ALMA MATER STUDIORUM - UNIVERSITÁ DI BOLOGNA
CAMPUS BOLOGNA

Dipartimento Informatica, Scienze e Ingegneria
Corso di laurea magistrale in Ingegneria Informatica

Rilevazione di anomalie da immagini mediante deep-learning

Elaborato in
SISTEMI DIGITALI M

Relatore

Prof. Stefano Mattoccia

Co-relatore

Dott. Alessio Mingozzi

Presentato da

Milo Marchetti

Anno accademico 2021-2022

Sommario

Il presente documento di tesi illustra le tecniche, i modelli e le soluzioni impiegate per far fronte al problema dell'individuazione di anomalie relative alla produzione industriale di componenti elettronici tramite tecniche di Computer Vision e Deep Learning.

L'individuazione delle anomalie, o anomaly detection, è un campo di studio molto vasto dove la complessità delle soluzioni da impiegare è fortemente dipendente dal contesto in cui vengono applicate. In questo caso infatti, la scarsa quantità di dati a disposizione e la ridotta dimensione delle anomalie all'interno delle immagini ne rende particolarmente complessa l'individuazione.

Lo studio di possibili soluzioni è partito dai modelli basati sull'architettura GAN, Generative Adversarial Network. I risultati ottenuti con uno dei modelli più promettenti, denominato GANomaly, hanno dimostrato le ottime capacità della rete quando addestrata con la giusta parametrizzazione. Purtroppo però l'impossibilità del modello di andare ad evidenziare nell'immagine le anomalie da esso identificate, ha portato allo studio di altri tipi di architetture, i Variational Autoencoders. Gli approcci basati su autoencoders hanno eguagliato i risultati ottenuti da GANomaly, dando in più la possibilità di verificare la correttezza delle anomalie individuate.

Infine, nel tentativo di sopperire alla scarsità di immagini, si sono tentati vari approcci di data augmentation per ingrandire le dimensioni del dataset e migliorare i risultati ottenuti, nonché verificare la robustezza dei modelli realizzati. Inizialmente si è optato per augmentation classica, quindi basata su trasformazioni geometriche, per poi passare ad approcci parametrizzati come DCGAN e WGAN. Tuttavia, in entrambi i casi la capacità di generare immagini sintetiche non è stata abbastanza efficace da poter essere utilizzata con successo per addestrare reti di anomaly detection.

Indice

1. Introduzione

Oggigiorno la produzione in larga scala di beni strumentali o di consumo, in particolare di quelli tecnologici, è molto spesso supportata da sistemi autonomi in grado gestire il processo realizzativo di un prodotto. Molte delle mansioni che prima erano svolte da operatori umani adesso sono sempre più delegate a robot o sistemi adattivi. Fra queste ve n'è una che solitamente si trova alla fine del processo produttivo che però risulta essere fondamentale, cioè il controllo qualità e l'identificazione di anomalie.

L'avanzamento tecnologico nel campo del deep learning applicato alla computer vision ha permesso di automatizzare anche quest'ultimo. Grazie all'ausilio delle reti neurali più recenti e un quantitativo consono di dati a disposizione è possibile analizzare le immagini dei prodotti sulla linea di produzione e verificare la presenza di anomalie.

Il quantitativo e la complessità dei dati presi in considerazione giocano un ruolo fondamentale nella riuscita dell'applicazione di queste tecniche. Acquisire un numero sufficiente di dati per produzioni su larga scala potrebbe non sembrare un problema. Nel caso invece di prototipi e prodotti innovativi, i campioni realizzati sono spesso limitati e in questi casi l'individuazione delle anomalie risulta essere complessa o addirittura impraticabile.

In questo studio, svolto in collaborazione con Marposs Spa, verranno prese in considerazione alcune delle tecniche più recenti di anomaly detection e applicate nel contesto sopranzi descritto così da verificare il funzionamento e la robustezza dei vari modelli. Inoltre si cercherà di esplorare la possibilità di risolvere i problemi dovuti alla limitatezza del dataset tramite tecniche di data augmentation anch'esse basate su deep learning.

Per l'implementazione, l'addestramento e il test delle reti verrà utilizzato il supercomputer Marconi100, uno dei più potenti in Europa, facente parte del centro di calcolo Cineca.

2. Strumenti Utilizzati

Durante lo svolgimento di questo studio sono state impiegate varie librerie e risorse hardware per l'implementazione e l'addestramento dei modelli impiegati.

2.1. Librerie

Le principali librerie utilizzate ricadono principalmente sotto due diverse categorie: quelle per l'implementazione di reti neurali e quelle relative alla manipolazione delle immagini.

Per quanto riguarda le reti sono state utilizzate:

- Tensorflow

  TensorFlow è una libreria software open source per machine learning, che fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio.

- Keras

  Keras è una libreria open source per machine learning e le reti neurali, scritta in Python. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, e supporta come back-end le librerie TensorFlow, Microsoft CNTK e Theano.

- Scikit-learn

  Scikit-learn è una libreria open source di apprendimento automatico per Python. Contiene algoritmi di classificazione, regressione e clustering e macchine a vettori di supporto, regressione logistica, classificatore bayesiano, k-mean e DBSCAN, ed è progettato per operare con le librerie NumPy e SciPy

- Horovod

  Horovod è un framework gratuito e open source per la realizzazione di modelli di deep learning distribuiti che utilizzano TensorFlow, Keras, PyTorch e Apache MXNet.

Relativamente alle immagini invece abbiamo:

- OpenCV

  OpenCV è una libreria software multipiattaforma nell'ambito della visione artificiale in tempo reale.

- Albumentation

  Albumentation è una libreria software utilizzata per migliorare le performance delle reti convoluzionali profonde tramite data augmentation.

2.2. Hardware

Tutti i modelli utilizzati sono stati realizzati e addestrati su Marconi100, il nuovo cluster accelerato basato sull'architettura IBM Power9 e Volta NVIDIA GPU's. Di seguito sono riportate alcune delle principali caratteristiche del cluster:

- Nodi - 980
- Processori - 2x16 cores IBM POWER9 AC922 at 3.1 GHz
- Acceleratori - 4 x NVIDIA Volta V100 GPUs, Nvlink 2.0, 16GB
- Cores - 32core/node
- Ram -256 GB/node
- Peak Performance - ~32 PFlop/s

3. Dataset

I dati presi in considerazione in questo progetto sono stati forniti da MARPOSS Spa. sotto accordo di riservatezza. Per questa ragione ogni raffigurazione esplicita dei dati, o della loro elaborazione, nel corso del documento è stata sostituita con dei mockup a scopo illustrativo.

Le immagini rappresentano dei componenti elettronici relativi a delle piastrine di batterie. Il dataset è caratterizzato da 207 immagini prive di anomalie e 53 con anomalie, per un totale di 260 elementi in formato *bmp* di dimensione 4096×6000 pixel. Considerando la dimensione delle immagini e la loro complessità si è deciso di rimuovere tutti gli elementi estranei dalla piastrina isolandola dal resto dell'immagine. Attraverso delle operazioni di edge detection è stata estrapolata dal background e ritagliata. In questo modo la dimensione delle immagini si è ridotta fino a 240×1200 pixel.

3.1. Suddivisione

Le immagini sono caratterizzate da tre principali componenti che formano complessivamente la piastrina: anodo, vetrino e catodo. Ognuno di essi è diverso e può presentare anomalie specifiche. Data la complessità generale dell'immagine e la sua dimensione elevata, nonostante sia già stata ridotta, al fine di applicare le tecniche di anomaly detection si è deciso di prendere in considerazione singolarmente le varie sottoparti. Il dataset è stato quindi ulteriormente modificato ed elaborato al fine di generare tre diversi dataset, uno per ogni elemento (Figure 1).

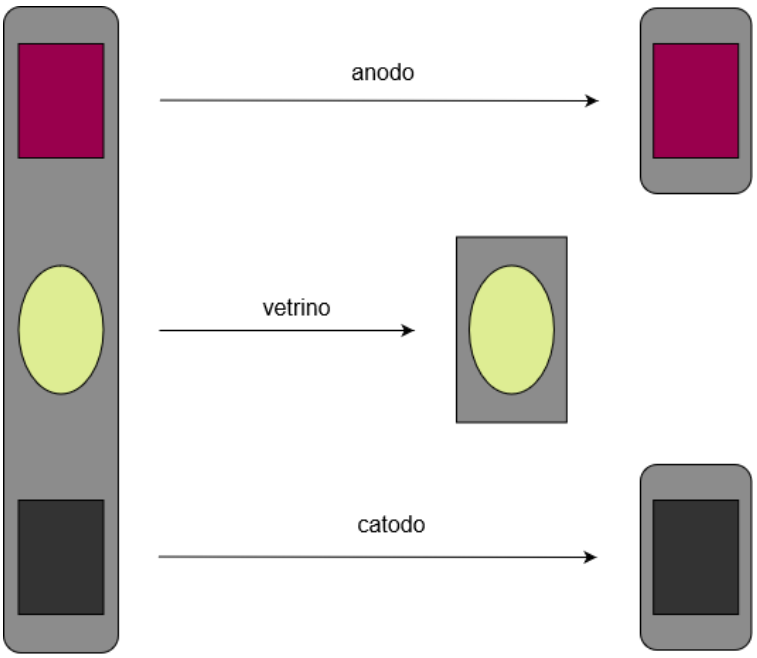

Figure 1. Suddivisione dataset

4. Generative Adversarial Network

Nel campo del machine/deep learning, si definisce generative adversarial network, o GAN, una particolare struttura, introdotta per la prima volta da Ian Goodfellow, in cui due reti neurali competono in un gioco *minmax*. Questo framework si pone come obiettivo quello di definire un modello in grado di rappresentare una distribuzione di probabilità che rispecchi dati reali che si utilizzano quando si applica il machine/deep learning. Prima delle GANs sono stati proposti dei modelli, definiti Deep Generative Models, i quali però non hanno riscontrato un gran successo a causa dell'incapacità di approssimare complessi calcoli probabilistici.

Nella sua versione originale, una rete generativa avversaria è composta da due componenti: un modello generativo, o generatore G, e un modello discriminativo, o discriminatore D, entrambi realizzati tramite reti neurali. Il generatore G ha lo scopo di approssimare la distribuzione di probabilità dei dati forniti, mentre il discriminatore D ha il compito di stimare la probabilità che il dato ricevuto sia stato o meno generato da G (Figure 2). La natura competitiva, dovuta alla scelta delle loss impiegate nel processo di backpropagation, fa sì che, dopo un adeguato periodo di training, il generatore sia in grado di creare falsi dati quasi indistinguibili dai dati reali.

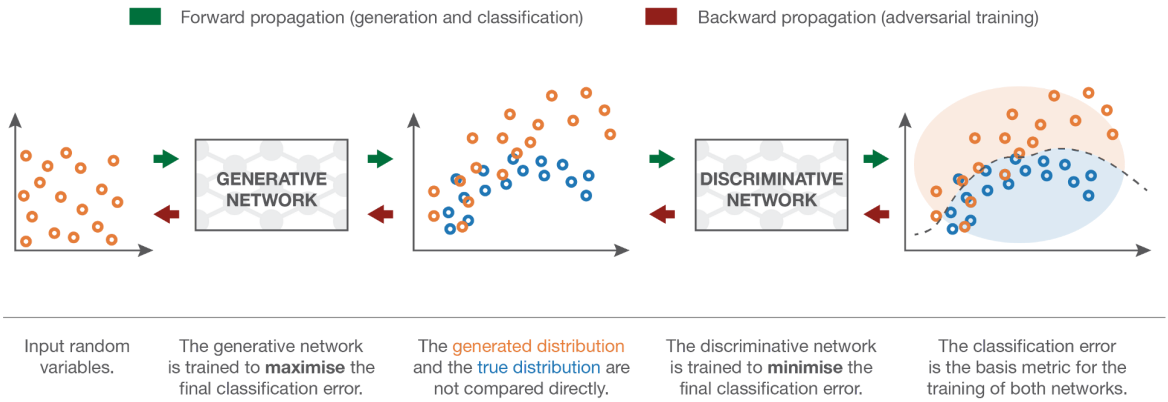

Figure 2. GAN generatore e discriminatore

Al fine di approssimare la distribuzione p_g relativa ai dati x, viene definita una variabile di solo rumore $p_z(z)$ che verrà fornita in input a G, dove G è un

multilayer perceptron $G(z; \theta_g)$ con parametri θ_g. Inoltre, viene definito un secondo *multilayer perceptron* $D(x; \theta_d)$ che da come risultato uno scalare che rappresenta la probabilità che x sia generato da G. D viene addestrato al fine di massimizzare la probabilità di classificare correttamente i dati forniti in input, mentre G viene addestrato per minimizzare $\log(1 - D(G(z)))$. In altre parole D e G giocano ad un gioco minmax a due giocatori con la *value function* $V(G, D)$:

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

La figura 3 (Figure 3) mostra come questo si traduce nell'architettura finale del modello.

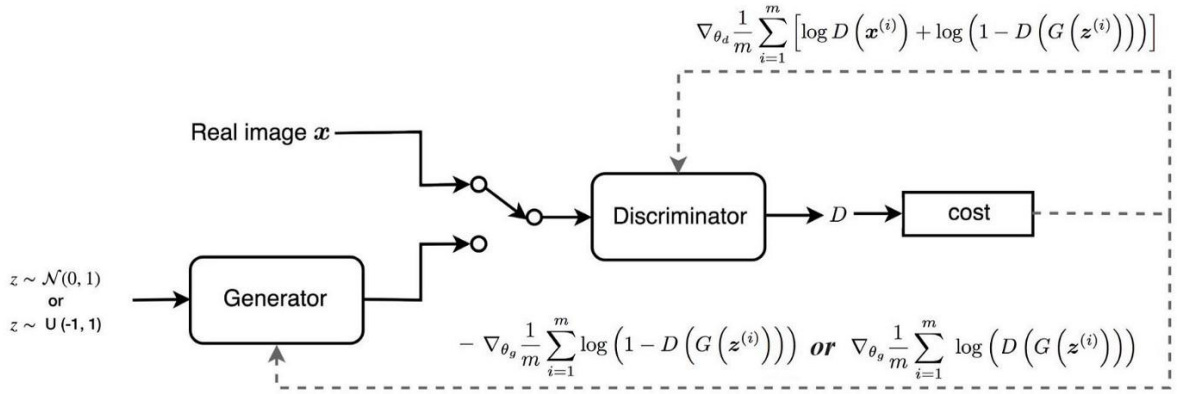

Figure 3. Architettura GAN

5. GANomaly

Il problema anomaly detection in computer vision e in data mining si può riassumere in poche e semplici parole: distinguere dati anomali rispetto a dati normali. La base di conoscenza a disposizione in questi casi è solitamente composta da un dataset sbilanciato nei confronti dei dati normali. Nonostante questo problema risulti ricadere logicamente nella categoria dei supervised learning problems, è significativamente più interessante tentare di identificare anomalie sconosciute o non ancora verificatesi. Questo perché, nella maggior parte dei casi reali, esempi di dati anomali sono difficili da trovare e non è possibile modellare un sistema supervisionato, come un classificatore binario, per risolvere il problema. Di conseguenza la strategia che viene applicata è quella di utilizzare i dati normali al fine di apprendere la distribuzione di probabilità e poter identificare i dati anomali come quei dati che si distanziano da tale distribuzione. Il problema quindi può essere ridefinito come un one-class semi-supervised learning problem.

Negli ultimi anni sono stati proposti molti modelli in grado di eseguire anomaly detection con buoni risultati, e molti di essi si sono basati sulle nuove scoperte in ambito machine learning come le sopracitate GAN (Generative Adversarial Network). GANomaly [1] infatti è uno di questi. A seguire viene riportata la definizione formale del problema e dell'architettura che caratterizza questo approccio.

5.1. Definizione del problema

È dato un dataset di dimensione D comprendente M immagini normali tale che $D = \{X_1,..., X_M\}$ e un dataset di dimensioni ridotte $\widehat{D}$ comprendente N (dove $M \gg N$) immagini normali e anomale tale che $\widehat{D} = \{(\widehat{X}_1, y_1),..., (\widehat{X}_N, y_N)\}$ dove $y_i \in [0, 1]$ denota la label delle immagini. L'obiettivo è quello di utilizzare D per approssimare la distribuzione dei dati normali per poi identificare le anomalie in $\widehat{D}$ durante la fase di inferenza. Le anomalie vengono rilevate grazie ad un sistema di punteggio $A(x)$, dove per una data immagine $\widehat{x}$ un valore di $A(\widehat{x})$ che supera la soglia ϕ rappresenta un anomalia.

5.2. Architettura

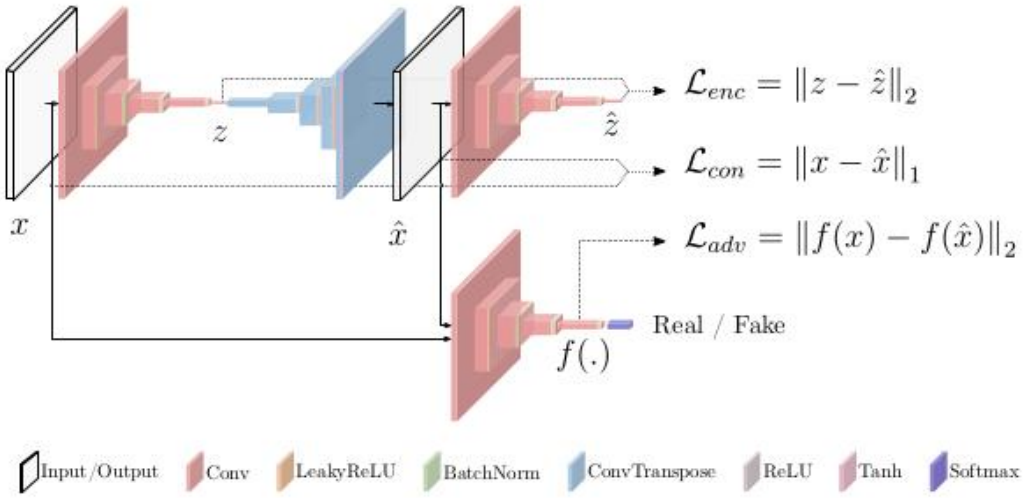

Figure 4. Architettura GANomaly

Come mostrato in figura (Figure 4), GANomaly prevede l'impiego di due encoder, un decoder e un discriminatore che vanno a comporre un sistema di tre sottoreti.

- La prima sottorete è caratterizzata da un autoencoder (un encoder più un decoder) che si comportano da generatore del modello GAN. Il generatore G infatti, legge le immagini in input x e le invia all'encoder G_E il quale le ridimensiona comprimendo x in un vettore z. Quest'ultimo viene definito come *bottleneck features* di G e si ipotizza che abbia la capacità di racchiudere le *features* rilevanti di x mantenendo però una dimensione ridotta. Il decoder G_D adotta l'architettura Deep Convolutional GAN [8] ed ha il compito di ridimensionare z alle dimensioni originali di x e così ricostruire l'immagine $\hat{x}$.
- La seconda sottorete è caratterizzata da un encoder E che comprime $\hat{x}$ in $\hat{z}$ con una diversa parametrizzazione rispetto a G_E. La dimensione di $\hat{z}$ coincide con la dimensione di z.
- La terza sottorete è caratterizzata dal discriminatore D il quale al compito di identificare le immagini ricevute in input rispettivamente come reali o generate.

Costruendo l'architettura in questo modo, e dopo aver addestrato adeguatamente la rete, si ipotizza che quando un'immagine contenente una anomalia viene analizzata dalla prima sottorete, G_D non sia in grado di ricostruire le anomalie contenute nell'immagine. Il motivo è che la sottorete G è stata addestrata utilizzando soltanto immagini normali, di conseguenza la sua parametrizzazione non le consente di ricostruire le anomalie. Inoltre, un output $\hat{x}$ simile a x ma con le anomalie mancanti, porterà alla creazione di un vettore $\hat{z}$ da parte di E in cui mancherà la rappresentazione delle *features* collegate alle anomalie, creando una divergenza tra $\hat{z}$ e z. Attraverso queste divergenze è possibile identificare come anomala un'immagine in input, definendo l'*anomaly score* come $A(x) = \| G_E(x) - E(G(x)) \|_1$. Le loss function utilizzate nel processo di backpropagation delle reti sono tre, ognuna delle quali è associata ad una specifica sottorete, rispettivamente:

- $L_{cond} = E_{x \sim px} \| x - G(x) \|_1$ per la sottorete G, *reconstruction loss*
- $L_{enc} = E_{x \sim px} \| G_E(x) - E(G(x)) \|_2$ per la sottorete E, *encoder loss*
- $L_{adv} = E_{x \sim px} \| f(x) - E_{x \sim px} f(G(x)) \|_2$ per la sottorete D, *adversarial loss*

5.3. Dettagli implementativi

I test effettuati con GANomaly sono stati eseguiti partendo non dall'implementazione ufficiale del paper [1], la quale utilizza PyTorch come framework, ma la sua controparte realizzata con TensorFlow2 [4].

Al fine di adattare la suddetta implementazione ai dati a disposizione si sono svolte due principali operazioni preliminari: pre-processing delle immagini e riadattamento delle reti, le quali sono strutturate per gestire immagini di piccole dimensioni relative a dataset di test come Mnist o CIFAR. Per quanto riguarda il pre-processing, le immagini sono state convertite da bmp a png, tagliate e ridotte ad una dimensione inferiore. Ridurre la dimensionalità delle immagini è fondamentale per evitare l'esaurimento di memoria che può verificarsi quando le reti sono particolarmente profonde come nel caso di GANomaly. I dettagli relativi

a G_E, G_D, E e D sono riportati di seguito e prendono in considerazione immagini di dimensioni 256×256 pixel e un vettore z di dimensione 100.

Layer (type)	Output shape
conv2d (Conv2D)	(None, 256, 256, 64)
batch_normalization (BatchNorm)	(None, 256, 256, 64)
leaky_re_lu (leakyReLU)	(None, 256, 256, 64)
conv2d_1 (Conv2D)	(None, 128, 128, 128)
batch_normalization_1 (BatchNorm)	(None, 128, 128, 128)
leaky_re_lu_1 (leakyReLU)	(None, 128, 128, 128)
conv2d_2 (Conv2D)	(None, 64, 64, 256)
batch_normalization_2 (BatchNorm)	(None, 64, 64, 256)
leaky_re_lu_2 (leakyReLU)	(None, 64, 64, 256)
conv2d_3 (Conv2D)	(None, 32, 32, 512)
batch_normalization_3 (BatchNorm)	(None, 32, 32, 512)
leaky_re_lu_3 (leakyReLU)	(None, 32, 32, 512)
conv2d_4 (Conv2D)	(None, 16, 16, 1024)
batch_normalization_4 (BatchNorm)	(None, 16, 16, 1024)
leaky_re_lu_4 (leakyReLU)	(None, 16, 16, 1024)
conv2d_5 (Conv2D)	(None, 8, 8, 2048)
batch_normalization_5 (BatchNorm)	(None, 8, 8, 2048)
leaky_re_lu_5 (leakyReLU)	(None, 8, 8, 2048)
conv2d_6 (Conv2D)	(None, 1, 1, 100)
Params	53103360

Figure 5. GANomaly $G_E - D$

Layer (type)	Output shape
conv2d_transpose (Conv2DTranspose)	(None, 1, 1, 100)
batch_normalization (BatchNorm)	(None, 1, 1, 100)
re_lu (ReLU)	(None, 1, 1, 100)
conv2d_transpose_1 (Conv2DTranspose)	(None, 8, 8, 2048)
batch_normalization_1 (BatchNorm)	(None, 8, 8, 2048)
re_lu_1 (ReLU)	(None, 8, 8, 2048)
conv2d_transpose_2 (Conv2DTranspose)	(None, 16, 16, 256)
batch_normalization_2 (BatchNorm)	(None, 16, 16, 256)
re_lu_2 (ReLU)	(None, 16, 16, 256)
conv2d_transpose_3 (Conv2DTranspose)	(None, 32, 32, 512)
batch_normalization_3 (BatchNorm)	(None, 32, 32, 512)
re_lu_3 (ReLU)	(None, 32, 32, 512)
conv2d_transpose_4 (Conv2DTranspose)	(None, 64, 64, 256)
batch_normalization_4 (BatchNorm)	(None, 64, 64, 256)
re_lu_4 (ReLU)	(None, 64, 64, 256)
conv2d_transpose_5 (Conv2DTranspose)	(None, 128, 128, 128)
batch_normalization_5 (BatchNorm)	(None, 128, 128, 128)
re_lu_5 (ReLU)	(None, 128, 128, 128))
conv2d_transpose_6 (Conv2DTranspose)	(None, 256, 256, 64)
Params	53103360

Figure 6. GANomaly G_D

Layer (type)	Output shape
conv2d (Conv2D)	(None, 256, 256, 64)
batch_normalization (BatchNorm)	(None, 256, 256, 64)
leaky_re_lu (leakyReLU)	(None, 256, 256, 64)
conv2d_1 (Conv2D)	(None, 128, 128, 128)
batch_normalization_1 (BatchNorm)	(None, 128, 128, 128)
leaky_re_lu_1 (leakyReLU)	(None, 128, 128, 128)
conv2d_2 (Conv2D)	(None, 64, 64, 256)
batch_normalization_2 (BatchNorm)	(None, 64, 64, 256)
leaky_re_lu_2 (leakyReLU)	(None, 64, 64, 256)
conv2d_3 (Conv2D)	(None, 32, 32, 512)
batch_normalization_3 (BatchNorm)	(None, 32, 32, 512)
leaky_re_lu_3 (leakyReLU)	(None, 32, 32, 512)
conv2d_4 (Conv2D)	(None, 16, 16, 1024)
batch_normalization_4 (BatchNorm)	(None, 16, 16, 1024)
leaky_re_lu_4 (leakyReLU)	(None, 16, 16, 1024)
conv2d_5 (Conv2D)	(None, 8, 8, 2048)
batch_normalization_5 (BatchNorm)	(None, 8, 8, 2048)
leaky_re_lu_5 (leakyReLU)	(None, 8, 8, 2048)
conv2d_6 (Conv2D)	(None, 1, 1, 1)
activation_sigmoid (Sigmoid)	(None, 1, 1, 1)
Params	44747265

Figure 7. GANomaly D

Con questa configurazione (Figure 6, Figure 7) l'intera architettura di GANomaly presenta un numero di parametri pari a 204,057,345 di cui 31,872 non addestrabili. L'addestramento delle reti, solitamente di durata pari a 30 epoche, è stato effettuato considerando batch di 32 immagini e utilizzando Adam come ottimizzatore con i seguenti parametri:

- learning rate = 0.0002
- beta1 = 0.5
- beta2 = 0.999

Ogni loss è stata pesata facendo riferimento ai valori descritti nel paper [1] (1 per l'adversarial e l'encoder loss e 50 per la reconstruction) che sperimentalmente forniscono i migliori risultati.

Nonostante il numero delle epoche possa sembrare riduttivo è stato verificato che, a causa della ristrettezza del dataset, con un numero maggiore di epoche non si ottengono miglioramenti né in termini di riduzione di loss né di efficacia del modello.

5.4. Risultati

I risultati di GANomaly sono espressi in termini di AUROC (o *auc*), cioè Area Under the Receiver Operating Characteristic Curve e in termini di AUPRC (o *average precision*), cioè Area Under the Precision-Recall Curve.

AUROC indica quanto un modello sia in grado di classificare correttamente i dati (positivi e negativi), e rappresenta la probabilità che un dato positivo, casualmente selezionato, venga classificato come positivo rispetto a un dato negativo. L'AUROC è calcolato come l'area sottesa alla curva che misura il trade off tra tasso di veri positivi (True Positive Rate) e tasso di falsi positivi (False Positive Rate) a diversi valori di soglia. AUPRC indica la capacità di un modello di identificare tutti i dati positivi senza classificare accidentalmente troppi dati negativi come positivi. L'AUPRC è calcolato come l'area sottesa alla curva che misura il trade off tra la precision e il recall per diversi valori di soglia. Risultati soddisfacenti sono solitamente associati ad alti valori di AUROC e AURPC.

Al fine di definire una baseline relativa alla capacità di GANomaly di eseguire anomaly detection sui dati, la rete è stata addestrata in diverse istanze incrociando i principali parametri (mantenendo fissi quelli descritti nella sezione precedente) e utilizzando come immagini sia quelle complete che le singole sottoparti. I parametri presi in considerazione sono la dimensione del vettore z, la cui lunghezza può variare da 32 a 1024 elementi, e la dimensione di input delle immagini, la quale definisce la profondità e le dimensione delle reti che le elaboreranno. L'insieme di dati di test preso in considerazione è formato in modo bilanciato da dati normali e anomali, così da essere il più rappresentativo possibile.

5.4.1. Immagini complete

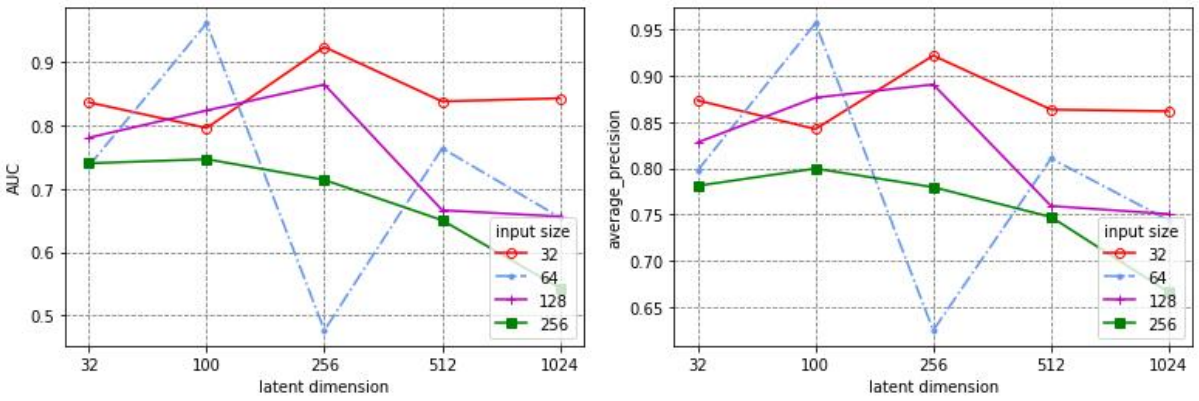

Figure 8. GANomaly con immagini complete

I test effettuati sulle immagini complete (Figure 8), raccolti effettuando una media su più esecuzioni, mostrano l'efficacia di GANomaly secondo le varie parametrizzazioni. È possibile notare come i risultati, in termini di auc e average precision siano promettenti nonostante la complessità e la poca quantità di immagini a disposizione. Di seguito sono riportati i risultati dell'addestramento di una delle parametrizzazioni più promettenti: input_size = 128 e latent_dimension = 256 (Figure 9).

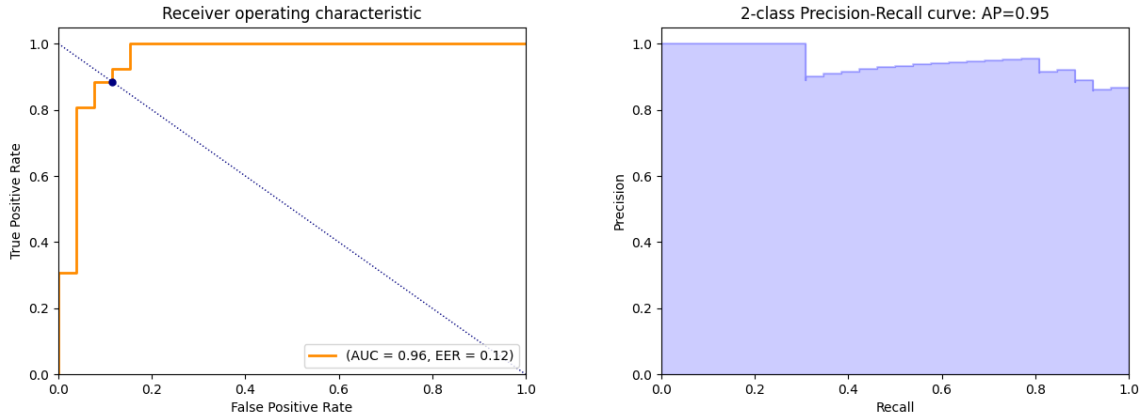

Figure 9. GANomaly con immagini complete: auc e average precision

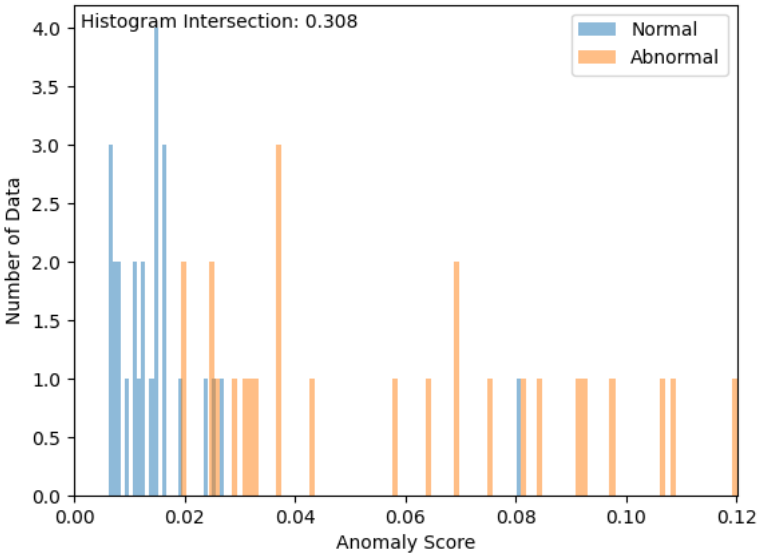

Figure 10. GANomaly con immagini complete: istogramma anomaly score

L'istogramma (Figure 10) mostra come l'anomaly score, scegliendo la soglia corretta, consente una buona distinzione fra le immagini normali e anomale facenti parte dell'insieme di test.

Di nostro interesse però è anche la capacità di individuare gli elementi che caratterizzano le anomalie, così da poter effettivamente verificare che le immagini vengano designate come anomale per il giusto motivo. A questo proposito un metodo possibile è quello di confrontare le immagini originali e le rispettive ricostruzioni di G, così da mostrare cosa è andato perduto in fase di ricostruzione quando i dati attraversano la prima sottorete.

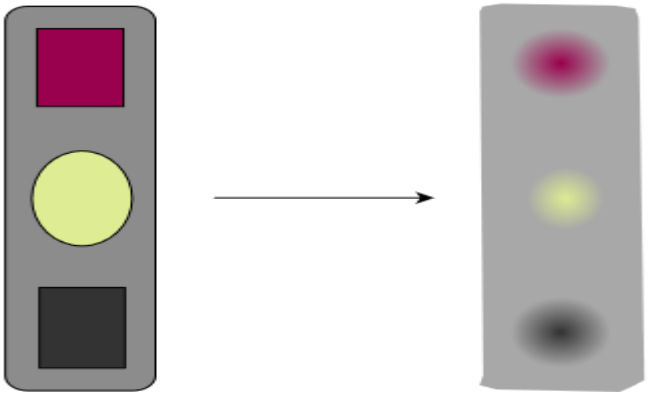

Figure 11. GANomaly ricostruzione immagine completa

La figura (Figure 11) mostra come la povertà della ricostruzione, probabilmente dovuta alla complessità delle immagini, non consente di identificare le piccole anomalie che le caratterizzano. Per questa ragione si è deciso di applicare lo stesso

approccio ma considerando le singole sottoparti, anodo e catodo, così da ridurre la complessità delle immagini da ricostruire. Non viene preso in considerazione il vetrino in quanto le immagini con anomalie nella zona del vetrino sono praticamente nulle, il che rende i test effettuabili praticamente irrilevanti.

5.4.2. Anodo

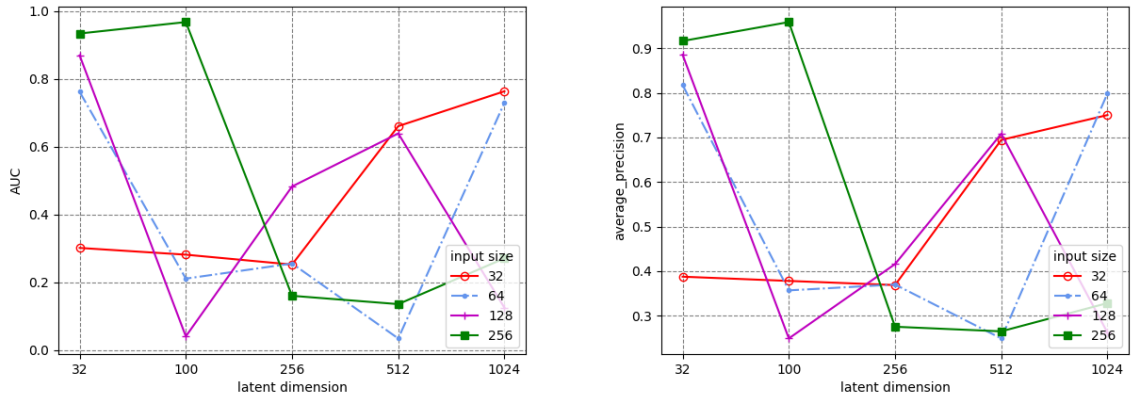

Figure 12. GANomaly con immagini anodo

I risultati ottenuti con l'anodo (Figure 12) mostrano un calo nel rendimento del modello, questo è probabilmente dovuto al fatto che le immagini complete racchiudono complessivamente un maggior numero di anomalie rispetto alle sottoparti. Questo si può tradurre in una difficoltà maggiore di individuazione nel caso di immagini caratterizzate da poche e piccole anomalie. I risultati sono comunque buoni considerando la migliore parametrizzazione: input_size = 256 e latent_dimension = 100 (Figure 13).

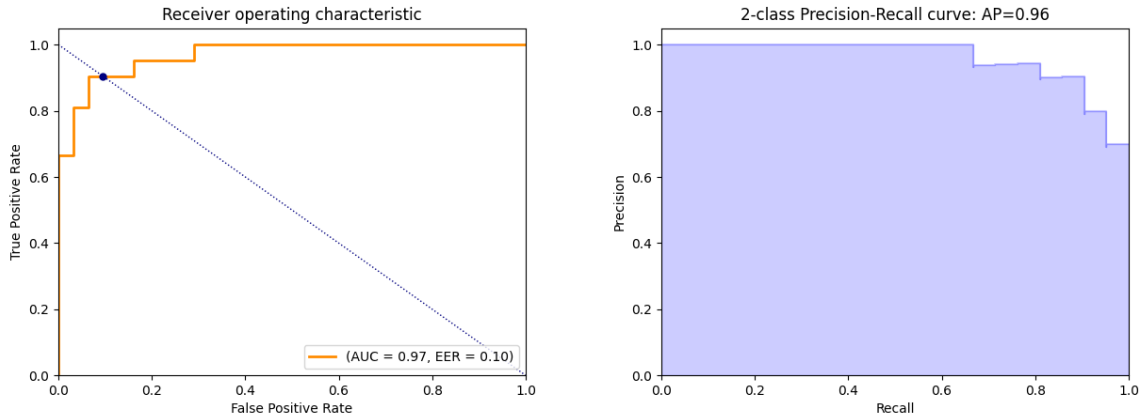

Figure 13. GANomaly con immagini anodo: auc e average precision

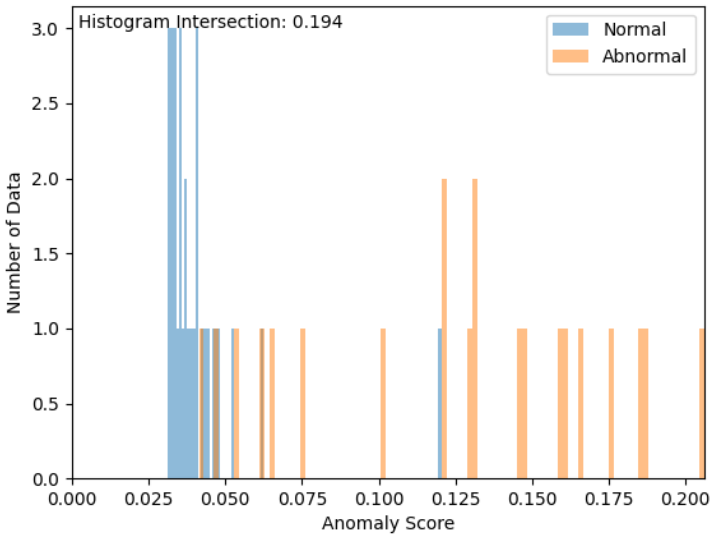

Figure 14. GANomaly con immagini anodo: istogramma anomaly score

Nonostante le immagini ridotte in termini di dimensione e complessità, la ricostruzione (Figure 15) rimane insufficiente all'identificazione delle anomalie.

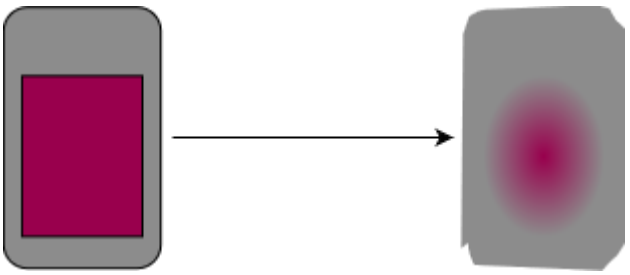

Figure 15. GANomaly ricostruzione anodo

5.4.3. Catodo

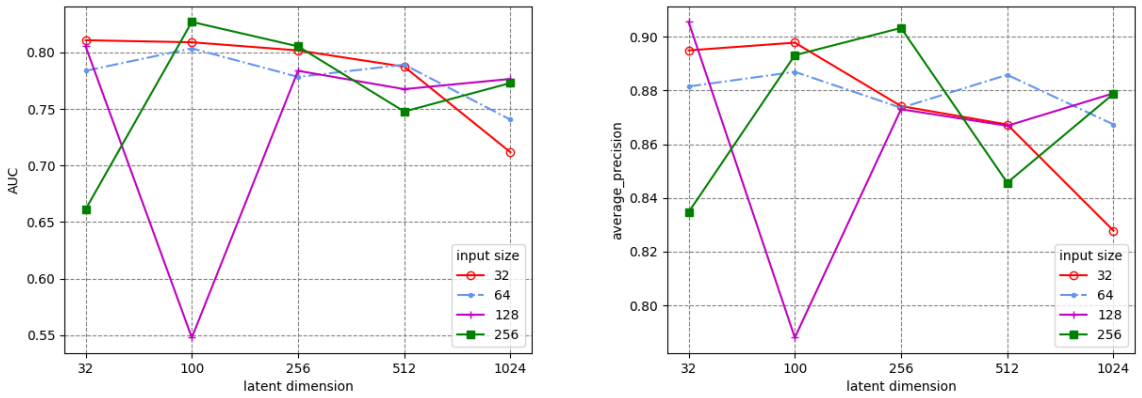

Figure 16. GANomaly con immagini catodo

Per il catodo valgono le stesse considerazioni dell'anodo. Il miglioramento generale delle performance del catodo (Figure 16) dipende probabilmente dal maggiore tasso di anomalie che ne rende più semplice l'individuazione. Di seguito sono riportati i risultati ottenuti prendendo in considerazione la stessa parametrizzazione vista dell'anodo (Figure 17).

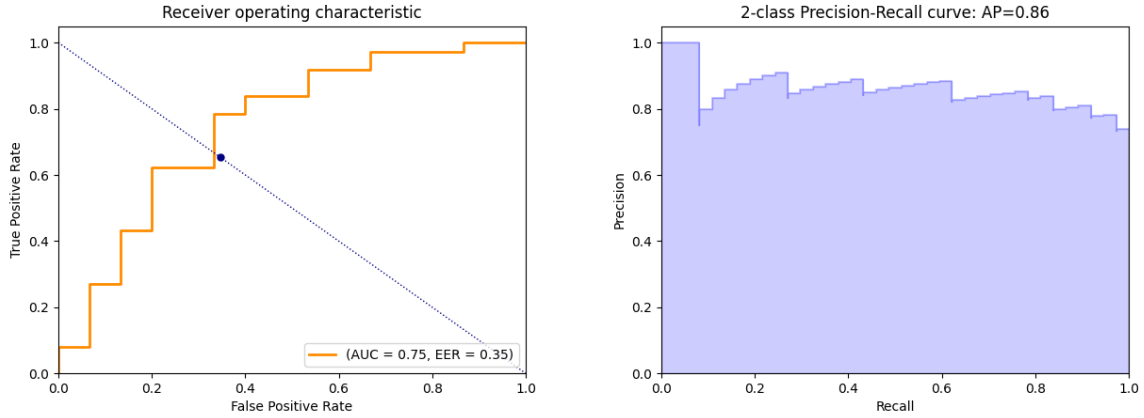

Figure 17. GANomaly con immagini catodo: auc e average precision

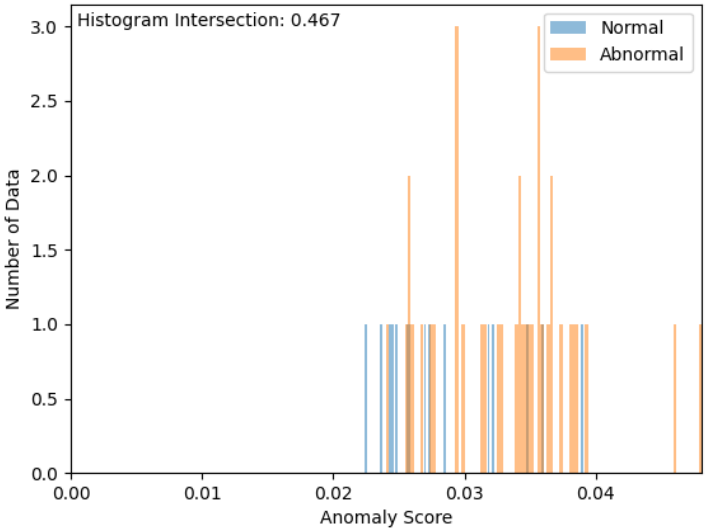

Figure 18. GANomaly con immagini catodo: istogramma anomaly score

Anche in questo caso la ricostruzione (Figure 19) non è sufficiente a distinguere le possibili anomalie.

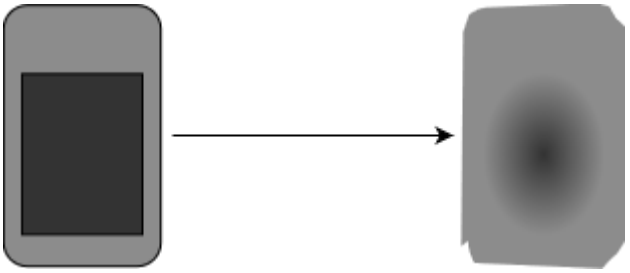

Figure 19. GANomaly ricostruzione catodo

6. Autoencoders

Un importante modello, usato sia in ambito machine/deep learning che in modelli generativi, è sicuramente l'autoencoder [7]. Gli autoencoders sono reti neurali in grado di imparare la miglior codifica dei dati che vengono forniti in input. Un autoencoder è formato da due componenti, un encoder E e un decoder D. L'encoder riceve in input i dati x e li mappa in un sottospazio ridotto chiamato *latent space*, dove il dato di input si trasforma in ciò che viene definito *latent representation* $z = E(x)$. Il decoder invece svolge un ruolo diametralmente opposto: partendo dalla rappresentazione ridotta di x, cioè z, tenta di ricostruire il dato originale ottenendo così $\hat{x} = D(E(x)) = D(z)$. L'addestramento delle due reti avviene attraverso un processo di backpropagation basato sull'errore calcolato confrontando i dati di input x con la rispettiva ricostruzione $\hat{x}$ (Figure 20).

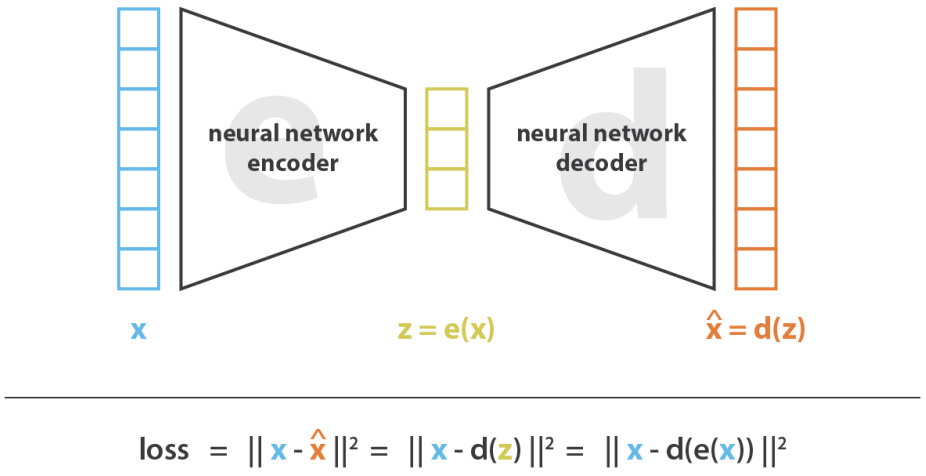

Figure 20. Autoencoder

Ad oggi, riduzione del rumore e riduzione della dimensionalità per la visualizzazione dei dati, sono considerate tra le applicazioni più interessanti degli autoencoders. Inoltre negli ultimi anni essi, insieme alle GAN, rappresentano le due tecniche più avanguardistiche con cui eseguire anomaly detection. Gli autoencoders infatti sono alla base dei convolutional variational autoencoders e degli adversarial variational autoencoders, due possibili modelli di rete neurale per effettuare anomaly detection.

7. Convolutional Variational Autoencoders

7.1. Limitazioni degli autoencoders

Abbiamo visto nel capitolo precedente come gli autoencoders siano strumenti molto utili nell'ambito della riduzione della dimensionalità, i quali possono ottenere risultati anche migliori rispetto ad approcci più classici come la PCA. Prendendo in considerazione invece l'ambito generativo è lecito chiedersi come questo modello possa essere utilizzato per generare nuovi dati e come possa rivelarsi utile nel caso della anomaly detection.

Visto che un autoencoder risulta essere molto simile al generatore di un modello GAN, possiamo pensare che, prendendo arbitrariamente un punto qualsiasi nel latent space e inviandolo come input al decoder, sia effettivamente possibile generare nuovi dati (Figure 21).

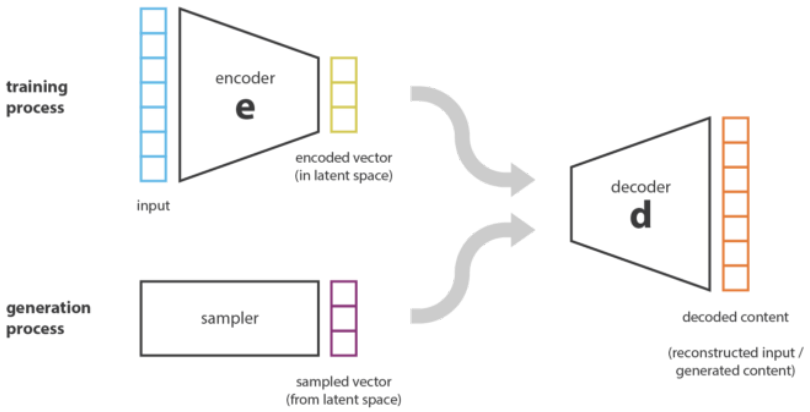

Figure 21. Decoder come generatore

Questo però non è del tutto vero in quanto nel momento in cui si effettua la riduzione di dimensionalità, mantenendo comunque una buona capacità di ricostruzione, si va a ridurre la capacità di interpretare e decodificare il latent space. Questo processo prende il nome di perdita di regolarità del latent space [7]. Infatti, se un autoencoder viene addestrato su un insieme di dati N con la giusta parametrizzazione può essere in grado di ricostruire con estrema precisione i dati forniti in input. Questo può essere dovuto ad un alto tasso di overfitting, che consente quindi al autoencoder, partendo da specifiche latent representation, di ricostruire i dati originali senza errori. Questo però porta ad una incapacità dell'autoencoder di decodificare latent representations che si discostano anche

lievemente da quelle viste in fase di training, il che implica che molti punti del latent space risultino praticamente senza significato una volta utilizzati dal decoder (Figure 22). Il tentativo quindi di generare nuovi dati, dando in pasto punti casuali del latent space al decoder, risulta del tutto inutile a causa della mancanza di regolarizzazione.

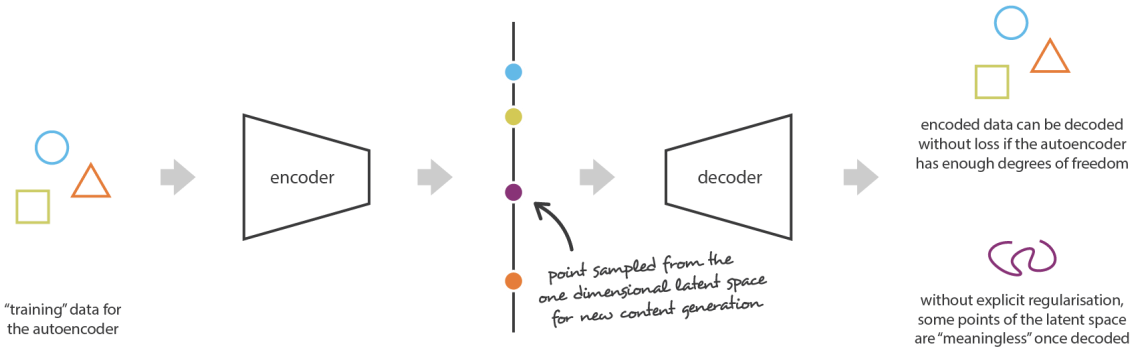

Figure 22. Mancanza di regolarizzazione

Per ovviare a questo problema è necessario strutturare il processo per far sì che non ci sia come unico obiettivo quello di ridurre al minimo l'errore di ricostruzione, ma anche quello di fornire un metodo di regolarizzazione del latent space.

7.2. Variational Autoencoders

Si possono definire variational autoencoders quegli autoencoders il cui addestramento è regolarizzato così da evitare overfitting e assicurarsi una miglior organizzazione del latent space. Esattamente come i normali autoencoders i variational autoencoders sono caratterizzati da un encoder e un decoder e vengono addestrati al fine di ridurre il più possibile l'errore di ricostruzione. Oltre a questo vengono introdotti dei cambiamenti necessari alla regolarizzazione del latent space: il dato in input dell'encoder non viene più mappato in un punto (latent representation), bensì in una distribuzione di probabilità nel latent space (Figure 23). Così facendo l'errore di ricostruzione viene calcolato tra il dato di input x e la ricostruzione $D(z)$, dove z non è più la rappresentazione di x mappata nel latent space, bensì il dato campionato dalla distribuzione di probabilità mappata da x nel latent space.

Figure 23. VAE

Un'altra sostanziale differenza è l'errore di ricostruzione. Anche quest'ultimo deve essere aggiornato per far fronte al nuovo sistema di codifica, che passa da essere un singolo punto ad una distribuzione. La loss viene quindi ampliata con un termine che aiuta la regolarizzazione del latent space (Figure 24). Lo scopo è quello di far assomigliare il più possibile la distribuzione risultante dall'encoder a una distribuzione normale standard. Questo termine prende il nome di divergenza di Kullback-Leibler, e rappresenta una misura della differenza tra due distribuzioni di probabilità.

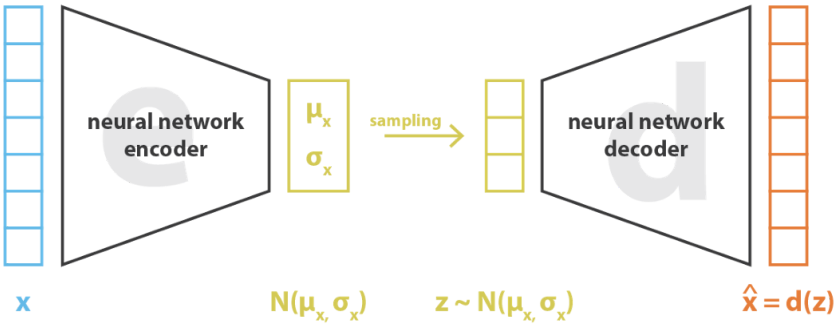

Figure 24. VAE + KL

Grazie all'introduzione di questo termine è possibile garantire al latent space una sorta di "gradiente" associato alle informazioni codificate. La seguente immagine (Figure 25) mostra intuitivamente la differenza tra un latent space non regolarizzato e uno regolarizzato.

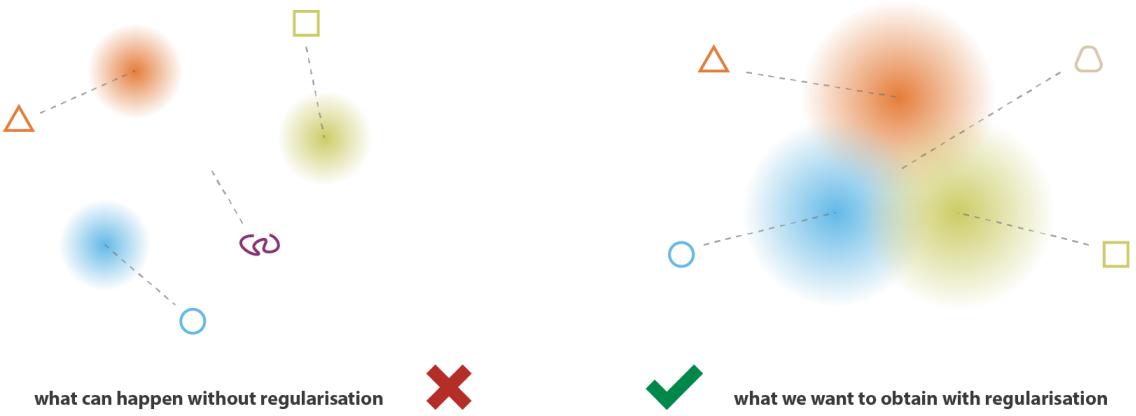

Figure 25. Effetti della regolarizzazione

7.3. VAE e anomaly detection

Le caratteristiche dei variational autoencoders sono tali da poter essere impiegate per risolvere problemi di anomaly detection. Prendiamo in considerazione lo stesso problema già illustrato per GANomaly (capitolo 5.1). Addestrando la rete esclusivamente con i dati del dataset D si ottiene un modello il cui latent space è in grado di replicare la distribuzione di probabilità rappresentante i dati normali. A questo punto basterà dare in input al modello il dataset $\widehat{D}$ e calcolare l'errore, espresso in termini di errore quadratico medio, che intercorre tra il dato fornito come input x e la sua ricostruzione a seguito della mappatura nel latent space $\hat{x}$. Questo valore può essere utilizzato in maniera analoga al anomaly score definito da GANomaly per distinguere dati anomali da dati normali.

7.4. Dettagli implementativi

I test eseguiti processando le immagini con i variational autoencoders sono stati effettuati partendo da un'implementazione realizzata per effettuare anomaly detection sul dataset Mnist [12]. Come nel caso di GANomaly, è stato necessario ridefinire la rete per consentirgli di lavorare con le immagini del nostro dataset. In questo caso, considerando l'anomaly score espresso come errore quadratico medio tra l'immagine di input e la sua ricostruzione, si è deciso di utilizzare come dimensione delle immagini 256×256 pixel, ossia il massimo che la rete riesce

a gestire senza terminare la memoria a disposizione. Di seguito sono riportate le caratteristiche dell'encoder e del decoder che formano il variational autoencoder.

Layer (type)	Output shape
conv2d (Conv2D)	(None, 256, 256, 64)
elu (ELU)	(None, 256, 256, 64)
max_pooling (MaxPooling2D)	(None, 128, 128, 64)
conv2d_1 (Conv2D)	(None, 128, 128, 128)
elu_1 (ELU)	(None, 128, 128, 128)
max_pooling_1 (MaxPooling2D)	(None, 64, 64, 128)
conv2d_2 (Conv2D)	(None, 64, 64, 256)
elu_2 (ELU)	(None, 64, 64, 256)
max_pooling_2 (MaxPooling2D)	(None, 32, 32, 256)
conv2d_3 (Conv2D)	(None, 32, 32, 512)
elu_3 (ELU)	(None, 32, 32, 512)
reshape (Reshape)	(None, 524288)
dense (Dense)	(None, 512)
dense_1 (Dense)	(None, 256)
Mean value (μ)	
dense_2 (Dense)	(None, 128)
Variance value (σ)	
dense_3 (Dense)	(None, 128)
Params	273252672

Figure 26. CVAE E

Come si evince dalla struttura dell'encoder *E* (Figure 26) il vettore *z* preso in considerazione ha una dimensione pari a 256. Il vettore viene poi diviso dalla rete in due diversi vettori di dimensione 128. Questo perché, a differenza dei normali autoencoders, il risultato della codifica sono i valori di media (μ) e varianza (σ) che caratterizzano la distribuzione.

Layer (type)	Output shape
dense (Dense)	(None, 512)
dense_1 (Dense)	(None, 524288)
reshape (Reshape)	(None, 32, 32, 512)
conv2d (Conv2D)	(None, 32, 32, 512)
elu (ELU)	(None, 32, 32, 512)
conv2d_1 (Conv2D)	(None, 32, 32, 512)
elu_1 (ELU)	(None, 32, 32, 512)
conv2d_transpose (Conv2DTranspose)	(None, 64, 64, 256)
elu_2 (ELU)	(None, 64, 64, 256)
conv2d_3 (Conv2D)	(None, 64, 64, 256)
elu_3 (ELU)	(None, 64, 64, 256)
conv2d_transpose_1 (Conv2DTranspose)	(None, 128, 128, 128)
elu_4 (ELU)	(None, 128, 128, 128)
conv2d_4 (Conv2D)	(None, 128, 128, 128)
elu_5 (ELU)	(None, 128, 128, 128)
conv2d_transpose_2 (Conv2DTranspose)	(None, 256, 256, 64)
elu_6 (ELU)	(None, 256, 256, 64)
conv2d_5 (Conv2D)	(None, 256, 256, 64)
elu_7 (ELU)	(None, 256, 256, 64)
conv2d_3 (Conv2D)	(None, 256, 256, 3)
sigmoid (Sigmoid)	(None, 256, 256, 3)
Params	276070467

Figure 27. CVAE D

Il decoder (Figure 27) non è del tutto simmetrico all'encoder in quanto come input riceve direttamente il *sample* ottenuto dalla distribuzione descritta dai valori di media e varianza risultanti dall'encoder.

Con questa configurazione l'architettura presenta un numero di parametri pari a 549,323,139. L'addestramento delle reti, solitamente di durata pari a 100 epoche, è stato effettuato considerando batch di 32 immagini e utilizzando Adam come ottimizzatore con i seguenti parametri:

- learning rate = 0.0001
- beta1 = 0.9
- beta2 = 0.999

7.5. Risultati

I risultati relativi ai variational autoencoders sono espressi, come nel caso di GANomaly, in termini di auc, average precision e capacità di identificare le anomalie grazie alla ricostruzione delle immagini. Quest'ultima è mostrata dall'istogramma che prende in considerazione l'insieme di dati di test mostrando per ogni elemento l'errore quadratico medio.

7.5.1. Immagini complete

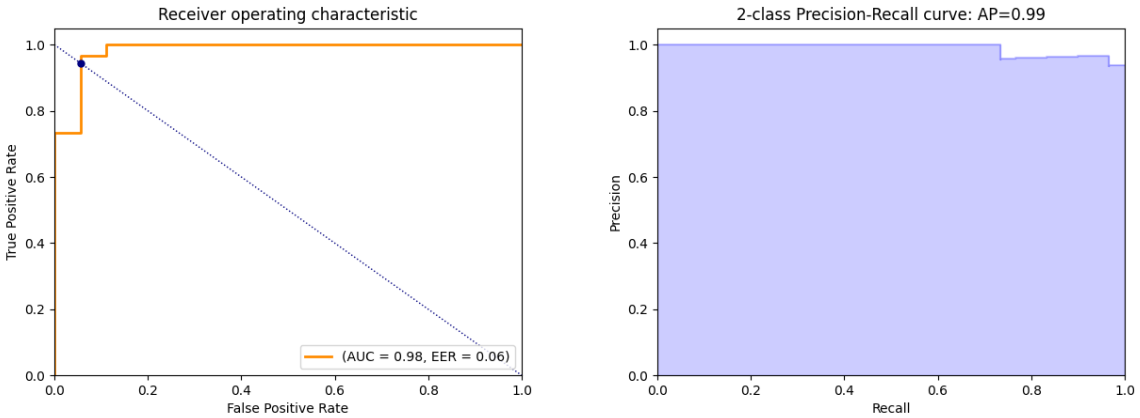

Figure 28. VAE con immagini complete: auc e average precision

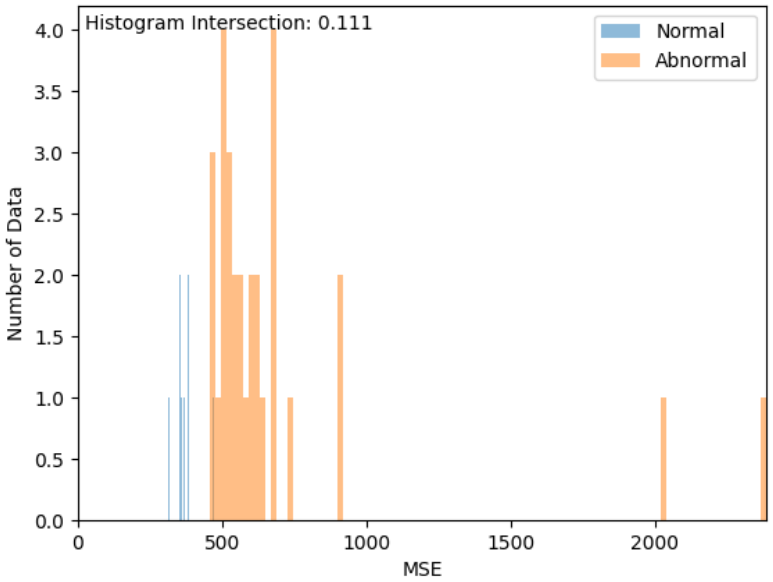

Figure 29. VAE con immagini complete: istogramma MSE

I risultati ottenuti (Figure 28) in termini di auc e average precision mostrano le ottime prestazioni del modello. Anche guardando l'istogramma è possibile notare che, scegliendo il giusto valore di soglia, è possibile eseguire una buona distinzione tra i dati normali e anomali.

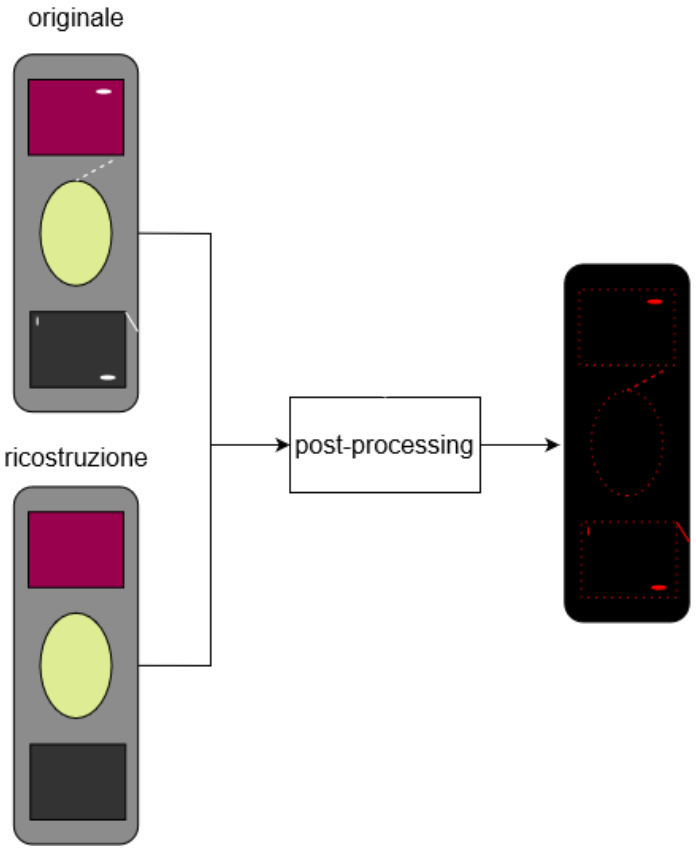

Figure 30. VAE ricostruzione immagine completa

Il modello mostra decisamente una maggiore capacità ricostruttiva rispetto a GANomaly (Figure 30). In questo modo è possibile applicare delle semplici operazioni di post-processing per evidenziare le differenze tra le immagini originali e le relative ricostruzioni. Le operazioni svolte sono:

- Differenza tra x e $\hat{x}$ elevata al quadrato
- Normalizzazione del risultato tra 0 e 255
- Applicazione di una colormap per esaltare il risultato

Volendo analizzare più nel dettaglio le immagini si è proseguito eseguendo gli stessi test sulle singole sottoparti.

7.5.2. Anodo

Analogamente alle immagini complete anche l'anodo mostra buoni risultati in termini di auc e average precision, rispettivamente 0.95 e 0.94. Inoltre considerando che come nel caso delle immagini complete la dimensione delle è fissa a 256×256 pixel è possibile vedere ancora più nel dettaglio le differenze tra x e $\hat{x}$ (Figure 31).

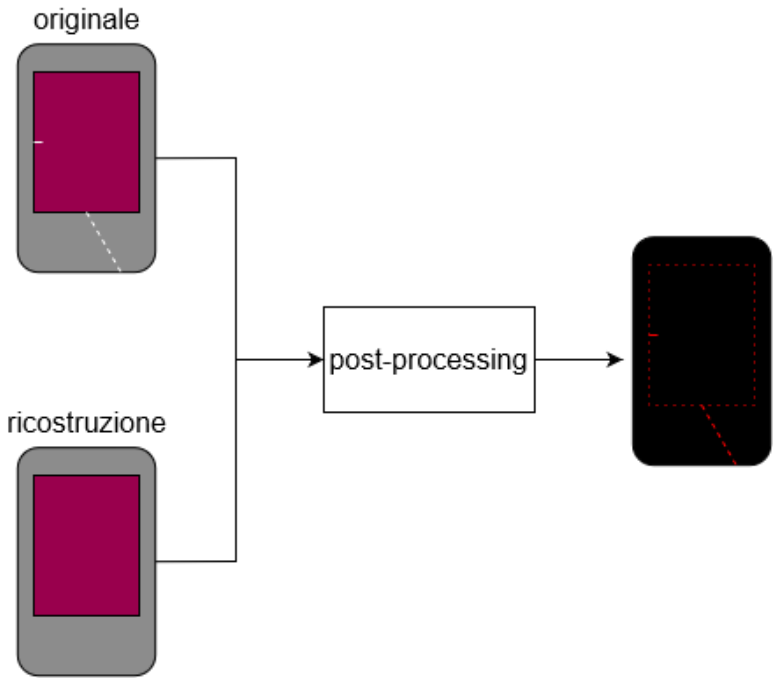

Figure 31. VAE ricostruzione immagine anodo

7.5.3. Catodo

Nel caso del catodo si ha qualche calo di performance in termini di auc e average precision. Le quali scendono a 0.75 e 0.87. Nonostante ciò, il livello di ricostruzione del modello risulta essere comunque in linea con anodo e immagini complete (Figure 32).

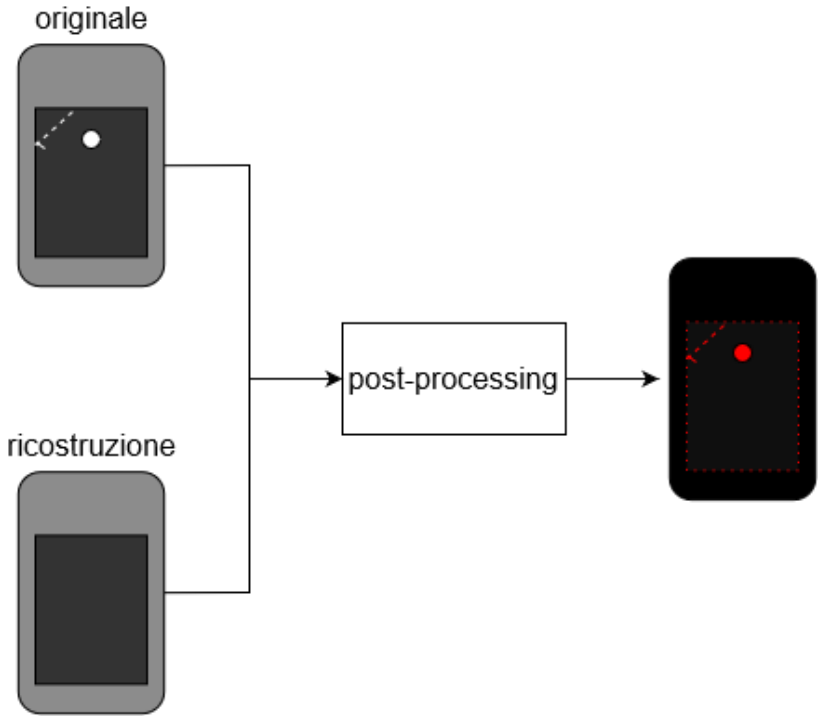

Figure 32. VAE ricostruzione immagine catodo

8. Adversarial Variational Autoencoders

Gli adversarial variational autoencoders, o adVAE [9], rappresentano l'unione tra gli autoencoders e il modello di addestramento avversario introdotto dalle GAN. La grande differenza introdotta da adVAE è la regolarizzazione del latent space attraverso il sistema di loss avversarie al posto della divergenza di Kullback-Leibler (Figure 33).

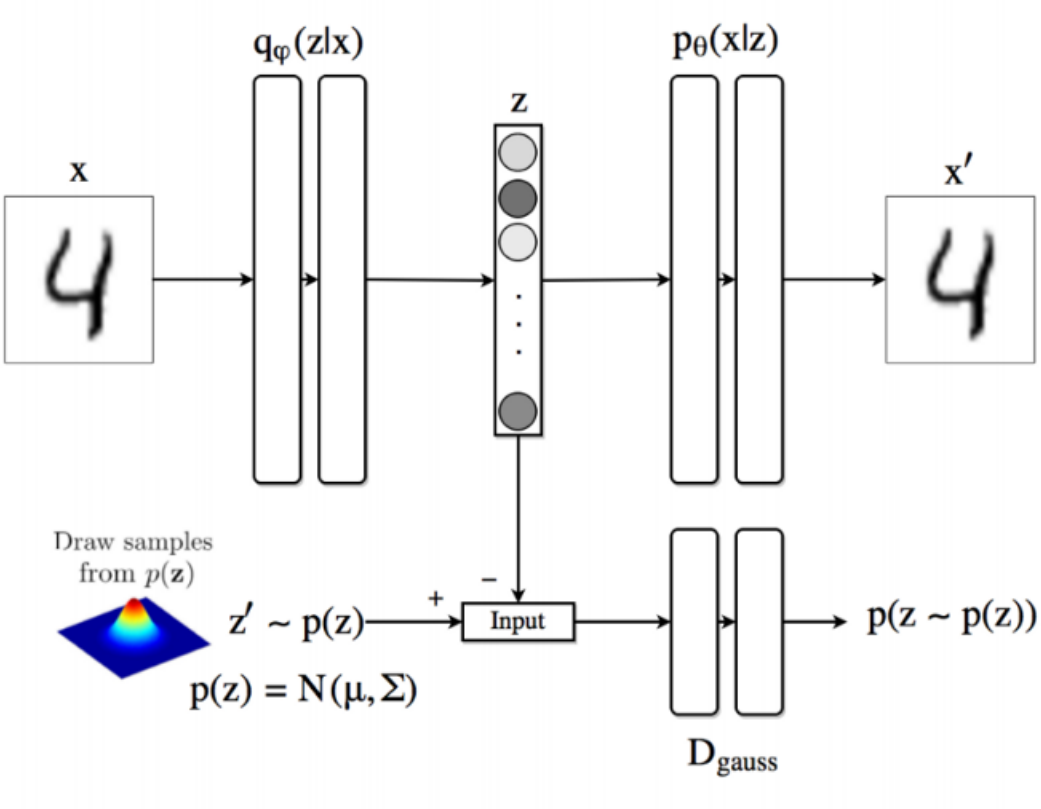

Figure 33. Architettura adVAE

Inoltre, a differenza delle normali GAN il discriminatore in questo caso riceve come input non delle immagini, bensì la rappresentazione dell'immagine nel latent space. Questo significa che il discriminatore dovrà essere in grado di discernere vettori *z* campionati dalla distribuzione reale dei dati e i vettori *z'* campionati dalla distribuzione generata.

8.1. Problemi dei modelli generativi per anomaly detection

L'utilizzo di adVAE nel contesto di anomaly detection è stato introdotto per far fronte a due problematiche relative ai metodi basati sui modelli generativi:

1. I modelli generativi mirano soltanto a ricreare la distribuzione di probabilità che più di tutte si avvicina a quella dei dati normali. Il che è

indirettamente molto utile per distinguere i dati normali da quelli anomali, però così facendo la rete non impara l'individuazione delle anomalie in senso stretto.

2. La versione classica di VAE utilizza la divergenza di Kullback-Leibler per limitare la capacità dell'encoder, ma nei generatori classici (stile GAN) non vi è presenza di regolarizzazione. Questo perché teoricamente un generatore di adeguata "grandezza", essendo una rete neurale in grado di fungere da approssimatore di funzioni, dovrebbe essere in grado di coprire ogni possibile distribuzione di probabilità senza l'ausilio delle latent variables. Attraverso degli esperimenti è stato dimostrato che questo può portare a dei problemi. Infatti, se un generatore può modellare una distribuzione di probabilità $P_{data}(x)$ senza utilizzare le informazioni relative alla rappresentazione nel latent space z, sarà più incline a farlo. Questo comportamento può verificarsi anche nel caso di VAE. Infatti, al fine di ridurre il costo della divergenza di KL, l'encoder tende a perdere molte informazioni riguardo x e mappa $P_{data}(x)$ semplicemente come $P(z)$ invece che come $P(z|x)$. Questo causa l'overfitting di VAE rispetto alla distribuzione normale dei dati. Per evitare ciò la capacità dei VAE dovrebbe essere ulteriormente limitata da un tasso maggiore di regolarizzazione.

Le novità introdotte da adVAE con assunzione di probabilità Gaussiana [10] mirano a risolvere questi problemi.

8.2. adVAE con assunzione di probabilità Gaussiana

Il modello assume che la distribuzione di probabilità sia dei dati normali che dei dati anomali sia Gaussiana e che i valori di media e varianza siano diversi fra loro:

- Distribuzione dati normali = $N(\mu_1, \sigma_1)$
- Distribuzione dati anomali = $N(\mu_2, \sigma_2)$
- $\mu_1 \neq \mu_2 ; \sigma_1 \neq \sigma_2$

Ciò non esclude però che le distribuzioni possano sovrapporsi nel latent space. Per questa ragione al modello precedentemente descritto va aggiunto un trasformatore

Gaussiano denominato T. T è addestrato per sintetizzare vettori nel latent space z_t in modo che sembrino anomali ma simili ai normali z.

8.2.1. Architettura

L'architettura completa della rete ed il processo di addestramento è descritto in figura 34 (Figure 34).

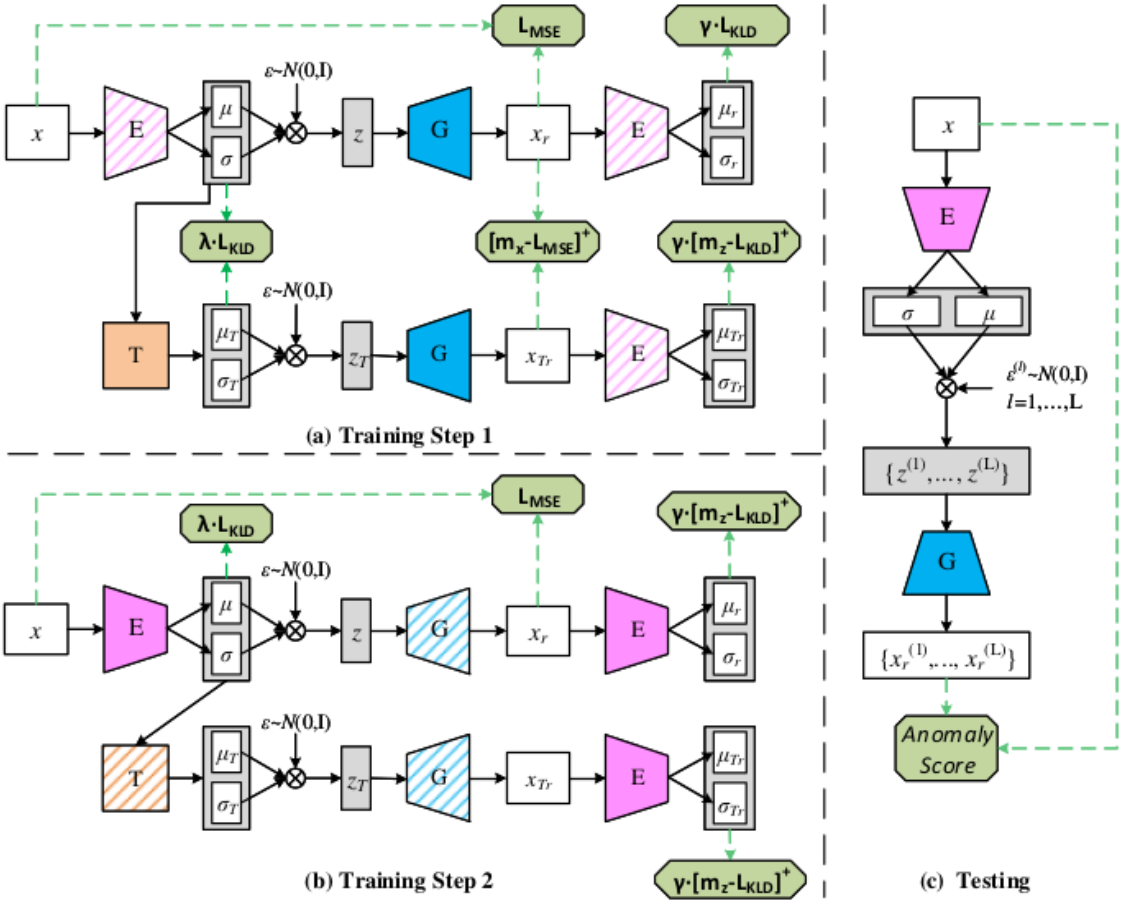

Figure 34. Architettura di adVAE con assunzione di probabilità Gaussiana

La struttura avversaria del modello può essere riassunta nei due seguenti punti:

- E è addestrato con l'obiettivo di discriminare i dati originali x e i dati ricostruiti x_r
- G è addestrato per distinguere i vettori z dai vettori z_t sintetizzati da T

In questo modo nella prima fase di addestramento (Figure 34.a) T cerca di ingannare G mentre quest'ultima funge da discriminatore, nella seconda (Figure 34.b) G genera dati realistici che E dovrà essere in grado di distinguere da quelle reali.

Con l'introduzione di questi obiettivi l'encoder E ed il decoder G acquisiscono la capacità di distinguere dati normali da dati anomali e viene incrementato il tasso di regolarizzazione del modello, così da prevenire l'overfitting.

8.2.2. Anomaly score

Come mostrato in figura 34.c (Figure 34.c) nella fase di test sono utilizzati solo l'encoder E ed il generatore G, analogamente a quanto accadeva in VAE. In questo caso però l'anomaly score è calcolato in maniera differente. Viene preso un dato di test x e fornito in input all'encoder E che effettuerà una stima dei valori delle variabili Gaussiane μ e σ. Una volta fatto ciò verranno campionati un insieme di L vettori $z = \{z^{(1)}, z^{(2)}, \dots, z^{(L)}\}$ dalla distribuzione $N(\mu, \sigma^2)$. Il generatore a questo punto riceve un vettore $z^{(l)}$ come input (dove $l = 1, 2, \dots, L$) e da in output la ricostruzione $x_r^{(l)}$. L'errore tra gli input x e la loro ricostruzione media $\sum_{l=1}^{L} x_r^{(l)}$ riflettete la divisione tra i dati normali e i dati anomali appresa dal modello. Per questa ragione l'anomaly score di un mini-batch di n dati è definito come:

$$S = e \cdot e^T = \left(x - \frac{1}{L}\sum_{l=1}^{L} x_r^{(l)}\right) \cdot \left(x - \frac{1}{L}\sum_{l=1}^{L} x_r^{(l)}\right)^T,$$

$s = \{S_{11}, S_{22}, \dots, S_{nn}\}$ rappresenta quindi l'anomaly score del mini-batch di n elementi. I dati che presentano un alto anomaly score sono categorizzati come anomali.

8.3. Dettagli implementativi

Come nel caso di GANomaly non è stata utilizzata l'implementazione ufficiale del paper, in quanto realizzata con PyTorch come framework, bensì la sua controparte realizzata con TensorFlow [11]. Anche questa volta, come in entrambi i casi precedenti, le reti sono state riadattate per l'elaborazione delle immagini del dataset.

Layer (type)	Output shape
conv2d (Conv2D)	(None, 256, 256, 32)
elu (ELU)	(None, 256, 256, 32)
max_pooling (MaxPooling2D)	(None, 128, 128, 32)
conv2d_1 (Conv2D)	(None, 128, 128, 64)
elu_1 (ELU)	(None, 128, 128, 64)
max_pooling_1 (MaxPooling2D)	(None, 64, 64, 64)
conv2d_2 (Conv2D)	(None, 64, 64, 128)
elu_2 (ELU)	(None, 64, 64, 128)
max_pooling_2 (MaxPooling2D)	(None, 32, 32, 128)
conv2d_3 (Conv2D)	(None, 32, 32, 256)
elu_3 (ELU)	(None, 32, 32, 256)
reshape (Reshape)	(None, 262144)
dense (Dense)	(None, 512)
dense_1 (Dense)	(None, 256)
Mean value (μ)	
dense_2 (Dense)	(None, 128)
Variance value (σ)	
dense_3 (Dense)	(None, 128)
Params	135521824

Figure 35. adVAE E

Layer (type)	Output shape
dense (Dense)	(None, 512)
dense_1 (Dense)	(None, 262144
reshape (Reshape)	(None, 32, 32, 256)
conv2d (Conv2D)	(None, 32, 32, 256)
elu (ELU)	(None, 32, 32, 256)
conv2d_1 (Conv2D)	(None, 32, 32, 256)
elu_1 (ELU)	(None, 32, 32, 256)
conv2d_transpose (Conv2DTranspose)	(None, 64, 64, 128)
elu_2 (ELU)	(None, 64, 64, 128)
conv2d_3 (Conv2D)	(None, 64, 64, 128)
elu_3 (ELU)	(None, 64, 64, 128)
conv2d_transpose_1 (Conv2DTranspose)	(None, 128, 128, 64)
elu_4 (ELU)	(None, 128, 128, 64)
conv2d_4 (Conv2D)	(None, 128, 128, 64)
elu_5 (ELU)	(None, 128, 128, 64)
conv2d_transpose_2 (Conv2DTranspose)	(None, 256, 256, 64)
elu_6 (ELU)	(None, 256, 256, 32)
conv2d_5 (Conv2D)	(None, 256, 256, 32)
elu_7 (ELU)	(None, 256, 256, 32)
conv2d_3 (Conv2D)	(None, 256, 256, 3)
sigmoid (Sigmoid)	(None, 256, 256, 3)
Params	136341027

Figure 36. adVAE G

Layer (type)	Output shape
dense (Dense)	(None, 256)
Params	33024

Figure 37. adVAE T

Con questa configurazione l'architettura presenta un numero di parametri pari a 271,862,851. L'addestramento delle reti, solitamente di durata pari a 100 epoche, è stato effettuato considerando batch di 32 immagini e utilizzando Adam come ottimizzatore con i seguenti parametri:

- learning rate = 0.0001
- beta1 = 0.9
- beta2 = 0.999

8.4. Risultati ottenuti

I risultati ottenuti sono espressi considerando le stesse metriche dei casi precedenti, inoltre sono state prese in considerazioni le stesse immagini viste nel caso VAE per mostrare la capacità ricostruttiva del modello.

8.4.1. Immagini complete

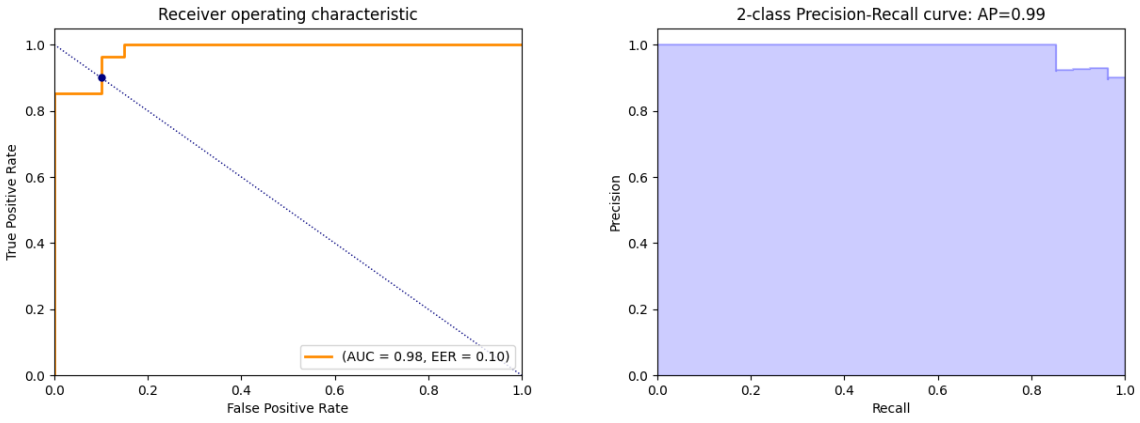

Figure 38. adVAE con immagini complete: auc e average precision

I test ottenuti con le immagini complete mostrano risultati che corrispondono a quelli ottenuti con VAE (Figure 38). Anche la qualità ricostruttiva del modello è buona anche se leggermente inferiore (Figure 40).

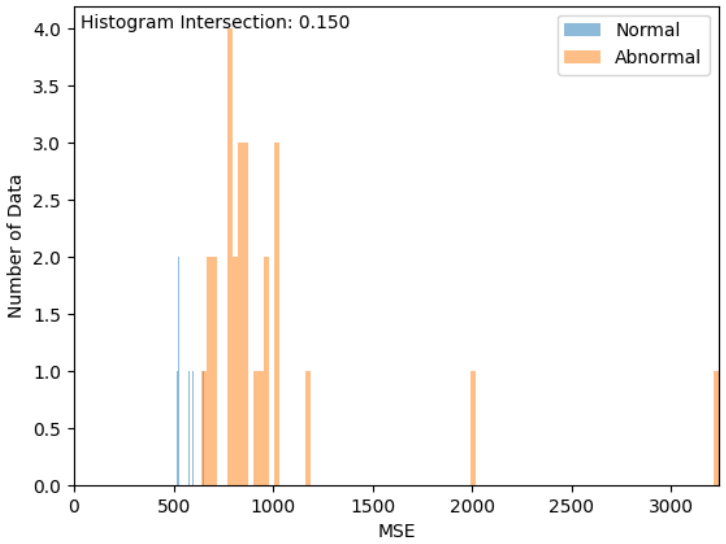

Figure 39. adVAE con immagini complete: istogramma MSE

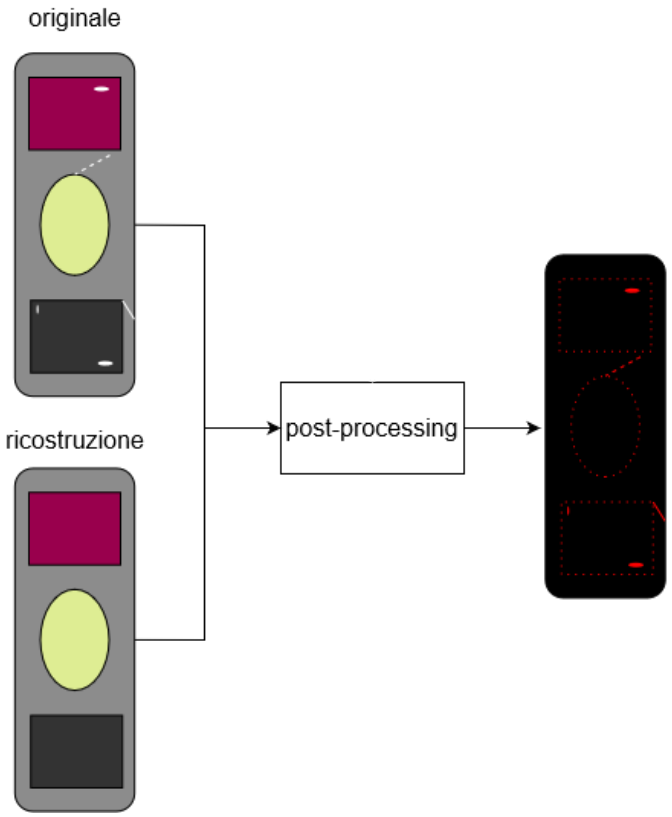

Figure 40. VAE ricostruzione immagine completa

8.4.2. Anodo

I risultati ottenuti da adVAE con le immagini relative all'anodo sono lievemente inferiori rispetto sia a quelli ottenuti con le immagini complete che a quelle ottenute con VAE. Il valore di auc si afferma a 0.92 e quello di average precision a 0.89.

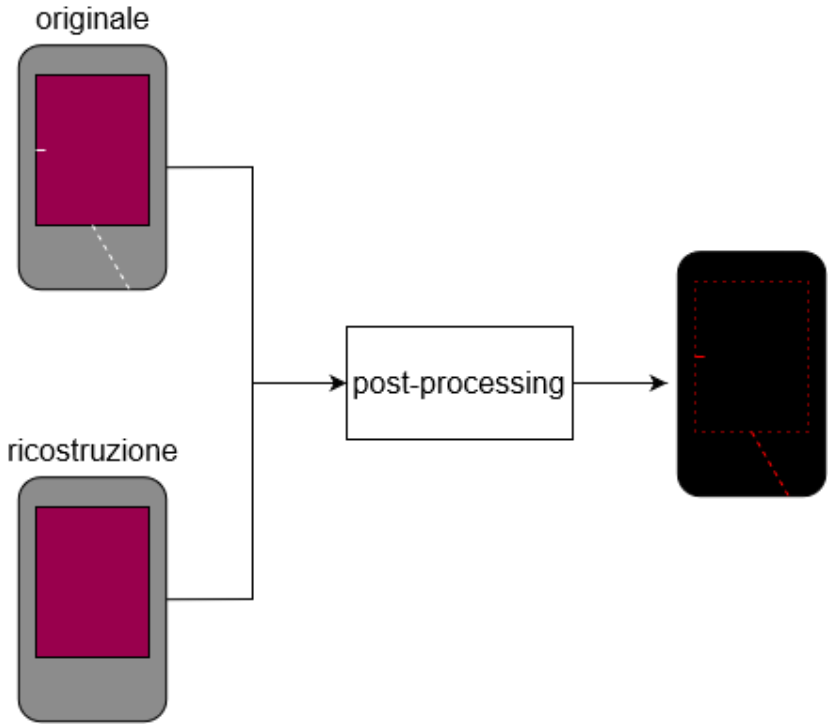

Figure 41. adVAE ricostruzione immagine anodo

8.4.3. Catodo

Nel caso del catodo il calo delle prestazioni è ancora più visibile. Il valore si riduce a 0.72 e quello di average precision a 0.83. Nonostante ciò la capacità ricostruttiva è ancora discreta (Figure 42).

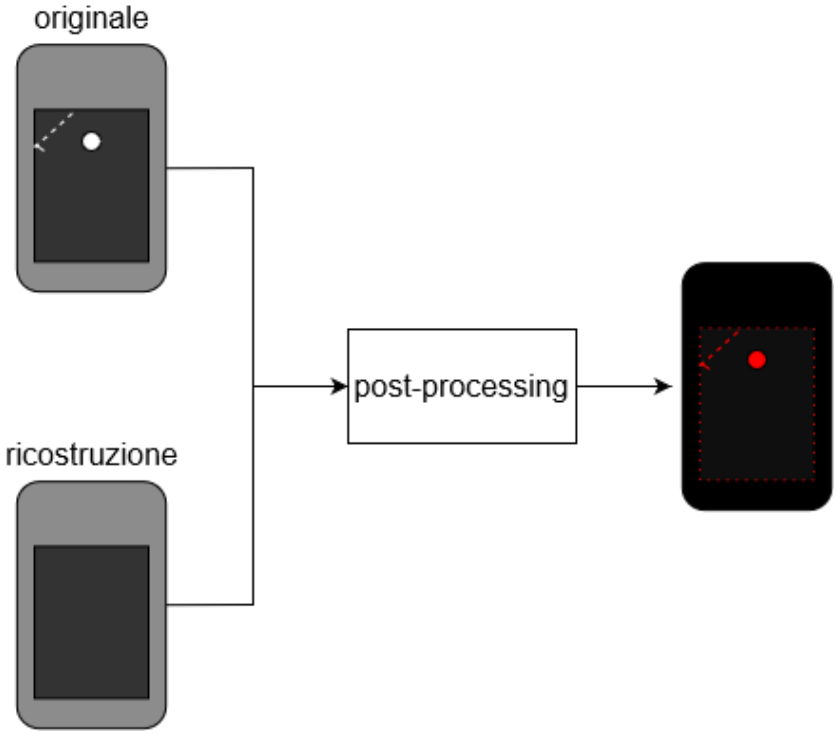

Figure 42. adVAE ricostruzione immagine catodo

9. Data Augmentation

Come già discusso nel capitolo 3, una delle più grandi limitazioni è sicuramente il quantitativo di immagini a disposizione. Nel tentativo di sopperire alla limitatezza dei dati, così da poter effettuare dei test ancora più consistenti di quelli riportati negli scorsi capitoli e ridurre il tasso di overfitting dei modelli, si è tentato di applicare operazioni di data augmentation. Con data augmentation ci si riferisce a quell'insieme di tecniche volte ad ampliare il dataset senza acquisire nuovi elementi. Esistono principalmente due tipi di data augmentation, la prima è basata sulle trasformazioni geometriche e la seconda basata sulla generazioni di dati sintetici attraverso metodi generativi, come le GAN [5].

9.1. Data Augmentation classica

Un primo approccio tramite metodi classici di data augmentation è stato realizzato tramite l'ausilio della libreria Albumentation [3], la quale mette a disposizione un'ampia gamma di trasformazioni. Il dataset delle immagini complete è stato esteso di circa 10,000 immagini applicando due diverse classi di trasformazioni, a livello spaziale e a livello di pixel. Come trasformazioni spaziali sono state prese in considerazione le seguenti:

- random scale
- rotate
- vertical flip
- horizontal flip

mentre invece, per quelle a livello di pixel abbiamo:

- blur (molto tenue)
- random brightness contrast (molto tenue)

Ogni trasformazione ha probabilità di essere applicata parti al 30%.

Il nuovo dataset creato è stato utilizzato per addestrare GANomaly così da verificare l'efficacia dell'approccio. Per rendere il test più consistente sono state verificate le varie parametrizzazioni.

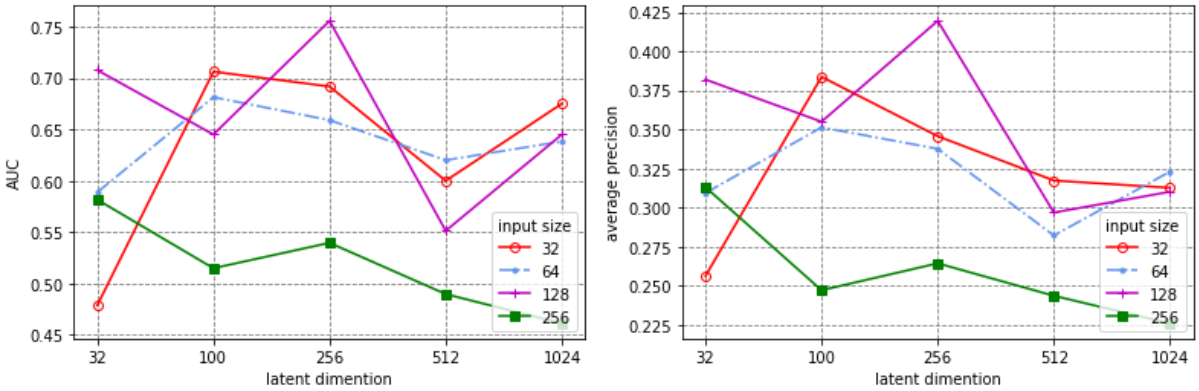

Figure 43. GANomaly con Albumentation

I risultati ottenuti (Figure 43) mostrano chiaramente il calo vertiginoso di performance dovuto alla data augmentation, sia in termini si auc che di average precision. Visti gli scarsi risultati, si è deciso di non estendere i test agli altri metodi di anomaly detection discussi nei capitoli precedenti.

9.2. Modelli generativi

Vista l'inefficacia delle tecniche di data augmentation classiche e considerando l'obiettivo finale, cioè anomaly detection su dati molto dettagliati e simili fra loro, si è scelto di provare con alcuni dei più comuni modelli generativi.

9.2.1. DCGAN

I modelli parametrici per la generazione di dati sintetici sono stati ampiamente studiati negli ultimi anni, nonostante ciò la generazione di immagini reali non ha riscosso un grande successo tranne che in tempi recenti. Molti dei modelli pionieri in questo ambito hanno ottenuto risultati interessanti ma sempre imprecisi, generando immagini sfocate, caratterizzate da rumore o del tutto incomprensibili (come nel caso delle GAN). Oltretutto scalare le GAN utilizzando l'architettura classica delle reti neurali convoluzionali non si è rivelata inizialmente una sfida facile per i ricercatori. Nonostante ciò le linee guida per la creazione di deep convolutional GAN (DCGAN) sono state definite dall'articolo "Unsupervised representational learning with deep convolutional GAN" [8]. La DCGAN per la data augmentation è stata implementata seguendo le linee guida descritte nell'articolo:

- Sostituire tutti i layer di max pooling con layer convoluzionali normali o trasposti
- Usare la batch normalization sia nel generatore che nel discriminatore
- Rimuovere i layer fully connected che non siano in input o output
- Uso della relu come funzione di attivazione nel generatore, tranne che per l'output
- Utilizzare la leaky relu come funzione di attivazione per tutti i layer del discriminatore

9.2.2. Wasserstein GAN-GP

Un altro modello che ha ottenuto buoni risultati in termini di generazione di immagini reali è noto come Wasserstein GAN [2]. Sempre basato sulla struttura di DCGAN, WGAN risolve alcune delle debolezze di quest'ultimo tramite l'ausilio di una diversa funzione di loss:

$$L = E_{\hat{x} \sim P_g}[D(\hat{x})] - E_{x \sim P_r}[D(x)]$$

Invece di utilizzare il discriminatore per classificare o predire la probabilità che le immagini ricevute in input siano reali o generate, WGAN lo sostituisce con un elemento denominato *critic* (Figure 44) che definisce quale sia la probabilità che l'immagine in analisi sia reale. Questo cambiamento è motivato da ragioni matematiche che indicano come più efficiente l'addestramento del modello basato sulla minimizzazione della distanza che c'è tra la distribuzione dei dati osservati e quella dei dati generati. La Wasserstein loss infatti è una funzione che approssima l'Earth Mover's Distance, ovvero il lavoro necessario per trasformare una distribuzione di probabilità in un'altra.

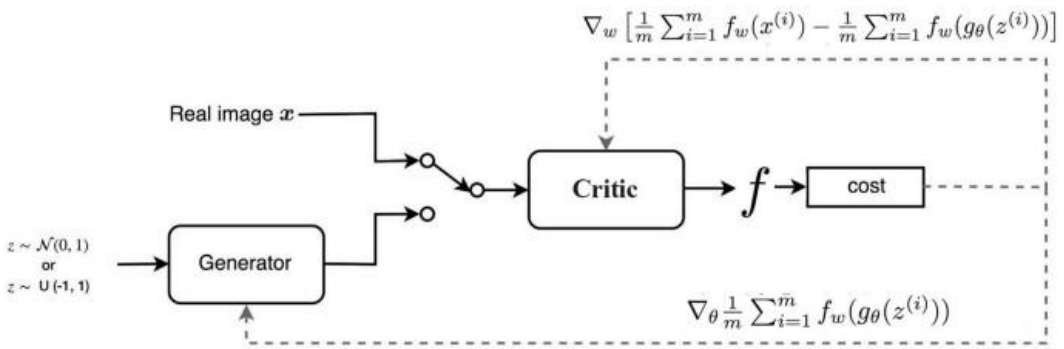

Figure 44. Architettura WGAN

Per poter utilizzare la Wasserstein loss è necessario che il discriminatore, o critic, sia 1-L continuo, ovvero la norma del gradiente deve essere al massimo 1 in ogni punto. Se questa condizione viene a mancare il modello rischia di generare immagini di bassa qualità o addirittura di non convergere. Ciò accade per esempio se si cerca di garantire la continuità attraverso il *gradient-clipping*. Per risolvere questi problemi e allo stesso tempo garantire la condizione di 1-L continuo, alla loss del discriminatore viene rimosso il gradient-clipping e aggiunto un termine alla loss denominato *gradient-penalty* [6].

$$L = \underset{\hat{x}\sim P_g}{E}[D(\hat{x})] - \underset{x\sim P_r}{E}[D(x)] + \lambda \underset{\hat{x}\sim P_{\hat{x}}}{E}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2]$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

loss iniziale — gradient-penality

9.3. Dettagli implementativi

Considerando la complessità delle immagini complete si è optato per la generazione delle singole sottoparti, quindi anodo vetrino e catodo. Come possibile notare dall'architettura delle reti, il generatore prende in input un vettore di 100 elementi e restituisce in output l'immagine di dimensione 256×256 pixel. Il discriminatore, analogamente a quanto già visto nel caso dei modelli basati sulle GAN, prende in input sia le immagini originali che quelle sintetiche e restituisce la classificazione.

9.3.1. DCGAN

Layer (type)	Output shape	Params #
dense (Dense)	(None, 7680)	768000
reshape (Reshape)	(None, 5, 3, 512)	0
conv2d_transpose (Conv2DTranspose)	(None, 25, 15, 512)	6554112
batch_normalization (BatchNorm)	(None, 25, 15, 512)	2048
conv2d_transpose_1 (Conv2DTranspose)	(None, 50, 30, 256)	3277056
batch_normalization_1 (BatchNorm)	(None, 50, 30, 256)	1024
conv2d_transpose_2 (Conv2DTranspose)	(None, 100, 60, 128)	819328
batch_normalization_2 (BatchNorm)	(None, 100, 60, 128)	512
conv2d_transpose_3 (Conv2DTranspose)	(None, 200, 120, 64)	204864
batch_normalization_3 (BatchNorm)	(None, 200, 120, 64)	256
conv2d_transpose_4 (Conv2DTranspose)	(None, 400, 240, 3)	4803

Figure 45. DCGAN discriminatore

Layer (type)	Output shape	Params #
conv2d (Conv2D)	(None, 400, 240, 64)	1792
batch_normalization (BatchNorm)	(None, 400, 240, 64)	256
leaky_re_lu (leakyReLU)	(None, 400, 240, 64)	0
conv2d_1 (Conv2D)	(None, 200, 120, 128)	73856
batch_normalization_1 (BatchNorm)	(None, 200, 120, 128)	512
leaky_re_lu_1 (leakyReLU)	(None, 200, 120, 128)	0
conv2d_2 (Conv2D)	(None, 100, 60, 256)	295168
batch_normalization_2 (BatchNorm)	(None, 100, 60, 256)	1024
leaky_re_lu_2 (leakyReLU)	(None, 100, 60, 256)	0
conv2d_3 (Conv2D)	(None, 50, 30, 512)	1180160
batch_normalization_3 (BatchNorm)	(None, 50, 30, 512)	2048
leaky_re_lu_3 (leakyReLU)	(None, 50, 30, 512)	0
flatten (Flatten)	(None, 768000)	0
dropout (Dropout)	(None, 768000)	0
dense (Dense)	(None, 1)	768001

Figure 46. DCGAN generatore

In fase di addestramento è stato utilizzato un learning rate pari a 0.002 e un momentum di 0.5 con Adam come ottimizzatore. Per ogni sottoparte sono state fatte alcune variazioni in termini di batch size, ripetizione di dataset e piccole modifiche alla rete, così da trovare i parametri più efficaci.

Nel tentativo di migliorare i risultati ottenuti da DCGAN è stato tentato anche un approccio parallelizzato: il modello è stato distribuito su 4 GPU e i parametri sono stati scalati tramite l'ausilio di Horovod.

9.3.2. WGAN-GP

L'implementazione della WGAN, a meno della loss e il sistema di gradient penality, risulta essere molto simile a quella della DCGAN. Infatti, il generatore resta pressoché invariato mentre il discriminatore introduce qualche cambiamento (Figure 47).

Layer (type)	Output shape	Params #
conv2d (Conv2D)	(None, 400, 240, 64)	1792
layer_normalization (LeakyNorm)	(None, 400, 240, 64)	256
leaky_re_lu (leakyReLU)	(None, 400, 240, 64)	0
conv2d_1 (Conv2D)	(None, 200, 120, 128)	73856
layer_normalization_1 (LeakyNorm)	(None, 200, 120, 128)	512
leaky_re_lu_1 (leakyReLU)	(None, 200, 120, 128)	0
conv2d_2 (Conv2D)	(None, 100, 60, 256)	295168
layer_normalization_2 (LeakyNorm)	(None, 100, 60, 256)	1024
leaky_re_lu_2 (leakyReLU)	(None, 100, 60, 256)	0
conv2d_3 (Conv2D)	(None, 50, 30, 512)	1180160
layer_normalization_3 (LeakyNorm)	(None, 50, 30, 512)	2048
leaky_re_lu_3 (leakyReLU)	(None, 50, 30, 512)	0
flatten (Flatten)	(None, 768000)	0
dense (Dense)	(None, 1)	768001

Figure 47. WGAN generatore

9.4. Risultati

Di seguito sono riportate una selezione di immagini ottenute con le tecniche sopra citate. Le immagini selezionate non rappresentano la media delle immagini ottenute dai modelli ma una selezione delle migliori, in particolare per il catodo. A esse sono affiancate i grafici rappresentati l'evoluzione della loss di generatore e discriminatore durante le 200 epoche di training eseguite. Nel caso dei dati relativi alla WGAN è riportato solo il discriminatore in quanto da solo definisce l'andamento di addestramento del modello.

9.4.1. Anodo

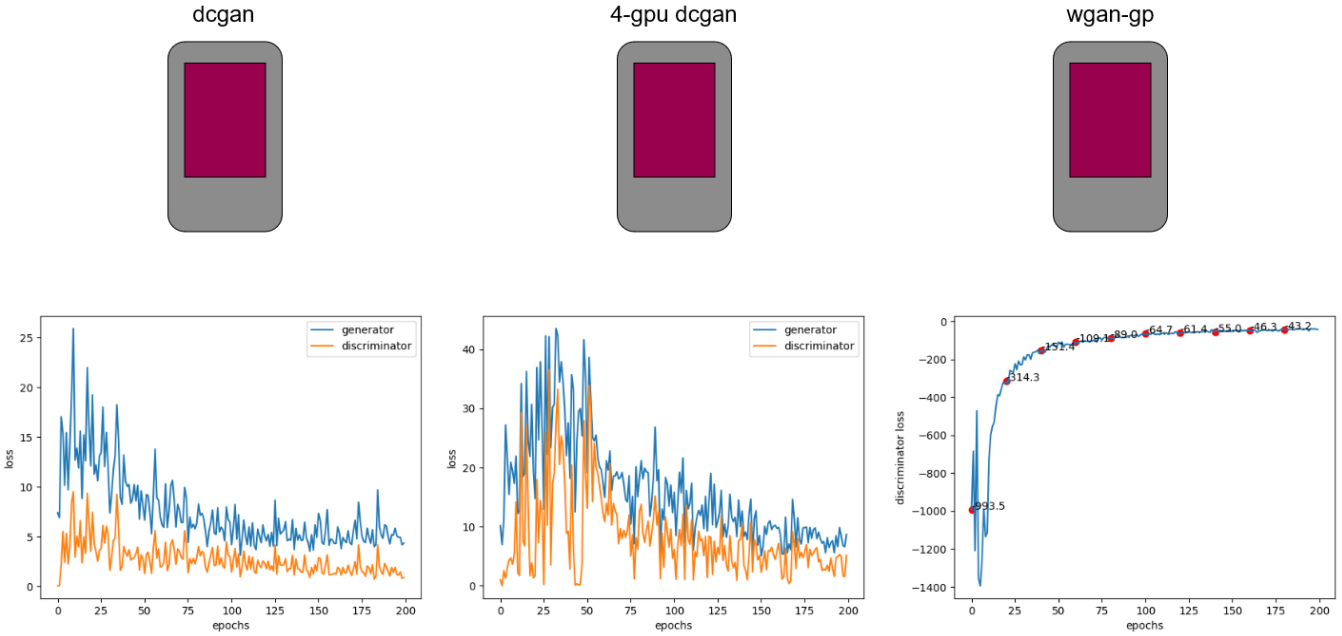

Figure 48. Generazione anodo

9.4.2. Vetrino

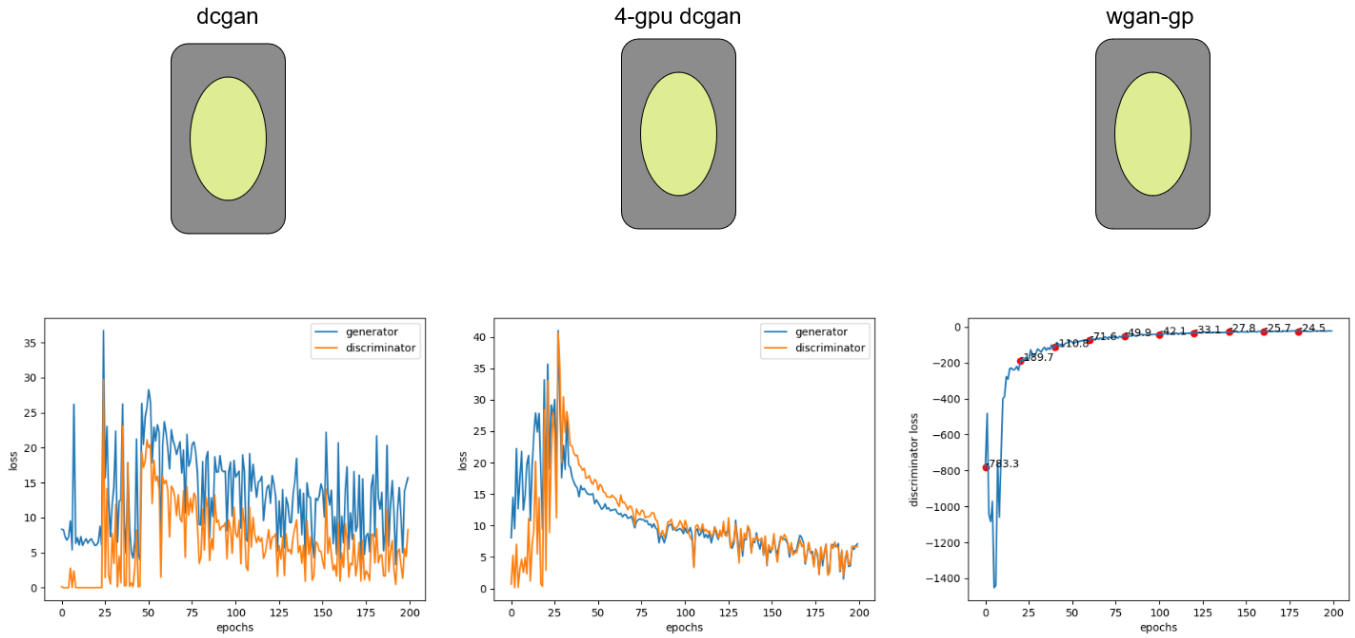

Figure 49. Generazione vetrino

9.4.3. Catodo

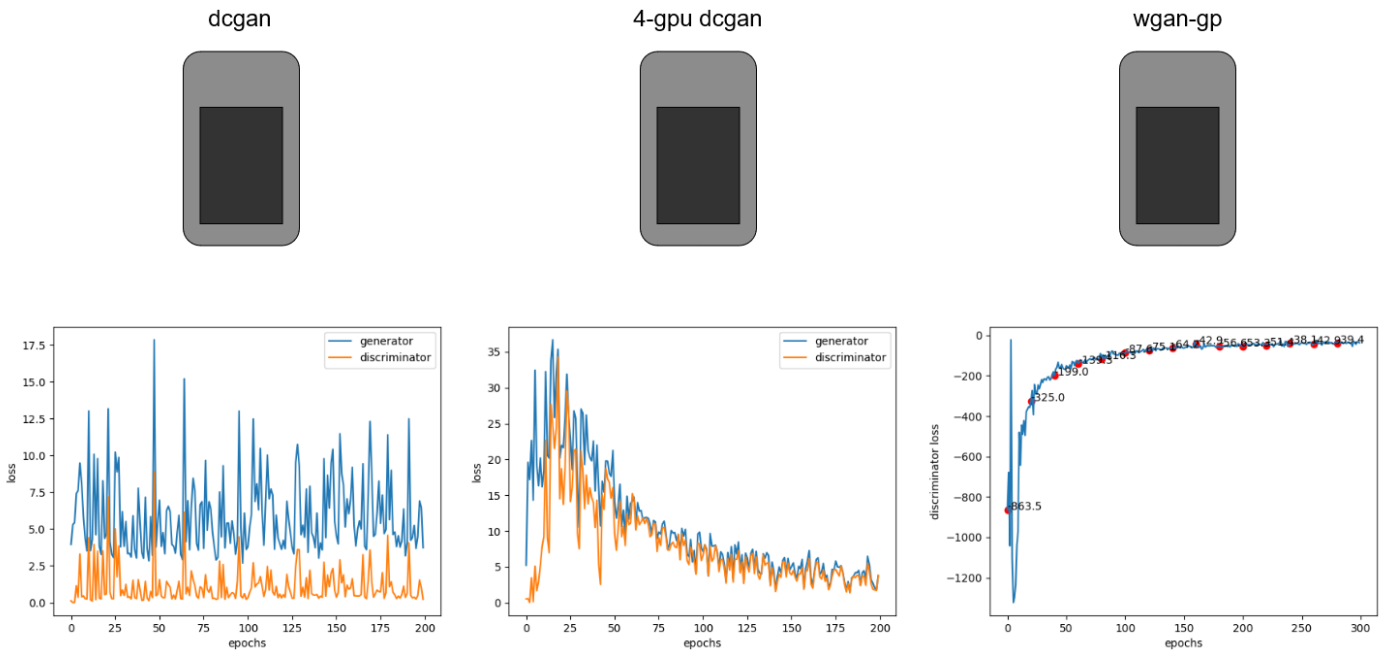

Figure 50. Generazione catodo

9.4.4. Immagini complete

Confrontando una piastrina artificiale ottenuta unendo le varie sottoparti, anodo (WGAN-GP) più vetrino (DCGAN) più catodo (WGAN-GP), con una piastrina reale si può notare complessivamente il risultato ottenuto da questi approcci (Figure 51). Qualitativamente le immagini ottenute risultano essere di buona qualità, ma nonostante ciò vi sono ancora piccole imprecisioni che potrebbero essere scambiate per difetti dalle tecniche di Anomaly Detection precedentemente illustrate.

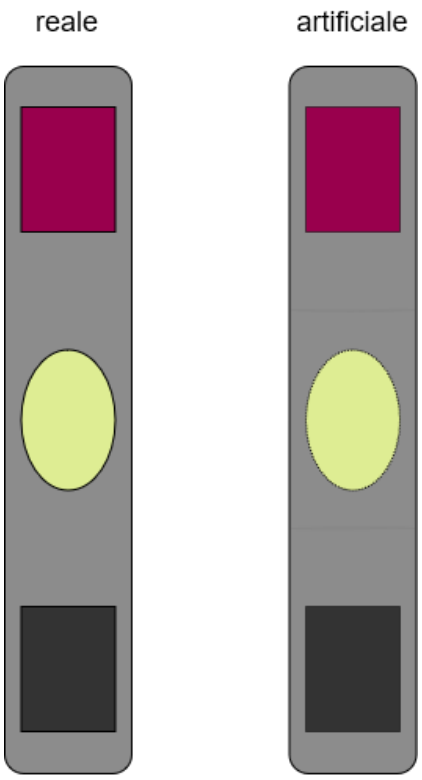

Figure 51. Confronto dati reali e artificiali

Tuttavia questi modelli potrebbero essere sicuramente impiegati con maggiore successo nel caso di immagini di minori dimensioni e meno complesse. In questo caso, considerando anche il numero esiguo di dati a disposizione per l'addestramento delle reti, la qualità raggiunta nella generazione non è sufficiente a ingannare i modelli di anomaly detection. Molto probabilmente la tipologia di anomalia che si sta analizzando (piccoli graffi, ammaccature, ecc.) è facilmente confondibile con errori di ricostruzione, il che porta a generare quasi unicamente immagini anomale. Inoltre la generazione delle immagini non è rapida e immediata. Per generare delle buone immagini è necessario compiere un lavoro di selezione non indifferente. Su 100 immagini artificiali di catodo o anodo, potendo addestrare la rete con così pochi dati, quelle abbastanza fedeli alle originali non sono solitamente più di 5. Ciò rende il processo lento, laborioso e difficile da automatizzare. Questo si ripercuote anche nelle immagini complete che sono l'unione delle 3 sottoparti singolarmente generate.

10. Conclusioni

Il lavoro svolto ha portato alla luce diverse considerazioni relativamente all'impiego di deep learning per l'identificazione di anomalie e data augmentation. Visti i risultati ottenuti dai vari modelli è possibile notare che, nonostante la complessità dei dati, le dimensioni delle anomalie e la scarsa cardinalità del dataset, con la giusta parametrizzazione e pre-processing delle immagini è possibile ottenere buoni risultati per quanto riguarda l'anomaly detection. La scelta del miglior modello da utilizzare è fortemente influenzata dal caso d'uso. In particolare gli approcci basati su autoencoders risultano essere più funzionali nel momento in cui siamo interessati ad ottenere una buona ricostruzione delle immagini e la successiva individuazione delle anomalie all'interno di esse. Nel caso in cui la quantità e qualità delle immagini a disposizione è tale da consentire l'esecuzione di test affidabili, anche gli approcci basati su GAN come GANomaly risultano essere un'ottima scelta, in quanto i risultati ottenuti in termini di classificazione sono molto promettenti.

D'altro canto la data augmentation, e in particolare la generazione di immagini artificiali, si sono rivelate inefficaci per due principali ragioni: la prima è il tempo necessario selezionare un numero cospicuo di immagini generate correttamente dalla rete, mentre la seconda è relativa al modo in cui queste immagini vengono impiegate. L'utilizzo di immagini generate per addestrare modelli di anomaly detection, che impiegano reti convoluzionali profonde per l'analisi delle immagini, si scontra pesantemente con le piccole imprecisioni che le caratterizzano. Nonostante ciò modelli come WGAN hanno mostrato di essere in grado di sintetizzare dati complessi anche partendo da un dataset tutt'altro che massiccio. In condizioni più favorevoli avrebbero potuto essere una risorsa valida per l'ampliamento dei dati.

11. Bibliografia

[1] Akcay, Samet, Amir Atapour-Abarghouei, and Toby P. Breckon. 2018. "GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training." (May).

[2] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. "Wasserstein GAN." (Jan).

[3] Buslaev, Alexander, Alex Parinov, Vladimir Iglovikov, and ... n.d. Albumentations: fast and flexible image augmentations. https://albumentations.ai/.

[4] "chychen/tf2-ganomaly: Tensorflow2 implementation of the paper GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training." n.d. GitHub. https://github.com/chychen/tf2-ganomaly.

[5] Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, and ... 2014. "Generative Adversarial Networks." (Jun).

[6] Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, and ... 2017. "Improved Training of Wasserstein GANs." (Mar).

[7] Mao, Yi. 2019. "Understanding Variational Autoencoders (VAEs) | by Joseph Rocca." Towards Data Science. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[8] Radford, Alec, Luke Metz, and Soumith Chintala. 2015. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." (Nov).

[9] Rashad, Fathy. 2020. "Adversarial Auto Encoder (AAE). Combination of VAE and GAN | by Fathy Rashad | ViTrox-Publication." Medium.

https://medium.com/vitrox-publication/adversarial-auto-encoder-aae-a3fc86f71758.

[10] Wang, Xuhong, Ying Ying Du, Shijie Lin, and ... 2019. “adVAE: A self-adversarial variational autoencoder with Gaussian anomaly prior knowledge for anomaly detection.” (Mar).

[11] “YeongHyeon/adVAE: Implementation of 'Self-Adversarial Variational Autoencoder with Gaussian Anomaly Prior Distribution for Anomaly Detection.'” n.d. GitHub. https://github.com/YeongHyeon/adVAE.

[12] “YeongHyeon/CVAE-AnomalyDetection: Example of Anomaly Detection using Convolutional Variational Auto-Encoder (CVAE).” n.d. GitHub. https://github.com/YeongHyeon/CVAE-AnomalyDetection.