Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Identificazione di immagini sintetiche tramite codifica JPEG AI

Candidato
Rustichini Edoardo

Relatore
Prof. Piva Alessandro

Correlatori
Shullani Dasara
Baracchi Daniele

Anno Accademico 2024-2025

Indice

Introduzione

I continui progressi nel campo dell'intelligenza artificiale hanno sviluppato tecniche per la generazione di contenuti digitali con un livello di realismo tale da renderli indistinguibili da quelli reali. La diffusione di queste tecnologie al grande pubblico aumenta il rischio di utilizzi illeciti, con potenziali conseguenze negative. Il rilevamento di questo tipo di contenuti è uno dei temi principali della *multimedia forensics*, disciplina che consiste nello studio e analisi di contenuti digitali con il fine di verificarne l'autenticità.
Questo lavoro di tesi triennale si propone di esaminare l'uso della codifica di JPEG AI, un codec di compressione basato su reti neurali, per svolgere task di *fake detection* su immagini di volti; in particolare si vuole verificare se l'encoder di JPEG AI riesca ad estrarre dalle immagini rappresentazioni latenti sufficientemente ricche di informazioni da rilevare se un'immagine sia sintetica o reale.

Struttura relazione La relazione è strutturata come segue: nel capitolo 1 vengono presentati gli aspetti teorici necessari per la comprensione del lavoro; nel capitolo 2 viene presentato la metodologia e gli aspetti più tecnici riguardanti il lavoro svolto, mentre nel capitolo 3 sono esposti gli esperimenti svolti e i loro risultati. Infine nel capitolo 4 sono tratte le conclusioni.

Capitolo 1

Stato dell'arte

1.1 Immagini fake: rischi, generazione e rilevamento

L'incremento degli studi nel campo dei modelli generativi, ossia tecniche di intelligenza artificiale dedicate alla generazione di dati simili a quelli di addestramento, ha portato allo sviluppo di tecnologie in grado di produrre immagini indistinguibili da quelle reali; la diffusione al pubblico di questi strumenti rende la creazione di immagini sintetiche con alto livello di realismo un'operazione semplice ed accessibile anche a utenti non esperti.
Tra le immagini false si possono distinguere due categorie principali: *cheapfakes*, tecniche di manipolazione meno sofisticate, come *splicing* e *copy-move ecc*, e *deepfakes*, metodi che fanno uso di intelligenza artificiale.
Il termine deepfake nasce dalla combinazione di "deeplearning" e "fake" e vuole indicare la manipolazione di contenuti già esistenti o la generazione di nuovi tramite l'utilizzo di modelli di deeplearning. L'attenzione di questo lavoro sarà concentrata su quest'ultima categoria.

L'uso più comune dei deepfakes riguarda la produzione e rielaborazione di immagini rappresentanti volti umani, con applicazioni nel mondo dell'intrattenimento o del marketing; le stesse tecnologie possono però essere utilizzate per fini illeciti come la creazione di profili falsi ma apparentemente autentici, diffusione di disinformazione e produzione di materiale diffamatorio nei confronti di personaggi pubblici o privati.

Manipolazioni di volti Tra le più comuni manipolazioni si trovano [1]: *entire face synthesis*, ossia la creazione di immagini di volti di persone non reali; *identity swap*, che consiste nella sostituzione del volto di un individuo con quello di un altro; *attribute manipulation*, ovvero la modifica di attributi/caratteristiche del volto e *expression swap*, che ha lo scopo di modificare l'espressione facciale della persona scelta. Nella figura 1.1 sono mostrati esempi per ogni tipo di manipolazione. Questo lavoro si concentra sulla *entire face synthesis* e ci riferiremo principalmente a questa parlando di generazione e identificazione di immagini sintetiche.

1.1.1 Generazione

I modelli generativi hanno l'obbiettivo di apprendere la distribuzione dei dati di addestramento e trovare il miglior modo per rappresentarli in modo da generare nuovi campioni simili a quelli originali. Per la creazione di immagini di volti sintetiche sono usate principalmente tre architetture [2]: Autoencoders, Diffusion Models e GANs. Comprenderne il funzionamento è fondamentale per sviluppare metodi di identificazione.

Autoencoders Sono architetture composte da un encoder, che ha l'obbiettivo di trasformare l'input in una rappresentazione latente compatta, ed

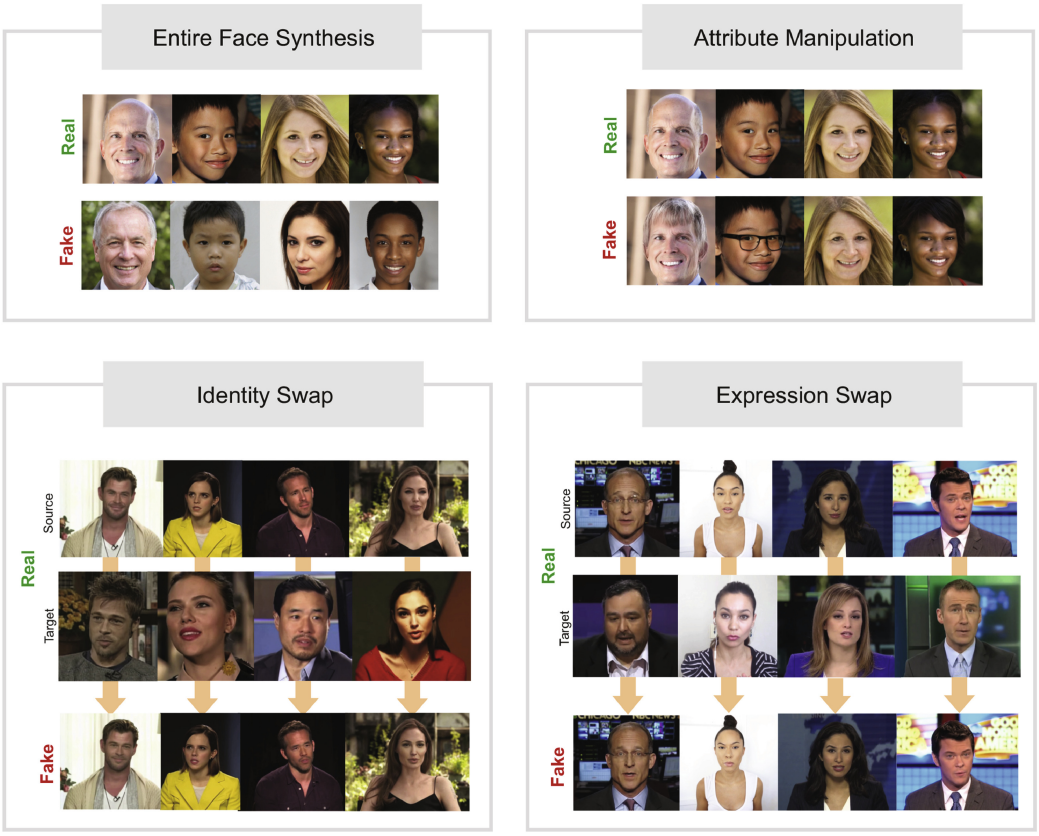


Figura 1.1: Esempi per ogni tipo di manipolazione su immagini di volti

un decoder che ricostruisce l'immagine desiderata. Sono spesso usate nell'identity swapping sfruttando un encoder comune che apprende una rappresentazione latente condivisa, e decoder separati che si specializzano nella ricostruzione di immagini relative ad un individuo specifico [2], così da ottenere uno scambio di identità decodificando la rappresentazione compatta della persona A con il decoder della persona B.

Diffusion Models I modelli di diffusione sono modelli generativi il cui funzionamento si basa su due fasi: nel *forward process* un'immagine reale viene corrotta introducendo rumore in un determinato numero di iterazioni; successivamente nel *reverse process* il modello viene addestrato a sintetizzare un'immagine realistica rimuovendo progressivamente rumore da rumore puro [3]. L'approccio iterativo di queste fasi permette di generare immagini di alta qualità riuscendo a catturare dettagli fini; per questo i Diffusion Models rappresentano attualmente lo stato dell'arte nel campo della generazione di immagini.

1.1.2 GANs

GAN (*Generative Adversarial Networks*) è l'architettura proposta nel 2014 da Goodfellow et al. [4] composta da due reti neurali: generatore e discriminatore. Il generatore G riceve in input rumore casuale z e lo mappa in un'immagine sintetica $G(z)$; il discriminatore D riceve in input immagini reali o sintetiche ed ha il compito di classificarle. Le due reti vengono addestrate contemporaneamente competendo in un *min-max game* nel quale G, generando immagini sempre più realistiche, ha l'obbiettivo di massimizzare la probabilità che D compia un errore nella classificazione, mentre D deve

minimizzarla. L'equazione che descrive questo gioco è la seguente:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1.1)$$

Durante l'addestramento il gradiente della funzione di perdita di D verrà usato per aggiornare G. Inizialmente G genererà immagini rumorose e facilmente identificabili come falsi da D, ma con il procedere della fase di addestramento la qualità delle immagini aumenterà fino a farle diventare abbastanza realistiche da ingannare D.
Questa architettura ha rappresentato il primo salto di qualità nel campo della generazione di immagini e per anni è stata il punto di riferimento. Di seguito viene approfondita la variante di GAN usata per creare il dataset di immagini sintetiche per gli esperimenti svolti (vedi sez 2.2).

StyleGAN StyleGAN [5] presenta un'importante innovazione rispetto alla struttura classica: il generatore non riceve come input direttamente il rumore casuale z, ma questo viene prima trasformato da un mapping network $f : \mathcal{Z} \to \mathcal{W}$ in un vettore $f(z) = w$ in uno spazio latente intermedio $\mathcal{W}$ non vincolato dalla distribuzione del dataset di addestramento, proprietà che invece vincola $\mathcal{Z}$. Successivamente w viene trasformato in una serie di stili y che sono interpretati da ogni layer del generatore come istruzioni su come operare attraverso un'operazione chiamata *Adaptive Instance Normalization* (AdaIN), controllando così il contributo di ogni layer alla generazione dell'immagine.
L'architettura è visbile in fig. 1.2. L'innovazione di StyleGAN consiste quindi nel modo innovativo di generare immagini basandosi su una collezione di stili; in particolare gli effetti di ogni stile sono localizzati nella rete, permettendo di scegliere sottoinsiemi di stili per modificare solo alcuni aspetti dell'immagine. Da notare come StyleGAN non modifica la struttura del discriminatore,

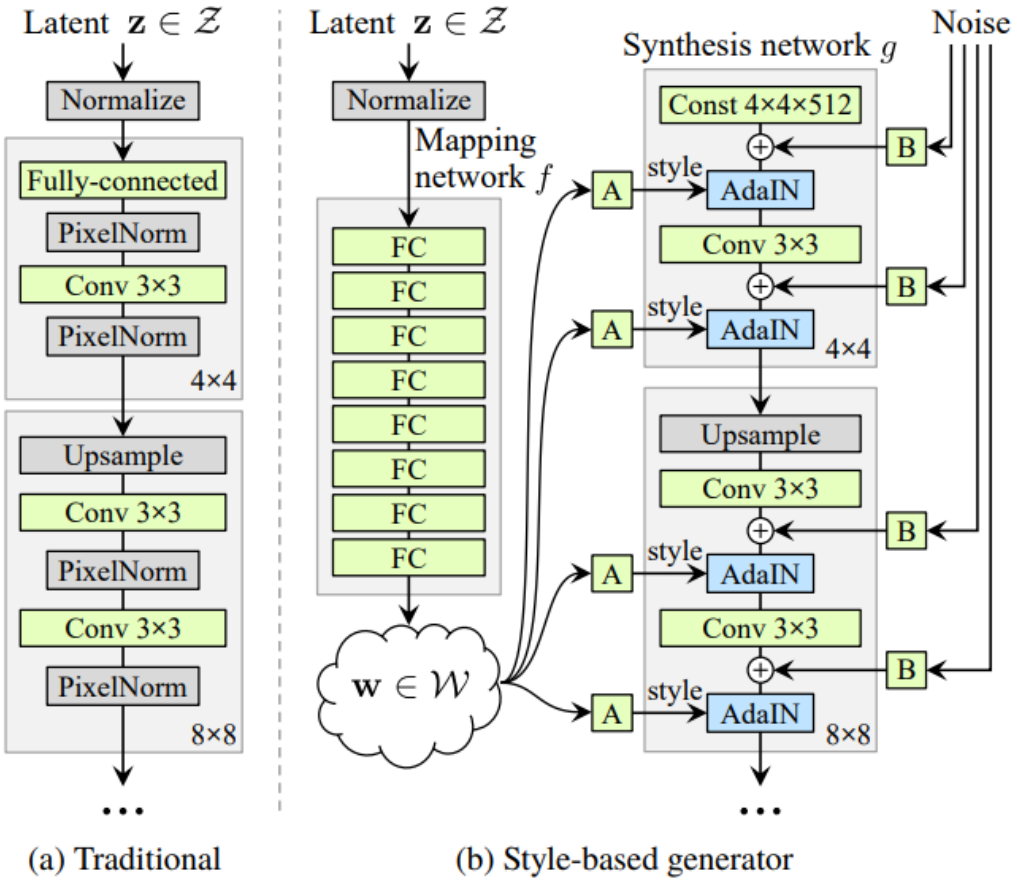


Figura 1.2: Architettura di StyleGAN descritta in [5]

che rimane quello classico, ma cambia il controllo e la qualità del generatore, permettendo di ottenere immagini più realistiche e con un maggior controllo sugli attributi generati.

1.1.3 Rilevamento

Data un'immagine I un detector di immagine sintetiche ha il compito di rendere un valore chiamato detection score che indica la probabilità che I sia sintetica. La ricerca riguardante tecniche di rilevamento è in continuo sviluppo data la necessità di riuscire a contrastare le tecnologie di generazione sempre più sofisticate. I metodi di rilevamento si possono suddividere in due categorie: quelli classici e quelli che sfruttano modelli di intelligenza artificiale.

Approcci classici I metodi di rilevamento classici si basano sull'analisi delle immagini in ricerca di anomalie create durante la fase di generazione. I *Physical-based methods* cercano di sfruttare inconsistenze nell'immagine, come illuminazione o il riflesso della luce negli occhi, che non sarebbero presenti in un'immagine reale. I *Physiological-based Methods* invece investigano l'aspetto semantico, analizzando anomalie riguardanti l'asimmetria di attributi dei volti come diversi colori negli occhi, differenti forme delle pupille o un numero errato di denti. Altri metodi invece cercano di trovare artefatti nelle caratteristiche "strutturali" delle immagini tra cui: imperfezioni causate dal sistema ottico delle fotocamere nel processo di acquisizione, processing interno o esterno alla fotocamera.
Una caratteristica studiata è la PRNU (*Photo Response Non-Uniformity*): ogni fotocamera lascia un pattern di rumore specifico nelle fotografie che può essere usato come una sorta di "impronta digitale"; l'assenza di questo pat-

tern in una regione dell'immagine è un forte indicatore di manipolazione.
Allo stesso modo modelli generativi come GAN risultano lasciare degli artefatti chiamati "*GAN fingerprint*" particolarmente riconoscibili analizzando le immagini nel dominio della frequenza per questo molti metodi classici analizzano lo spettro di Fourier.

Approcci basati su deeplearning Gli approcci più recenti sfruttano il deep learning, usando reti neurali per estrarre le feature sulle quali fare le decisioni di classificazione in modo automatico. Nel 2020 si è svolta la *Kaggle Deepfake Detection Challenge* [6] , e le migliori soluzioni proposte erano basate su EfficientNet B7 [7] e XceptionNet [8]. Un'altra architettura molto usata è ResNet [9], che introduce il concetto di *residual connetions* permettendo prestazioni superiori. Recentemente i Vision transformer basati sulla self-attention, inizialmente sviluppata per Natural Language Processing, sono stati usati in questo campo per la capacità di catturare relazioni globali nell'immagine.
Con l'avvento dei Diffusion Models alcuni ricercatori hanno provato a sviluppare tecniche basate sull'inversione del processo di diffusione sfruttando l'idea che l'immagine generata può essere riportata alla sua rappresentazione latente originale: facendo questo e ricostruendo l'immagine si possono analizzare gli errori di ricostruzione per la classificazione, e identificare immagini reali ipotizzando che dovrebbero presentare maggiori errori di ricostruzione [10].

Confronto tra i due approcci Gli approcci classici riescono ad ottenere buoni risultati su immagini non compresse, ma questa efficienza diminuisce drasticamente quando si lavora con immagini compresse, come quelle condivise sui social media, dove invece i metodi basati su deeplearning riescono a

mantenere un buon livello di accuratezza. I metodi più recenti hanno però due svantaggi principali: il problema della non explainability, e della generalizzabilità. La explainability è un requisito fondamentale nel campo della forensics, dato che la decisione di classificare un'immagine come sintetica deve essere motivata da prove concrete nel caso di ambiti legali. Mentre come dimostrato in [11], nel rilevamento di *Fake face images*, strumenti che risultano avere buone performance non riescono a generalizzare bene quando vengono testate su immagini sintetiche generate da tipi di architetture non presenti nel dataset di addestramento.

1.2 JPEG AI

La qualità e la risoluzione delle immagini è in continua crescita, e così la loro dimensione in formato non compresso; considerando anche l'aumento del numero di immagini prodotte e visualizzate quotidianamente, si è sviluppata la necessità di trovare dei nuovi metodi di compressione in modo da consentire una trasmissione e un'archiviazione più efficienti. Un nuovo promettente paradigma di compressione è chiamato"*Learning-based image compression*" o "*Neural image compression*" e sfrutta l'apprendimento automatico facendo leva sulle reti neurali per raggiungere livelli di compressione maggiori.

1.2.1 Compressione di immagini

"*A picture is worth a thousand words*"

La compressione dei dati è un problema ampiamente studiato nell'informatica: l'obiettivo è quello di rappresentare un'informazione usando un numero ridotto di bit rispetto alla rappresentazione originale. Nel caso delle immagini questo equivale a rappresentare la stessa informazione visiva, almeno apparentemente, riducendo lo spazio di archiviazione utilizzato. La compressione si divide in due approcci: lossless, senza perdita di informazione, e lossy, ovvero con perdita di informazione.
La compressione senza perdita viene utilizzata nelle applicazioni in cui è richiesta una perfetta ricostruzione dell'immagine originale, per esempio nel caso di immagini mediche. Quella con perdita viene usata per esempio nella condivisione delle immagini sul web perché consente di raggiungere un maggior livello di compressione scartando parte dell'informazione ma mantenendo una qualità accettabile.

Compressione lossy classica Tradizionalmente il processo per la compressione lossy prevede i seguenti passaggi: inizialmente viene applicata una trasformata all'immagine che mappa i pixel dal dominio spaziale ad uno più efficiente per la compressione, convertendoli in coefficienti. Questi vengono successivamente quantizzati, operazione nella quale avviene la principale perdita di informazione. Infine si effettua una codifica dell'entropia lossless per trasformare la sequenza di coefficienti quantizzati in un bitstream finale.
Il più importante e diffuso esempio di compressione lossy è JPEG [12]; il suo successo è dovuto al basso costo computazionale necessario per il calcolo della DCT (*Trasformata discreta del coseno*), che, applicata a blocchi di 8×8 pixel, mappa i pixel dal dominio spaziale a quello della frequenza, permettendo di concentrare la maggior parte dell'informazione rilevante in pochi coefficienti. Dopo questa trasformata i coefficienti vengono quantizzati in base alla loro frequenza, usando maggiore precisione per le frequenze più basse, che contengono più informazione per l'occhio umano, e approssimando maggiormente quelle più alte.
In generale le tecniche tradizionali sono state progettate "a mano" dai ricercatori sfruttando la conoscenza della struttura probabilistica dell'informazione, ed infatti fanno uso di una trasformata predeterminata e fissa, avendo lo svantaggio di non essere abbastanza flessibile e ottimale per tutti i tipi di immagine.

Learned image compression

In questo nuovo paradigma vengono utilizzate le reti neurali: i primi articoli che propongono tecniche di questo tipo risalgono già agli anni '90 [13, 14], quando ancora la loro implementazione non era possibile nella pratica. Recentemente, con i progressi nell'apprendimento automatico e lo sviluppo di

hardware specializzato come le GPU, anche il campo del learned image compression si è evoluto.
La differenza principale con i codec tradizionali come JPEG risiede nel fatto che in questo nuovo approccio ogni componente classica (trasformata, quantizzazione, codifica dell'entropia) è sostituita da una rete neurale, portando alla creazione di un *learned image codec*.

Architettura La maggior parte delle tecniche sono basate sugli autoencoder [15], un tipo di rete neurale in grado di apprendere la mappatura tra l'input e uno spazio di una rappresentazione latente compatta, spesso chiamato *bottleneck*.
L'architettura che ha riscosso più successo è quella proposta da [16] che ha dato prova dell'efficienza di questa tecnica, migliorata successivamente da [17].
Uno degli aspetti fondamentali è l'approccio end-to-end nel quale tutti i moduli, dall'encoder al decoder, vengono addestrati insieme mediante un'unica *global loss-function* permettendo un'ottimizzazione "unificata" che massimizza l'efficienza di compressione. Questo rappresenta un vantaggio rispetto ai metodi tradizionali in cui ogni componente viene ottimizzata in modo indipendente.
Oltre agli autoencoder, si possono trovare anche altri tipi di architettura, e come proposto in [18] si possono classificare in base a questi criteri:

- tipologia di rete neurale utilizzata: oltre a quelli già citati, sono usate anche RNN, per elaborare l'immagine in modo sequenziale, oppure GAN (descritti in sez. 1.1.2)

- dimensione dell'unità di codifica: alcune tecniche scelgono di elaborare l'intera immagine, mentre altre decidono di lavorare a blocchi di pixel

riducendo la complessità

- strategia del controllo del bit rate: alcuni metodi scelgono di usare lo stesso modello per comprimere a più bitrate, altri invece utilzzano diversi modelli ottimizzati per un singolo livello di qualità

1.2.2 Motivazioni per lo sviluppo di JPEG AI

La principale motivazione dietro lo sviluppo è descritta nel seguente modo:

> "*The scope of JPEG AI is to create a learning-based image coding standard that provides a single-stream, compact compressed domain representation targeting both human visualization,.., and effective performance for image processing and computer vision tasks, with the goal of supporting royalty-free baseline* [19]"

L'obbiettivo può essere paragonato alla creazione di un "*linguaggio comune*" che consenta una rappresentazione efficiente sia per la visione umana che per macchine. Questa scelta risulta rilevante considerando che ormai i contenuti digitali non sono più visualizzati solo dagli umani, ma in molte applicazioni, come sistemi di sorveglianza intelligente, sono le macchine i destinatari principali. L'idea è quindi di usare la stessa rappresentazione compatta e ricca di informazione sia per task di *image processing*, che *computer vision*, oppure potrebbe essere ricostruita (vedi fig. 1.3) per renderla visibile ad un utente umano.

Un unico bitstream multitask offre due vantaggi: il primo riguarda la riduzione della complessità necessaria a svolgere task di image processing o computer vision, consentendo di evitare completamente la parte di ricostruzione dell'immagine e di agire direttamente sulla rappresentazione latente.

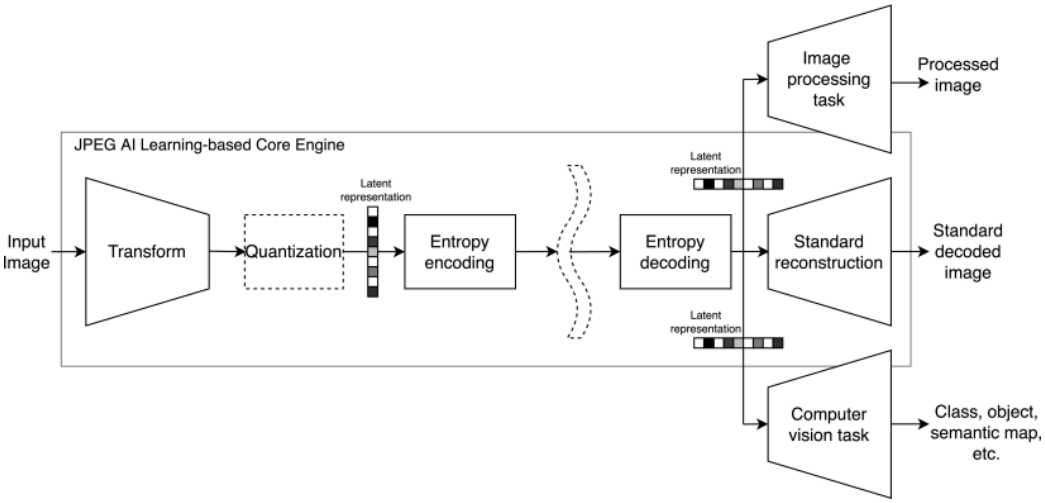


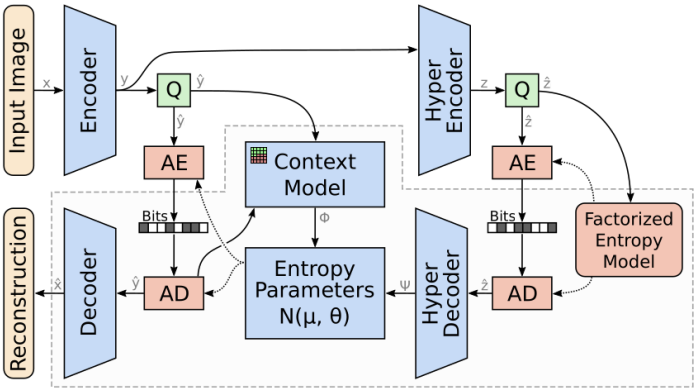
Figura 1.3: Framework JPEG AI

Il secondo vantaggio riguarda il potenziale incremento nell'accuratezza delle operazioni: essendo JPEG AI stato progettato, addestrato ed ottimizzato per trovare una rappresentazione compatta latente che riesca a contenere l'informazione (anche semantica) contenuta nell'immagine originale, svolgere task direttamente su questa rappresentazione potrebbe garantire prestazioni migliori rispetto ad utilizzare l'immagine lossy ricostruita, specialmente a bitrate bassi.

Per ottenere questo l'encoder di JPEG AI deve generare un bitstream indipendente da tutti task, ossia non ottimizzato per nessun compito specifico, mantenendo però il requisito fondamentale di un alto livello di compressione.

1.2.3 Architettura

L'architettura di JPEG AI segue lo schema dei learning-based codec [17] osservabile nella fig.1.4; si possono distinguere due moduli. Il modulo principale è rappresentato dall'autoencoder, composto dalla coppia *encoder-decoder*. L'encoder apprende una trasformata non-lineare chiamata *Analysis*



Component	Symbol
Input Image	$\boldsymbol{x}$
Encoder	$f(\boldsymbol{x};\boldsymbol{\theta}_e)$
Latents	$\boldsymbol{y}$
Latents (quantized)	$\hat{\boldsymbol{y}}$
Decoder	$g(\hat{\boldsymbol{y}};\boldsymbol{\theta}_d)$
Hyper Encoder	$f_h(\boldsymbol{y};\boldsymbol{\theta}_{he})$
Hyper-latents	$\boldsymbol{z}$
Hyper-latents (quant.)	$\hat{\boldsymbol{z}}$
Hyper Decoder	$g_h(\hat{\boldsymbol{z}};\boldsymbol{\theta}_{hd})$
Context Model	$g_{cm}(\boldsymbol{y}_{<i};\boldsymbol{\theta}_{cm})$
Entropy Parameters	$g_{ep}(\cdot;\boldsymbol{\theta}_{ep})$
Reconstruction	$\hat{\boldsymbol{x}}$

Figura 1.4: Architettura descritta in [17]

transform, che mappa un'immagine x in una rappresentazione latente più compatta y. Il decoder apprendende una trasformata non-lineare chiamata *Synthesis transform* che ricostruisce un'approssimazione dell'immagine orginale partendo dalla versione quantizzata $\hat{y}$ della rappresentazione latente.
Il secondo modulo corrisponde al *Modello per l'entropia*: composto da *Hyperprior* e *Context-model* apprende la distribuzione di probabilità dei valori dei tensori latenti per svolgere in modo più efficiente la codifica dell'entropia [16].
L'*Hyperprior* è un autoencoder ausiliario composto da un *hyper-encoder*, che estrae da y delle *side-information/hyper-latent* z che vengono quantizzate ($\hat{z}$) ed incluse nel bitstream, ed un *hyper-decoder*, che partendo $\hat{z}$ aiuta a predire la distribuzione dei valori del tensore latente. Il *Context model*, utilizzando un approccio autoregressivo, usa il contesto dato dai valori già codificati $\hat{y}_{<i}$ per stimare la distribuzione dei valori.

Fase di Encoding Il processo di encoding di un'immagine si può osservare in 1.5: inizialmente l'analysis transform, una rete neurale composta da layer convoluzionali e attivazioni non lineari, trasforma l'immagine in un tensore

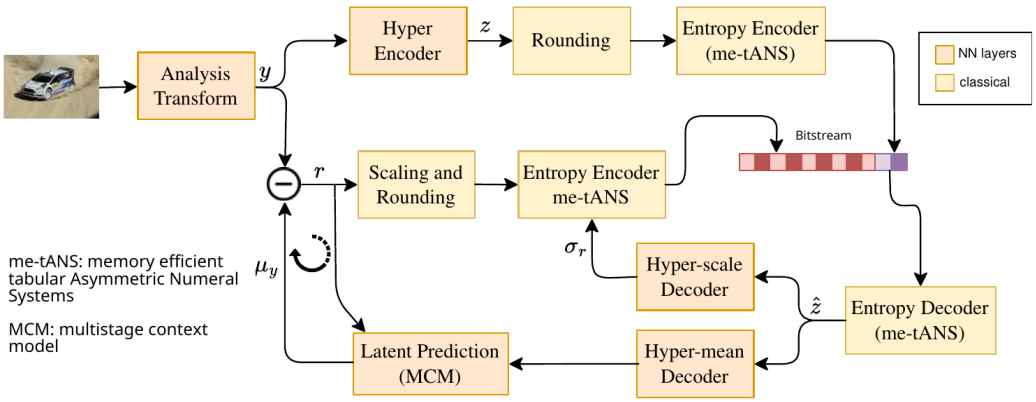


Figura 1.5: Schema della fase di encoding

latente y; questo viene elaborato dall'hyper-encoder che estrae da y il tensore z, successivamente arrotondato ($\hat{z}$) e compresso nell'ultima parte del bitstream. Per la codifica entropica di y viene usato un hyper-scale decoder, che riceve $\hat{z}$, e genera i parametri, per esempio le varianze, necessarie per descrivere la distribuzione di probabilità dei valori di y. Questi verranno usati per fare la codifica entropica dei residui, mentre un hyper-mean decoder produce una stima della media del tensore latente, usata nel modulo di Latent Prediction per stimare i valori di mu_y. Sottraendo μ_y al latente originale y, vengono calcolati i residui; dopo esser stati scalati e arrotondati, sono compressi nel bitstream usando me-tANS.

Fase di Decoding Nella fase di decoding (vedi fig. 1.6) avviene il processo inverso: inizialmente viene fatta la decodifica delle informazioni ausiliari $\hat{z}$, che vengono passate all'hyper-decoder (identico a quello della fase di encoding); i parametri calcolati da quest'ultimo vengono usati per decodificare in modo più efficiente i residui, ottenndo $\hat{r}$. Attraverso un modulo di latent prediction vengono nuovamente stimati $\hat{y}$, che è ricevuta dalla synthesis

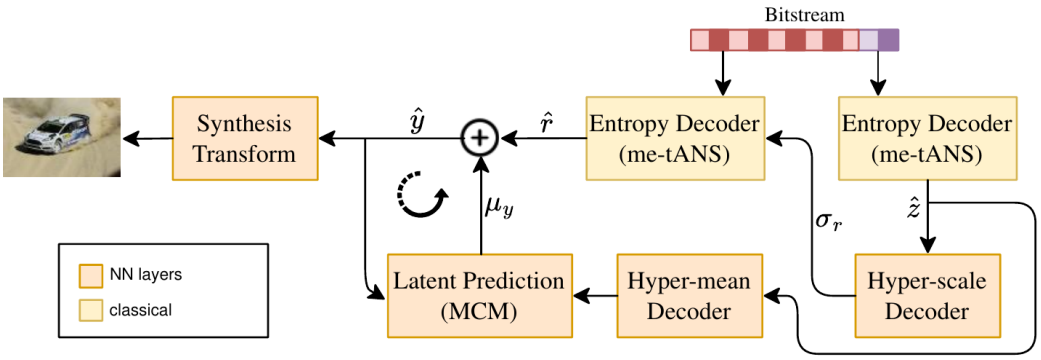


Figura 1.6: Fase di Decoding di JPEG AI

transform che produce un'immagine partendo dalla rappresentazione latente ricevuta.

Risultati ottenuti da JPEG AI I primi esperimenti hanno riscontrato risultati promettenti: con le prime versioni delle implementazioni JPEG AI è riuscito a raggiungere un BD-rate gain (*Bjøntegaard-Delta Rate*, una metrica per la valutazione dell'efficienza di compressione) del 28% [19] rispetto a VVC Intra (lo standard attdi compressione più effciente attualmente disponibile) usando la configurazione più semplice "*tools-off*" senza ottimizzazioni avanzate. Inoltre è stato notato anche un miglioramento nella qualità soggettiva a parità di bitrate [19], confermando il potenziale di questo strumento.

1.2.4 Conseguenze sulla multimedia forensics

La creazione di JPEG AI introduce nuove sfide nel campo della multimedia forensics. In [20] viene definito il termine *miscompression* per indicare gli errori creati nella ricostruzione delle immagini che portano ad avere dettagli semantici diversi tra l'immagine reale e quella ricostruita. Poichè l'architettura è simile a quella dei generatori di immagini sintetici, in [21] viene spiegato come le immagini compresse con JPEG AI presentano artefatti simili a quelle

false, che porta i detetctor esistenti a classificarli in modo errato rilevandole come immagini sintetiche. In [22] vengono proposti tre indizi forensi per il rilevamento e l'analisi di immagini compresse con JPEG AI: correlazioni di colore, quantizzazione nello spazio latente e analisi della rate-distortion.

Capitolo 2

Metodologia

2.1 JPEG AI Verification Model

Lo strumento utilizzato per ottenere la codifica JPEG AI è "jpeg-ai-reference-software" [23] che rappresenta il software di riferimento per la prima implementazione del sistema proposto chiamato *JPEG AI Verification Model* [24]. Sono proposti due encoder con diversi livelli di complessità: `Enc0`, il più semplice, progettato per dispositivi mobili, ed `Enc1`, più complesso, composto da *attention blocks* e *transformer*, necessitando quindi capacità computazionale elevata. Sono supportati tre diversi "operation point" SOP (*Simple Operation Point*), BOP (*Basic Operation Point*) e HOP (*High Operation Point*) ottimizzati rispettivamente per l'uso su dispositivi dotati di CPU, dispositivi mobili, e dispositivi dotati di hardware specializzato come le GPU. JPEG AI supporta la codifica a tasso variabile,

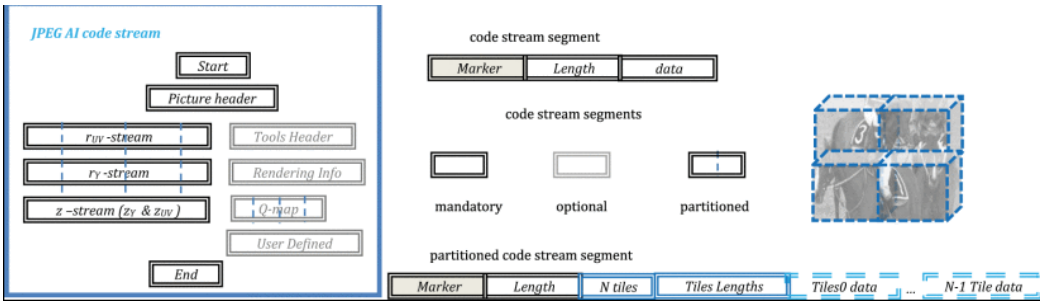


Figura 2.1: JPEG AI bitstream

2.2 Dataset utilizzato

Il dataset usato negli esperimenti è *140k Real and Fake Faces* [25], una raccolta di 140.000 immagini di volti, suddivise in 70.000 immagini reali e 70.000 immagini fake. I dati sono già suddivisi in training set, test set, e validation set, rispettivamente composti da 100.000, 20.000 e 20.000 immagini; ogni insieme mantiene un perfetto bilanciamento tra le due classi; queste caratteristiche rendono il dataset ideale per l'addestramento semplificando la preprarazione dei dati. Le immagini sono in formato JPEG e sono state ridimensionate in 256x256 pixel; sono inclusi file csv contententi le etichette e il percorso di ogni immagine.

Immagini reali Le immagini reali fanno parte del dataset *Flickr-Faces-HQ* (FFHQ) [26], il dataset creato per lo sviluppo di StyleGAN, e consistono in fotografie di volti umani di alta qualità (risoluzione a 1024x1024). Questo si distingue da altri dataset per la presenza di una grande varietà di soggetti per età ed etnia, ma anche per la presenza di accessori come occhiali o cappelli [5] rendendolo molto diversificato; queste immagini sono state collezionate da Flickr, un servizio web che offre la possibilità di pubblicare immagini ad artisti e fotografi.

Immagini sintetiche Le immagini fake invece fanno parte del dataset *1 Milion Fake Faces* ,un insieme di immagini generate tramite StyleGAN (vedi sez. 1.1.2).

2.3 Classificatori utilizzati

Sono stati scelti due diversi algoritmi di classificazione, Random Forest e GradientBoosting, entrambi appartenenti alla categoria degli *ensemble methods*.

2.3.1 Random Forest

Random Forest è un algoritmo di apprendimento supervisionato appartenente all'insieme dei "ensemble methods". Questi si basano sull'idea di combinare le predizioni di più modelli di base detti 'deboli' (*weak learners*) per creare un modello finale con maggior accuratezza e riducendo distorsione e varianza rispetto all'uso di un singolo modello. Nei random forest i modelli deboli sono alberi di decisione.

Alberi di decisione Gli alberi di decisione sono algoritmi di apprendimento automatico che rappresentano funzioni che mappano dati in input su un insieme di classi (per problemi di classificazione) attraverso una serie di test rappresentabili con strutture ad alberi. Una volta costruito l'albero di decisioni, la classificazione di un nuovo dato consiste semplicemente nel partire dalla radice e valutare l'esito di ogni test che si incontra. La costruzione si basa sul concetto di ripartizione ricorsiva dello spazio di input in sottinsiemi più piccoli in base ai valori degli attributi. La scelta dell'attributo da usare per la ripartizione viene fatta valutando il guadagno informativo derivato da

quella scelta, cercando di minimizzare l'impurità dei sottoinsiemi. L'aspetto critico di questa tecnica è l'alta varianza, infatti cambiamenti nei dati di addestramento influenzano molto la struttura dell'albero finale.

Random Forest I Random Forest [27] risolvono questo problema combinando la tecnica di *bagging* con la selezione casuale di attributi. Il bagging (Bootstrap Aggregating) consiste nel creare K diversi sottoinsiemi campionando con reinserimento (*Bootstrap*) dall'insieme di addestramento originale. Su ogni sottoinsieme verrà addestrato un modello base, e la predizione finale viene fatta aggregando (*Aggregating*) le predizioni dei K modelli; per i problemi di classificazione si può usare la media o il voto di maggioranza

$$y = \frac{1}{k} \sum_{i=1}^{k} y_i \tag{2.1}$$

dove y_i è la predizione del modello i. Il bagging permette una riduzione della varianza, ma presenta un problema riguardante la forte correlazione che i modelli base tendono ad avere, sopratutto quando un attributo risulta avere un guadagno informativo più alto, diventando radice per molti alberi.
Nei Random Forest questo problema viene risolto variando in ogni nodo l'insieme di attributi tra cui poter scegliere quello su cui poi fare il test, garantendo una bassa correlazione tra gli alberi.

2.3.2 Gradient Boosting

Il Gradient Boosting è un algoritmo di apprendimento supervisionato molto popolare per problemi nei quali i dati sono tabellari. La differenza principale rispetto ai random forest risiede nell'uso del *boosting* invece del bagging. Il boosting introduce l'idea insieme di addestramento pesato, dove ogni esempio ha un peso che indica la sua importanza nell'addestramento.

L'idea è quella di allenare un insieme di modelli deboli in modo sequenziale: inizialmente ad ogni esempio viene assegnato lo stesso peso, e su questo insieme viene addestrato il primo modello. Prima di allenare il modello successivo, i pesi degli esempi vengono aggiornati facendo sì che il peso degli esempi classificati in errato viene aumentato, mentre quello degli esempi classificati correttamente viene ridotto. Iterando il processo per K iterazioni vengono prodotti K modelli deboli tra i quali almeno un modello riuscirà a classificare in modo corretto anche gli esempi più difficili.
Per fare la predizione, ogni modello debole vota, e la predizione finale viene fatta considerando il voto di maggioranza.
Nel Gradient Boosting, il processo di boosting non si basa sui pesi, ma sul gradiente negativo della funzione di perdita: si parte con una funzione di perdita differenziabile, si fanno le predizioni e si calcola per ogni esempio il gradiente negativo della funzione di perdita. Con questa informazione viene addestrato un nuovo modello debole aggiornando i parametri muovendosi nella direzione del gradiente negativo.

2.3.3 Hyperparameter Tuning

Le prestazioni dei modelli di apprendimento spesso dipendono dalla scelta degli iperparametri, parametri che riguardano l'architettura/classe dei modelli. Questi non vengono appresi automaticamente, ma anzi sono considerabili come variabili di configurazione da specificare prima di procedere con l'addestramento. Per questo è buona pratica adottare la tecnica di *hyperparameter tuning* per trovare la combinazione di iperparametri che offre migliori prestazioni, che verrano poi utilizzate per creare il modello finale da allenare. La ricerca viene fatta scegliendo modello e lo spazio dei valori degli iperparametri da esplorare. Negli esperimenti sono state usate la ricerca a

griglia (*GridSearch*) e la ricerca casuale (*Random Search*).
La ricerca a griglia che compie una ricerca esaustiva sulla griglia degli iperparametri specificati, generando un modello per ogni combinazione possibile. Questa, anche se molto onerosa dal punto di vista computazionale, garantisce di trovare la combinazione ottimale tra quelle specificate. La ricerca casuale invece effettua un campionamento di un numero finito di parametri seguendo una distribuzione scelta. Questa tecnica risulta molto meno dispendiosa rispetto alla ricerca a griglia, riuscendo comunque a trovare buone combinazioni di iperparametri; viene usata principalmente quando lo spazio di ricerca ha alta dimensionalità.
Per la valutazione dei modelli viene utilizzata la *k-fold cross-validation*, un'operazione che suddivide l'insieme di addestramento in k sottoinsiemi, e addestra il modello su k-1 sottoinsiemi per poi testarlo su quello rimanente; il modello scelto sarà quello che ha ottenuto una migliore accuratezza.

2.4 Addestramento dei classificatori

2.4.1 Rappresentazioni latenti estratte da JPEG AI

Le immagini prima di essere elaborate dall'encoder vengono pre-processate: inizialmente sono convertite portando negli spazio di colori YUV BT.709, separando le componenti di luminanza e crominanza, e successivamente viene applicato la *Conditional Color Separation* [28], un approccio per cui la componente primaria, la luminanza (Y), viene compressa con una rete neurale più potente, mentre le componenti di crominanza (UV) vengono compresse usando informazioni della luminanza. La separazione delle componenti è mantenuta anche nel bitstream finale 2.1 dando la possibilità di scartare la parte di crominanza se non necessaria per l'applicazione scelta.

L'encoder genera un dizionario che raccoglie tutti i valori necessari utilizzati nel resto dell'architettura (vedi fig. 1.4), ma quelli di nostro interesse sono y ed $\hat{y}$, entrambi composti da tensori `model-y` e `model-uv` per rappresentare componenti di luminanza e crominanza separatamente come voluto da meccanismo di CCS. Questi tensori hanno rispettivamente dimensione (160, 16,16) per la luminanza e (96, 16, 16) per la crominanza.
y rappresenta l'output dell'*Analysis transform* (vedi fig.1.5) mentre $\hat{y}$ rappresenta l'input dell'*Synthesis transform* (vedi fig. 1.6).

2.4.2 Preprocessing dei latenti

Per allenare i modelli di classificazione scelti è necessario trasformare i tensori prodotti dall'encoder in vettori unidimensionali; si sono state utilizzate diverse strategie.

Flattening La prima strategia testata è stata quella più semplice ed intuitiva da realizzare, trattandosi di una semplice operazione di *flattening* del tensore: dopo aver concatenato le componenti di luminanza e crominanza, ottenendo un tensore di dimensione $(256, 16, 16)$, viene trasformato in un vettore di dimensione $(65536,)$. Questa decisione, come si vedrà negli esperimenti, presenta un problema dovuto al fatto che i campioni ottenuti hanno una dimensione molto elevata, rendendo l'addestramento dei modelli lento e dispendioso nelle risorse.

Estrazione di patch La seconda strategia è stata quella di estrarre delle patch dalle rappresentazioni latenti, dove ogni patch è la concatenazione dei valori presenti nella stessa posizione spaziale (x,y) per tutti i canali (di dimensione 16x16) del tensore. In questo modo si ottiene un vettore di dimensione

inferiore rispetto all'uso della tecnica di flattening. Questo approccio permette di aumentare il numero di campioni disponibili per l'addestramento, e renderà l'addestramento più rapido.

Capitolo 3

Esperimenti

3.1 Specifiche hardware e software

Gli esperimenti sono stati condotti tramite connessione ssh ad un server del laboratorio *LESC - Signal Processing & Communications LAB* dell'Università degli Studi di Firenze; la macchina utilizzata presenta le seguenti caratteristiche:

- CPU: Intel Core i9-9940X @ 3.30GHz, architettura x86_64
- RAM: 125 GB
- GPU: 2 NVIDIA GeForce RTX 2080 Ti
- Sistema Operativo: Ubuntu 22.04.5 LTS (Jammy)
- Python: 3.7.16
- Librerie principali utilizzate:
  - PyTorch 1.10.2
  - scikit-learn 1.0.2

- NumPy 1.19.1
- Pandas 1.3.5

Le GPU presenti sono state fondamentali per svolgere l'estrazione delle rappresentazioni latenti delle immagini tramite l'uso dell'encoder neurale di JPEG AI in tempi rapidi, mentre l'addestramento e valutazione dei modelli di classificazione sono stati completamente svolti sulle CPU disponibili.

3.2 Codice sorgente del lavoro

Il codice sorgente sviluppato è reso disponibile tramite GitHub all'indirizzo `https://github.com/edorustichini/thesis.git`. L'intero progetto è sviluppato in Python, e nella cartella `src` sono presenti gli script per il setup degli esperimenti, metodi per processare i dati, e script per l'addestramento e valutazione dei modelli. Sono inoltre presenti dei notebook Jupyter per l'analisi dei risultati.
L'estrazione delle rappresentazioni latenti delle immagini viene fatta utilizzando il software ufficiale di riferimento di JPEG AI, disponibile all'indirizzo `https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software`.

3.3 Esperimenti condotti

Come dichiarato nelle sezioni precedenti, l'obbiettivo del lavoro consiste nel verificare l'efficacia della codifica JPEG AI nel compito di identificazione di immagini generate.
Gli esperimenti sono stati condotti per ogni classificatore scelto variando 3 fattori principali: metodo di preprocessing delle feature estratte, componente

scelta e `target_bpp`. Nella tabella 3.1 è riportato un riepilogo dei fattori considerati.

Tabella 3.1: Fattori considerati durante gli esperimenti

Fattore	**Valori / Descrizione**
Compressione (Bpp)	12bpp, 6bpp. Usati per capire l'effetto del livello di compressione sui risultati
Componente target	Y e YUV - Y: classificatori allenati esclusivamente sulla componente della luminanza. - YUV: classificatori allenati su tutte e tre le componenti (Y, U, V) concatenandole.
Preprocessing	Come descritto in 2.4.2 sono stati esplorati diversi metodi di preprocessing dei latenti prima dell'addestramento: Flatten, Single Patch e Multiple Patches.

Il metodo di preprocessing delle feature influenza la struttura del dataset di addestramento e le modalità di valutazione del modello. Nella tabella 3.2 viene proposta una schematizzazione dei metodi.

Tabella 3.2: Confronto tra metodi di preprocessing delle feature.

Method	**Samples per Image**	**Feature Dimension**
Flatten Latents (Y)	1	$C_y \cdot H \cdot W = 40960$
Flatten Latents (YUV)	1	$(C_y + C_{uv}) \cdot H \cdot W = 65536$
Single Patch (Y)	1	$C_y = 160$
Single Patch (YUV)	1	$C_y + C_{uv} = 256$
N Patches (Y)	N	$C_y = 160$
N Patches (YUV)	N	$C_y + C_{uv} = 256$

Nota: H, W sono le dimensioni di ogni canale latente, C_y, C_{uv} sono il numero di canali della componente Y e delle componenti U e V rispettivamente.

Nota sul metodo N Patches Per il metodo N Patches, la fase di testing viene svolta in maniera differente rispetto alle altre: nella fase di addestramento, per ogni immagine vengono estratte N patches ottenendo un insieme di addestramento di cardinalità $(N \cdot |\mathcal{D}|)$ (dove $\mathcal{D}$ è il dataset originale). Per la fase di testing invece si utilizza la strategia di *majority voting*: per ogni immagine vengono estratte tutte le patch, ed il classificatore produce una predizione indipendente su ognuna di queste che si può pensare come una votazione. La predizione finale sarà la classe più frequente tra tutte le votazioni.

3.3.1 Metodo di addestramento

Come descritto nella sez. 2.3.3, per ogni esperimento è stato svolto il tuning degli iperparametri: inizialmente è stata applicata una ricerca casuale su un intervallo di valori ampio; osservando i risultati ottenuti, è

stata svolta una ricerca a griglia ristretta sui migliori 3 valori trovati per ogni iperparametro. Questo procedimento è stato ripetuto a dimensioni diverse $(15.000, 30.000, 50.000)$ del dataset di addestramento, e i parametri migliori sono stati usati per allenare il modello finale su 30.000 immagini, e poi valutarlo su 3.000 immagini di test. La libreria scelta per gli algoritmi di machine learning è Scikit-learn, libreria open-source scritta in Python, che offre strumenti semplici ma efficienti per molte task, tra cui quelle di classificazione. In particolare, per Random Forest e Gradient Boosting sono stati scelti i modelli nominati `RandomForestClassifier` e `HistGradientBoostingClassifier`; mentre per gli algoritmi di ricerca degli iperparametri sono stati usati `RandomizedSearchCV` e `GridSearchCV`.

3.4 Risultati

I risultati verrano presentati in sezioni separate per tipo di classificatore, e per ognuno vengono mostrate le varie combinazioni dei fattori descritti nella sez. 3.3, presentando prima una tabella con i risultati per quel modello, e poi diagrammi a barre per la visuallizzazione. Infine verrà fatto un confronto tra le prestazioni dei due classificatori.

3.4.1 RandomForest

Al termine del tuning degli iperparametri, la combinazione che ha avuto un `cv-score` più alto è:

```
1 RandomForest(random_state=42, oob_score=True,
2 verbose=0, n_estimators=463, max_depth=30,
    max_features='sqrt',
```

```
3       min_samples_leaf=2)
```

Listing 3.1: Miglior combinazione di parametri per RandomForestClassifier

Nella tabella 3.3 sono riportati i risultati sull'accuratezza ottenuti dal modello scelto negli esperimenti.

Tabella 3.3: Risultati per Random Forest

Modello	Bpp	Comp.	process.	y	y_hat
RF	6	Y	single	0.683	0.683
RF	6	Y	multiple	0.566	0.564
RF	6	Y	flatten		
RF	12	Y	multiple	0.568	0.560
RF	12	Y	single	0.683	0.682
RF	12	Y	flatten		
RF	6	YUV	flatten	0.820	0.811
RF	6	YUV	single	0.720	0.716
RF	6	YUV	multiple	0.596	0.588
RF	12	YUV	flatten	0.824	0.804
RF	12	YUV	single	0.720	0.718
RF	12	YUV	multiple	0.560	0.567

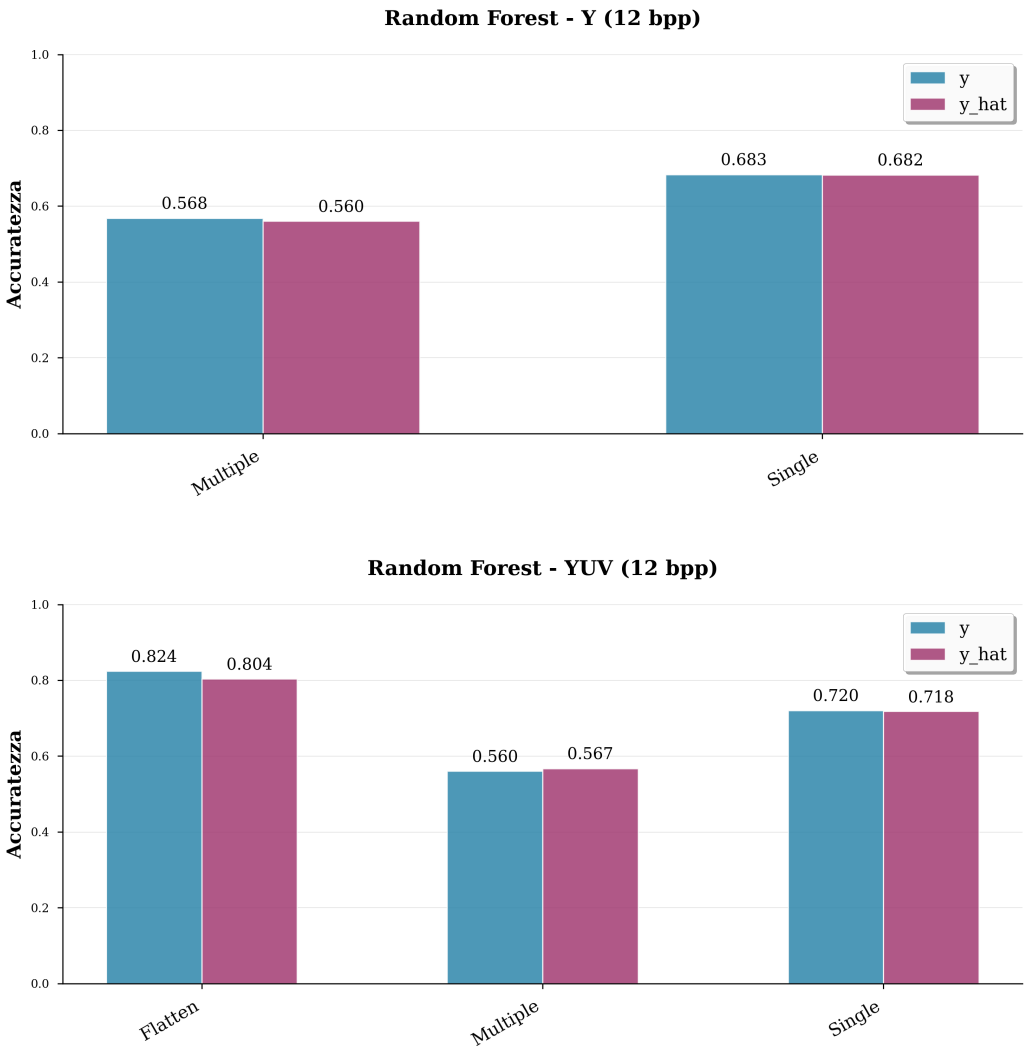


Figura 3.1: Confronto tra prestazioni di due Random Forest diversi allenati rispettivamente su Y e YUV.

Dalla figura 3.1 si nota come le prestazioni del Random Forest allenato su YUV siano migliori rispetto a quello allenato solo su Y, in particolare nel caso del metodo di processing "single patch".

3.4.2 GradientBoosting

La combinazione migliore di iperparametri trovata per l'algoritmo di Gradient Boosting è:

```
1     GradientBoostingClassifier(
2         learning_rate=0.12965912630431367,
3         max_iter=596,
4         max_leaf_nodes=117,
5         min_samples_leaf=18,
6         l2_regularization=0.003042734607209594,
7         random_state=42,
8         validation_fraction=0.1)
```

Nella tabella (3.4) sono riportati i risultati sull'accuratezza ottenuti dal modello scelto negli esperimenti, e di seguito sono riportati alcuni diagrammi a barre per la visualizzazione.

Tabella 3.4: Risultati per Gradient Boosting

Modello	Bpp	Comp.	process.	y	y_hat
GB	6	Y	single	0.696	0.688
GB	6	Y	multiple	0.543	0.535
GB	6	Y		0.936	
GB	12	Y	single	0.695	0.700
GB	12	Y	multiple	0.554	0.534
GB	12	Y	flatten		
GB	6	YUV	flatten	0.940	0.940
GB	6	YUV	single	0.744	0.753
GB	6	YUV	multiple	0.520	0.540
GB	12	YUV	flatten	0.944	0.940
GB	12	YUV	single	0.744	0.744
GB	12	YUV	multiple	0.540	0.534

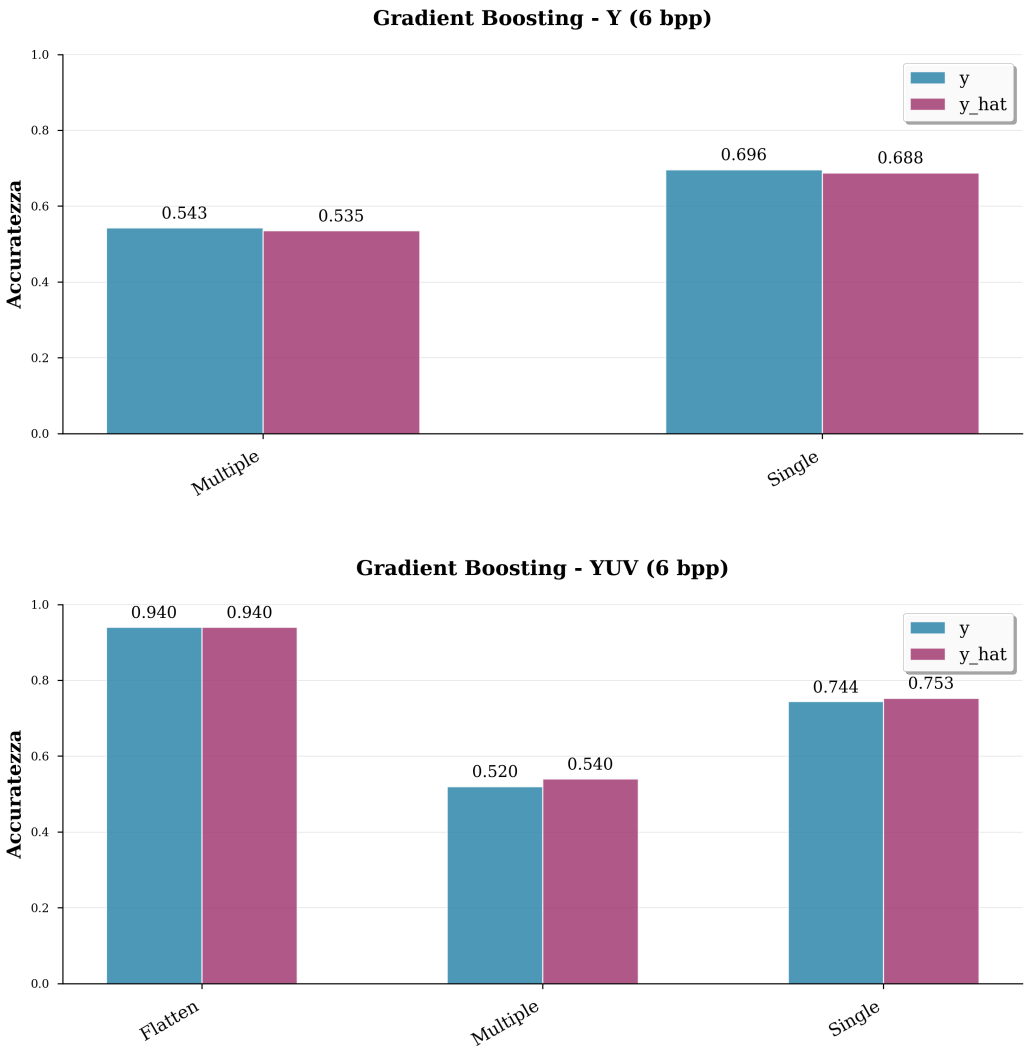
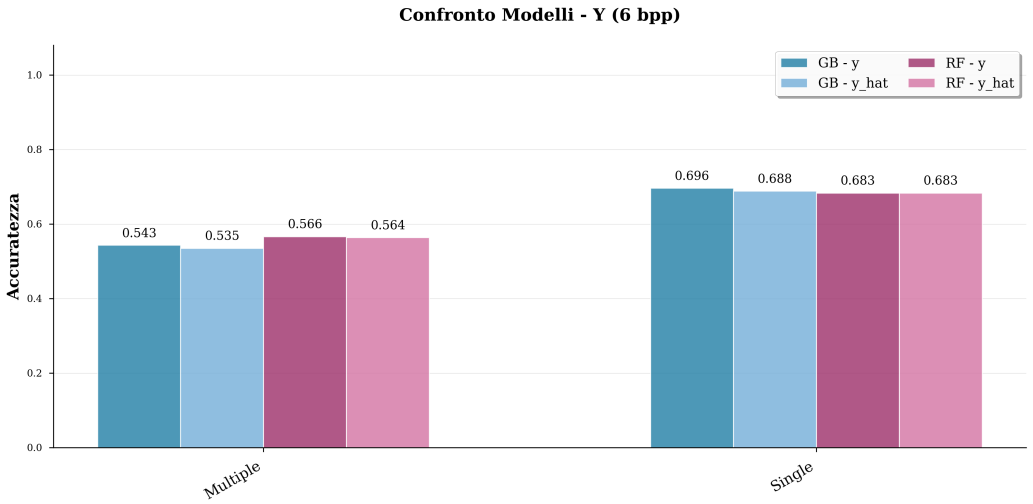
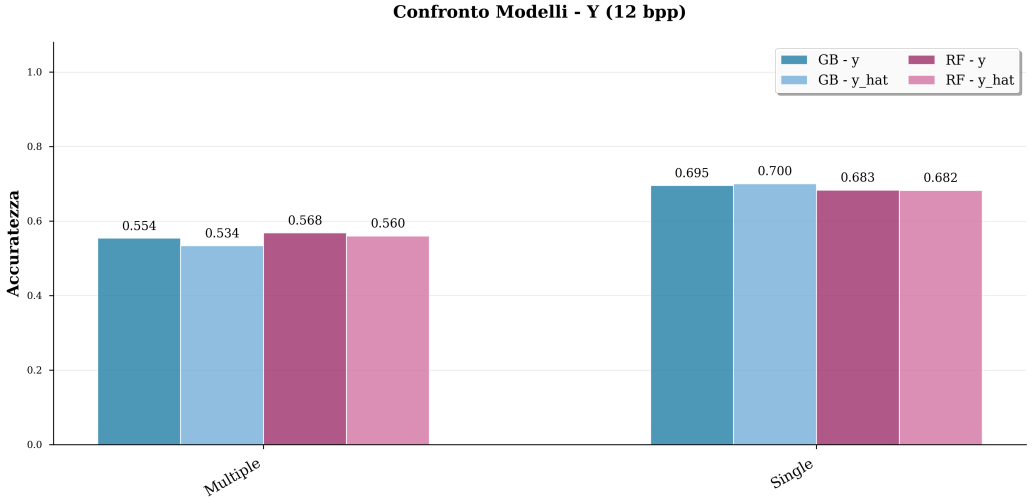


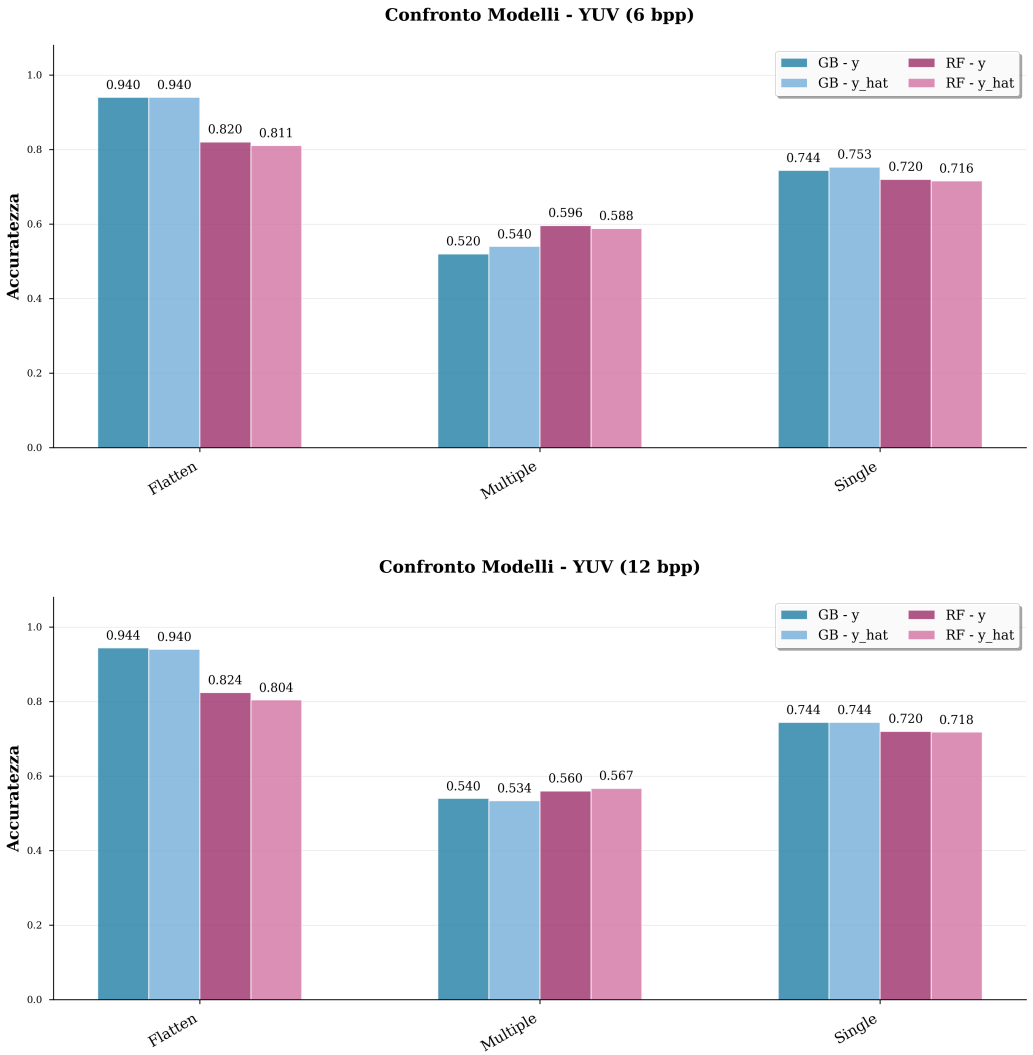
Figura 3.2: Esempi di diagrammi a barre riguardanti performance di Gradient Boosting

3.4.3 Confronto tra classificatori

Nei diagrammi seguenti viene mostrato il confronto diretto tra le prestazioni dei due classificatori a diversi livelli di compressione.


Confronto Modelli - Y (6 bpp)
GB - y
GB - y_hat
RF - y
RF - y_hat
Accuratezza
0.0
0.2
0.4
0.6
0.8
1.0
0.543
0.535
0.566
0.564
0.696
0.688
0.683
0.683
Multiple
Single


Confronto Modelli - Y (12 bpp)
GB - y
GB - y_hat
RF - y
RF - y_hat
Accuratezza
0.0
0.2
0.4
0.6
0.8
1.0
0.554
0.534
0.568
0.560
0.695
0.700
0.683
0.682
Multiple
Single



Si può notare come l'accuratezza non subisca grandi varianzioni ai diversi livelli di compressione scelti, mentre è evidente la differenza tra le performance dei modelli allenati esclusivamente sulla componente di luminanza e quelli allenati anche usando la crominanza; questo è ragionevole in quanto la differenza risiede nel fatto che nel secondo caso il classificatore ha più informazioni da sfruttutare nella classificazione.

Inoltre, grazie al confronto tra i due algoritmi, si nota come Gradient Boosting ottenga un'accuratezza in generale più alta rispetto alla variante Ran-

dom Forest.

Infine, la scelta del metodo con cui processare le feature ha un impatto significativo sulle prestazioni. Il metodo flatten a livello di prestazioni risulta il migliore però, come espresso nella tabella 3.2, i dati dell'insieme di addestramento hanno un'elevata dimensionalità rispetto agli altri metodi, il che comporta un aumento dei tempi di addestramento e valutazione e una maggior necessità di risorse computazionali.

Capitolo 4

Conclusioni

Bibliografia

[1] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. *Information Fusion*, 64:131–148, 2020.

[2] Tharindu Fernando, Darshana Priyasad, Sridha Sridharan, Arun Ross, and Clinton Fookes. Face deepfakes–a comprehensive review. 2025.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[4] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[5] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[6] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset, 2020.

[7] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[10] Zhendong Wang, Jianmin Bao, Wengang Zhou, Weilun Wang, Hezhen Hu, Hong Chen, and Houqiang Li. Dire for diffusion-generated image detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22445–22455, 2023.

[11] Ali Khodabakhsh, Raghavendra Ramachandra, Kiran Raja, Pankaj Wasnik, and Christoph Busch. Fake face detection methods: Can they be generalized? In *2018 international conference of the biometrics special interest group (BIOSIG)*, pages 1–6. IEEE, 2018.

[12] Gregory K Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[13] Sonehara, Kawato, Miyake, and Nakane. Image data compression using a neural network model. In *International 1989 Joint Conference on Neural Networks*, pages 35–41. IEEE, 1989.

[14] GL Sicuranza, GIOVANNI Ramponi, and Stefano Marsi. Artificial neural network for image compression. *Electronics letters*, 26(7):477–479, 1990.

[15] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[16] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. 2018.

[17] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

[18] J Ascenso and P Akayzi. Report on the state-of-the-art of learning based image coding. *ISO/IEC JTC 1/SC29/WG1, Geneva, Document N83058*, 2019.

[19] João Ascenso, Elena Alshina, and Touradj Ebrahimi. The jpeg ai standard: Providing efficient human and machine visual data consumption. *Ieee Multimedia*, 30(1):100–111, 2023.

[20] Nora Hofer and Rainer Böhme. A taxonomy of miscompressions: Preparing image forensics for neural compression. In *2024 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2024.

[21] Edoardo Daniele Cannas, Sara Mandelli, Nataša Popović, Ayman Alkhateeb, Alessandro Gnutti, Paolo Bestagini, and Stefano Tubaro. Is jpeg

ai going to change image forensics? *arXiv preprint arXiv:2412.03261*, 2024.

[22] Sandra Bergmann, Fabian Brand, and Christian Riess. Three forensic cues for jpeg ai images. *arXiv preprint arXiv:2504.03191*, 2025.

[23] JPEG / JPEG AI / JPEG AI Reference Software · GitLab. https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software, July 2025.

[24] ICQ. Description of the JPEG AI verification model under consideration and associated software integration procedure. Technical Report wg1n100279, ISO/IEC JTC 1/SC 29/WG 1, 2022.

[25] 140k Real and Fake Faces. https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces.

[26] NVlabs/ffhq-dataset. NVIDIA Research Projects, August 2025.

[27] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[28] Panqi Jia, Ahmet Burakhan Koyuncu, Georgii Gaikov, Alexander Karabutov, Elena Alshina, and André Kaup. Learning-based conditional image coder using color separation. In *2022 Picture Coding Symposium (PCS)*, pages 49–53, 2022.