



UNIVERSITÀ TELEMATICA INTERNAZIONALE  
UNINETTUNO

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in  
Ingegneria Informatica

ELABORATO FINALE

in

Big Data

**Face Studio: a GAN based approach for  
designing faces**

RELATORE

**Prof. Luigi Laura**

CANDIDATO

**Marco Parrillo**

ANNO ACCADEMICO 2021/2022



*Ringrazio me stesso*

# Indice

<b>Elenco delle figure</b>	<b>VI</b>
<b>Elenco delle tabelle</b>	<b>IX</b>
<b>Elenco dei listati</b>	<b>X</b>
<b>1 Panoramica sul Deep Learning</b>	<b>1</b>
1.1 Relazione tra Intelligenza Artificiale, Machine Learning, Neural Networks e Deep Learning . . . . .	1
1.2 Perché il Machine Learning . . . . .	4
1.3 Il sistema cognitivo . . . . .	5
1.4 Reti neurali artificiali nel corso degli anni . . . . .	7
1.5 Approcci di apprendimento . . . . .	10
<b>2 Perché il Deep Learning</b>	<b>12</b>
2.1 Machine Learning vs. Deep Learning: Limite Decisionale . . . . .	12
2.2 Machine Learning vs. Deep Learning: Feature Engineering . . . . .	13
2.3 Diversi tipi di reti neurali nel Deep Learning . . . . .	14
2.4 Cosa é l'Artificial Neural Networks (ANN) e le sue applicazioni . . . . .	15
2.4.1 Artificial Neural Networks (ANN) . . . . .	16
2.4.2 Sfide con Artificial Neural Networks (ANN) . . . . .	16
2.5 Cosa é Recurrent Neural Network (RNN) e le sue applicazioni . . . . .	18

2.5.1	Vantaggi delle Recurrent Neural Network (RNN) . . . . .	18
2.5.2	Sfide con le Recurrent Neural Network (RNN) . . . . .	19
2.6	Cosa sono le Convolution Neural Network (CNN) e le sue applicazioni	20
2.6.1	Vantaggi delle Convolution Neural Network (CNN) . . . . .	21
2.7	MPL(ANN) vs RNN vs CNN . . . . .	22
<b>3</b>	<b>Convolution Neural Network (CNN)</b>	<b>23</b>
3.1	Rappresentazione uniforme di dati eterogenei . . . . .	24
3.2	Struttura di una CNN . . . . .	25
3.2.1	Convolutional Layer . . . . .	26
3.2.2	Pooling Layer . . . . .	27
3.2.3	Flattening Layer . . . . .	29
3.2.4	Fully-Connected Layers . . . . .	29
<b>4</b>	<b>Generative Adversarial Networks</b>	<b>31</b>
4.1	Generative model . . . . .	31
4.2	Struttura del GAN . . . . .	32
4.2.1	Interazione . . . . .	33
4.2.2	Comportamento . . . . .	35
4.2.3	Il training . . . . .	36
<b>5</b>	<b>StyleGAN e StyleGAN2</b>	<b>39</b>
5.1	Background . . . . .	40
5.2	StyleGAN dettagli . . . . .	43
5.3	Mapping Network . . . . .	44

5.4	Style Modules (AdaIN) . . . . .	45
5.5	Rimuovere gli input tradizionali . . . . .	46
5.6	Variazione Stocastica . . . . .	46
5.7	Mix di stili . . . . .	47
5.8	Sapere quando fermarsi . . . . .	48
5.9	Tuning . . . . .	48
5.10	StyleGAN2 . . . . .	49
<b>6</b>	<b>Esplorare lo spazio latente</b>	<b>54</b>
6.1	Background . . . . .	54
6.2	Compressione dei dati . . . . .	54
6.3	Definizione di spazio . . . . .	56
6.4	Concetto di simili . . . . .	57
6.5	Importanza dello spazio latente . . . . .	58
6.5.1	Autoencoder e Modelli Generativi . . . . .	58
6.5.2	Interpolazione dello spazio latente . . . . .	60
<b>7</b>	<b>Caso Sperimentale: Applicazione dello spazio latente a StyleGan2</b>	<b>62</b>
7.1	Background . . . . .	63
7.1.1	Python . . . . .	63
7.1.2	Google Colab . . . . .	64
7.1.3	TensorFlow . . . . .	65
7.2	Implementazione . . . . .	66
7.2.1	Caricamento Google Drive . . . . .	68

7.2.2	Download del codice sorgente . . . . .	68
7.2.3	Installazione librerie . . . . .	68
7.2.4	Installazione TensorFlow . . . . .	69
7.2.5	Import Librerie e Setup ambiente . . . . .	69
7.2.6	Preparazione dello StyleGAN2 . . . . .	70
7.2.7	Caricamento del modello . . . . .	72
7.2.8	Pulizia del Driver . . . . .	72
7.2.9	Generazione dei volti . . . . .	73
7.2.10	Immagini Custom . . . . .	74
7.2.11	Controlli latenti . . . . .	79
7.2.12	Generazione del Widget . . . . .	80
<b>8</b>	<b>Conclusioni</b>	<b>91</b>
<b>Riferimenti bibliografici</b>		<b>93</b>

# Elenco delle figure

1.1	Gerarchia dell'IA . . . . .	1
1.2	Reti non profonde vs nurali profonde . . . . .	4
1.3	Deep Learning Time Line . . . . .	8
2.1	Limite Decisionale . . . . .	13
2.2	Dati non lineari . . . . .	13
2.3	Machine Learning vs Deep Learning . . . . .	14
2.4	ANN . . . . .	15
2.5	Perceptron . . . . .	16
2.6	Backpropagation . . . . .	17
2.7	ANN vs CNN . . . . .	18
2.8	Sequence . . . . .	19
2.9	Unfolded RNN . . . . .	19
2.10	Vanish Gradient . . . . .	20
2.11	Convoluzione . . . . .	20
2.12	CNN . . . . .	21
2.13	Applicazione dei filtri . . . . .	22
3.1	LeNet-5 Architettura . . . . .	25
3.2	CNN . . . . .	25
3.3	Convoluzione . . . . .	26

3.4	Max Pooling . . . . .	28
3.5	Flattering . . . . .	29
3.6	Connected Layers . . . . .	30
4.1	Immagine generata con StyleGan2 . . . . .	32
4.2	Architettura GAN . . . . .	33
4.3	Discriminator-Generator Architettura . . . . .	33
4.4	GAN Algorimto . . . . .	36
4.5	Discriminante Generatore con backpropagation . . . . .	36
5.1	StyleGAN mix di stili . . . . .	40
5.2	GAN overview . . . . .	41
5.3	PROGAN . . . . .	42
5.4	PROGAN Training . . . . .	43
5.5	PROGAN Mapping Network . . . . .	44
5.6	ADAIN . . . . .	45
5.7	Input Constante . . . . .	46
5.8	Rumore . . . . .	47
5.9	Unione di 2 immagini. L'immagine al centro é il risultato dell'unione tra l'immagine di destra e l'immagine di sinistra . . . . .	48
5.10	StyleGan Overview . . . . .	49
5.11	Difetti immagini generate con StyleGAN . . . . .	50
5.12	Evoluzione architettura StyleGAN . . . . .	50
5.13	Immagini demodulate . . . . .	51
5.14	Risultati StyleGAN2 . . . . .	52

5.15	Occhi e naso restano fissi . . . . .	52
5.16	Generatore parte superiore e discriminantore parte inferiore . . . . .	53
6.1	Compressione dei dati . . . . .	55
6.2	Rappresentazione di una CNN . . . . .	55
6.3	Matrice 5x5x1 . . . . .	56
6.4	Matrice 3x1 . . . . .	56
6.5	Esempio di simili . . . . .	57
6.6	Autoencoder . . . . .	59
6.7	Immagini generate usando lo spazio latente . . . . .	60
6.8	Interpolazione di dati, l'immagine centrale é stata generata interpolando le immagini di destra e sinistra . . . . .	61
7.1	Architettura GAN . . . . .	71
7.2	Immagine Generata con StyleGAN2 . . . . .	74
7.3	Immagine originale vs immagine allineata . . . . .	76
7.4	Immagine originale vs immagine Generata . . . . .	78
7.5	Widget . . . . .	86
7.6	Widget - invecchiato . . . . .	86
7.7	Widget - femminile . . . . .	87
7.8	Widget - senza sorriso . . . . .	87
7.9	Widget - senza sorriso . . . . .	88
7.10	Widget - al femminile, invecchiato, bocca aperta . . . . .	88
7.11	Widget - dettaglio . . . . .	89
7.12	Widget - con immagine generata dal modello . . . . .	90

# Elenco delle tabelle

2.1 MPL(ANN) vs RNN vs CNN . . . . .	22
--------------------------------------	----

# Elenco dei listati

7.1	Caricamento Google Drive . . . . .	68
7.2	Download codice sorgente . . . . .	68
7.3	Installazione librerie . . . . .	68
7.4	Installazione TensorFlow . . . . .	69
7.5	Import Librerie e Setup ambiente . . . . .	69
7.6	Preparazione dello StyleGAN2 . . . . .	71
7.7	Caricamento del modello . . . . .	72
7.8	Pulizia del Driver . . . . .	72
7.9	Generazione dei volti 1/2 . . . . .	73
7.10	Output Generazione dei volti 1/2 . . . . .	73
7.11	Porzione del contenuto del vettore latente . . . . .	73
7.12	Generazione dei volti 2/2 . . . . .	73
7.13	Import immagini nell'ambiente di lavoro . . . . .	75
7.14	Allineamento dell'immagine . . . . .	75
7.15	Creazione del dataset StyleGAN2 . . . . .	76
7.16	Output Creazione del dataset StyleGAN2 . . . . .	76
7.17	Proiezione nello spazio latente . . . . .	76
7.18	Vettore latente generato . . . . .	77
7.19	Output Vettore latente generato . . . . .	77
7.20	Immagine originale vs generata . . . . .	77
7.21	Direzioni latenti . . . . .	80
7.22	Output Direzioni latenti . . . . .	80
7.23	Creazione del Widget . . . . .	81

# Sommario

Fin dall'inizio le reti neurali hanno avuto una grande rilevanza nel campo dell'Intelligenza Artificiale e oggigiorno gli algoritmi di Deep Learning vengono ampiamente utilizzati in diversi ambiti per analizzare ed estrarre conoscenza da grandi quantità di dati.

Questa tesi vole dare una panoramica sull'evoluzione del Deep Learning fino ad arrivare ad una delle recenti implementazioni ritenuta tra le piú interessanti, attorno la quale si stanno sviluppando diversi ambiti di ricerca e discussioni, ovvero le reti GAN (Generative Adversarial Network) e in particolare lo StyleGAN2.

Il lavoro proposto prende spunto dal sito web <https://www.thispersondoesnotexist.com/>, dove ad ogni refresh della pagina appare un nuovo volto generato a runtime, ma questo non é il solo ambito di applicazione in quanto le GAN posso essere applicate a diversi domini di ricerca e studio.

La tesi si conclude con l'implementazione di una semplice applicazione scritta in Python che vuole mettere in pratica i concetti affrontati durante lo svolgimento della documento, mostrare la qualità delle immagini generate, la semplicitá con cui é possibile implementare questi tipi di reti, ma anche dare spunti su possibili miglioramenti o studi futuri.

Il documento proposto é composto dai seguenti capitoli:

- **Capitolo 1:** Questo capitolo vuole essere una introduzione al concetto di Deep Learning, spiegando la relazione che intercorre tra Intelligenza Artificiale, Machine Learning, Reti Neurali e Deep Learning, il perché é cosi importante e la sua evoluzione nel corso degli anni.
- **Capitolo 2:** Questo capitolo spiega perché il Deep Learning nella forma di: convolutional neural networks (CNN), recurrent neural networks (RNN) e artificial neural networks (ANN), viene preferito agli algoritmi di Machine Learning.

- **Capitolo 3:** Questo capitolo approfondisce le Convolution Neural Network (CNN), in particolare andremo ad analizzare il perché sono così importanti nel campo dell’Image Processing e Computer Vision, e il valore aggiunto che questi tipi di reti portano.
- **Capitolo 4:** Questo capitolo affronta le Generative Adversarial Networks (GAN) introdotte da Ian J. Goodfellow che sono la base su cui questa tesi viene sviluppata.
- **Capitolo 5:** Questo capitolo mette in relazione lo StyleGAN con lo StyleGAN2, partendo dallo StyleGAN conclude con i miglioramenti apportati dallo StyleGAN2.
- **Capitolo 6:** Questo capitolo affronta il concetto di spazio latente cercando di spiegarne la sua importanza e la sua utilità in ambito Deep Learning.
- **Capitolo 7:** Questo ultimo capitolo è dedicato a una delle possibili applicazioni pratiche dello StyleGAN2.
- **Conclusioni:** Questo capitolo è dedicato alle conclusioni in cui si analizza ciò che è stato implementato, dando spunti su possibili sviluppi futuri.

# Summary

From the beginning, neural networks have had a great importance in the field of Artificial Intelligence and nowadays Deep Learning algorithms are widely used in various fields to analyze and extract knowledge from large amounts of data.

This thesis aims to give an overview of the evolution of Deep Learning up to one of the recent implementations considered to be among the most interesting, around which various areas of research and discussions are being developed, namely the GAN (Generative Adversarial Network) and in particular the StyleGAN2.

The proposed work is inspired by the website <https://www.thispersondoesnotexist.com/>, where a new face generated at runtime appears at each refresh of the page, but the GAN can be applied to different researches and study domains.

The thesis ends with the implementation of a simple application written in Python that wants to put into practice the concepts addressed during the development of the document, show the quality of the generated images, the simplicity with which it is possible to implement these types of networks, but also give ideas on possible improvements or future studies.

The proposed document consists of the following chapters:

- **Chapter 1:** This chapter is intended to be an introduction to the concept of Deep Learning, explaining the relationship between Artificial Intelligence, Machine Learning, Neural Networks and Deep Learning, why it is so important and its evolution over the years.
- **Chapter 2:** This chapter explains why Deep Learning in the form of: convolutional neural networks (CNN), recurrent neural networks (RNN) and artificial neural networks (ANN), is preferred to Machine Learning algorithms.
- **Chapter 3:** This chapter explores the Convolution Neural Networks (CNN), in particular we will analyze why they are so important in the field of Image

Processing and Computer Vision, and the added value that these types of networks bring.

- **Chapter 4:** This chapter deals with the Generative Adversarial Networks (GAN) introduced by Ian J. Goodfellow which are the basis on which this thesis is developed.
- **Chapter 5:** This chapter relates StyleGAN to StyleGAN2, starting from StyleGAN and concluding with the improvements made by StyleGAN2.
- **Chapter 6:** This chapter deals with the concept of latent space trying to explain its importance and its usefulness in the Deep Learning field.
- **Chapter 7:** This last chapter is dedicated to one of the possible practical applications of StyleGAN2.
- **Conclusions:** This chapter is dedicated to the conclusions in which we analyze what has been implemented, giving hints on possible future developments.

# 1 Panoramica sul Deep Learning

## 1.1 Relazione tra Intelligenza Artificiale, Machine Learning, Neural Networks e Deep Learning

Nel campo dell'Informatica, questi tre concetti sono spesso impiegati erroneamente in maniera pressoché identica, a volte addirittura scambiati. La verità è che hanno un significato molto diverso. Per capire la differenza tra Intelligenza Artificiale, Machine Learning e Deep Learning, può essere utile vederli come una matrioska, l'uno contenuto nell'altro.

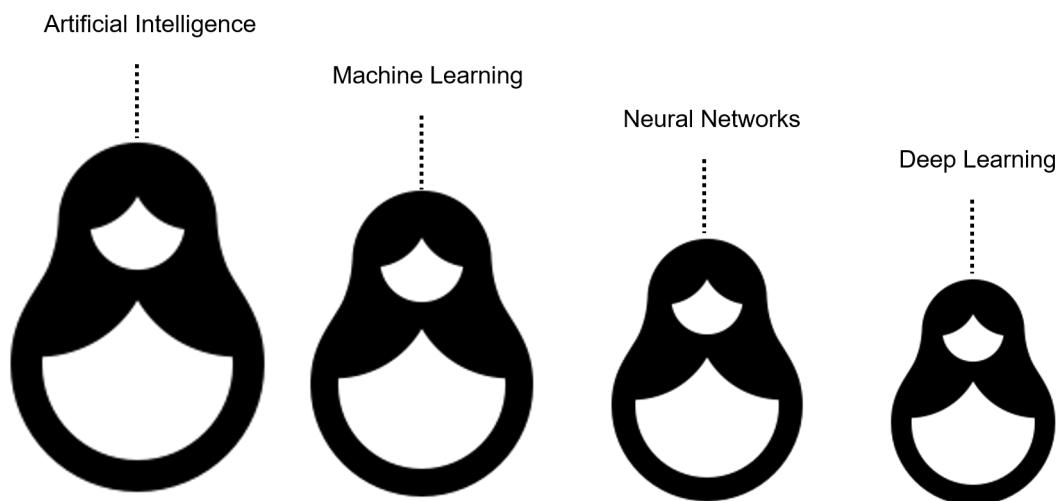


Figura 1.1: Gerarchia dell'IA

La bambola esterna rappresenta l'**Intelligenza Artificiale**. L'intelligenza artificiale è l'obiettivo che vorremmo raggiungere. Nello specifico, vogliamo creare software o hardware in grado di pensare e risolvere problemi proprio come un essere umano. I problemi di cui stiamo parlando sono quelli che influiscono maggiormente sulla nostra vita quotidiana e gli esseri umani sono abituati a capirli abbastanza facilmente. Sfortunatamente, le macchine non sono dotate di una capacità nativa per gestire questo tipo di sfide, con alcune che includono ad esempio l'interpretazione del testo o il riconoscimento delle immagini.

L'obiettivo generale è un'IA generale, un sistema altamente intelligente come quelli che possiamo vedere nella fantascienza, come C-3PO di Star Wars, che è in grado di emulare il comportamento umano fino in fondo. Al momento, non siamo lontanamente vicini a questo tipo di intelligenza artificiale, quindi in pratica dobbiamo riferirci ad essa come a un obiettivo (molto) a lungo termine.

L'esempio che abbiamo appena discusso viene anche chiamato **IA forte**. Quello che abbiamo al giorno d'oggi, invece, può essere identificato solo come **IA debole**, che è l'uso di tecniche e algoritmi con l'obiettivo di risolvere solo alcuni compiti individuali.

All'interno della bambola più grande che rappresenta l'Intelligenza Artificiale possiamo trovare una bambola più piccola che rappresenta il **Machine Learning**. Il Machine Learning riguarda l'uso di big data e una serie di algoritmi di classificazione che ribaltano gli approcci comuni alla programmazione dei computer. Un programmatore tradizionale, infatti, è abituato a scrivere algoritmi sempre più complessi ma in un certo senso sa esattamente come funziona ogni singola affermazione.

L'idea di base quando si vuole creare un classificatore è molto diversa. Il primo passo consiste nel recuperare una quantità significativa di dati, e quindi creare un insieme di funzioni che possono aiutarci a capire a quali di questi dati siamo veramente interessati. Questo approccio dovrebbe portare a un graduale miglioramento dei risultati che otteniamo e, inoltre, la possibilità di ottenere un sistema in grado di prendere decisioni in base ai dati disponibili, senza scrivere tutto l'algoritmo specifico.

La maggior parte di questi classificatori si basa su funzioni matematiche ben note, come funzioni lineari, funzioni polinomiali, raggruppamenti di vario tipo, funzioni

statistiche e così via. Molti classificatori possono anche prevedere qualche tipo di andamento futuro sfruttando serie storiche, come il prezzo di un prodotto, le stime di vendita media prevista per i mesi successivi o anche alcune indicazioni sugli investimenti finanziari.

All'interno del Machine learning troviamo un'altra bambola che rappresenta le **Neural Networks**. Le Neural Networks o reti neurali, o più nello specifico, le reti neurali artificiali (ANN – Artificial Neural Network) mimano la mente umana attraverso un set di algoritmi. Ad alto livello possiamo considerare le reti neurali come composte da quattro maggiori componenti: input, weights, bias o threshold, output. Annidato all'interno del Machine Learning esiste un altro ramo chiamato **Deep Learning**. Tutte le tecniche di Deep Learning si basano sulle reti neurali artificiali, parte di un software che tenta di replicare, per alcuni aspetti, il funzionamento delle cellule cerebrali.

Dal 2012 in poi, gli sforzi dei ricercatori di Google hanno completamente rivoluzionato lo stato dell'arte del Deep Learning. Il termine "deep" si riferisce al numero di livelli, o strati, che caratterizza la profondità dell'architettura della rete neurale. Il modo in cui funziona una rete neurale è ancora più interessante. È infatti l'algoritmo stesso che determina i classificatori sfruttando un insieme di dati solitamente molto più grande di quelli utilizzati nel normale Machine Learning. In altre parole, la macchina sceglie e definisce i classificatori specifici da utilizzare. Questi classificatori, in breve, non sono preselezionati dai ricercatori, ma sembrano avere un impatto significativamente maggiore dal punto di vista dei risultati desiderabili che vorremmo aspettarci.

Quando si parla delle tre terminologie: Intelligenza Artificiale, Machine Learning e Deep Learning; dobbiamo essere consapevoli dei loro significati molto diversi. L'Intelligenza Artificiale è l'obiettivo che vogliamo raggiungere, il Machine Learning è uno dei possibili approcci per affrontarlo, e il Deep Learning è una particolare tecnica avanzata di Machine Learning che potremmo sfruttare, forse ora la più promettente.

## 1.2 Perché il Machine Learning

Il Deep Learning è una branca dell’Intelligenza Artificiale (in particolare del Machine Learning) che si occupa di risolvere alcuni tipi di problemi attraverso l’uso di **reti neurali multistrato** (multilayer neural networks).

Rappresenta un approccio innovativo rispetto alle prime reti poco profonde che erano composte da un basso numero di strati (tipicamente tre: input, hidden e output layer) dove la maggior parte dei neuroni era situata nell’intermedio ampio.

Sebbene ci siano studi che dimostrano come le reti poco profonde siano in grado di risolvere gli stessi problemi delle reti profonde[44], l’approccio a strati è più desiderabile per una moltitudine di ragioni. Il più rilevante riguarda la riduzione dei parametri. È stato dimostrato che il potere di rappresentazione di una rete neurale con  $k$  strati con molti neuroni polinomiali deve essere espresso con molti neuroni esponenzialmente se viene utilizzato uno strato ( $k-1$ ) strutturato. Ulteriori supporti vengono dal confronto con la corteccia visiva del cervello umano sia in termini di struttura, la rete neurale umana è effettivamente un’architettura profonda, sia in termini di comportamento, gli esseri umani tendono a rappresentare concetti a un livello di astrazione come la composizione di concetti ai livelli inferiori.

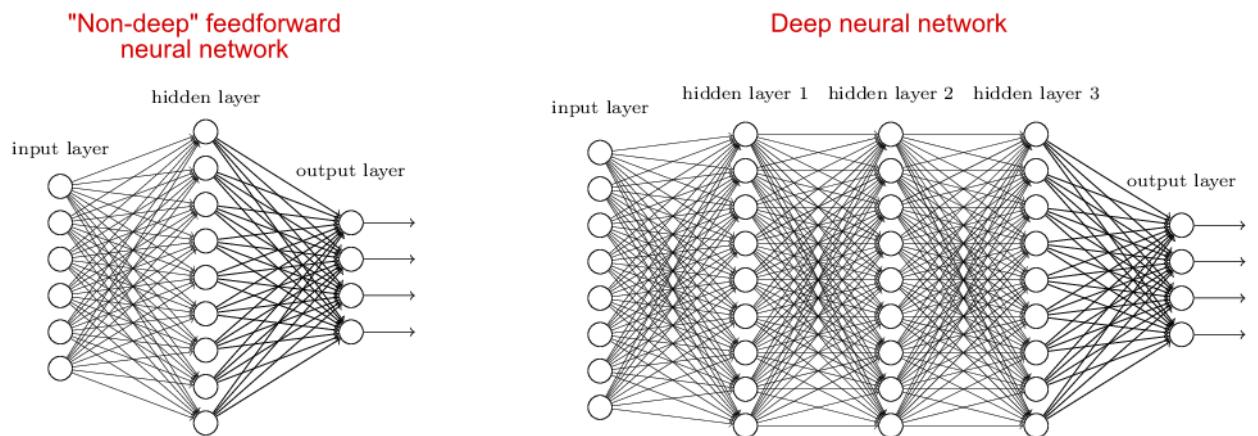


Figura 1.2: Reti non profonde vs nurali profonde

Basandosi sulle precedenti affermazioni, alcuni ricercatori[43], fanno riferimento al Deep Learning anche con il termine “Hierarchical Feature Learning” perché la principale abilità di queste reti è modellare astrazioni di alto livello partendo dalle caratteristiche di livello più basso, producendo in pratica una struttura gerarchica di caratteristiche. Gli algoritmi di deep learning, infatti, cercano di sfruttare la struttura sconosciuta nella distribuzione degli input per scoprire buone rappresentazioni, spesso a più livelli, con caratteristiche apprese al livello superiore definite in termini di caratteristiche al livello inferiore. L’obiettivo è rendere queste rappresentazioni di livello superiore ancora più astratte, con le loro caratteristiche individuali più invarianti rispetto alla maggior parte delle variazioni che sono tipicamente presenti durante il training, preservando collettivamente il più possibile le informazioni dell’input.

I campi di ricerca coinvolti spaziano dal riconoscimento delle immagini e della visione artificiale, al riconoscimento vocale, alla scoperta di nuovi farmaci, all’analisi del testo e molto altro.

Oggigiorno tutti i principali colossi tecnologici investono in Deep Learning a fini di ricerca e innovazione. In prima linea troviamo ad esempio Google, Facebook, Microsoft, Amazon, ecc.

### 1.3 Il sistema cognitivo

I progressi moderni sulle reti neurali artificiali sono stati resi possibili grazie all’ambizione umana di comprendere la funzionalità del nostro sistema cognitivo[25]. Il primo tentativo è attribuibile ad Aristotele (300 a.C.) che ispirò la Teoria dell’Associazionismo attraverso il seguente pensiero:

*Quando, quindi, compiamo un atto di reminiscenza, attraversiamo una certa serie di movimenti precursori, fino ad arrivare a un movimento al quale quello che cerchiamo è abitualmente conseguente. Quindi, è anche che cacciamo attraverso il treno mentale, escogitando dal presente o da qualche altro, e dal simile o contrario o coadiuvante. Attraverso questo processo avviene la reminiscenza. Giacché i movimenti sono in questi casi talora nello stesso tempo, talora parti del medesimo tutto, sicchè il*

*movimento successivo è già compiuto più della metà.*

Ispirato da Platone, Aristotele ha esplorato i processi di ricordo e richiamo e ha creato quattro leggi di associazione:

- **Contiguità:** Le cose o gli eventi con la vicinanza spaziale o temporale tendono ad essere associati nella mente.
- **Frequenza:** Il numero di occorrenze di due eventi è proporzionale alla forza dell'associazione tra questi due eventi.
- **Somiglianza:** Il pensiero di un evento tende a innescare il pensiero di un evento simile.
- **Contrasto:** Il pensiero di un evento tende a innescare il pensiero di un evento opposto.

Queste idee sono state rivisitate e riformulate da molti filosofi o psicologi, specialmente dal XVI secolo in poi.

Oltre alle leggi esistenti, David Hartley (1705-1757), come medico, ha proposto la sua discussione che la memoria potrebbe essere concepita come vibrazioni in scala ridotta nelle stesse regioni del cervello come esperienza sensoriale originale. Queste vibrazioni possono collegarsi per rappresentare idee complesse e quindi fungere da base materiale per il flusso di coscienza. Questa idea ha potenzialmente ispirato *Hebbian Learning Rule*, introdotta da Donald O. Hebb nel suo lavoro *The Organization of Behavior*.

Nel 1949, Hebb ha dichiarato la famosa Regola: "Cells that fire together, wire together", che enfatizza sul comportamento di attivazione delle cellule co-fired. Più specificamente, nel suo libro, ha scritto che:

*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that As efficiency, as one of the cells firing B, is increased.*

Quest paragrafo puó essere riscritto usando il linguaggio del machine learning come:

$$\Delta w_i = nx_iy$$

Dove  $\Delta w_i$  indica per il cambio di pesi sinaptici ( $w_i$ ) del neurone  $i$ , di cui il segnale di ingresso è  $x_i$ .  $y$  denota la risposta postsinaptica e  $n$  rappresenta la velocitá di apprendimento. In altre parole, la regola di apprendimento di Hebb afferma che la connessione tra due unità dovrebbe essere rafforzata in quanto la frequenza delle co-occorrenze di queste due unità aumenta.

Anche se Hebb è considerato il pioniere della rete neurale, va notato che questa regola solleva alcuni problemi che possono portare a una forma di instabilitá: poiché al crescere delle co-occorrenze, i pesi delle connessioni continueranno a crescere e i pesi di un segnale dominante aumenteranno esponenzialmente.

Questo problema é stato risolto da Erkki Oja (1982), che ha aggiornato le HLR aggiungendo un fattore di normalizzazione. La sua regola di apprendimento aggiornata avrebbe successivamente dimostrato che un il comportamento di un neurone non fa altro che approssimarsi al comportamento di un principio analizzatore (PCA - Principal Component Analyzer).

## 1.4 Reti neurali artificiali nel corso degli anni

La prima importante ricerca nel campo ANN risale al 1943, quando un neurofisiologo, Warren McCulloch, e un matematico, Walter Pitts, hanno ipotizzato i lavori interni dei neuroni e li hanno modellati una rete neurale primitiva [41]. Il loro modello neurale è stato ispirato dal comportamento dei circuiti elettronici, dove le uscite binarie sono determinate da una combinazione di ingressi ponderati lineari. A differenza delle ANNs moderne, questo modello è stato caratterizzato da pesi fissi, quindi non regolabile nel tempo. Questo apprendimento è stato introdotto solo dal 1949, con la regola di apprendimento di Hebb (Hebbian Learning Rule).

Nel 1958, Rosenblatt stava indagando sull'area dei sistemi di visione e in questo contesto ha implementato il primo dispositivo elettronico chiamato Perceptron [35]. Perceptron ha rappresentato la transizione cruciale da sistemi biologici ai sistemi

artificiali.

Fondamentalmente, il potere di rappresentazione di Rosenblatt Perceptron riguarda la definizione dei confini decisionali lineari per gestire le operazioni logiche di base, come AND, OR e NOT. Qui, è emerso anche una carenza critica evidenziata da Minski e Papert (1969) [31]. Hanno mostrato come un singolo perceptron non può risolvere XOR e NXOR poiché è richiesto un limite decisionale non lineare per separare correttamente le classi di uscita binaria.

Dal 1974 in poi, la svolta proveniva dall'introduzione di backpropagation e Multi-Layer Perceptron (MLP). I pesi del neurone non sono più percepiti come parametro fisso e backpropagation, ma mette in pratica il concetto di apprendimento fornendo alla rete un approccio di auto-miglioramento.

È stato anche dimostrato come una semplice architettura MLP può facilmente capire il problema XOR. Inoltre, MLP consente di modellare processi cognitivi come classificazione, riconoscimento, comportamenti sensomotorici, associazione e atteggiamenti di memorizzazione.

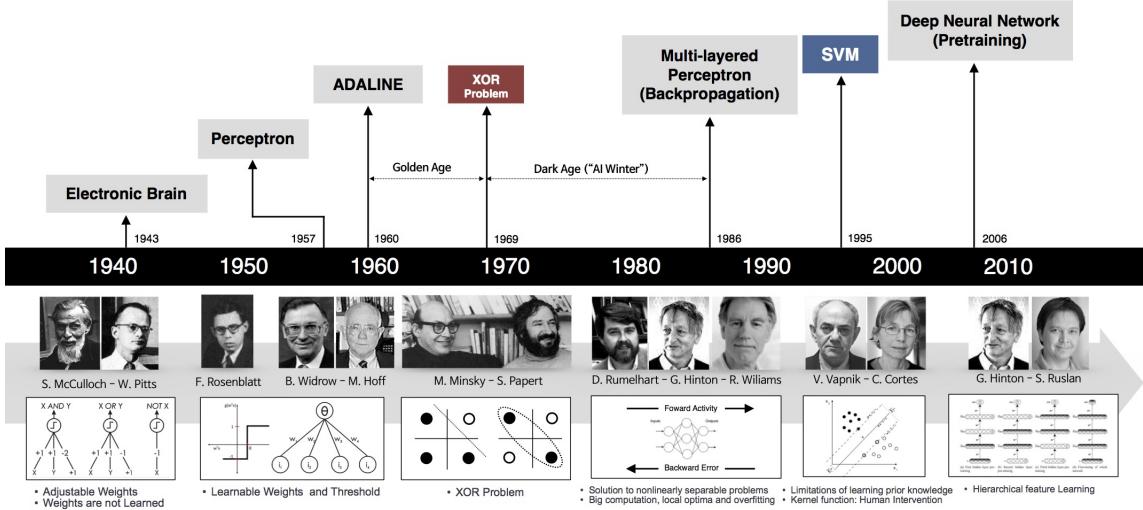


Figura 1.3: Deep Learning Time Line

L'aspetto dell'apprendimento della MLP è implementato sulla base della "Regola Delta" (Delta Rule) dando origine alla prima forma di apprendimento supervisionato, in cui la rete può migliorarsi cambiando il suo peso in base agli errori commessi

(rappresentati come la differenza tra l'uscita predetta e l'uscita prevista).

Altri lavori notevoli sono stati effettuati anche nel contesto dell'apprendimento non supervisionato. Kohonen (1981) ha introdotto le Self-Organizing Maps, un nuovo tipo di rete che non aveva bisogno di un supervisore esterno ma sfruttando le proprietà statistiche dei dati possono identificare una serie di modelli categoriali.

Ricerche sull'apprendimento non superviso procedono con le reti di Hopfield e le macchine di Boltzmann, culminando con Deep Boltzmann Machine e Deep Belief Networks. In particolare l'approccio di auto-organizzazione consente la modellazione della struttura gerarchica delle caratteristiche in cui il livello di astrazione cresce su strati nascosti. Quindi, investigando la rappresentazione dei dati latenti in questi strati nascosti, è stata proposta un'altra idea interessante: la rete neurale può essere utilizzata anche come autoencoder? La risposta è sì e in particolare questi autoencoder possono essere impiegati con successo per compiti come la riduzione delle dimensionalità e la riduzione del rumore.

Intorno al 2000, l'entusiasmo riguardo le ANN e le loro applicazioni pratiche è gradualmente sbiadita a causa dell'indisponibilità della grossa mole di dati richiesta e di adeguate risorse computazionali.

Solo nel 2009, quando i primi risultati sul riconoscimento vocale sono cresciuti, le reti neurali sono state riconsiderate e le aspettative tornarono a crescere. Nel 2013, le Convolutional Neural Networks (CNN) sono state impiegate con successo nelle attività di Computer Vision. Nel 2014 sono state utilizzate reti neurali ricorrenti per i primi modelli di traduzione automatica.

Un'ulteriore spinta è arrivata dalla diffusione di grandi moli di dati e dalle GPU sempre più veloci e a basso costo. Questi due contributi rappresentano in realtà le soluzioni cruciali a tutti i problemi che hanno causato l'ultima crisi, anni prima.

Attualmente siamo in grado non solo di analizzare grandi quantità di dati, ma siamo anche in grado di usarli per fare previsioni ed estrarre ulteriori informazioni. Con una vista al futuro, ora siamo costruendo le basi per l'intelligenza artificiale forte.

## 1.5 Approcci di apprendimento

Nel Machine Learning, tutti gli approcci di apprendimento conosciuti condividono fondamentalmente lo stesso principio: cercare di imparare qual è il migliore output **O** da fornire per ogni possibile ingresso **I** [30].

La precedente affermazione può essere espressa più accuratamente in termini matematici: tentando di trovare una funzione  $\mathbf{F}: \mathbf{I} \rightarrow \mathbf{O}$  il più efficiente possibile, dove **I** è il set dei possibili ingressi e **O** il set delle possibili uscite.

Al di là di questo principio, alcuni approcci di apprendimento della macchina migliorarono prendendo ispirazione da approcci di apprendimento della vita reale, come addestrare un animale domestico o insegnare a un bambino. Questi esempi di solito comportano un'alternanza di sessioni di formazione e test, in cui quest'ultimo ha lo scopo di controllare i progressi raggiunti dopo la formazione.

Sfortunatamente, non tutti i problemi possono essere risolti adottando un approccio standard unico e questo dipende spesso dai dati che abbiamo a nostra disposizione. Possiamo avere esempi di dati in cui abbiamo sia gli inserimenti che le uscite: (**I**, **O**). In alcuni casi, possiamo avere solo gli input **I**. A volte, invece, non abbiamo accesso diretto all'output corretto, ma possiamo ottenere una misura della qualità di un'output **O** dopo l'input **I**.

L'approccio utilizzato per affrontare il primo tipo di problemi è anche chiamato apprendimento sorvegliato (**supervised learning**). L'idea consiste nello sfruttare un ampio set di dati di allenamento codificati come coppie (**I**, **O**). La corrispondenza tra input e output è rappresentata da una funzione che spesso dipende da un gran numero di parametri. L'obiettivo della macchina è trovare un'assegnazione ottimale per tali parametri e, in caso di reti neurali, questo può essere raggiunto regolando i pesi del neurone attraverso la backpropagation. Gli esempi tipici sono i rilevatori di spam addestrati sui su e-mail esplicitamente etichettate con etichette spam e non spam.

Il risultato atteso è quello di ottenere un modello in grado di generalizzare su dati freschi, o in altre parole, il processo di apprendimento dovrebbe evitare di sovraccaricare il set di addestramento.

La parte cruciale dell'apprendimento sorvegliato è principalmente correlato alla costituzione del set di dati. Poiché i dati sono tenuti a essere correttamente etichettati, di solito ha bisogno di un enorme sforzo manuale. In generale, la maggior parte dei problemi ha i dati in input senza una conoscenza preliminare sull'output. Inoltre, molti scenari di data mining iniziano senza avere una idea del tipo di output cercato. Spesso, l'output più desiderato è ciò che non ti aspetti e questo può essere scoperto solo investigando profondamente sui dati di input, come nella ricerca di correlazione tra funzioni o rilevare valori anomali con tecniche di clustering. Normalmente questi approcci sono conosciuti come apprendimento non supervisionato (**unsupervised learning**).

Nel campo della classificazione del linguaggio naturale e della classificazione dei testi, in particolare, la ricerca di parole contenute rappresenta uno dei modelli non supervisionati più rilevanti.

Per motivi di completezza, discutiamo brevemente anche di un terzo tipo di apprendimento che negli ultimi anni sta conducendo risultati impressionanti. Copre la categoria dei problemi in cui è possibile ottenere una stima della qualità dell'output. L'approccio di cui stiamo parlando è noto come apprendimento di rinforzo (**reinforcement learning**).

Csaba Szepesvari [8] ha spiegato l'apprendimento di rinforzo (**reinforcement learning**) come paradigma di apprendimento per controllare un sistema in modo da massimizzare una misura numerica che esprime un obiettivo a lungo termine. Ciò che distingue il reinforcement learning dall'apprendimento supervisionato è che viene dato solo un feedback parziale al modello riguardo le previsioni predette. Inoltre, le previsioni possono avere effetti a lungo termine attraverso l'influenza dello stato futuro del sistema controllato. Quindi, il tempo gioca un ruolo importante. L'obiettivo nel reinforcement learning è quello di sviluppare algoritmi di apprendimento efficienti, nonché di comprendere i meriti e le limitazioni dell'algoritmo. Il reinforcement learning è di grande interesse a causa del gran numero di applicazioni pratiche che possono essere utilizzate, che vanno dai problemi di intelligenza artificiale alle operazioni di ricerca e controllo anche in ambienti ingegneristici.

## 2 Perché il Deep Learning

Questo capitolo spiega perché il Deep Learning nella forma di: convolutional neural networks (CNN), recurrent neural networks (RNN) e artificial neural networks (ANN), viene preferito agli algoritmi di Machine Learning.

Alcuni dei motivi di base per cui viene scelto il Deep Learning rispetto al Machine Learning sono:

- Il limite decisionale
- Feature Engineering

### 2.1 Machine Learning vs. Deep Learning: Limite Decisionale

Ogni algoritmo di Machine Learning apprende la mappatura dall'input all'output. Nel caso di modelli parametrici, l'algoritmo apprende una funzione con alcuni insiemi di pesi:

$$\text{input} -> f(w_1, w_2, \dots, w_n) -> \text{output} \quad (2.1)$$

Nel caso di problemi di classificazione, l'algoritmo apprende la funzione che separa 2 classi, nota come **Limite Decisionale**. Il limite decisionale ci aiuta a determinare se un dato punto appartiene a una classe positiva o negativa. Ad esempio, nel caso della *logistic regression*, la funzione di apprendimento è una funzione Sigmoid che tenta di separare le 2 classi.

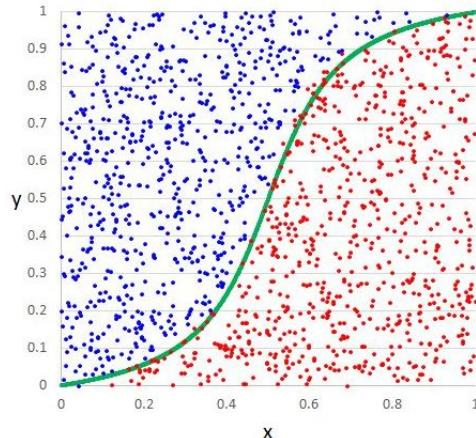


Figura 2.1: Limite Decisionale

Come si può notare nell’immagine precedente, l’algoritmo di logistic regression impara il limite decisionale lineare. Non può imparare il limite decisionale da dati non lineari come nella figura successiva.

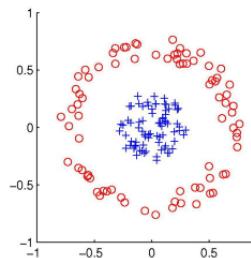


Figura 2.2: Dati non lineari

Allo stesso modo, ogni algoritmo di Machine Learning non è in grado di apprendere tutte le funzioni. Ciò limita i problemi che questi algoritmi possono risolvere.

## 2.2 Machine Learning vs. Deep Learning: Feature Engineering

Il Feature Engineering è un passaggio chiave nel processo di costruzione del modello. È un processo in due fasi:

1. Estrazione delle caratteristiche
2. Selezione delle caratteristiche

Nell'estrazione delle caratteristiche, estraiamo tutte le funzionalità richieste per la nostra dichiarazione del problema e nella selezione delle caratteristiche, selezioniamo le funzionalità importanti che migliorano le prestazioni del nostro modello di machine learning o deep learning.

Si può considerare un problema di classificazione delle immagini. L'estrazione manuale di caratteristiche da un'immagine richiede una profonda conoscenza dell'argomento e del dominio. È un processo estremamente dispendioso in termini di tempo. Grazie al Deep Learning, possiamo automatizzare il processo di Feature Engineering!

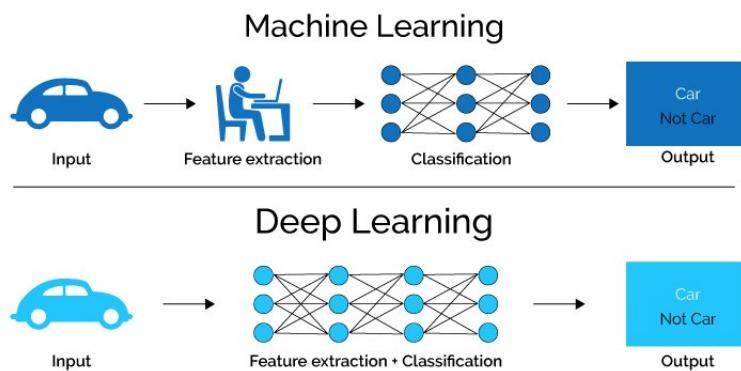


Figura 2.3: Machine Learning vs Deep Learning

## 2.3 Diversi tipi di reti neurali nel Deep Learning

Da questo punto in poi ci concentreremo su tre importanti tipi di reti neurali che costituiscono la base per la maggior parte dei modelli pre-addestrati nel Deep Learning:

- Artificial Neural Networks (ANN)
- Convolution Neural Networks (CNN)

- Recurrent Neural Networks (RNN)

## 2.4 Cosa é l'Artificial Neural Networks (ANN) e le sue applicazioni

Un singolo perceptron (o neurone) può essere immaginato come una regressione logistica. La rete neurale artificiale, o ANN, è un gruppo di più percettron/neuroni a ogni livello. ANN è anche noto come **Feed-Forward Neural Network** perché gli input vengono elaborati solo in una direzione, la direzione di avanzamento.

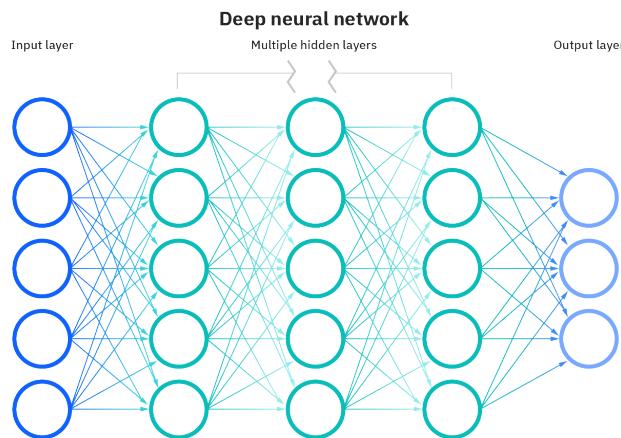


Figura 2.4: ANN

Come si può vedere nella figura 2.4, l'ANN è composta da 3 livelli: Input, Hidden e Output. Il livello di input accetta gli input, il livello hidden elabora gli input e il livello di output produce il risultato. In sostanza, ogni livello cerca di apprendere determinati pesi.

L'ANN può essere utilizzata per risolvere problemi relativi a:

- Dati tabulari
- Immagini
- Dati di testo

### 2.4.1 Artificial Neural Networks (ANN)

La rete neurale artificiale è in grado di apprendere qualsiasi funzione non lineare. Quindi, queste reti sono popolarmente conosciute come **Approssimatori di Funzioni Universali**. Le ANN hanno la capacità di apprendere i pesi che mappano qualsiasi input sull'output.

Uno dei motivi principali alla base dell'approssimazione universale è la **funzione di attivazione**. Le funzioni di attivazione introducono proprietà non lineari nella rete. Questo aiuta la rete ad apprendere qualsiasi relazione complessa tra input e output.

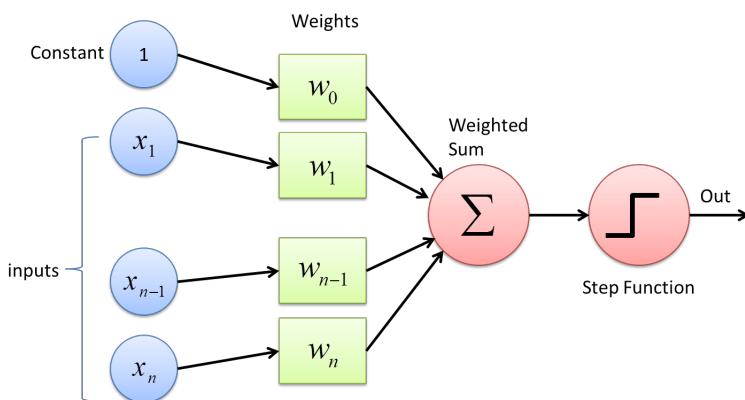


Figura 2.5: Perceptron

L'output su ciascun neurone è l'attivazione di una somma ponderata di input. Ma, cosa succede se non è presente alcuna funzione di attivazione? La rete apprende solo la funzione lineare e non può mai apprendere relazioni complesse.

*An activation function is a powerhouse of ANN!*

### 2.4.2 Sfide con Artificial Neural Networks (ANN)

Durante la risoluzione di un problema di classificazione dell'immagine utilizzando l'ANN, il primo passaggio consiste nel convertire un'immagine bidimensionale in un vettore monodimensionale prima di addestrare il modello. Questo ha due inconvenienti:

1. Il numero di parametri addestrabili aumenta drasticamente con l'aumento della dimensione dell'immagine. Se ad esempio la dimensione dell'immagine è 224\*224, il numero di parametri addestrabili al primo livello nascosto con solo 4 neuroni è 602.112. È enorme!
  
2. L'ANN perde le caratteristiche spaziali di un'immagine. Le caratteristiche spaziali si riferiscono alla disposizione dei pixel in un'immagine.

Un problema comune in tutte queste reti neurali è il **Vanishing and Exploding Gradient**. Questo problema è associato all'algoritmo di **backpropagation**. I pesi di una rete neurale vengono aggiornati tramite questo algoritmo di backpropagation trovando i gradienti.

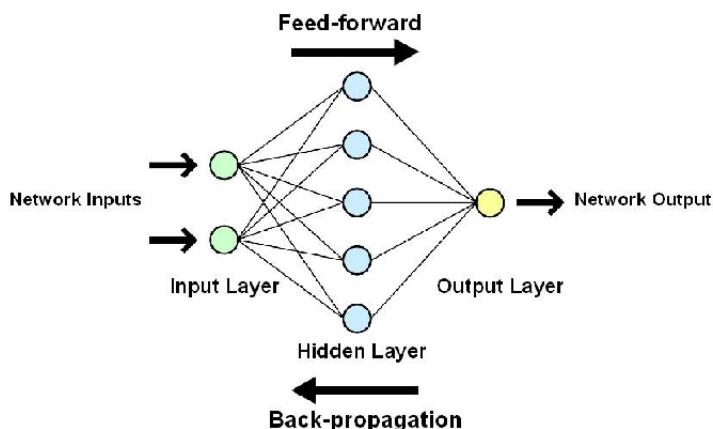


Figura 2.6: Backpropagation

Quindi, nel caso di una rete neurale molto profonda (rete con un gran numero di strati nascosti), il gradiente svanisce o esplode mentre si propaga all'indietro, il che porta a un gradiente che scompare ed esplode.

L'ANN non è in grado di acquisire informazioni sequenziali nei dati di input necessari per gestire dati sequenziali.

## 2.5 Cosa é Recurrent Neural Network (RNN) e le sue applicazioni

Prima di tutto cerchiamo di capire le differenze tra RNN e ANN sotto il punto di vista architettonale.

Un loop su uno dei livelli nascosti di una ANN trasforma la rete in una RNN.

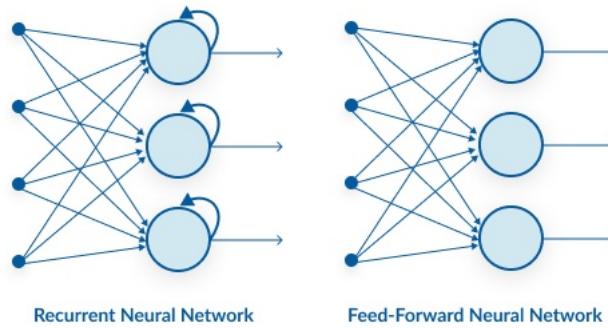


Figura 2.7: ANN vs CNN

Come si può vedere nell’immagine precedente, l’RNN ha una connessione ricorrente nello stato nascosto. Questo vincolo di loop assicura che le informazioni sequenziali vengano acquisite nei dati di input.

Possiamo usare le RNN per risolvere i seguenti tipi di problemi:

- Dati di serie temporali
- Dati di testo
- Dati audio

### 2.5.1 Vantaggi delle Recurrent Neural Network (RNN)

L’RNN acquisisce le informazioni sequenziali presenti nei dati di input, ovvero la dipendenza tra le parole nel testo mentre si effettuano previsioni.

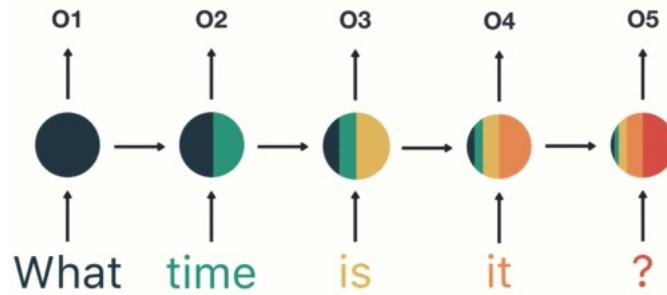


Figura 2.8: Sequence

L'immagine precedente mostra come gli output ( $o_2, o_3, o_4, o_5$ ) dipendono non solo dalla parola che stanno analizzando ma anche dallo stato precedente.

Le RNN condividono i parametri in diversi passaggi temporali. Questo è popolarmente noto come **Parameter Sharing**. Ciò si traduce in un minor numero di parametri da addestrare e diminuisce il costo computazionale.

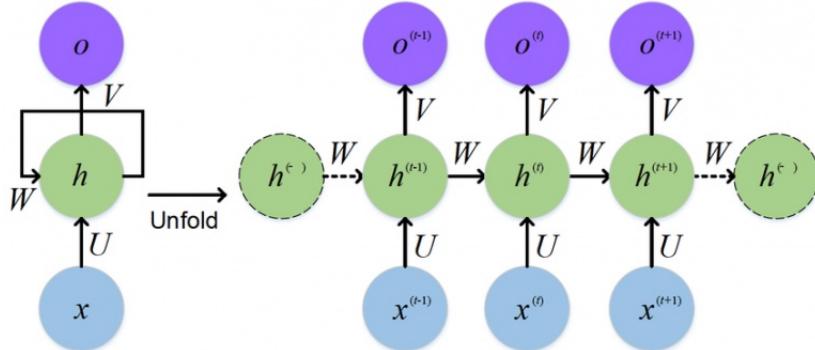


Figura 2.9: Unfolded RNN

Come mostrato nella figura precedente, le 3 matrici di peso:  $U$ ,  $W$ ,  $V$ , sono le matrici di peso condivise in tutti i passaggi temporali.

### 2.5.2 Sfide con le Recurrent Neural Network (RNN)

Anche le RNN profonde (RNN con un numero elevato di passaggi temporali) soffrono del problema del gradiente di scomparsa ed esplosione che è un problema comune in tutti i diversi tipi di reti neurali.

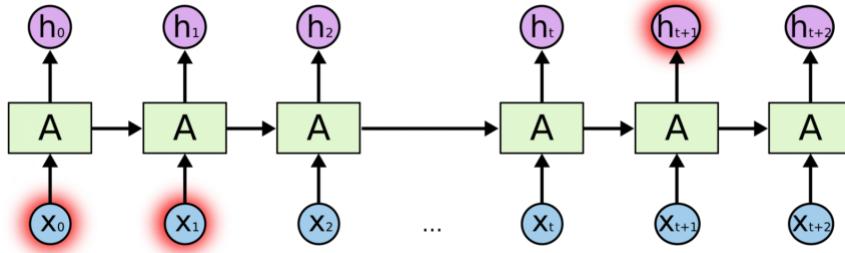


Figura 2.10: Vanish Gradient

L’immagine precedente mostra come il gradiente calcolato nell’ultimo passaggio temporale svanisce quando raggiunge il passaggio temporale iniziale.

## 2.6 Cosa sono le Convolution Neural Network (CNN) e le sue applicazioni

Le Convolution Neural Network (CNN) sono di attuale importanza nella comunità del deep learning. I modelli CNN vengono utilizzati in diverse applicazioni e domini e sono particolarmente diffusi nei progetti di elaborazione di immagini e video.

Gli elementi costitutivi delle CNN sono i filtri, noti anche come kernel. I kernel vengono utilizzati per estrarre le funzionalità rilevanti dall’input utilizzando l’operazione di convoluzione. Proviamo a cogliere l’importanza dei filtri usando le immagini come dati di input. Unendo un’immagine con i filtri si ottiene una mappa delle caratteristiche[33].

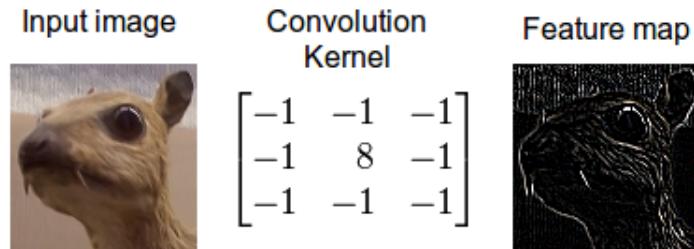


Figura 2.11: Convoluzione

Sebbene le reti neurali convoluzionali siano state introdotte per risolvere i problemi

relativi alle immagini, hanno dimostrato prestazioni importanti anche sugli input sequenziali.

### 2.6.1 Vantaggi delle Convolution Neural Network (CNN)

La CNN apprende i filtri automaticamente senza menzionarli esplicitamente. Questi filtri aiutano a estrarre le caratteristiche giuste e rilevanti dai dati di input.

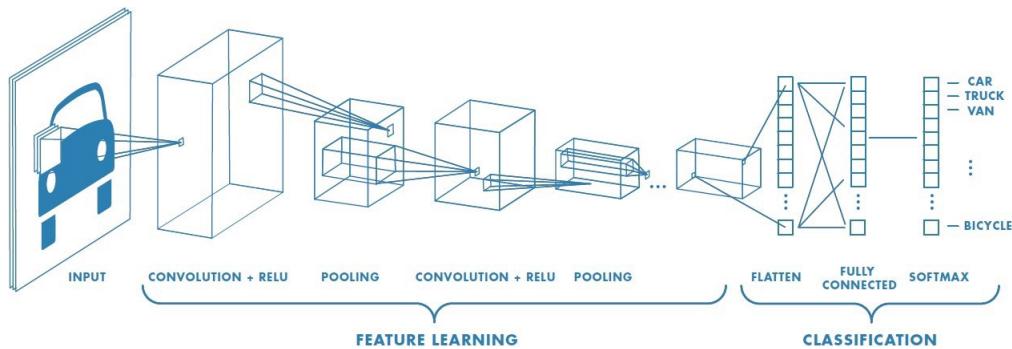


Figura 2.12: CNN

La CNN acquisisce le caratteristiche spaziali da un'immagine. Le caratteristiche spaziali si riferiscono alla disposizione dei pixel e alla relazione tra loro in un'immagine. Ci aiutano a identificare accuratamente l'oggetto, la posizione di un oggetto e la sua relazione con altri oggetti in un'immagine.

Ad esempio, guardando l'immagine di una persona, possiamo facilmente identificare il volto di un essere umano osservando caratteristiche specifiche come occhi, naso, bocca e così via. Possiamo anche vedere come queste caratteristiche specifiche sono organizzate in un'immagine. Questo è esattamente ciò che le CNN sono in grado di catturare.

La CNN segue anche il concetto di condivisione dei parametri. Un singolo filtro viene applicato a diverse parti di un input per produrre una mappa delle caratteristiche.

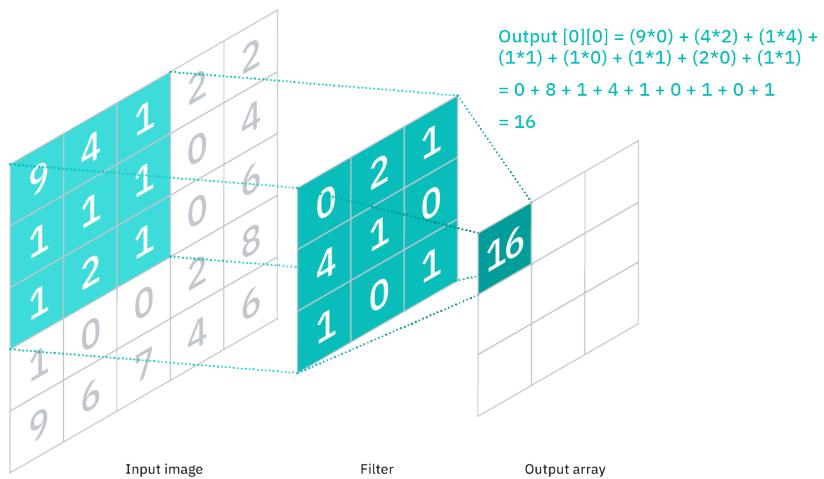


Figura 2.13: Applicazione dei filtri

## 2.7 MPL(ANN) vs RNN vs CNN

In questo capitolo riassumiamo in modo tabellare le principali differenze tra le reti viste in questo capitolo.

	MPL	RNN	CNN
Data	Tabular Data	Sequence Data	Image Data
Recurrent Connections	No	Yes	No
Parameter Sharing	No	Yes	Yes
Spatial Relationship	No	No	Yes
Vanishing and Exploding Gradient	Yes	Yes	Yes

Tabella 2.1: MPL(ANN) vs RNN vs CNN

# 3 Convolution Neural Network (CNN)

Questo capitolo approfondisce le Convolution Neural Network (CNN), in particolare analizza il perché sono così importanti nel campo del Image Processing e Computer Vision, e il valore aggiunto che questi tipi di reti portano.

La loro capacità di apprendere da grandi collezioni di dati: mappature complesse, non lineari e multi dimensionali, rende queste reti le candidate ideali per operazioni di Image Recognition, ma non senza problemi.

Innanzitutto, per alimentare un'immagine in una rete neurale, è necessario che il numero di unità dello strato di input sia uguale al numero di pixel dell'immagine. Pertanto, collegare completamente lo strato di input con lo strato nascosto adiacente richiede diverse decine di migliaia di connessioni e quindi parametri di peso. L'hardware e gli esempi di addestramento necessari all'addestramento della rete crescono proporzionalmente con la capacità della rete.

Un'altra importante limitazione delle reti completamente connesse è che non forniscono un'invarianza interna nella fase di estrazione delle caratteristiche. La sfida principale nella classificazione delle immagini, nell'OCR e nel riconoscimento facciale consiste nel rilevare oggetti che possono presentarsi con posizione, proporzioni, dimensione e qualità diversi. Inoltre, poiché un oggetto è rappresentato da un sottoinsieme di pixel adiacenti, la rete neurale dovrebbe occuparsi della topologia dell'input e della correlazione delle caratteristiche spaziali.

La soluzione fornita dalle CNN per gestire l'invarianza e le proprietà locali consiste nel condividere il peso attraverso lo spazio 2D. Ciò riduce drasticamente il numero di parametri indipendenti da regolare durante l'addestramento.

Quindi, in conclusione, l'uso di architetture completamente connesse è ragionevole per quei casi in cui l'ordine degli input è irrilevante per la stima dell'output. In effetti, le possibili correlazioni tra gli input possono essere viste sotto due aspetti: lo spazio e il tempo.

L’elaborazione video è un contesto tipico in cui entrambi dovrebbero essere considerati. In tal caso, ad esempio, una CNN può affrontare l’aspetto spaziale del contenuto dei singoli frame mentre le Recurrent Neural Networks (come LSTM[36] e GRU[23]) possono essere utilizzate per considerare la sequenza temporale dei frame.

### **3.1 Rappresentazione uniforme di dati eterogenei**

Dati tabellari, immagini, testi, video, audio e più in generale ogni tipo di dato può essere organizzato in uno specifico formato, ma questo non significa che una rete neurale li possa ingerire così come sono.

Questo ci porta al concetto di pre-processamento dei dati. Ogni linguaggio di programmazione o framework di deep learning, richiede una rappresentazione dell’input indipendentemente dal tipo di dato. In generale, ogni esempio di addestramento è sempre rappresentato come un vettore numerico o una matrice. Questi concetti sono generalizzati a matrici di ordine superiore, o più comunemente tensori.

La rappresentazione con tensori di un’immagine è abbastanza intuitiva: due dimensioni sono associate alla larghezza e all’altezza mentre la terza dipende dalla specifica notazione del colore. Generalmente è RGB o BGR, che indicano i canali rosso, verde e blu.

In sintesi, un’immagine a colori viene convertita in un tensore di ordine 3 di valori [da 0 a 255]. Nel caso dell’immagine in scala di grigi il tensore degenera in un tensore di ordine 2, o in altri termini una semplice matrice.

Meno ovvio, invece, è come rappresentare altri tipi di dati sempre in formato tensore numerico. Ad esempio dati testuali e in particolare al concetto di parola. In tal caso, infatti, le rappresentazioni dovrebbero essere in grado di esprimere correlazioni di tipo ancora più sofisticato, come semantica e sintattica, e considerare anche il contesto in cui le parole si trovano. Alcuni lavori si concentrano sulla gestione dei dati di testo come segnale grezzo unidimensionale, proprio come accade per le immagini.

## 3.2 Struttura di una CNN

Il padre delle **Convolutional Neural Network (CNN)** é Yann LeCun che nel 1998 ha introdotto il primo prototipo chiamato LeNet-5 per il riconoscimento di caratteri.

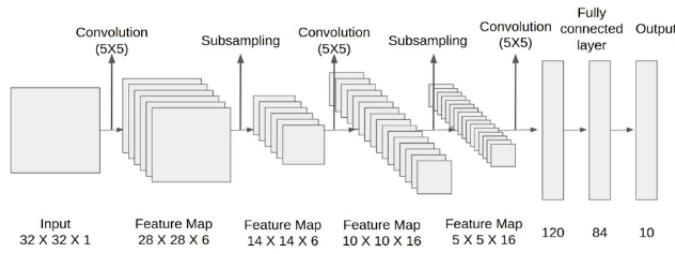


Figura 3.1: LeNet-5 Architettura

Dal 1998 sono stati proposti molti modelli come: LeNet, AlexNet, ZFNet, GoogLeNet, VGGNet. Il piú importante é sicuramente Inception-V3[42].

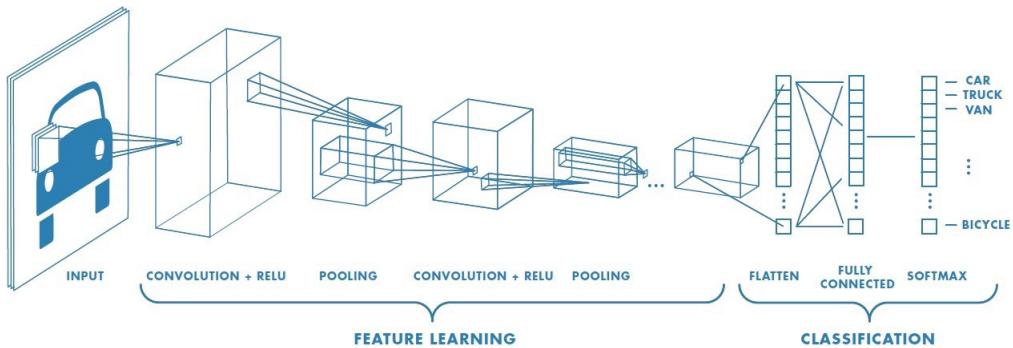


Figura 3.2: CNN

Come é possibile notare dalla Figura 3.2 é possibile identificare 4 categorie principali:

- Convolutional Layer
- Pooling Layer

- Flattering Layer
- Fully-Connected Layer

### 3.2.1 Convolutional Layer

Convolutional Layer rappresenta il cuore di questo tipo di reti. Un'operazione di convoluzione può essere espressa come:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau \quad (3.1)$$

Questa formula è molto comune nel processamento di segnali, dove l'integrale riguarda il dominio del tempo. Nel nostro caso l'immagine rappresenta un segnale discreto di pixel nello spazio 2D.

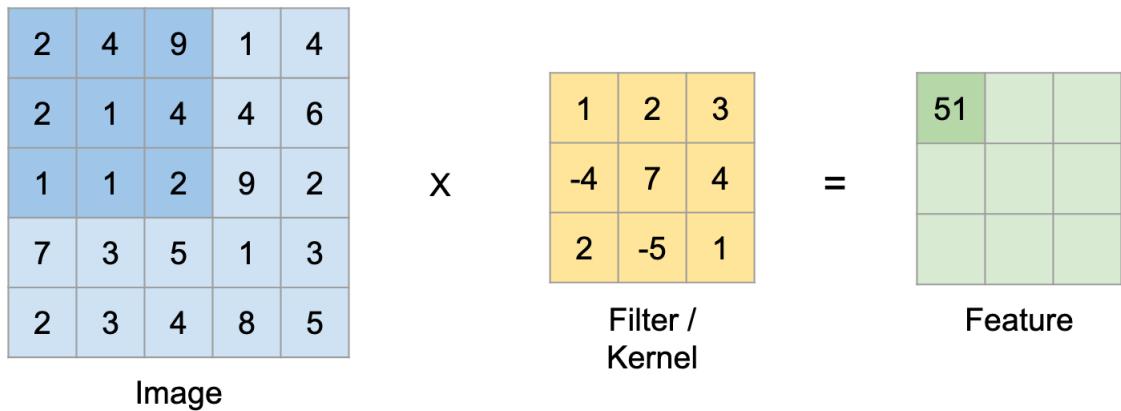


Figura 3.3: Convoluzione

La Figura 3.3 mostra un esempio di convoluzione. Durante l'esecuzione, un **feature detector** (noto anche come **kernel**) esegue una scansione dell'intera immagine di input producendo quella che viene chiamata **feature map** (nota anche come mappa convoluta o mappa di attivazione).

La Feature map è il risultato di un prodotto a matrice per elementi e si può notare nell’immagine la dimensione dell’immagine di input è stata ridotta. Il tasso di riduzione dipende da un parametro chiamato stride che definisce di quanti pixel il kernel deve essere spostato dalla posizione corrente ad ogni passaggio (nell’esempio lo stride è impostato su uno).

$$Featuresize = \frac{Imagesize - Kernelsize}{Stride} + 1 \quad (3.2)$$

I benefici della convoluzione sono enfatizzare alcune caratteristiche e allo stesso tempo scartare informazioni non necessarie, riducendo la dimensione e di conseguenza i tempi di computazione.

Questo processo assomiglia al comportamento del cervello umano, infatti quando guardiamo un’immagine non ci concentriamo sui singoli pixel ma su una porzione di pixel, un oggetto o un insieme di caratteristiche.

In pratica, il processo di convoluzione è molto più complesso di quello descritto, in quanto i rilevatori di caratteristiche tipicamente non sono uno solo, come nell’esempio, ma sono diversi e ognuno viene applicato sull’immagine di input e quindi produce una mappa di caratteristiche diversa.

I valori della matrice del kernel non sono casuali. Sono studiati e utilizzati da quasi tutte le applicazioni di elaborazione delle immagini per applicare alcune tecniche di filtraggio specifiche: sfocatura, sfocatura gaussiana, sfocatura movimento, trova bordi, nitidezza, rilievo, filtro medio e mediano e così via[38].

Infine, poiché le immagini sono spesso caratterizzate da un’elevata non linearità, soprattutto sulle linee di confine tra oggetti distinti, è pratica comune applicare la funzione di attivazione ReLU alle mappe delle caratteristiche generate in modo da rompere quelle linearità[29].

### 3.2.2 Pooling Layer

il primo step di una CNN è tipicamente caratterizzato da una serie di convoluzioni alternate e pooling layer.

L’operazione di pooling funziona allo stesso modo della convoluzione, ma con alcune

importanti differenze. Il principale punto di forza è quello di iniettare nella rete neurale la proprietà di invarianza spaziale.

L'operazione viene eseguita sulle feature maps prodotte dopo la convoluzione e, come per la convoluzione, l'input viene scansionato da in alto a sinistra fino in basso a destra da una finestra quadrata che aggrega un insieme di pixel in uno. Anche in questo caso, lo spostamento è regolato con un parametro stride e la mappa delle caratteristiche risultante viene ridotta a favore della velocità di calcolo.

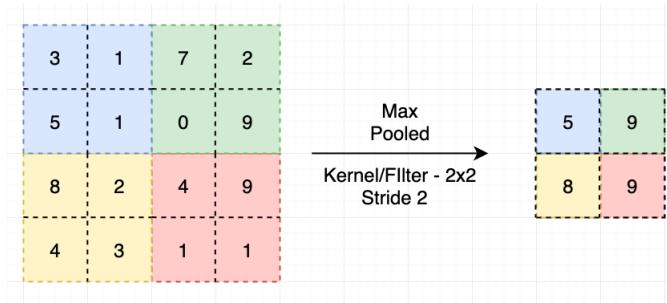


Figura 3.4: Max Pooling

Al contrario, ora non abbiamo a che fare con il rilevatore di funzionalità, la funzione di mappatura è determinata dal tipo specifico di operazione di pooling utilizzata (la più comune è Max Pooling che filtra solo il valore massimo di pixel all'interno della finestra, come illustrato nella Figura 3.4).

Intuitivamente, le informazioni perse dopo la riduzione sono proprio quelle che tengono traccia della varianza spaziale e dei potenziali rumori, consentendo di concentrarsi solo sulle caratteristiche necessarie, prevenendo così l'overfitting e portando alla generalizzazione.

Per quanto riguarda le possibili operazioni di pooling, ci sono ovviamente delle alternative proposte rispetto al Max Pooling. Il motivo per cui il Max Pooling viene enfatizzato non è casuale, infatti le ricerche dimostrano che Max Pooling è di gran lunga superiore per catturare invarianti in dati simili a immagini, rispetto ad altre operazioni come Subsampling, che invece si prende cura di tutti i pixel calcolandone la media (Average Pooling).

### 3.2.3 Flattening Layer

Dopo aver superato la sequenza delle trasformazioni di convoluzione e pooling, l'input è pronto per essere classificato.

A tal fine, una CNN classica termina sempre con un perceptron multistrato composto da uno o più strati completamente connessi.

Per creare un ponte tra i due stadi e quindi alimentare il primo livello completamente connesso, le features maps devono essere convertite in vettori unidimensionali.

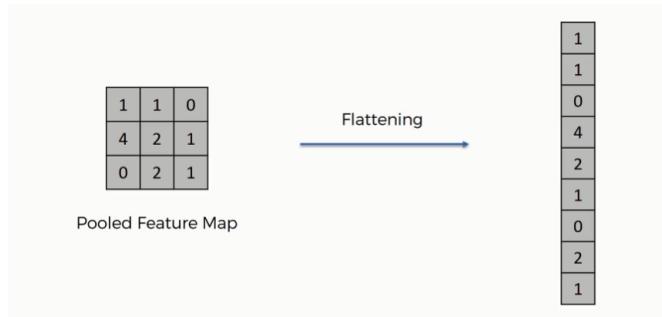


Figura 3.5: Flattering

Consiste nel concatenare le righe della matrice una dopo l'altra e infine trasporre il tutto in una colonna. Come possiamo vedere, l'operazione è piuttosto semplice e non necessita di ulteriori spiegazioni.

### 3.2.4 Fully-Connected Layers

L'obiettivo è riutilizzare tutte le mappe delle caratteristiche prodotte al termine della prima fase e tradurre il contenuto informativo in una classificazione. La classificazione finale dovrebbe attivare il neurone associato all'oggetto rilevato.

### *Convolution Neural Network (CNN)*

---

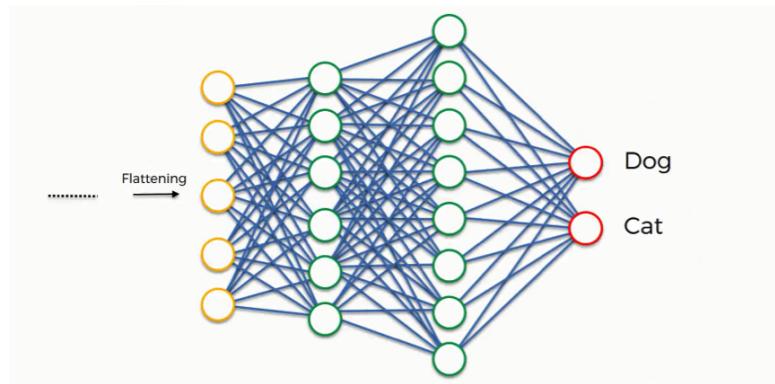


Figura 3.6: Connected Layers

Supponendo di essere in un contesto di apprendimento supervisionato, la classificazione non è immediata ma potrebbe richiedere un allenamento adeguato con diversi passaggi in avanti e backpropagation.

I passaggi visti fino ad ora affrontano l'argomento ad alto livello, sono intesi per dare solo una introduzione al tema delle reti neurali e nello specifico le CNN. I modelli reali normalmente consistono in diversi milioni di parametri e richiedono GPU ad alte prestazioni per completare l'addestramento in un tempo ragionevole.

# 4 Generative Adversarial Networks

In questo capitolo parla delle *Generative Adversarial Networks* (**GAN**) introdotte da Ian J. Goodfellow[24] che sono la base su cui questa tesi viene sviluppata.

## 4.1 Generative model

Prima di iniziare a parlare di GAN é necessario chiedersi cosa sia un modello generativo e le sue possibili applicazioni. Come dice il nome stesso un modello generativo serve per generare qualcosa. Questi tipi di reti possono essere usate per generare dati, come ad esempio immagini o testi che non sono mai stati generati prima. Questo é reso possibile partendo da un dataset (collezione di immagini o testi) usati per addestrare la rete.

Uno degli scopi di questi modelli é quello di generare contenuti che sono talmente simili agli originali che é difficile distinguere l'originale dalla quello generato. Una volta raggiunto questo tipo di accuratezza é possibile iniziare a generare contenuti completamente originali e unici.[32][40][39]



Figura 4.1: Immagine generata con StyleGan2

Nella figura 4.1 é raffigurato il volto di una donna generato usando l'algoritmo StyleGan2 argomento di questa tesi.

## 4.2 Struttura del GAN

Il modello é composto da due attori principali: **Generative Model (G)** e **Discriminative Model (D)**.

Accanto al generatore, il sistema deve essere supportato con un ampio database di immagini del mondo reale.

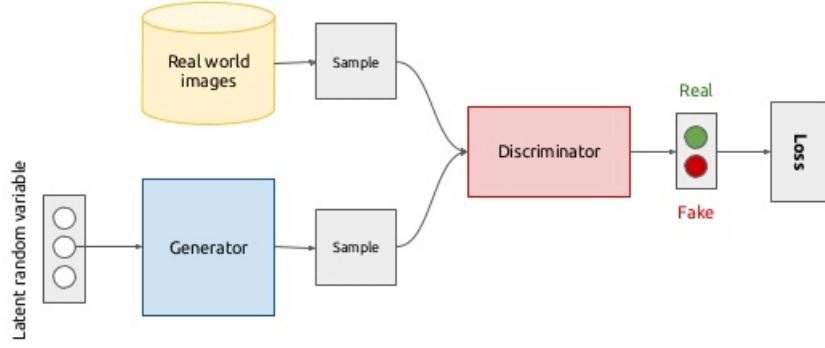


Figura 4.2: Architettura GAN

Iniziamo a visualizzare la struttura dei componenti. Sia il generatore che il discriminatore sono reti neurali profonde. La dualità G-D assicura un’importante proprietà simmetrica, in effetti, sono immagini speculari l’uno dell’altro. In particolare, il Generatore è meglio riconosciuto come Deconvolutional Neural Network. Una Deconvolutional Neural Network non è altro che una CNN.

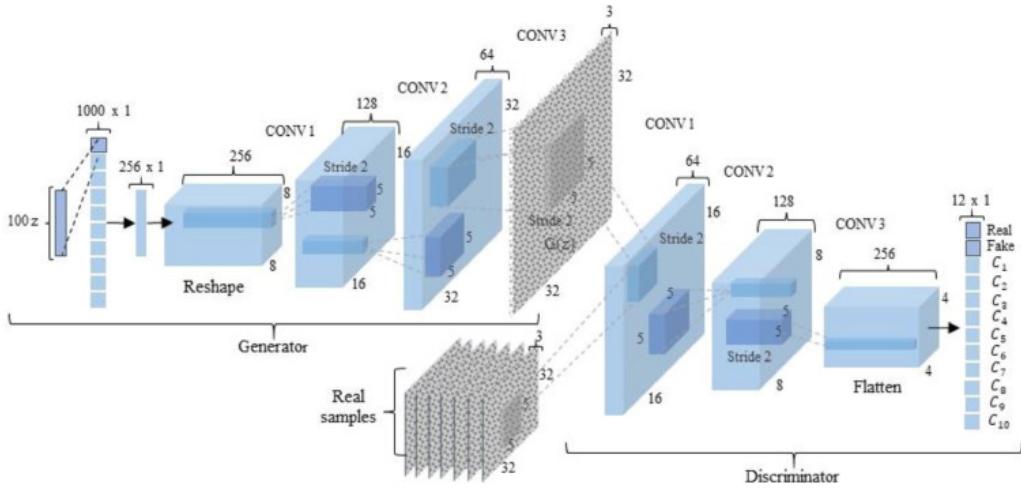


Figura 4.3: Discriminator-Generator Architettura

#### 4.2.1 Interazione

L’interazione tra G e D rappresenta il classico gioco a somma zero[1].

Nel caso specifico del GAN, il guadagno e la perdita corrispondono esattamente

alle funzioni dei costi dei due partecipanti.

Il generatore e il discriminatore sono costantemente in contatto e lavorano in Tandem. Il primo (come dovrebbe suggerire il nome) è responsabile di generare un flusso di dati, come ad esempio delle immagini.

Queste immagini vengono quindi presentate una per una al discriminatore che fondamentalmente deve rispondere alla seguente domanda: quanto è probabile che questa immagine sia simile a ciò che possiamo vedere nel mondo reale?

Ovvero: è reale o fake?

Per capire ciò che è reale, il Discriminator deve essere consapevole del concetto di realtà. Ecco perché deve essere fornito un set di dati di immagini reali accanto al generatore.

Ad esempio, supponiamo che il nostro obiettivo siano i cani. Il set di dati dovrebbe contenere una grande collezione di foto di cani di diverse razze, in diverse posizioni, diverse prospettive, colori e così via. Il generatore dovrebbe produrre nuove immagini che sembrino il più possibile dei cani. Mentre il discriminatore ottiene i dati da entrambe le fonti, deve processare ogni immagine come falsa (proveniente dal generatore) o reale (proveniente dal dataset).

Questa classificazione, favorisce sempre l'aumento dell'apprendimento. Intuitivamente, ogni volta che il discriminatore indovina la previsione il generatore sa che deve migliorare se stesso, quindi la prossima volta ha più possibilità di ingannare con successo il suo avversario. In alternativa, è il discriminatore che deve migliorare se stesso usando la backpropagation.

Istruire una rete GAN è un compito molto difficile che deve prendere in considerazione anche più aspetti rispetto alle tradizionali pratiche di apprendimento discusse nei capitoli precedenti.

La ragione è abbastanza ovvia; prima avevamo modelli singoli con la propria funzione di perdita, mentre ora ne abbiamo due che inoltre devono anche essere sincronizzate per realizzare il gioco somma zero. Una mancanza di sincronizzazione può degenerare nello scenario di collapsed scenario.

Per capire meglio l'idea alla base dello scenario è necessario fare un'analogia con un esempio reale. Citando le parole di Ian J. Goodfellow:

*The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency.*

*Il modello generativo può essere pensato come analogo a una squadra di contraffattori, cercando di produrre valuta falsa e usarla senza rilevamento, mentre il modello discriminativo è analogo alla polizia, cercando di rilevare la valuta contraffatta.*

#### 4.2.2 Comportamento

Questa sezione descrive come funziona l'algoritmo alla base del GAN. Il tutto può essere riassunto tramite la seguente funzione:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (4.1)$$

Il generatore  $G$  è associato a una distribuzione  $p_g$  sui dati  $x$  e il suo vettore di rumore di ingresso è  $p_z(Z)$ .  $D(x)$  definisce la classificazione fatta dal Discriminator, un valore compreso tra 0 e 1 che misura la probabilità che  $x$  provenga dal dataset reale piuttosto che da  $p_g$ .

Il motivo di  $\max_D$  è immediato, bisogna allenare  $D$  per massimizzare la probabilità di assegnare l'etichetta corretta sia agli esempi di formazione che ai campioni di  $G$ . Dato che stiamo giocando alla somma zero, ci aspettiamo che una controparte minimizzi, e questo è il compito di  $\min_G$ , che è espresso come  $\log(1 - D(G(z)))$ .

Nella funzione obiettivo  $\mathbb{E}$  indica la funzione di perdita di entropia binaria, in modo che l'obiettivo generale sia massimizzare l'entropia del Discriminator e ridurre al minimo l'entropia del Generator. Nel suo articolo, Goodfellow fornisce un pseudo-algoritmo per descrivere il processo di formazione.

```

for number of training iterations do
    for  $k$  steps do
        • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
        • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

    end for
    • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
    • Update the generator by descending its stochastic gradient:
        
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

end for

```

Figura 4.4: GAN Algoritmo

### 4.2.3 Il training

L'uso delle reti GAN oggi è oggetto di molti studi e ricerche. In particolare la parte più difficile è correlata a come addestrare il modello generativo rispetto al discriminativo.

*It is easier to recognize a Monet's painting than drawing one. [28]*

Al fine di ottenere un risultato ottimale, durante l'intero addestramento è fondamentale mantenere una corretta sincronizzazione tra i due attori.

L'immagine seguente illustra la procedura di backpropagation sia sul discriminatore che sul generatore.

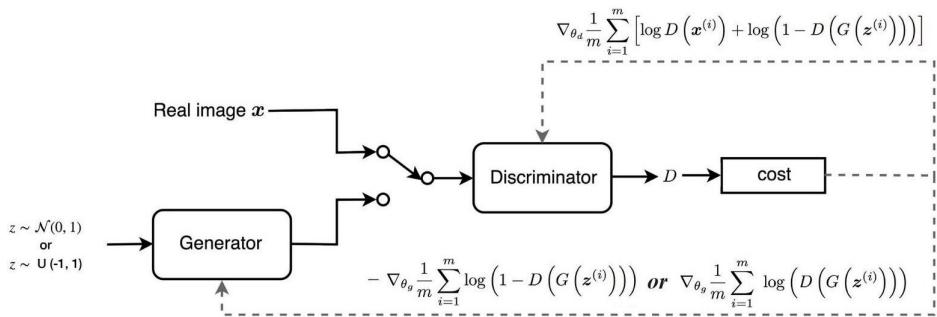


Figura 4.5: Discriminante Generatore con backpropagation

Le cause principali che possono portare a una cattiva convergenza del modello sono:

- **Vanishing gradient:** Il discriminatore impara più velocemente rispetto al generatore. Pertanto inizia a indovinare tutto il dataset in modo che il gradiente del generatore svanisce drammaticamente causando l'impossibilità dell'apprendimento.
- **Non-convergence:** Scoperto da Arjovsky[22] cerca di risolvere il problema del vanishing gradient provando a bloccare il generatore e addestrando solo il discriminatore ma ottenendo un modello instabile.
- **Mode collapse:** Uno degli eventi più critici da risolvere in GAN è noto come Mode Collapse.

Nel GAN, ci riferiamo al Mode Collapse, nella situazione in cui il generatore inizia a produrre sempre lo stesso output e quindi imbroglia sempre il discriminatore. Pertanto, l'allenamento complessivo deve essere considerato un flop, poiché il generatore non riesce ad apprendere la distribuzione dei dati nel mondo reale e rimane bloccato in uno spazio ristretto con una varietà estremamente bassa.

Un esempio pratico può essere osservato nella sezione sperimentale del paper “Unrolled GAN”[37], dove l'autore ha confrontato il risultato del loro modello, applicato sul dataset MNIST, con lo stesso risultato ottenuto utilizzando un GAN standard. Quest'ultimo caso mostra una situazione di “Helvetica Scenario” in cui il modello collassa generando continuamente la stessa cifra.

Matematicamente parlando, il Mode Collapse si verifica quando il modello raggiunge una condizione spaziale chiamata Local Nash Equilibrium (LNE) associata a una performance generativa non ottimale[26].

L'obiettivo della formazione è invece quello di convergere verso un Global Nash Equilibrium (GNE), che spesso può essere arbitrariamente lontano dalla LANE ottenuta.

In letteratura sono state proposte diverse soluzioni per prevenire il più possibile il rischio di Mode Collapse, come i GANG (Generative Adversarial Network Games)[34] e i Coulomb GAN[27].

Il primo parte formalizzando le adversarial networks in un ambiente di gioco finito,

GANG finite. Questo tipo di soluzione teoricamente non soffre di LNE ma in pratica è computazionalmente intrattabile poiché intende analizzare tutte le possibili strategie candidate che finiscono in un LNE. Pertanto gli autori hanno escogitato una soluzione basata su Resource-Bounded Best-Responses (RBBR) che cerca il NE limitato dalle risorse (RB-NE), in particolare si concentra su un sottoinsieme di strategie e usando una ricerca euristica prova a fornire nuove strategie che ampiano lo spazio di ricerca.

Per quanto riguarda i GAN di Coulomb, l'idea è di modellare il problema dell'apprendimento come un campo potenziale in cui gli esempi generati si respingono per evitare la replica dell'output.

L'ispirazione viene dalla fisica elettrostatica dove le cariche elettriche situate nello spazio generano ciascuna un campo elettrico che interessa l'intero sistema. Nel GAN, le generazioni possibili sono l'equivalente delle cariche e per mantenere una situazione di equilibrio globale (in questo caso un Global Nash Equilibrium) dovrebbero essere disposte in una configurazione adeguata mantenendo le distanze adeguate.

## 5 StyleGAN e StyleGAN2

Le reti GAN sono un concetto relativamente nuovo nel panorama del Machine Learning, introdotte per la prima volta nel 2014[2]. Il loro obiettivo è sintetizzare campioni artificiali, come ad esempio immagini, che siano indistinguibili dalle immagini originali. Un esempio comune di un'applicazione GAN è la generazione di immagini di volti artificiali imparando da un set di dati di volti di celebrità. Sebbene le immagini GAN siano diventate sempre più realistiche nel tempo, una delle loro principali sfide è il controllo del loro output, ovvero la modifica di caratteristiche specifiche come la posa, la forma del viso e l'acconciatura dei capelli.

Una pubblicazione di NVIDIA, *A Style-Based Generator Architecture for GANs (StyleGAN)*[3], presenta un nuovo modello che affronta questa sfida. StyleGAN genera l'immagine artificiale gradualmente, partendo da una risoluzione molto bassa e proseguendo fino ad una risoluzione elevata ( $1024 \times 1024$ ). Modificando l'input di ogni livello separatamente, è possibile controllare le caratteristiche visive che sono espresse in quel livello, dalle caratteristiche grossolane (posa, forma del viso) ai dettagli fini (colore dei capelli), senza influire sugli altri livelli.

Questa tecnica non solo consente una migliore comprensione dell'output generato, ma produce anche risultati all'avanguardia: immagini ad alta risoluzione che sembrano più autentiche rispetto alle immagini generate in precedenza.

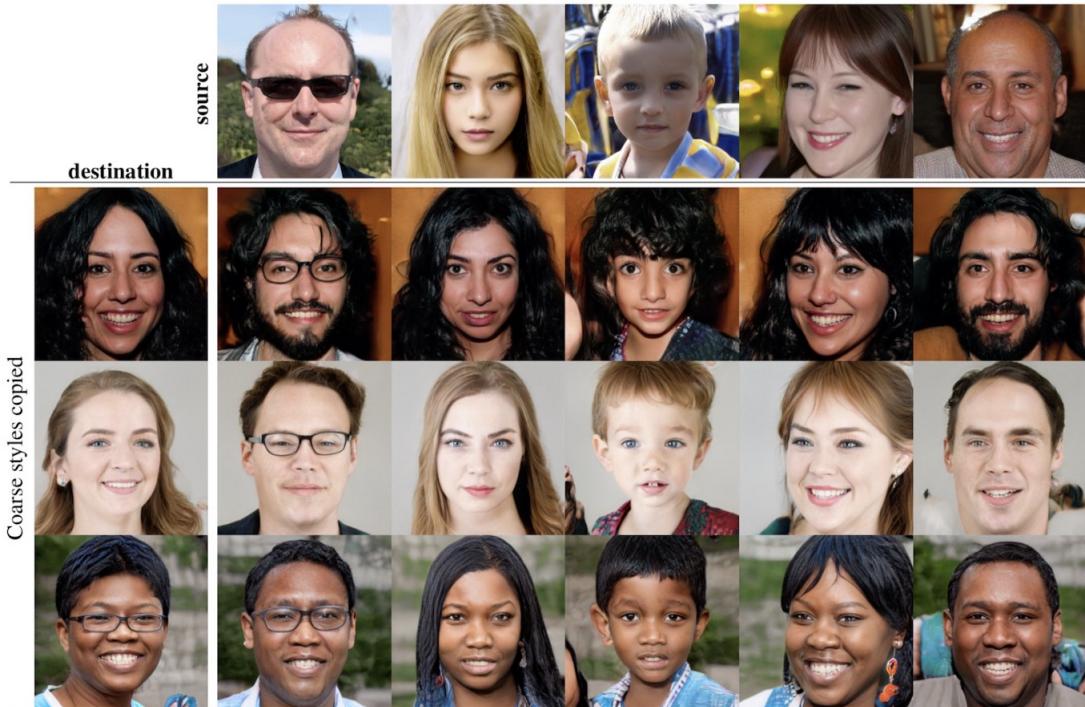


Figura 5.1: StyleGAN mix di stili

## 5.1 Background

I componenti di base di ogni GAN sono due reti neurali: un generatore che sintetizza nuovi campioni da zero e un discriminatore che preleva campioni sia dai dati di addestramento che dall'output del generatore e prevede se sono "reali" o "falsi". L'ingresso del generatore è un vettore casuale ( $\mathbf{z}$  o rumore) e quindi anche la sua uscita iniziale sarà rumore. Nel tempo, ricevendo feedback dal discriminatore, il generatore impara a sintetizzare immagini sempre più “realistiche”. Anche il discriminatore migliora nel tempo confrontando i campioni generati con campioni reali, rendendo più difficile per il generatore ingannarlo.

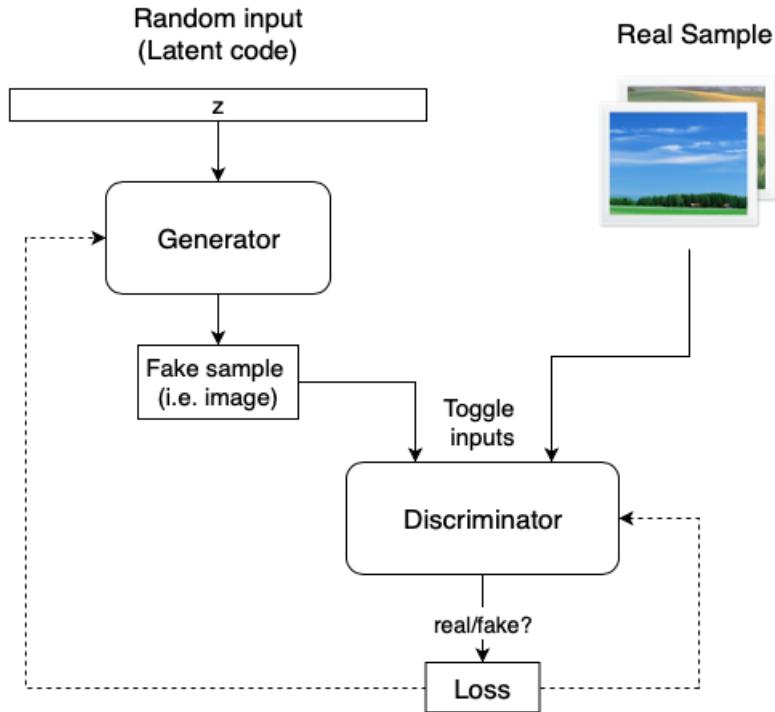


Figura 5.2: GAN overview

I ricercatori hanno avuto problemi a generare immagini di grandi dimensioni ad alta qualità (ad es.  $1024 \times 1024$ ) fino al 2018, quando NVIDIA affronta per la prima volta la sfida con ProGAN[4].

L’innovazione chiave di ProGAN è l’allenamento progressivo: inizia addestrando il generatore e il discriminatore con un’immagine a risoluzione molto bassa (ad es.  $4 \times 4$ ) e aggiunge ogni volta un livello di risoluzione più elevato.

Questa tecnica crea prima le basi dell’immagine apprendendo le caratteristiche di base che appaiono in un’immagine a bassa risoluzione e apprende sempre più dettagli all’aumentare della risoluzione. Allenare le immagini a bassa risoluzione non è solo più facile e veloce, ma aiuta anche ad allenare i livelli più alti e, di conseguenza, anche l’allenamento totale è più veloce.

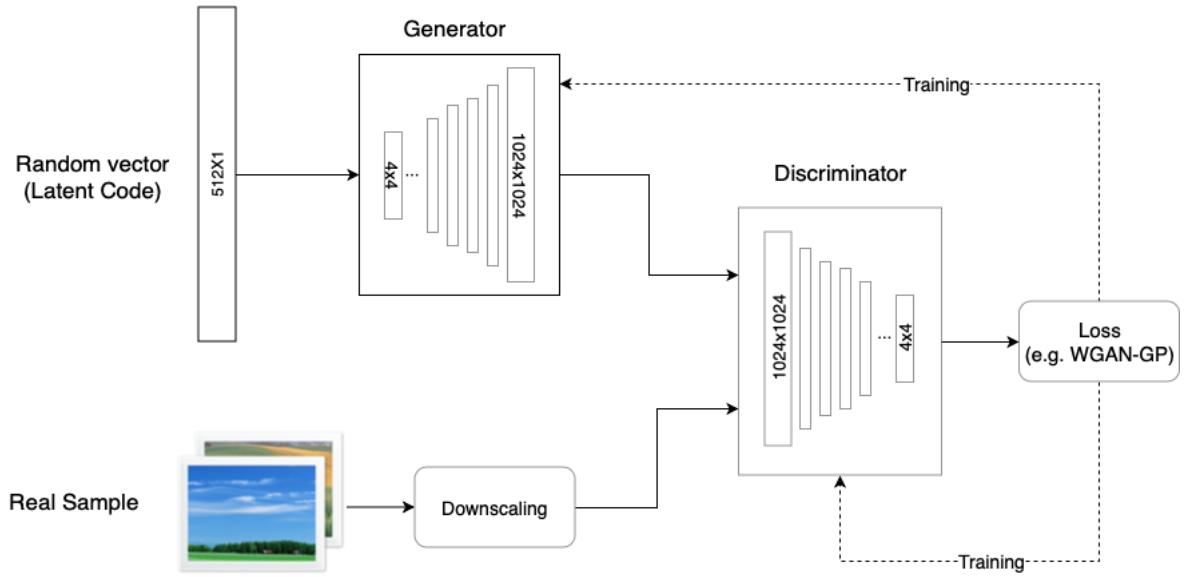


Figura 5.3: PROGAN

ProGAN genera immagini di alta qualità ma, come nella maggior parte dei modelli, la sua capacità di controllare caratteristiche specifiche dell'immagine generata è molto limitata. In altre parole, le funzionalità sono intrecciate e quindi il tentativo di modificare l'input, anche solo leggermente, di solito influisce su più funzionalità contemporaneamente. Una buona analogia sarebbe quella dei geni, in cui la modifica di un singolo gene potrebbe influenzare più tratti.

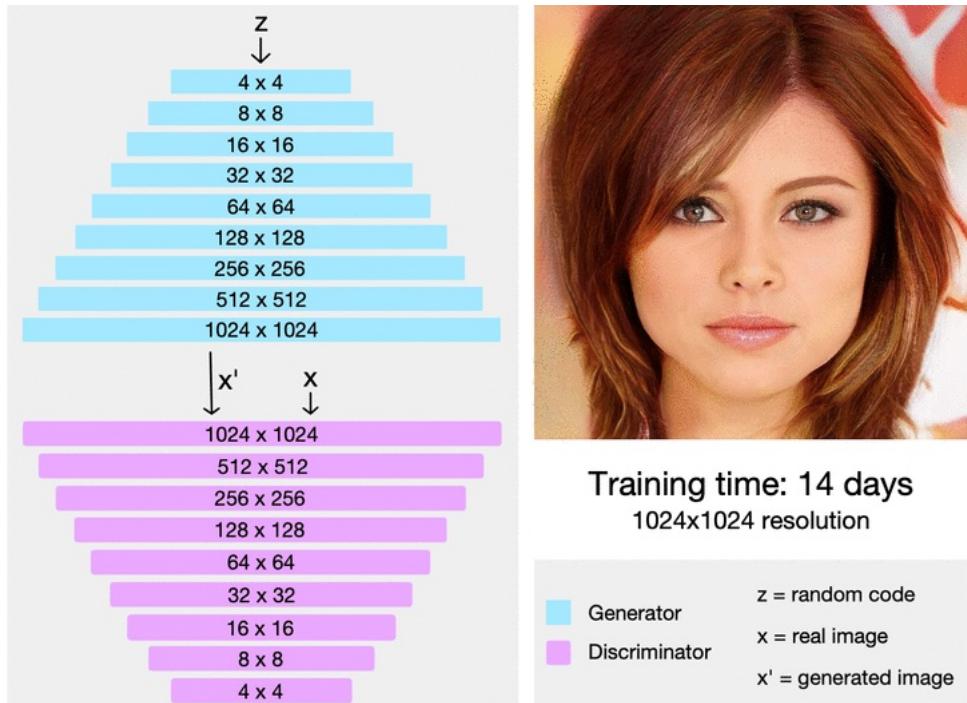


Figura 5.4: PROGAN Training

## 5.2 StyleGAN dettagli

La pubblicazione relativa al StyleGAN offre una versione aggiornata del generatore di immagini di ProGAN, con particolare attenzione alla rete del generatore. Gli autori osservano che un potenziale vantaggio dei livelli progressivi del ProGAN è la loro capacità di controllare diverse caratteristiche visive dell'immagine, se utilizzate correttamente. Più basso è il livello (e la risoluzione), più grossolane saranno le caratteristiche che influiscono. Il documento divide le caratteristiche in tre tipi:

1. **Coarse**: risoluzione fino a 82 pixel - influenza la posa, l'acconciatura generale, la forma del viso, ecc
2. **Middle**: risoluzione da 162 a 322 pixel - influisce sulle caratteristiche facciali più fini, sull'acconciatura, sugli occhi aperti/chiusi, ecc.
3. **Fine**: risoluzione da 642 a 1024 pixel - influisce sulla combinazione di colori (occhi, capelli e pelle) e sulle micro caratteristiche.

Il nuovo generatore include diverse aggiunte rispetto ai generatori di ProGAN:

### 5.3 Mapping Network

L'obiettivo del Mapping Network è codificare il vettore di input in un vettore intermedio i cui diversi elementi controllano diverse caratteristiche visive. Questo è un processo non banale poiché la capacità di controllare le caratteristiche visive con il vettore di input è limitata, poiché deve seguire la densità di probabilità dei dati di addestramento. Ad esempio, se le immagini di persone con i capelli neri sono più comuni nel set di dati, verranno mappati più valori di input su quella funzione. Di conseguenza, il modello non è in grado di mappare parti dell'input (elementi nel vettore) a determinate caratteristiche, un fenomeno chiamato entanglement delle caratteristiche. Tuttavia, utilizzando un'altra rete neurale il modello può generare un vettore che non deve seguire la distribuzione dei dati di addestramento e può ridurre la correlazione tra le caratteristiche.

La rete di mappatura è composta da 8 livelli completamente connessi e il suo output  $\mathbf{W}$  ha le stesse dimensioni del livello di input ( $512 \times 1$ ).

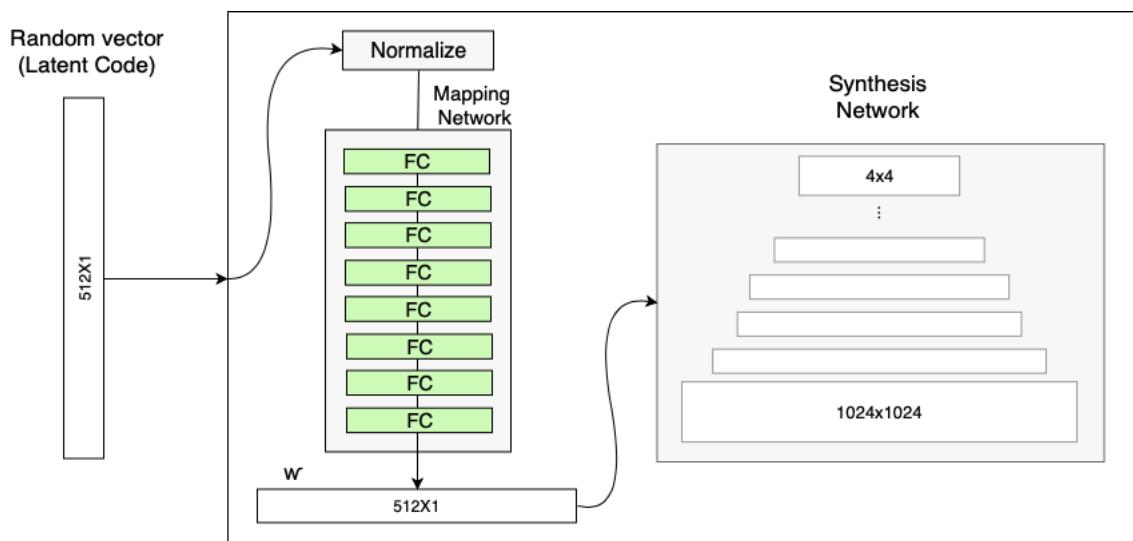


Figura 5.5: PROGAN Mapping Network

## 5.4 Style Modules (AdaIN)

Il modulo AdaIN[5] (Adaptive Instance Normalization) trasferisce le informazioni codificate  $\mathbf{W}$ , create dal Mapping Network, nell'immagine generata. Il modulo viene aggiunto a ciascun livello di risoluzione della rete di sintesi e definisce l'espressione visiva delle caratteristiche in quel livello:

1. Ciascun canale dell'output del livello di convoluzione viene prima normalizzato per assicurarsi che il ridimensionamento e lo spostamento del passaggio 3 abbiano l'effetto previsto.
2. Il vettore intermedio  $\mathbf{W}$  viene trasformato utilizzando un altro livello completamente connesso (contrassegnato come  $\mathbf{A}$ ) in una scala e una polarizzazione per ciascun canale.
3. I vettori di scala e di polarizzazione spostano ciascun canale dell'uscita della convoluzione, definendo così l'importanza di ciascun filtro nella convoluzione. Questa ottimizzazione traduce le informazioni da  $\mathbf{W}$  in una rappresentazione visiva.

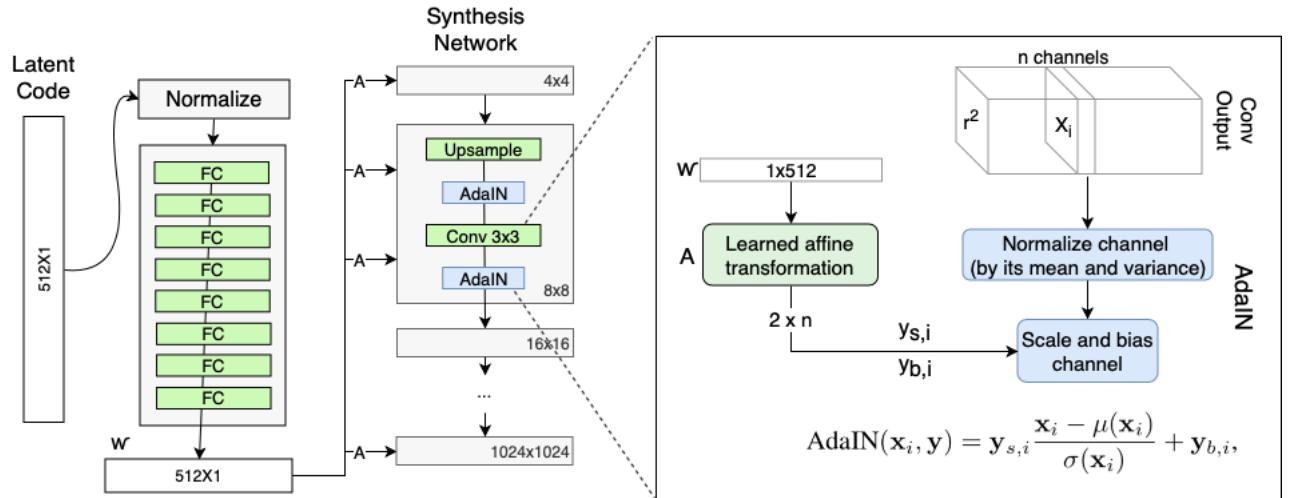


Figura 5.6: ADAIN

## 5.5 Rimuovere gli input tradizionali

La maggior parte dei modelli, e tra questi ProGAN, utilizzano l'input casuale per creare l'immagine iniziale del generatore (cioè l'input del livello  $4 \times 4$ ). Il team di StyleGAN ha scoperto che le caratteristiche dell'immagine sono controllate da  $\mathbf{W}$  e AdaIN, e quindi l'input iniziale può essere omesso e sostituito da valori costanti. Sebbene il documento non spieghi perché migliora le prestazioni, un presupposto sicuro è che riduca l'entanglement delle funzionalità: è più facile per la rete imparare solo utilizzando  $\mathbf{W}$  senza fare affidamento sul vettore di input entangled.

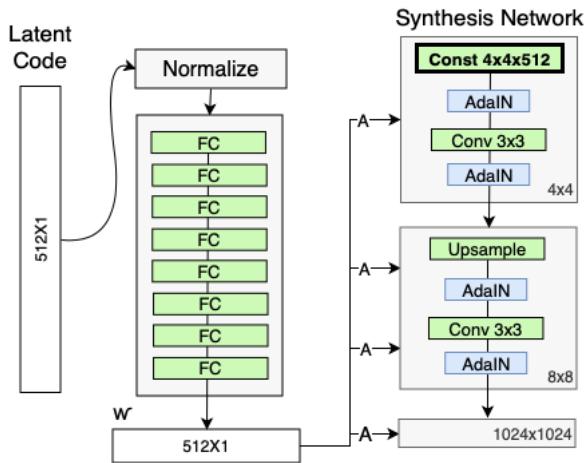


Figura 5.7: Input Costante

## 5.6 Variazione Stocastica

Ci sono molti aspetti nei volti delle persone che sono piccoli e possono essere visti come stocastici, come le lentiggini, la posizione esatta dei capelli, le rughe, caratteristiche che rendono l'immagine più realistica e aumentano la varietà di output. Il metodo comune per inserire queste piccole caratteristiche nelle immagini GAN consiste nell'aggiungere rumore casuale al vettore di input. Tuttavia, in molti casi è difficile controllare l'effetto del rumore a causa del fenomeno di entanglement delle caratteristiche descritto sopra, che porta a influenzare altre caratteristiche dell'immagine.

Il rumore in StyleGAN viene aggiunto in modo simile al meccanismo AdaIN — Un

rumore scalato viene aggiunto a ciascun canale prima del modulo AdaIN e cambia leggermente l'espressione visiva delle caratteristiche del livello di risoluzione su cui opera.

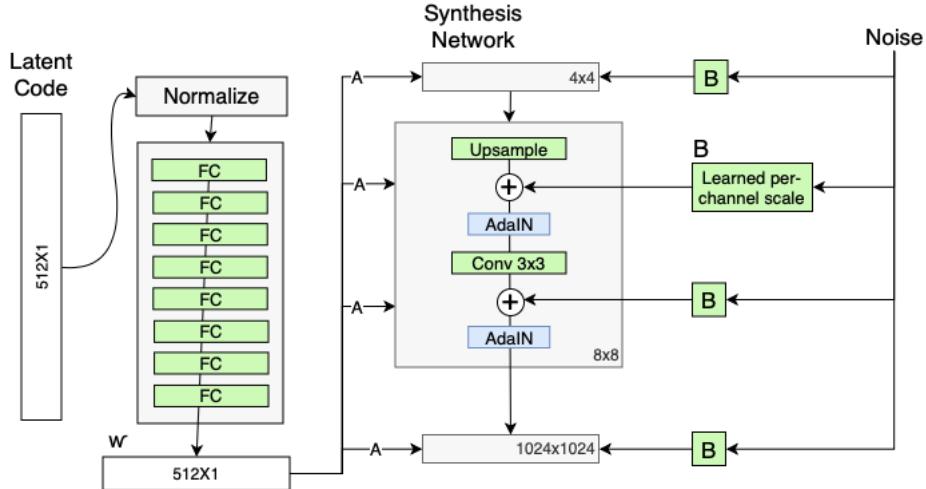


Figura 5.8: Rumore

## 5.7 Mix di stili

Il generatore StyleGAN utilizza il vettore intermedio in ogni livello della rete di sintesi, il che potrebbe far sì che la rete apprenda che i livelli sono correlati. Per ridurre la correlazione, il modello seleziona casualmente due vettori di input e genera per essi il vettore intermedio **W**. Quindi allena alcuni dei livelli con il primo e passa (in un punto casuale) all'altro per allenare il resto dei livelli. Lo switch casuale assicura che la rete non impari e si basi su una correlazione tra i livelli.

Sebbene non migliori le prestazioni del modello su tutti i set di dati, questo concetto ha un effetto collaterale molto interessante: la sua capacità di combinare più immagini in modo coerente. Il modello genera due immagini **A** e **B** e poi le combina prendendo le caratteristiche di basso livello da **A** e il resto delle caratteristiche da **B**.



Figura 5.9: Unione di 2 immagini. L’immagine al centro é il risultato dell’unione tra l’immagine di destra e l’immagine di sinistra

## 5.8 Sapere quando fermarsi

Una delle sfide nei modelli generativi è occuparsi di aree scarsamente rappresentate nei dati di addestramento. Il generatore non è in grado di impararli e creare immagini somiglianti (e invece crea immagini poco significative). Per evitare di generare immagini scadenti, StyleGAN tronca il vettore intermedio  $\mathbf{W}$ , costringendolo a rimanere vicino al vettore intermedio “medio”.

Dopo aver addestrato il modello, viene prodotta una media  $\mathbf{W}_{avg}$  selezionando molti input casuali; generare i loro vettori intermedi con la rete di mappatura; e calcolare la media di questi vettori. Quando si generano nuove immagini, invece di utilizzare direttamente l’output di Mapping Network,  $\mathbf{W}$  viene trasformato in  $\mathbf{W}_{new} = \mathbf{W}_{avg} + \Psi(\mathbf{W} - \mathbf{W}_{avg})$ , dove il valore di  $\Psi$  definisce quanto può essere lontana l’immagine dall’immagine “media” (e quanto diverso può essere l’output). È interessante notare che, utilizzando un  $\Psi$  diverso per ogni livello, prima del blocco di trasformazione affine, il modello può controllare quanto è lontano dalla media ciascun insieme di funzionalità.

## 5.9 Tuning

Un ulteriore miglioramento di StyleGAN su ProGAN è stato l’aggiornamento di diversi iperparametri di rete, come la durata dell’allenamento e la funzione di

perdita, e la sostituzione dell'aumento/ridimensionamento dai vicini più vicini al campionamento bilineare.

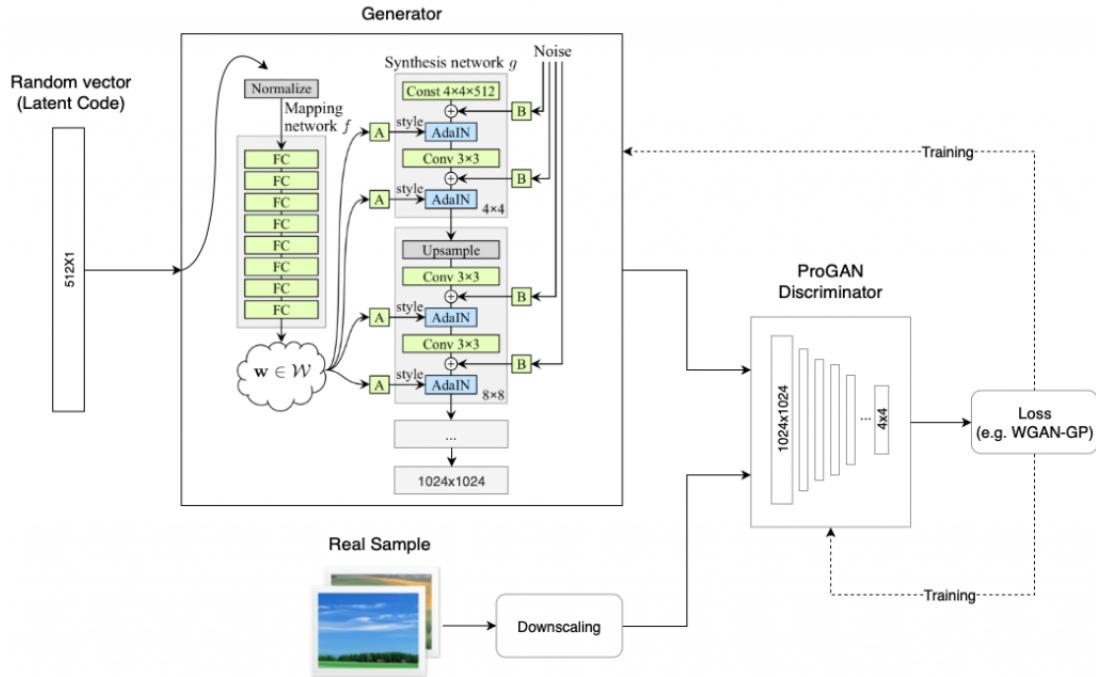


Figura 5.10: StyleGAN Overview

## 5.10 StyleGAN2

StyleGAN2[9] nasce per migliorare alcuni difetti presenti nelle immagini generate con StyleGAN[10].

Nell'immagine sottostante, si possono vedere i difetti o la porzione sfocata nell'immagine generata con StyleGAN che proviene dalla risoluzione iniziale 64x64. Questa è la ragione principale che ha portato a ridisegnare la parte del generatore, migliorare la qualità delle immagini generate.

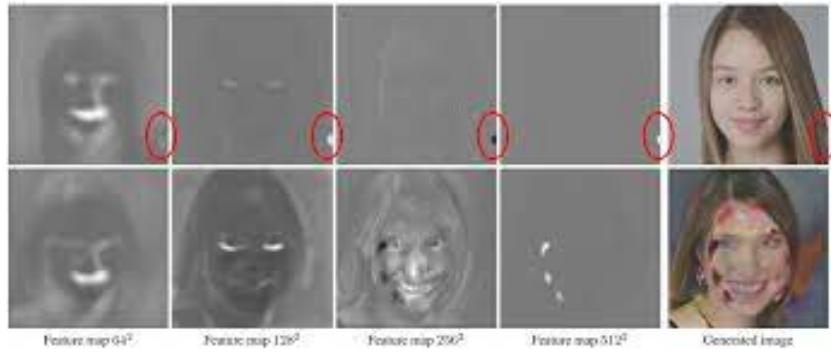


Figura 5.11: Difetti immagini generate con StyleGAN

Nell’immagine sottostante si possono vedere quali cambiamenti nell’architettura della rete migliorano le prestazioni dell’immagine generata passo dopo passo.

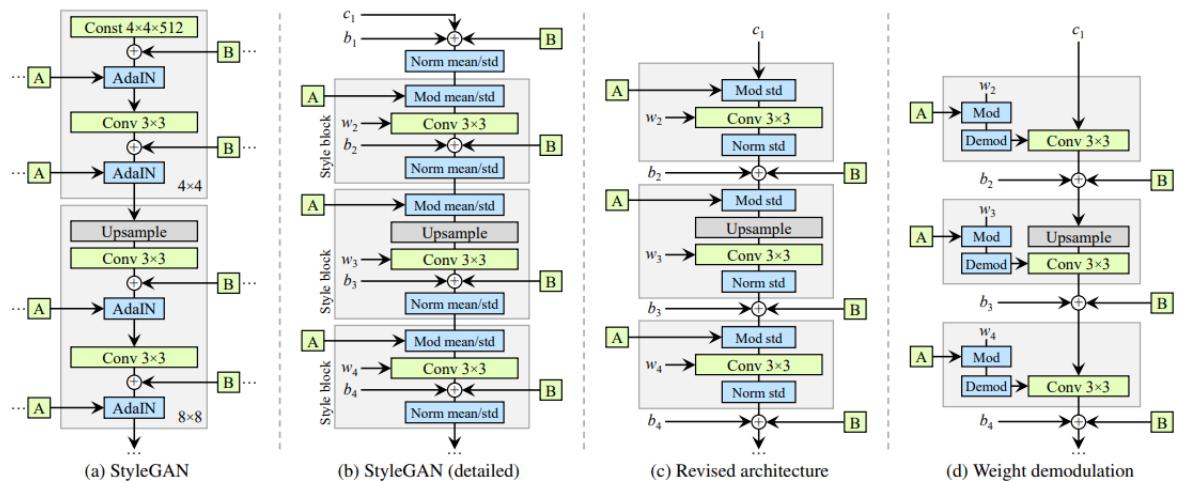


Figura 5.12: Evoluzione architettura StyleGAN

La parte A è l’architettura di StyleGAN, la parte B mostra la vista dettagliata dell’architettura di Stylegan. Nella parte C, é stato sostituito l’ADA-IN (Adaptive Istance Normalization) con la modulazione (o il ridimensionamento dei fattori) e della normalizzazione. Di seguito è possibile vedere le equazioni per la modulazione (5.1) e la normalizzazione (5.2).

$$w_{ijk} = s_i w_{ijk} \quad (5.1)$$

$$\sigma_j = \sqrt{\sum_{i,j} w_{ijk}^2} \quad (5.2)$$

Inoltre, nella parte C é stata spostata l'aggiunta di rumore e bias al di fuori del blocco. Nella parte D si puó vedere che i pesi sono regolati con lo stile e la normalizzazione viene sostituita con un'operazione di "demodulazione", le operazioni combinate sono chiamate "demodulazione del peso"(5.3).

$$w_{ijk}^* = \frac{w_{ijk}}{\sqrt{\sum_{i,j} w_{ijk}^2} + \epsilon} \quad (5.3)$$

Si puó vedere che l'equazione 5.3 sembra la combinazione delle due equazioni di modulazione 5.1 e di normalizzazione 5.2 di cui sopra (Epsilon è un piccolo valore costante, utilizzato per evitare problemi numerici come divisione per zero). I risultati possono essere visualizzati sulle uscite seguenti, dopo aver sostituito la normalizzazione con la demodulazione rimuove i manufatti simili a goccioline.

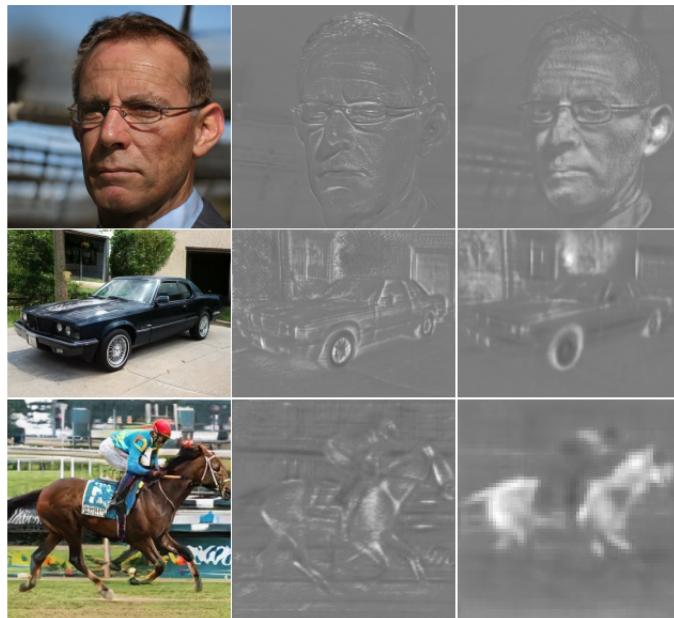


Figura 5.13: Immagini demodulate

Una volta visto i miglioramenti sotto forma di immagini generate. Vediamo i miglioramenti misurati usando metriche come la FID (Frechet Inception Distance), la lunghezza percettiva del percorso (introdotta nel documento di Stylegan, abbassando il PPL l'immagine viene generata meglio).

Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
	FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A Baseline StyleGAN [24]	4.40	212.1	<b>0.721</b>	0.399	3.27	1484.5	<b>0.701</b>	0.435
B + Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C + Lazy regularization	4.38	158.0	0.719	0.427	2.83	981.6	0.688	0.493
D + Path length regularization	4.34	<b>122.5</b>	0.715	0.418	3.43	651.2	0.697	0.452
E + No growing, new G & D arch.	3.31	124.5	0.705	0.449	3.19	471.2	0.690	0.454
F + Large networks (StyleGAN2)	<b>2.84</b>	145.0	0.689	<b>0.492</b>	<b>2.32</b>	<b>415.5</b>	0.678	<b>0.514</b>
Config A with large networks	3.98	199.2	0.716	0.422	–	–	–	–

Figura 5.14: Risultati StyleGAN2

L'immagine precedente riporta la tabella dove si possono vedere i miglioramenti delle configurazioni dopo aver applicato metodi diversi. La regolarizzazione della lunghezza del percorso e la regolarizzazione Lazy vengono utilizzate per mantenere il punteggio PPL basso, e mostra che le immagini generate sono più chiare o nitide. La crescita progressiva della rete genera immagini di alta qualità ma causa anche gli artefatti caratteristici (o gli artefatti di fase) cioè gli occhi e i denti della persona sembrano bloccati in uno stesso punto ovunque si muove il volto della persona.



Figura 5.15: Occhi e naso restano fissi

Per risolvere questo problema sono state provate altre connessioni sia sul generatore

che sulla rete discriminatrice e hanno visto che il salto della connessione funziona meglio per il generatore, e le reti residue danno risultati migliori sul discriminatore. Nella figura inferiore, nella parte B è raffigurato un generatore e in C un discriminatore senza crescita progressiva.

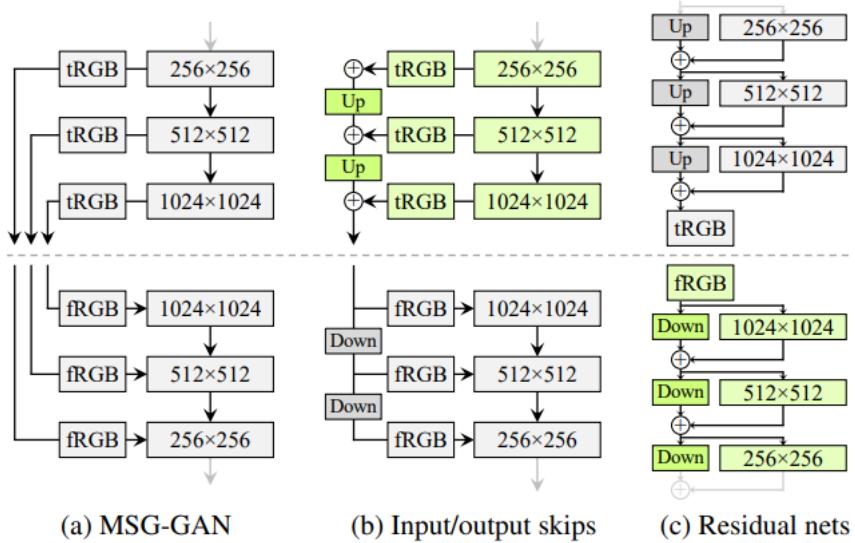


Figura 5.16: Generatore parte superiore e discriminantore parte inferiore

# 6 Esplorare lo spazio latente

Il concetto di "spazio latente" è importante perché la sua utilità è al centro del "Deep Learning", apprendendo le caratteristiche dei dati e semplificarne le rappresentazioni allo scopo di trovare modelli.

## 6.1 Background

Immaginiamo un dataset di immagini raffiguranti numeri scritti a mano. Immagini di numeri raffiguranti lo stesso numero sono le più simili l'una con l'altra rispetto ad altre immagini di numeri diversi (ad esempio il numero 3 rispetto al numero 1). È possibile scrivere un algoritmo per riconoscere queste somiglianze?

Se è stato addestrato un modello per classificare le cifre, allora è stato addestrato anche un modello per apprendere le "somiglianze strutturali" tra le immagini. In effetti, questo è il modo in cui il modello è in grado di classificare le cifre, imparando le caratteristiche di ogni cifra.

Questo processo di riconoscimento è appunto latente (nascosto).

## 6.2 Compressione dei dati

La compressione dei dati è definita come il processo di codifica delle informazioni utilizzando meno bit rispetto alla rappresentazione originale.

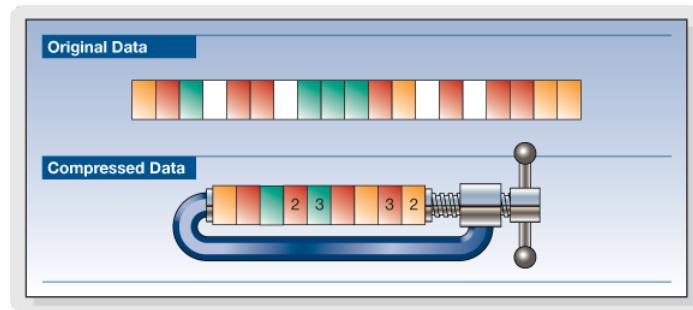


Figura 6.1: Compressione dei dati

Nel machine learning la compressione dei dati è un concetto molto importante in quanto viene usata per apprendere informazioni sui dati.

Se volessimo addestrare un modello per classificare immagini utilizzando una fully convolutional neural network (FCN). Il modello 'impara', le caratteristiche di apprendimento ad ogni livello (bordi, angoli, ecc.) e attribuisce una combinazione di funzionalità ad ogni specifico output.

Ogni volta che il modello impara attraverso un punto di dati, la dimensione dell'immagine viene compressa prima di essere ricostruita (come mostrato nell'immagine sottostante).

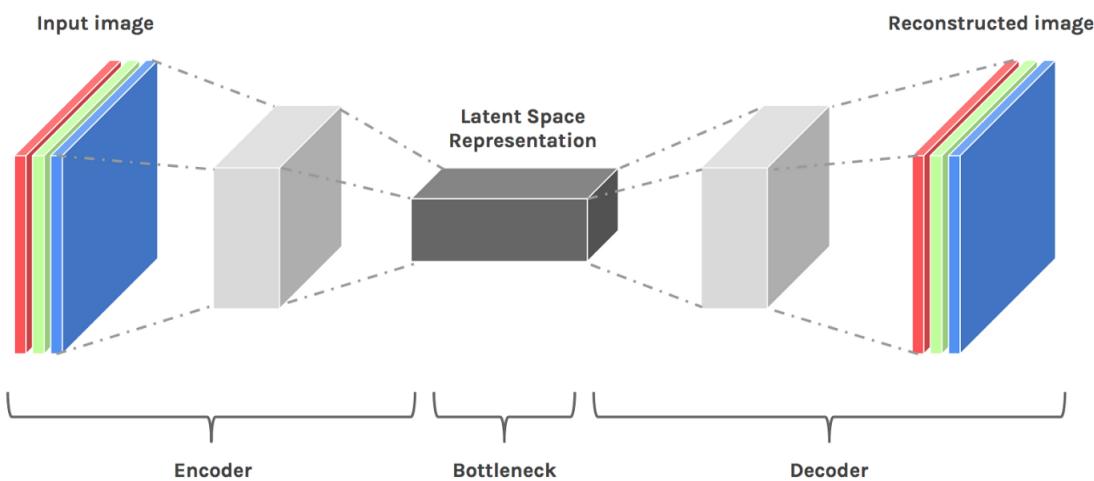


Figura 6.2: Rappresentazione di una CNN

Poiché il modello è necessario per poi ricostruire i dati compressi (Decoder), deve

imparare a memorizzare tutte le informazioni pertinenti e ignorare il rumore. L'importanza della compressione: consente di sbarazzarsi di qualsiasi informazione non necessarie e si concentra solo sulle funzionalità più importanti.

Questa compressione è la rappresentazione nello spazio latente dei nostri dati.

### 6.3 Definizione di spazio

In questo paragrafo cerchiamo di capire la relazione tra spazio e dati compressi. Ipotizziamo che il dataset è composto da immagini di dimensioni 5x5x1. Lo spazio latente che verrà usato per contenere i dati compressi sarà un vettore di dimensioni 3x1.

Ora, ciascun punto dati compresso è definito in modo univoco da solo 3 numeri.

$$\begin{bmatrix} 1 & 0.5 & 0.8 & 0.2 & 0.7 \\ 0.2 & 0.11 & 0.78 & 0.3 & 0.2 \\ 1 & 0.01 & 0 & 0.9 & 0.56 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0.4 & 0.2 & 0.1 & 0.01 \end{bmatrix}$$

Figura 6.3: Matrice 5x5x1

$$\begin{bmatrix} 0.4 \\ 0.3 \\ 0.8 \end{bmatrix}$$

Figura 6.4: Matrice 3x1

Ciò significa che possiamo graficare questi dati su un piano 3D.

*Questa rappresentazione 3D non è altro che lo spazio latente.*

I punti nello spazio latente, possono essere visti come coordinate nello spazio in cui i punti che sono "simili" sono vicini tra di loro.

Questa natura 3D dello spazio latente porta ad una perdita di informazioni nel momento in cui si vogliono elaborare dati nello spazio n-dimensionale dove  $n > 3$ . Per rappresentazioni nello spazio n-dimensionale si può fare riferimento all'algoritmo t-SNE[11].

## 6.4 Concetto di simili

Guardando a tre immagini, due raffiguranti sedie e la terza raffigurante una scrivania, le due immagini delle sedie sono le più simili mentre la scrivania è la più diversa dalle altre.



Figura 6.5: Esempio di simili

Cosa rende queste due immagini "più simili?" Una sedia ha caratteristiche distinguibili (lo schienale, nessun cassetto, connessioni tra le gambe). Questi elementi possono essere "compresi" dai nostri modelli imparando dai pattern nei bordi, angoli, ecc.

Come è stato visto in precedenza, questi concetti sono presenti nella rappresentazione dei dati nello spazio latente.

Quando avviene la compressione, le informazioni non necessarie come il colore, vengono rimosse dalla rappresentazione nello spazio latente in quanto non sono informazioni significative.

Riducendo lo spazio dimensionale, le caratteristiche delle due sedie diventano sempre più simili. Rappresentando questi punti nello spazio tridimensionale, questi punti saranno vicini tra di loro.

## 6.5 Importanza dello spazio latente

La rappresentazione nello spazio latente dei nostri dati contiene tutte le informazioni importanti necessarie per rappresentare i nostri punti.

Questa rappresentazione deve contenere le caratteristiche dei dati originali.

Ovvero, il modello impara le caratteristiche dei dati e le semplifica per semplificare la loro analisi.

Questo è il cuore di un concetto chiamato Representation Learning, definito come un insieme di tecniche che consentono a un sistema di scoprire le rappresentazioni necessarie per il rilevamento delle caratteristiche o la classificazione dai dati grezzi.

In questo use case, le nostre rappresentazioni dello spazio latente vengono utilizzate per trasformare forme più complesse di dati grezzi (ad esempio immagini, video), in rappresentazioni più semplici che sono "più facili da elaborare" e analizzare.

### 6.5.1 Autoencoder e Modelli Generativi

Un tipo comune di modello di deep learning che manipola la "vicinanza" dei dati nello spazio latente è l'autoencoder, una rete neurale che funge da funzione di identità. In altre parole, un codificatore automatico impara a produrre tutto ciò che viene immesso.

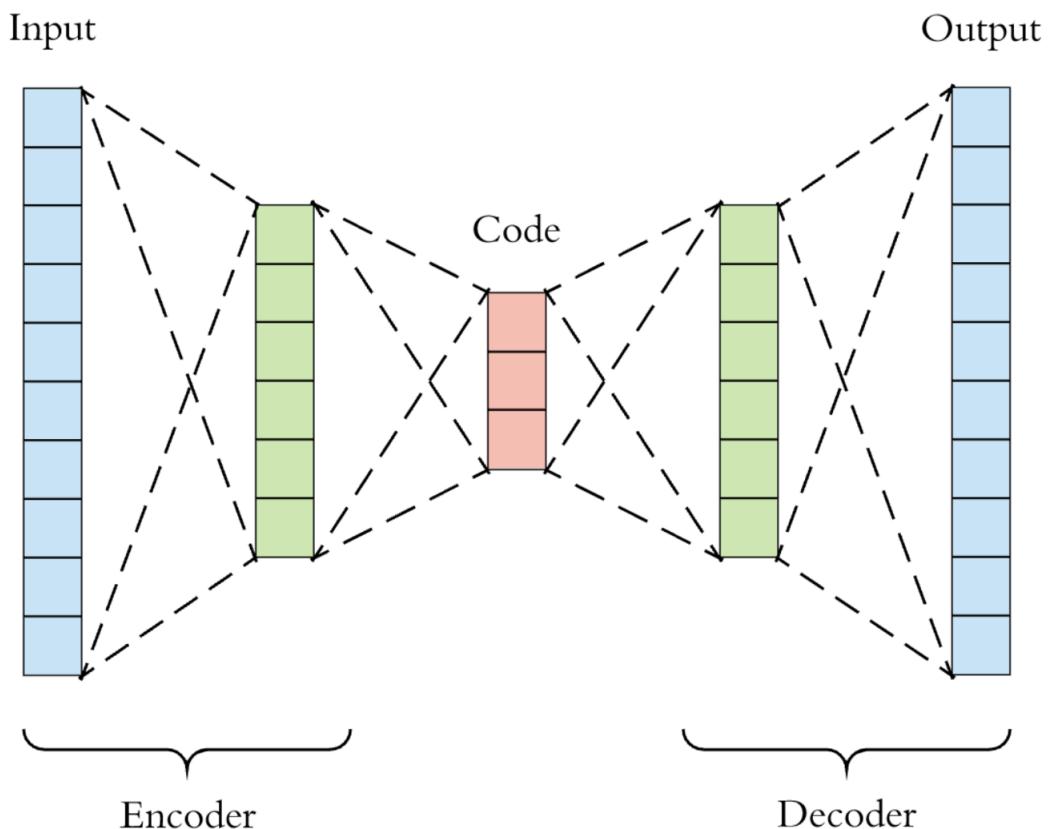


Figura 6.6: Autoencoder

Quando si forza un modello a diventare una funzione di identità, lo si costringe a memorizzare tutte le caratteristiche rilevanti dei dati in una rappresentazione compressa in modo che i dati compressi contengano sufficienti informazioni da permettere al modello di ricostruire l'input in maniera fedele. Questo non è altro che lo spazio latente.

Si è visto come i modelli possano essere scoperti più facilmente nello spazio latente poiché punti di dati simili tenderanno a raggrupparsi insieme, ma non si è ancora visto come sia possibile campionare punti da questo spazio latente per generare apparentemente "nuovi" dati.

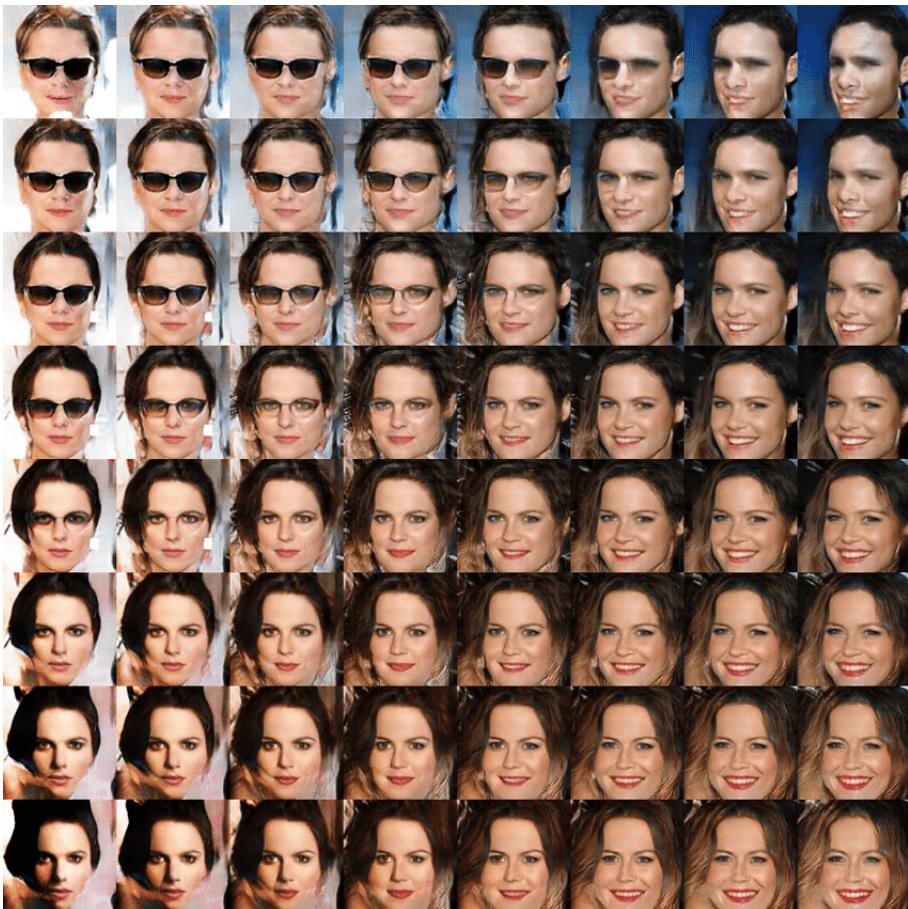


Figura 6.7: Immagini generate usando lo spazio latente

L'immagine precedente contiene visi generati interpolando lo spazio latente, e usando il decoder per ricostruire la rappresentazione dello spazio latente in un'immagine 2D con le stesse dimensioni dell'input originale.

### 6.5.2 Interpolazione dello spazio latente

Supponiamo di aver compresso le immagini della sedia della sezione precedente nei seguenti vettori 2D, [0.8, 0.2] e [0.22, 0.35]. Supponiamo che la scrivania è compressa a [0.6, 0.75]. Interpolando lo spazio latente, verranno usati i punti nello spazio latente tra il cluster "sedia" e il cluster "scrivania".

Inserendo questi vettori 2D campionati nel decodificatore del modello si ottengono

immagini "nuove" che sembrano una metamorfosi tra una sedia e una scrivania.

Nuove è tra virgolette perché queste immagini generate non sono tecnicamente indipendenti dal campione di dati originale.

La generazione di immagini è ancora un'area attiva di ricerca e lo spazio latente è un concetto essenziale che deve essere compreso.



Figura 6.8: Interpolazione di dati, l'immagine centrale é stata generata interpolando le immagini di destra e sinistra

# 7 Caso Sperimentale: Applicazione dello spazio latente a StyleGan2

Questo ultimo capitolo é dedicato a una delle possibili applicazioni pratiche dello StyleGAN2[3] e il suo spazio latente.

Andremo ad utilizzare lo StyleGAN2, sviluppato da NVIDIA, applicato su uno dei tanti dataset pubblici, nello specifico: FFHQ (Flicker-Faces-HQ). Il training di questo dataset é stato eseguito su una collezione contenente circa 70.000 volti umani (ad alta risoluzione 1024x1024) presi dal popolare social network fotografico Flickr[12].

L'idea é quella di generare volti che non esistono, proiettarli nello spazio latente per poi modificarne alcuni parametri come: l'eta, il genere, il sorriso, la dimensione delle labbra, l'apertura della bocca, l'apertura degli occhi e molti altri.

Il codice che verrá proposto é stato organizzato in modo tale da riuscire ad accettare in input non solo immagini generate tramite l'algoritmo StyleGAN, ma anche immagini proprietarie.

L'implementazione usata é quella originale risalente al 2019. Essendo il GAN un argomento molto popolare nel mondo del machine learning, nel corso del tempo a questa implementazione ne sono seguite altre, la lista completa é possibile reperirla al seguente link: <https://nvlabs.github.io/stylegan2/versions.html>.

Dato che in ogni progetto di Machine Learning uno degli aspetti piú importanti, ma anche difficile da reperire é il dataset da sfruttare, inoltre, il dataset deve essere abbastanza grande da considerare tutti i possibili casi ed evitare che il modello si adatti solo a un sottoinsieme di essi. Per risolvere questo inconveniente gli sviluppatori hanno messo a disposizione (<https://nvlabs-fi-cdn.nvidia.com/stylegan2/networks/>) diversi dataset:

- FFQH: contenente volti umani ad alta risoluzione 1024x1024
- CAT: contenente gatti in varie risoluzioni
- CAR: contenente auto
- HORSE: contenente cavalli
- CHURCH: contenente chiese

In rete è possibile trovare diversi tipi di dataset già pronti da usare, oppure è possibile crearsi il proprio con le istruzioni che sono contenute nella documentazione. Per motivi di tempo, per lo svolgimento di questa tesi, si è deciso di usare uno dei dataset messi a disposizione.

## 7.1 Background

Per lo svolgimento relativo la parte sperimentale della tesi è stato scelto Google Colab[13].

Dato che il codice sorgente messo a disposizione da NVIDIA è in linguaggio Python che fa uso di librerie Tensorflow, Google Colab si è rivelata la scelta ideale per la stesura della parte sperimentale del progetto in quanto mette a disposizione un notebook Jupyter[14], con il relativo supporto alle principali librerie Python incluse quelle di machine learning, e il supporto a GPU e TPU utili nello svolgimento e implementazione di progetti in ambito Machine e Deep Learning.

### 7.1.1 Python

Python è un linguaggio di programmazione open source introdotto nel 1991 da Guido Van Rossum e poi ulteriormente sviluppato nei successivi 25 anni dalla Python Software Foundation[16].

Python è un linguaggio multipiattaforma in quanto può essere integrato molto facilmente con molti strumenti e può essere eseguito su diverse piattaforme, tra cui piattaforme cloud come AWS o Google Cloud Platform.

Attualmente numerosi siti web sono realizzati utilizzando il linguaggio di programmazione Python e molte aziende stanno sviluppando applicazioni di successo con

questo linguaggio. Questo è il motivo per cui è considerato il linguaggio di oggi e del futuro. Anche se Python ha ancora molta strada da fare, rappresenta un linguaggio fondamentale nel mondo dell'IA.

Con la sua comunità in continua crescita: sviluppatori, ricercatori e appassionati di apprendimento, viene ottimizzato e migliorato ogni giorno. Poiché la domanda di professionisti dell'IA è in aumento, così come la domanda di Python ciò significa che nel corso degli anni l'industria si rivolgerà gradualmente verso esperti che hanno una buona presa su questo linguaggio. Molte aziende hanno persino proposto la migrazione del loro sistema dal vecchio codice al software dipendente da Python oppure abilitare la nuova integrazione Python in modo che i sistemi legacy possano funzionare con Python in futuro. Nei prossimi anni, ci sarà molto sviluppo nel deep learning e nell'apprendimento automatico basato sul linguaggio Python.

### 7.1.2 Google Colab

Google Colab è una piattaforma usata in ambito Deep e Machine Learning. Permette di acquisire esperienza e sviluppare un'intuizione su aspetti di deep learning come l'ottimizzazione degli iperparametri, la preelaborazione dei dati, la complessità del modello, l'overfitting ed altro ancora.

I notebook Jupyter e di conseguenza anche Google Colab vengono usati dagli adetti ai lavori in ambito di intelligenza artificiale e machine learning, in quanto mettono a disposizione una piattaforma che fornisce una notevole potenza di calcolo utile per l'implementazione di modelli abbastanza robusti e efficienti. Mentre per le prime sperimentazioni può essere sufficiente affidarsi alla propria macchina, con l'aumentare della dimensione e la sempre crescente risoluzione dei dataset, l'esecuzione di algoritmi di addestramento complessi come quelli di deep learning diventa rapidamente proibitiva.

Jupyter è uno strumento molto popolare tra i data scientist e gli analisti. Uno dei principali vantaggi è che supporta visualizzazioni e documentazione nel notebook stesso. Pertanto, l'utente non deve eseguire una parte di codice ogni volta che desidera esaminare un particolare elemento visivo. Infatti, una volta che l'oggetto è stato generato nel notebook, è lì per rimanere a meno che non venga sovrascritto con qualche altro output. Jupyter supporta sia R che Python, i due linguaggi più

richiesti e più utilizzati nell'IA.

Google Colab è simile ai notebook Jupyter con l'unica differenza che funziona sul cloud di Google invece di essere connesso al sistema locale ed è disponibile gratuitamente. Il vantaggio principale di Google Colab è che è completamente indipendente dalla configurazione del sistema locale poiché funziona online sul cloud di Google. Ciò significa che l'utente non deve preoccuparsi delle configurazioni prima di iniziare un esperimento o un progetto. Gli utenti possono sfruttare le strutture hardware cloud di Google, inclusi GPU e TPU gratuitamente. Questi servizi sono spesso molto difficili da portare nel sistema locale e, quindi, è una struttura molto apprezzata. Colab viene fornito con librerie preinstallate di Google, comprese tutte le principali librerie AI Python. Quindi, fa risparmiare tempo nella creazione dell'ambiente. Nel caso in cui manchino alcune librerie, possono essere facilmente scaricate tramite il gestore di pacchetti pip, anch'esso integrato in Google Colab. Tutto il lavoro che viene sviluppato su Google Colab è salvato su Google Drive per impostazione predefinita, poiché funziona su Google Cloud System, il che significa che è facilmente accessibile quando e dove serve. I notebook sviluppati da un utente possono anche essere condivisi e integrati con gli input di altri utenti tramite una semplice condivisione di link online e, proprio come in Google Drive, anche in Google Colab sono possibili revisioni e controlli di versione. Quindi si può dire che Google Colab facilita un alto grado di collaborazione.

### 7.1.3 TensorFlow

*TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications[15].*

TensorFlow offre addestramento e previsione distribuiti. Quando si lavora con i dati nel mondo reale, i set di dati sono enormi, non è possibile addestrare il modello su una singola macchina locale, quindi è necessario configurare più macchine in un modulo cluster, in genere su una piattaforma cloud come GCP, AWS o Azure. TensorFlow semplifica l'addestramento distribuito e la previsione, infatti dispone

di API che è possibile utilizzare per creare i modelli di rete neurale. TensorFlow consente la ricerca e la prototipazione di nuovi algoritmi di apprendimento automatico ed è possibile utilizzarlo per lo sviluppo oltre che su scala di produzione. Infatti, TensorFlow dispone di solide API pronte per la produzione che è possibile utilizzare nel mondo reale. Se si lavora su TensorFlow, le persone in genere usano l'API Python, in quanto è stabile, robusta e pronta per la produzione. TensorFlow é considerato efficiente e performante in quanto imposta l'esecuzione di calcolo su una rete neurale.

## 7.2 Implementazione

In questa sezione andremo a descrivere in dettaglio l'applicazione che é stata sviluppata per implementare la parte pratica della tesi.

La soluzione proposta é stata implememntata usando lo StyleGAN2 addestrato sul dataset FFHQ.

Il notebook Colab usato é stato strutturato in diverse sezioni:

Sezione	Descrizione
Caricamento Google Driver	Sezione dedicata al caricamento dello spazio di archiviazione Google Drive. Google Drive è necessario per scaricare il codice sorgente e memorizzare le immagini generate o caricate
Download del codice sorgente	In questa sezione viene scaricato il codice sorgente StyleGAN2 nel Google Drive
Installazione librerie	In questa sezione vengono scaricate e installate alcune librerie utili all'esecuzione dell'applicazione
Installazione TensorFlow	In questa sezione viene scaricata e installata la versione di TensorFlow 1.14
Import Librerie e Setup ambiente	In questa sezione viene fatto il setup iniziale dell'ambiente
Preparazione dello StyleGAN2	In questa sezione vengono dichiarate alcune funzioni per l'istanziamento e l'uso vero e proprio dello StyleGAN2
- Caricamento del modello	In questa sotto sezione viene caricato il modello StyleGAN2 implementato nel passo precedente
Pulizia del Driver	In questa sezione vengono pulite le directory usate per memorizzare le immagini
Generazione dei volti	In questa sezione vengono generate immagini casuali usando StyleGAN2 e il dataset FFHQ
Immagini Custom	In questa sezione è possibile caricare immagini proprietarie per poi proiettarle nello spazio latente
Controlli latenti	In questa sezione vengono elencati i controlli latenti che è possibile usare
Generazione del Widget	In questa sezione viene implementato un widget che rende semplice l'applicazione dei controlli latenti alle immagini scelte

### 7.2.1 Caricamento Google Drive

Sezione dedicata al caricamento dello spazio di archiviazione Google Drive. Google Drive é necessario per scaricare il codice sorgente e memorizzare le immagini generate o caricate.

Un passo preliminare prima di eseguire questo comando é necessario avere accesso al un repository google drive.

---

```
1 from google.colab import drive  
2 drive.mount('/content/drive')  
3 %cd /content/drive/My\Drive/tesi
```

---

Listato 7.1: Caricamento Google Drive

Questa istruzione viene eseguita solo una volta durante la prima esecuzione dell'applicazione.

### 7.2.2 Download del codice sorgente

In questa sezione viene scaricato il codice sorgente StyleGAN2 dal repository GIT ufficiale nel Google Drive.

---

```
1 !git clone  
     https://github.com/AmarSaini/Epoching_StyleGan2_Setup.git
```

---

Listato 7.2: Download codice sorgente

Questa istruzione viene eseguita solo una volta durante la prima esecuzione dell'applicazione.

### 7.2.3 Installazione librerie

In questa sezione vengono scaricate e installate alcune librerie utili all'esecuzione dell'applicazione. Per l'installazione delle librerie viene usato il comando pip[17].

Pip é un gestore di pacchetti per Python, server per installare in modo semplice librerie esterne che non fanno parte del core Python.

---

```
1 !pip install requests
```

---

```
2 !pip install Pillow
3 !pip install tqdm
4 !pip install dlib
```

Listato 7.3: Installazione librerie

Questa istruzione viene eseguita solo una volta durante la prima esecuzione dell'applicazione.

#### 7.2.4 Installazione TensorFlow

In questa sezione viene installata e caricata nell'ambiente di lavoro la versione di TensorFlow 1.14.

```
1 %tensorflow_version 1.x
2 import tensorflow as tf
```

Listato 7.4: Installazione TensorFlow

Questa istruzione viene eseguita solo una volta durante la prima esecuzione dell'applicazione.

#### 7.2.5 Import Librerie e Setup ambiente

In questa sezione vengono caricate nell'ambiente di sviluppo le librerie Python utili all'esecuzione del progetto, in aggiunta vengono settate alcune variabili globali:

- `model_path`: contenente il path del dataset FFHQ
- `results_size`: dimensione dell'immagine di output

```
1 import os
2 import sys
3 sys.path.append('stylegan2/')
4 from stylegan2 import pretrained_networks
5 from stylegan2 import dnnlib
6 from stylegan2.dnnlib import tflib
7 from pathlib import Path
8 from PIL import Image
9 import pickle
10 import numpy as np
```

```
11
12 import ipywidgets as widgets
13 from tqdm import tqdm
14 model_path = 'gdrive:networks/stylegan2-ffhq-config-f.pkl'
15 results_size = 1024
```

Listato 7.5: Import Librerie e Setup ambiente

Questa istruzione viene eseguita solo durante la prima esecuzione dell'applicazione.

### 7.2.6 Preparazione dello StyleGAN2

In questa sezione vengono dichiarate alcune funzioni per l'instanziamento e l'uso vero e proprio dello StyleGAN2. Come visto in precedenza lo StyleGAN2 è basato sul concetto generale di GAN(Generative Adversarial Networks), il quale consiste essenzialmente di due modelli:

- Il **Generatore (Generator)**: Un modello che converte lo spazio latente in un output, nel nostro caso una immagine.
- Il **Discriminate (Discriminator)**: Un modello che dato un input determina se è vero o falso.
  - *Vero*: Se l'immagine proviene dal dataset
  - *False*: Se l'immagine è stata generata dal generatore

L'input del Generatore è un vettore latente  $z$  di 512 elementi.

- Durante il training il vettore è generato casualmente.
- Una volta che il vettore è stato generato possiamo definirlo come variabile casuale latente
- Questo vettore latente contiene le informazioni necessarie al Generatore per creare l'output

- Come visto nel capitolo precedente, il vettore latente di un determinato input è possibile rappresentarlo in forma compressa.

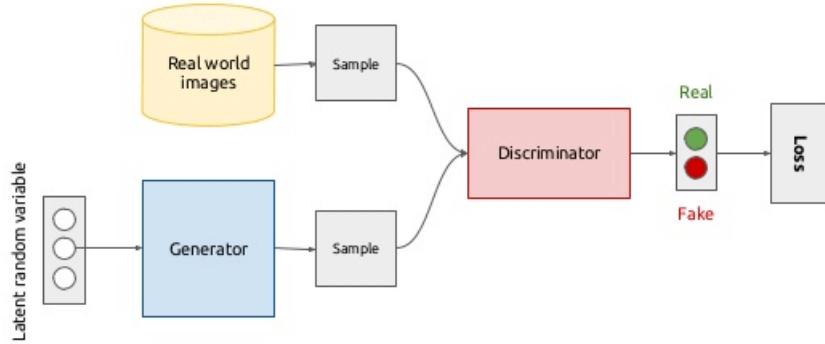


Figura 7.1: Architettura GAN

Di seguito viene mostrato il codice per la preparazione del modello StyleGAN2.

```

1 # Carica il modello StyleGAN2
2 def load_model():
3     _G, _D, Gs = pretrained_networks.load_networks(model_path)
4
5     noise_vars = [var for name, var in
6                   Gs.components.synthesis.vars.items() if
7                   name.startswith('noise')]
8
9     Gs_kw_args = dnnlib.EasyDict()
10    Gs_kw_args.output_transform =
11        dict(func=tflib.convert_images_to_uint8,
12             nchw_to_nhwc=True)
13    Gs_kw_args.randomize_noise = False
14
15    return Gs, noise_vars, Gs_kw_args
16
17 # Genera immagini partendo da un numero casuale (Integer)
18 def generate_image_random(rand_seed):
19     rnd = np.random.RandomState(rand_seed)
20     z = rnd.randn(1, *Gs.input_shape[1:])
21     tflib.set_vars({var: rnd.randn(*var.shape.as_list()) for
22                     var in noise_vars})
23     images = Gs.run(z, None, **Gs_kw_args)
24     return images, z
25
26 # Genera immagini partendo da un vettore latente (vector of
27 # size [1, 512] )
  
```

```
22 def generate_image_from_z(z):
23     images = Gs.run(z, None, **Gs_kwarg)
24     return images
25
26 def generate_image_from_projected_latents(latent_vector):
27     images = Gs.components.synthesis.run(latent_vector,
28                                         **Gs_kwarg)
29     return images
30
31 def get_concat_h(im1, im2):
32     dst = Image.new('RGB', (im1.width + im2.width, im1.height))
33     dst.paste(im1, (0, 0))
34     dst.paste(im2, (im1.width, 0))
35     return dst
```

Listato 7.6: Preparazione dello StyleGAN2

### 7.2.7 Caricamento del modello

In questa sotto sezione viene caricato il modello StyleGAN2 implementato nel passo precedente.

---

```
1 Gs, noise_vars, Gs_kwarg = load_model()
```

Listato 7.7: Caricamento del modello

### 7.2.8 Pulizia del Driver

In questa sezione vengono pulite le directory usate per memorizzare le immagini.

---

```
1 import shutil
2 file_path = 'imgs/test1.jpg'
3 orig_img_path = Path('imgs')
4 aligned_imgs_path = Path('aligned_imgs')
5 results_path = Path('results')
6
7 if os.path.exists(file_path):
8     os.remove(file_path)
9
10 if orig_img_path.exists():
11     #orig_img_path.rmdir()
12     shutil.rmtree(orig_img_path)
13     orig_img_path.mkdir()
14
15 if aligned_imgs_path.exists():
```

```
16     shutil.rmtree(aligned_imgs_path)
17     aligned_imgs_path.mkdir()
18
19 if results_path.exists():
20     shutil.rmtree(results_path)
21     results_path.mkdir()
```

Listato 7.8: Pulizia del Driver

### 7.2.9 Generazione dei volti

In questa sezione vengono generate immagini casuali usando StyleGAN2 e il dataset FFHQ.

Il primo blocco di codice genera un volto partendo da un numero generato casualmente e ritorna il vettore latente associato.

```
1 import random
2 images, latent_code1 =
3     generate_image_random(random.randrange(0, 3999999999))
4 result = Image.fromarray(images[0]).resize((results_size,
5     results_size))
6 latent_code1.shape
```

Listato 7.9: Generazione dei volti 1/2

```
1 (1, 512)
```

Listato 7.10: Output Generazione dei volti 1/2

```
1 array([-0.18813926,  0.57340194,  1.0504329 ,  0.5564614 ,
2  0.45630399,  0.89482661,  1.58452916, -0.13526257,
3  1.24084847, -0.06594399])
```

Listato 7.11: Porzione del contenuto del vettore latente

Il secondo blocco di codice salva l'immagine generata su disco e la mostra nel Notebook.

```
1 result.save(file_path)
2 result
```

Listato 7.12: Generazione dei volti 2/2



Figura 7.2: Immagine Generata con StyleGAN2

### 7.2.10 Immagini Custom

In questa sezione è possibile caricare immagini proprietarie per poi proiettarle nello spazio latente.

Prima di caricare le nostre immagini è necessario spiegare alcuni punti:

1. Proiettare un'immagine nello spazio latente significa trovare il codice latente (vettore  $z$  [512]) che il Generatore genera associato alla nostra immagine,
2. Il codice latente viene trovato dando in input la nostra immagine all'encoder VAEs[18],
3. La ricerca del codice latente è possibile riassumerlo nei seguenti passi:
  - Il generatore genera un output partendo da un vettore latente

- Viene data in input al VGG16 sia l'immagine generata che quella originale
- Si prende l'immagine generata all'uscita del VGG16
- Si prende l'immagine originale all'uscita del VGG16
- Si calcola la loss difference tra le due immagini
- Backpropagation

Prima di poter "giocare" con le nostre immagini nello spazio latente sono necessari alcuni step intermedi in modo tale da rendere queste immagini "usabili" dal modello StyleGAN2.

- Il primo passo è quello di importare l'immagine o le immagini nell'ambiente di lavoro. Prima di importare le immagini nell'ambiente di lavoro bisogna copiare un'immagine a scelta contentente il volto di una persona nella directory *imgs* del progetto, per questo scopo verrà usata una immagine dell'autore.
- Dopo aver importato l'immagine nell'ambiente di lavoro è necessario allinearla.
- Dopo averla importata e allineata è possibile generare il relativo vettore latente.

---

```

1 from align_face import align_face
2
3 # Align all of our images using a landmark detection model!
4 all_imgs = list(orig_img_path.iterdir())
5 for img in all_imgs:
6     align_face(str(img)).save(aligned_imgs_path/(aligned_+img.name))

```

---

Listato 7.13: Import immagini nell'ambiente di lavoro

---

```

1 aligned_img_set = list(aligned_imgs_path.iterdir())
2 aligned_img_set.sort()
3 aligned_img_set = [Image.open(x) for x in aligned_img_set]
4
5 orig_img_set = list(orig_img_path.iterdir())
6 orig_img_set.sort()
7 orig_img_set = [Image.open(x) for x in orig_img_set]
8
9 get_concat_h(orig_img_set[0], aligned_img_set[0])

```

---

Listato 7.14: Allineamento dell'immagine

Nell'immagine sottostante è possibile vedere, nella parte sinistra l'immagine originale, nella parte destra quella allineata.



Figura 7.3: Immagine originale vs immagine allineata

Creiamo un dataset StyleGAN2 compatibile con l'immagine caricata e allineata.

```
1 !python -W ignore stylegan2/dataset_tool.py create_from_images  
      datasets_stylegan2/custom_imgs aligned_imgs/
```

Listato 7.15: Creazione del dataset StyleGAN2

```
1 Loading images from "aligned_imgs/"  
2 Creating dataset "datasets_stylegan2/custom_imgs"  
3 Added 1 images.
```

Listato 7.16: Output Creazione del dataset StyleGAN2

Proiettiamo la nostra immagine nello spazio latente. Questo step usa le librerie TensorFlow.

```
1 tot_aligned_imgs = 1  
2 !python -W ignore stylegan2/epoching_custom_run_projector.py  
      project-real-images --network=$model_path \  
3   --dataset=custom_imgs --data-dir=datasets_stylegan2  
      --num-images=$tot_aligned_imgs --num-snapshots 500
```

Listato 7.17: Proiezione nello spazio latente

Dopo aver proiettato l'immagine nello spazio latente é possibile vedere il vettore latente associato che é stato creato.

```

1 def get_final_latents():
2     all_results = list(Path('results').iterdir())
3     all_results.sort()
4
5     last_result = all_results[-1]
6
7     latent_files = [x for x in last_result.iterdir() if
8                     'final_latent_code' in x.name]
9     latent_files.sort()
10
11    all_final_latents = []
12
13    for file in latent_files:
14        with open(file, mode='rb') as latent_pickle:
15            all_final_latents.append(pickle.load(latent_pickle))
16
17    return all_final_latents
18
19 latent_codes = get_final_latents()
20 len(latent_codes), latent_codes[0].shape

```

Listato 7.18: Vettore latente generato

---

```
1 (1, (1, 18, 512))
```

Listato 7.19: Output Vettore latente generato

La parte interessante del nostro output é (1, 18, 512), questo vettore latente generato risulta diverso da quello ottenuto generando l'immagine usando il dataset FFHQ visto nell'output 8.10 (1, 512). Ciò è dovuto a uno dei progetti di architettura di StyleGAN2, che in realtà reitera il vettore latente di base a diversi livelli nel generatore per consentire piccole deviazioni nel codice latente e supportare la varianza nello stile. Mentre durante l'apprendimento viene utilizzato un solo vettore latente statico nella forma (1,512).

Ricarichiamo il modello e compariamo l'immagine generata con quella originale.

---

```

1 def load_model():
2     _G, _D, Gs = pretrained_networks.load_networks(model_path)
3
4     noise_vars = [var for name, var in
5                   Gs.components.synthesis.vars.items() if
6                   name.startswith('noise')]

```

```
5      Gs_kwargs = dnnlib.EasyDict()
6      Gs_kwargs.output_transform =
7          dict(func=tflib.convert_images_to_uint8,
8              nchw_to_nhwc=True)
9      Gs_kwargs.randomize_noise = False
10
11     return Gs, noise_vars, Gs_kwargs
12
13 def generate_image_from_projected_latents(latent_vector):
14     images = Gs.components.synthesis.run(latent_vector,
15         **Gs_kwargs)
16     return images
17
18 Gs, noise_vars, Gs_kwargs = load_model()
19
20 images = generate_image_from_projected_latents(latent_codes[0])
21 recreated_img1 =
22     Image.fromarray(images[0]).resize((results_size,
23         results_size))
24 orig_img1 = aligned_img_set[0].resize((results_size,
25         results_size))
26 get_concat_h(orig_img1, recreated_img1)
```

Listato 7.20: Immagine originale vs generata

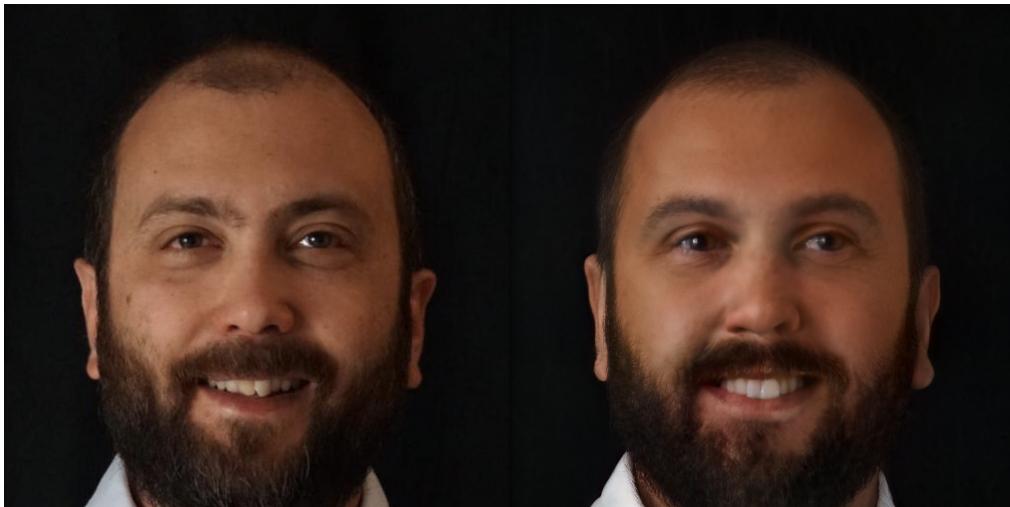


Figura 7.4: Immagine originale vs immagine Generata

L’immagine precedente mostra a sinistra l’immagine originale, mentre a destra quella generata con il nostro modello.

Come spiegato in precedenza le due immagini risultano molto vicine tra di loro ma allo stesso tempo mostrano delle piccole differenze, come si può notare dai dettagli della pelle, dei denti o della barba.

Nell'immagine generata la pelle è più omogenea e satinata perdendo alcuni dettagli come ad esempio i nei, i denti risultano più bianchi e omogenei cancellando la scheggiatura sull'incisivo, la barba più scura, riccia e meno definita.

### 7.2.11 Controlli latenti

In questa sezione vengono elencati i controlli latenti che è possibile usare.

Per controllare le diverse caratteristiche delle immagini, esistono diversi modi per addestrare le direzioni latenti (supervisionate o non supervisionate) nello spazio latente.

Gli sviluppatori hanno già reso opensource alcuni vettori latenti direzionali per StyleGAN2 che ci permettono di "muoverci" nello spazio latente e controllare una particolare caratteristica.

I miglioramenti di StyleGAN sulla separazione dello spazio latente consentono di esplorare i singoli attributi del dataset in modo piacevole e ortogonale (ovvero senza influenzare altri attributi). Mentre i modelli discriminatori imparano i confini per separare gli attributi target (es. Maschio/Femmina, sorriso/non sorriso ecc.), ciò che ci interessa qui è attraversare quei confini, muovendoci perpendicolarmente ad essi.

Ad esempio, se inizio da una faccia triste, posso passare lentamente ma costantemente ad una versione sorridente della stessa faccia. Questo dovrebbe già fornire un suggerimento su come apprendere nuove direzioni. Per prima cosa raccogliamo più campioni (immagine + latente) dal nostro modello e classifichiamo manualmente le immagini per il nostro attributo target (es. Sorridente vs non sorridente), cercando di garantire un corretto equilibrio di rappresentazione della classe. Quindi, addestriamo un modello per classificare o regredire sui nostri valori latenti. A questo punto possiamo usare le funzioni apprese da questi modelli di supporto come direzioni di transizione.

Per muoverci in una direzione latente possiamo fare la seguente operazione:

$$\text{latent\_code} = \text{latent\_code} + \text{latent\_direction} \times \text{magnitude} \quad (7.1)$$

- **latent\_code**: il nostro codice latente.
- **latent\_direction**: un vettore direzionale di forma (18, 512). Questo vettore ti dice dove muoverti nello spazio latente per controllare una caratteristica, ma non quanto muoverti.
- **magnitude**: è l'importo di cui spostarsi nella direzione latent\_direction.

Ciò significa che possiamo creare più interpolazioni nello spazio latente. Invece di mescolare due codici latenti insieme, aggiungiamo lentamente più magnitudine al nostro codice latente di base e osserviamo come cambia rispetto alla grandezza. Per il nostro esperimento useremo le direzioni latenti messe a disposizione da Robert Luxemburg[19].

```

1 def get_control_latent_vectors(path):
2     files = [x for x in Path(path).iterdir() if
5         str(x).endswith('.npy')]
3     latent_vectors = {f.name[:-4]: np.load(f) for f in files}
4     return latent_vectors
5
6 latent_controls =
7     get_control_latent_vectors('stylegan2directions/')
7 len(latent_controls), latent_controls.keys(),
    latent_controls['age'].shape

```

Listato 7.21: Direzioni latenti

```

1 (16,
2 dict_keys(['age', 'eye_distance', 'eye_eyebrow_distance',
3             'eye_ratio', 'eyes_open', 'gender', 'lip_ratio',
3             'mouth_open', 'mouth_ratio', 'nose_mouth_distance',
3             'nose_ratio', 'nose_tip', 'pitch', 'roll', 'smile', 'yaw']),
3 (18, 512))

```

Listato 7.22: Output Direzioni latenti

### 7.2.12 Generazione del Widget

In questa sezione viene implementato un widget che rende semplice l'applicazione dei controlli latenti alle immagini scelte.

Andremo ad applicare i controlli latenti sia all'immagine custom che a quelle generate dal modello.

Eventuali possibili miglioramenti verranno discussi nelle conclusioni della tesi. Questo Widget implementa una interfaccia grafica per facilitare applicazione di alcuni parametri latenti alle immagini. Per semplicità andremo ad applicare solo alcuni dei controlli latenti messi a disposizione:

- age
- smile
- gender
- yaw
- lip\_ratio
- eyes\_open
- roll
- nose\_ratio
- nose\_mouth\_distance
- pitch
- eye\_distance
- nose\_mouth\_distance
- mouth\_open

---

```
1 def apply_latent_controls(self):  
2     image_outputs = controller.children[0]  
3     feature_sliders = controller.children[1]  
4  
5     slider_hboxes = feature_sliders.children  
6     #latent_movements = [(x.children[1].value,  
7     x.children[0].value) for x in slider_hboxes]
```

```

8
9 latent_movements = []
10 for x in slider_hboxes:
11     if x.description == 'lip_ratio':
12         if x.value == 'slim':
13             latent_movements.append(tuple((x.description, -15.0)))
14         elif x.value == 'big':
15             latent_movements.append(tuple((x.description, 15.0)))
16     else:
17         latent_movements.append(tuple((x.description, 0.0)))
18     if x.description == 'yaw':
19         if x.value == 'left':
20             latent_movements.append(tuple((x.description, 15.0)))
21         elif x.value == 'right':
22             latent_movements.append(tuple((x.description, -15.0)))
23     else:
24         latent_movements.append(tuple((x.description, 0.0)))
25     if x.description == 'gender':
26         if x.value == 'male':
27             latent_movements.append(tuple((x.description, 5.0)))
28         elif x.value == 'female':
29             latent_movements.append(tuple((x.description, -8.0)))
30     else:
31         latent_movements.append(tuple((x.description, 0.0)))
32     if x.description == 'smile':
33         if x.value == 'yes':
34             latent_movements.append(tuple((x.description, 5.0)))
35         elif x.value == 'no':
36             latent_movements.append(tuple((x.description, -5.0)))
37     else:
38         latent_movements.append(tuple((x.description, 0.0)))
39     if x.description == 'age':
40         if x.value == 'young':
41             latent_movements.append(tuple((x.description, -5.0)))
42         elif x.value == 'mid_age':
43             latent_movements.append(tuple((x.description, 1.20)))
44         elif x.value == 'old':
45             latent_movements.append(tuple((x.description, 8.0)))
46     else:
47         latent_movements.append(tuple((x.description, 0.0)))
48     if x.description == 'eyes_open':
49         if x.value == 'small':
50             latent_movements.append(tuple((x.description, -15.0)))
51         elif x.value == 'big':
52             latent_movements.append(tuple((x.description, 15.0)))
53     else:
54         latent_movements.append(tuple((x.description, 0.0)))
55     if x.description == 'roll':
56         if x.value == 'black':
57             latent_movements.append(tuple((x.description, -15.0)))
58         elif x.value == 'white':

```

```

59         latent_movements.append(tuple((x.description, 15.0)))
60     else:
61         latent_movements.append(tuple((x.description, 0.0)))
62     if x.description == 'nose_ratio':
63         if x.value == 'big':
64             latent_movements.append(tuple((x.description, -15.0)))
65         elif x.value == 'small':
66             latent_movements.append(tuple((x.description, 15.0)))
67         else:
68             latent_movements.append(tuple((x.description, 0.0)))
69     if x.description == 'nose_mouth_distance':
70         if x.value == 'big':
71             latent_movements.append(tuple((x.description, -15.0)))
72         elif x.value == 'small':
73             latent_movements.append(tuple((x.description, 15.0)))
74         else:
75             latent_movements.append(tuple((x.description, 0.0)))
76     if x.description == 'pitch':
77         if x.value == 'back':
78             latent_movements.append(tuple((x.description, 15.0)))
79         elif x.value == 'front':
80             latent_movements.append(tuple((x.description, -15.0)))
81         else:
82             latent_movements.append(tuple((x.description, 0.0)))
83     if x.description == 'eye_distance':
84         if x.value == 'big':
85             latent_movements.append(tuple((x.description, -15.0)))
86         elif x.value == 'small':
87             latent_movements.append(tuple((x.description, 15.0)))
88         else:
89             latent_movements.append(tuple((x.description, 0.0)))
90     if x.description == 'eye_eyebrow_distance':
91         if x.value == 'big':
92             latent_movements.append(tuple((x.description, 15.0)))
93         elif x.value == 'small':
94             latent_movements.append(tuple((x.description, -15.0)))
95         else:
96             latent_movements.append(tuple((x.description, 0.0)))
97     if x.description == 'mouth_open':
98         if x.value == 'open':
99             latent_movements.append(tuple((x.description, 15.0)))
100        elif x.value == 'closed':
101            latent_movements.append(tuple((x.description, -15.0)))
102        else:
103            latent_movements.append(tuple((x.description, 0.0)))
104 modified_latent_code = np.array(latent_code_to_use)
105 for feature, amount_to_move in latent_movements:
106     modified_latent_code +=
107         latent_controls[feature]*amount_to_move

```

```
108     images =
109         generate_image_from_projected_latents(modified_latent_code)
110     latent_img = Image.fromarray(images[0]).resize((512, 512))
111
112     latent_img_output = image_outputs.children[1]
113     with latent_img_output:
114         latent_img_output.clear_output()
115         display(latent_img)
116
116 def create_interactive_latent_controller():
117     orig_img_output = widgets.Output()
118
119     with orig_img_output:
120         orig_img_output.clear_output()
121         display(result)
122
123     latent_img_output = widgets.Output()
124
125     with latent_img_output:
126         latent_img_output.clear_output()
127         display(result)
128
129     image_outputs = widgets.VBox([orig_img_output,
130                                 latent_img_output])
130
131 #collapse-hide
132 generate_button = widgets.Button(description='Generate',
133                                 layout=widgets.Layout(width='75%', height='10%'))
133 generate_button.on_click(apply_latent_controls)
134
135 reset_button = widgets.Button(description='Reset_Latent_
136     Controls', layout=widgets.Layout(width='75%',
137                                         height='10%'))
136 #reset_button.on_click(reset_latent_controls)
137
138 feature_sliders = []
139 age_buttons = widgets.ToggleButtons(
140     options=['default', 'young', 'mid_age', 'old'],
141     description='age',)
142 smile_buttons = widgets.ToggleButtons(
143     options=['default', 'no', 'yes'],
144     description='smile',)
145 gender_buttons = widgets.ToggleButtons(
146     options=['default', 'male', 'female'],
147     description='gender',)
148 yaw_buttons = widgets.ToggleButtons(
149     options=['default', 'right', 'left'],
150     description='yaw',)
151 lip_ratio_buttons = widgets.ToggleButtons(
152     options=['default', 'slim', 'big'],
153     description='lip_ratio',)
```

```

154     eyes_open_buttons = widgets.ToggleButtons(
155         options=['default', 'small', 'big'],
156         description='eyes_open',)
157     roll_buttons = widgets.ToggleButtons(
158         options=['default', 'white', 'black'],
159         description='roll',)
160     nose_ratio_buttons = widgets.ToggleButtons(
161         options=['default', 'big', 'small'],
162         description='nose_ratio',)
163     nose_mouth_buttons = widgets.ToggleButtons(
164         options=['default', 'big', 'small'],
165         description='nose_mouth_distance',)
166     pitch_buttons = widgets.ToggleButtons(
167         options=['default', 'back', 'front'],
168         description='pitch',)
169     eye_distance_buttons = widgets.ToggleButtons(
170         options=['default', 'big', 'small'],
171         description='eye_distance',)
172     eye_eyebrow_buttons = widgets.ToggleButtons(
173         options=['default', 'big', 'small'],
174         description='eye_eyebrow_distance',)
175     mouth_open_buttons = widgets.ToggleButtons(
176         options=['default', 'open', 'closed'],
177         description='mouth_open',)
178     #feature_sliders.appe'd(widgets.HBox([label, age_buttons]))
179     feature_sliders.append(age_buttons)
180     feature_sliders.append(smile_buttons)
181     feature_sliders.append(gender_buttons)
182     feature_sliders.append(yaw_buttons)
183     feature_sliders.append(lip_ratio_buttons)
184     feature_sliders.append(eyes_open_buttons)
185     feature_sliders.append(roll_buttons)
186     feature_sliders.append(nose_ratio_buttons)
187     feature_sliders.append(nose_mouth_buttons)
188     feature_sliders.append(pitch_buttons)
189     feature_sliders.append(eye_distance_buttons)
190     feature_sliders.append(eye_eyebrow_buttons)
191     feature_sliders.append(mouth_open_buttons)
192     feature_sliders.append(generate_button)
193     feature_sliders = widgets.VBox(feature_sliders)
194
195     return widgets.HBox([image_outputs, feature_sliders])
196
197 latent_code_to_use = latent_codes[0]
198 image_to_use = result
199
200 controller = create_interactive_latent_controller()
201 controller

```

Listato 7.23: Creazione del Widget

### Caso Sperimentale: Applicazione dello spazio latente a StyleGan2

L'uso del widget è abbastanza intuitivo, con i bottoni si scelgono i controlli da applicare e poi si confermano facendo click sul il tasto "Generate".

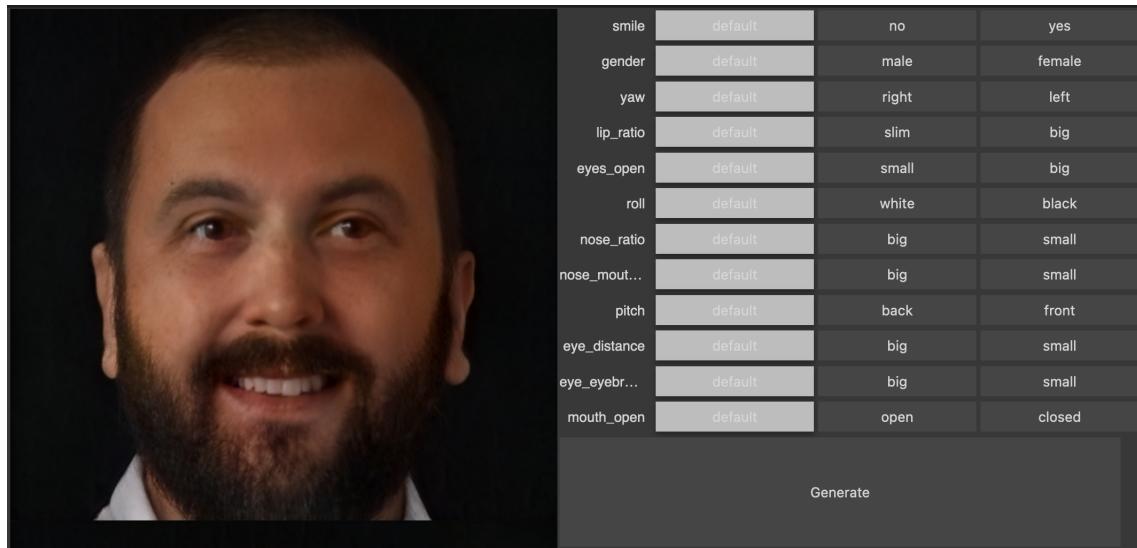


Figura 7.5: Widget

L'immagine sottostante mostra la foto del volto invecchiato.

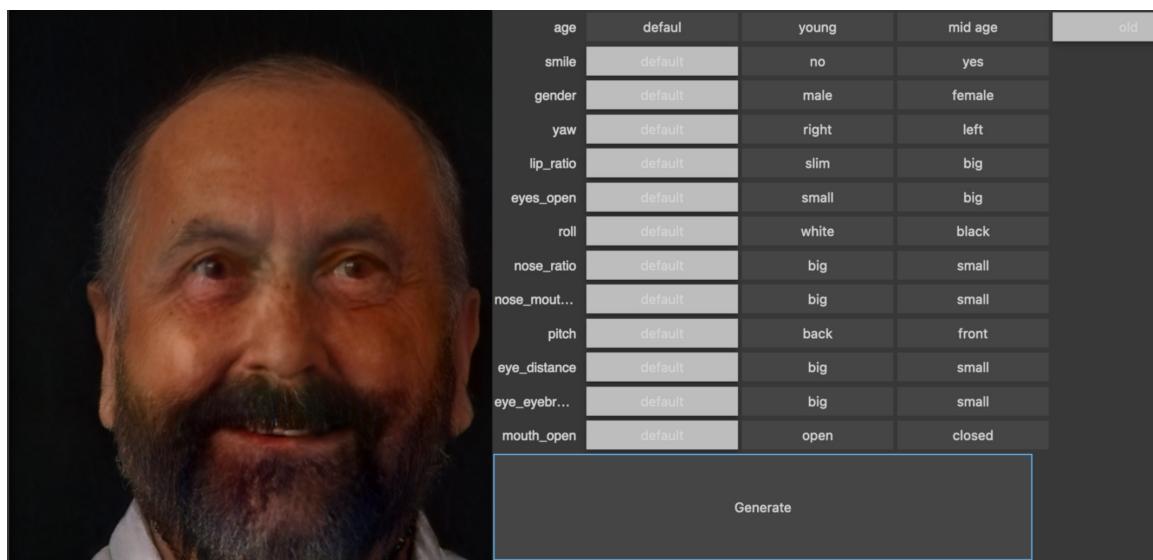


Figura 7.6: Widget - invecchiato

Caso Sperimentale: Applicazione dello spazio latente a StyleGAN2

L'immagine sottostante mostra la foto del volto al femminile.



Figura 7.7: Widget - femminile

L'immagine sottostante mostra la foto del volto senza il sorriso.



Figura 7.8: Widget - senza sorriso

L'immagine sottostante mostra la foto del volto con la testa piegata all'indietro.



Figura 7.9: Widget - senza sorriso

L'immagine sottostante mostra la foto del volto invecchiato, con sembianze femminili e la bocca aperta.



Figura 7.10: Widget - al femminile, invecchiato, bocca aperta

L’immagine sottostante mostra il dettaglio della foto mostrata nell’immagine precedente



Figura 7.11: Widget - dettaglio

L’immagine mostra delle imperfezioni ai contorni delle labbra e i denti inferiori. Un osservatore attento può notare che i bordi al lato sinistro delle labbra sono neri, mentre i denti nella parte inferiore destra della bocca sono assenti. Questo può essere dovuto alla presenza della barba, alla qualità del dataset usato per generare i controlli latenti, oppure alla qualità della foto usata come input. Queste osservazioni possono essere prese come spunto per imparare la strada ad ulteriori casi di studio come ad esempio: l’uso di dataset addestrati su immagini contenenti un numero maggiore di immagini di volti con barba, oppure cercare di migliorare l’algoritmo quando si usano più vettori latenti contemporaneamente.

L’immagine sottostante mostra un’immagine generata con il modello al quale sono stati applicati i controlli latenti di invecchiamento e cambio di genere.

Caso Sperimentale: Applicazione dello spazio latente a StyleGan2

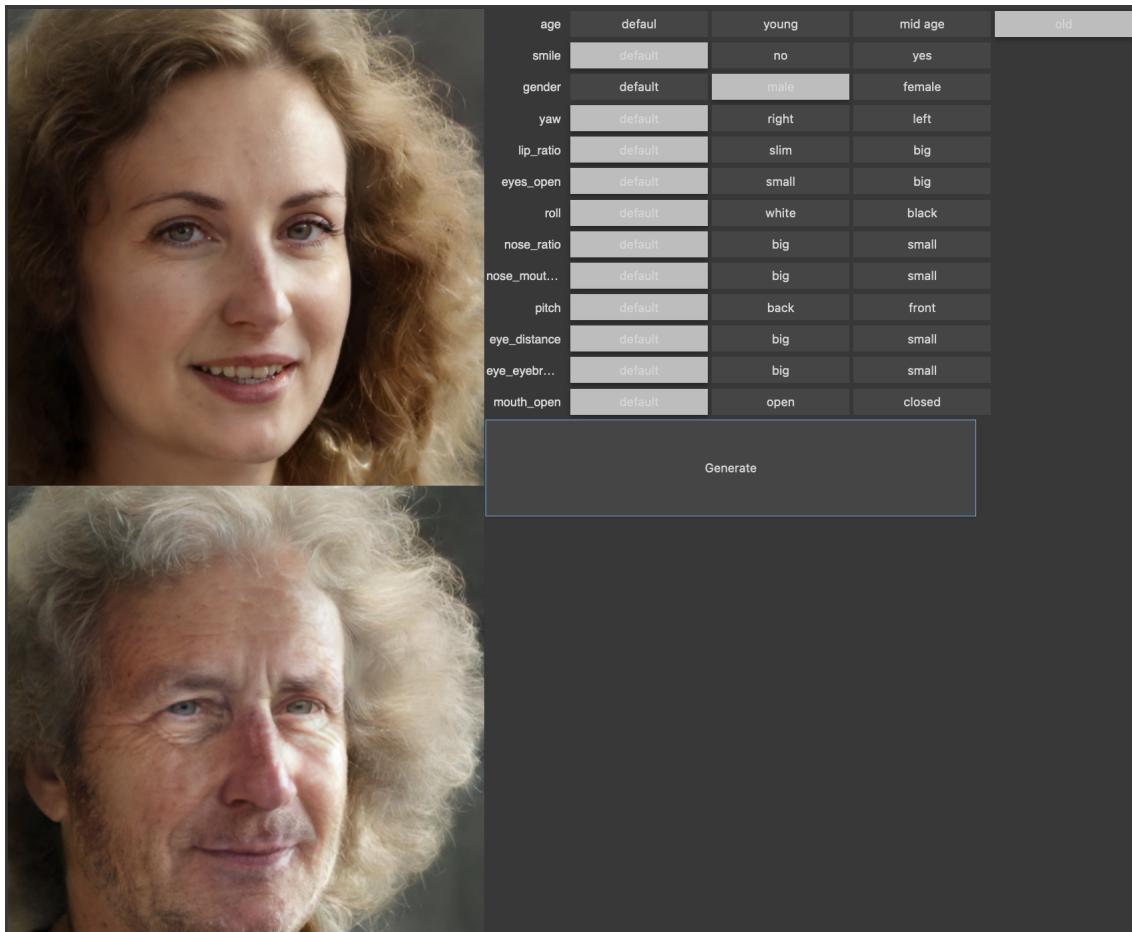


Figura 7.12: Widget - con immagine generata dal modello

## 8 Conclusioni

Con questa tesi si è voluto affrontare il tema delle Generative Adversarial Network (GAN), partendo dai concetti di Intelligenza Artificiale, Machine Learning e Reti Neurali ripercorrere la loro evoluzione per poi concentrare l'attenzione sempre di più sul GAN, e nello specifico su una delle sue ultime impletazioni lo StyleGAN2 e il concetto di spazio latente visto nel Capitolo 6.

Nel capitolo 7 si è voluto dare una dimostrazione pratica dei concetti visti implementando una semplice applicazione in grado di mostrare le potenzialità del modello e allo stesso tempo aprire alcuni spunti di riflessione su possibili sviluppi futuri.

Il progetto prende spunto dal sito web <https://www.thispersondoesnotexist.com/>, dove ad ogni refresh della pagina appare un nuovo volto generato a runtime. L'algoritmo dietro questo sito web deriva da un recente documento (pubblicato il 6 febbraio 2019) e basato sull'idea di StyleGAN [21] affrontato in varie parti durante lo sviluppo della tesi.

L'applicazione sviluppata ha voluto mettere in evidenza come, grazie al lavoro di società come NVIDIA e Google (Colab), oggigiorno sia possibile reperire in rete materiale opensource, di alta qualità, che con un approccio semplice e ad alto livello permettono di creare modelli complessi di deep learning, per la generazione di immagini ad alta risoluzione al pari di quelle reali.

La larga diffusione di questi strumenti accessibili liberamente, può essere usata come indicatore per capire il fermento intorno queste discipline. Basti pensare solo all'accesso gratuito che Google tramite Colab offre alle GPU e TPU.

Le GPU (Graphics Processing Unit)[20] sono dei componenti hardware molto costosi che senza un accesso gratuito e semplice come quello reso possibile tramite Colab, lo sviluppo di questa tesi sarebbe risultato dispendioso in termini di tempo e costi materiali.

## *Conclusioni*

---

Per motivi di tempo e risorse hardware disponibili la tesi é stata sviluppata usando il dataset FFHQ messo a disposizione dagli sviluppatori. Una possibile evoluzione dell'attuale progetto potrebbe essere l'addestramento della rete usando un caso d'uso particolare come ad esempio: la creazione di una miss di bellezza usando le immagini di ragazze che hanno partecipato ai concorsi di bellezza italiani o europei negli gli ultimi anni, oppure la creazione di una sorta di album dei calciatori usando i volti dei giocatori dei principali tornei di calcio mondiali. Oppure la creazione di direzioni latenti in modo da migliorare la qualità dell'immagine nel caso in cui si mischiano piú direzioni latenti nella creazione/modifica di un volto.

La creazione di volti é solo una delle possibili applicazioni delle reti GAN. Queste reti trovano ampio spazio sia per usi leciti: grafica, musica, creazione di testi, deepfake detection, medicina e robotica; ma anche in ambiti meno leciti come ad esempio l'applicazione del deepfake in ambito comunicazione o revenge porn.

A me piace pensare le reti GAN come degli strumenti che si integreranno sempre di piú nella nostra vita semplificandola e migliorandola. Ad esempio usando le reti GAN sarebbe possibile avere elettrodomestici che conoscendo il loro contenuto siano in grado di suggerire ricette totalmente nuove per sopperire alla momentanea pigrizia, mancanza di idee o voglia di sperimentazioni culinaree.

# Bibliografia

- [1] URL: [https://it.wikipedia.org/wiki/Gioco\\_a\\_somma\\_zero](https://it.wikipedia.org/wiki/Gioco_a_somma_zero).
- [2] URL: <https://arxiv.org/abs/1406.2661>.
- [3] URL: <https://arxiv.org/abs/1812.04948>.
- [4] URL: <https://arxiv.org/abs/1710.10196>.
- [5] URL: <https://arxiv.org/abs/1703.06868>.
- [6] URL: <https://www.youtube.com/watch?v=9QuDh3W3l0Y>.
- [7] URL: <https://www.youtube.com/watch?v=kSLJriaOumA>.
- [8] URL: <https://arxiv.org/pdf/1912.04958.pdf>.
- [9] URL: <https://github.com/NVlabs/stylegan2>.
- [10] URL: <https://github.com/NVlabs/stylegan>.
- [11] URL: <https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df>.
- [12] URL: <https://www.flickr.com/photos/tags/flicker/>.
- [13] URL: <https://colab.research.google.com/>.
- [14] URL: <https://jupyter.org/>.
- [15] URL: <https://www.tensorflow.org/>.
- [16] URL: <https://www.python.org/>.
- [17] URL: <https://pypi.org/project/pip/>.
- [18] URL: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>.
- [19] URL: <https://twitter.com/robertluxemburg/status/1207087801344372736>.
- [20] URL: [https://en.wikipedia.org/wiki/Graphics\\_processing\\_unit](https://en.wikipedia.org/wiki/Graphics_processing_unit).

## BIBLIOGRAFIA

---

- [21] Tero Karras Samuli Laine Timo Aila. «A Style-Based Generator Architecture for Generative Adversarial Networks». In: (2019).
- [22] Martin Arjovsky Leon Bottou. «Towards Principled Methods for Training Generative Adversarial Networks». In: (2018).
- [23] yunghyun Cho Bart van Merriënboer Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Yoshua Bengio. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». In: (2014).
- [24] Ian J. Goodfellow. «Generative Adversarial Nets». In: (2014).
- [25] Bhiksha Raj Haohan Wang. «On the Origin of Deep Learning». In: (2017).
- [26] Martin Heusel Hubert Ramsauer Thomas Unterthiner Bernard Nessler Sepp Hochreiter. «GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium». In: (2017).
- [27] Thomas Unterthiner Bernhard Nessler Calvin Seward Gunter Klambauer Martin Heusel Hubert Ramsauer Sepp Hochreiter. «Coulomb GANs: Provably Optimal Nash Equilibria via Potential Fields». In: (2018).
- [28] Jonathan Hui. «GAN - Why it is so hard to train Generative Adversarial Networks!». In: (2018).
- [29] Jay Kuo. «Understanding Convolutional Neural Networks with A Mathematical Model». In: (2016).
- [30] Pierre Lison. «An Introduction to Machine Learning». In: *Language Technology Group (LTG) Department of Informatics HiOA* (2012).
- [31] Seymour A. Papert Marvin L. Minsky. «Perceptrons: An Introduction to Computational Geometry». In: *MIT Press, Cambridge* (1969).
- [32] Cade Metz. «How A.I. Is Creating Building Blocks to Reshape Music and Art». In: *The New York Times* (2017).
- [33] NVidia. «Convolution». In: (). URL: <https://developer.nvidia.com/discover/convolution>.
- [34] Frans A. Oliehoek Rahul Savani Jose Gallego Elise van der Pol Roderich Gross. «Beyond Local Nash Equilibria for Adversarial Networks». In: (2018).

## BIBLIOGRAFIA

---

- [35] F. Rosenblatt. «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain». In: *Cornell Aeronautical Laboratory Psychological Review Vol. 65* (1968).
- [36] Sepp Hochreiter Jurgen Schmidhuber. «Long short-term memory». In: *Neural Computation Vol. 9* (1997).
- [37] Luke Metz Ben Poole David Pfau Jascha Sohl-Dickstein. «Unrolled Generative Adversarial Networks». In: (2018).
- [38] Lode Vandevenne. «Image Filtering». In: () .
- [39] James Vincent. «Christie sells its first AI portrait for 432500 dollar beating estimates of 10000 dollar». In: *The Verge* (2018).
- [40] James Vincent. «How Three French Students Used Borrowed Code To Put The First AI Portrait In Christie». In: *The Verge* (2018).
- [41] Walter Pitts Warren S. McCulloch. «A logical calculus of the ideas immanent in nervous activity». In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.
- [42] Zbigniew Wojna. «Rethinking the Inception Architecture for Computer Vision». In: (2015). DOI: UniversityCollegeLondon. URL: <https://arxiv.org/pdf/1512.00567v3.pdf>.
- [43] Bengio Yoshua. «Deep Learning of Representations for Unsupervised and Transfer Learning». In: *Workshop on Unsupervised and Transfer Learning* (2012).
- [44] Bengio Yoshua. «Learning Deep Architectures for AI». In: *Foundations and Trends in Machine Learning 2* (2009).