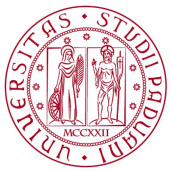


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"IMPIEGO DI GAN NELLA COMPRESSIONE DI IMMAGINI"

Relatore: Prof. Marco Cagnazzo

Laureando: Marco Antonello

ANNO ACCADEMICO 2022 – 2023

Data di laurea 16/11/2023

INDICE

INTRODUZIONE

L'ottimizzazione nel settore della compressione dei dati sta diventando sempre più di cruciale importanza. In un mondo in cui lo streaming di media digitali costituisce l'80% del traffico di Internet [1], riuscire a individuare tecniche di compressione sempre più efficaci è essenziale.
Per quanto riguarda la compressione di immagini le strategie di compressione convenzionali, quali ad esempio JPEG, WebP o BPG, non sono in grado di adattarsi al contenuto dell'immagine e quindi non permettorno il miglior livello possibile di compressione.
Oltre a ciò, quando il fattore di compressione aumenta, l'output di queste tecniche tende a degradare rapidamente (comparsa di artefatti a forma di blocco, sfocature, distorsioni di vario genere) rendendo le immagini compresse sgradevoli all'occhio umano e spesso compromettendo il corretto funzionamento di eventuali algoritmi di riconoscimento.

Il crescente successo riscosso dalle reti neurali nei campi della classificazione di immagini, dell'object detection, della segmentazione semantica e della super-resolution ha spinto i ricercatori ad ideare nuove architetture di compressione basate sul deep learning e sulla sua capacità di features extraction. Nel 2006 Hinton e Salakhutdinov [2] dimostrarono come sfruttare le capacità di features extraction di una Convolutional Neural Network (CNN) nell'ambito della riduzione di dimensionalità di un'immagine fosse preferibile rispetto all'applicazione della Principal Component Analysis (PCA) sulla stessa.
Le nuove tecniche basate sull'IA hanno il potenziale per superare gli svantaggi dei metodi convenzionali quali la perdita di qualità, la distorsione e la difficoltà di ripristino dell'immagine. Queste strategie si basano sull'addestramento di una rete neurale che ha il compito di imparare le proprietà statistiche del dataset di immagini su cui è addestrata. Terminato l'addestramento, la rete neurale può essere utilizzata per comprimere le immagini in modo da preservarne la qualità e ottenere un elevato rapporto di compressione.
Molti dei metodi di compressione basati sull'intelligenza artificiale sono ancora in fase di sviluppo, ma hanno il potenziale per rivoluzionare il modo in cui le immagini vengono compresse e archiviate.

Questa esposizione intende esaminare le varie tipologie di reti neurali utilizzate al giorno d'oggi nel settore della compressione di immagini e di valutarne i vantaggi e gli svantaggi rispetto ai codec tradizionali. Si porrà particolare attenzione ai progetti di ricerca incentrati sull'uso delle Generative Adversarial Networks (GAN).

CAPITOLO 1

Tipologie di reti neurali utilizzate nella compressione di immagini

Grazie alla loro capacità di imparare e sfruttare le proprietà statistiche dei dataset su cui sono addestrate, le reti neurali garantiscono diversi vantaggi rispetto ai metodi di compressione tradizionali:

- rapporto di compressione più elevato: reso possibile dall'utilizzo di modelli di predizione più sofisticati e dal fatto che è possibile ottimizzare la compressione per specifiche tipologie di immagini (es. volti umani, mappe satellitari, ecc.);
- migliore qualità dell'immagine, soprattutto per le immagini dettagliate o ad alta risoluzione;
- riduzione della distorsione: solitamente le reti neurali sono addestrate utilizzando loss function che tengono in considerazione il trade-off tra bitrate e distorsione delle immagini;

Di contro i sistemi di compressione basati sull'IA:

- richiedono grandi dataset di immagini per il loro addestramento;
- hanno in genere un costo computazionale maggiore;
- potrebbero portare a risultati non adeguati o inattesi: specie quelli basati su heatmap o mappe semantiche che generano da zero le zone dell'immagine non appartenenti alla ROI (Region of Interest).

I più recenti metodi per la compressione lossy di immagini si basano sulla codifica a trasformazione. In questo approccio la compressione dell'immagine si ottiene prima mappando i dati dei pixel in una rappresentazione latente, poi quantizzando tale rappresentazione e infine comprimendola in maniera lossless tramite codifiche entropiche (codifica aritmetica, di Huffman, ecc.). Tipicamente ciò che permette tale trasformazione è l'utilizzo di una CNN, la quale impara ad approssimare funzioni non lineari con la possibilità di mappare i pixel in uno spazio latente più comprimibile rispetto a quello ottenibile dalle trasformazioni lineari che caratterizzano i codec tradizionali.

1.1 CNN E RNN

Le CNN sono particolarmente adatte ad essere impiegate nell'elaborazione di immagini. Ciò che le rende così ideali in questo frangente è la loro capacità di apprendere relazioni spaziali anche complesse all'interno delle immagini stesse. Questo è reso possibile dalla bi-dimensionalità dei kernel utilizzati nelle convoluzioni (stesso numero di dimensioni delle immagini), dalla possibilità di analizzare più features locali simultaneamente e dal fatto che i kernel vengono appresi direttamente dai dataset di immagini invece di venire progettati manualmente, rendendoli di fatto uno strumento più adatto all'elaborazione di immagini rispetto alle trasformate tradizionali.

Un'altra categoria di reti neurali che può essere sfruttata nel processo di compressione di immagini è quella delle Recurrent Neural Networks (RNN). Queste reti hanno reso possibile la compressione a bitrate variabile delle immagini, ma trascinano con sè un costo computazionale molto grande.
A volte le RNN vengono usate assieme alle CNN e formano le Recurrent Convolutional Neural Networks.

Nella Figura 1 la stessa immagine è compressa a 0,4bpp con JPEG2000 (a sinistra) e tramite CNN e trasformata iWave [3] (a destra). La trasformata iWave è una trasformata wavelet-like che sfrutta le capacità delle CNN.

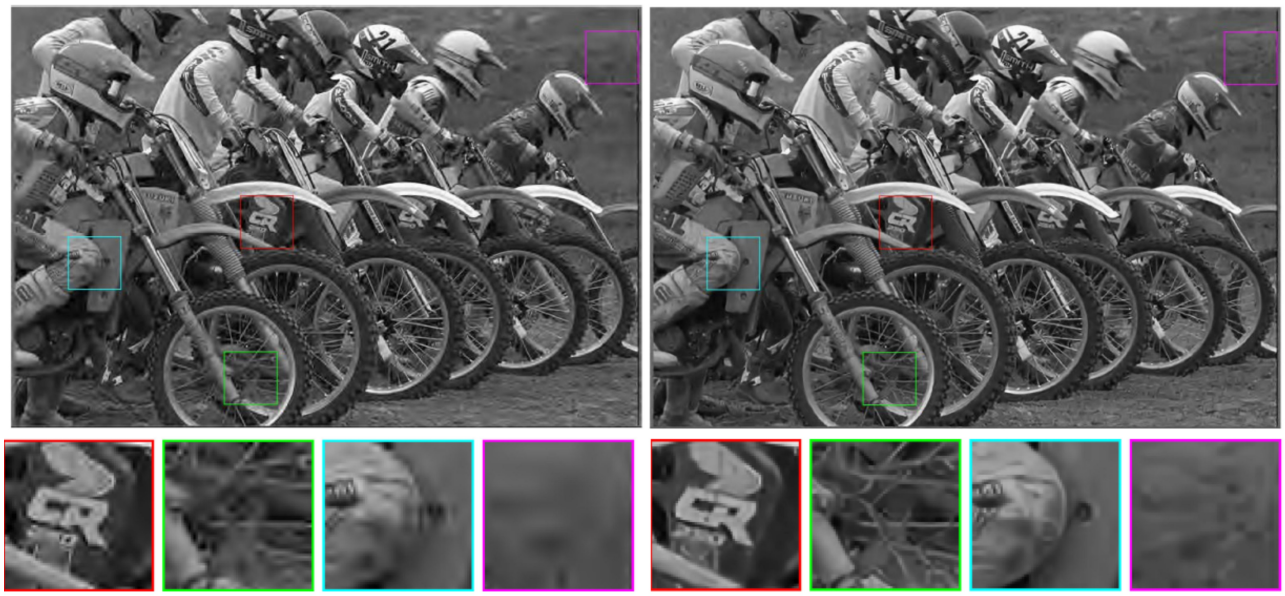


Figura 1: JPEG200 (sinista) messo a confronto con la trasformata iWave [3] (destra)

1.2 AUTOENCODER

Un autoencoder (AE) è una rete neurale che, tramite addestramento non supervisionato, apprende la struttura/distribuzione nascosta (non nota a priori) che caratterizza il dataset su cui è addestrato.
Una volta eseguito l'addestramento, l'AE consente di mappare le immagini in ingresso in uno spazio latente contraddistinto da un numero minore di dimensioni. Tale spazio latente è anche detto collo di bottiglia dell'AE e l'immagine codificata prende il nome di codice. Un autoencoder è generalmente composto da due blocchi: un codificatore (E) e un decodificatore (D).
Il codificatore prende i dati in input e li comprime nella loro rappresentazione latente.
Il decodificatore prende la rappresentazione latente e cerca di ricostruire i dati di ingresso.

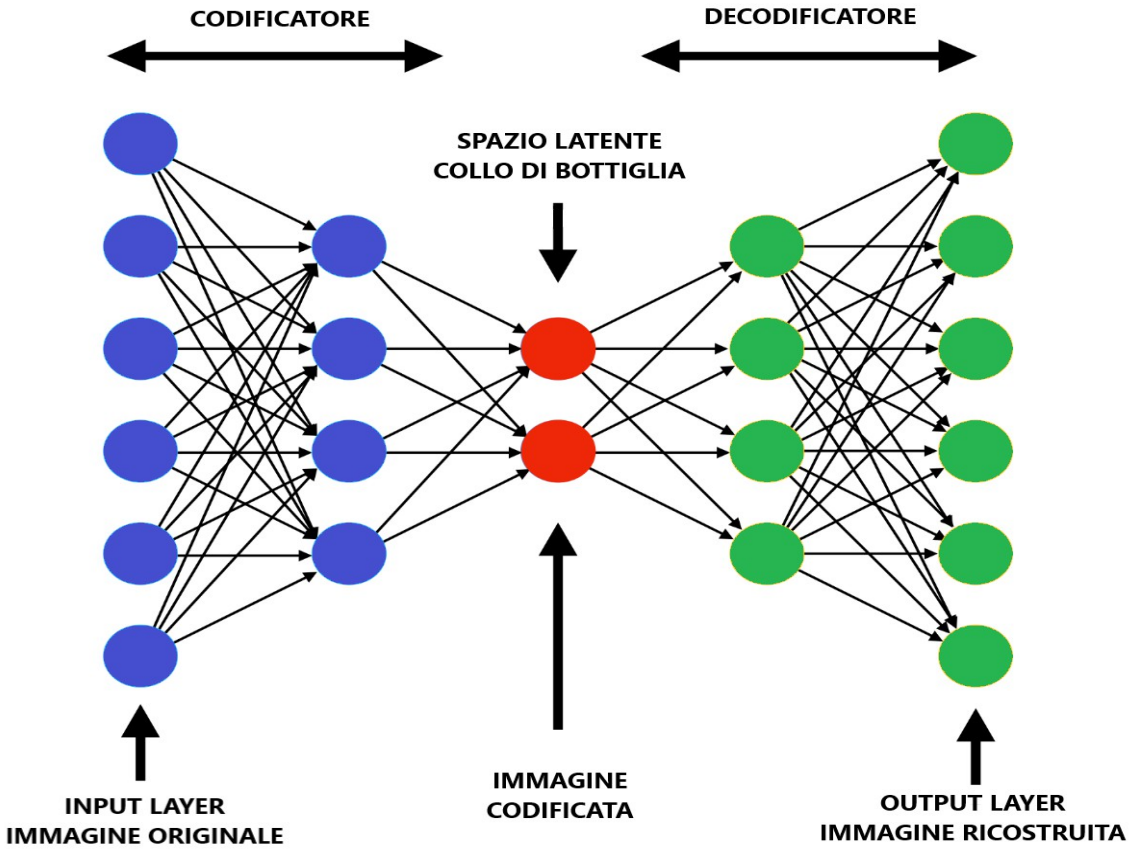


Figura 2: Architettura base di un autoencoder

Sia x l'immagine originale e x' l'immagine ricostruita avremo:
$x' = D(E(x))$

La loss-function utilizzata negli AE cerca di ridurre al minimo l'errore di ricostruzione, ovvero la differenza tra i dati in ingresso e i dati ricostruiti.
Tra le loss-functions più utilizzate troviamo:

- Mean Squared Error (MSE)

  $$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left\| (X(i,j) - Y(i,j))^2 \right\|$$

  dove X = immagine originale, Y = immagine ricostruita
  n = altezza di entrambe le immagini, m = larghezza di entrambe le immagini

- Peak Signal-to-Noise Ratio (PSNR)

  $$PSNR = 20\log\left(\frac{255}{\sqrt{MSE}}\right)$$

  Un valore più alto indica una qualità migliore

- Structural Similarity Index Measure (SSIM)

  $$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

  dove x = immagine originale, y = immagine ricostruita
  μ_x, μ_y = valori medi di x e y
  σ_x^2, σ_y^2 = varianza di x e y
  σ_{xy} = covarianza tra x e y
  c_1, c_2 = costanti utilizzate per stabilizzare la SSIM quando i valori medi e le varianze sono vicini allo zero
  Il valore finale varia tra 0 e 1; con SSIM = 1, l'immagine originale e quella ricostruita sono identiche. In genere viene calcolata su più finestre e non direttamente sull'intera immagine.

- Multi-Scale Structural Similarity Index Measure (MS-SSIM)
  A differenza della SSIM, la MS-SSIM tiene conto di più risoluzioni dell'immagine risultando una metrica più robusta alla comparsa di artefatti o distorsioni nell'immagine ricostruita. Ha lo svantaggio di essere computazionalmente più pesante della SSIM.

La SSIM e in particolare la MS-SSIM sono da preferire come metriche al MSE e al PSNR in quanto le ultime due non tengono conto degli artefatti o delle distorsioni che si possono formare durante la ricostruzione dell'immagine e quindi non riescono a misurare la qualità visiva percepita dell'immagine.

La Figura 3 rappresenta l'architettura utilizzata da Theis et al. nel suo articolo [4] riguardante gli autoencoder. Come è possibile notare, sia il codificatore che il decodificatore si identificano in una CNN.

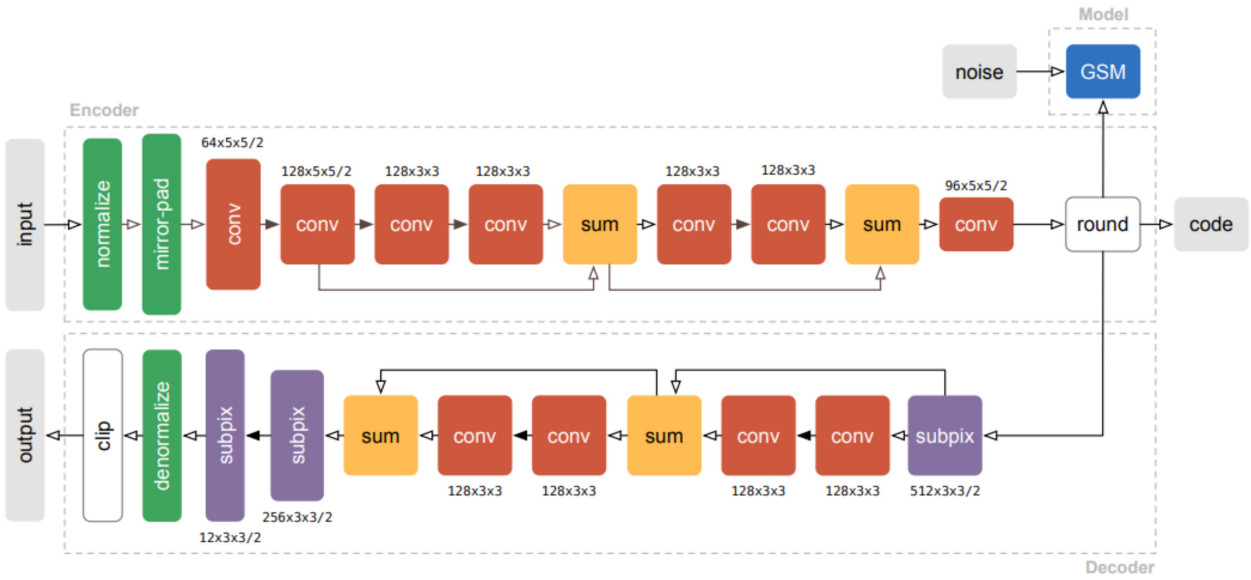


Figura 3: Architettura usata da Theis et al. nel suo articolo [4]

Nella Figura 5 viene mostrato come l'autoencoder di Theis produca ricostruzioni migliori rispetto ai metodi convenzionali (JPEG e JPEG2000).

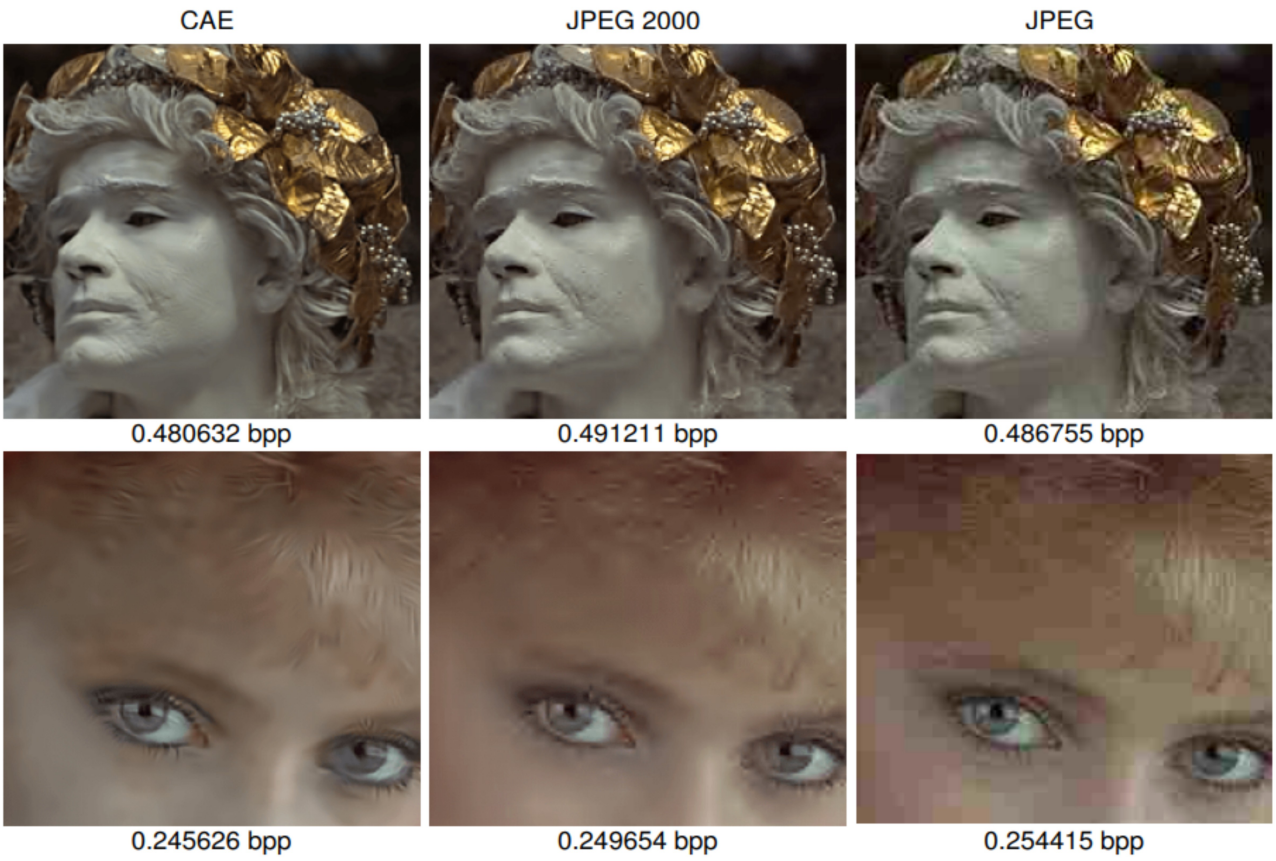


Figura 4: Architettura di Theis a confronto con JPEG e JPEG2000 [4]

1.3 VARIATIONAL AUTOENCODER

L'architettura dei variational autoencoder (VAE) deriva da quella degli autoencoder e si distingue da quest'ultima per il fatto che l'input viene mappato in una distribuzione di probabilità invece che in un codice fisso. Tale distribuzione è caratterizzata da un vettore media e da un vettore varianza e impedisce all'output di essere deterministico, inserisce cioè un elemento di casualità. Ciò si traduce nel fatto che un VAE può generare ricostruzioni diverse partendo dalla stessa immagine in input. Tra i vantaggi di questo approccio ci sono anche la possibilità di identificare le anomalie presenti nelle immagini in ingresso e la possibilità di rimuovere il rumore (denoising) da queste ultime.

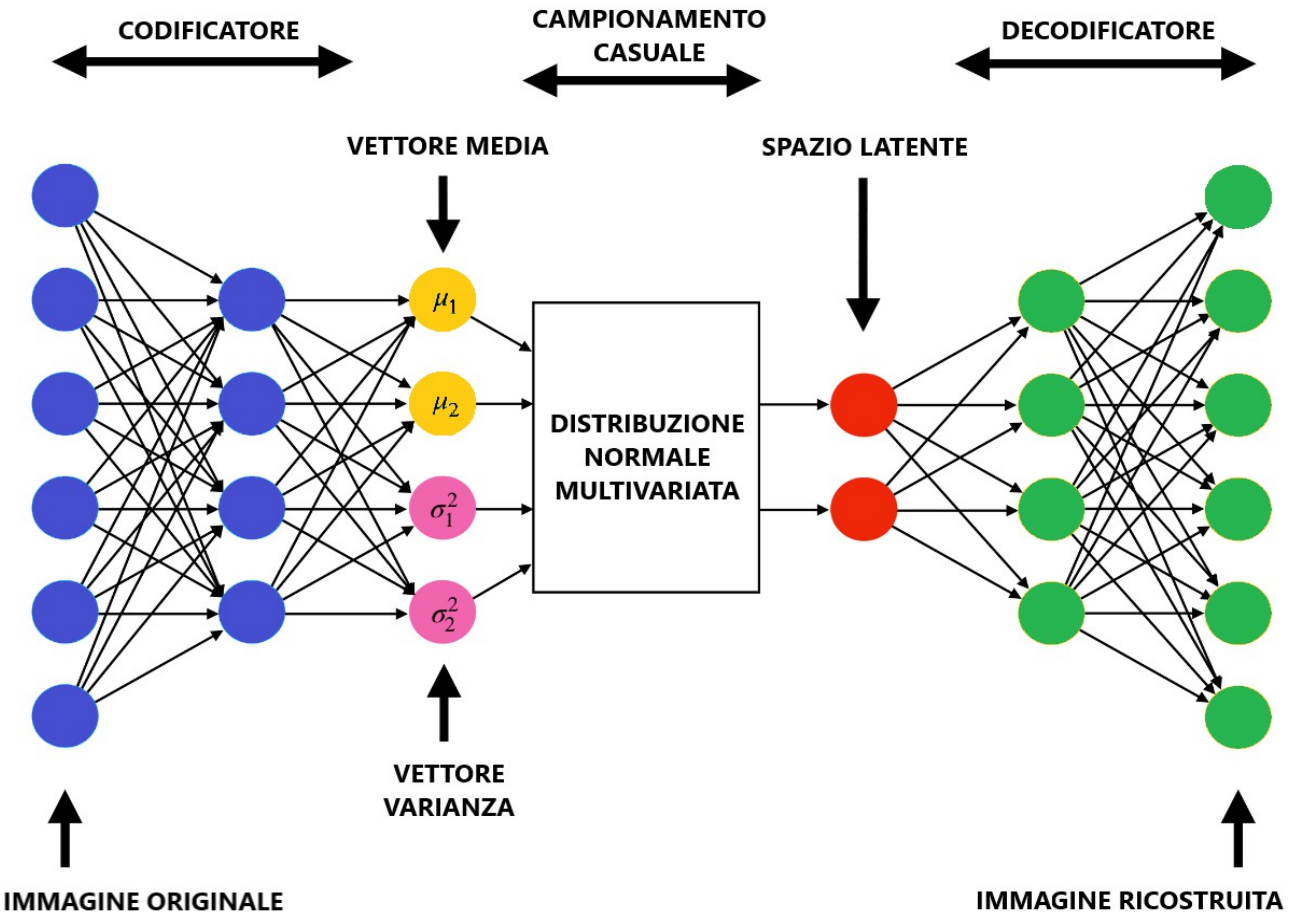


Figura 5: Architettura base di un variational autoencoder

Sia z la rappresentazione latente, il codificatore cerca di imparare la distribuzione di probabilità p(z|x), il decoder invece la distribuzione p(x|z).

La loss function dei VAE, oltre all'errore di ricostruzione, tiene conto di un termine di regolarizzazione che permette alla rappresentazione latente di seguire una distribuzione prestabilita (solitamente una distribuzione normale multivariata):

L_{VAE} = errore di ricostruzione + termine di regolarizzazione

Il termine di regolarizzazione è in genere calcolato utilizzando la divergenza di Kullback-Leibler:

$$D_{KL}(P\|Q) = \sum_i P(i)\log_2(\frac{P(i)}{Q(i)})$$

dove P = distribuzione caratterizzante lo spazio latente (p(z|x))
Q = distribuzione prestabilita (in genere una normale multivariata)
i = possibile valore della variabile aleatoria

Tale divergenza misura la differenza che esiste tra due distribuzioni di probabilità; minimizzarla significa rendere le due distribuzioni il più uguali possibile.
Seguire una distribuzione normale multivariata tramite il termine di regolarizzazione garantisce che rappresentazioni latenti vicine portino a immagini ricostruite simili (negli autoencoder normali ciò non è sempre garantito).

CAPITOLO 2

Utilizzo di GAN nella compressione di immagini

2.1 GENERATIVE ADVERSARIAL NETWORKS

Sin dalla loro introduzione nel 2014 dovuta a Ian Goodfellow et al. [5], le Generative Adversarial Networks hanno condotto a rapidi progressi nel settore della generazione di immagini sia a partire da distribuzioni condizionate che non, con alcune delle GAN odierne in grado di generare immagini foto-realistiche e ad alta risoluzione.
Oltre che alla generazione di immagini, questa nuova architettura ha portato a netti miglioramenti nei campi della rimozione di artefatti, della rimozione del rumore e della super-resolution.

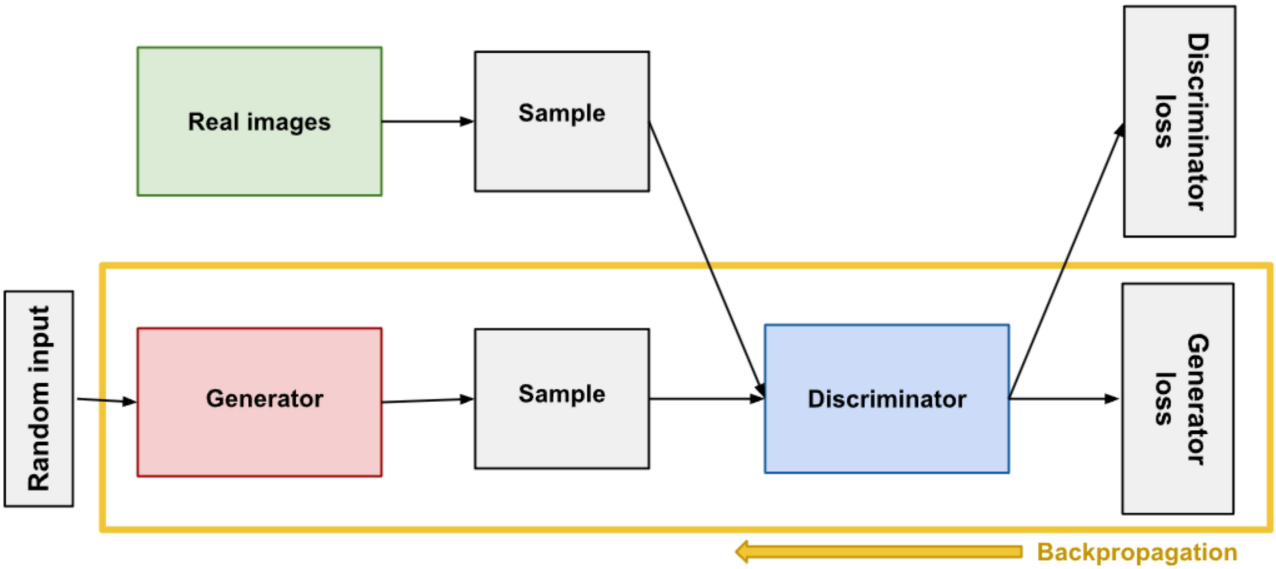


Figura 6: Architettura base di una GAN [6]

Una GAN è formata da due reti neurali:

- un generatore che impara a generare dati plausibili;
- un discriminatore che impara a distinguere i dati creati dal generatore da quelli reali

Lo scopo del generatore è quello di riuscire a imbrogliare il discriminatore tramite la creazione di campioni sempre più verosimili (è l'avversario del discriminatore).
Per conservare un equilibrio nelle capacità di entrambe le componenti, il generatore e il discriminatore vengono addestrati a fasi alterne mantenendo costante il generatore durante la fase di addestramento del discriminatore e viceversa. Questo modalità di addestramento però è la causa di uno dei problemi che caratterizzano le GAN, la loro instabilità. A differenza di una rete neurale semplice, infatti, in una GAN la convergenza è spesso uno stato di breve durata e quindi non stabile.

Una delle loss-function più utilizzate in questa architettura deriva dalla formula dell'entropia incrociata tra la distribuzione dei dati reali e la distribuzione dei dati generati ed è detta adversarial-loss:

$$\min_G \max_D L_{GAN} = E_{x\sim px}[\log(D(x))] + E_{z\sim pz}[\log(1-D(G(z)))]$$

dove D(x) = output del discriminatore quando il dato è reale (da massimizzare per il discriminatore)
D(G(z)) = output del discriminatore quando il dato è generato (da minimizzare per il discriminatore, da massimizzare per il generatore)
G(z) = output del generatore quando in input si ha il rumore z
p_x = distribuzione di probabilità dei dati reali, p_z = distribuzione di probabilità di z

2.2 GAN CONDIZIONALI

Una GAN è detta condizionale quando permette di generare immagini partendo da una distribuzione condizionata $p_{x|s}$ dove a x è associata un'informazione addizionale s (per esempio l'etichetta della classe o una mappa semantica). x e s sono inoltre legate da una distribuzione congiunta $p_{x,s}$ non nota.
Esattamente come nelle GAN normali, si addestrano due reti neurali rivali: un generatore G condizionato da s che mappa i campioni z da una distribuzione di probabilità stabilita e fissa p_z alla distribuzione $p_{x|s}$, e un discriminatore D che ha il compito di capire se un input (x, s) è stato generato da G o se proviene dalla distribuzione $p_{x|s}$.
La loss function in questo caso è:

$$\min_G \max_D L_{cGAN} = E_{x\sim p\ x|s}[\log(D(x,s))] + E_{z\sim pz}[\log(1-D(G(z,s),s))]$$

Nella pratica, per rendere l'addestramento del discriminatore più semplice ed evitare la saturazione, spesso si usano delle loss-function "non saturanti":

$$\max_G L_{cGAN} = E_{z\sim pz}[-\log(D(G(z,s),s))] \quad (2)$$

$$\max_D L_{cGAN} = E_{z\sim pz}[-\log(1-D(G(z,s),s))] + E_{x\sim p\ x|s}[-\log(D(x,s))] \quad (3)$$

La saturazione avviene quando il generatore è in grado di produrre immagini talmente realistiche che il discriminatore è sempre indotto a pensare che siano reali. Questo può portare il generatore a bloccarsi in un ottimo locale e rendere difficile l'addestramento del discriminatore.

2.3 GAN NELLA COMPRESSIONE DI IMMAGINI

Per quanto riguarda la compressione di immagini, sia X un dataset di immagini, una GAN può imparare ad approssimare la sua distribuzione p_x (non nota a priori) attraverso un generatore G(z) che cerca di mappare i campioni z da una distribuzione p_z alla distribuzione p_x. Di fatto, una volta addestrato, il generatore può sostituire il decodificatore nell'architettura di un VAE.

Il primo tentativo di incorporare una GAN nel processo di compressione di immagini è stato svolto da Rippel et al. nel suo articolo [7] del 2017.
L'architettura introdotta da Rippel prevede un autoencoder con analisi piramidale, modulo di codifica adattiva e decodificatore costituito dal generatore di una GAN.

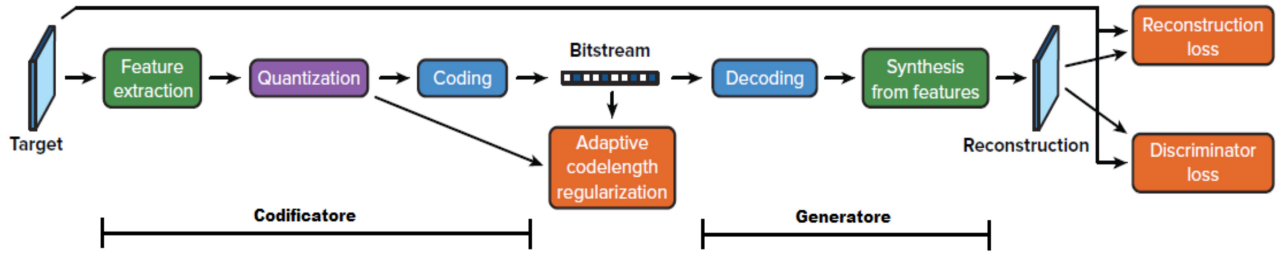


Figura 7: Architettura utilizzata in [7]

Il codificatore è costituito da una CNN che permette di eseguire una decomposizione a piramide con lo scopo di analizzare l'immagine in input a differenti livelli di dettaglio. Le features comuni ai diversi livelli vengono poi unite tramite allineamento multiscala e formano il tensore $y \in R^{CxHxW}$ con C = numero canali, H = altezza, W = larghezza.

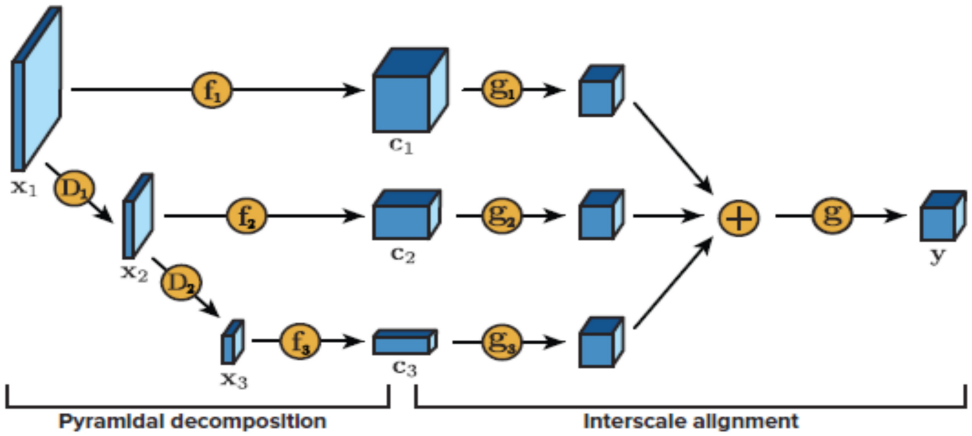


Figura 8: Esempio di decomposizione e allineamento a tre livelli nel codificatore. L'architettura reale [7] usa sei livelli

L'output y viene a questo punto quantizzato con una precisione a B bit:

$$y' = quantize_B(y)$$

Ogni canale ha così una sua rappresentazione binaria.

Ogni rappresentazione $y'_c \in R^{HxW}$ viene quindi decomposta in B piani di bit in maniera lossless.

Unendo tutti i B piani di tutti i C canali otteniamo il tensore binario:

$$b = bitplaneDecompose_B(y') \in \{0,1\}^{BxCxHxW}$$

Tale tensore viene infine codificato tramite codifica aritmetica adattiva (AAC) che permette agli input più complessi di essere mappati in codici più lunghi e agli input meno complessi in codici più corti.

Per poter ricostruire l'immagine partendo dal codice è necessario passare attraverso il decodificatore che, in questo caso, corrisponde al generatore della GAN.

Solitamente l'output del discriminatore di una GAN, durante l'addestramento, equivale a quello di un classificatore binario che prova a stabire se il dato in ingresso è un dato reale o un dato creato dal generatore. In questo caso invece al discriminatore vengono passate in ingresso sia l'immagine originale che l'immagine ricostruita. Il discriminatore dovrà quindi decidere quale delle due immagini in input è quella reale e quale quella generata. L'errore di ricostruzione utilizza come metrica il MS-SSIM.

Sostituire il normale decodificatore di un VAE con il generatore di una GAN permette di produrre ricostruzioni visivamente gradevoli (e quindi realistiche) anche a bitrate molto bassi.

I VAE infatti sono limitati dalla distribuzione di probabilità che sono in grado di generare. A bitrate bassi, la distribuzione di probabilità diventa più ristretta e ciò compromette l'abilità del VAE di generare immagini realistiche.

In Figura 9 sono presenti alcuni esempi dell'efficacia di questa architettura quando i bpp sono molto bassi (rispettivamate meno di 0,1 bpp e meno di 0,2 bpp). Il confronto è svolto con tre codec tradizionali: JPEG, JPEG2000 e WebP.

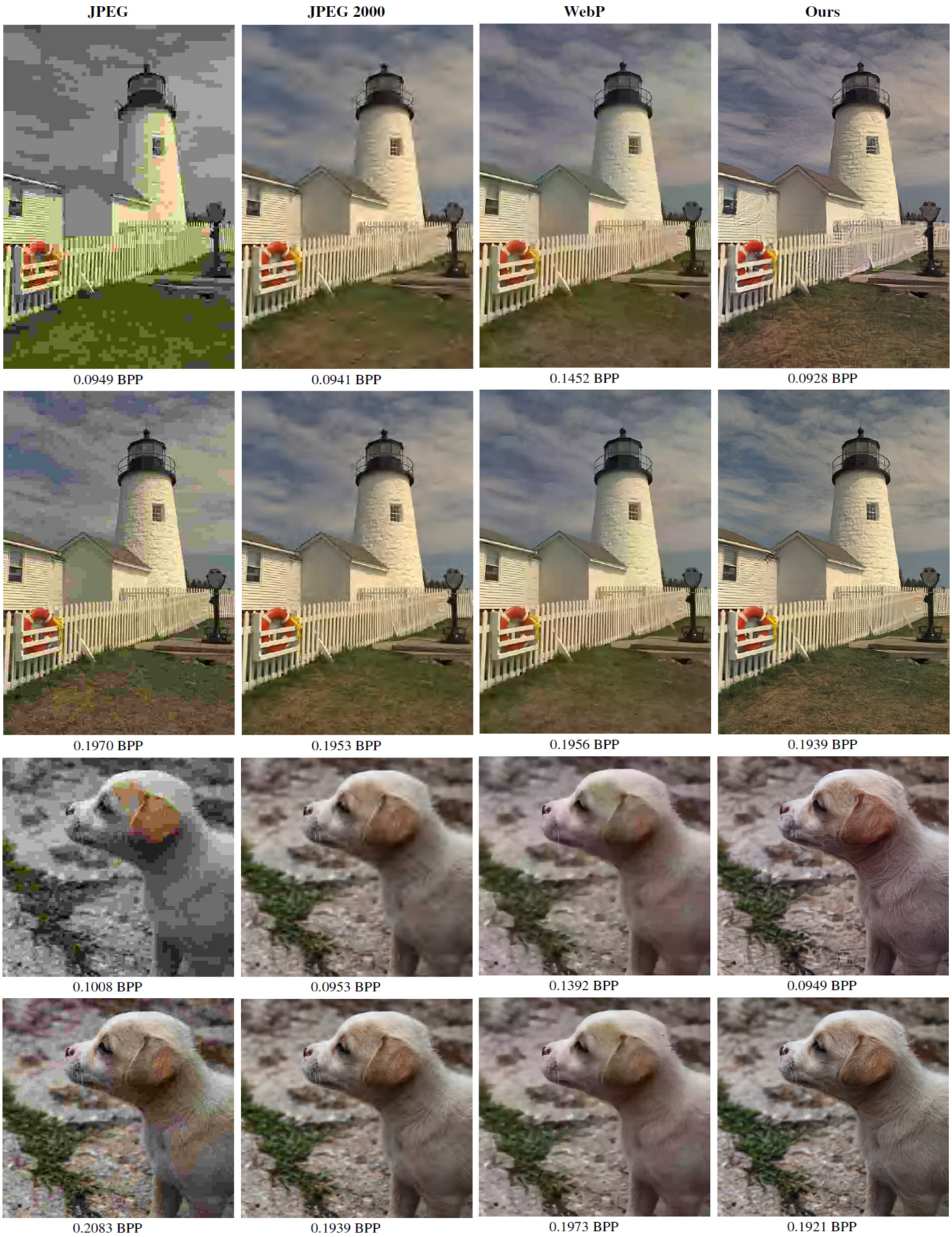


Figura 9: confronto tra JPEG, JPEG2000, WebP e l'architettura di Rippel [7]

E' facilmente riscontrabile come, ai livelli di compressione maggiore, i tradizionali metodi di compressione non reggano il confronto con l'architettura di Rippel per quanto riguarda la percezione visiva delle immagini decodificate.

Questa architettura, oltre ad ottenere buoni risultati a bitrate molto bassi, è caratterizzata dal fatto di essere leggera (per un'architettura basata su più di una rete neurale) e competitiva anche nei tempi di codifica/decodifica.
La Figura 10 espone i tempi medi di codifica e decodifica dei vari codec effettuati sul dataset RAISE-1k con immagini a risoluzione 512x768 e con MS-SSIM = 0,98.

Codec	RGB file size (kb)	YCbCr file size (kb)	Encode time (ms)	Decode time (ms)
Ours	21.4 (100%)	17.4 (100%)	8.6*	9.9*
JPEG	65.3 (304%)	43.6 (250%)	18.6	13.0
JP2	54.4 (254%)	43.8 (252%)	367.4	80.4
WebP	49.7 (232%)	37.6 (216%)	67.0	83.7

Figura 10: tempi medi di codifica e decodifica dei vari codec

** il codec è stato eseguito sulla GPU*

Per poter essere leggera e portatile l'architettura di Rippel ha dovuto risparmiare sulla complessità delle sue reti neurali. Questo ha permesso ad Agustsson et al. di presentare, nel 2018, un'architettura [8] basata su GAN che fosse, sotto l'aspetto della resa visiva delle sue ricostruzioni, ancora più performante di quella di Rippel.

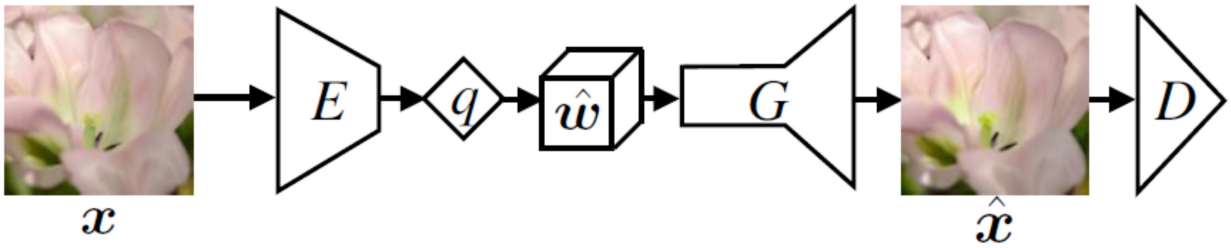


Figura 11: Architettura utilizzata da Agustsson in [8]

Nello studio di Agustsson, per comprimere un'immagine $x \in X$ vengono usati un codificatore E, un quantizzatore q e un generatore G.
Il codificatore E è una CNN che mappa l'immagine x nel vettore w appartenente allo spazio latente. Tale vettore viene in seguito quantizzato in L livelli $C = \{c_1, ..., c_L\}$ per ottenere la rappresentazione $w' = q(E(x))$
che viene poi trasformata in una sequenza di bit e codificata utilizzando la codifica aritmetica.

Il decodificatore/generatore G tenta quindi di ricostruire l'immagine partendo da w': x' = G(w')
Il numero medio di bit necessario a codificare w' è dato dall'entropia H(w').
Il trade-off tra la qualità della ricostruzione e il bitrate è dato dalla formula:

$$E[d(x, x')] + \beta\ H(w')$$

dove d = distanza che misura quanto simile x è a x'
β = peso che controlla il bitrate del modello

Siccome il numero delle dimensioni dim(w') e il numero di livelli di quantizzazione L sono finiti, l'entropia è limitata superiormente da:

$$H(w') \leqslant dim(w')\log_2(L) \qquad (1)$$

La rappresentazione w' può essere concatenata con il rumore v estratto da una distribuzione prestabilita p_v per formare il vettore latente z.
Il decodificatore/generatore G genera quindi l'immagine x' = G(z) che deve essere consistente con la distribuzione p_x e allo stesso tempo recuperare le caratteristiche dell'immagine codificata x.

Sia z = [w', v], la funzione obiettivo finale da ottimizzare è:

$$\min_{E,G}\max_{D} E[f(D(x))] + E[g(D(G(z)))] + \lambda E[d(x, G(z))] + \beta H(w')$$

con λ > 0 peso che controlla la qualità di ricostruzione
$f(y) = (y-1)^2$ (least-squares loss-function)
$g(y) = y^2$ (divergenza di Pearson tra due distribuzioni)

Di fatto la precedente può essere riscritta come la somma tra:

- la loss-function di una GAN normale
- un termine che regola la qualità di ricostruzione (e quindi la distorsione)
- un termine che regola il bitrate:

$$\min_{E,G}\max_{D}\ L_{GAN} + \lambda E[d(x, G(z))] + \beta H(w')$$

Ponendo β = 0 è comunque possibile controllare il bitrate attraverso (1)

In Figura 12 viene presentato un confronto tra immagini compresse da questa architettura e immagini compresse tramite BPG (stato dell'arte per quanto riguarda i metodi convenzionali). In entrambi i casi il fattore di compressione è molto alto (bpp < 0,05).

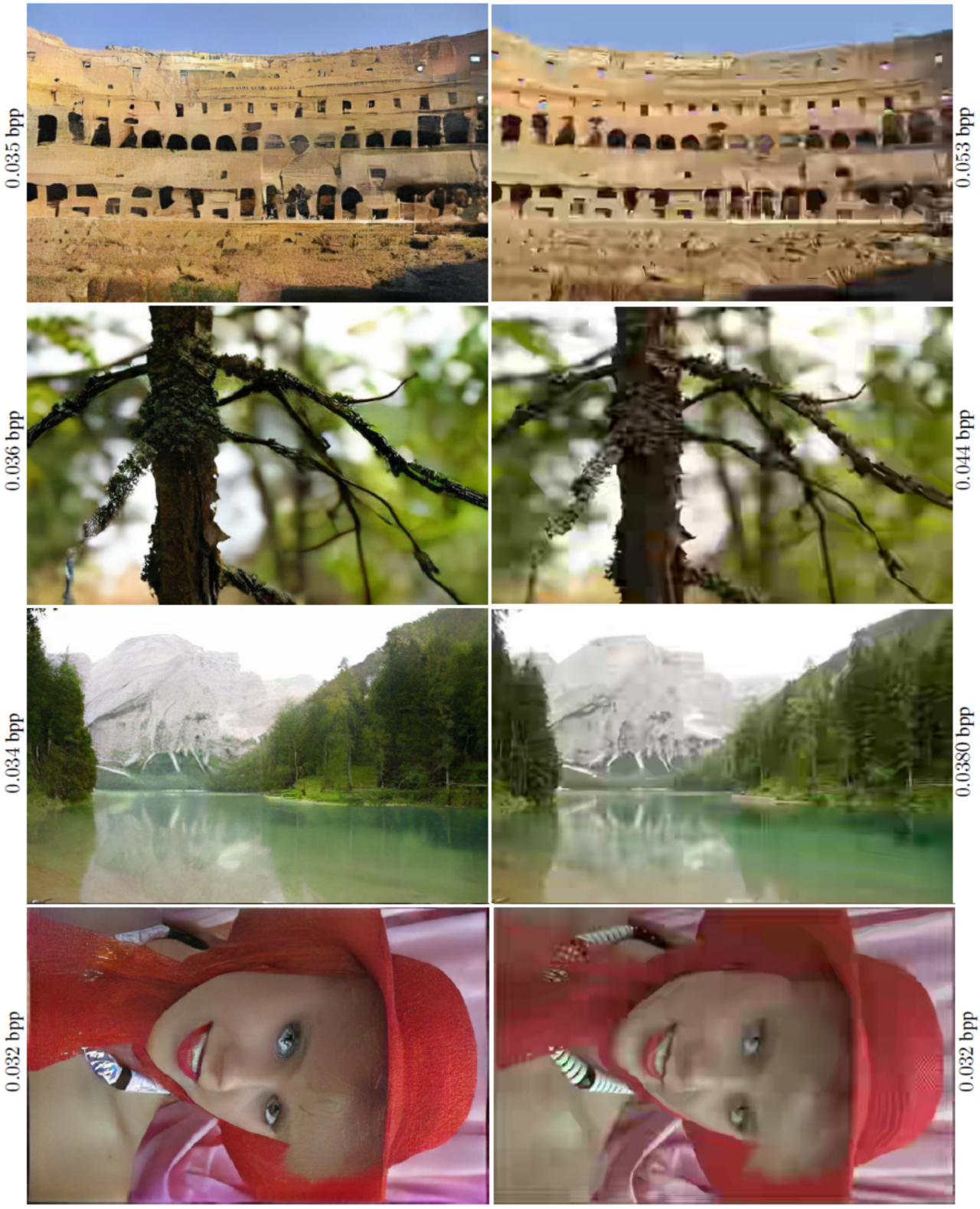


Figura 12: confronto tra l'architettura in [8] e BPG

Come è facile notare, a parità di bit per pixel, questa soluzione produce immagini molto più nitide di BPG, il quale, a questi livelli di compressione, soffre la presenza di artefatti a forma di blocco e di colori più blandi. Il modello proposto da Agustsson è in grado di ricostruire in modo convincente la texture di oggetti naturali come alberi, acqua e cielo, e si rivela efficace anche nella ricostruzione di immagini di persone.

2.4 GAN CONDIZIONALE E MAPPE SEMANTICHE

Una seconda architettura introdotta nell'articolo [8] prevede l'utilizzo di una GAN condizionale per poter lavorare con le mappe semantiche.
In quest'altra architettura, attraverso l'utilizzo di una heatmap (elemento m nella Figura 13), si indicano al generatore quali zone dell'immagine originale devono essere conservate e quali invece devono essere generate. Le zone generate dovranno seguire la mappa semantica s, la quale, prima di essere data in pasto al generatore G, deve passare attraverso un codificatore F che svolge la funzione di feature extractor e che riduce il numero di dimensioni di s mappandola nello stesso spazio latente di w'.
La distorsione d(x, x') in questo caso è calcolata solo sulle regioni che devono essere preservate. Seguendo la heatmap, G genera da zero tutte le zone da non preservare portando ad una drastica riduzione dei bit necessari a codificare la rappresentazione latente.
Chiaramente è necessario memorizzare a parte sia la heatmap m che la mappa semantica s.

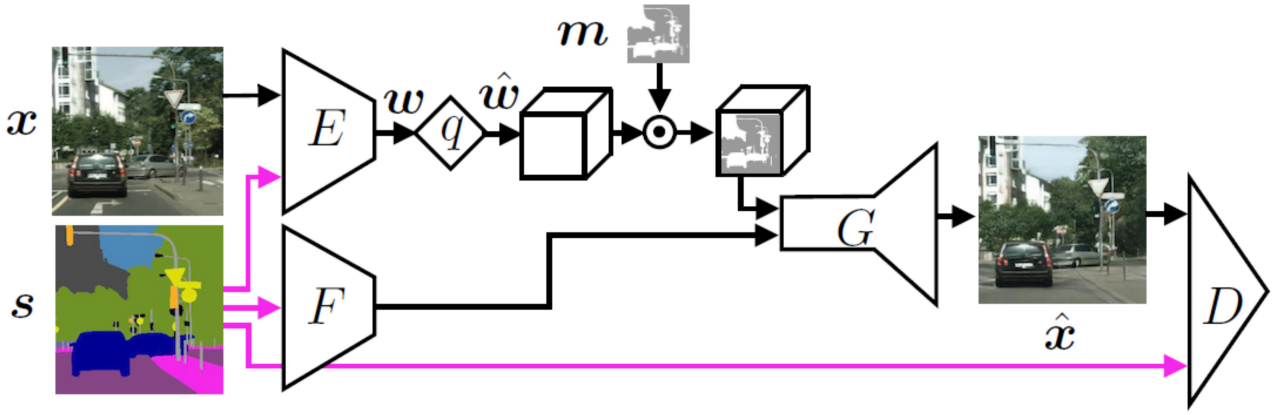


Figura 13: Architettura utilizzata in [8] che sfrutta una GAN condizionale e le mappe semantiche

Figura 14: La stessa immagine compressa utilizzando heatmap diverse [8]

Come è possibile notare, l'architettura riesce a fondere senza soluzione di continuità le zone dell'immagine conservate con quelle generate. La qualità delle ricostruzioni è alta quando ad essere sintetizzate sono strutture ripetitive (es. gli alberi, le strade, il cielo), è meno buona quando tocca ad oggetti più complessi (le persone, gli edifici).

Come sottolineato analizzando le figure precedenti, il problema di questo secondo approccio risiede nel fatto che la qualità dell'immagine, per quanto riguarda le zone generate, dipende fortemente dalla capacità del generatore di interpretare la mappa semantica.

Effettuiamo ora un confronto tra immagini ricostruite dal primo modello trattato di Agustsson e da quello di Rippel.

Figura 15: Confronto tra immagini generate dall'architettura presentata in [7] e quella presentata in [8]

Nonostante entrambe le ricostruzioni siano molto buone per i livelli di compressione trattati (bpp < 0,1), è possibile notare come i contorni degli oggetti siano in genere più definiti sulla ricostruzione di Agustsson e come le texture degli edifici siano più simili a quelle dell'immagine originale.

2.5 STATO DELL'ARTE

Nell'articolo [9] del 2020 Toderici et al. introdussero un'architettura (denominata High Fidelity Compression, HiFiC) che prometteva ricostruzioni visivamente piacevoli a diversi livelli di bitrate e applicabile anche alle alte risoluzioni. Tale architettura è basata su una GAN condizionale ed è caratterizzata da loss function che tengono conto del triplo trade-off "bitrate-distorsione-percezione". La distorsione viene calcolata attraverso una metrica di similarità (esattamente come nelle architetture già viste) mentre la qualità percepita attraverso la divergenza tra la distribuzione delle immagini reali p_x e quella delle immagini ricostruite $p_{x'}$. Questo trade-off era già stato studiato nel 2019 da Blau e Michaeli [10] i quali avevano dimostrato come, fissato il bitrate, una miglior qualità percepita comportasse sempre una maggior distorsione. Allo stesso modo cercare di minimizzare solo la distorsione portava inevitabilmente a una peggior qualità visiva.
Le loss-function uilizzate da HiFiC derivano da (2) e (3) :

$$L_{EGP} = E_{x\sim px}[\lambda r(y) + d(x,x') - \beta \log(D(x',y))]$$

$$L_D = E_{x\sim px}[-\log(1-D(x',y))] + E_{x\sim px}[-\log(D(x,y))]$$

dove y = E(x)

λ, β = iperparametri

P = modello di probabilità di y

λr(y) = termine che regola il bitrate = -λlog(P(y))

Il termine d(x, x') misura la distorsione, cioè quanto l'immagine ricostruita differisce da quella originale e in questa architettura si compone di due elementi:

$d(x, x') = k_M$ MSE + k_P d_P

con k_M e K_P iperparametri, d_P = LPIPS

LPIPS (Learned Perceptual Image Patch Similarity) è una metrica che, come SSIM, misura quanto due immagini sono diverse in termini percettivi. E' stato dimostrato che LPIPS è fortemente correlata con l'idea di similitudine che un uomo può avere di due immagini.

Figura 16: Ricostruzioni effettuate con (destra) e senza (sinistra) il termine dp

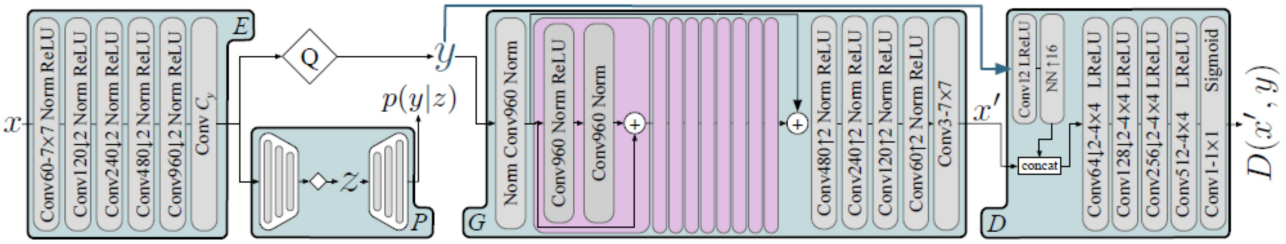


Figura 17: Architettura HiFiC

Variare il peso β significa variare la qualità percepita dell'immagine. Come già detto però, un aumento della qualità visiva comporta un aumento anche della distorsione. E' possibile osservare questo fenomeno nella Figura 18.

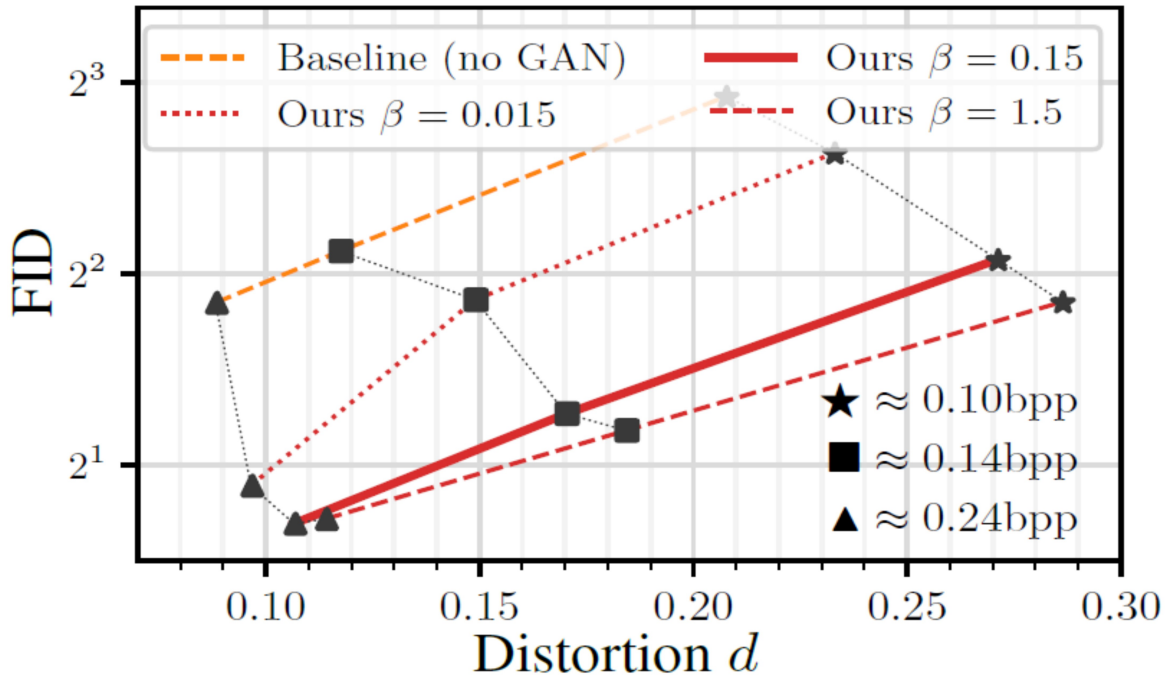


Figura 18: Rapporto tra β e la distorsione delle immagini ricostruite

Sull'asse y, FID (Fréchet Inception Distance) è una metrica che indica quanto due distribuzioni sono simili (in questo caso la distribuzione delle immagini compresse e quella delle immagini originali). Può essere usata per misurare la qualità percepita e valori minori indicano una qualità migliore. Baseline no GAN indica l'utilizzo della stessa architettura (con la stessa formula per la distorsione d), ma senza allenamento basato su GAN.
Fissando un bitrate, è possibile notare come un aumento di β e quindi della qualità visiva (diminuzione di FID) provochi un aumento della distorsione.

La Figura 19 presenta il confronto tra le ricostruzioni ottenute utilizzando HiFiC, l'architettura di Rippel e quella di Agustsson.

Figura 19: Confronto tra le ricostruzioni ottenute utilizzando [9], [7] e [8]

La ricostruzione a 0,162bpp eseguita da HiFiC è più nitida e preserva meglio le texture e i colori rispetto sia all'architettura di Rippel che a quella di Agustsson. Da notare però che l'architettura di Agustsson è testata a una quantità minore di bpp e questo di sicuro influisce sul risultato finale (che si discosta parecchio dall'immagine originale).

Nonostante HiFiC sia considerato lo stato dell'arte per quanto riguarda i sistemi di compressione lossy basati su modello generativo, esso presenta alcuni svantaggi rispetto alle architetture concorrenti:

- computazionalmente oneroso
- ancora in via di sviluppo
- addestrando G per la ricostruzione di immagini con una loss-function che punta sul realismo, può produrre immagini che sono molto diverse da quelle originali ed è quindi sconsigliabile utilizzarlo in tutte quelle applicazioni che trattano dati sensibili (ad esempio immagini mediche o dal valore legale). Per la compressione di questo tipo di immagini è preferibile utilizzare un modello generativo basato su ROI.

CAPITOLO 3

Rimozione di artefatti tramite GAN

Nel capitolo precedente abbiamo analizzato come sia possibile sfruttare le GAN durante il processo di compressione di un'immagine. In questo capitolo invece esaminiamo come le GAN possono essere impiegate per rimuovere eventuali artefatti comparsi a seguito di una compressione.
La comparsa di artefatti risulta fastidiosa in quanto essi eliminano dettagli presenti nell'immagine originale e aggiungono strutture a forma di blocco, sfocature e contorni smussati. Queste distorsioni rendono le immagini meno gradevoli all'occhio umano e possono anche compromettere il corretto funzionamento di eventuali algoritmi di riconoscimento.

Rimuovere gli artefatti dovuti alla compressione può essere visto come un problema di trasformazione dell'immagine. Gli approcci generativi sono in genere molto bravi a risolvere questo tipo di problema. Una FCN (Fully Convolutional neural Network) cioè una CNN in cui nessun layer è fully connected, è in grado di eseguire trasformazioni locali e non lineari in immagini di qualsiasi risoluzione. Nel 2017 Galtieri et al. [11], ricercatori dell'Università di Firenze, introdussero una FCN addestrata tramite GAN che avesse come obiettivo proprio la cancellazione degli artefatti dalle immagini.

Nella pratica rimuovere gli artefatti significa ricostruire un'immagine I^{RQ} da un'immagine compressa di ingresso $I^{LQ} = A(I^{HQ})$ dove I^{HQ} è l'immagine originale, I^{LQ} è l'immagine compressa, A è l'algoritmo di compressione e I^{RQ} è l'immagine a cui sono stati rimossi gli artefatti.
I^{HQ}, I^{LQ}, I^{RQ} sono matematicamente visti come tensori con dimensione WxHxC con W = larghezza, H = altezza e C = numero canali.
Se il processo di compressione può essere riassunto nella formula:
$I^{LQ} = A(I^{HQ}) \in [0, 255]^{WxHxC}$
il processo di rimozione degli artefatti consiste nel trovare la funzione inversa $G \approx A^{-1}$:
$G(I^{LQ}) = I^{RQ} \approx I^{HQ}$
Per far ciò è necessario addestrare una CNN $G(I^{LQ}; \theta_G)$ con θ_G parametri della rete (pesi e bias) ottimizzando una funzione L_{AR} tramite la risoluzione di:

$$\theta'_G = arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} L_{AR}(I^{HQ}, G(I^{LQ}, \theta_G))$$

con N = numero immagini del training set

Generatore

Il generatore è una FCN in cui ogni layer convoluzionale è seguito da una funzione di attivazione di tipo LeakyReLu (fatta eccezione per l'ultimo che usa come funzione una tangente iperbolica). I 15 blocchi residuali sono utilizzati per rendere più semplice l'addestramento ed evitare l'overfitting.

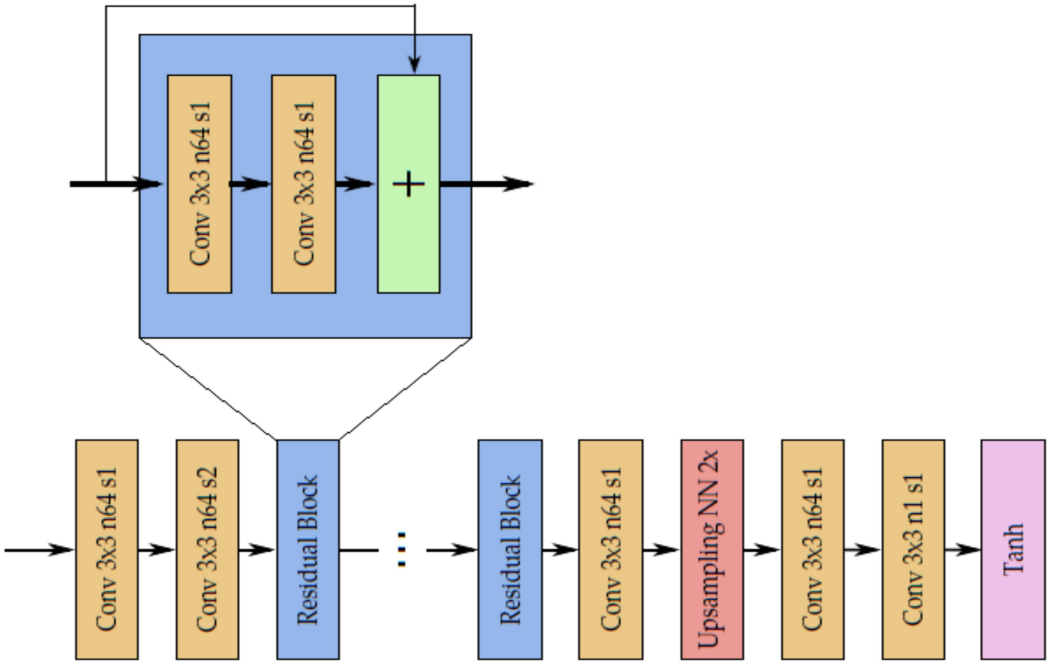


Figura 20: Generatore utilizzato da Galtieri. n indica il numero di filtri del layer mentre s lo stride

Discriminatore

Come il generatore, anche il discriminatore si identifica in una FCN in cui ogni layer convoluzionale è seguito da una funzione di attivazione di tipo LeakyReLu (fatta eccezione per l'ultimo che usa come funzione una sigmoide).
La loss-function da minimizzare è:

$$L_D = -\log(1 - D(I^{RQ} | I^{LQ})) - \log(D(I^{HQ} | I^{LQ}))$$

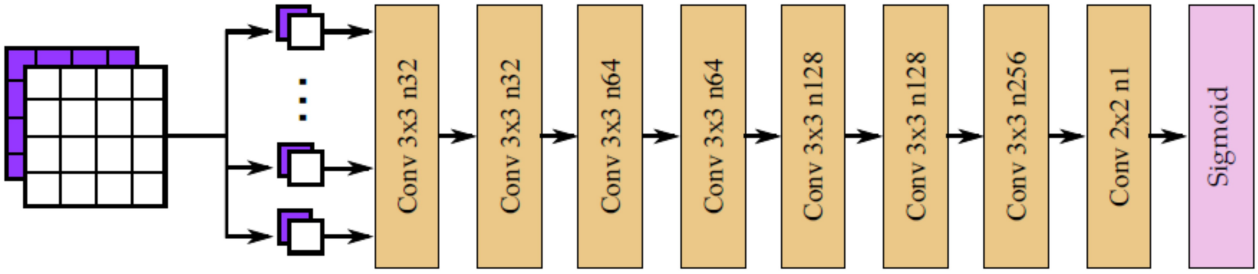


Figura 21: Discriminatore utilizzato da Galtieri, stride = 1

Siccome gli algoritmi di compressione tradizionali suddividono le immagini in finestre e gli artefatti tendono a comparire all'interno di tali finestre, il discriminatore lavora a livello di sotto-finestre. I^{HQ} e I^{RQ} vengono suddivise in P finestre di dimensioni 16x16 e poi vengono passate al discriminatore. In Figura 22 si può notare l'effetto benefico di questo approccio.

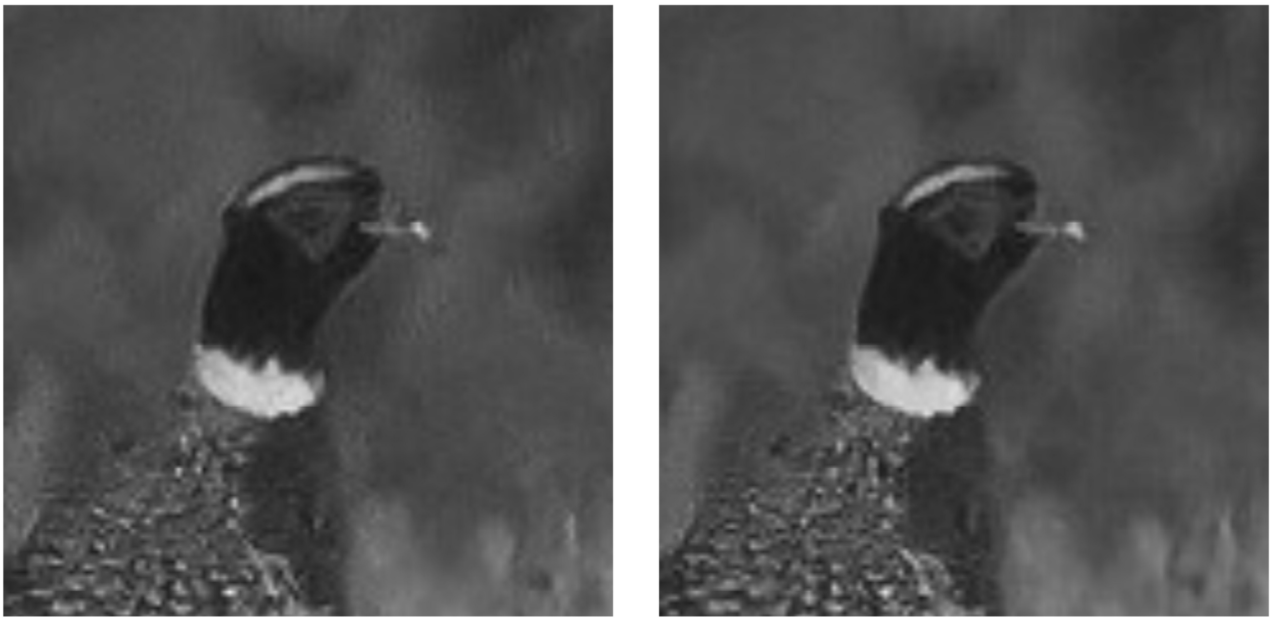


Figura 22: Sinistra: ricostruzione in cui non si è addottata la strategia di sub-patch. Destra: ricostruzione in cui si è addottata la strategia di sub-patch. Il rumore è visibilmente ridotto

La loss function del generatore è:

$L_G = L_P + \lambda L_{ADV}$

con λ iperparametro.

L_P è una funzione basata sulla differenza percettiva tra la rappresentazione in uno spazio delle feature di I^{HQ} e la rappresentazione nello stesso spazio delle feature di I^{RQ}. La funzione che mappa le due immagini nello spazio delle feature è Φ e la differenza è calcolata usando la distanza euclidea.

$$L_P = \frac{1}{W_f H_f} \sum_{x=1}^{W_f} \sum_{y=1}^{H_f} \left(\Phi\left(I^{HQ}\right)_{x,y} - \left(\Phi\left(I^{RQ}\right)_{x,y}\right)\right)^2$$

con W_f e H_f rispettivamente larghezza e altezza delle feature map

L_{ADV} è la tipica adversarial-loss:

$$L_{ADV} = -\log\left(D\left(I^{RQ} | I^{LQ}\right)\right)$$

In Figura 23 è possibile osservare l'operato di questo modello su un'immagine compressa con JPEG.

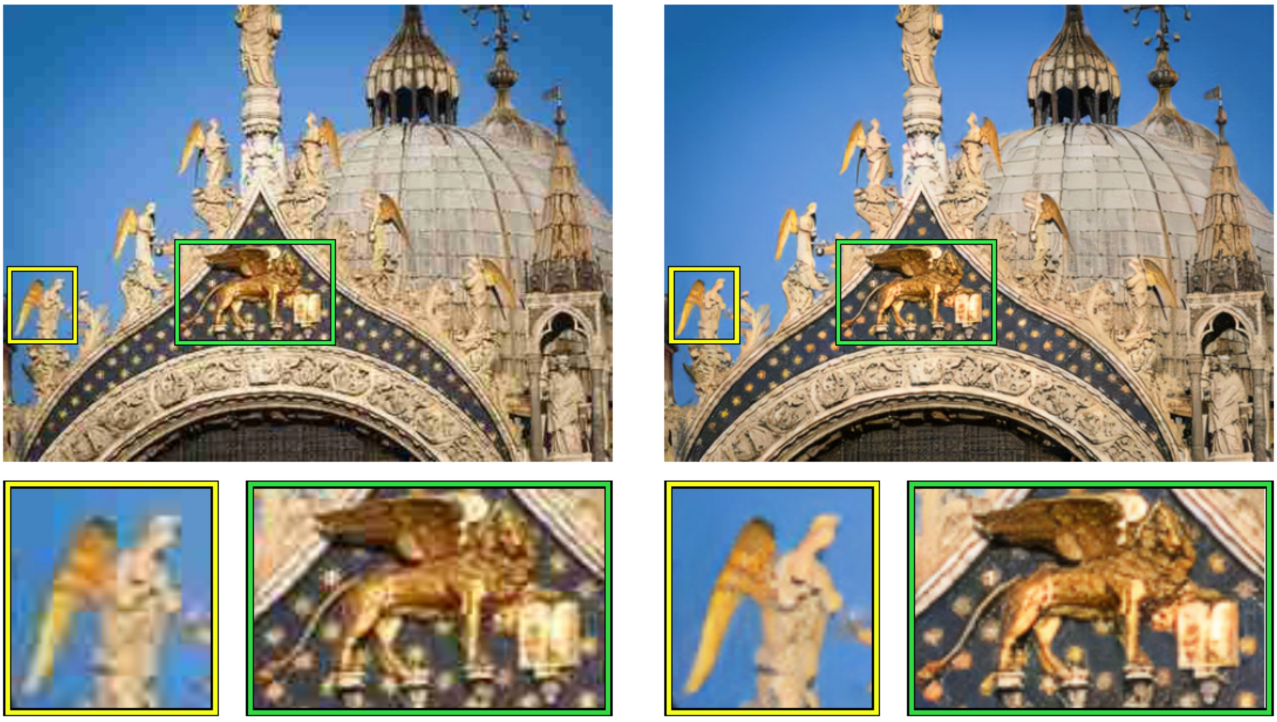


Figura 23: Sinistra: immagine compressa con JPEG che presenta delle aree degradate.

Destra: immagine ricostruita. La maggior parte degli artefatti è stata rimossa e le aree risultano più nitide.

Il miglioramento delle performance che questa architettura può apportare ad un sistema di object detection è riassunto in Figura 24. I valori indicano la precisione con cui il detector è riuscito a riconoscere l'oggetto raffigurato nelle immagini del dataset PASCAL VOC2007.

JPEG	0.587	0.692	0.516	0.434	0.350	0.673	0.71	0.559	0.334	0.559	0.579
Our MSE	0.647	0.696	0.512	0.406	0.409	0.713	0.75	0.542	0.386	0.546	0.614
Our SSIM	0.655	0.706	0.513	0.417	**0.411**	0.713	0.746	0.555	0.387	0.538	**0.615**
Our GAN	**0.666**	**0.753**	**0.565**	**0.475**	0.395	**0.727**	**0.770**	**0.725**	**0.403**	**0.684**	0.602
Original	0.698	0.788	0.692	0.559	0.488	0.769	0.798	0.858	0.487	0.762	0.637

JPEG	0.532	0.691	0.665	0.638	0.260	0.482	0.434	0.707	0.570	0.549
Our MSE	0.595	0.713	0.668	0.664	**0.310**	0.485	0.522	0.676	0.600	0.573
Our SSIM	0.596	0.720	0.666	0.663	0.308	0.482	0.532	0.668	0.598	0.574
Our GAN	**0.718**	**0.753**	**0.707**	**0.670**	0.303	**0.625**	**0.586**	**0.712**	**0.611**	**0.623**
Original	0.790	0.802	0.757	0.763	0.376	0.683	0.672	0.777	0.667	0.691

Figura 24: Tabella coi valori di precisione del detector

"Our MSE" indica il modello allenato senza GAN e con MSE come loss-function, "Our SSIM" indica il modello allenato senza GAN e con SSIM come loss-function, mAP indica la precisione media (comprendendo tutti gli oggetti) del detector per il codec considerato.

CAPITOLO 4

Attività sperimentali

In questo capitolo vengono presentate le attività sperimentali a supporto della tesi.
Il lavoro svolto è stato suddiviso in due sezioni entrambe caratterizzate dal confronto tra HiFiC e due codec convenzionali: JPEG e WebP.
Il codice che permette di sfruttare la compressione HiFiC è reperibile su github presso l'URL [12].
Il PSNR è stato calcolato utilizzando il programma PSNR 1.2 scaricabile presso l'URL [13].
Il SSIM è stato calcolato implementando il codice reperibile presso l'URL [14].

Nella prima sezione del capitolo vengono compresse due immagini diverse (una rappresentante Porta Portello e una rappresentante un koala) e vengono calcolati per entrambe i valori di PSNR e SSIM. L'esperienza è ripetuta tre volte per evidenziare le differenze tra i tre codec a tre differenti livelli di risoluzione: 1920x1080, 960x540 e 320x180.

Nella seconda parte del capitolo vengono analizzati i risultati ottenuti dalla compressione di un sottoinsieme di immagini del dataset Kodak [15]. Per entrambe le metriche (PSNR e SSIM) viene visualizzato un grafico e calcolata la media.

Nell'analizzare i dati è bene tener conto che la compressione indicata (in bit per pixel) è quella utilizzata dal modello HiFiC e costituisce un lower bound per quanto riguarda quelle usate da JPEG e WebP. Nella pratica ciò si traduce nel fatto che le immagini compresse in JPEG o WebP possono risultare fino a 1-2KB più grandi rispetto a quelle compresse con HiFiC.

4.1 SEZIONE I

Immagine 1

Risoluzione: 1920x1080. Compressione: 0,187bpp

Originale

HiFiC, PSNR: 26.06, SSIM: 0.826

JPEG, PSNR: 24.56, SSIM: 0.741

WebP, PSNR: 26.71, SSIM: 0.791

Risoluzione: 960x540. Compressione: 0,224bpp

Originale

HiFiC, PSNR: 22.38, SSIM: 0.741

JPEG, PSNR: 21.94, SSIM: 0.680

WebP, PSNR: 23.57, SSIM: 0.725

Risoluzione: 320x180. Compressione: 0,290bpp

Originale

HiFiC, PSNR: 21.92, SSIM: 0.783

JPEG, PSNR: 22.94, SSIM: 0.711

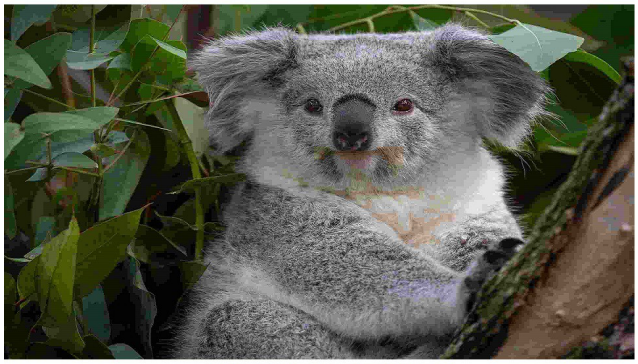
WebP, PSNR: 22.90, SSIM: 0.713

Immagine 2 [16].

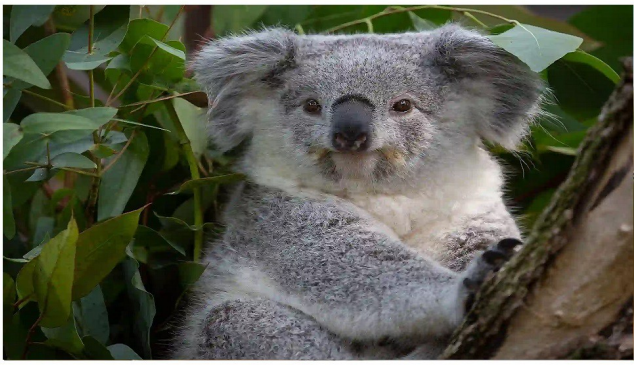
Risoluzione: 1920x1080. Compressione: 0,160bpp

Originale

HiFiC, PSNR: 25.42, SSIM: 0.721



JPEG, PSNR: 26.15, SSIM: 0.712

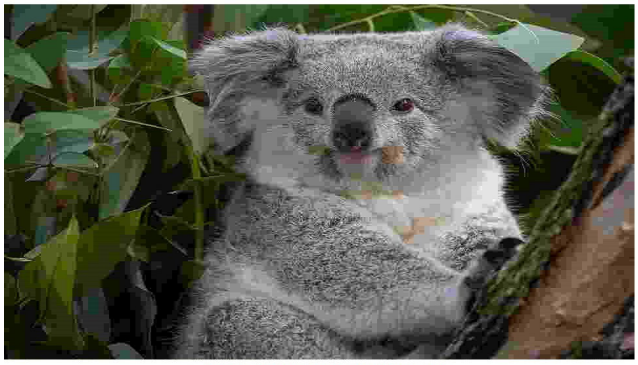


WebP, PSNR: 26.79, SSIM: 0.729

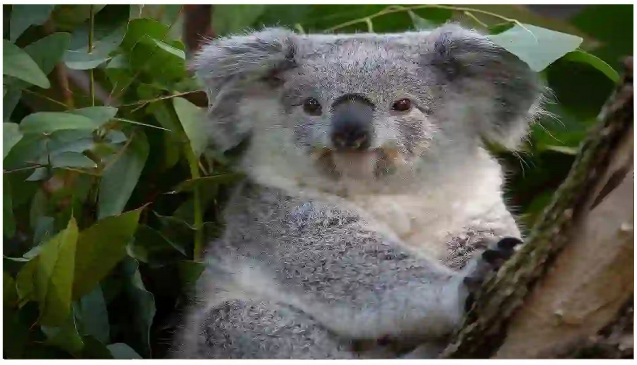
Risoluzione: 960x540. Compressione: 0,190bpp

Originale

HiFiC, PSNR: 23.63, SSIM: 0.659



JPEG, PSNR: 23.96, SSIM: 0.611



WebP, PSNR: 24.65, SSIM: 0.648

Risoluzione: 320x180. Compressione: 0,281bpp

Originale

HiFiC, PSNR: 25.01, SSIM: 0.687

JPEG, PSNR: 23.90, SSIM: 0.575

WebP, PSNR: 25.59, SSIM: 0.656

4.2 SEZIONE II

In questa sezione ho eseguito un confronto utilizzando un sottoinsieme del dataset Kodak. Le immagini impiegate sono quelle di Figura 25.

Figura 25

La tabella sottostante riporta i valori ottenuti durante il test.

Immagine	PSNR HiFiC	SSIM HiFiC	PSNR JPEG	SSIM JPEG	PSNR WebP	SSIM WebP	Compres. (bpp)
1	24,49	0,690	24,27	0,666	25,71	0,703	0,249
2	24,73	0,770	23,81	0,701	25,19	0,744	0,295
3	23,90	0,772	23,25	0,733	24,20	0,736	0,289
4	31,82	0,860	29,23	0,773	31,95	0,854	0,125
5	26,25	0,717	25,61	0,665	27,07	0,712	0,237
6	29,87	0,808	28,51	0,751	31,03	0,817	0,141
7	25,98	0,786	24,82	0,732	26,98	0,789	0,175
8	32,37	0,884	30,12	0,806	33,01	0,883	0,119
9	24,76	0,762	25,11	0,723	26,29	0,742	0,233
10	27,62	0,721	27,82	0,716	28,77	0,726	0,186
Media	27,18	0,777	26,26	0,727	28,02	0,771	

Come già detto la compressione indicata è quella utilizzata dal modello HiFiC e costituisce un lower bound per quanto riguarda quelle usate da JPEG e WebP.

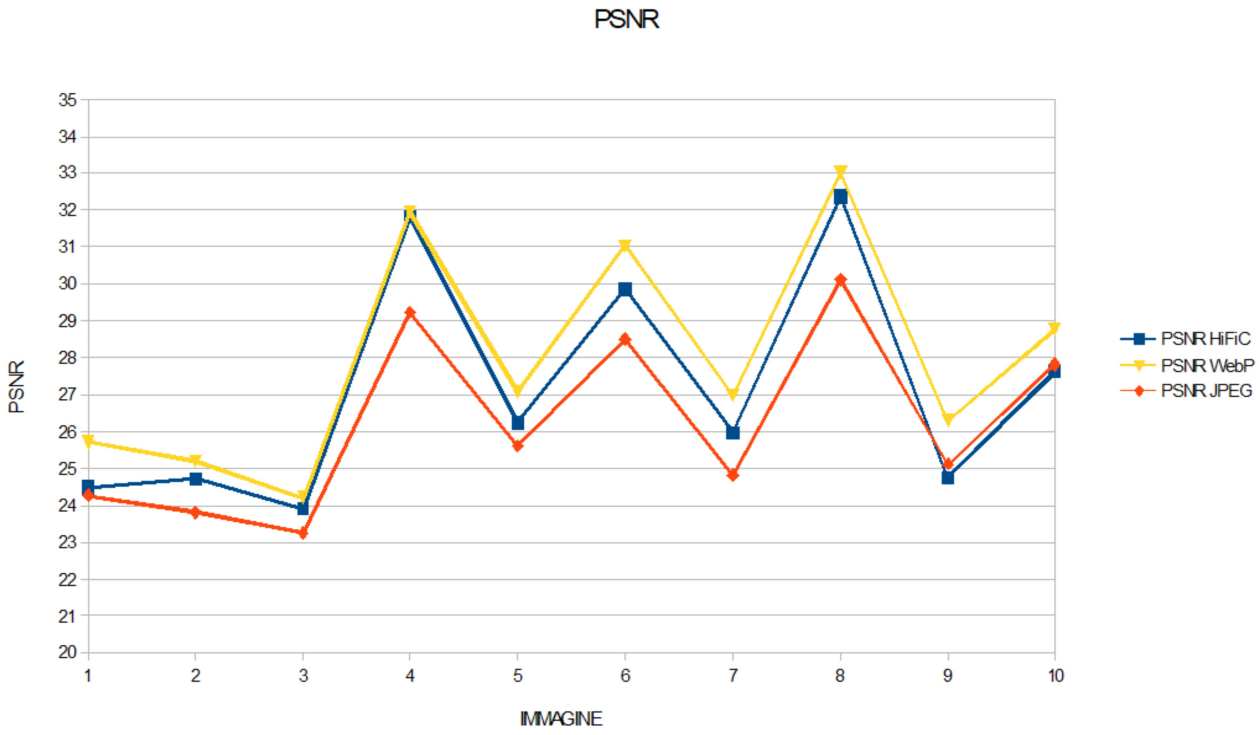


Grafico dei valori PSNR

Non eesendo in grado di considerare la qualità percepita dell'immagine, il PSNR valuta le ricostruzioni di HiFiC migliori di quelle di JPEG, ma peggiori di quelle di WebP.

Grafico dei valori SSIM

Come è possibile rilevare osservando la tabella o il grafico, i valori della SSIM di HiFiC sono quasi sempre quelli più alti. In Figura 26 sono presenti le immagini in cui WebP ottiene sulla carta dei risultati migliori. Osservandole si può riscontrare come nella pratica la SSIM non sempre riesce a valutare in maniera perfetta la qualità percepita da un umano di un'immagine (nonostante una miglior capacità rispetto a MSE e PSNR).

Figura 26: Le immagini compresse con HiFiC risultano più nitide e dettagliate, nonostante un SSIM peggiore

CONCLUSIONI

La ricerca nel campo della compressione sta portando all'ideazione di molte nuove e interessanti architetture basate sull'intelligenza artificiale. Questo lavoro di tesi ha voluto analizzarne alcune mostrandone, tramite anche l'ausilio di immagini, i benefici in termini di bitrate e qualità visiva. Le attività sperimentali svolte nel capitolo 4 hanno voluto evidenziare come un'architettura basata su reti neurali possa ottenere risultati competitivi (se non migliori) rispetto a quelli dei codec tradizionali. Si è infine fatto notare come le metriche utilizzate finora (come PSNR e SSIM) siano perlopiù inadeguate a valutare l'operato di queste tecniche emergenti.

BIBILIOGRAFIA

[1] Cisco Visual Networking Index: Forecast and Trends, 2017–2022

[2] G.E. Hinton , R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313, 2006

[3] H. Ma, D. Liu, R. Xiong, F. Wu, iWave: CNN-based wavelet-like transform for image compression, IEEE Transactions on Multimedia, 2019

[4] L. Theis , W. Shi , A. Cunningham , F. Huszár , Lossy image compression with compressive autoencoders, Proceedings of the International Conference on Learning Representations, 2017

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, in Advances in Neural Information Processing Systems, 2014

[6] https://developers.google.com/machine-learning/gan/generator?hl=it

[7] Oren Rippel and Lubomir Bourdev, Real-time adaptive image compression, Proceedings of the 34th International Conference on Machine Learning, 2017

[8] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, L. van Gool, Generative adversarial networks for extreme learned image compression, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019

[9] Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson, High-fidelity generative image compression, Advances in Neural Information Processing Systems, 2020

[10] Yochai Blau and Tomer Michaeli, Rethinking lossy compression: The rate-distortion-perception tradeoff, arXiv preprint arXiv:1901.07821, 2019

[11] L. Galteri, L. Seidenari, M. Bertini, A. Del Bimbo, Deep generative adversarial compression artifact removal, Proceedings of the IEEE International Conference on Computer Vision, 2017

https://it.wikipedia.org/wiki/Peak_signal-to-noise_ratio

https://en.wikipedia.org/wiki/Structural_similarity

https://it.wikipedia.org/wiki/Divergenza_di_Kullback-Leibler

[12] https://github.com/tensorflow/compression/tree/master/models/hific

[13] https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Others/PSNR.shtml

[14] https://code.adonline.id.au/structural-similarity-index-ssim-in-python/

[15] https://r0k.us/graphics/kodak/

[16] https://wall.alphacoders.com/big.php?i=1058202