Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

PERFORMANCE DEI VARIATIONAL AUTOENCODERS IN RELAZIONE AL TRAINING SET

Relatore:
Chiar.mo Prof.
ANDREA ASPERTI

Presentata da:
DANIELE RAVAGLIA

II Sessione
Anno Accademico 2018/2019

Introduzione

Lo scopo delle tecniche di apprendimento automatico, o machine learning, è quello di estrarre conoscenza dai dati, similmente a come fa il nostro cervello, in modo da poter fare previsioni, classificare altri dati o generarne di nuovi. Proprio quest'ultimo aspetto è di grande interesse.

> *"What I cannot create, I do not understand."*
>
> –Richard Feynman

Non si può essere in grado di generare un certo tipo di dato senza prima averne compreso a fondo la natura e la struttura. Questa frase di Feynman racchiude quindi il motivo per il quale i così detti modelli generativi sono oggi di grande interesse in diversi campi, quali la computer vision ([15]) e la chimica ([10]).
L'approccio dei Variational Autoencoders ([9], [13], [7]) è di grande interesse: essi mirano ad imparare una distribuzione $P'(X)$ che approssimi la distribuzione $P(X)$ di un certo dataset, dataset tipicamente composto da immagini. In questo modo, per generare nuove immagini, basterà fare un sampling da $P'(X)$.
Saper generare immagini non basta, si rende infatti necessaria anche una metrica per valutarne la qualità, a questo proposito viene quindi usata la Frechèt Inception Distance ([8]).
In questa tesi viene approfondito il funzionamento dei VAE e le diverse strategie che si possono adottare per migliorarne le performance: come adottare un'architettura a due livelli ([4]) o stimare ex-post la distribuzione dello spazio latente ([7]).
Viene in seguito studiata la FID in modo da verificare il suo comportamento a seconda del tipo di dataset usato.
Verrà quindi testata con datasets composti da immagini ripetute progressivamente e datasets contenenti immagini scomposte in varie sezioni e ricomposte casualmente. L'idea, è che si possano ottenere FID score simili allenando un VAE su un certo dataset e allenando lo stesso VAE sul medesimo dataset ma ridotto.

Unitamente all'idea sopracitata si prova a selezionare il dataset secondo due criteri: l'errore di ricostruzione e la distanza di Mahalanobis. Tentando quindi di ridurre la dimensione del dataset e aumentarne la qualità, in modo tale da allenare il VAE a generare immagini alle quali la FID sia più sensibile.
Tutti i modelli vengono testati usando anche un secondo livello e alternativamente facendo una stima ex-post con una Gaussian Mixture Model.
In conclusione verranno presentati i risultati e paragonati gli score di tutti i modelli.

Struttura della tesi

- Nella prima parte vengono introdotti i concetti basilari dell'apprendimento automatico, utili alla comprensione dell'elaborato.
- Nella seconda parte vengono introdotti i modelli generativi, modelli a variabile latente e i Variational Autoencoders. Questa parte si concentra su quest'ultimi, illustrandone la funzione di costo, problematiche e diverse strategie per tentare di sopperire ad esse.
- Infine nella terza parte viene illustrata la Frechèt Inception Distance e la distanza di Mahalanobis, mostrando poi i risultati degli esperimenti eseguiti.

Indice

Elenco delle figure

Elenco delle tabelle

Parte I

1 Reti neurali e Deep Learning

Una rete neurale è un modello computazionale che ricalca vagamente una rete neurale biologica.
La rete è un insieme di nodi, dove ogni nodo della rete simula il comportamento di un neurone. Con questo modello si riescono ad implementare tecniche per l'apprendimento automatico.

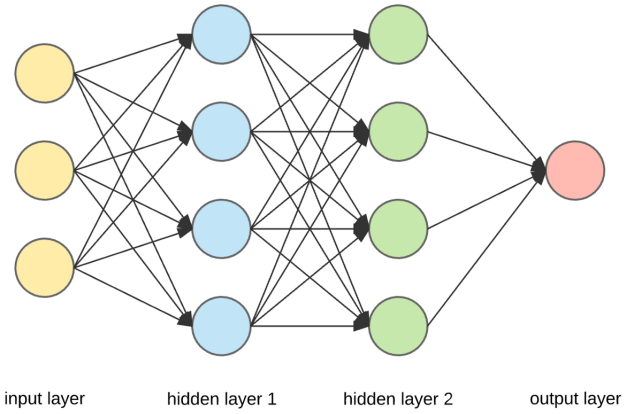

Figura 1: Struttura di una rete neurale.
Fonte: towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Comunemente le reti sono divise in layers di nodi successivi: ogni nodo è connesso con gli altri, più layers (e quindi nodi) la rete ha e migliori potranno essere le sue performance.

Il Deep Learning è una branca del Machine Learning: generalmente gli algoritmi di Deep Learning si basano su reti neurali deep, profonde, con molteplici layers grazie ai quali riescono ad estrarre progressivamente features sempre più avanzate da un dato grezzo dato in input.
Tipicamente esistono tre diverse aree del machine learning. L' *Unsupervised learning*, dove non abbiamo conoscenza dei nostri dati e usiamo una rete neurale per estrarre features significative da essi. Il *Reinforcement learning* che ha come obiettivo quello di sviluppare un sistema che mi-

gliori le proprie prestazioni attraverso interazioni con l'ambiente. Infine il *Supervised learning* dove lo scopo è quello di ricavare un modello, attraverso l'addestramento con dati etichettati, che possa effettuare previsioni attendibili su dati futuri ([12]).

1.1 Algoritmo di retropropagazione dell'errore

L'algoritmo di retropropagazione dell'errore, in inglese backpropagation, viene utilizzato nelle reti neurali feedforward, ovvero le reti dove le connessioni fra i nodi non formano dei cicli, per l'apprendimento supervisionato. Come detto in precedenza le reti neurali simulano una vera e propria rete neurale biologica, dove ogni nodo corrisponde ad un neurone.
I neuroni biologici ricevono segnali in input e al raggiungimento di una certa soglia mandano un segnale di output, lo stesso principio è usato dai neuroni artificiali.

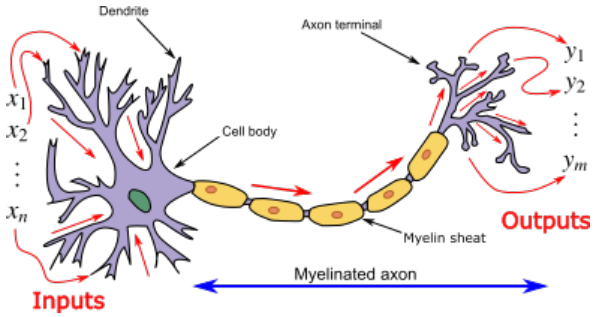

Figura 2: Schema di un neurone biologico.
Fonte: upload.wikimedia.org/wikipedia/commons/4/44/Neuron3.png

Siano $x_1, ..., x_n$ i segnali che il neurone artificiale riceve in input, $w_1, ...; w_n$ sono i suoi pesi, z combinazione lineare di $w_1x_1 + ... + w_nx_n$ e $\phi(.)$ la funzione di attivazione con una soglia θ.
La funzione ϕ è definita come

$$\phi(z) = \begin{cases} y', & \text{se } z \geq \theta \\ y'', & \text{altrimenti} \end{cases}$$

L'output del neurone sarà quindi:

$$y = \phi(z)$$

Esistono diverse funzioni di attivazione, le due più comuni sono la funzione *sigmoid*, $\frac{1}{1+e^{-x}}$ e la *ReLu*, $max(0, x)$.
Calcolato l'output, tramite una funzione di costo si calcola l'errore (confrontando l'output con il risultato atteso) e attraverso il calcolo del gradiente si aggiustano i pesi w_i dei neuroni in modo da minimizzare la funzione di costo. Da qui il nome retropropagazione dell'errore.

Parte II

2 Modelli generativi

I modelli generativi sono uno strumento impiegato per diversi problemi in statistica e nel campo dell'apprendimento automatico.
Matematicamente un modello generativo, a partire da un insieme di dati in input $x_1, ..., x_n$, con una loro distribuzione $P(x)$, cerca di riprodurre una distribuzione $P'(x)$ che sia il più possibile simile $P(x)$.

2.1 Modello a variabile latente

La difficoltà dei modelli generativi risiede nel fatto che spesso le distribuzioni che si cerca di riprodurre sono molto complicate. Da qui l'idea di aggirare il problema cercando di approssimare queste distribuzioni.
In un modello con variabili latenti si assume che esista un insieme di variabili z, non osservabili, grazie alle quali è possibile generare un insieme di dati x.
Quindi $p(x|\theta, z)$, dove θ rappresenta i parametri del modello, descrive il processo di generazione.
In questo modo non dobbiamo modellare direttamente la distribuzione $p(x)$ ma possiamo esprimerla come $p(x, z) = p(x|z)p(z)$, scegliendo a priori una distribuzione $p(z)$ che sia trattabile, comunemente una Gaussiana. Questo ci permette di formulare $p(x)$ come:

$$p(x) = \int p(x, z)dz = \int p(x|z)p(z)dz \tag{1}$$

per poi applicare il teorema di Bayes e ricavare la probabilità a posteriori:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

da notare però che la (1) spesso è intrattabile. Sarà quindi necessario ricavarla tramite altri metodi come ad esempio l'inferenza variazionale [9].

2.2 Variational Autoencoders

I Variational Autoencoders, abbreviati in VAE, sono un modello di reti neurali che permettono di implementare un modello generativo.
L'architettura di un VAE si basa sostanzialmente su quella di un Autoencoder[1].
Un Autoencoder è formato da due reti neurali: l'encoder, che prende in input un dato e ne crea una sua rappresentazione interna (codificata) di dimensione minore e il decoder che prende in input una rappresentazione codificata dal decoder e la decodifica, restituendo in output il dato passato in input all'encoder.

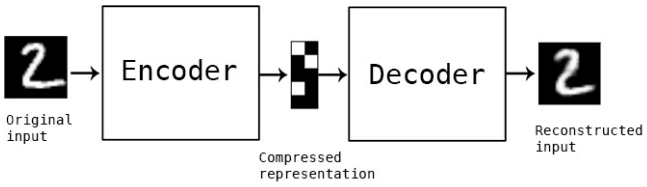

Figura 3: Struttura di un Autoencoder.
Fonte: blog.keras.io/building-autoencoders-in-keras.html

L'idea è quella di sfruttare il decoder per generare nuovi dati, il problema però è che la struttura dello spazio (latente) dal quale fare il sampling dei punti da dare in input al decoder è sconosciuto (sparso).
La soluzione è quindi quella di forzare l'encoder a generare uno spazio latente che sia continuo. ([6])
La struttura generale del VAE rimane invariata ma al contrario degli Autoencoders l'encoder genererà due vettori: un vettore di medie μ e un vettore delle varianze Σ. Si genera poi un vettore z utilizzando la distribuzione $\mathcal{N}(\mu_i, \Sigma_i)$ che saranno i punti dello spazio latente. Grazie a questo sampling il processo diventa stocastico e quindi anche la codifica dello stesso dato preso in input più volte non sarà per forza identica, ma varierà attorno alla media secondo la sua varianza.
Ancora però non si è sicuri di avere uno spazio latente continuo, infatti

[1] https://blog.keras.io/building-autoencoders-in-keras.html

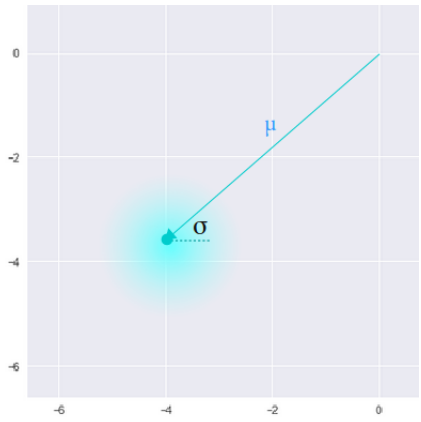

Figura 4: In azzurro la regione nella quale può essere codificata una certa caratteristica.
Fonte: towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

le varie regioni potrebbero essere sparse. Per evitare ciò alla funzione di costo si aggiunge la divergenza di Kullback-Leibler (KL), essa è una misura (non simmetrica) che quantifica la differenza fra due distribuzioni di probabilità.
La KL è definita come:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) log \left(\frac{P(x)}{Q(x)} \right)$$

cercare di minimizzarla significa quindi cercare di ottenere una distribuzione simile ad un'altra presa come riferimento, tipicamente una Gaussiana. In questo modo si riesce a forzare l'encoder a creare uno spazio latente il più possibile continuo.

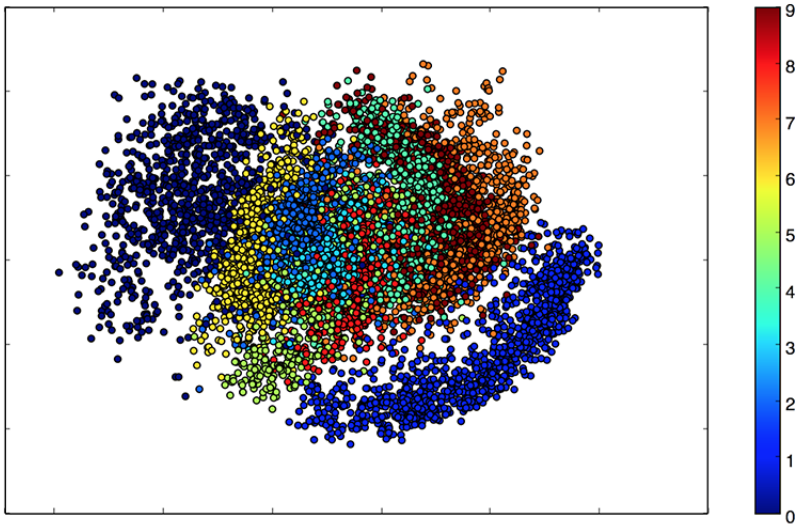

Figura 5: Configurazione di uno spazio latente di un VAE allenato su MNIST aggiungendo la KL alla funzione di costo.
Fonte: https://blog.keras.io/building-autoencoders-in-keras.html

La funzione di minimizzazione o loss function (decomposta in singoli termini che dipendono dall'i-esimo punto) sarà quindi:

$$l_i(\phi, \theta) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + \mathbb{KL}(q_\theta(z|x_i)||p(z))$$

Dove ϕ e θ sono i parametri della rete.
La funzione di loss totale sarà data dalla somma delle loss degli i-esimi punti.
Assumendo che $p_\theta(x_i|z)$ sia una Gaussiana allora il suo logaritmo è la distanza quadratica fra x_i e la sua ricostruzione, quindi il primo termine può essere sostituito con l'errore quadratico medio, mean squared error (mse) che fornisce la differenza (media quadratica) tra i valori stimati e i valori osservati.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i')^2$$

Dove n è il numero di stime fatte, Y_i è la i-esima osservazione e Y_i' è la i-esima stima.

Valori piccoli dell'mse indicano una buona stima.
Il secondo termine è invece la distanza di Kullback-Leibler.
Nonostante la KL però, lo spazio latente potrebbe non avere la distribuzione attesa.
La distribuzione dello spazio latente può essere appresa mediante l'uso di un secondo livello VAE o stimata attraverso un fitting di Gaussiane.

2.2.1 Two Stage VAE

Come detto poc'anzi la KL non garantisce che lo spazio latente abbia la distribuzione voluta. Di conseguenza non si sarà in grado di generare samples significativi, nemmeno se dopo il training si fosse in grado di ricostruire perfettamente il sample dato in input. Questo perché la distribuzione dello spazio latente $q(z)$ non è uguale a quella a priori (Gaussiana).
Il metodo proposto (in [4])è il seguente:

1. Allenare un VAE su di un dataset e generare lo spazio latente del VAE;
2. Allenare un secondo VAE, con parametri indipendenti dal primo, usando come dataset lo spazio latente del primo VAE;
3. Infine per generare nuovi samples bisognerà fare un sampling da una Gaussiana, usare il decoder del secondo VAE per generare la rappresentazione latente del primo e tramite il decoder del primo VAE generare i samples.

Generalmente si assume che lo spazio latente del secondo livello sia minore del primo.

2.2.2 Stima dello spazio latente

Un altro approccio utilizzabile è stimare la distribuzione dello spazio latente, non fare assunzioni su di esso e provare ad approssimarlo usando un certo stimatore di densità come una Gaussian Mixture Model (GMM).
Una Gaussian Mixture Model è un modello che combina un numero finito

K di distribuzioni Gaussiane $\mathcal{N}(x|\mu_k, \Sigma_k)$ dove:

$$p(x|\theta) = \sum_{k=1}^{K} \mathcal{N}(x|\mu_k, \Sigma_k)$$

$$0 \leq \pi_k \leq 1, \sum_{k=1}^{K} \pi_k = 1$$

e θ è definito come la collezione dei parametri del modello:

$$\theta := \{\mu_k, \Sigma_k, \pi_k : k = 1, ..., K\}$$

La stima dei parametri della mixture viene tipicamente fatta attraverso l'algoritmo iterativo EM (Expectation-Maximization) ([11]).
Questa combinazione di Gaussiane permette così di modellare distribuzioni complesse.

2.2.3 Legge della varianza

Con regola della varianza si intende il fatto che lo spazio latente di un VAE debba avere una media ed una varianza che siano attorno a 0 e 1 rispettivamente.
Per ogni elemento i del dataset il VAE genererà nello spazio latente una Gaussiana con media μ_i e varianza Σ_i, quindi l'intero spazio latente sarà un insieme di Gaussiane (GMM), dove la media è la media delle medie di tutte le Gaussiane mentre la varianza è la varianza delle medie più la media delle varianze (delle Gaussiane).
Poiché la KL viene calcolata fra la Gaussian Mixutre Model e una Gaussiana con media e varianza pari a 0 e 1 rispettivamente, è ragionevole aspettarsi che la media e la varianza della GMM siano attorno a questi valori ([1]).

2.2.4 Collasso delle variabili latenti

All'aumentare della dimensione dello spazio latente si può osservare il verificarsi del collasso delle variabili latenti: alcune variabili diventano

inutilizzate (inattive) dal generatore.
La KL forza le variabili ad avere una distribuzione Normale (0,1), quindi all'inizio del training le variabili avranno una varianza intorno ad uno. Proseguendo però, visto il grande numero di punti da codificare, la varianza cala (questo è quello che ci si aspetta). Come viene mostrato in ([2], [3]) alcune variabili non si comportano in questo modo, infatti queste variabili, dette inattive, hanno una media della varianza intorno a 1 e non intorno a 0, come ci si aspetterebbe.

Parte III

3 Esperimenti

In questa sezione verrà presentata e testata la Frechét Inception Distance, in seguito si proverà a selezionare i training set secondo l'errore di ricostruzione (mean squared error) e la distanza di Mahalanobis, per poi testare diversi modelli di VAE su tali training set.
La distanza di Mahalanobis è la distanza fra un punto ed una distribuzione. Formalmente:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

dove $\vec{x}$ è il vettore delle osservazioni, $\vec{\mu}$ le medie e S la matrice di covarianza.
Idealmente si vuole comporre il training set usando le immagini più rappresentative della distribuzione del dataset usato come modello per calcolare la FID.
Saranno poi presentati i risultati ottenuti e verranno comparati con i risultati presenti in bibliografia.
Gli esperimenti sono stati condotti utilizzando CIFAR-10 ([10]), un dataset composto da 60000 immagini a colori della dimensione di 32x32 pixel. Tali immagini appartengono a dieci categorie: aeroplani, macchine, volatili, gatti, cervi, cani, rane, cavalli, barche e camion.
Il dataset utilizzato per calcolare la FID (`x_test`) è composto da 10000

immagini scelte casualmente, mentre il dataset usato per il training delle reti (`x_train`) è composto da 50000 immagini.

3.1 Frechét Inception Distance

La metrica inizialmente utilizzata per misurare la qualità delle immagini generate era l'Inception Score (IS) ([14]). Utilizzando sulle immagini generate la rete Inception-V3 ([16]), rete pre-allenata sul dataset ImageNet ([5]), si ricava la probabilità condizionata delle etichette $p(y|x)$. Se le immagini generate sono buone, tale probabilità avrà una bassa entropia. Ci si aspetta inoltre che le immagini generate siano varie, quindi la distribuzione marginale delle etichette $p(y) = \int_x p(y|x)p(y)dx$ dovrebbe avere un'alta entropia.
Secondo questa metrica, più lo score è alto, migliori saranno le immagini generate.
La Frechét Inception Distance (FID) viene proposta come un miglioramento dell'IS ([8]).
La FID misura la distanza tra la distribuzione delle features delle immagini generate e la distribuzione delle features delle immagini reali.
Questa metrica è basata sulla Fréchet Distance che misura la similitudine tra due curve. Le features vengono estratte usando la rete Inception V3, allenata su ImageNet.
Formalmente la FID è definita come:

$$FID = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g + 2(\Sigma_r\Sigma_g)^{1/2})$$

Dove $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ e $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$.
La FID viene usata per cercare di quantificare la bontà delle immagini generate da una rete neurale: più lo score sarà basso, più le features delle immagini generate saranno simili a quelle reali.
Nei seguenti esperimenti si è cercato di capire come si comporta la FID al variare degli insiemi di immagini sui quali viene calcolato lo score.

L'intento è quello di verificare quanto la metrica sia sensibile alla qualità delle immagini e se sia possibile lavorare su dataset più ridotti senza influenzare lo score in maniera significativa.

3.1.1 FID score per insiemi di immagini crescenti

Lo score è stato calcolato fra `x_test` e un insieme composto progressivamente da 10, 20, ..., 5120 immagini differenti prese casualmente da `x_train`, ripetute fino ad ottenere un insieme di 10000 immagini.

Tabella 1: FID score al variare della composizione del set.

n. immagini	media score	varianza score
10	272.123	85.793
20	236.672	61.045
40	197.087	22.918
80	156.184	17.338
160	114.150	0.970
320	76.921	0.708
640	46.918	0.150
1280	25.600	0.0640
2560	14.923	0.011
5120	7.746	0.001
10000	5.212	0.0003

Per ogni insieme lo score è stato calcolato 10 volte: nella seconda colonna è presente la media dei 10 valori mentre nella terza è stata inserita la varianza.
Ad ogni iterazione è stato fatto uno shuffling di `x_train`.
Si può osservare che lo score sembra non essere molto sensibile alla varietà delle immagini che compongono il dataset: lo scarto tra lo score calcolato su 10000 immagini e lo score calcolato su 5000 immagini non è significativo. L'idea è pertanto quella di sfruttare questa caratteristica allenando un VAE su metà del dataset così da ottenere risultati simili ad un VAE allenato sull'intero dataset.

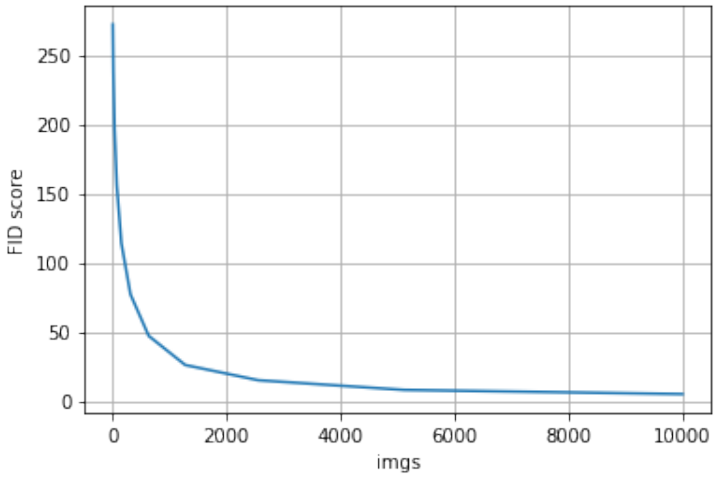

Figura 6: Sull'asse y lo score, sull'asse x la cardinalità degli insiemi.

3.1.2 FID score per immagini scomposte

Per testare la "robustezza" della FID si è calcolato lo score per immagini scomposte in diverse sezioni uguali e poi ricomposte casualmente.
Progressivamente le immagini sono state scomposte in 2, 4, 16, 64, 256 e 1024 sezioni.

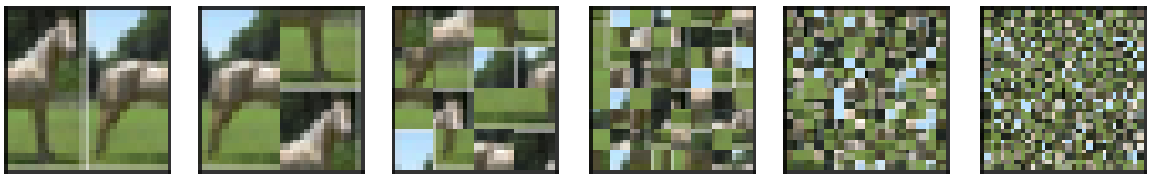

Figura 7: Permutazioni progressive della stessa immagine.

Ogni score è stato calcolato utilizzando `x_test` come distribuzione di confronto.
Da questa tabella si vede facilmente come la FID sia sensibile alla scomposizione e alla permutazione delle immagini.

Tabella 2: FID score per permutazioni crescenti.

n. di sezioni	media score	varianza score
2	8.404	0.006
4	35.279	0.044
16	103.580	0.052
64	236.659	0.102
256	300.193	0.061
1024	341.631	0.058

3.2 Struttura delle reti utilizzate

Nelle precedenti
Tutti i modelli sono stati implementati usando la libreria Keras[2]. Di seguito viene illustrata l'architettura delle reti usate.

Tabella 3: Architettura VAE primo livello.

VAE primo livello	
Encoder	Decoder
ConvNet 128 Batch Normalization ReLu	Dense 8x8x1024 Batch Normalization ReLu
ConvNet 256 Batch Normalization ReLu	Reshape (8,8,1024)
ConvNet 512 Batch Normalization ReLu	ConvNetT 256 Batch Normalization ReLu
ConvNet 1024 Batch Normalization ReLu	ConvNetT 3 Sigmoid
Flatten	
Dense 128, Dense 128	

[2] https://keras.io/

Tabella 4: Architettura VAE secondo livello.

VAE secondo livello

Encoder	Decoder
Dense 2048 ReLu	Dense 512
Dense 2048	Dense 2048 ReLu
Dense 512	Dense 512
Dense 2048 ReLu	Dense 2048
Dense 512	Dense 2048 ReLu
Dense 64, Dense 64	Dense 2048
	Dense 128

Sono stati creati quattro modelli:

1. VAE: allenato su tutto `x_train`,
   loss: 0.0035,
   legge della varianza: 0.994,
   variabili inattive: 0;

2. VAE-H: allenato su metà `x_train` (25000 immagini),
   loss: 0.0034,
   legge della varianza: 0.984,
   variabili inattive: 0;

3. VAE-RL: allenato sulle 25000 immagini che avevano l'errore di ricostruzione più basso,
   loss: 0.0034,
   legge della varianza: 0.991,
   variabili inattive: 0;

4. VAE-MD: allenato sulle 25000 immagini che avevano la distanza di Mahalanobis minore,
   loss: 0.0033,
   legge della varianza: 0.971,

variabili inattive: 0;

In seguito ognuno dei quattro modelli è stato testato aggiungendo un secondo livello, ottenendo rispettivamente:

1. VAE-2STG,
   loss: 0.0630,
   legge della varianza: 1.119,
   variabili inattive: 0;

2. VAE-H-2STG,
   loss: 0.0103,
   legge della varianza: 0.798,
   variabili inattive: 0;

3. VAE-RL-2STG,
   loss: 0.0114,
   legge della varianza: 1.030,
   variabili inattive: 0;

4. VAE-MD-2STG,
   loss: 0.0092,
   legge della varianza: 0.896,
   variabili inattive: 0;

Il learning rate di partenza per entrambi i VAE è pari a 0.0001 e viene dimezzato quando la loss smette di calare.
Tutti i primi livelli sono stati allenati per 600 epoche mentre i secondi livelli per 1200.

3.3 Risultati

Per ogni modello è stata calcolata la FID di generazione facendo sampling da una Gaussiana ($\mathcal{N}$), facendo il fitting dello spazio latente con una GMM di 10 componenti (GMM) e la FID di ricostruzione (Rec).

Tabella 5: FID score dei vari modelli.
I risultati dei modelli * sono presi da [7].
I risultati dei modelli ** sono presi da [4]

	$\mathcal{N}$	GMM	Rec
VAE	98.459	83.037	18.078
VAE-H	117.867	100.126	37.507
VAE-RL	116.931	98.609	38.640
VAE-MD	118.391	103.456	40.947
VAE-2STG	92.394	—	—
VAE-H-2STG	120.898	—	—
VAE-RL-2STG	115.398	—	—
VAE-MD-2STG	111.656	—	—
VAE*	106.37	103.78	57.94
2-Stage VAE**	72.9	—	—
VAE**	106.0	—	—

Generalmente sembra che il VAE con secondo livello dia risultati peggiori rispetto ad una stima con una GMM. Guardando invece ai risultati in letteratura pare il contrario. I risultati riguardanti il VAE sono concordi. Inoltre, sembra che selezionare le immagini del training set secondo l'errore di ricostruzione o la distanza di Mahalanobis non porti particolari vantaggi. Stessa cosa dicasi per la riduzione del training set: infatti, dimezzando il training set lo score aumenta di circa 20 punti.

(a) VAE-H

(b) VAE-RL

(c) VAE-MD

Figura 8: Immagini generate facendo sampling da una Gaussiana.

(a) VAE-H

(b) VAE-RL

(c) VAE-MD

Figura 9: Immagini generate facendo il fitting con una GMM.

4 Conclusioni

In questa tesi è stata fatta una panoramica generale sull'apprendimento automatico, per poi concentrarsi sui modelli generativi, come i Variational Autoencoders. Ne sono stati spiegati il funzionamento, i vantaggi e i problemi con relative possibili soluzioni.
Si è poi discusso della Frechét Inception Distance, metrica usata per valutare le immagini generate, qui testata per studiarne il comportamento in relazione al dataset dato in input.
Utilizzando le tecniche citate nella seconda parte della tesi sono stati infine implementati diversi modelli di VAE, in seguito allenati su dataset composti in maniera differente: dataset dimezzato, errore di ricostruzione, distanza di Mahalanobis. Da ultimo, i modelli sono stati testati in fase di generazione e ricostruzione, dove possibile.
I risultati ottenuti fanno pensare che ridurre il training set non sia particolarmente conveniente. Allo stesso modo, neanche le metriche usate per ridurre e selezionare il dataset hanno dato i risultati sperati.
Nonostante i risultati non soddisfacenti è possibile che con una sperimentazione più approfondita si riescano ad ottenere dei miglioramenti.

Riferimenti bibliografici

[1] Andrea Asperti. ≪About Generative Aspects of Variational Autoencoders≫. In: *Proceedings of the Fifth International Conference on Machine Learning, Optimization, and Data Science – September 10-13, 2019 – Certosa di Pontignano, Siena – Tuscany, Italy*. LNCS (to appear). Springer, 2019.

[2] Andrea Asperti. ≪Sparsity in Variational Autoencoders≫. In: *Proceedings of the First International Conference on Advances in Signal Processing and Artificial Intelligence, ASPAI 2015, Barcelona, Spain, 20-22 March 2019*. 2019. URL: http://arxiv.org/abs/1812.07238.

[3] Andrea Asperti. ≪Variational Autoencoders and the variable collapse phenomenon≫. In: *Sensors & Transducers, to appear* (2019).

[4] Bin Dai e David Wipf. *Diagnosing and Enhancing VAE Models*. 2019. arXiv: 1903.05789 [cs.LG].

[5] J. Deng et al. ≪ImageNet: A Large-Scale Hierarchical Image Database≫. In: *CVPR09*. 2009.

[6] Carl Doersch. *Tutorial on Variational Autoencoders*. 2016. arXiv: 1606.05908 [stat.ML].

[7] Partha Ghosh et al. *From Variational to Deterministic Autoencoders*. 2019. arXiv: 1903.12436 [cs.LG].

[8] Martin Heusel et al. ≪GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium≫. In: *NIPS*. 2017, pp. 6626–6637.

[9] Diederik P. Kingma e Max Welling. ≪Auto-Encoding Variational Bayes≫. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. URL: http://arxiv.org/abs/1312.6114.

[10] Matt J. Kusner, Brooks Paige e José Miguel Hernández-Lobato. ≪Grammar Variational Autoencoder≫. In: *Proceedings of the 34th International Conference on Machine Learning*. A cura di Doina Precup e Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, ago. 2017, pp. 1945–1954. URL: http://proceedings.mlr.press/v70/kusner17a.html.

[11] A Aldo Faisal Marc Peter Deisenroth e Cheng Soon Ong. *Mathematics for Machine Learning*. 2019, p. 407. URL: https://mml-book.github.io/.

[12] Sebastian Raschka e Vahid Mirjalili. *Python Machine Learning, 3rd Ed.* Birmingham, UK: Packt Publishing, 2019, p. 748. ISBN: 978-1789955750.

[13] Danilo Jimenez Rezende, Shakir Mohamed e Daan Wierstra. ≪Stochastic Backpropagation and Approximate Inference in Deep Generative Models≫. In: *Proceedings of the 31st International Conference on Machine Learning*. A cura di Eric P. Xing e Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, giu. 2014, pp. 1278–1286. URL: http://proceedings.mlr.press/v32/rezende14.html.

[14] Tim Salimans et al. ≪Improved Techniques for Training GANs≫. In: *arXiv e-prints*, arXiv:1606.03498 (giu. 2016), p. 10. arXiv: 1606.03498 [cs.LG].

[15] Kihyuk Sohn, Honglak Lee e Xinchen Yan. ≪Learning Structured Output Representation using Deep Conditional Generative Models≫. In: *Advances in Neural Information Processing Systems 28*. A cura di C. Cortes et al. Curran Associates, Inc., 2015, pp. 3483–3491. URL: http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf.

[16] Christian Szegedy et al. ≪Rethinking the Inception Architecture for Computer Vision≫. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2016.