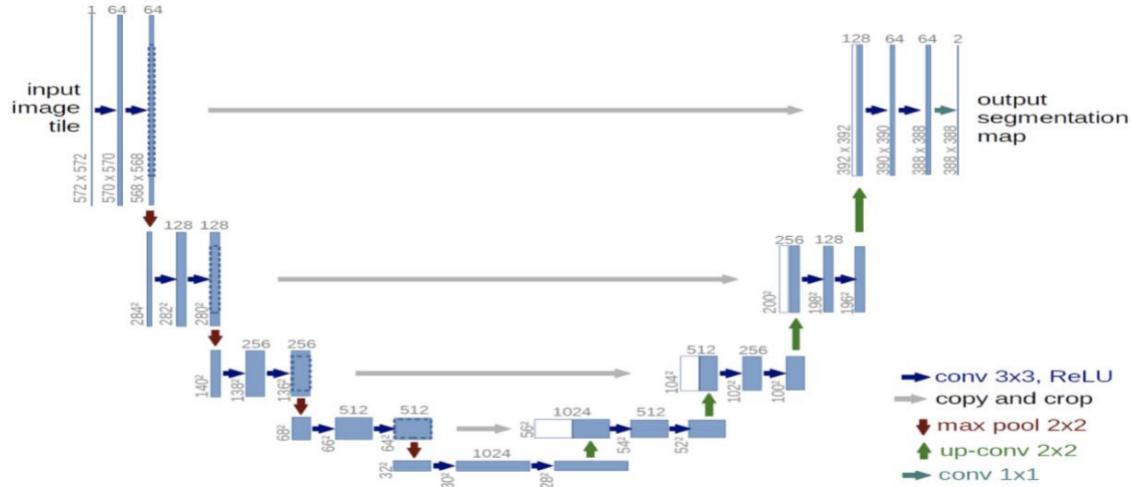


Figura 2.4: Denoising U-net di Paavani Dua[6]

Questa rete è caratterizzata da una struttura simmetrica encoder-decoder, arricchita da connessioni di salto che collegano direttamente gli strati dell'encoder con quelli corrispondenti del decoder. Questi collegamenti, noti come *skip connections*[1], consentono il trasferimento di informazioni di contesto ad alta risoluzione dal percorso di contrazione al percorso espansivo, garantendo

che dettagli cruciali non vengano persi durante la codifica e decodifica dei dati.

Anche i *Denoising Autoencoder* (**DAE**) e i *Deep Denoising Autoencoders* (**DDAE**) rappresentano un'avanzata evoluzione nel trattamento di dati complessi, dove il denoising gioca un ruolo cruciale nel migliorare la qualità e l'affidabilità delle interpretazioni.

Il principio alla base del DAE è quello di forzare lo strato nascosto dell'autoencoder a catturare caratteristiche più robuste mediante la ricostruzione di input puliti da quelli corrotti. Lo studio di Wang et al. [11] ha implementato un DAE per analizzare dati trascrittomici integrati da 13 studi pubblicati su tumori polmonari, comprendenti 1916 campioni di tessuto polmonare umano. L'utilizzo del DA ha permesso di scoprire una firma molecolare composta da multipli geni per l'adenocarcinoma polmonare (ADC).

Contrariamente alle tradizionali CNN, che sono generalmente orientate verso compiti di classificazione o regressione, i DAE e i DDAE sono progettati con l'obiettivo specifico di ricostruire l'immagine input, eliminando efficacemente le componenti di rumore senza sacrificare le informazioni essenziali del segnale originale.

Questo è ottenuto tramite un'architettura che, similmente all'U-net, include un encoder per comprimere i dati di ingresso e un decoder per la ricostruzione, utilizzando strati convoluzionali che permettono di apprendere rappresentazioni a più livelli dell'input. Nell'articolo "*Deep Denoising Autoencoder for Seismic Random Noise Attenuation*" di Saad e Chen [15], i DAE sono utilizzati per l'attenuazione del rumore casuale nei dati sismici, dimostrando la loro superiorità rispetto a metodi tradizionali.

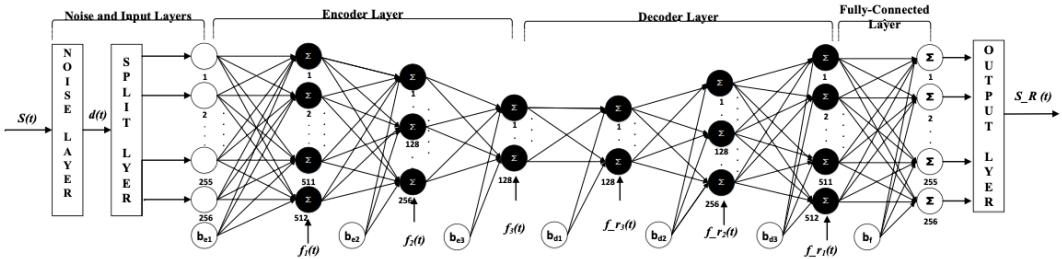


Figura 2.5: Deep Denoising Autoencoder di Saad e Chen[15]

Attraverso l'addestramento supervisionato su dati sintetici e l'applicazione non supervisionata su dati reali, i DAE mostrano una buona capacità di adattamento e flessibilità.

Capitolo 3

Dataset

Il cuore della nostra ricerca si fonda sull'utilizzo di un dataset variegato, costituito da circa 3.000 immagini di impronte digitali in formato .tif.

Queste immagini, tutte in scala di grigio, sono state messe a disposizione dalla **Fingerprint Verification Competition** (FVC) [19], un'iniziativa congiunta del **Biometric System Lab** dell'Università di Bologna, del **Pattern Recognition and Image Processing Laboratory** della Michigan State University e del **Biometric Test Center** della San Jose State University.

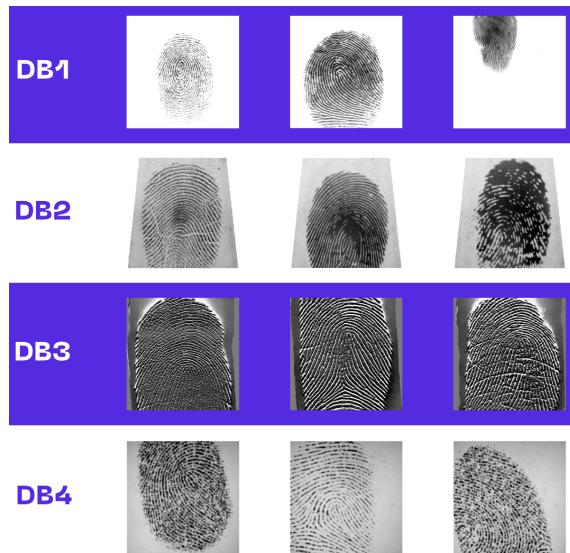


Figura 3.1: Suddivisione Dataset FVC 2004

La particolarità di questo dataset risiede nella sua suddivisione in quattro subset distinti, ciascuno caratterizzato da una specifica metodologia di acquisizione delle impronte digitali. Questa diversificazione è stata pensata per testare la robustezza e l'adattabilità dei sistemi di riconoscimento biometrico di fronte a varie condizioni di acquisizione, rendendolo uno strumento ideale

per la valutazione comparativa delle nostre architetture di deep learning nel contesto del denoising.

L'eterogeneità del dataset FVC offre un terreno di prova adeguato per verificare l'efficacia dei modelli nel **generalizzare** su differenti tipi di rumore e distorsioni, sfidando così le architetture a dimostrare la loro capacità di elaborare e ripulire immagini biometriche in **scenari realistici**.

La scelta di questo dataset per il nostro progetto di benchmarking non solo segue la tradizione delle competizioni FVC, che si sono affermate come benchmark di riferimento nell'ambito biometrico, ma apre anche nuove prospettive nella ricerca applicata al miglioramento delle tecniche di denoising, con l'obiettivo ultimo di potenziare la sicurezza e l'affidabilità dei sistemi di autenticazione biometrica.

Capitolo 4

Metodi

In questo capitolo, esploreremo in dettaglio i metodi e le tecniche adottate per la realizzazione della nostra ricerca. Centrali per il benchmarking sono le architetture delle reti neurali, scelte per la loro capacità di gestire complesse trasformazioni e apprendimenti dai dati. Particolare attenzione è stata rivolta alla selezione delle metriche di valutazione, cruciali per misurare l'efficacia e la precisione dei modelli proposti in contesti di denoising di immagini.

4.1 Architetture

La scelta delle architetture per il benchmarking è ricaduta su:

- **SeConvNet;**
- **Denoising Autoencoder;**
- **Residual Dense Net;**
- **Denoising U-net.**

Essendo le immagini in scala di grigio, i modelli proposti si limiteranno a gestire un unico canale colore.

4.1.1 SeConvNet

Quest'architettura sfrutta un blocco selettivo per focalizzarsi su aree specifiche dell'immagine dove i pixel sono considerati rumorosi e cercando di recuperare i valori originali basandosi sui valori dei pixel vicini non rumorosi.

```
@keras.saving.register_keras_serializable()
def SeConv_block(keras.layers.Layer):

    # non-noisy pixels map:
    M_hat = tf.math.not_equal(inputs, 0)
    M_hat = tf.dtypes.cast(M_hat, tf.float32)

    conv_input = tf.nn.conv2d(inputs, self.kernel, strides=[1, 1, 1, 1],
        padding='SAME', data_format='NHWC', dilations=None, name=None)
    conv_M_hat = tf.nn.conv2d(M_hat, self.kernel, strides=[1, 1, 1, 1],
        padding='SAME', data_format='NHWC', dilations=None, name=None)

    # find 0 in conv_M_hat and change to 1:
    is_zero_conv_M_hat = tf.equal(conv_M_hat, 0)
    is_zero_conv_M_hat = tf.dtypes.cast(is_zero_conv_M_hat, tf.float32)
    change_zero_to_one_conv_M_hat = tf.math.add(conv_M_hat,
                                                is_zero_conv_M_hat)

    S = tf.math.divide(conv_input, change_zero_to_one_conv_M_hat)

    # noisy pixels map:
    M = 1 - M_hat

    # calculate R:
    kernel_ones = np.ones((self.kernel_size, self.kernel_size,
        self.input_channels, 1))
    kernel_ones = tf.constant(kernel_ones, dtype=tf.float32)

    R = tf.nn.conv2d(M_hat, kernel_ones, strides=[1, 1, 1, 1],
        padding='SAME', data_format='NHWC', dilations=None, name=None)
    R = tf.math.greater_equal(R, tf.constant(self.kernel_size-2,
        dtype=tf.float32))

    R = tf.dtypes.cast(R, tf.float32)

    y = tf.math.multiply(tf.math.multiply(S, M), R) + inputs

    return y

#...
```

Il **Blocco Selettivo Convolutivo** (SeConv) è stato sviluppato per affrontare in maniera efficace il problema del rumore salt-and-pepper (SAP) nelle immagini. Tale rumore, caratterizzato da pixel casuali saturi (bianchi o neri), rappresenta una sfida notevole nel processo di denoising poiché può compro-

mettere significativamente la qualità dell'immagine se non gestito correttamente.

In questo blocco, viene inizialmente creata una mappa di pixel non rumorosi, dove i pixel con valore diverso da zero sono considerati non rumorosi (M_{hat}). In seguito, viene eseguita una convoluzione sia sull'input che sulla mappa dei pixel non rumorosi, utilizzando lo stesso *kernel* di convoluzione. Una volta trasformati gli zeri in uno per evitare divisioni per zero (S), si procede con la normalizzazione di entrambi gli output della convoluzione e, quindi, si genera una mappa di pixel rumorosi (M), l'inverso della mappa dei pixel non rumorosi (" M_{hat} "). Si utilizza, poi, un **kernel** di "soli uno" (**kernel_ones**) per eseguire una convoluzione su M , al fine di calcolare un peso (R) che tenga conto dell'intorno di ogni pixel. Infine, si applica la mappa finale pesata dei pixel rumorosi all'output normalizzato, e poi si aggiunge l'input originale per conservare i pixel non rumorosi (moltiplicazione di S , M ed R).

4.1.2 Denoising Autoencoder

La funzione principale di un autoencoder è di imparare una rappresentazione (encoding) per un set di dati, tipicamente per il compito di riduzione della dimensionalità o per scopi di denoising.

```
# Encoder
x = layers.Conv2D(filters=n_filters, kernel_size=3,
                  strides=2, activation='relu', padding='same')(inputs)
x = layers.Conv2D(filters=n_filters*2, kernel_size=3,
                  strides=2, activation='relu', padding='same')(x)

# Bottleneck
x = layers.Conv2D(filters=n_filters*4, kernel_size=3,
                  activation='relu', padding='same')(x)

# Decoder
x = layers.Conv2DTranspose(filters=n_filters*2, kernel_size=3,
                           strides=2, activation='relu', padding='same')(x)
x = layers.Conv2DTranspose(filters=n_filters, kernel_size=3,
                           strides=2, activation='relu', padding='same')(x)

# Output layer with sigmoid activation to ensure
# output values are between 0 and 1
outputs = layers.Conv2D(filters=1, kernel_size=3,
                       activation='sigmoid', padding='same')(x)
```

Questa architettura può essere suddivisa in tre parti principali: l'encoder, il bottleneck e il decoder.

Encoder

La prima parte dell'autoencoder è l'encoder. Questa sottorete lavora per comprimere l'input in una rappresentazione latente a dimensione ridotta. Ogni convoluzione (**Conv2D**) è seguita da un'attivazione ReLU (*Rectified Linear Unit*), che introduce la non linearità nel modello, permettendo alla rete di apprendere rappresentazioni più complesse. Man mano che si procede verso il bottleneck, il numero di filtri raddoppia, permettendo alla rete di catturare caratteristiche sempre più astratte dell'input.

Bottleneck

Questa è la parte centrale dell'autoencoder, dove i dati di input sono ridotti alla loro forma più compressa: Il bottleneck è dove si ottiene la rappresentazione latente o codificata dell'input. Viene eseguita, quindi, una convoluzione su questa rappresentazione, che è supposta contenere l'essenza informativa dei dati.

Decoder

La seconda metà dell'autoencoder è il decoder. Questo sottorete lavora per ricostruire l'input originale dalla rappresentazione latente. Per invertire il processo di riduzione dimensionale fatto dall'encoder, il decoder utilizza il **Conv2DTranspose** (a volte chiamato **deconvoluzione**). Questi layer incrementano la dimensione spaziale dei dati con l'obiettivo di riportare la rappresentazione latente alla dimensione originale dell'input. Infine, un layer Conv2D con attivazione **sigmoid** viene utilizzato per produrre l'output ricostruito. La funzione di attivazione sigmoid assicura che i valori di output siano mappati tra 0 e 1, il che è particolarmente utile per i dati di input normalizzati o per le immagini in cui i pixel hanno valori in questo range.

4.1.3 Residual Dense Net

L'RDN è un'architettura progettata principalmente per applicazioni di super-risoluzione di immagini. Questa rete utilizza un approccio innovativo per migliorare la qualità delle immagini ingrandite, combinando le idee di connessioni dense e residue per catturare un'ampia gamma di feature, fungendo da catalizzatore per il flusso di informazioni e gradienti attraverso la rete.

```
def dense_block(filters, kernel_size, C):
    def block(x):
        concatenated_features = x
        for _ in range(C):
            out = layers.Conv2D(filters, kernel_size,
                               padding='same', activation='relu')
            concatenated_features = layers.Concatenate()(
                [concatenated_features, out])
        # Local Feature Fusion
        out = layers.Conv2D(filters, kernel_size,
                           padding='same', activation='relu')
        concatenated_features = layers.Concatenate()(
            [concatenated_features, out])
    # Residual Connection
    out = layers.Add()([x, out])
    return out
    return block

def RDN(D, C, G, G0, kernel_size, scale, c_dim, input_shape):
    inputs = keras.Input(shape=input_shape)
    sfe1 = layers.Conv2D(G0, kernel_size, padding='same',
                        activation='relu')(inputs)
    sfe2 = layers.Conv2D(G, kernel_size, padding='same',
                        activation='relu')(sfe1)
    x = sfe2
    for _ in range(D):
        x = dense_block(G, kernel_size, C)(x)
    # Global Feature Fusion
    x = layers.Conv2D(G0, 1, padding='same',
                      activation='relu')(x)
    x = layers.Add()([x, sfe1])
    for _ in range(scale):
        x = layers.Conv2DTranspose(G0, kernel_size,
                                  strides=scale, padding='same')(x)
    output = layers.Conv2D(c_dim, kernel_size,
                          padding='same')(x)
    model = keras.Model(inputs=inputs, outputs=output)
    return model
```

L'architettura RDN si distingue per la sua capacità di apprendere feature dettagliate a livelli multi-scala, migliorando significativamente la precisione della ricostruzione dell'immagine. Questo viene realizzato attraverso diversi

componenti chiave:

- ***Shallow Feature Extraction (SFE)***: il primo passaggio nell’architettura RDN è l’estrazione di feature superficiali dall’immagine di input. Questo viene fatto attraverso due strati convoluzionali consecutivi. Questi strati iniziali sono responsabili dell’apprendimento di rappresentazioni primitive delle immagini che saranno ulteriormente elaborate dai blocchi successivi.
- ***Dense Block con collegamento residuale***: il cuore dell’RDN è costituito da blocchi densi, in cui ogni layer convoluzionale è connesso a tutti gli altri layer precedenti tramite concatenazioni. Questa struttura permette di catturare un insieme complesso e ricco di feature a diversi livelli. Ogni blocco denso si conclude con una ***Local Feature Fusion (LFF)***, che combina tutte le feature apprese nel blocco in un insieme unificato di feature. Ogni blocco denso include anche un collegamento residuo che somma l’input del blocco con il suo output. Questo aiuta a mitigare il problema del degrado della formazione in reti molto profonde, facilitando il flusso di informazioni e gradienti.
- ***Global Feature Fusion (GFF)***: dopo l’elaborazione attraverso i blocchi densi, le feature vengono combinate con una *Global Feature Fusion*, che integra le informazioni apprese attraverso diversi blocchi per formare una rappresentazione complessiva.
- ***Up-Sampling e Reconstruction***: l’ultimo passaggio dell’RDN è l’upsampling delle feature fuse a livello globale per raggiungere la dimensione desiderata dell’immagine di output, seguito da uno strato convoluzionale che produce l’immagine finale ad alta risoluzione.

L’implementazione proposta è stata concepita per essere flessibile e adattarsi a diverse dimensioni di input e requisiti di super-risoluzione. Infatti, è possibile decidere il numero di blocchi densi-residuali (**D**), il numero di convoluzioni all’interno di ciascun blocco (**C**) e il numero di filtri (**G** o **G0** per in numero di filtri iniziale).

4.1.4 Denoising U-net

La **U-net** è stata scelta per la sua capacità di preservare i bordi delle immagini mentre rimuove il rumore, un aspetto critico nel denoising di immagini, specialmente in quelle mediche. L'approccio utilizzato sfrutta le capacità delle reti neurali convoluzionali (CNN) per ottenere risultati superiori rispetto ai metodi tradizionali, con un'efficienza computazionale notevolmente migliorata.

```
def conv2d_block(input_tensor, n_filters, kernel_size=3):
    x = input_tensor
    for i in range(2):
        x = Conv2D(filters=n_filters,
                   kernel_size=(kernel_size, kernel_size),
                   kernel_initializer="Orthogonal",
                   padding="same")(x)
    x = Activation("relu")(x)
    return x

def denoising_unet(input_img_shape=(256, 256, 1), n_filters=32):
    inputs = Input(input_img_shape)
    # Encoder
    c1 = conv2d_block(inputs, n_filters=n_filters*1, kernel_size=3)
    p1 = layers.MaxPooling2D((2, 2))(c1)
    c2 = conv2d_block(p1, n_filters=n_filters*2, kernel_size=3)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    # Bottleneck
    bn = conv2d_block(p2, n_filters=n_filters*4, kernel_size=3)

    # Decoder
    u1 = layers.Conv2DTranspose(n_filters*2, (3, 3),
                               strides=(2, 2), padding='same')(bn)
    u1 = layers.concatenate([u1, c2])
    c3 = conv2d_block(u1, n_filters=n_filters*2,
                      kernel_size=3)
    u2 = layers.Conv2DTranspose(n_filters*1, (3, 3),
                               strides=(2, 2), padding='same')(c3)
    u2 = layers.concatenate([u2, c1])
    c4 = conv2d_block(u2, n_filters=n_filters*1,
                      kernel_size=3)

    # Output
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c4)
    model = keras.Model(inputs=[inputs], outputs=[outputs])
    return model
```

La rete sfrutta un blocco convoluzionale (**conv2d_block**), composto da due layer convoluzionali. Ogni layer utilizza un inizializzatore **ortogonale** per i pesi, favorendo la convergenza durante l'addestramento, e mantiene le dimen-

sioni spaziali delle immagini attraverso il **padding "same"**. L'attivazione **ReLU** è applicata per introdurre non linearità, permettendo al modello di apprendere rappresentazioni complesse.

La struttura del codice presentato può essere scomposta e analizzata come segue, per evidenziare le caratteristiche tecniche dell'implementazione di una **U-net** per denoising di immagini:

- **Encoder:** è formato da due **conv2d_block**, seguiti, ognuno, da un'operazione di **Max Pooling**. Il Max Pooling è una tecnica utilizzata nelle reti neurali convoluzionali per ridurre le dimensioni delle feature map, selezionando il valore massimo da finestre (o *kernel*) di dimensioni definite. Questo processo aiuta a rendere il modello più efficiente e robusto, enfatizzando le caratteristiche più importanti mentre si ottiene invarianza alla traslazione e si riduce l'overfitting.
- **Bottleneck:** situato al punto più profondo della rete, elabora le informazioni più astratte, utilizzando un **conv2d_block** senza ulteriore riduzione delle dimensioni spaziali.
- **Decoder:** si utilizzano operazioni di **Conv2DTranspose** per incrementare le dimensioni spaziali delle feature map. Questo è seguito dalla concatenazione con le corrispondenti feature map dall'encoder (**skip connections**), un approccio chiave nella U-net che aiuta a preservare i dettagli ad alta risoluzione durante la ricostruzione dell'immagine.
- **Output Layer:** L'ultimo strato, un layer **Conv2D** con una singola unità e attivazione **sigmoid**, produce l'immagine denoised. L'uso dell'attivazione sigmoid implica che l'output è una probabilità per ciascun pixel, indicando la presenza o l'assenza di caratteristiche rilevanti dopo il processo di denoising.

4.2 Metriche

Le metriche di valutazione giocano un ruolo fondamentale nei contesti di addestramento delle reti neurali in quanto forniscono misure quantitative della distorsione o della perdita di qualità rispetto all'immagine originale, permettendo di valutare l'efficacia di vari algoritmi di elaborazione delle immagini.

Le metriche selezionate per la sperimentazione sono:

- Mean Squared Error;
- Peak Signal-to-Noise Ratio;
- Structural Similarity Index Measure.

Mean Squared Error

Il *Mean Squared Error* (o **MSE**) misura la differenza quadratica media tra i valori dei pixel dell'immagine originale e quelli dell'immagine modificata. Valori più bassi di MSE indicano una minore distorsione e, di conseguenza, una migliore qualità dell'immagine ricostruita. La formula è la seguente:

$$MSE = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - y_{ij})^2$$

Figura 4.1: Formula MSE

dove:

- m e n sono rispettivamente l'altezza e la larghezza dell'immagine;
- x_{ij} è il valore del pixel nella posizione i, j dell'immagine originale;
- y_{ij} è il valore del pixel nella posizione i, j dell'immagine ricostruita.

Il MSE è ampiamente utilizzato per la sua semplicità e chiarezza interpretativa. Tuttavia, un limite significativo è che non tiene conto della percezione visiva umana: due immagini possono avere lo stesso MSE ma apparire molto diverse a un osservatore umano.

Peak Signal-to-Noise Ratio

Il *Peak Signal-to-Noise Ratio* (o **PSNR**) è una misura derivata dal MSE che esprime il rapporto tra il massimo possibile potere del segnale e il potere di distorsione (rumore) che influisce sulla sua rappresentazione. Valori più alti di PSNR indicano una migliore qualità dell'immagine. La formula è la seguente:

$$PSNR = 20 \cdot \log_{10} \left(\frac{MAX\{I\}}{\sqrt{MSE}} \right)$$

Figura 4.2: Formula PSNR

dove $MAX\{I\}$ rappresenta il valore massimo possibile dei pixel, che per le immagini a 8 bit è 255.

Il PSNR è più interpretabile rispetto al MSE e viene spesso utilizzato nell'ambito della compressione delle immagini e del video per valutare la perdita di qualità. Tuttavia, come il MSE, non riflette completamente la percezione visiva umana.

Structural Similarity Index Measure

La *Structural Similarity Index Measure* (o **SSIM**) misura la somiglianza strutturale tra due immagini. A differenza di MSE e PSNR, SSIM è progettato per emulare la percezione visiva umana, valutando le modifiche in termini di luminosità, contrasto e struttura. Il valore di SSIM oscilla fra -1 e +1, rappresentando, rispettivamente, totale discrepanza e massima somiglianza. La formula è la seguente:

$$SSIM(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}.$$

Figura 4.3: Formula SSIM

dove:

- μ_x e μ_y sono le medie dei pixel delle immagini x e y ;
- σ_x^2 e σ_y^2 sono le varianze dei pixel delle immagini x e y ;

- σ_{xy} è la covarianza tra le immagini x e y ;
- c_1 e c_2 sono due costanti piccole utilizzate per stabilizzare la divisione quando il denominatore è molto piccolo.

SSIM è ampiamente usato quando è importante mantenere la qualità visiva dell'immagine, come nel denoising, nella super-risoluzione e nella compressione. Fornisce una misura più vicina alla valutazione qualitativa umana rispetto a MSE e PSNR.

4.3 Optimizer

Sulla linea dell'articolo "*Hybrid Transformer-CNN for Real Image Denoising*" di M. Zhao et al. [23], la scelta dell'ottimizzatore è ricaduta su **AdamW**.

AdamW è una variante dell'ottimizzatore **Adam** che introduce la regolarizzazione dei pesi direttamente nell'aggiornamento dei pesi, piuttosto che come parte della funzione di perdita.

AdamW è spesso preferibile per via della sua migliore generalizzazione (tende a produrre modelli che generalizzano meglio su dati non visti) e per il suo maggior controllo sulla regolarizzazione dei pesi durante l'addestramento, punto critico del precedente Adam.

Capitolo 5

Sperimentazione

5.1 Tecnologie Utilizzate

La sperimentazione si è avvalsa della piattaforma cloud **Google Colab** usando **Python**.

Specifiche del sistema

Specifiche CPU:

- Modello: Intel(R) Xeon(R) CPU @ 2.30GHz
- Numero di socket: 1
- Core per socket: 4
- Thread per core: 2

Specifiche GPU:

- Nome GPU: Tesla T4
- Prestazioni: P8
- Consumo energetico: fino a 70W
- Utilizzo memoria: fino a 15360MiB
- CUDA Version: 12.2

Framework

La fase di sperimentazione si è avvalsa del framework **TensorFlow 2.15.0**, in particolare dell'interfaccia di alto livello **Keras**, per la costruzione, l'addestramento e la valutazione dei modelli di deep learning. La scelta di ta-

le framework ha garantito flessibilità, facilità di uso e l'accesso a un'ampia gamma di strumenti ottimizzati per la ricerca nel campo dell'intelligenza artificiale.

5.2 Data Preparation

La preparazione dei dati rappresenta un tassello fondamentale nel puzzle dell'apprendimento automatico. Affrontare le sfide poste dal set di dati eterogeneo richiede una strategia di normalizzazione che garantisca la compatibilità fra le diverse collezioni.

Per lo studio in questione, si è affrontata la disparità nelle dimensioni delle immagini provenienti da quattro distinti dataset di impronte digitali. Questi sono stati aggregati in un unico dataset, contenendo quindi l'insieme completo delle fingerprint disponibili. A tal fine, è stato cruciale il processo di **normalizzazione** dei dati, ovvero l'azione di uniformare il dataset per omogeneizzare alcune caratteristiche delle singole immagini, quali dimensioni e colore.

5.2.1 Rumore

Una volta ottenuto il dataset di immagini "pulite", sono state create quattro cartelle distinte, contenenti le immagini delle impronte digitali a cui è stato applicato **rumore** (diverso per ciascuna cartella):

- Salt and pepper 5%
- Salt and pepper 50%
- Blur
- Mixed (Salt and Pepper 20% + Blur)

Salt and pepper 5%

Il rumore "**salt and pepper**" si manifesta come pixel bianchi o neri sparsi casualmente sull'immagine, simulando l'effetto di granuli di sale e pepe. Con una densità del 5%, questo tipo di rumore rappresenta una condizione rela-

tivamente lieve, in cui la maggior parte dell'immagine rimane intatta e solo una piccola frazione di pixel viene alterata.

La scelta di includere questa leggera perturbazione ha permesso di valutare la sensibilità dei modelli agli interventi minimi, verificando la loro capacità di riconoscere e correggere le anomalie senza sovradimensionare la pulizia su immagini già quasi pulite.

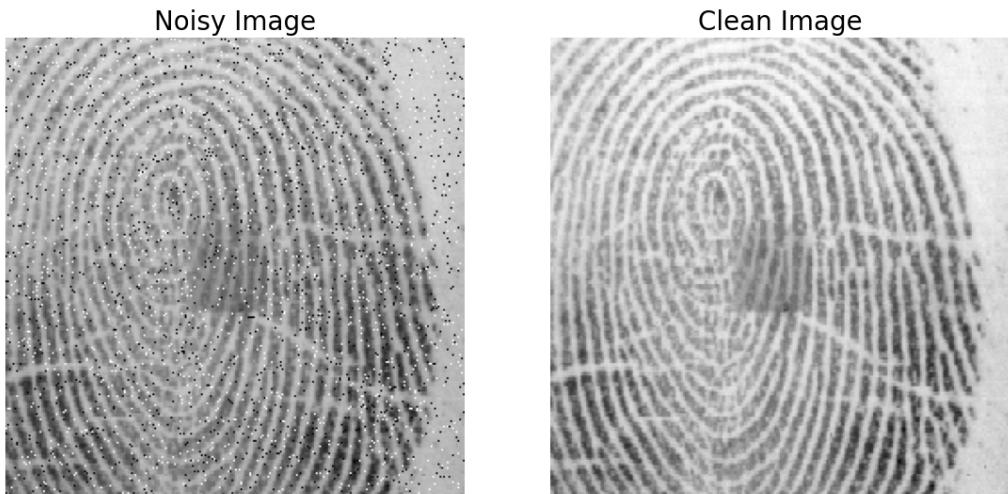


Figura 5.1: Esempio di immagine sporcata, 5% SAP

Salt and pepper 50%

Incrementando la densità del rumore "**salt and pepper**" al 50%, la sfida diventa significativamente più complessa. A questo livello, metà dei pixel dell'immagine sono alterati, provocando una sostanziale perdita di informazioni visive e dettagli. Questa condizione estrema testa la robustezza del modello nel ricostruire fedelmente le immagini a partire da dati fortemente corrotti, evidenziando l'efficacia dell'algoritmo nel gestire situazioni di elevata incertezza e nel ripristinare elementi fondamentali dell'immagine.

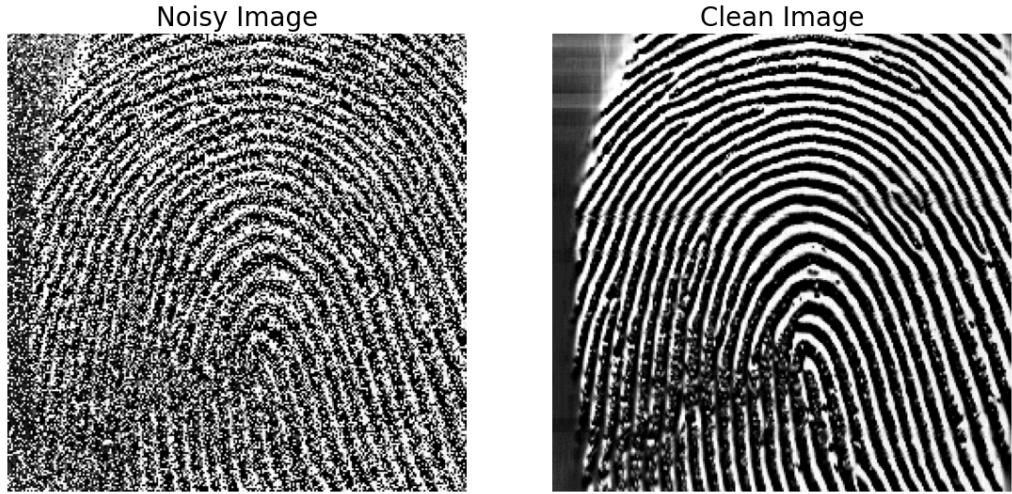


Figura 5.2: Esempio di immagine sporcata, 50% SAP

Blur

Il rumore di tipo "**blur**" si riferisce a un effetto di sfocatura sull'immagine, causato tipicamente da movimento o da limitazioni ottiche durante l'acquisizione dell'immagine stessa. A differenza del rumore "salt and pepper", che colpisce pixel individuali, il blur altera l'immagine in modo più uniforme e può mascherare dettagli fini e texture. La sua inclusione negli esperimenti mira a valutare la capacità dei modelli di denoising di affrontare perdite di nitidezza, ripristinando la chiarezza e la definizione delle immagini biometriche.

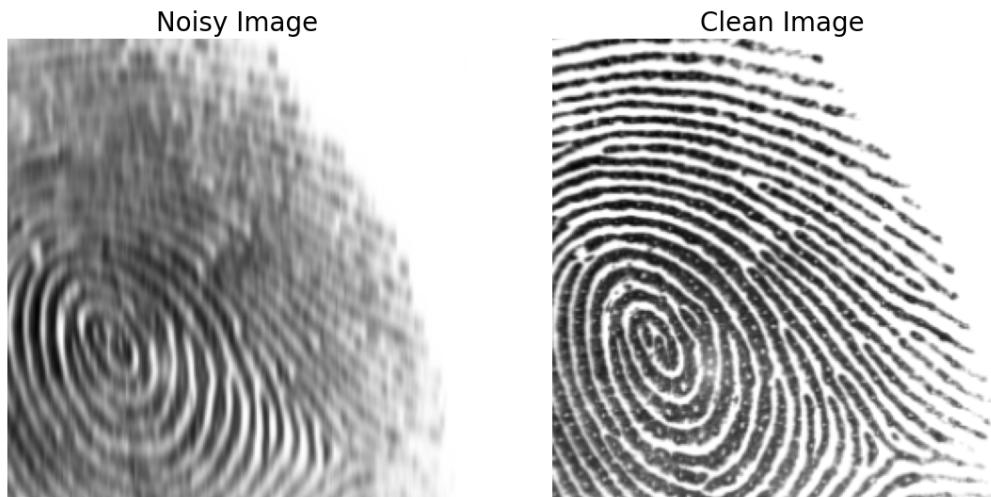


Figura 5.3: Esempio di immagine sporcata, BLURRED

Mixed (Salt and Pepper 20% + Blur)

La combinazione di rumore "**salt and pepper**" ad alta densità (20%) con l'effetto "**blur**" rappresenta il test più impegnativo tra quelli considerati. Questo scenario "mixed" simula condizioni realistiche e particolarmente avverse, in cui più fattori di degradazione dell'immagine coesistono, sfidando il modello su più fronti. La scelta di testare i modelli sotto queste condizioni complesse non solo dimostra la loro versatilità e capacità di adattamento ma fornisce anche indicazioni preziose su come differenti tipologie di rumore interagiscono tra loro e sulle strategie più efficaci per affrontarle simultaneamente.

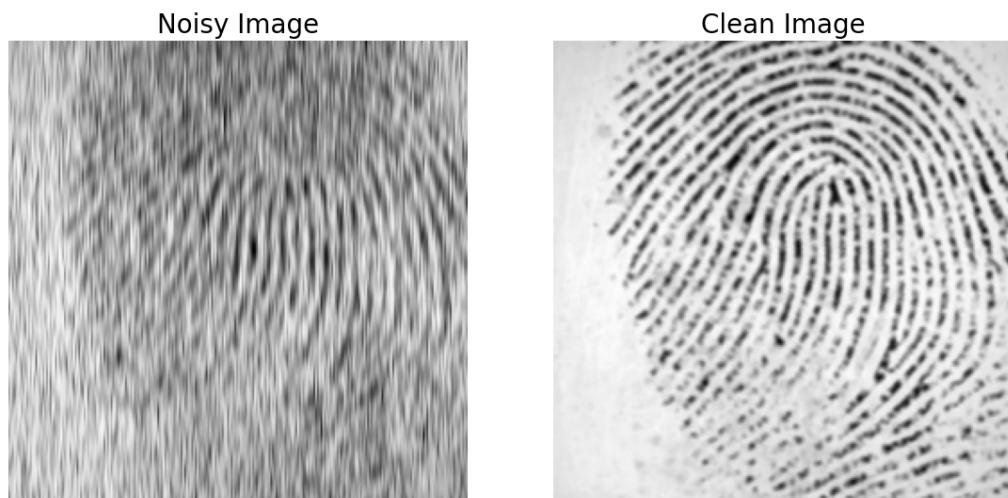


Figura 5.4: Esempio di immagine sporcata, MIXED

5.2.2 Divisione del Dataset

La divisione del dataset ha seguito le proporzioni di 75/15/10 per gli insiemi di addestramento (train), validazione (validation) e test.

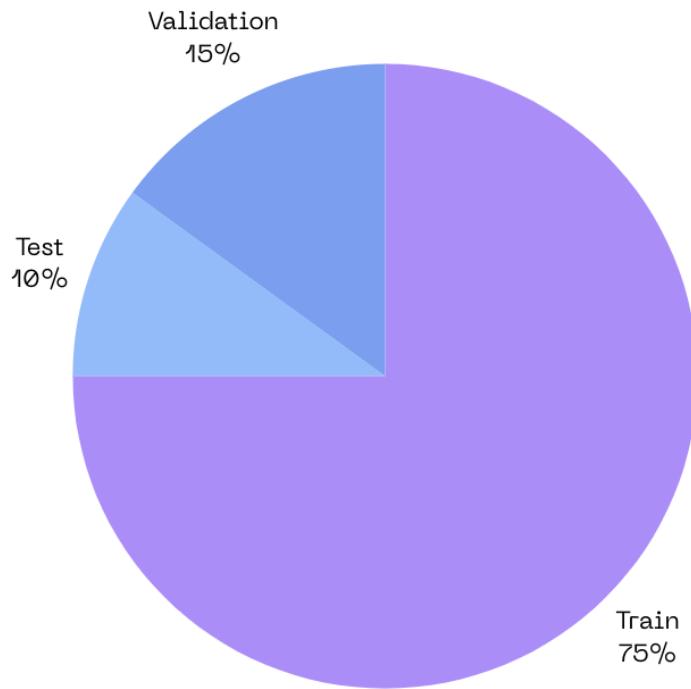


Figura 5.5: Divisione dataset

- **Addestramento:** assegnare la maggior parte del dataset all'insieme di addestramento è fondamentale per garantire che il modello abbia accesso a una quantità sufficientemente ampia di dati;
- **Validazione:** l'insieme di validazione svolge un ruolo cruciale nel processo di sviluppo del modello, consentendo di monitorare la sua performance su un set di dati non utilizzati durante l'addestramento;
- **Valutazione:** riservare una porzione del dataset per il test finale assicura che si possa valutare in modo affidabile quanto bene il modello sia in grado di generalizzare su nuovi dati, ovvero dati che non ha mai visto durante l'addestramento o la validazione.

5.2.3 Normalizzazione

Inizialmente si è contemplata l'applicazione di tecniche di **padding** e **reshaping** in diverse dimensioni per uniformare le immagini: tale approccio non si è rivelato ottimale. Un'analisi più approfondita ha suggerito che il padding avrebbe potuto introdurre artefatti nei dati, che la rete avrebbe dovuto imparare a ignorare nel corso dell'addestramento, potenzialmente prolungando la fase di apprendimento e distogliendo l'attenzione della rete dalle caratteristiche rilevanti per il compito primario, ossia il denoising.

Pertanto, si è deciso di adottare una strategia alternativa: il **crop** delle immagini.

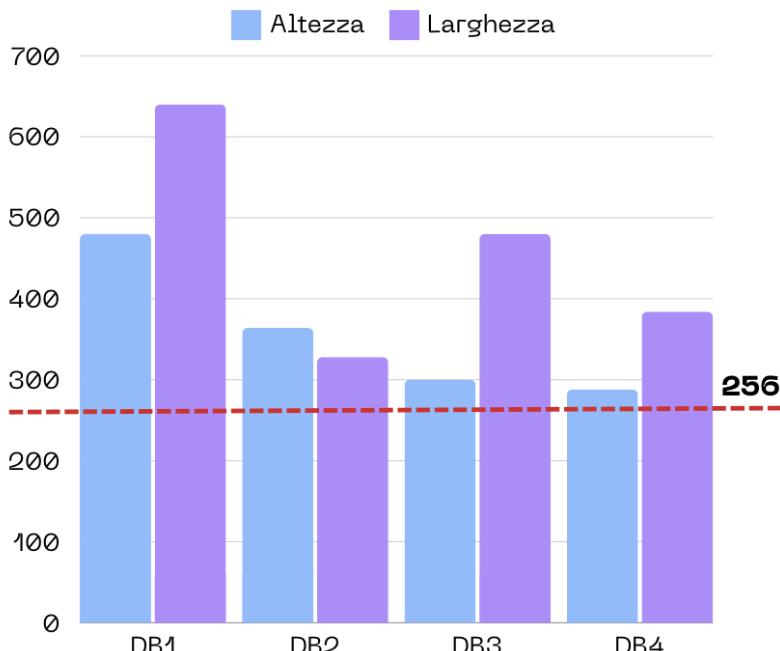


Figura 5.6: Dimensioni delle immagini nei quattro subset FVC

Selezionando un'area centrale di **256x256 pixel**, una dimensione comune e inferiore rispetto a tutte le risoluzioni presenti nei dataset, si è optato per un approccio che potrebbe intuitivamente sembrare una semplificazione, ma che in realtà si basa su osservazioni empiriche. Attraverso una serie di esperimenti preliminari, è stato ipotizzato che il crop concentri l'attenzione della rete neurale sulle informazioni essenziali, eliminando l'esigenza di allocare risorse computazionali per discriminare tra informazioni rilevanti e porzioni di immagine non informative.

Benché questo processo possa comportare la perdita di una porzione di dati, si è ritenuto che il vantaggio di avere un set di dati uniformato e di dimensioni gestibili fosse una soluzione pragmatica per massimizzare l'efficienza del modello nel riconoscere le caratteristiche cruciali delle impronte digitali e, quindi, rimuovere il rumore da esse.

5.2.4 Data Generator

L'integrazione del crop e della normalizzazione delle immagini è avvenuto all'interno della classe "***FingerprintDataGenerator***", sviluppata per fornire un flusso continuo e ottimizzato di dati ai nostri modelli.

```

class FingerprintDataGenerator(keras.utils.Sequence):
    def __init__(self, clean_images_dir, noised_images_dir,
                 batch_size, crop_size=(altezza, lunghezza)):
        self.clean_images_dir = clean_images_dir
        self.noised_images_dir = noised_images_dir
        self.batch_size = batch_size
        self.crop_size = crop_size
        self.image_filenames = [f for f
                               in os.listdir(clean_images_dir)
                               if os.path.isfile(os.path.join(clean_images_dir,
                               f))]
        random.shuffle(self.image_filenames)

    def __len__():
        return len(self.image_filenames) // self.batch_size

    def __getitem__(self, idx):
        batch_filenames = self.image_filenames[idx *
                                              self.batch_size:(idx + 1) * self.batch_size]

        batch_x = np.zeros((self.batch_size,) +
                           self.crop_size + (1,), dtype=np.float32)
        batch_y = np.zeros((self.batch_size,) +
                           self.crop_size + (1,), dtype=np.float32)

        for i, filename in enumerate(batch_filenames):
            clean_path = os.path.join(self.clean_images_dir,
                                      filename)
            noised_path = os.path.join(self.noised_images_dir,
                                       filename)

            clean_img = Image.open(clean_path).convert('L')
            noised_img = Image.open(noised_path).convert('L')

            # Cropping
            clean_img = self.crop_center(clean_img,
                                         self.crop_size[0], self.crop_size[1])
            noised_img = self.crop_center(noised_img,
                                         self.crop_size[0], self.crop_size[1])

            # Convert to numpy array and normalize
            batch_x[i] = np.expand_dims(np.array(noised_img)/255.0,
                                         axis=-1)
            batch_y[i] = np.expand_dims(np.array(clean_img)/255.0,
                                         axis=-1)
        return batch_x, batch_y

    def crop_center(self, img, cropx, cropy):
        img_width, img_height = img.size
        startx = img_width//2 - cropx//2
        starty = img_height//2 - cropy//2
        return img.crop((startx, starty,
                         startx+cropx, starty+cropy))

```

FingerprintDataGenerator è una classe personalizzata che estende **keras.utils.Sequence**. Essa rappresenta un approccio per gestire l'input e l'output di un modello di deep learning dedicato al denoising di impronte digitali, sfruttando i vantaggi della generazione di dati **"on-the-fly"** durante l'addestramento.

Nel metodo **__init__**, il generatore accetta percorsi delle directory contenenti immagini "pulite" e "rumorose", una dimensione del **batch** (ossia il numero di campioni osservati contemporaneamente dal modello in fase di apprendimento), e le dimensioni per il crop delle immagini. Una lista di nomi di file viene generata esplorando la directory delle immagini pulite e **mescolata** per garantire la varietà nell'addestramento.

Il metodo **__len__** calcola il numero totale di **lotti** disponibili nel dataset, dividendo la lunghezza dell'elenco dei nomi dei file per la dimensione del batch specificata.

__getitem__ si occupa della preparazione effettiva dei lotti di dati. Dopo il **cropping**, le immagini vengono convertite in un formato elaborabile dalle reti neurali, ovvero in **vettori NumPy**, e successivamente **normalizzate**, dividendole per 255. Questo passaggio trasforma i valori dei pixel da un intervallo di 0-255 (scala di grigio) a 0-1, rendendo i dati più gestibili per la rete neurale durante l'addestramento.

Vantaggi dell'approccio

L'approccio del generatore di dati basato su Sequence ha diversi vantaggi:

- **Efficienza della Memoria:** caricando in memoria solo i batch di dati necessari per ogni step di addestramento, si riduce il consumo di risorse, consentendo di lavorare con dataset di grandi dimensioni.
- **Parallelismo:** l'addestramento può sfruttare il parallelismo su più core, accelerando il processo grazie alla natura *thread-safe* della classe Sequence. Un oggetto, classe o funzione è considerata thread-safe se mantiene la coerenza dei dati e si comporta correttamente quando ne si concede l'accesso contemporaneo da più thread senza causare conflitti o stati inconsistenti.
- **Flessibilità:** il generatore può facilmente adattarsi a variazioni nelle dimensioni delle immagini e nelle strategie di preprocessing, rendendolo uno strumento versatile per l'addestramento di modelli di deep learning.

5.3 Addestramento

L'addestramento dei modelli è la fase centrale di questo studio. Essa mira a ottimizzare i pesi della rete neurale affinché possa eseguire con successo il denoising di immagini. Il modello apprende iterativamente a ridurre le discrepanze (o "perdite") tra le sue previsioni e i valori reali o attesi.

```

def execute_training_procedure(model, model_name, path_save_finals,
noisy_folder, epochs, batch_size, flag_testing = False, num_samples = 200):
    # Callbacks
    callbacks_path = '/content/drive/MyDrive/fingerprints/training'

    timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    destination_folder_name = f'{model_name}_{timestamp}'
    destination_path = os.path.join(callbacks_path, destination_folder_name)

    logs_path_save = os.path.join(destination_path, 'logs')
    os.mkdir(destination_path)
    os.mkdir(logs_path_save)

    print(f'Path di destinazione: {str(destination_path)}')
    test_clean_path = os.path.join(clean_path, 'test')
    val_clean_path = os.path.join(clean_path, 'val')
    train_clean_path = os.path.join(clean_path, 'train')
    test_noisy_path = os.path.join(noisy_folder, 'test')
    val_noisy_path = os.path.join(noisy_folder, 'val')
    train_noisy_path = os.path.join(noisy_folder, 'train')
    data_generator = FingerprintDataGenerator(train_clean_path,
                                              train_noisy_path, batch_size=batch_size, divide = divide)
    val_generator = FingerprintDataGenerator(val_clean_path,
                                              val_noisy_path, batch_size=batch_size, divide = divide)
    test_generator = FingerprintDataGenerator(test_clean_path,
                                              test_noisy_path, batch_size = batch_size, divide = divide)
    v_callbacks=[

        keras.callbacks.ModelCheckpoint(os.path.join(destination_path,
                                                     'model_checkpoint.keras'), save_best_only=True),
        keras.callbacks.EarlyStopping(monitor='val_loss',
                                     min_delta=1e-5, patience=10, verbose=1, mode='min'),
        keras.callbacks.TensorBoard(log_dir=logs_path_save),
        keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                         min_delta=3e-5, factor=0.1, patience=5,
                                         min_lr=1e-5, verbose=1, mode='min', cooldown=2)
    ]

    model.fit(data_generator, epochs=epochs,
              validation_data=val_generator, verbose = 1,
              callbacks = v_callbacks)

    # Valutazione del modello
    print("Evaluating model...")
    model.evaluate(test_generator, verbose = 1, callbacks = v_callbacks)

    path_model_s = os.path.join(path_save_finals, model_name)
    model.save(path_model_s)

```

Il codice inizia configurando i percorsi per salvare i risultati dell’addestramento e i log, oltre a creare generatori di dati per l’addestramento, la validazione e il test. Il metodo ***model.fit*** viene utilizzato per avviare l’addestramento del modello sfruttando i dati forniti dai generatori di addestramento e di validazione. I callbacks definiti vengono passati a fit per monitorare e guidare l’addestramento. Dopo l’addestramento, il modello viene valutato sul dataset di test per verificare la sua capacità di generalizzare su dati non visti durante l’addestramento.

Infine, il modello addestrato viene salvato in un percorso specificato per usi futuri o ulteriori analisi.

Di seguito, i parametri utilizzati per il training:

- **epochs** = 100;
- **batch_size** = 16;
- **learning_rate** iniziale = 10^{-3}
- **val_loss** = MSE.

5.3.1 Callbacks

I **callbacks** sono strumenti di Keras che permettono di monitorare l’addestramento di un modello a vari livelli di granularità, di intervenire automaticamente sotto certe condizioni (ad esempio, per prevenire l’overfitting) o di eseguire azioni specifiche a determinati punti dell’addestramento. Di seguito una breve descrizione di ogni callback utilizzato:

ModelCheckpoint

Salva il modello al percorso specificato dopo ogni epoca, ma solo se il modello migliorato rispetto all’epoca precedente in termini di **val_loss**. Questo garantisce di mantenere sempre la versione del modello con le migliori prestazioni sulla validazione.

EarlyStopping

Interrompe l'addestramento quando una metrica monitorata **val_loss** non ha formato un miglioramento di valore minimo **min_delta**. La *patience* determina quante epoche attendere dopo l'ultimo miglioramento prima di interrompere l'addestramento. Questo evita sprechi di risorse e previene l'overfitting.

TensorBoard

Fornisce una visualizzazione dettagliata dell'addestramento, consentendo di monitorare metriche come la perdita e l'accuratezza, nonché i pesi e i gradienti del modello, attraverso un'interfaccia web.

ReduceLROnPlateau

Riduce il **learning rate** quando una metrica monitorata (**val_loss**) non ha formato un miglioramento di valore minimo **min_delta**. Riducendo il learning rate, si cerca di "affinare" ulteriormente l'ottimizzazione, evitando potenzialmente di rimanere bloccati in minimi locali. Rappresenta un parametro fondamentale in fase di apprendimento lo **scheduling** del learning rate è da sempre un tema importante nell'ambito delle reti neurali[5]. Si può impostare un limite inferiore al learning rate attraverso il parametro **min_lr**.

Capitolo 6

Risultati

Questo capitolo mira a illustrare nel dettaglio i risultati ottenuti dal processo di addestramento dei modelli sui diversi tipi di rumore trattati, utilizzando come metriche di riferimento l'**SSIM** (*Structural Similarity Index Measure*), il **PSNR** (*Peak Signal-to-Noise Ratio*) e il **MSE** (*Mean Squared Error*) come funzione di perdita (**loss**).

Per ogni rumore trattato:

- Verrà prodotta una tabella riassuntiva delle metriche ottenute dai modelli in fase di **testing**.
- Attraverso l'utilizzo di TensorBoard verrà fornito al contempo un feedback visivo sulla convergenza del modello e sulla sua capacità di generalizzazione. Per semplicità, sono state mostrate solo le metriche acquisite in fase di validazione.
- In aggiunta alla presentazione dei risultati, il paragrafo includerà anche esempi visivi delle previsioni (***predictions***) del modello.

6.1 Salt And Pepper, 5%

	SeConvNet	DAE	RDN	UNet
MSE	5.7e-03	5.4813e-04	2.5678e-04	3.0195e-04
PSNR	23.5518	36.0727	42.1236	40.4362
SSIM	0.8683	0.9876	0.9963	0.9953

Figura 6.1: Risultati SAP - 5%

Con un ottimo risultato da tutte le architetture, il rumore “Salt And Pepper” al 5% di intensità rappresenta lo scenario più semplice fra quelli proposti. La SeConvNet si dimostra non performante come le altre architetture proposte.

6.1.1 Tensorboard

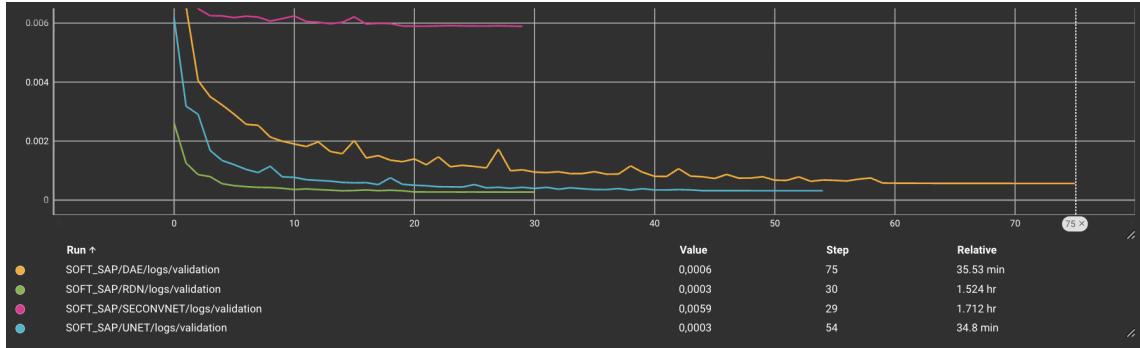


Figura 6.2: Validation loss (MSE) - SAP 5%

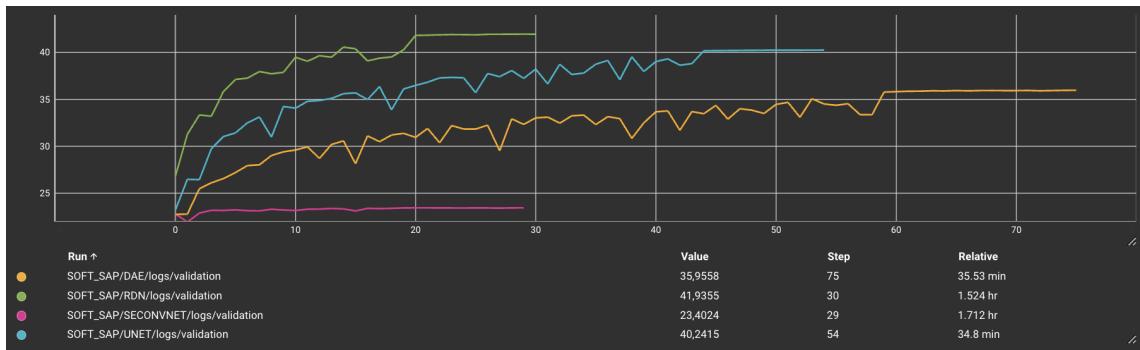


Figura 6.3: PSNR - SAP 5%

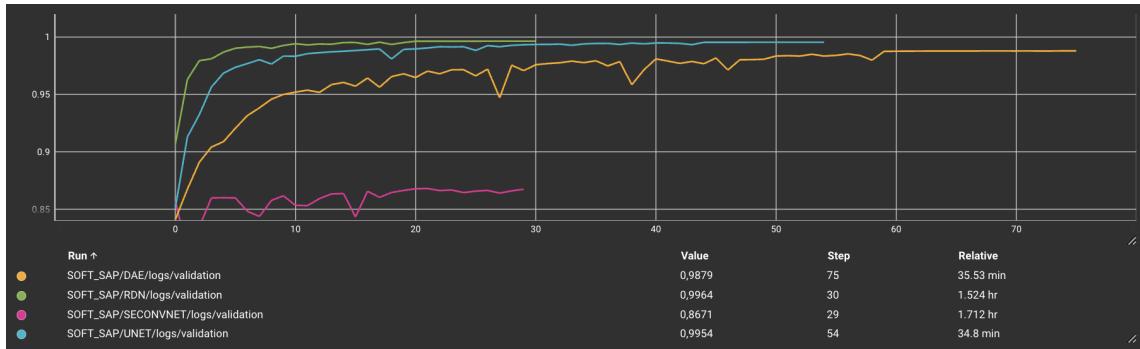


Figura 6.4: SSIM - SAP 5%

Dall’analisi dei dati di addestramento, emerge un distinto contrasto nelle prestazioni della **SeConvNet** rispetto alle altre architetture considerate. In particolare, la SeConvNet ha attivato il meccanismo di *early stopping* attorno alla trentesima epoca, segnalando una performance che, sebbene rientri in una soglia di accettabilità in termini assoluti, si rivela modesta al confronto diretto con le controparti.

Parallelamente, la *Residual Dense Network* (**RDN**) ha anch’essa innescato l’*early stopping* nella medesima finestra temporale delle epoche, tuttavia mani-

festando un livello di efficacia superiore a tutte le altre architetture esaminate. È cruciale osservare che il numero di epoche necessarie per l'addestramento non riflette linearmente il tempo computazionale richiesto: nonostante una convergenza più rapida, evidenziata da un minor numero di epoche, la RDN ha richiesto significativamente più tempo, circa un'ora e mezza, per completare il suo ciclo di addestramento, quasi il triplo del tempo impiegato dai modelli **DAE** (*Denoising Autoencoder*) e **U-net**.

I modelli DAE e U-net, d'altra parte, hanno evidenziato prestazioni elevate con tempi di addestramento più contenuti, oscillando tra i 30 e i 40 minuti. Questa osservazione sottolinea l'efficienza di tali architetture nel bilanciare efficacemente il *trade-off* tra qualità dei risultati e impegno computazionale, posizionandosi come soluzioni ottimali per contesti in cui le risorse di calcolo sono un fattore critico.

6.1.2 Predictions

Di seguito, una dimostrazione grafica di previsione (*prediction*) per l'esperimento "Salt And Pepper - 5%".



Figura 6.5: Prediction di un campione casuale nel test set - Salt And Pepper 5%

Segue un dettaglio sulla prediction della **RDN**.

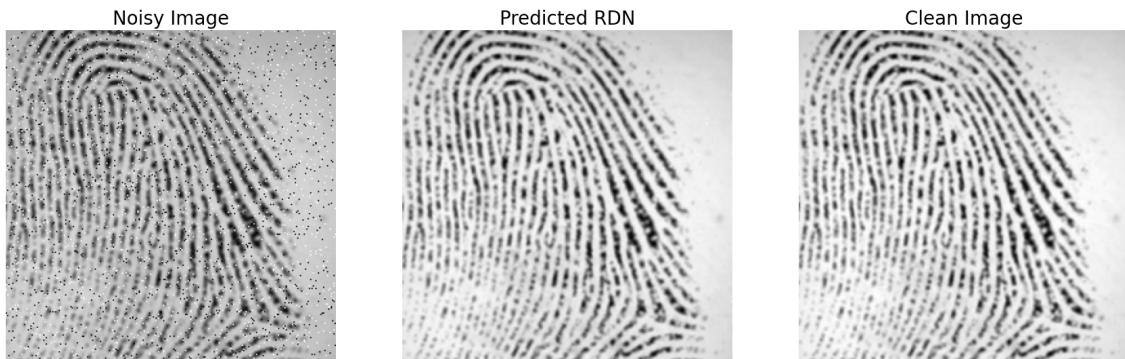


Figura 6.6: Dettaglio prediction RDN - Salt And Pepper 5%

6.2 Salt And Pepper, 50%

	SeConvNet	DAE	RDN	UNet
MSE	7.065e-02	4.319e-03	5.346e-03	4.269e-03
PSNR	12.5807	27.3627	25.8811	27.5152
SSIM	0.4199	0.9266	0.8962	0.9304

Figura 6.7: Risultati SAP - 50%

Nel contesto “Salt And Pepper” al 50% di intensità, sia U-Net che DAE hanno raggiunto degli ottimi risultati, dimostrando grandi capacità di ricostruzione dell’immagine gravemente deteriorata.

6.2.1 Tensorboard

Per garantire la chiarezza e la leggibilità della presentazione dei risultati, si è scelto di focalizzare l'attenzione sulle architetture DAE, RDN e U-net. Questa decisione è stata guidata dalla considerazione che le performance relativamente inferiori della SeConvNet potrebbero compromettere l'efficacia della visualizzazione grafica, rendendo meno immediata la comprensione del confronto tra le diverse architetture.

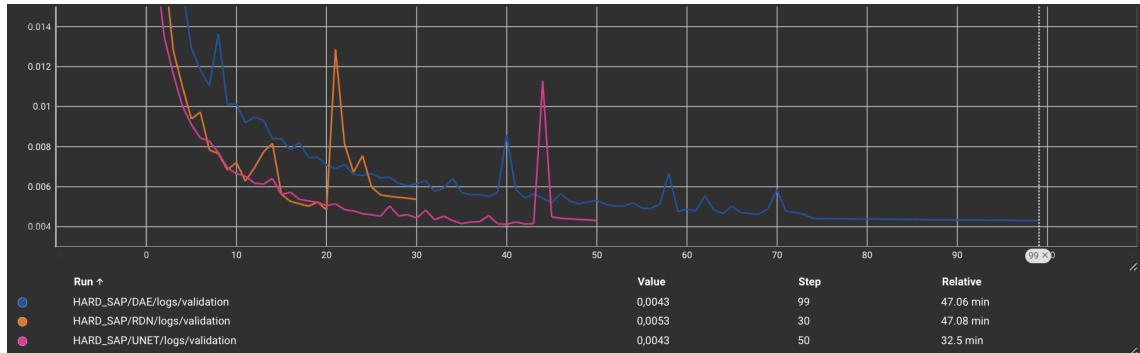


Figura 6.8: Validation loss (MSE) - SAP 50%

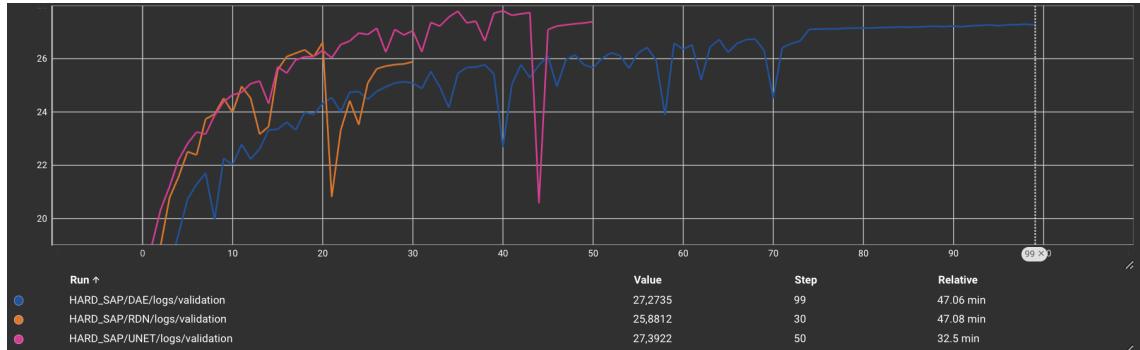


Figura 6.9: PSNR - SAP 50%

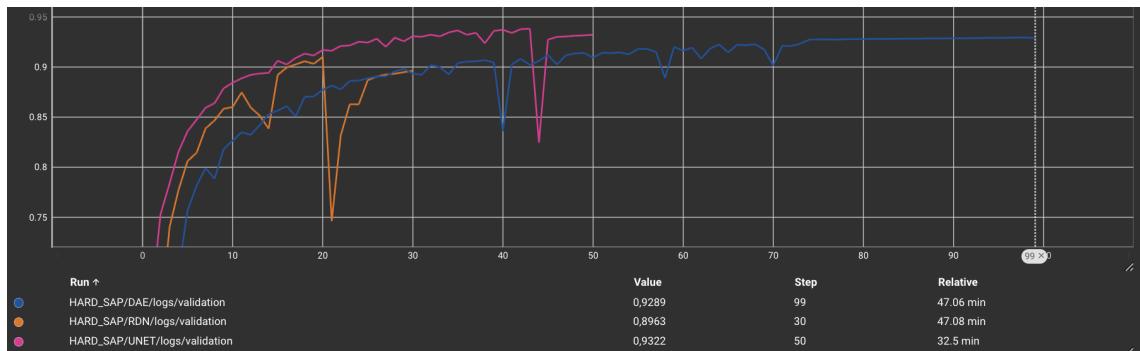


Figura 6.10: SSIM - SAP 50%

Durante l'addestramento, è possibile osservare dei picchi negativi nelle performance di tutte e tre le architetture, suggerendo momenti in cui i modelli

potrebbero aver imparato caratteristiche non ottimali o meno rilevanti per il compito di denoising.

Questi picchi negativi, potenzialmente attribuibili a un'apprendimento temporaneo di feature non utili o addirittura **fuorvianti**, segnalano delle deviazioni nel processo di ottimizzazione che potrebbero aver portato le reti a concentrarsi su dettagli non pertinenti alla rimozione efficace del rumore. Nonostante tali oscillazioni, la rete **DAE** mostra una **resilienza** maggiore, con picchi di intensità inferiore rispetto alle controparti U-net e RDN. Questa caratteristica ha permesso al DAE di proseguire l'addestramento per un numero maggiore di epoch, suggerendo una capacità di recupero e adattamento superiori in presenza di apprendimenti potenzialmente errati.

Interessante notare come, in questo scenario, la **RDN** abbia terminato il suo ciclo di addestramento prima delle altre due reti, senza tuttavia manifestare un vantaggio prestazionale. Al contrario, questo arresto anticipato, non accompagnato da una superiorità nelle metriche considerate, potrebbe indicare una minore efficacia dell'architettura RDN nel contesto specifico del denoising, almeno rispetto alla capacità della DAE di gestire e superare le fluttuazioni dell'apprendimento.

U-net, pur essendo stata influenzata da oscillazioni simili a quelle osservate per la RDN, ha dimostrato un andamento complessivo migliore. La presenza di picchi negativi, sebbene indicativa di momenti di apprendimento non ottimale, non ha precluso la prosecuzione dell'addestramento oltre il punto di arresto della RDN, suggerendo una potenziale resilienza dell'architettura U-net.

6.2.2 Predictions

Di seguito, una dimostrazione grafica di previsione (*prediction*) per l'esperimento "Salt And Pepper - 50%".



Figura 6.11: Prediction di un campione casuale nel test set - Salt And Pepper 50%

Segue un dettaglio sulla prediction della **U-net**.

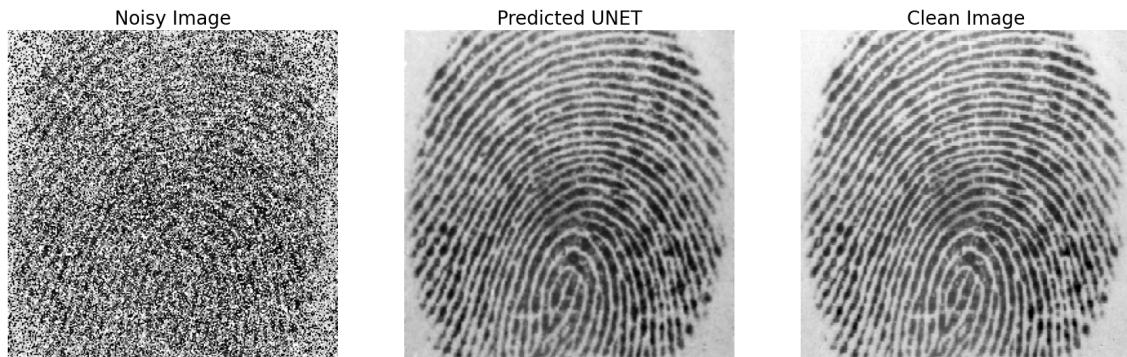


Figura 6.12: Dettaglio prediction U-net - Salt And Pepper 50%

Al di là della mera eliminazione del rumore, la predizione dell'U-net ha manifestato la capacità di **esaltare** le caratteristiche intrinseche dell'impronta digitale, incrementando il contrasto tra le linee dell'impronta e lo sfondo.

6.3 Blur

	SeConvNet	DAE	RDN	UNet
MSE	2.881e-2	5.994e-03	4.351e-03	4.284e-03
PSNR	17.5694	24.5174	25.6795	26.3922
SSIM	0.5327	0.8803	0.9067	0.9153

Figura 6.13: Risultati Blur

Il Blur rappresenta un altro tipo di rumore, completamente diverso rispetto al SAP: l'esperimento mette alla prova l'adattabilità dei modelli ai vari contesti rumorosi.

6.3.1 Tensorboard

Per garantire la chiarezza e la leggibilità della presentazione dei risultati, si è scelto di continuare a focalizzare l'attenzione sulle architetture DAE, RDN e U-net.

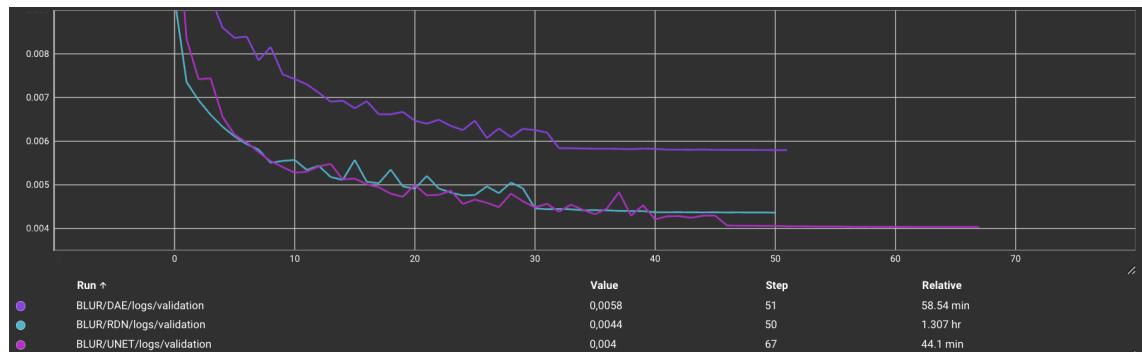


Figura 6.14: Validation loss (MSE) - Blur



Figura 6.15: PSNR - Blur

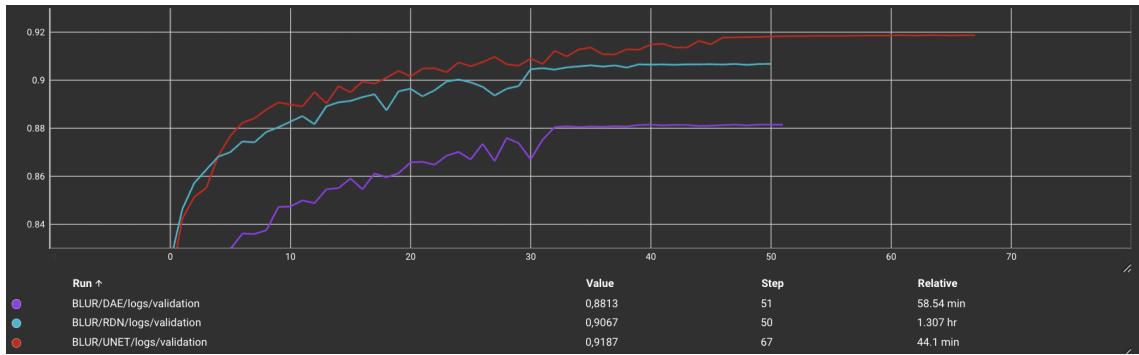


Figura 6.16: SSIM - Blur

L'esperimento di denoising focalizzato sulla rimozione del rumore di tipo blur dalle immagini ha portato a risultati e osservazioni intriganti. Contrariamente a quanto si potrebbe supporre, data la natura visivamente più evidente e perturbante del rumore SAP - 50%, il trattamento del rumore blur ha registrato valori delle metriche lievemente inferiori e ha richiesto un impegno computazionale maggiore del precedente esperimento, sia in termini di tempo che di numero di epoche di addestramento.

La complessità nell'affrontare il rumore blur rispetto al SAP può essere attribuita alle sue caratteristiche intrinseche, che tendono a **diffondere** le informazioni attraverso l'immagine, rendendo più sfidante per le reti neurali l'identificazione e la ricostruzione delle caratteristiche originarie.

Nello specifico, il *Denoising Autoencoder (DAE)* non ha dimostrato le stesse prestazioni superiori osservate nell'esperimento con il rumore SAP. Questa volta, la sua capacità di generalizzare e ripulire efficacemente le immagini blur è stata superata da quella delle altre due architetture, evidenziando come la specificità del tipo di rumore possa influenzare significativamente l'efficacia di un modello.

Sorprendentemente, la *Residual Dense Network* (**RDN**) ha superato le aspettative, estendendo il suo addestramento fino alla 50esima epoca, un netto allungamento rispetto ai precedenti punti d'arresto. Questo allungamento delle epoche segnala un adattamento dell'architettura RDN alle sfide poste dal rumore blur.

U-net, tuttavia, è emersa come la chiara vincitrice di questo round di sperimentazione, spingendosi fino alla 67esima epoca e mostrando la migliore performance complessiva nel trattamento del rumore blur.

6.3.2 Predictions

Di seguito, una dimostrazione grafica di previsione (*prediction*) per l'esperimento "Blur".

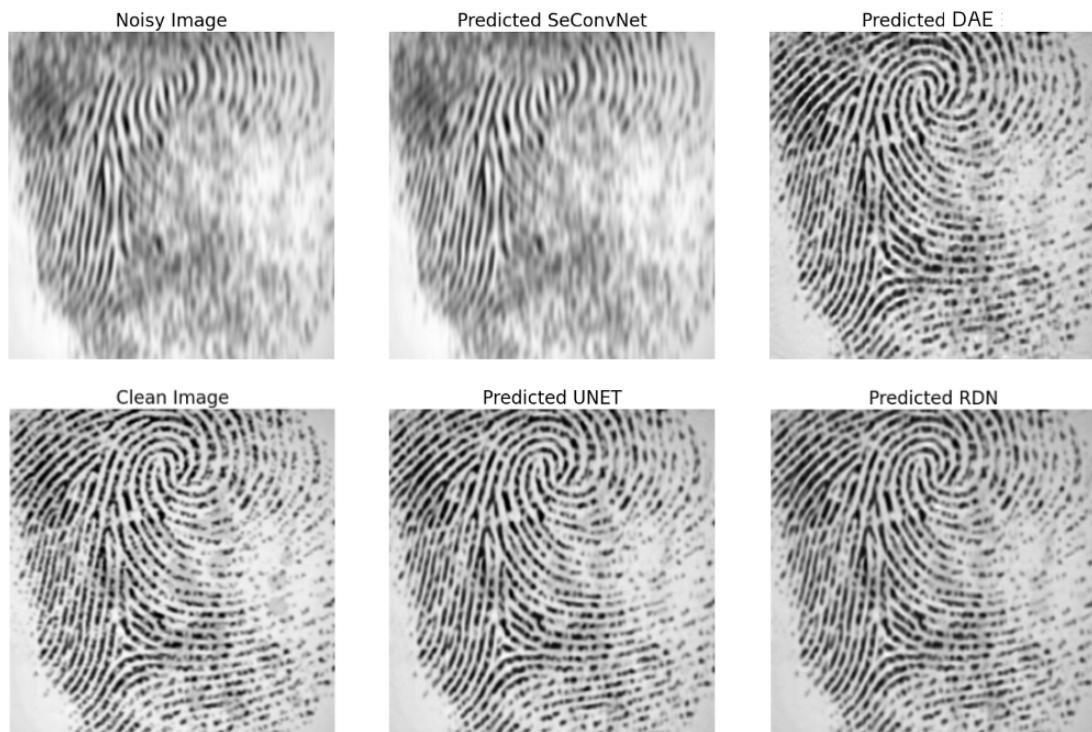


Figura 6.17: Prediction di un campione casuale nel test set - Blur

Segue un dettaglio sulla prediction della **U-net** e della **RDN**.



Figura 6.18: Dettaglio prediction U-net e RDN - Blur

6.4 Mixed (Salt and Pepper - 20% + Blur)

Il rumore "Mixed" ha posto la sfida più complessa per le architetture proposte. Sebbene le prestazioni abbiano risentito di questa complessità, rivelando un margine di miglioramento, i risultati ottenuti riflettono la potenzialità di evoluzione dei sistemi di denoising.

	SeConvNet	DAE	RDN	UNet
MSE	3.794e-02	1.185e-02	1.085e-02	1.016e-02
PSNR	15.5384	20.5199	20.7167	21.2819
SSIM	0.3017	0.7357	0.7499	0.7783

Figura 6.19: Risultati Mixed

6.4.1 Tensorboard

Per garantire la chiarezza e la leggibilità della presentazione dei risultati, si è scelto di continuare a focalizzare l'attenzione sulle architetture DAE, RDN e U-net.

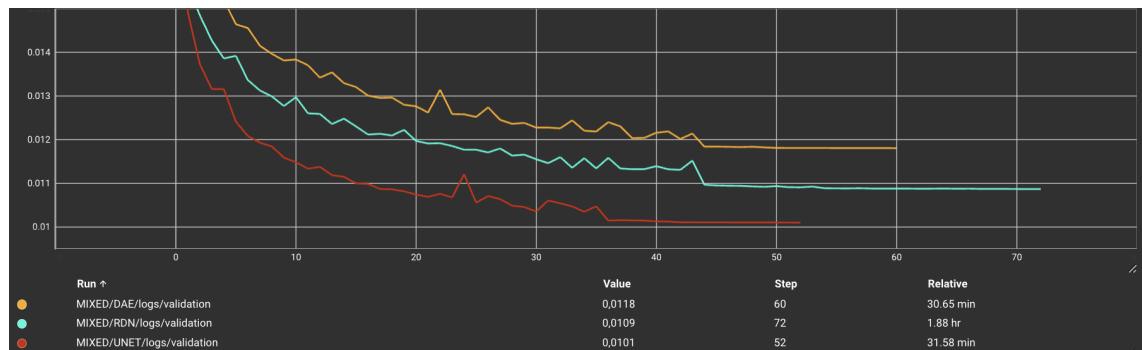


Figura 6.20: Validation loss (MSE) - Mixed



Figura 6.21: PSNR - Mixed

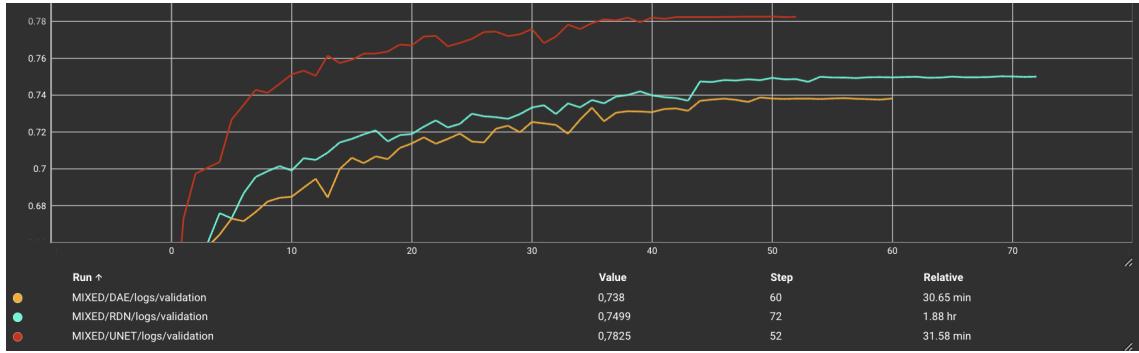


Figura 6.22: SSIM - Mixed

L'affaticamento dei modelli, evidenziato dalla riduzione delle performance nelle metriche analizzate, suggerisce che la combinazione di blur e SAP al 20% rappresenti un ostacolo notevolmente maggiore per l'apprendimento, rispetto alla presenza di un singolo tipo di rumore. Questo scenario di rumore mixed richiede dalle reti una maggiore capacità di generalizzazione e una più accurata estrazione delle caratteristiche rilevanti per il recupero delle informazioni visive originali.

Contrariamente agli esperimenti precedenti, l'**RDN** ha mostrato un impegno temporale considerevole, con quasi due ore necessarie per completare l'addestramento, riflettendo forse la maggiore complessità nel gestire simultaneamente i due tipi di rumore. Nonostante ciò, l'RDN ha proseguito l'addestramento fino alla 72esima epoca, superando in durata sia il **DAE** (60 epoche) sia l'**U-net** (52 epoche). Questa estensione delle epoche, insolita rispetto agli altri esperimenti, indica un tentativo del modello di adattarsi a un contesto di apprendimento particolarmente esigente.

U-net ha dimostrato una notevole superiorità, non solo in termini di metriche di performance ma anche di velocità di convergenza. Tale risultato sottolinea l'efficacia dell'architettura U-net nell'affrontare situazioni di rumore complesso, probabilmente grazie alla sua capacità di preservare le informazioni spaziali critiche attraverso le connessioni di salto.

6.4.2 Predictions

Di seguito, una dimostrazione grafica di previsione (*prediction*) per l'esperimento "Mixed".

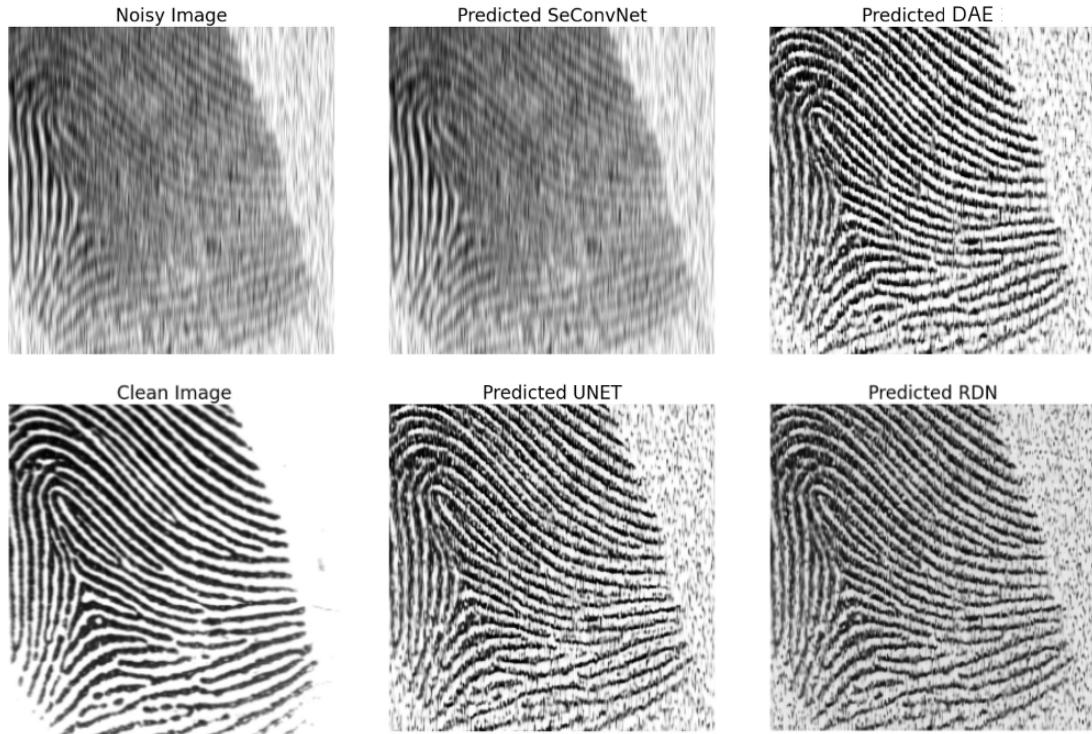


Figura 6.23: Prediction di un campione casuale nel test set - Mixed

Segue un dettaglio sulla prediction della **U-net**.

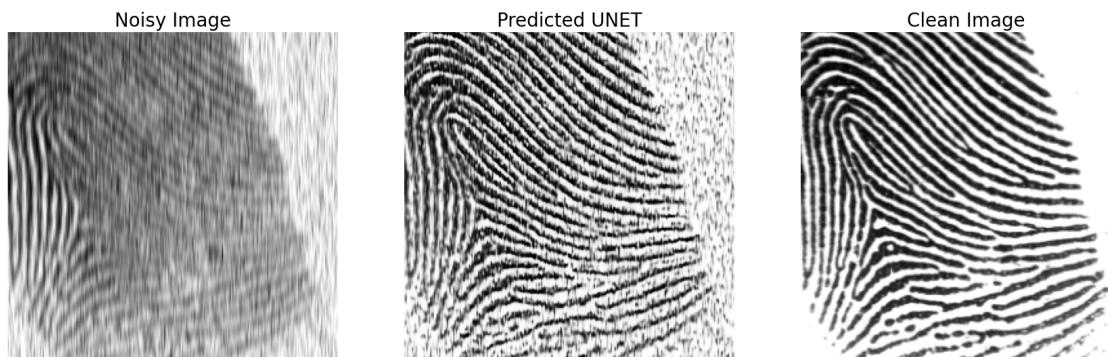


Figura 6.24: Dettaglio prediction U-net - Mixed

Sebbene sia palpabile la divergenza tra l'immagine originale priva di disturbi e quella ricostruita dal modello, l'architettura U-net emerge con particolare rilievo, evidenziando potenziali ambiti di ulteriore indagine nel complesso panorama del denoising, specialmente in presenza di rumori misti. Tale os-

servazione si fonda sul riconoscimento che, nonostante le metriche di valutazione attestino prestazioni piuttosto soddisfacenti, le immagini ricostruite dal modello non presentano una pulizia completa.

Capitolo 7

Conclusione

La ricerca ha dimostrato come gli approcci per affrontare il problema del denoising non siano universali ma, piuttosto, specifici al tipo di distorsione presente nell'immagine. Per il rumore SAP al 5%, sebbene tutte le architetture abbiano ottenuto risultati promettenti, la SeConvNet si è distinta per una performance meno competitiva. Questo risultato sottolinea l'importanza di adattare e ottimizzare le reti neurali al contesto specifico del rumore da trattare. D'altra parte, per ciò che riguarda il rumore SAP al 50% e il rumore blur, U-net e DAE hanno dimostrato una notevole capacità di ricostruzione, testimoniando l'efficacia di queste architetture nel ripristinare l'integrità delle immagini compromesse.

La RDN ha mostrato una performance contrastante a seconda dell'intensità del rumore: mentre da un lato ha riscontrato difficoltà nell'affrontare scenari di rumore ad alta intensità, dall'altro ha invece eccelso in contesti caratterizzati da livelli di rumore più contenuti, dove ha raggiunto risultati notevolmente migliori, evidenziando la sua efficacia nel denoising a bassa intensità.

Il contesto di rumore mixed ha presentato le sfide più complesse, mettendo in evidenza l'affaticamento dei modelli. Nonostante ciò, **U-net** ha mostrato una superiorità assoluta, emergendo come l'architettura più resiliente e capace di adattarsi a scenari di rumore complesso, offrendo spunti per future indagini.

L'analisi dei risultati pone le basi per un'esplorazione più approfondita delle potenzialità e dei limiti di queste architetture.

Sfide Emergenti e Possibili Direzioni Future

Riduzione dei Costi Computazionali

Mentre l'efficacia del denoising è stata dimostrata, l'ottimizzazione delle risorse computazionali rimane una sfida critica. Esplorare tecniche come la **pruning** dei modelli (alcuni già esposti nel lavoro di Li et al. "*Deep Learning Based Method for Pruning Deep Neural Networks*" [13]), l'efficiente progettazione delle reti e l'uso di hardware specializzato può contribuire a ridurre significativamente i costi computazionali.

Miglioramento della Generalizzazione

I modelli attuali mostrano una forte capacità nel trattare specifici tipi di rumore, ma la generalizzazione a rumori non visti o complessi rimane una sfida. L'introduzione di metodologie di apprendimento trasferibile, dove i modelli pre-addestrati su vasti dataset possono essere ritoccati (**fine-tuning**) per applicazioni specifiche, potrebbe migliorare significativamente la generalizzazione dei modelli.

Tecnologie di Acquisizione Avanzate

La collaborazione con le ultime tecnologie di acquisizione di immagini potrebbe non solo migliorare la qualità iniziale delle impronte digitali ma anche facilitare il compito dei modelli di denoising. L'integrazione di **sensori avanzati** con algoritmi di **preprocessing in tempo reale** potrebbe ridurre il livello di rumore prima che le immagini raggiungano la fase di denoising, migliorando l'efficacia complessiva del processo.

Sviluppo di Soluzioni Pervasive

L'adattamento degli algoritmi per l'esecuzione su dispositivi mobili e sistemi embedded apre la strada all'adozione di soluzioni biometriche pervasive. Ottimizzare i modelli per garantire la privacy e la sicurezza dei dati su questi dispositivi rappresenta una direzione promettente per la ricerca futura.

Bibliografia

- [1] Nikolas Adaloglou. «Intuitive explanation of skip connections in deep learning». In: *AI Summer* 23 (2020) (cit. a p. 8).
- [2] Mahmoud Farhang Ahmad Ali Rafiee. "A deep convolutional neural network for salt-and-pepper noise removal using selective convolutional blocks". 2023. URL: <https://doi.org/10.1016/j.asoc.2023.110535> (cit. a p. 5).
- [3] Bee Lim et al. "Enhanced Deep Residual Networks for Single Image Super-Resolution". 2017. URL: <https://doi.org/10.48550/arXiv.1707.02921> (cit. a p. 6).
- [4] Barry T. Bosworth et al. «Estimating Signal-to-Noise Ratio (SNR)». In: *IEEE Journal of Oceanic Engineering* 33.4 (2008), pp. 414–418. DOI: [10.1109/JOE.2008.2001780](https://doi.org/10.1109/JOE.2008.2001780) (cit. a p. 5).
- [5] Joseph Chang Christian Darken e John Moody. "Learning Rate Schedules For Faster Stochastic Gradient Search". 1992. URL: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9db554243d7588589569aea127d676c9644d069a> (cit. a p. 38).
- [6] Paavani Dua. "Image Denoising Using a U-net". 2019. URL: https://stanford.edu/class/ee367/Winter2019/dua_report.pdf (cit. alle pp. 7, 8).
- [7] Fan H. et al. Fan L. Zhang F. "Brief review of image denoising techniques". 2019. URL: <https://doi.org/10.1186/s42492-019-0016-7> (cit. a p. 4).
- [8] Gabriela Ghimpețeanu et al. «A Decomposition Framework for Image Denoising Algorithms». In: *IEEE Transactions on Image Processing* 25.1 (2016), pp. 388–399. DOI: [10.1109/TIP.2015.2498413](https://doi.org/10.1109/TIP.2015.2498413) (cit. a p. 3).
- [9] Bhawna Goyal et al. «Image denoising review: From classical to state-of-the-art approaches». In: *Information Fusion* 55 (2020), pp. 220–244. ISSN: 1566-2535. DOI: <https://doi.org/10.1016/j.inffus.2019.09.003>. URL: <https://www.sciencedirect.com/science/article/pii/S1566253519301861> (cit. a p. 4).

- [10] Ilesanmi T. Ilesanmi A. "Methods for image denoising using convolutional neural network: a review. *Complex Intell. Syst.* 7, 2179–2198 (2021)". 2021. URL: <https://doi.org/10.1007/s40747-021-00428-4> (cit. a p. 4).
- [11] Xueying Xie et al. Jun Wang. "Denoising Autoencoder, A Deep Learning Algorithm, Aids the Identification of A Novel Molecular Signature of Lung Adenocarcinoma". 2019. URL: <https://doi.org/10.1016/j.gpb.2019.02.003> (cit. a p. 9).
- [12] Dai-Gyoung Kim et al. «Hybrid Deep Learning Framework for Reduction of Mixed Noise via Low Rank Noise Estimation». In: *IEEE Access* 10 (2022), pp. 46738–46752. DOI: 10.1109/ACCESS.2022.3170490 (cit. a p. 4).
- [13] Lianqiang Li, Jie Zhu e Ming-Ting Sun. «Deep Learning Based Method for Pruning Deep Neural Networks». In: *2019 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. 2019, pp. 312–317. DOI: 10.1109/ICMEW.2019.00-68 (cit. a p. 56).
- [14] Asavaron Limshuebchuey, Rakkrit Duangsoithong e Mongkol Saejia. «Comparison of Image Denoising using Traditional Filter and Deep Learning Methods». In: *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2020, pp. 193–196. DOI: 10.1109/ECTICON49241.2020.9158242 (cit. a p. 4).
- [15] Omar M. Saad e Yangkang Chen. «Deep Denoising Autoencoder for Seismic Random Noise Attenuation». In: *Geophysics* 85 (apr. 2020). DOI: 10.1190/geo2019-0468.1 (cit. a p. 9).
- [16] Hana Mechria, Khaled Hassine e Mohamed Salah Gouider. «Effect of Denoising on Performance of Deep Convolutional Neural Network For Mammogram Images Classification». In: *Procedia Computer Science* 207 (2022), pp. 2345–2352. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2022.09.293>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050922011814> (cit. a p. 3).
- [17] Sameera V. Mohd Sagheer e Sudhish N. George. «A review on medical image denoising algorithms». In: *Biomedical Signal Processing and Control* 61 (2020), p. 102036. ISSN: 1746-8094. DOI: <https://doi.org/10.1016/j.bspc.2020.102036>.

- 1016/j.bspc.2020.102036. URL: <https://www.sciencedirect.com/science/article/pii/S1746809420301920> (cit. a p. 4).
- [18] Chunwei Tian et al. "Deep learning on image denoising: An overview". 2020. URL: <https://doi.org/10.1016/j.neunet.2020.07.025> (cit. a p. 4).
- [19] UniBo. "Fingerprint Verification Competition". 2004. URL: <http://bias.csr.unibo.it/fvc2004/> (cit. a p. 11).
- [20] Zhou Wang et al. «Image quality assessment: from error visibility to structural similarity». In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: [10.1109/TIP.2003.819861](https://doi.org/10.1109/TIP.2003.819861) (cit. a p. 5).
- [21] Yapeng Tian et al. Yulun Zhang. "Residual Dense Network for Image Restoration". 2020. URL: <https://doi.org/10.48550/arXiv.1812.10477> (cit. alle pp. 6, 7).
- [22] Kai Zhang et al. «Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising». In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155. DOI: [10.1109/TIP.2017.2662206](https://doi.org/10.1109/TIP.2017.2662206) (cit. alle pp. 4, 5).
- [23] Mo Zhao et al. «Hybrid Transformer-CNN for Real Image Denoising». In: *IEEE Signal Processing Letters* 29 (2022), pp. 1252–1256. DOI: [10.1109/LSP.2022.3176486](https://doi.org/10.1109/LSP.2022.3176486) (cit. a p. 24).