



# Progetto Machine Learning

ALAD (Adversarially Learned Anomaly Detection)

---

**Edoardo Misurelli**

Corso di Intelligenza Artificiale

# Introduzione

---

Il progetto ha riguardato lo studio e l'implementazione dell'architettura **ALAD** (Adversarially Learned Anomaly Detection);

Nel lavoro svolto mi sono concentrato su alcuni punti:

- Studio del modello.
- Sviluppo dell'architettura.
- Adattamento dell'architettura ai nuovi dati.
- Studio delle prestazioni del modello variando la dimensione latente.

# GANs e Anomaly Detection

---

Le GANs consistono di due reti neurali principali:

- **Generatore** : Produce dati falsi che cercano di imitare i dati reali.
- **Discriminatore** : Valuta se i dati provengono dal dataset reale o sono stati generati dal Generatore.

Nei task di anomaly detection le GANs vengono addestrate spesso a ricostruire i dati ‘normali’, ma il costo computazionale in fase di inferenza rimane molto alto.

Un approccio spesso utilizzato è confrontare la rappresentazione nella dimensione latente degli esempi per individuare le anomalie.

Lo scopo di **ALAD** è ridurre il costo in fase di inferenza, il quale rende proibitivo l'utilizzo delle GANs soprattutto in casi in cui i dati siano ad alta dimensionalità (come nelle immagini).

# Architettura ALAD

L'architettura del modello, si basa su una classe di GANs che incorpora una rete Encoder, la quale viene addestrata in parallelo a mappare i dati in ingresso (indicati con  $x$ ), nella dimensione latente (indicata con  $z$ ).

## Encoder

- si occupa di ridurre la dimensionalità dei dati, mappandoli in uno spazio latente. La rete Encoder viene addestrata in parallelo alla rete GAN;

## Generator/Decoder

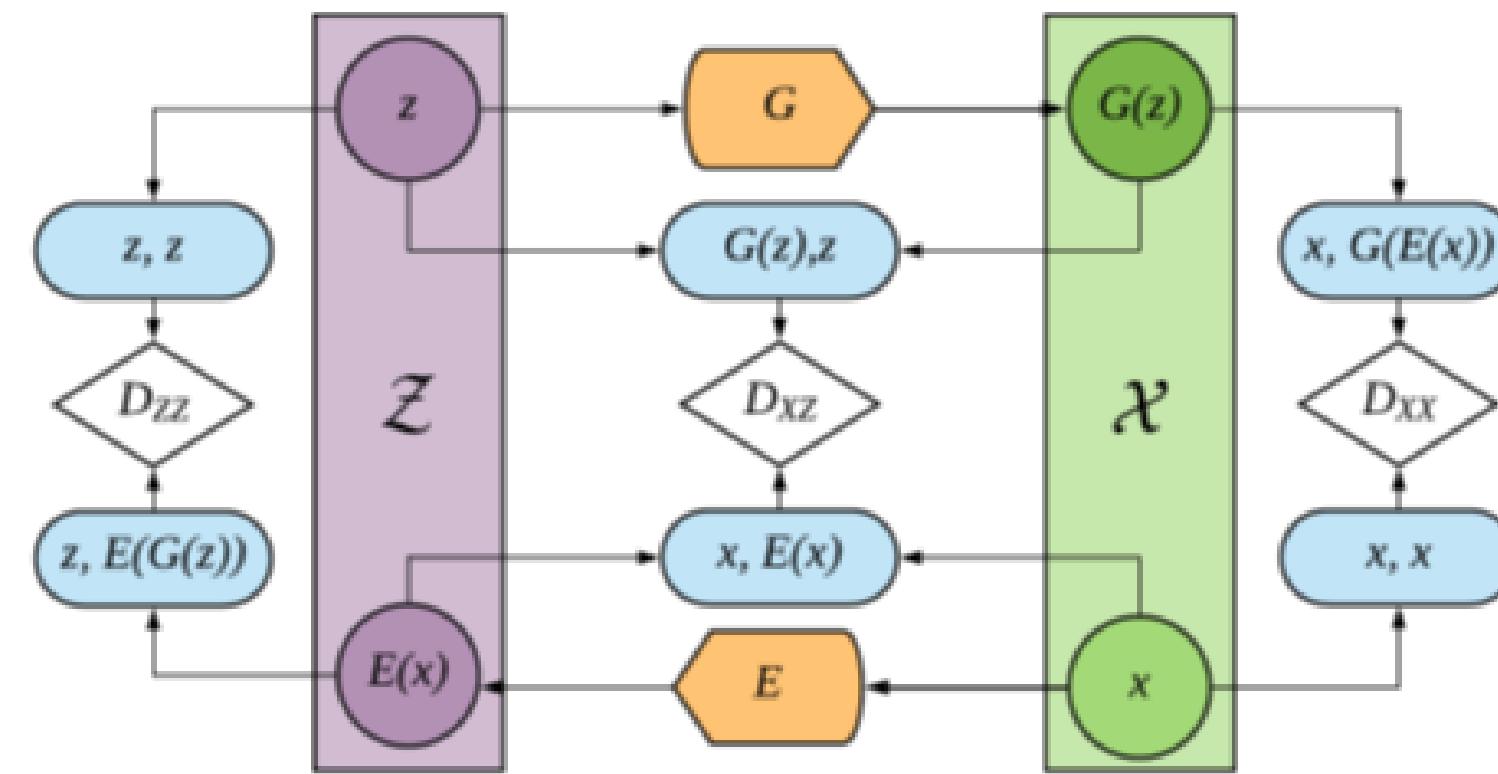
- il generatore deve creare immagini simili alle originali, ricostruendole però dalla dimensione latente;

## Discriminatore $xz$

- discriminatore che discrimina tra spazio dei dati e spazio delle features, questo particolare discriminatore viene introdotto nel modello BiGAN

## Discriminatore $xx$ e Discriminatore $zz$

- entrambi introdotti per stabilizzare il training del modello



Struttura della GAN : in bianco i discriminatori, in arancio Generatore ed Encoder;  
Le due regioni rappresentate in viola e verde rappresentano rispettivamente  
spazio delle features e spazio dei dati

# Punti di forza

---



## Efficienza

La presenza della rete **Encoder**, migliora le prestazioni della GAN



## Stabilità

I due **Discriminatori** secondari, aiutano a stabilizzare il training della GAN



## Reconstruction Based

Veloce e naturale per la struttura intrinseca della rete

# Approccio Reconstruction Based

---

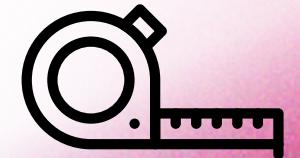
La tecnica di rilevamento delle anomalie è basata sulla ricostruzione, il modello valuta quanto un campione si discosta dalla sua ricostruzione. I campioni normali dovrebbero essere ricostruiti accuratamente, mentre i campioni anomali probabilmente saranno ricostruiti in modo approssimativo.

Nel paper però si fa riferimento ad un nuovo tipo di **score**, quest'ultimo calcola la distanza tra i sample nello spazio delle features di Dxx (**Cycle Consistency Discriminator**).

$$A(x) = \|f_{xx}(x, x) - f_{xx}(x, G(E(x)))\|_1$$

# Obiettivi del progetto

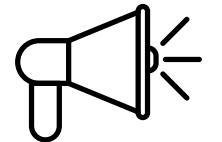
---



**Adattamento  
architettura**



**Studio sulla  
dimensione  
latente**



**Analisi e  
confronto tra  
modelli**

# Adattamento Architettura

---

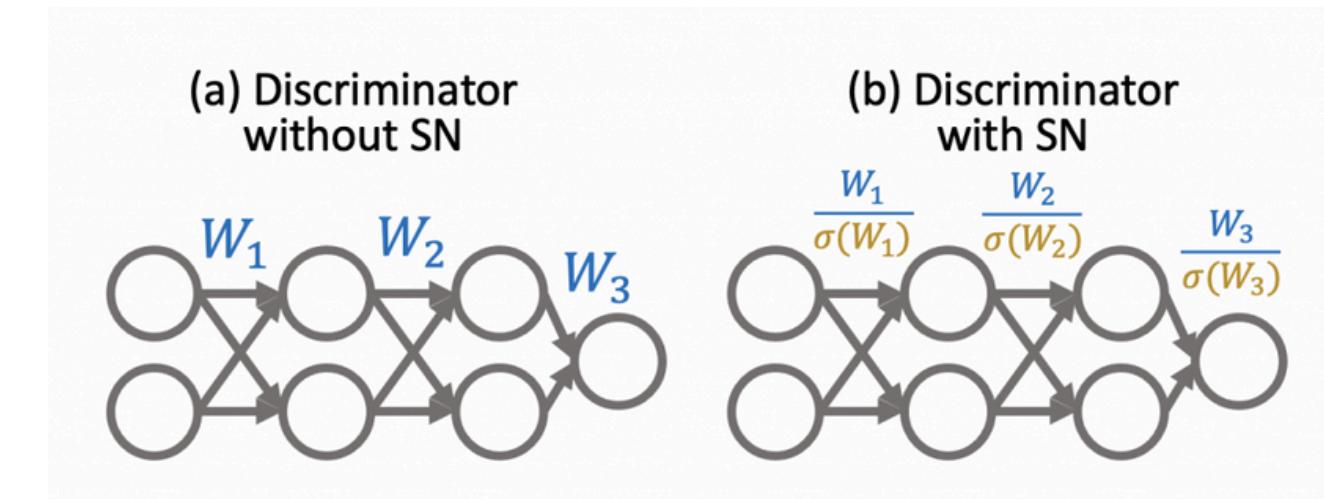
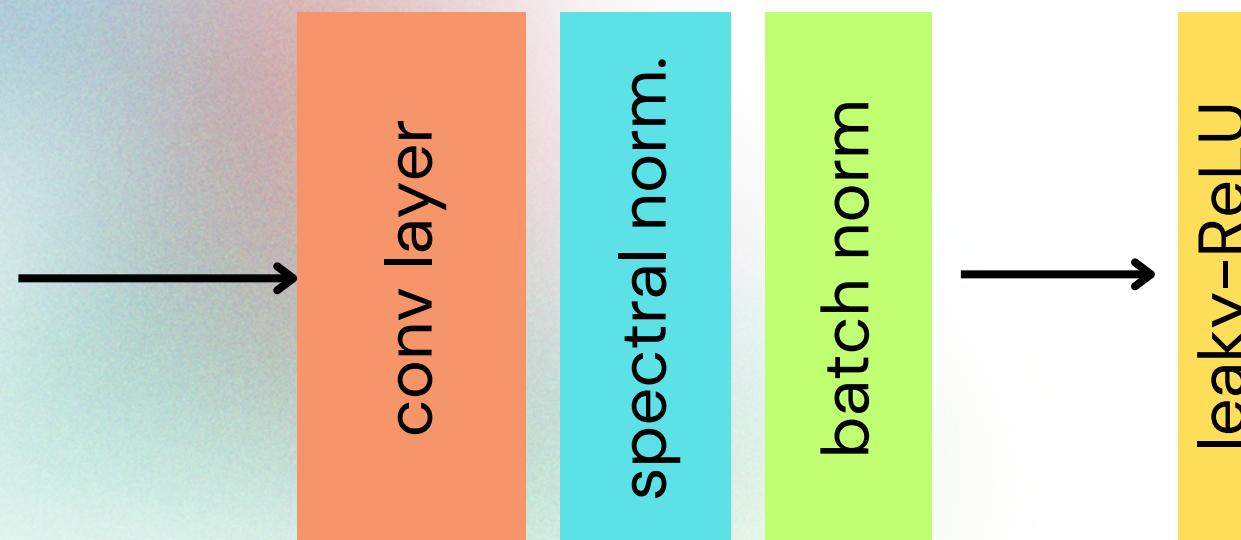
L'adattamento della rete ha riguardato specifiche modifiche sui moduli in modo di rendere in grado il modello di processare immagini di dimensione maggiore.

I moduli interessati sono stati: Encoder, Generatore, Discriminatore xz.

- **Encoder Net:** aggiunta di un primo blocco Convoluzionale seguito da un layer di Batch Normalization ed un layer di attivazione (LeakyReLU);
- **Decoder Net:** aggiunta di un ultimo blocco Convoluzionale Trasposto, seguito da un layer di Batch Normalization ed un layer di attivazione (ReLU);
- **Discriminator xz Net:** aggiunta di un primo blocco Convoluzionale seguito da un layer di Batch Normalization ed un layer di attivazione (LeakyReLU);

A tutti i blocchi convoluzionali è stata applicata la Spectral Normalization, seguendo l'architettura orginale. Questa tecnica migliora la stabilità del training della GAN.

Esempio di blocco aggiunto  
alla rete Encoder



La **Spectral Normalization**  
agisce direttamente sui pesi  
della rete , migliorando la  
stabilità nel training...

# Setup and Training

---

- **Tecnologie utilizzate:** Python, Tensorflow 2.x, Colab.
- **Dataset utilizzati:** CIFAR-10 (32x32), LINNAEUS-5 (64x64)
- **Traininig modello 32x32:** 80% training, 20% testing, (25 % del t.s. utilizzato come validation set);
- **Traininig modello 64x64:** 80% training, 20% testing, (10 % del t.s. utilizzato come validation set), dataset soggetto a data-augmentation tramite trasformazioni di zoom, rotazione e cambiamento di tonalità dei pixel
- **Classe utilizzata negli esperimenti:** come classe **normale** ho utilizzato la classe 'Bird' presente in entrambi i dataset. La strategia utilizzata è la **One vs All**.

	<b>Modello originale</b>	<b>Modello riadattato</b>
<b>Epoche</b>	100	100
<b>Dimensione Input</b>	32 x 32	64 x 64
<b>Learning Rate</b>	0.0002	0.0002
<b>Batch Size</b>	32	64
<b>Dimensione Training Set</b>	4473 (CIFAR-10)	3011
<b>Dimensione Test Set</b>	10000 (CIFAR-10)	2000
<b>Dimensione Validation Set</b>	527 (CIFAR-10)	120
<b>Ema decay</b>	0.999	0.999
<b>Dimensione Latente</b>	100	100/300/500/800

# Utilizzo Validation Set

Il validation set è stato utilizzato per monitorare l'errore di ricostruzione, durante il training della rete;

Nell'esperimento proposto l'errore di ricostruzione sul validation è stato calcolato tramite **norma L2**, diversamente da quello originale, dove si è optato di utilizzare la **norma L1** tra le distanze nel **features space**.

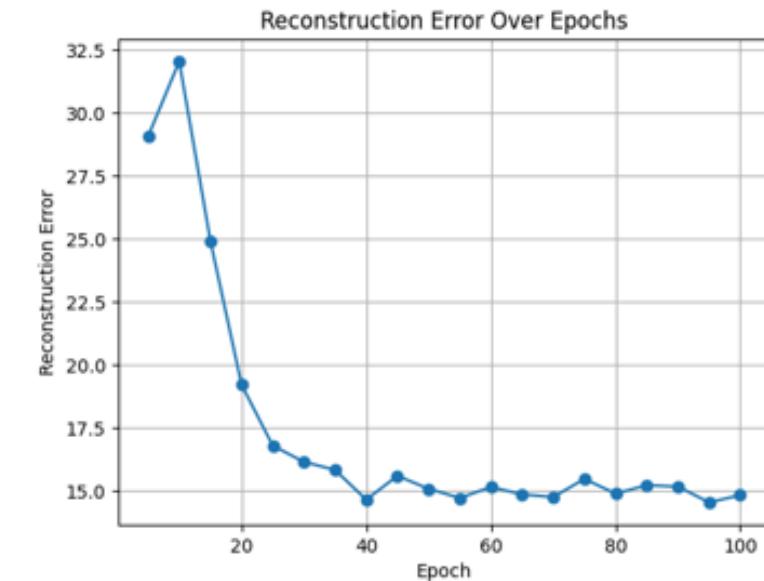


Figura 3: Architettura 32x32 errore di ricostruzione sul Validation set

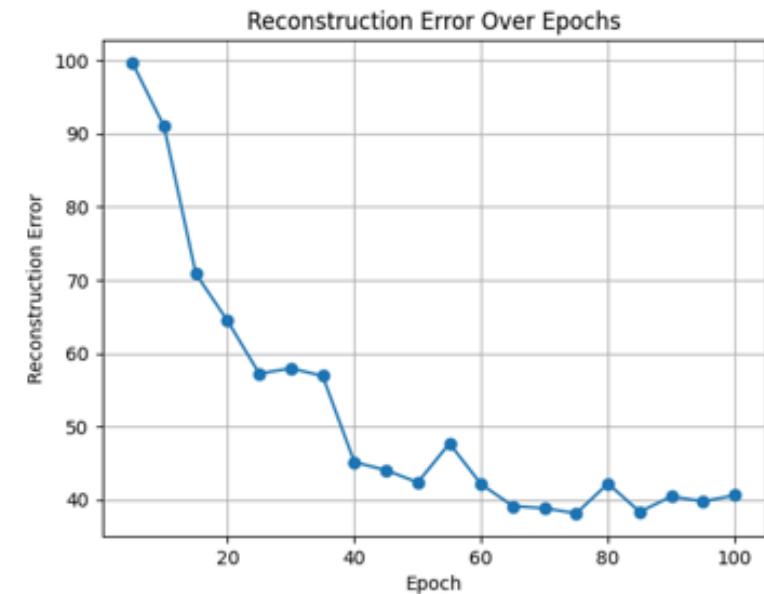
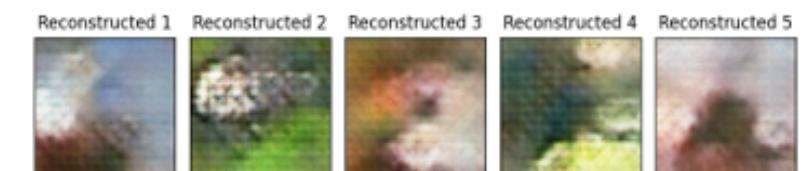


Figura 4: Architettura 64x64 errore di ricostruzione sul Validation set



# Testing

- Confronto delle prestazioni tra modello **originale** e modello **riadattato**:
- Studio sulla **dimensione latente**; il nuovo modello è stato testato variando l'iperparametro inerente alla dimensione latente.

$$L_1 : A(x) = \|x - x'\|_1$$

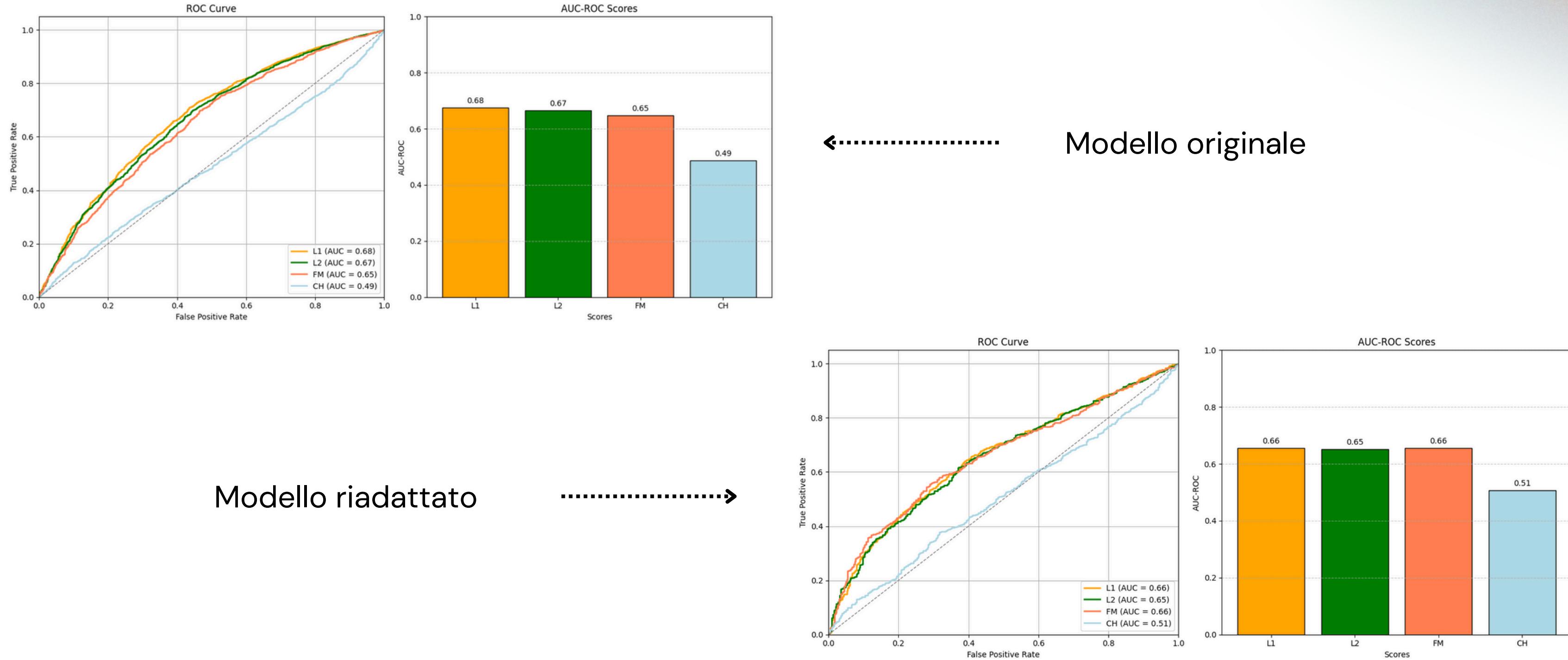
$$L_2 : A(x) = \|x - x'\|_2$$

$$\text{Logits} : A(x) = \log(D_{xx}(x, x'))$$

$$\text{Features} : A(x) = \|f_{xx}(x, x') - f_{xx}(x, x')\|_1$$

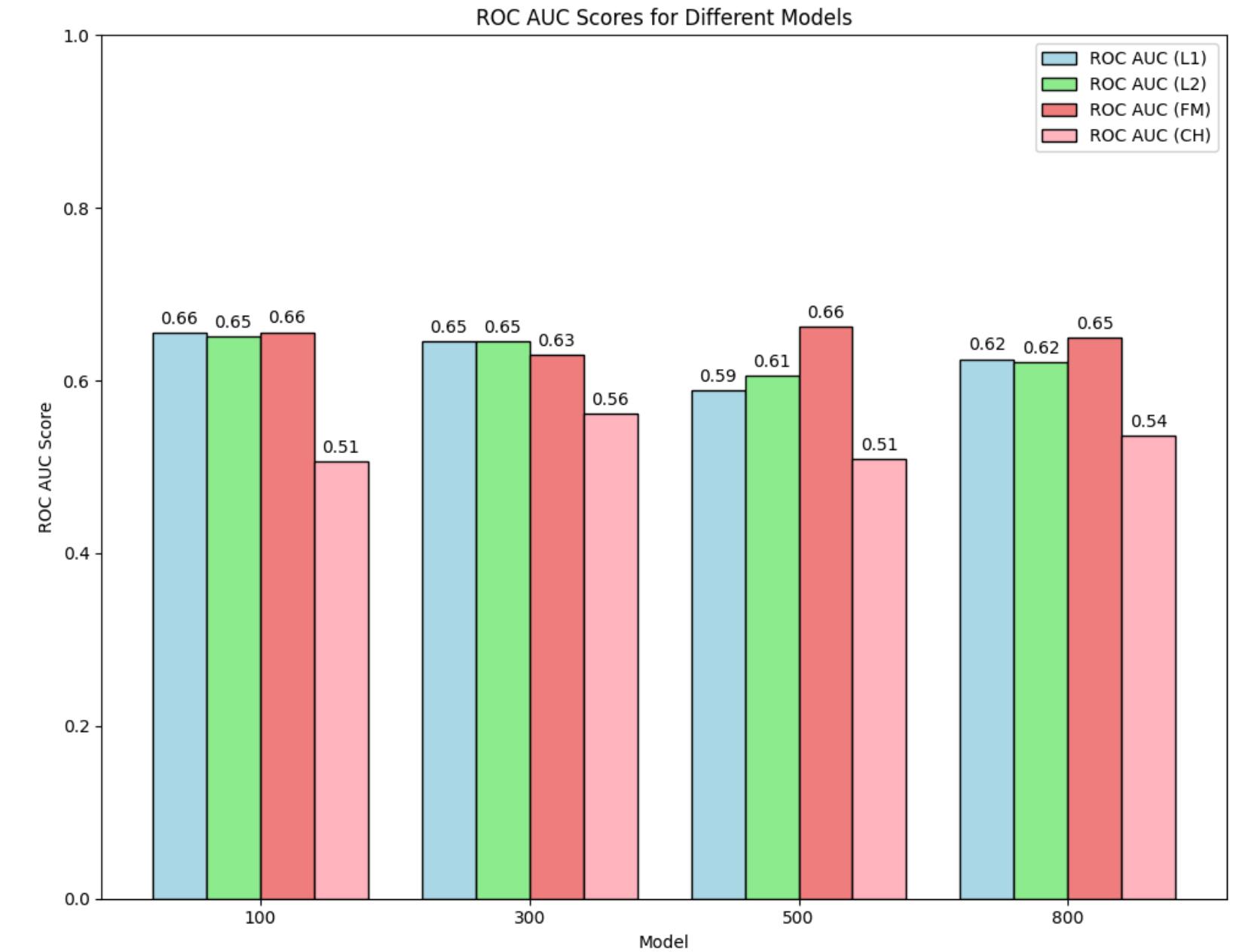
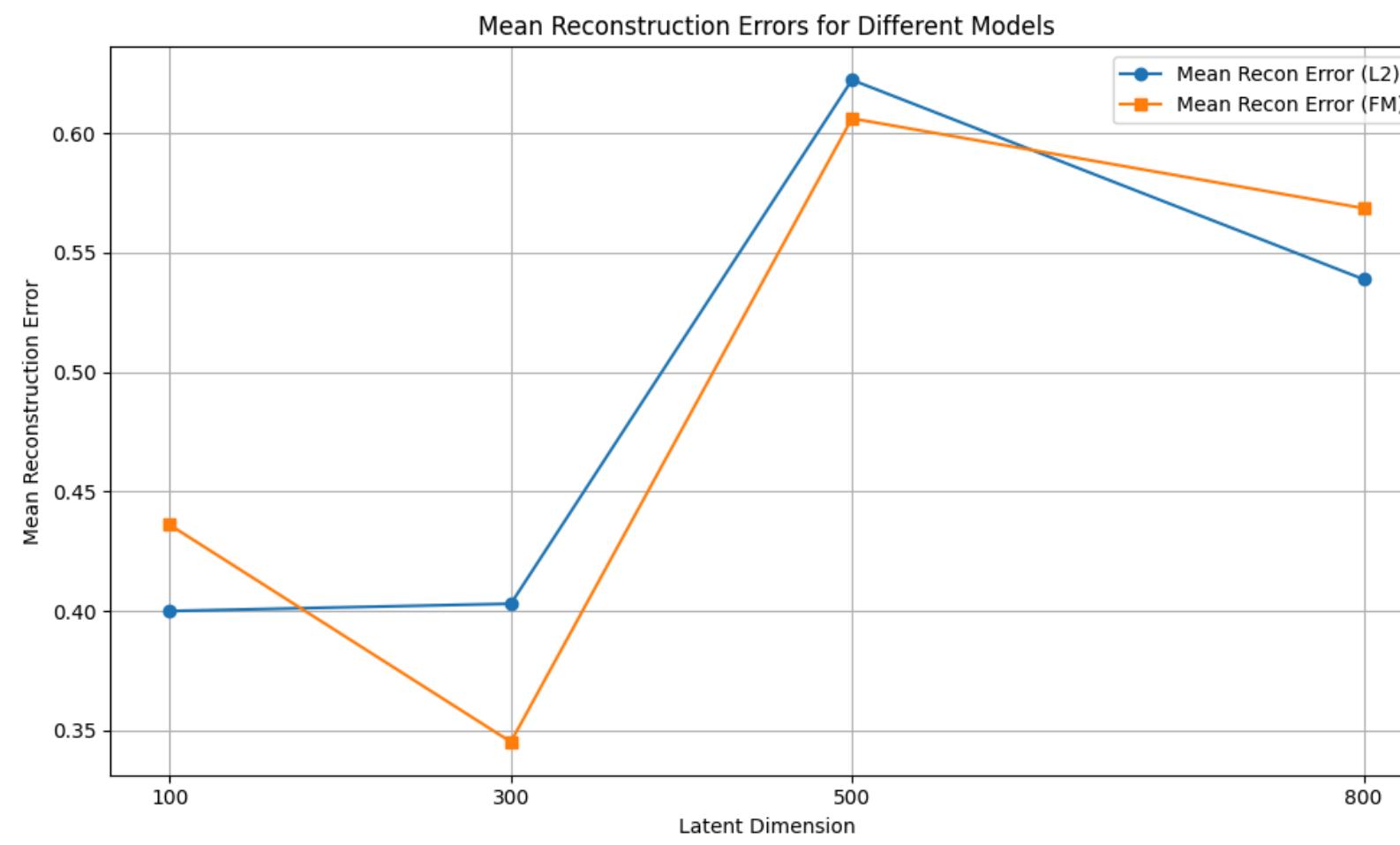
- Le metriche utilizzate nel testing fanno riferimento a score calcolati sull'**errore di ricostruzione** tra immagini ricostruite e originali.

# Risultati ottenuti



# Studio sulla dimensione latente

---



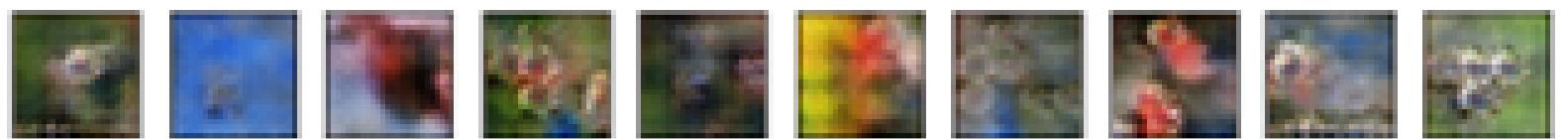
# Esempi di ricostruzione

---

Ricostruzione esempi **normali** – classe 'Bird'



Ricostruzione esempi **anomali**



# Conclusioni

---

L'esperimento ha mostrato come il modello si comporti bene, pur aumentando la dimensione dei sample in input alla rete, confermando anche la scelta di utilizzare come errore di ricostruzione lo **score FM** (invariante al cambiamento di dimensione);

Utilizzare come score sul validation **L2**, non sembra aver comunque cambiato le prestazioni finali del modello.

Lo studio sulla dimensione latente, ha portato invece alla luce un aspetto controintuitivo, ovvero aumentando la dimensione dello spazio, l'errore di ricostruzione medio non è migliorato anzi...

Questo può essere dovuto sicuramente a diversi fattori, uno di questi potrebbe essere il numero ridotto di epoche utilizzate, infatti una **dimensione latente maggiore** potrebbe portare il modello a convergere più lentamente e dunque ad avere un alto errore di ricostruzione, dopo poche epoche.

Lo score utilizzato **FM**, sembra comunque essere una buona soluzione anche a questo problema, mantenendo curve **ROC-AUC** simili, nonostante le differenze di dimensione latente tra i modelli.

**Grazie per l'attenzione !**