Report della quinta edizione del ML_INFN Hackathon

Domingo Ranieri

domingo.ranieri@cnaf.infn.it

7 Dic 2023

INFN - CNAF

ML_INFN

Progetto triennale nato nel 2020 con l'obiettivo di favorire lo sviluppo di tecnologie AI-driven. A tal fine il progetto fornisce sia una piattaforma , comune ed espandibile, sia la possibilità di condividere le conoscenze già acquisite da ricercatori e tecnologi.

WP1	WP2	WP3
Stefano Dal Pra	**Francesca Lizzi**	**Luca Giommi**
Infrastruttura & Provisioning	Formazione	Casi scientifici

WP1 – Infrastruttura & Provisioning

La base infrastrutturale di ML_INFN è costituita da **INFN Cloud**.
Tramite questo è possibile accedere ad un cluster di risorse ad alte performance (tra cui GPU) progettato per adattarsi ai progetti dell'INFN che utilizzano tecniche di Machine Learning.

WP2 – Formazione

Primo Hackathon
Base level
online
7-9 Giu 2021
54 partecipanti

Secondo Hackathon
Base level
online
13-15 Dic 2021
60 partecipanti

Terzo Hackathon
Anvanced level
Bari
21-24 Nov 2022
23 partecipanti

Quarto Hackathon
Base level
online
21-23 Giu 2023
48 partecipanti

Quinto Hackathon
Advanced level
Pisa
13-16 Nov 2023
25 partecipanti

WP3 – Casi scientifici

Organizza meeting settimanali con seminari relativi ad applicazioni di Machine learning su temi di interesse per l'INFN. Nel 2023 sono stati organizzati 22 seminari.
Raccolta e ampliamento della collezione di casi d'uso realistici di tecniche di Machine Learning nelle varie linee di ricerca dell'Ente.
Come entry point del progetto verso l'esterno e come catalogo degli use case viene usato **Confluence**

Table of Use cases

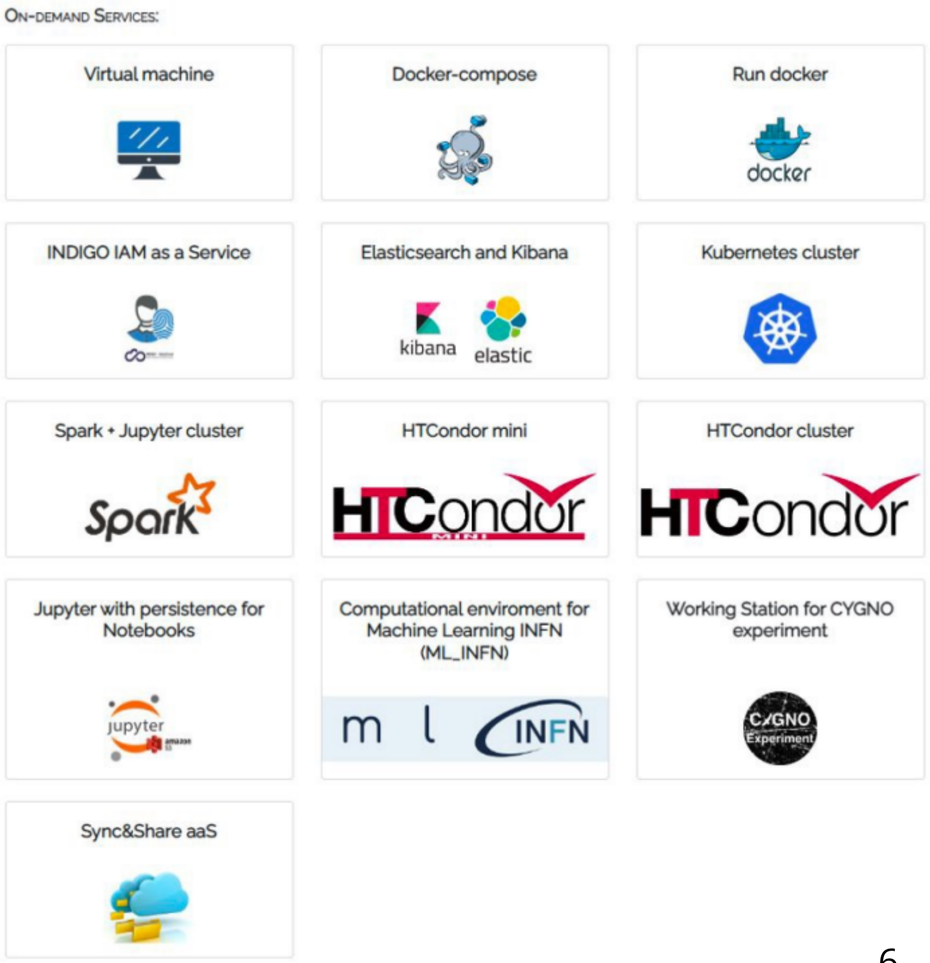
Name and Link	Goal	ML Algorithms	Scientific Field	ML Tools	Comments
Btagging in CMS (templated version)	Classification	CNN, LSTM	High Energy Physics	Keras + Tensorflow	Realistic application
LHCb Masterclass, with Keras	Density estimation and classification	MLP	High Energy Physics	ROOT + Keras + TF	Introductory tutorial
MNIST in a C header	Classification	MLP		Keras	Free-styling tutorial
LUMIN: Lumin Unifies Many Improvements for Networks	Technological	CNN, RNN, GNN	High Energy Physics	PyTorch	Package use examples
INFERNO: Inference-Aware Neural Optimisation	Classification	NN	High Energy Physics	Keras + Tensorflow	Technique application example
An introduction to classification with CMS data	Classification	Fisher, BDT, MLP	High Energy Physics	Scikit-learn, TF2	Tutorials for Master Students
Virgo Autoencoder tutorial	Data Compression	Autoencoder	General Relativity	Python Keras	Tutorial for student
Distributed training of neural networks with Apache Spark	Technological	DNN	High Energy Physics	Spark + BigDL	Tutorial
FTS log analysis with NLP	Self-supervised, clustering	NLP	High Energy Physics, Computing	Word2Vec + Rake + sklearn	Tutorial

Programma dell'hackathon

Lunedì	Martedì	Mercoledì	Giovedì
	Seminari: Autoencoders, modelli generativi e transformesrs	Seminari: NN per risolvere equazioni differenziali, CNN e U-NeT	Seminari: Riflessioni e applicazioni di tecniche di Machine Learning
Seminari: Infrastruttura INFN e servizi Cloud	Hands on: • Transformers and Visual Transformer • **Anomaly detection with Autoencoders in CMS**	Hands on: • **Lung Sementation on Chest X-Ray images with U-Net** • Solving differential equation with deep learning	

Infrastructure, Platform and Software as Services in INFN Cloud (Luca Giommi)

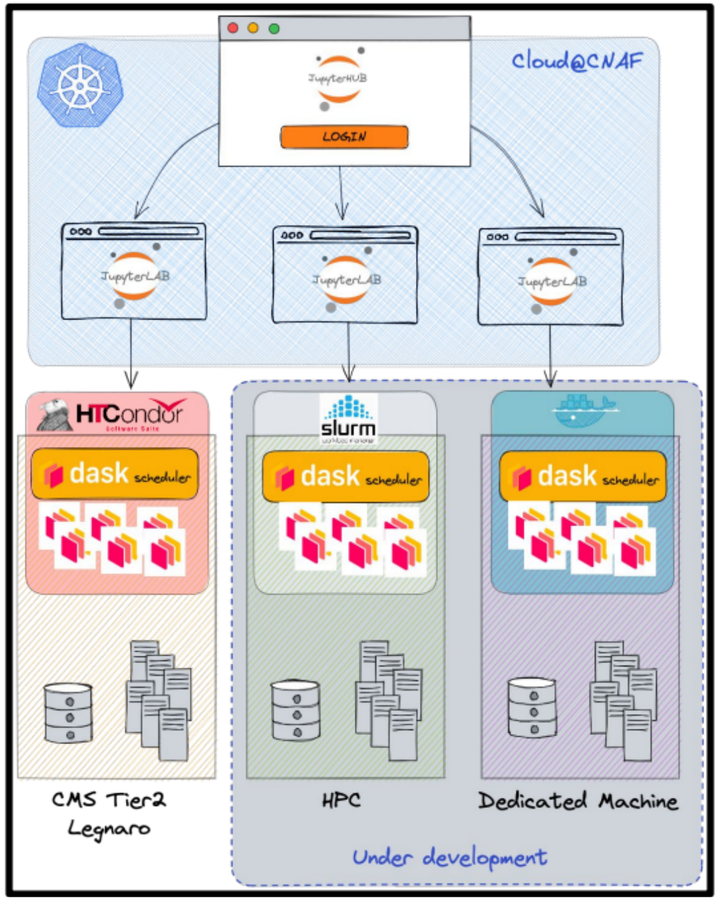
INFN Cloud è un progetto reso disponibile da Marzo 2021 che mette a disposizione risorse di calcolo e un portfolio di servizi.
L'infrastruttura si basa su una backbone che collega il CNAF con ReCas (Bari) alla quale si collegano altri centri di calcolo federati.
I servizi sono descritti seguendo un approccio dichiarativo e istanziati con paradigma Infrastructure as Code (IaS).



A scientific perspective on Cloud scalability (Daniele Spiga)

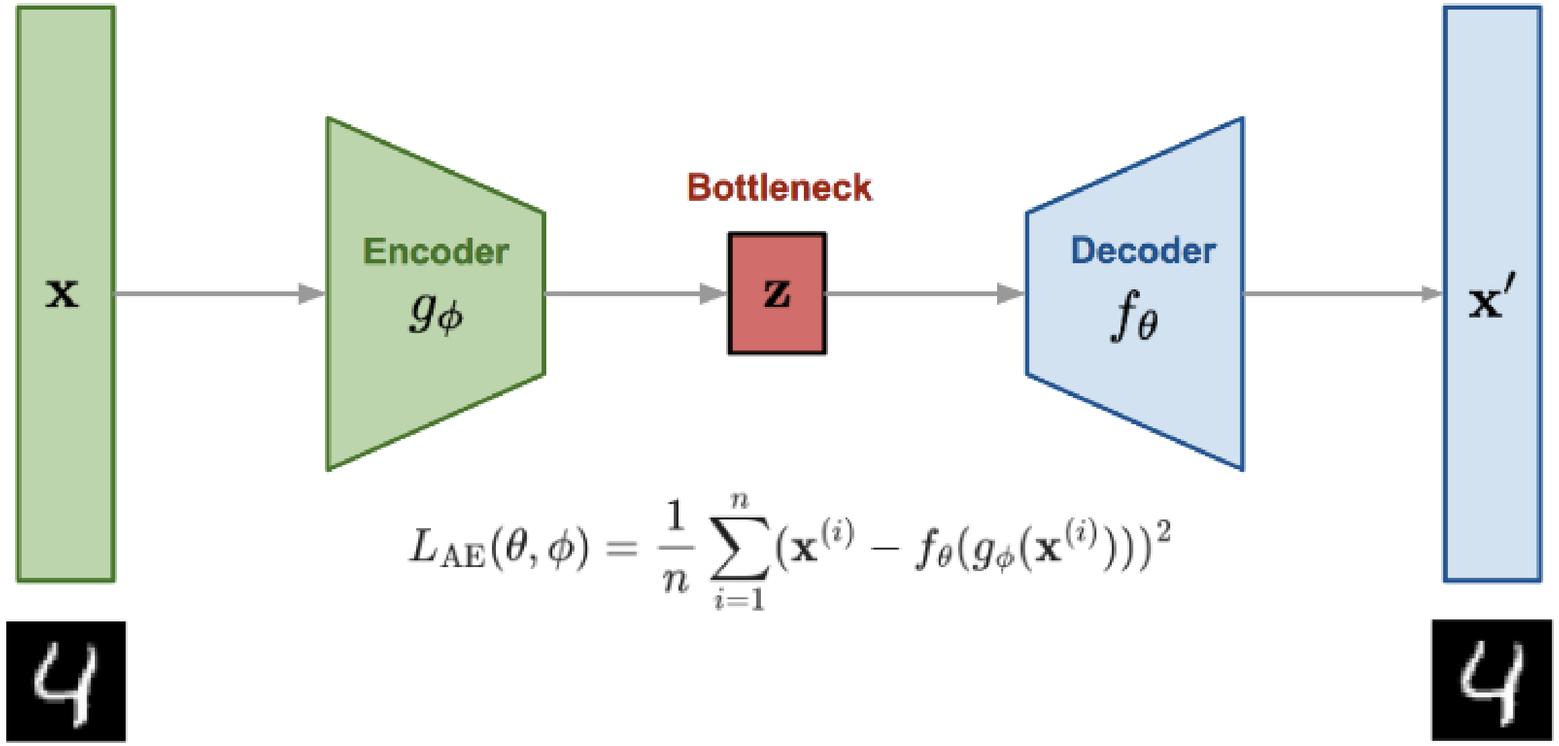
Come sappiamo la quantità di dati prodotta in diversi esperimenti, eccede le capacità dei laptop di poterli elaborare.

Nel progetto **interTwins** si sta sviluppando un servizio che permette agli utenti di accedere tramite **Cloud** a infrastrutture WLCG utilizzando appositi batch-systems (HTCondor, slurm) usando JupyterHub come entrypoint.



Autoencoders

- L'Encoder codifica l'input **x** nello spazio latente di dimensione inferiore restituendo **z**
- Il Decoder decodifica **z** dallo spazio latente a quello di partenza producendo **x'**
- La loss è minimizzata se **x** e **x'** sono uguali

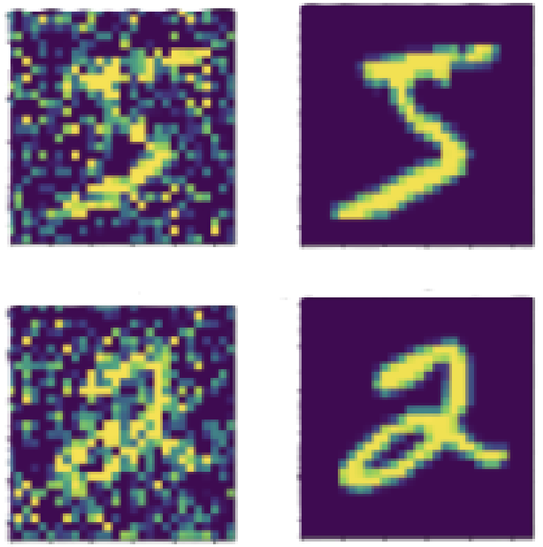


Possibili applicazioni

- **Anomaly detection:** il modello allenato con uno specifico tipo di immagini ha una reconstruction loss elevata quando viene testato su immagini di tipologia diversa

- **Denoising:** il modello può essere allenato a tale scopo fornendogli immagini con del rumore e confrontando gli output con le stesse immagini prive di rumore

- **Super resolution:** usando un'idea simile alla precedente è possibile aumentare la risoluzione di immagini. Spesso in questi modelli si usano anche le CNN

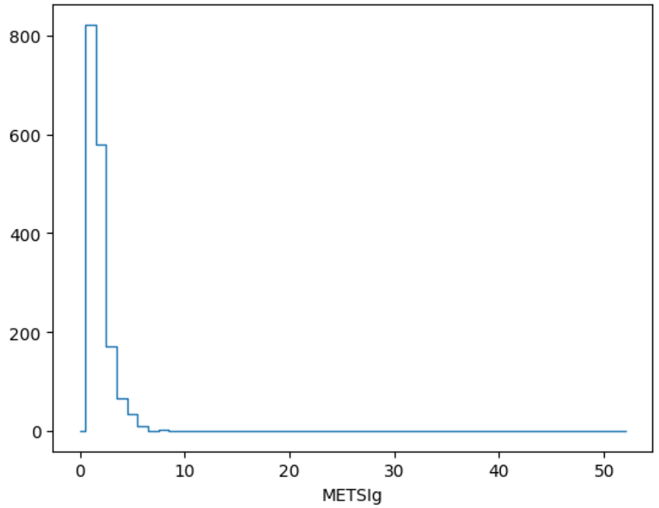


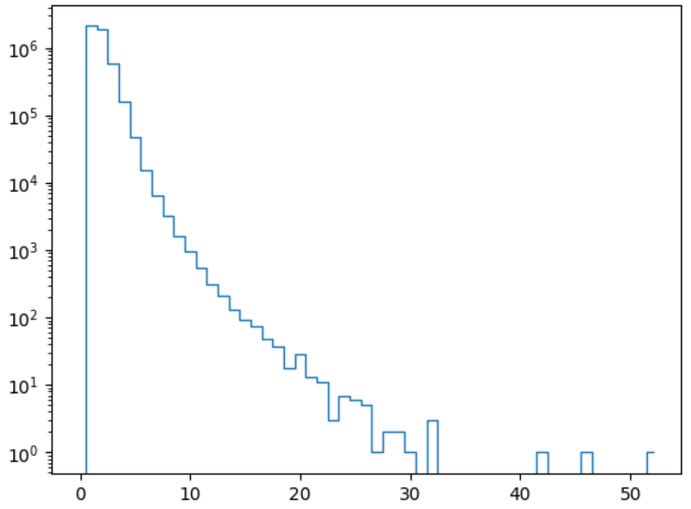
Hands on : Anomaly detection with Autoencoders in CMS

L'esperimento Compact Muon Solenoid (CMS), è uno dei grandi rilevatori di particelle di indirizzo generale costruiti sul Large Hadron Collider al CERN. Lo scopo dell'esperimento CMS è compiere ricerche su una vasta gamma di fenomeni fisici, attraverso la rilevazione dei prodotti dalla collisione protone-protone.

$$\mathrm{MET} = |\sum_i \vec{p}_{T,i}|$$

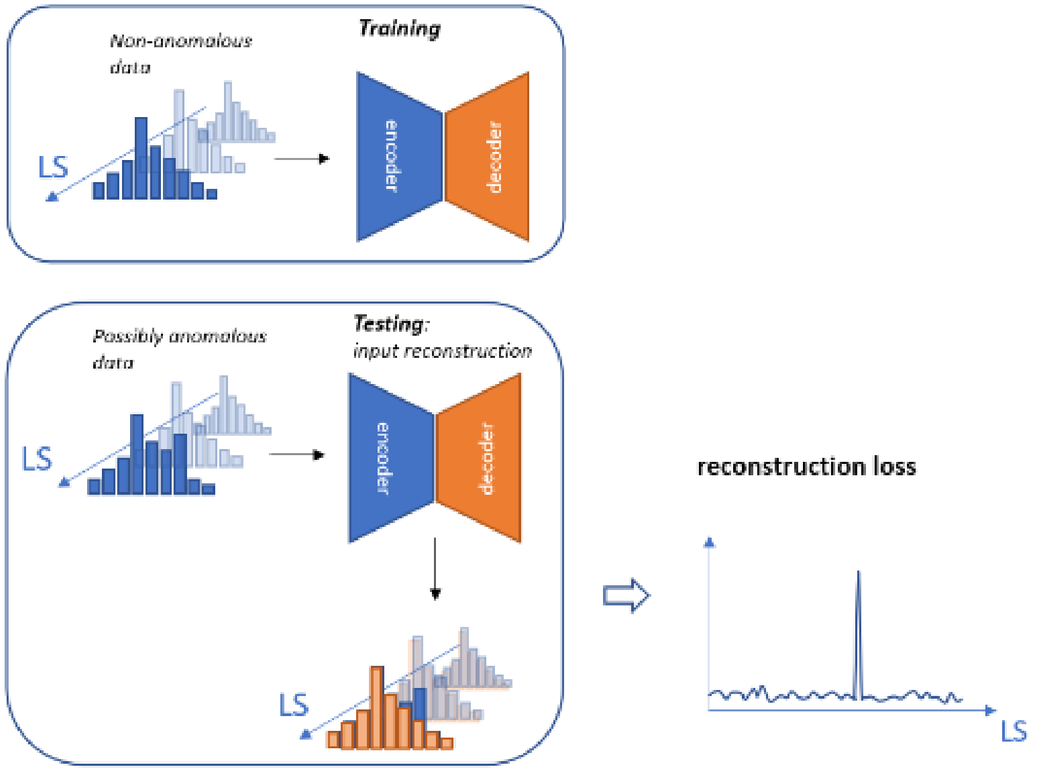
$$\mathrm{METSig} = \frac{\mathrm{MET}}{\sqrt{\sum_i |\vec{p}_{T,i}|}}$$





Hands on : Anomaly detection with Autoencoders in CMS

Durante il training la rete viene addestrata utilizzando dati che non presentano anomalie.
Nella fase di test ci attendiamo di osservare dei picchi nella reconstruction loss in presenza delle anomalie.



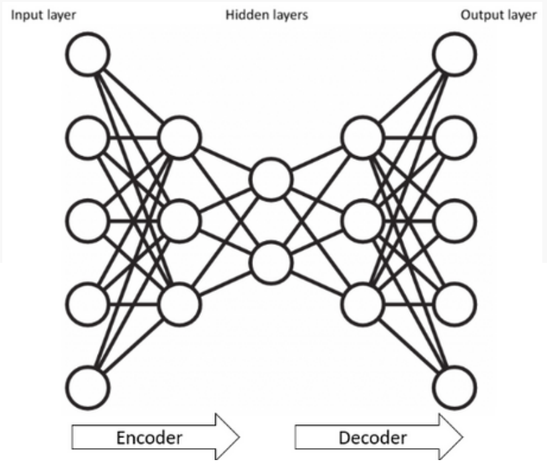
Hands on : Anomaly detection with Autoencoders in CMS

Una parte del **codice** sviluppato

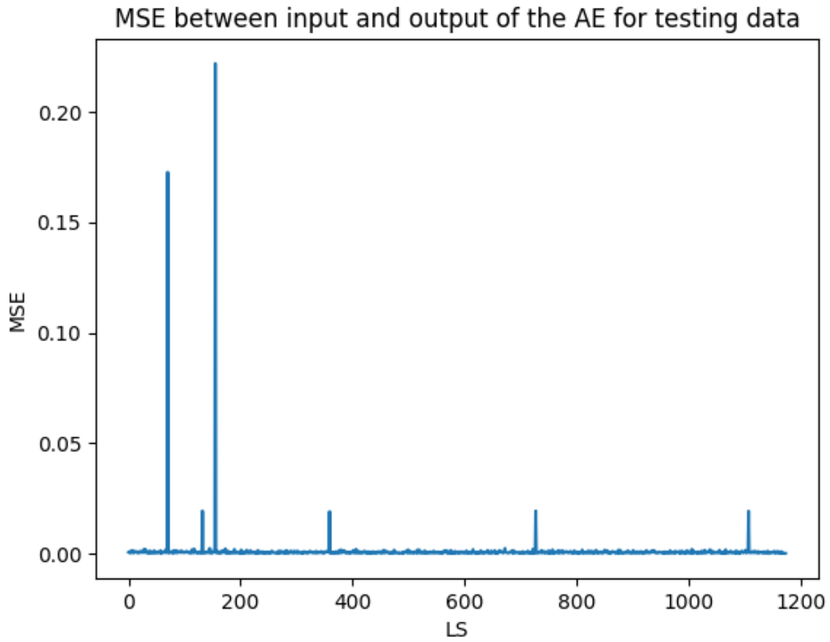
```
input = keras.Input(shape=(training.shape[1],))
learning_rate = 1e-7
encoding_dim = 64
encoding_dim_2 = 8

encoded = layers.Dense(encoding_dim,activation="relu", activity_regularizer=tf.keras.regularizers.l2(learning_rate))(input)
latent = layers.Dense(encoding_dim_2,activation="relu")(encoded)
decoded=layers.Dense(encoding_dim,activation="relu")(latent)
decoded=layers.Dense(training.shape[1],activation="sigmoid")(decoded)
metrics = ['mse']
model = keras.Model(input, decoded)
model.compile(optimizer='adam', loss='mse', metrics=metrics)
```

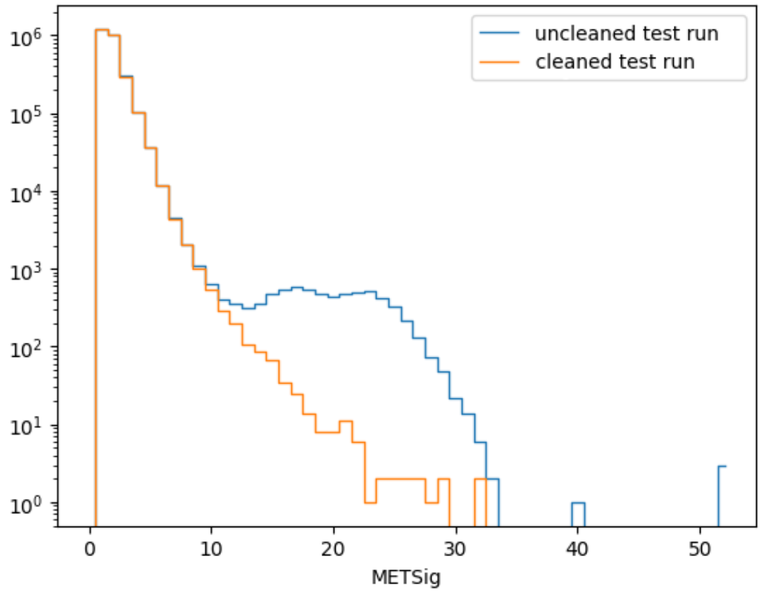
```
batch_size =100
epochs = 150
history=model.fit(training, training, batch_size=batch_size,epochs=epochs)
```



Hands on : Anomaly detection with Autoencoders in CMS



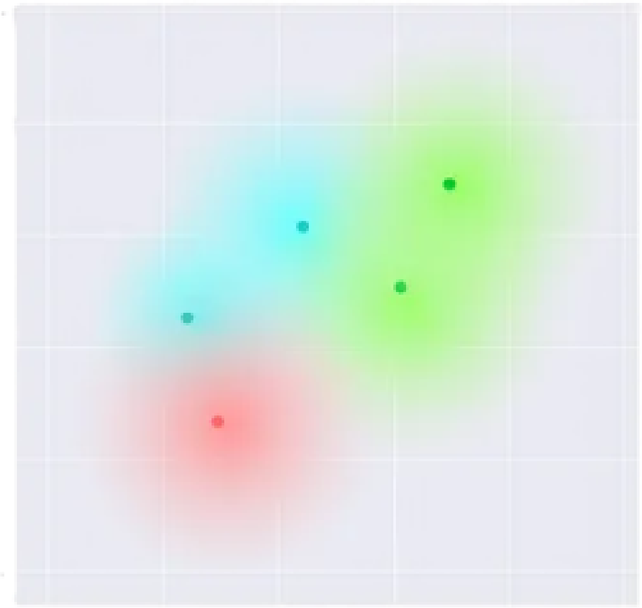
Reconstruction Loss sul dataset di test



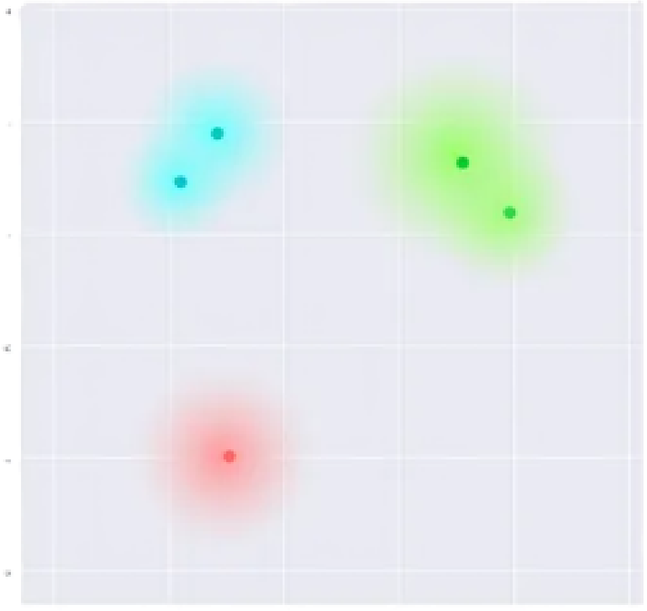
Eliminando le anomalie trovate, il dataset risulta compatibile con quello di training

Generative autoencoders

Al fine di poter usare questi modelli come generatori di nuove immagini, è necessario regolarizzare lo spazio latente.
Lo spazio latente regolare presenta le caratteristiche di **continuità** e **completezza**



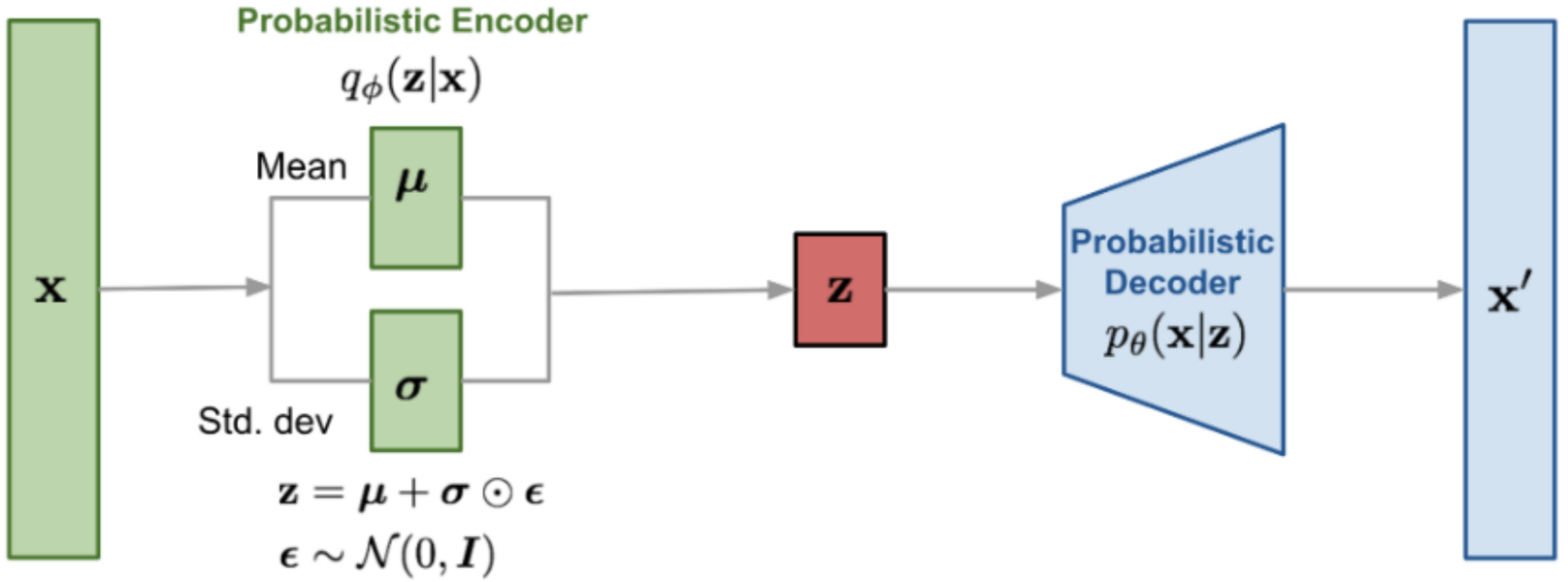
Spazio latente regolare



Spazio latente irregolare

Variational Autoencoders

- L'Encoder questa volta non produce un punto nello spazio latente ma una distribuzione
- L'elemento passato al Decoder viene estratto dalla distribuzione precedentemente trovata
- La loss è formata da due termini in competizione : la reconstruction loss e la divergeneza di Kullback-Leibler



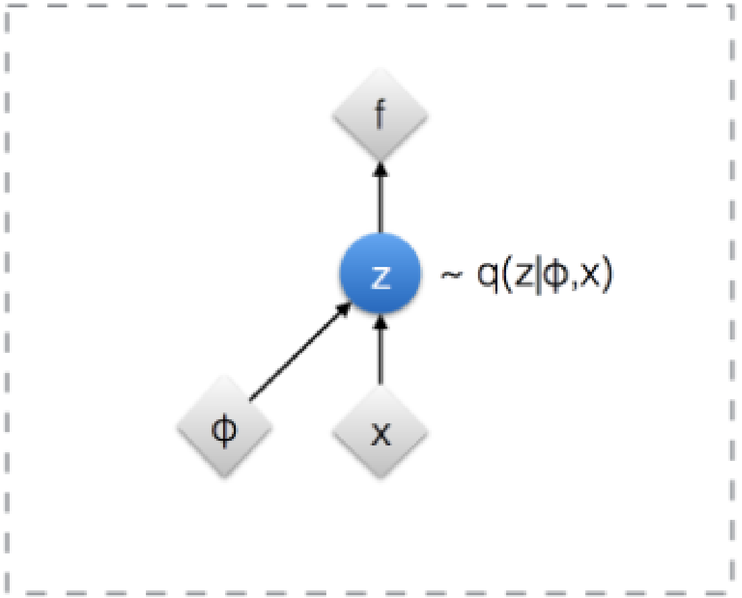
$$L_{\mathrm{VAE}}(\theta, \phi) = -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

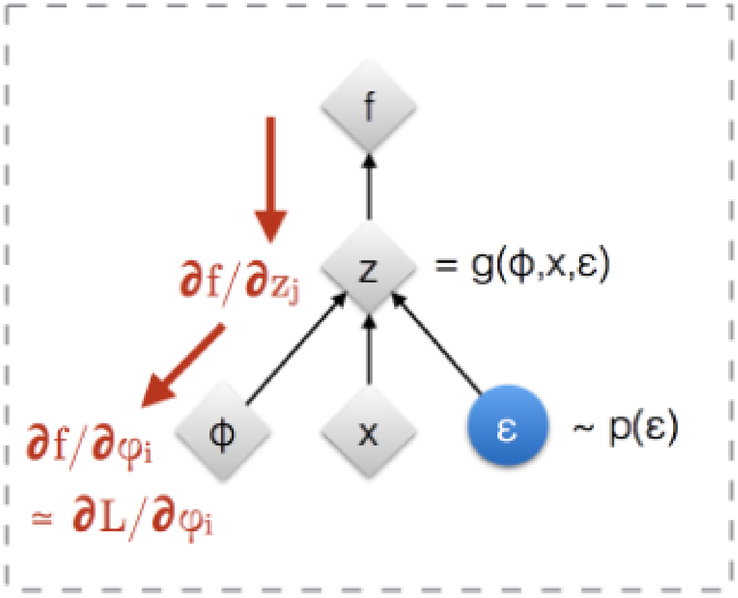
$$-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})$$

Variational Autoencoders

Il sampling è un processo stocastico dunque non è possibile eseguire la **backpropagation**

Con un'opportuna riparametrizzazione questo può essere evitato





: Nodo deterministico

: Nodo randomico

Variational Autoencoders

Problemi noti delle VAE:

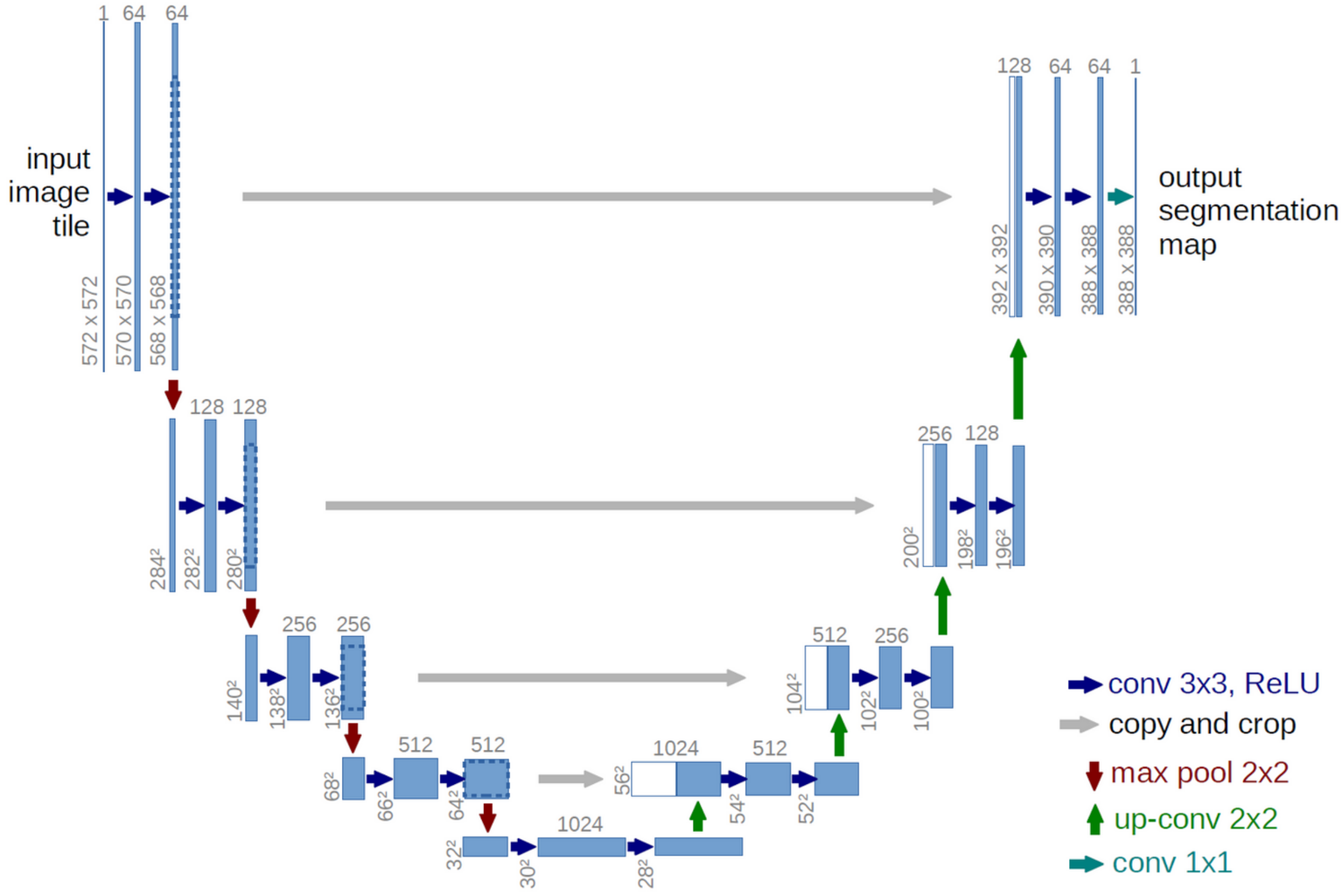
- **Mode collapse:** il modello genera solo alcune categorie di output
- **Blurriness:** le VAE spesso producono output poco nitidi
- **Training complesso:** causato dal bilanciamento dei termini della Loss

Immagini di input

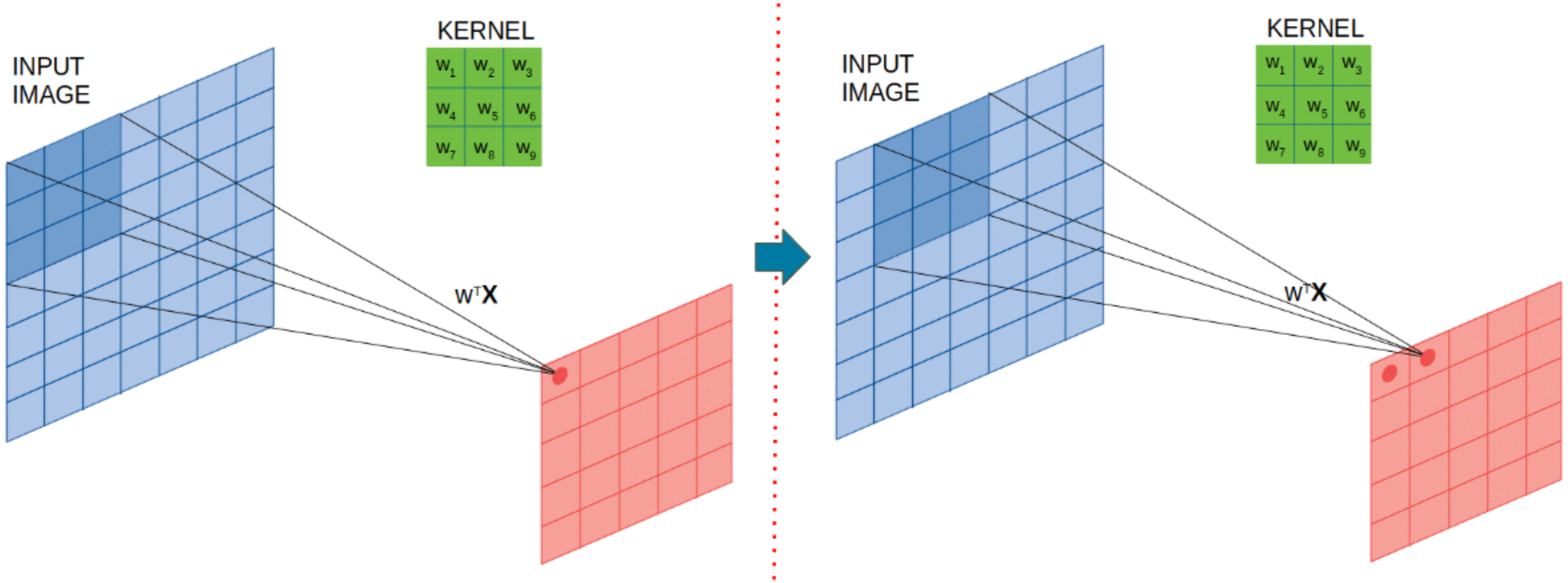
Immagini ricostruite con VAE

U-Net

La U-Net ha una struttura del tipo encoder-decoder tipica degli autoencoders.
Si tratta però di una **Fully Convolutional Neural Network** in quanto non contiene dense layers.

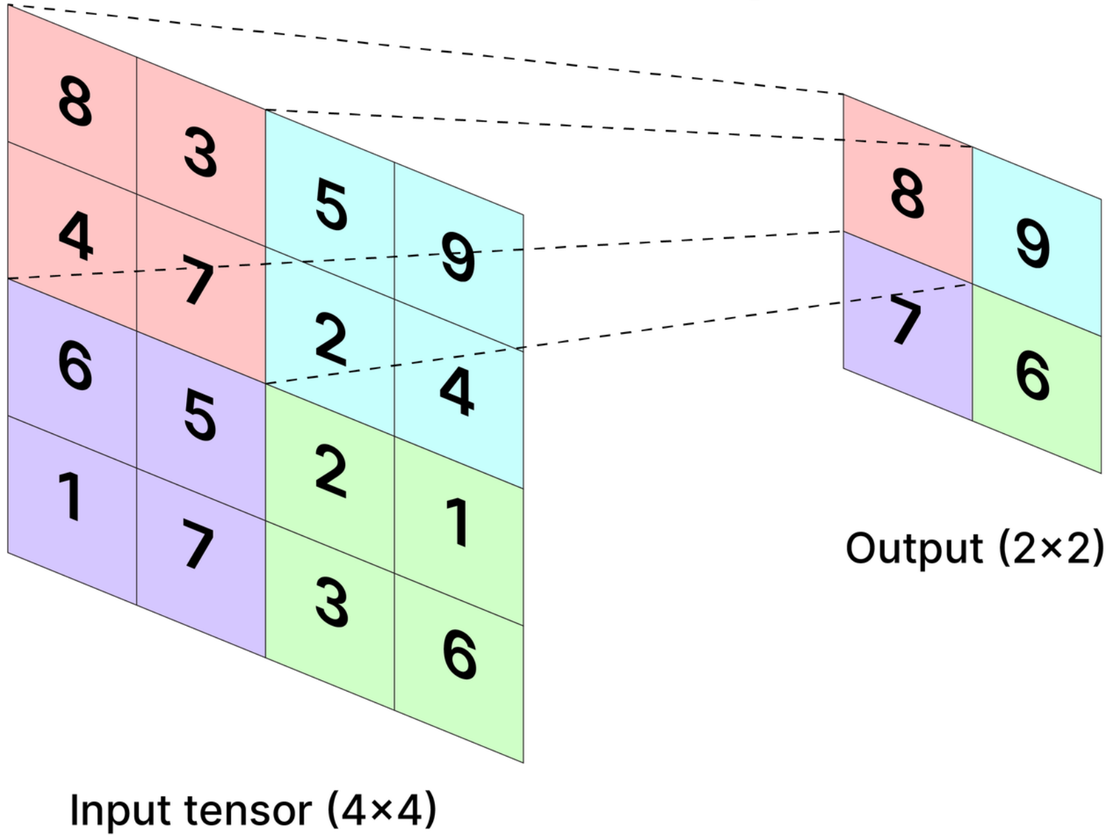


Convolution



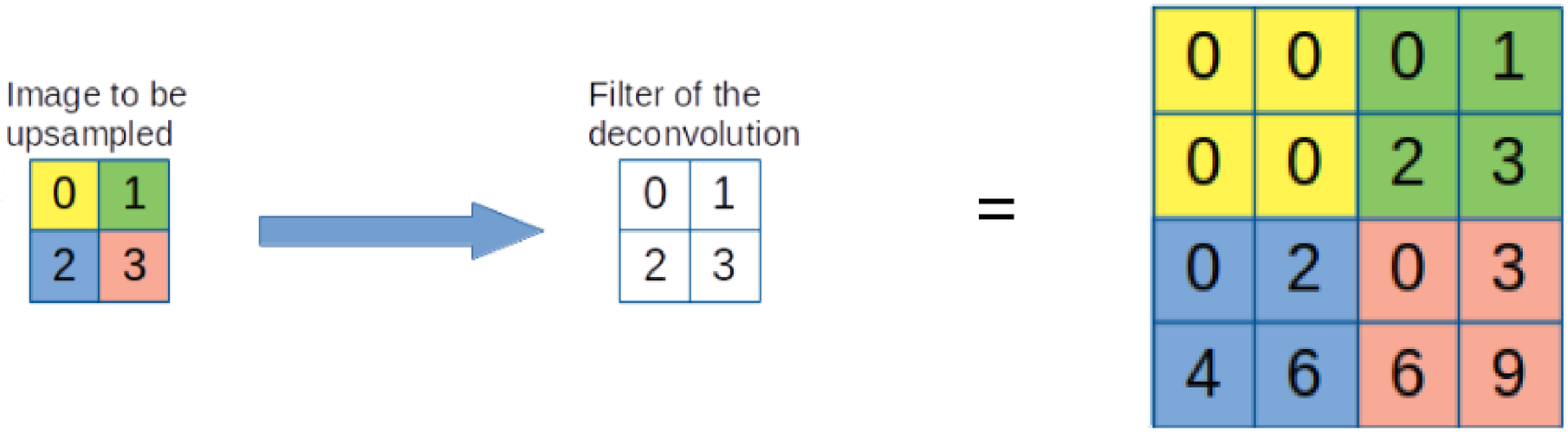
Il risultato dell'operazione di convoluzione è detta **activation map.**
I pesi sono appresi dalla rete neurale mentre alcuni iperparametri sono: il numero di kernel, lo stride e il padding.

Pooling



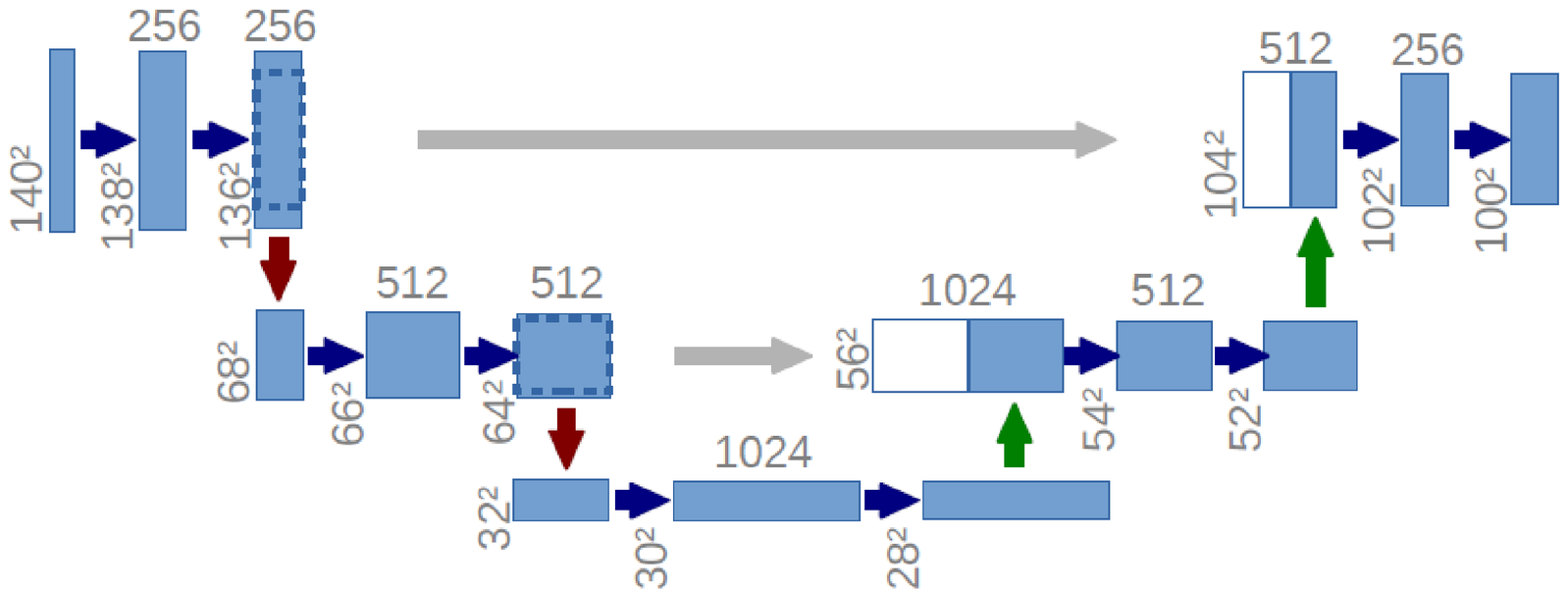
L’operazione di pooling riduce le dimensioni dell’activation map.
Può essere fatta in vari modi, un esempio è il **max pooling.**

Transposed convolution



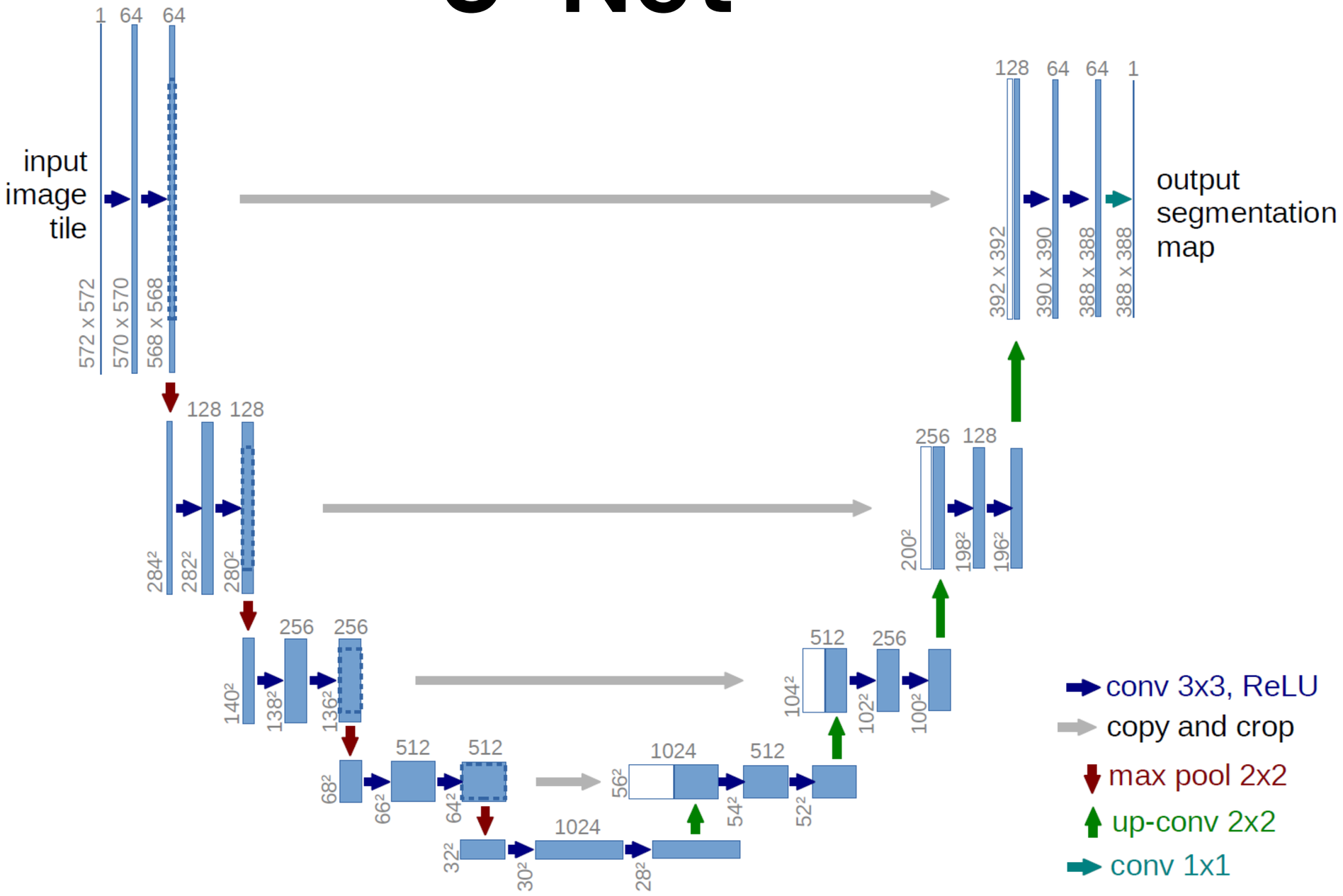
In questa operazione si aumentano le dimensioni dell'activation map attraverso dei **filtri di deconvoluzione.**

Skip connection



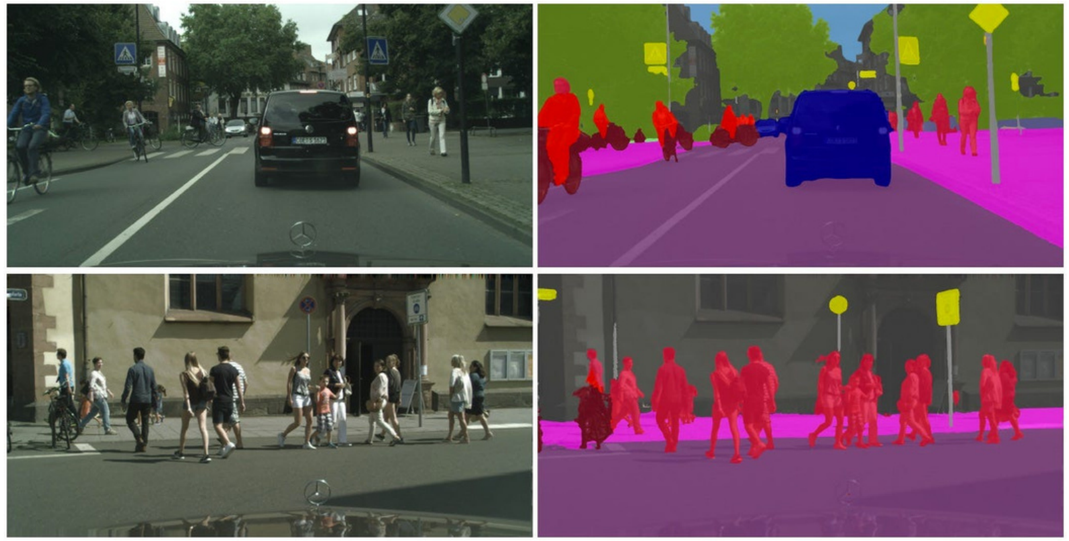
Le **skip connections** collegano activation maps non consecutive.
Durante il processo di encoding vengono perse delle informazioni che vengono recuperate in questo modo nella fase di decoding.

U-Net


input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024 512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 1
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Hands on : Lung Segmentation on Chest X-Ray images with U-Net

Una delle applicazioni principali delle U-Net è l'image segmentation.
In una radiografia al torace, i pixel che rappresentano i polmoni sono una parte ristretta del totale. Il nostro obiettivo è quello di evidenziare solo le zone dell'immagine che appartengono ai polmoni.



Hands on : Lung Segmentation on Chest X-Ray images with U-Net

Una parte del **codice** sviluppato

```
def unet_layer_left(previous_layer, n_filters, ker_size_1, strides_1, ker_size_2, strides_2, ker_size_3, strides_3, reg):
    '''Definition of layers of the left part of the u-net.
    Parameters
    ------------------
    previous_layer : input layer for this block;
    n_filters : number of filters of all the 3D convolutional layers;
    ker_size_1 : kernel dimension of the first conv3D layer of this block;
    strides_1 : strides size to be used for the convolution of the first Conv3D layer;
    ker_size_2/3 : kernel dimension of the second/third Conv3D layers;
    strides_2/3 : strides size to be used for the convolution of the second/third Conv3D layer;
    '''
    layer_L=Conv2D(filters=n_filters,kernel_size=ker_size_1,strides=strides_1,kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(previous_layer)
    layer_L=BatchNormalization(axis=-1)(layer_L)
    layer_L_shortcut=layer_L
    layer_L=Activation('relu')(layer_L)
    layer_L=SpatialDropout2D(0.2)(layer_L)
    layer_L=Conv2D(filters=n_filters,kernel_size=ker_size_2,strides=strides_2,padding='same',kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(layer_L)
    layer_L=BatchNormalization(axis=-1)(layer_L)
    layer_L=Activation('relu')(layer_L)
    layer_L=SpatialDropout2D(0.2)(layer_L)
    layer_L=Conv2D(filters=n_filters,kernel_size=ker_size_3,strides=strides_3,padding='same',kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(layer_L)
    layer_L=BatchNormalization(axis=-1)(layer_L)
    layer_L=Add()([layer_L,layer_L_shortcut])
    layer_L=Activation('relu')(layer_L)
    layer_L=SpatialDropout2D(0.2)(layer_L)
    return layer_L
```

Hands on : Lung Segmentation on Chest X-Ray images with U-Net

```
def unet_layer_bottleneck(previous_layer, n_filters, ker_size_1, strides_1, ker_size_2, strides_2, reg):
    '''Definition of the last layer of the network.
    Parameters
    ------------------
    previous_layer : input layer for this block;
    n_filters : number of filters of all the 3D convolutional layers;
    ker_size_1 : kernel dimension of the first conv3D layer of this block;
    strides_1 : strides size to be used for the convolution of the first Conv3D layer;
    ker_size_2 : kernel dimension of the second Conv3D layers;
    strides_2 : strides size to be used for the convolution of the second Conv3D layer;
    '''
    layer_L=Conv2D(filters=n_filters,kernel_size=ker_size_1,strides=strides_1,kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(previous_layer)
    layer_L=BatchNormalization(axis=-1)(layer_L)
    layer_L_shortcut=layer_L
    layer_L=Activation('relu')(layer_L)
    layer_L=Conv2D(filters=n_filters,kernel_size=ker_size_2,strides=strides_2,padding='same',kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(layer_L)
    layer_L=BatchNormalization(axis=-1)(layer_L)
    layer_L=Add()([layer_L,layer_L_shortcut])
    layer_L=Activation('relu')(layer_L)
    return layer_L
```

Hands on : Lung Segmentation on Chest X-Ray images with U-Net

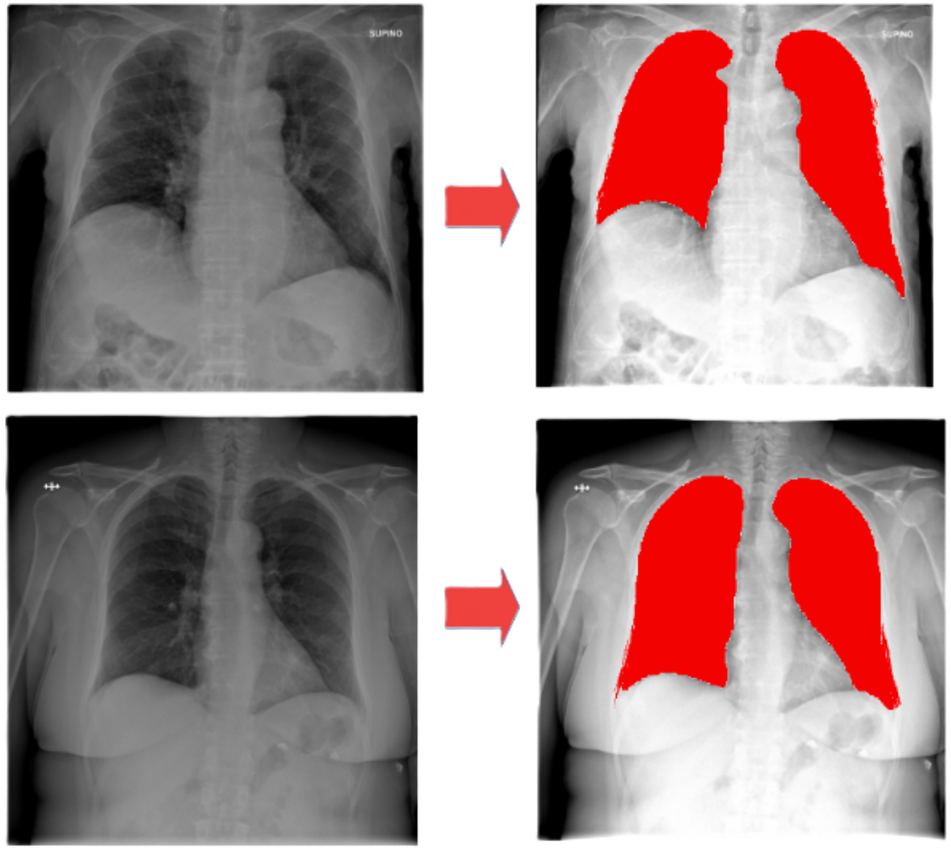
```
def unet_layer_right(previous_layer, layer_left, n_filters, ker_size_1, strides_1, output_pad, ker_size_2, strides_2, ker_size_3, strides_3, reg):
    '''Definition of layers of the right part of the u-net.
    Parameters
    ------------------
    previous_layer : input layer for this block;
    layer_left : output layer of the left part at the same depth;
    n_filters : int, number of filters of all the 3D convolutional layers;
    ker_size_1 : int, kernel dimension of the first conv3D layer of this block;
    strides_1 : int, strides size to be used for the convolution of the first Conv3D layer;
    output_pad : tuple, padding for the output;
    ker_size_2/3 : int, kernel dimension of the second/third Conv3D layers;
    strides_2/3 : int, strides size to be used for the convolution of the second/third Conv3D layer;
    '''
    layer_R=Conv2DTranspose(filters=n_filters,kernel_size=ker_size_1,strides=strides_1,output_padding=output_pad,kernel_regularizer=regularizers.l2(reg), kernel_initializer='random
    layer_R=BatchNormalization(axis=-1)(layer_R)
    layer_R_shortcut=layer_R
    layer_R=Activation('relu')(layer_R)
    merge=Concatenate(axis=-1)([layer_left,layer_R])
    layer_R=Conv2D(filters=n_filters,kernel_size=ker_size_2,strides=strides_2,padding='same',kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(merge)
    layer_R=BatchNormalization(axis=-1)(layer_R)
    layer_R=Activation('relu')(layer_R)
    layer_R=SpatialDropout2D(0.2)(layer_R)
    layer_R=Conv2D(filters=n_filters,kernel_size=ker_size_3,strides=strides_3,padding='same',kernel_regularizer=regularizers.l2(reg), kernel_initializer='random_normal')(layer_R)
    layer_R=BatchNormalization(axis=-1)(layer_R)
    layer_R=Add()([layer_R,layer_R_shortcut])
    layer_R=Activation('relu')(layer_R)
    layer_R=SpatialDropout2D(0.2)(layer_R)

    return layer_R
```

Hands on : Lung Segmentation on Chest X-Ray images with U-Net

```
def U_net(input_size):
    '''This function builds the network without compiling it.
    Parameters
    ----------
    input_size : tuple , size of the input
    reg : float , regularization parameters (L2)
    '''
    inputs=Input(shape=(input_size)) ##
    Level_1_L = unet_layer_left(inputs, n_filters=16, ker_size_1=1,strides_1=1,ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_2_L = unet_layer_left(Level_1_L, n_filters=32,ker_size_1=2, strides_1=2, ker_size_2=3,strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_3_L = unet_layer_left(Level_2_L, n_filters=64, ker_size_1=2, strides_1=2, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_4_L = unet_layer_left(Level_3_L, n_filters=128, ker_size_1=2, strides_1=2, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_5_L = unet_layer_left(Level_4_L, n_filters=256, ker_size_1=2, strides_1=2, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_6_L = unet_layer_bottleneck(Level_5_L, n_filters= 512, ker_size_1=1, strides_1=1, ker_size_2=3, strides_2=1, reg=0.1)
    Level_5_R = unet_layer_right(Level_6_L, Level_5_L, n_filters=256, ker_size_1=1, strides_1=1, output_pad=None, ker_size_2=3, strides_2=1, ker_size_3=3, strides_3=1, reg=0.1)
    Level_4_R = unet_layer_right(Level_5_R, Level_4_L, n_filters=128, ker_size_1=2, strides_1=2, output_pad=None,ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_3_R = unet_layer_right(Level_4_R, Level_3_L, n_filters=64, ker_size_1=2, strides_1=2, output_pad=None, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_2_R = unet_layer_right(Level_3_R, Level_2_L, n_filters=32, ker_size_1=2, strides_1=2, output_pad=None, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    Level_1_R = unet_layer_right(Level_2_R, Level_1_L, n_filters=16, ker_size_1=2, strides_1=2, output_pad=None, ker_size_2=3, strides_2=1, ker_size_3=1, strides_3=1, reg = 0.1)
    output=Conv2D(filters=1,kernel_size=1,strides=1, activation = 'sigmoid', kernel_regularizer=regularizers.l2(0.1))(Level_1_R)
    model=Model(inputs=inputs,outputs=output)
    return model
```

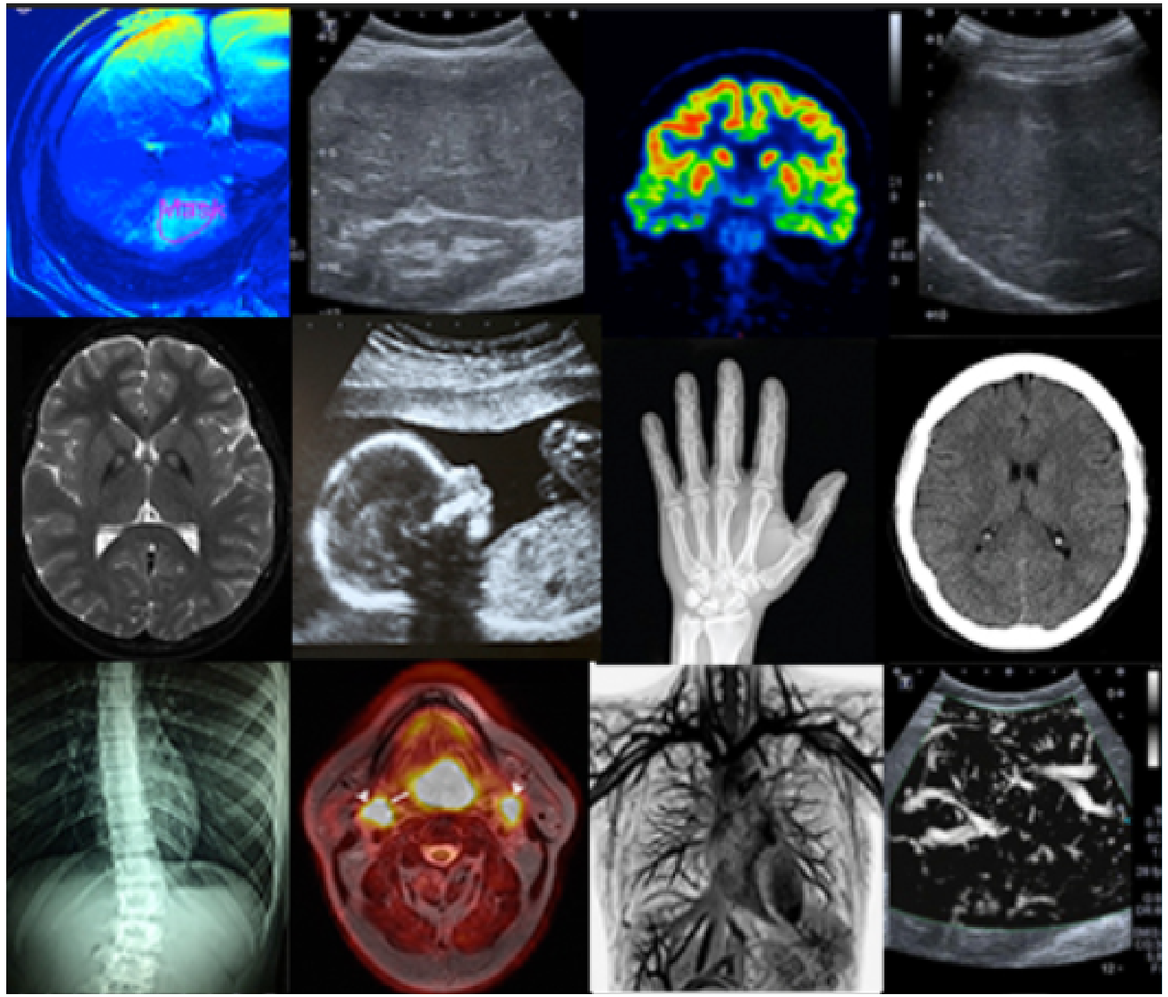
Hands on : Lung Segmentation on Chest X-Ray images with U-Net



Applicando la U-Net è stato possibile evidenziare solo i pixel interessati

Problemi aperti e obiettivi futuri

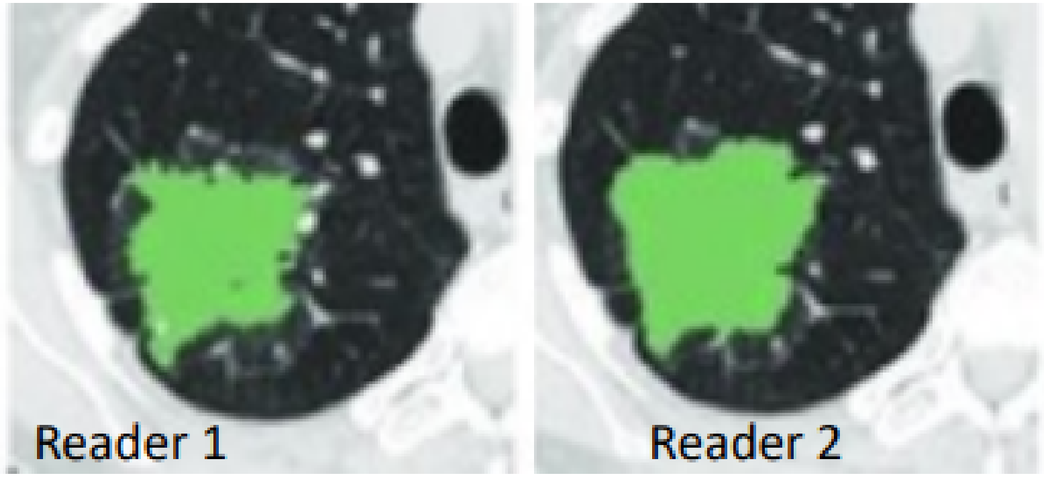
- Disponibilità di un numero di dati ridotto
- Eterogeneità dei dati
- Scarsa affidabilità dei sistemi basati su AI
- Ridotta capacità di spiegare i risultati ottenuti



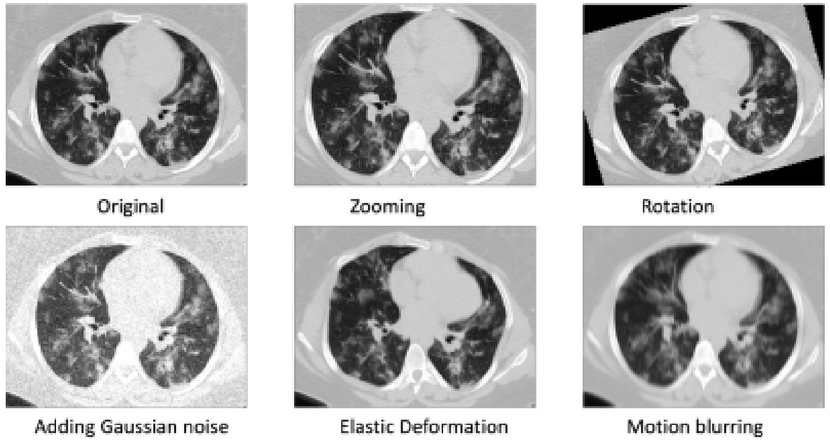
Ridotto numero di dati

I dati disponibili in questo ambito sono generalmente inferiori rispetto a quelli disponibili per altri studi. Inoltre per potervi assegnare delle label spesso è necessario un lungo lavoro, svolto da personale qualificato.

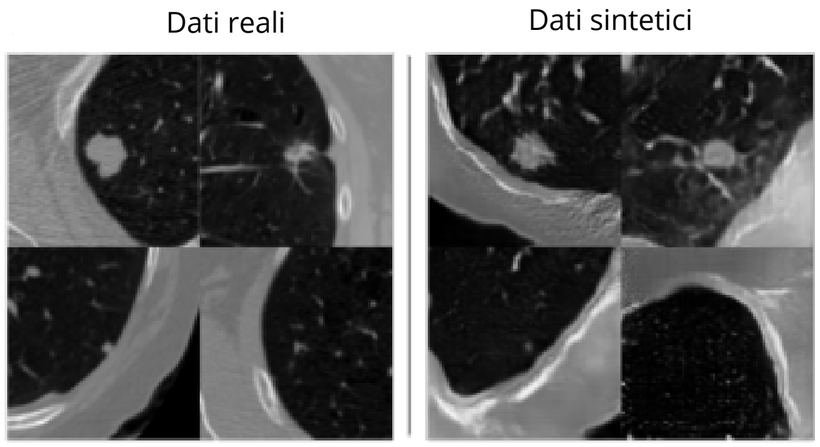
Questo lavoro è spesso soggetto ad una certa variabilità inter- e intrapersonale.



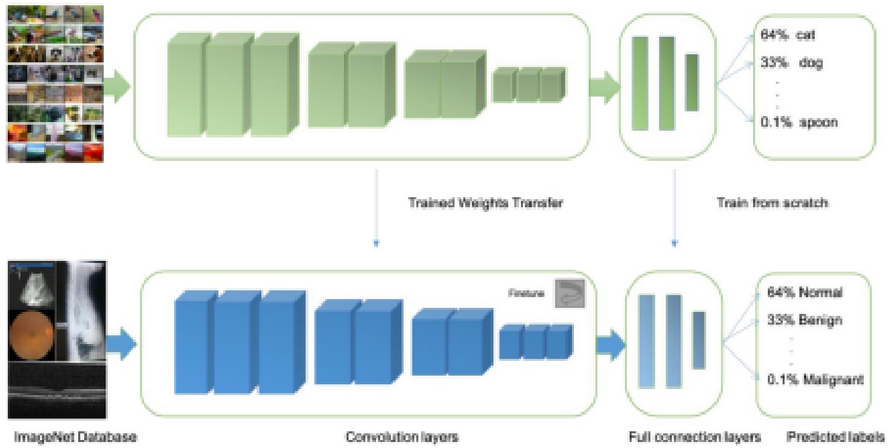
Ridotto numero di dati: possibili soluzioni



Data augmentation con tecniche tradizionali



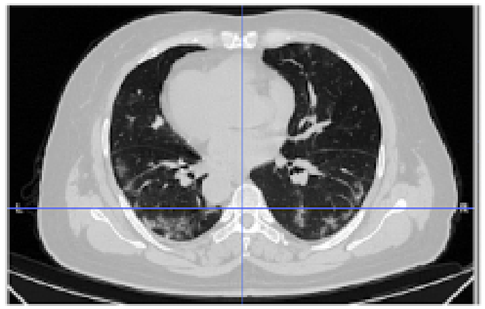
Data augmentation tramite creazione di dati sintetici

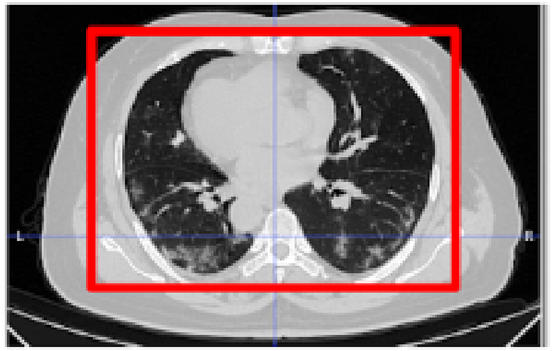


Transfered Learning

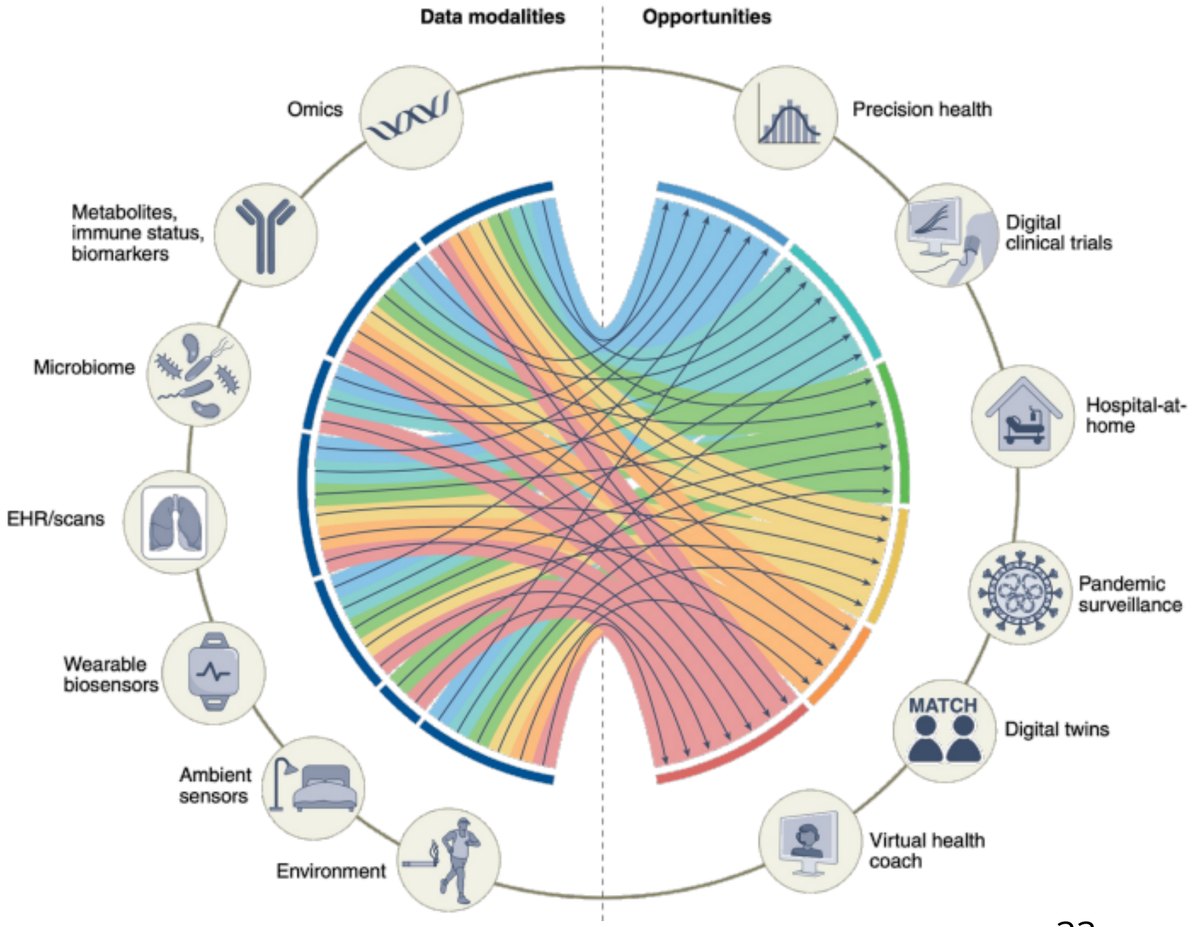
Eterogeneità dei dati

Stessa tipologia di dati acquisiti in modo diverso

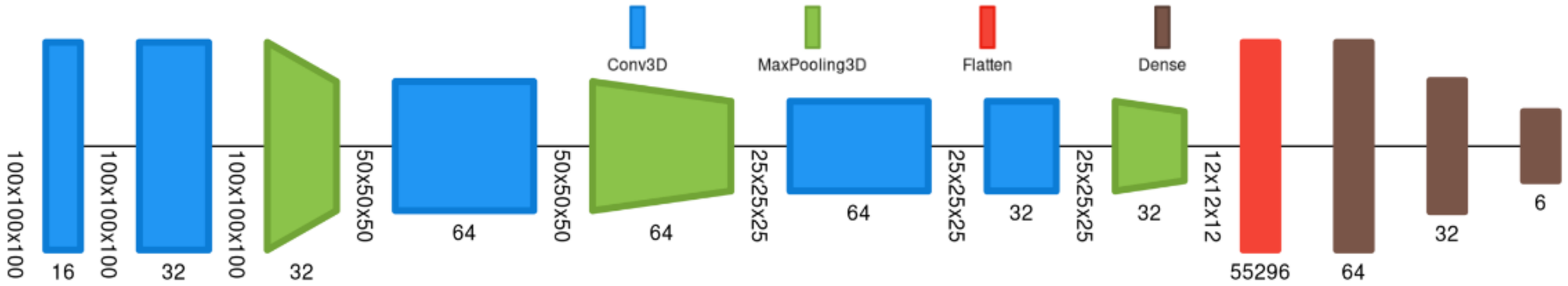




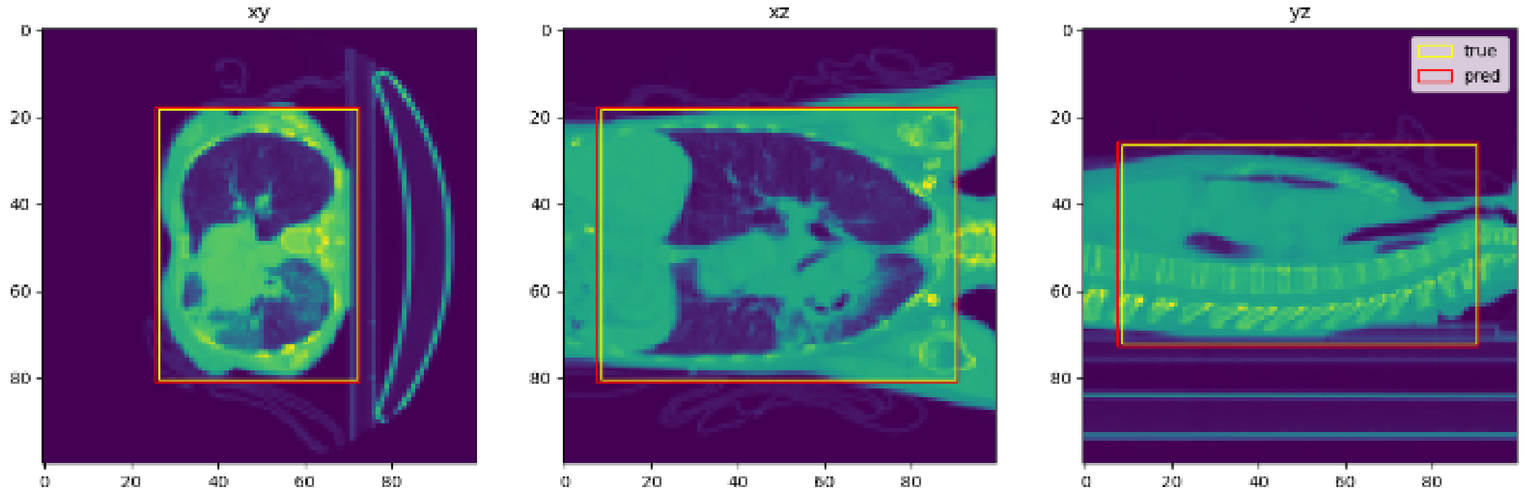
Tipologie di dati differenti



Standardizzare i dati

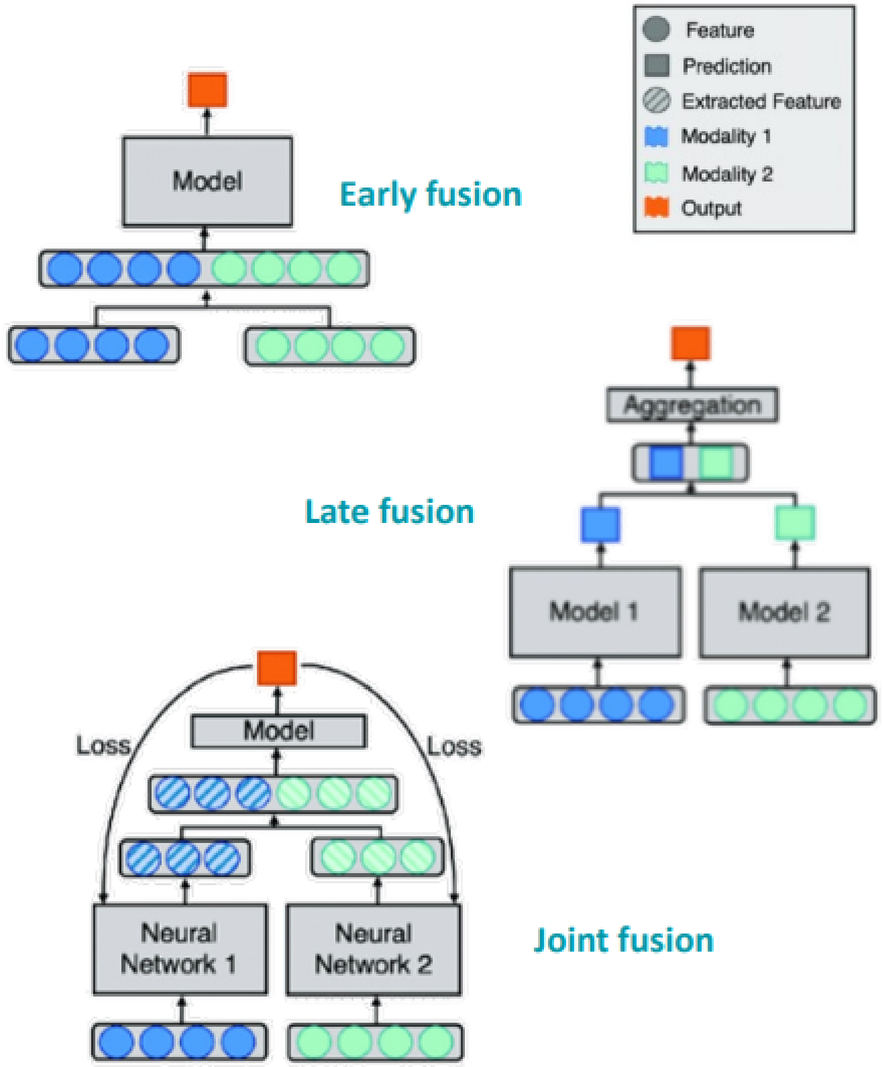


CNN semplice con 6 neuroni nell'ultimo layer per predire le 6 coordinate di 2 punti che definiscono univocamente un parallelepipedo



Apprendimento multimodale

- **Early fusion:** i dati di input sono concatenati prima di essere passati ad un modello
- **Late fusion:** i dati sono usati per il training di modelli diversi. le probabilità di output sono aggregate alla fine.
- **Joint fusion:** si utilizzano delle NN per estrarre le informazioni dai dati di input. Queste vengono concatenate e passate ad un modello che fornisce un output.

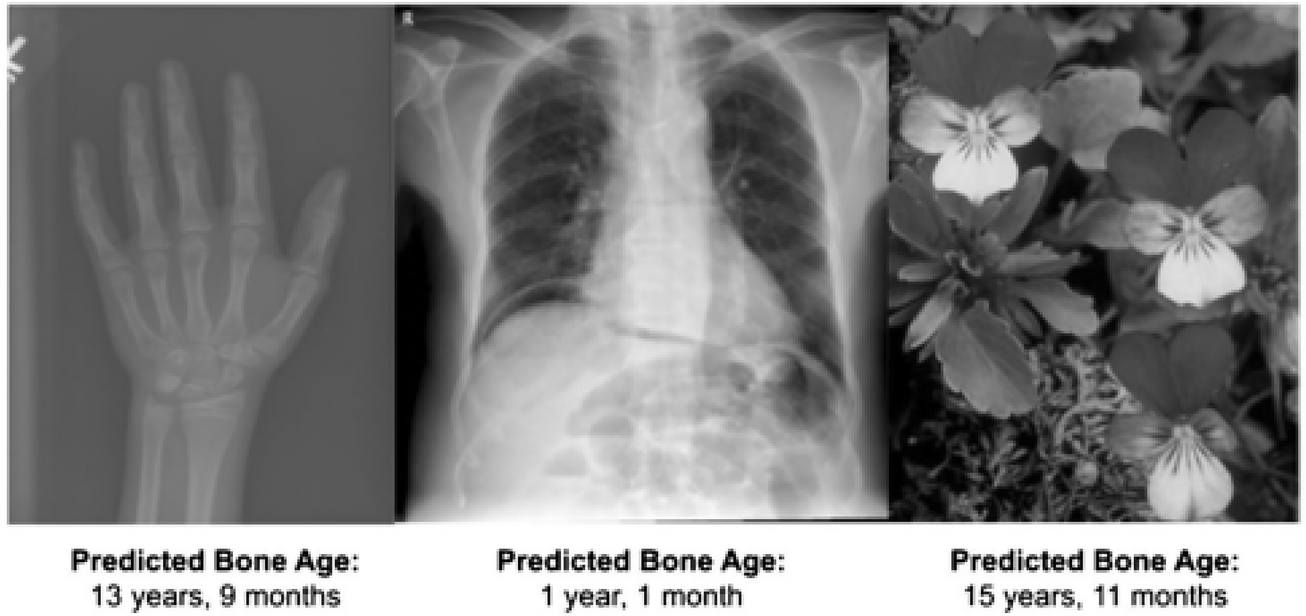


Affidabilità dei modelli

Quando un modello addestrato ad uno specifico task, viene eseguito fornendogli dati diversi da quelli di training, fornisce comunque una risposta.

Per evitare che ciò accada si possono usare altri modelli che scartano i dati non conformi a quelli attesi

Output di una CNN addestrata a predire l'età delle ossa a partire da una radiografia del polso sinistro

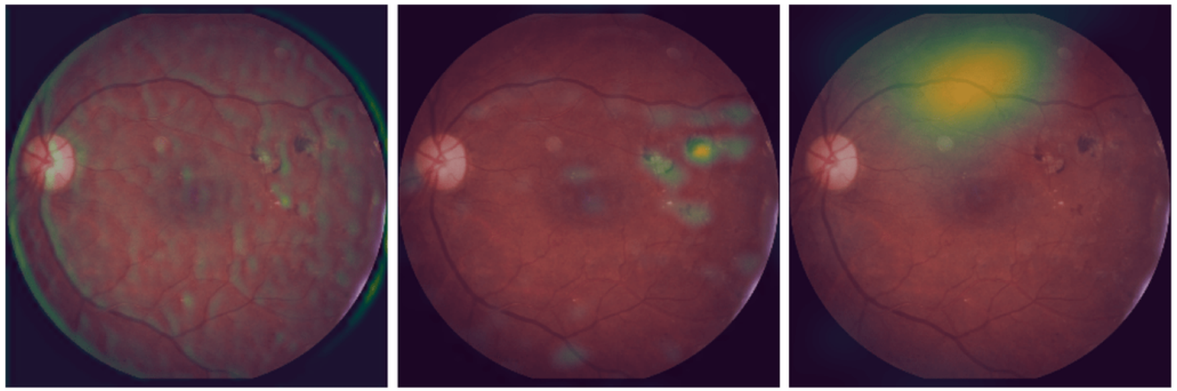


Rendere i risultati comprensibili (XAI)

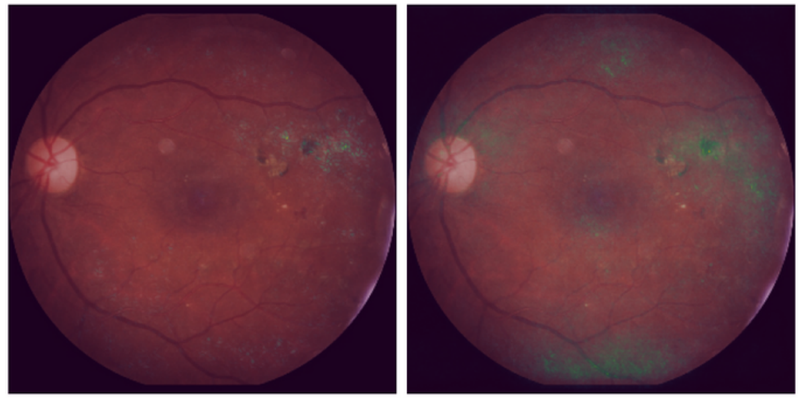
Spesso i modelli risultano delle black box che forniscono risposte poco comprensibili. Questo non solo diminuisce la fiducia degli utenti ma rende anche più complesse le eventuali operazioni atte a migliorare il modello.

Rendere i risultati comprensibili (XAI)

Visualisation of convolutions

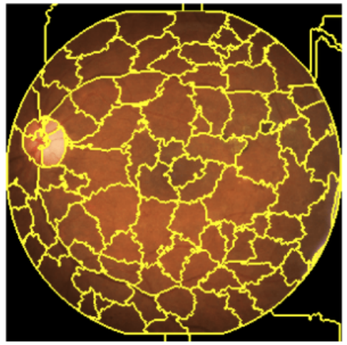


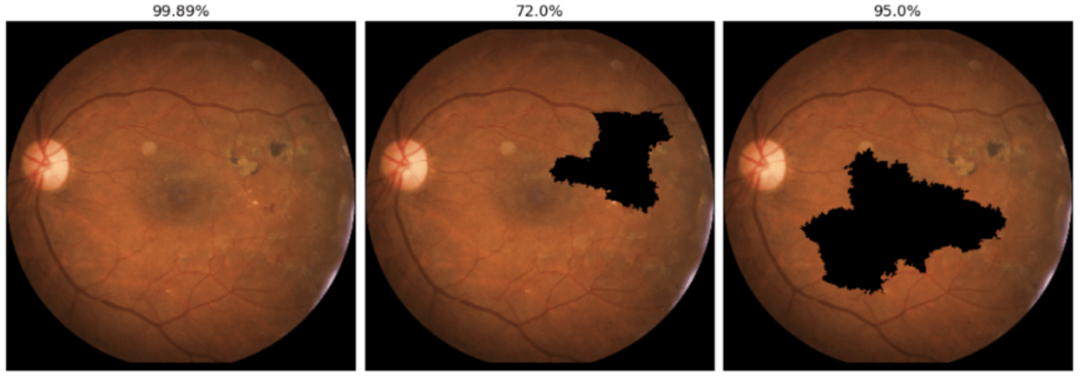
Saliency Map

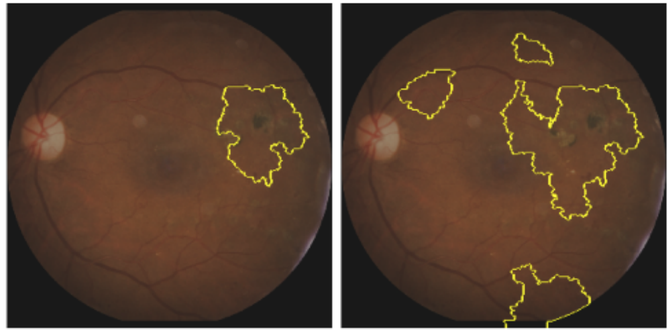


Vanilla saliency SmoothGrad

Local Interpretable Modelagnostic Explanations method (LIME)







Considerazioni finali

Aspetti positivi

- Possibilità di conoscere e confrontarsi con altri ricercatori
- Approfondire argomenti spesso trattati superficialmente
- Applicare le conoscenze apprese a casi d’uso specifici

Suggerimenti

- Organizzare più sessioni di hands-on
- Aggiungere seminari relativi al reinforcement learning
- Mostrare tecniche di hyperparameter tuning

Grazie per l’attenzione