



# Anomaly detection using Contrastive Predictive Coding

Presentazione del progetto del corso “Advanced Deep Learning Models and Methods”

Prof. Giacomo Boracchi & Matteo Matteucci

Anno accademico 2021/2022

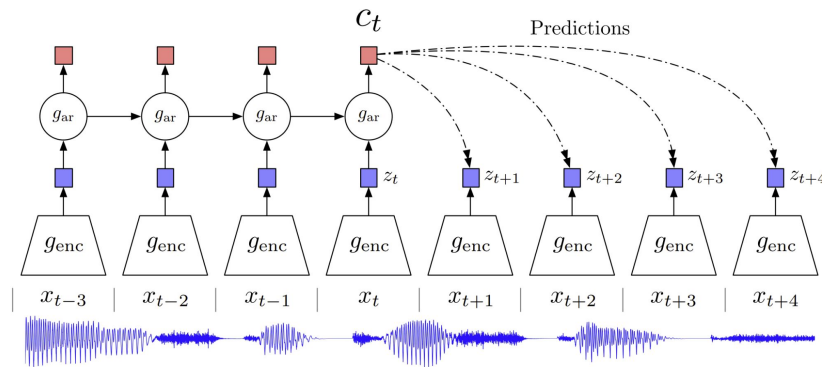


## Overview

- L'obiettivo di questo progetto è l'implementazione di un algoritmo di anomaly detection sfruttando il Contrastive Predictive Coding (CPC) e valutare la sua efficacia sfruttando tre diversi encoder.
- Per far ciò, abbiamo inizialmente studiato ed implementato parte del paper "[Representation Learning with Contrastive Predictive Coding](#)", che descrive il CPC applicato a diverse tipologie di dati, focalizzandoci sul dominio delle immagini.
- Siamo poi passati al cuore di questo progetto, riproducendo il paper "[Contrastive Predictive Coding for Anomaly Detection](#)", in cui il paradigma del CPC viene leggermente modificato ed applicato al fine di compiere segmentation e anomaly detection sulle immagini.
- Infine, abbiamo modificato l'architettura utilizzata nel precedente paper, provando diverse baseline ben studiate, tra cui Densenet, descritta nel paper "[Densely Connected Convolutional Networks](#)"

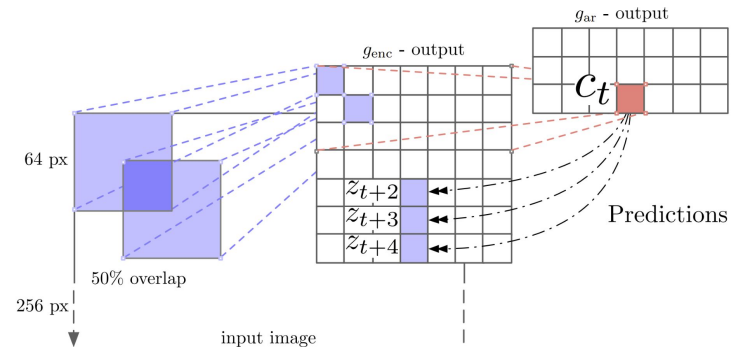
# Che cos'è il Contrastive Predictive Coding?

- Il CPC è un modello self-supervised che apprende come calcolare predizioni di rappresentazioni future nello spazio latente.
- Le predizioni vengono calcolate tramite l'impiego di potenti modelli autoregressivi.
- Nel dominio delle immagini, dividendo queste in patches, si interpreta ogni riga di patches come differente istante temporale



## Representation Learning with Contrastive Predictive Coding

- L'implementazione di questo paper è stata molto fedele alla descrizione. Come differenza principale abbiamo usato il noto [MVTec AD](#) dataset.
- Grandezza immagini iniziali: 256x256
- Grandezza patches: 64x64
- Encoder utilizzato: ResNet18 v2 fino al terzo residual block con un mean-pool per avere come output un vettore di 1024.
- Modello autoregressivo utilizzato: PixelCNN
- Le predictions vengono calcolate analizzando l'immagine dall'alto verso il basso, considerando quindi come patches "future" quelle sottostanti.
- Al context viene applicata una trasformazione lineare per calcolare le predictions. Ogni step  $k$  in avanti utilizza una diversa trasformazione lineare.





# InfoNCE

- Per quantificare la bontà delle predizioni viene utilizzata InfoNCE come funzione di loss, dove NCE sta per Noise Contrastive Estimation.
- InfoNCE loss

$$\mathcal{L}_k = -\mathbb{E}_X \left[ \log \frac{\exp(\mathbf{z}_{t+k}^\top \mathbf{W}_k \mathbf{c}_t)}{\sum_X \exp(\mathbf{z}_j^\top \mathbf{W}_k \mathbf{c}_t)} \right]$$

- Dove  $\mathbf{z}_t = g_{enc}(\mathbf{x}_t)$  è la rappresentazione nello spazio latente di una patch  $\mathbf{x}_t$  grazie all'utilizzo di un encoder non lineare.  $\mathbf{c}_t = g_{ar}(\mathbf{z}_{\leq t})$  è l'output di un modello autoregressivo ([PixelCNN](#)) and  $\mathbf{W}_k$  è la matrice delle trasformazioni lineari usata per calcolare le prediction.
- $X$  è l'insieme dei samples in cui è presente un positive sample  $(\mathbf{x}_t, \mathbf{x}_{t+k})$  e  $N - 1$  negativi  $(\mathbf{x}_t, \mathbf{x}_j)$  da cui  $\mathbf{x}_j$  viene pescato a random dal batch.

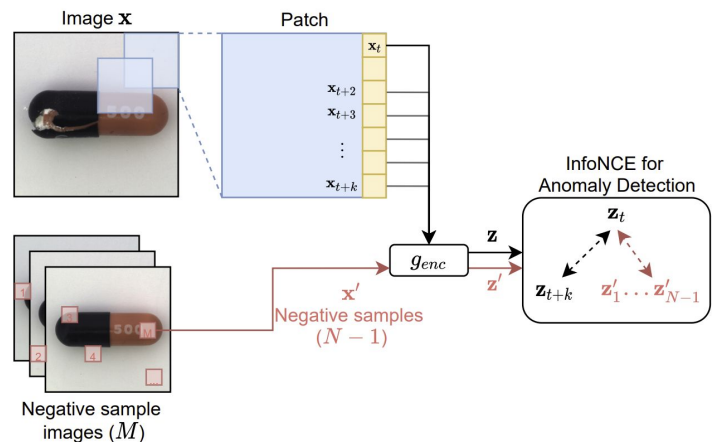



## Contrastive Predictive Coding for Anomaly Detection

- L'algoritmo di CPC viene modificato e adattato per compiere anomaly detection. In particolare vengono fatte due modifiche:
  1. Il modello autoregressivo utilizzato per calcolare le predizioni viene omissso. In questo modo si ottiene un modello più semplice ma comunque in grado di imparare rappresentazioni utili nello spazio latente. La funzione di loss quindi cambia, diventando:

$$\mathcal{L}_k = -\mathbb{E}_X \left[ \log \frac{\exp(\mathbf{z}_{t+k}^\top \mathbf{W}_k \mathbf{z}_t)}{\sum_X \exp(\mathbf{z}_j^\top \mathbf{W}_k \mathbf{z}_t)} \right]$$

2. Le patch usate come negative samples nella funzione di loss venivano in precedenza prese in maniera randomica dallo stesso batch. In questo caso, invece, per evitare che queste patch contengano anch'esse anomalie, cosa che ingannerebbe durante il calcolo della loss, le patches negative vengono prese sempre in maniera randomica ma dalle immagini utilizzate per il training, quindi senza anomalie, del dataset.



- 
- Il valore di loss di ogni immagine viene utilizzato direttamente come anomaly score.
  - Il modello dell'encoder utilizzato nel paper è ResNet-18 v2 fino al terzo blocco residuale.
  - Vengono trainati, inoltre, quattro modelli differenti per ogni immagine, uno per ogni direzione, per aumentare l'accuratezza. Viene poi calcolata la media delle quattro loss.



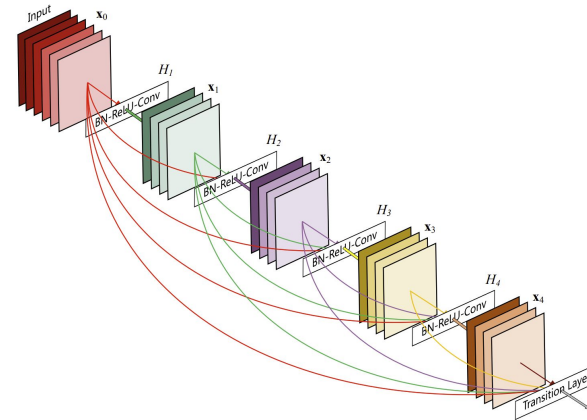


## Divisione in patches

- Il paper utilizza immagini di dimensione 768x768, dividendole in primo luogo in 25 patches da 256x256, e ognuna di queste in 49 sotto patches di dimensione 64x64. Totale  $25 \times 49 = 1.225$  patches per immagine.
- Un'epoca di training, usando la gpu offerta Kaggle (tempo limitato) e Pytorch, impiega circa 4 ore.
- Abbiamo quindi deciso di ridurre il numero di patches, utilizzando immagini di grandezza 256x256.
- Non potendo più comparare i risultati raggiunti con quelli del paper (i quali raggiunti con 1500 epoche di training con immagini 768x768), abbiamo concordato di provare diversi modelli noti per confrontarli tra loro.
- In particolare, abbiamo implementato, oltre all'originale ResNet-18, anche ResNet-50 e Densenet

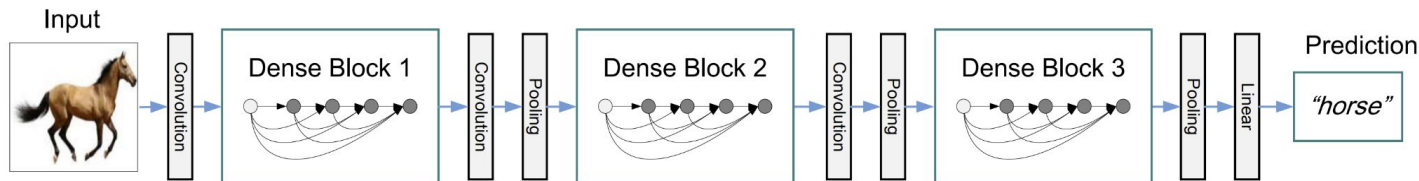
## Densely Connected Convolutional Networks

- Densenet è un'architettura che abbiamo implementato e ha portato miglioramenti consistenti, anche paragonati a ResNet18 e ResNet50
- La particolarità di questa rete è che ogni convolutional layer ha connessioni dirette a tutti i convolutional layer precedenti
- In particolare, le feature-maps di ogni convolutional layer vengono usate in input per tutti i layer successivi, concatenandole con le altre ricevute (a differenza di ResNet)



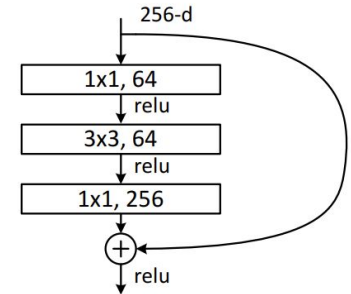
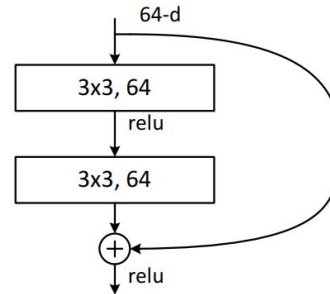
# Vantaggi di Densenet

- Vanishing gradient: questo problema viene robustamente contrastato in quanto ogni layer, anche quelli verso la fine della rete, hanno feature-maps in input soggette ad un basso numero di trasformazioni non-lineari.
- Pochi parametri: ogni convolutional layer calcola e aggiunge alle feature-maps dell'intera rete soltanto un limitato set di feature-maps ( $k$ , growth rate, numero di filtri delle convoluzioni) pari a quelle calcolate nel layer stesso.
- Ri-utilizzo di features: questo comporta a modelli più facili da trainare e con meno parametri.



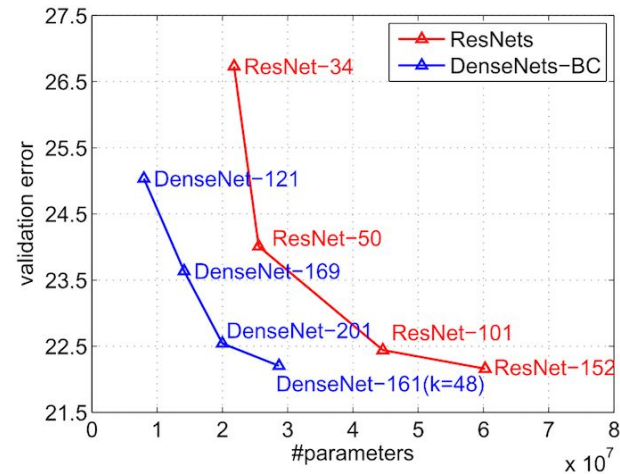
## ResNet18-v2 e ResNet50-v2

- ResNet18-v2 e ResNet50-v2 sono due ulteriori architetture che abbiamo implementato e confrontato con Densenet.
- Entrambe fino al terzo blocco residuale.
- ResNets fanno uso delle skip connections (identity functions), questo aiuta a contrastare il vanishing gradient problem.
- A sinistra un blocco residuale della ResNet18-v2
- Blocco residuale bottleneck della ResNet50-v2. I filtri 1x1 riducono e ripristinano le dimensioni, lasciando il layer nel mezzo con dimensioni minori.



## Dataset e metriche

- Il dataset utilizzato è ancora MVTec AD, utilizzando le prime 5 classi (Bottle, Cable, Capsule, Carpet, Grid).
- Come metrica abbiamo utilizzato AUC, ovvero l'area sottesa dalla curva ROC.
- Early stopping
- Kaggle come framework e Pytorch





## Risultati

- Risultati ottenuti con immagini RGB di dimensione 256x256
- Limitazioni di potenza e tempo di calcolo
- Densenet raggiunge sempre risultati migliori rispetto alle ResNets

Class	Epochs	ResNet18	ResNet50	DenseNet
Bottle	30	0.558	0.650	0.790
	55	0.419	0.689	0.826
	70	0.449	0.704	0.876
Cable	30	0.770	0.644	0.840
	55	0.721	0.635	0.821
	70	0.736	0.651	0.858
Capsule	30	0.553	0.598	0.609
Carpet	30	0.565	0.597	0.618
Grid	30	0.307	0.275	0.381