- Abbiamo creato due modelli leggermente modificati, uno con un transformer tra encoder e decoder e uno che utilizza una resnet come encoder
- Abbiamo poi testato i vari modelli introducendo un layer di classificazione tra encoder e decoder
- I dati riportati sono solo in riferimento ad alcuni dei casi testati e abbiamo evitato di riportarli tutti vista la scarsità dei risultati ottenuti

SSIM Nuovi Modelli

Modello con trasformer	Modello Resnet	Doppia loss base	Doppia loss trasformer
0.668 (std 0.15)	0.557 (std 0.11)	0.415 (std 0.17)	0.08 (std 0.07)

Modello con 7 strati convoluzionali (8,16,32,64,128,256,512) e 64 features

Risultati della classificazione utilizzando una rete neurale custom:

F1 score: 0.307
PR AUC: 0.318
ROC AUC: 0.433
Accuracy: 0.64
Confusion Matrix:
[[14 3]
 [6 2]]
Best Precision: 0.4
Best Recall: 0.25

Risultati della classificazione utilizzando machine learning:

Miglior risultato con classificatore MLP
Selector p_value and mode majority voting

TEST:
F1 score finale: 0.4545
Precision finale: 0.357
Recall finale: 0.625
Confusion Matrix finale:
[[8 9]
 [3 5]]
Accuracy finale: 0.52

Modello come precedente + Trasformer

Risultati della classificazione utilizzando una rete neurale custom:

F1 score: 0.38
PR AUC: 0.31
ROC AUC: 0.47
Accuracy: 0.48
Confusion Matrix:
[[8 9]
 [4 4]]
Best Precision: 0.31
Best Recall: 0.5

Risultati della classificazione utilizzando machine learning:

Miglior risultato con classificatore RandomForest Selector p_value and mode mean

F1 score finale: 0.28
Precision finale: 0.33
Recall finale: 0.25
Confusion Matrix finale:
[[13 4]
 [6 2]]
Accuracy finale: 0.6
threshold: 0.498
features: [0 1 2] e sono 3

Modello come precedente + Trasformer + Double Loss

Risultati della classificazione utilizzando una rete neurale custom:

F1 score: 0.17
PR AUC: 0.31
ROC AUC: 0.42
Accuracy: 0.6
Confusion Matrix:
[[14 3]
 [7 1]]
Best Precision: 0.25
Best Recall: 0.125

Risultati della classificazione utilizzando machine learning:

Miglior risultato con classificatore XgBoost
Selector mrmr and mode mean

F1 score finale: 0.4
Precision finale: 0.43
Recall finale: 0.375
Confusion Matrix finale:
[[13 4]
 [5 3]]
Accuracy finale: 0.64
threshold: 0.497
features: [0, 13] e sono 2

Modello Resnet

Risultati della classificazione utilizzando una rete neurale custom:

F1 score: 0.36
PR AUC: 0.3
ROC AUC: 0.44
Accuracy: 0.44
Confusion Matrix:
[[7 10]
 [4 4]]
Best Precision: 0.29
Best Recall: 0.5

Risultati della classificazione utilizzando machine learning:

Miglior risultato con classificatore RandomForest Selector p_value and mode mean

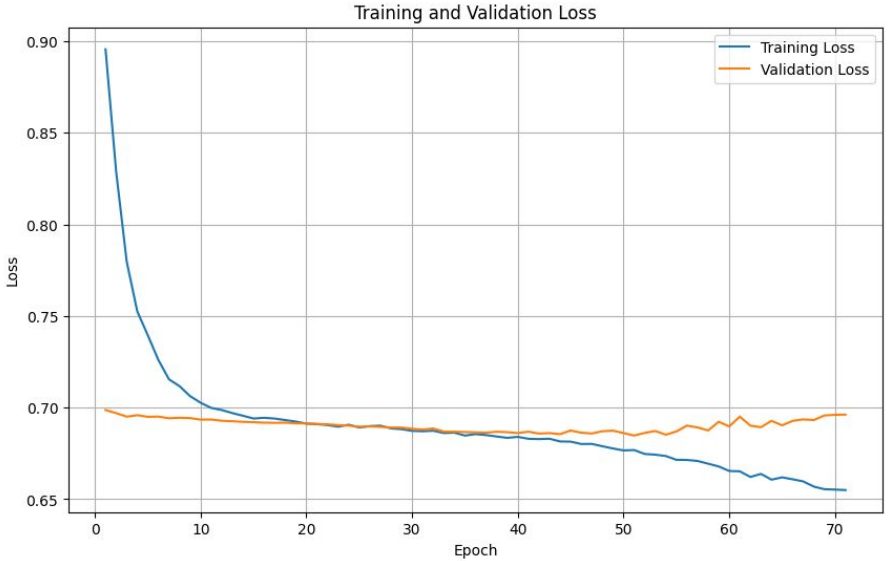
F1 score finale: 0.2857142857142857
Precision finale: 0.3333333333333333
Recall finale: 0.25
Confusion Matrix finale:
[[13 4]
 [6 2]]
Accuracy finale: 0.6
threshold: 0.498
features: [0 1 2] e sono 3

Tentativi senza ottenere miglioramenti:

- Abbiamo tentato varie reti cambiando il numero di strati, più semplici e più complesse, aggiunto regularizers
- Aggiungendo qualche layer di Dropout all'encoder l'accuracy della classificazione migliora leggermente ma le ricostruzioni peggiorano molto
- In generale aggiungere la doppia Loss porta il modello a overfittare e le ricostruzioni diventano dei semplici cerchi bianchi
- Aggiungendo la rete neurale come classificatore abbiamo visto come si comportano le loss durante l'allenamento e abbiamo riscontrato due casi: o si verifica overfitting molto velocemente, oppure entrambe le loss convergono a un valore di bce di 0.69 circa che corrisponde a un classificatore casuale
- abbiamo provato a aumentare maggiormente le immagini per bilanciare il dataset invece di usare smote ma ciò non ha aiutato le classificazione

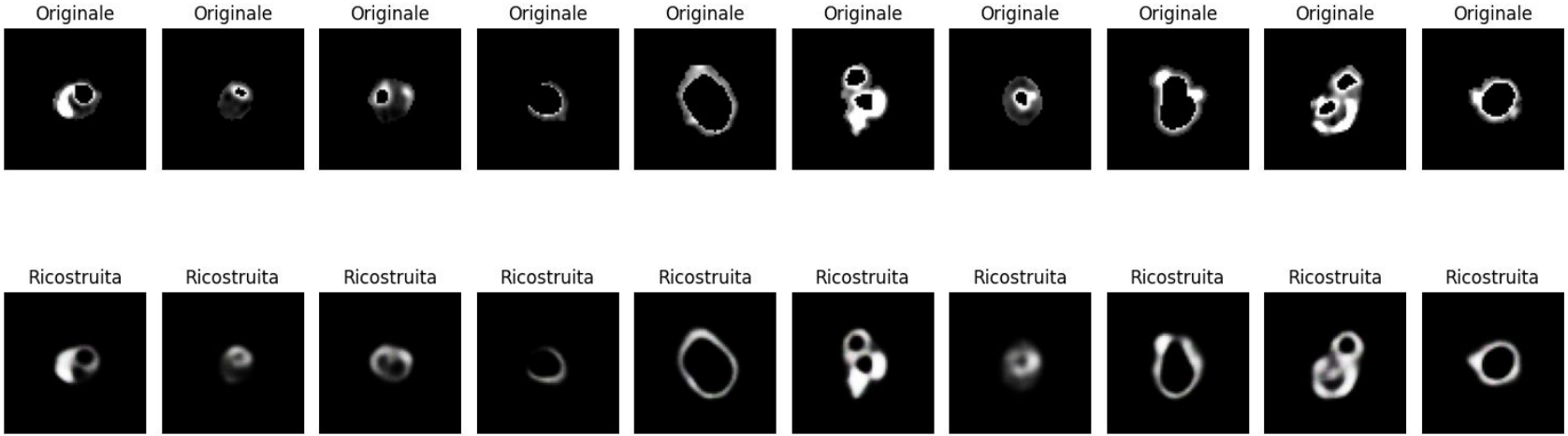
- Anche creando una rete neurale con qualche strato convoluzionale che prende in input le immagini e ha l'obiettivo unico di classificarle, comunque non riesce a classificare correttamente le immagini di test

F1 score: 0.434782608695652 16
PR AUC: 0.37096463585434175
ROC AUC: 0.5735294117647058
Accuracy: 0.48
Confusion Matrix:
[[7 10]
 [3 5]]
Best Precision:
0.3333333333333333
Best Recall: 0.625



```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Conv2DTranspose, Dense, Reshape, Input, UpSampling2D, Conv2D
from tensorflow.keras.models import Model
import tensorflow as tf

# 1. Encoder (ResNet50 pre-addestrata su ImageNet, modificata per 64x64x1 input)
def build_encoder():
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(64, 64, 3))
    encoder = Model(inputs=base_model.input, outputs=base_model.layers[-1].output)
    return encoder

# 2. Decoder (personalizzato per 64x64x1)
def build_decoder():
    decoder_input = Input(shape=(2, 2, 2048))  # Output del ResNet bottleneck
    x = Conv2DTranspose(512, kernel_size=3, strides=2, padding='same', activation='relu')(decoder_input)
    x = Conv2DTranspose(256, kernel_size=3, strides=2, padding='same', activation='relu')(x)
    x = Conv2DTranspose(128, kernel_size=3, strides=2, padding='same', activation='relu')(x)
    x = Conv2DTranspose(64, kernel_size=3, strides=2, padding='same', activation='relu')(x)
    x = Conv2DTranspose(3, kernel_size=3, strides=2, padding='same', activation='sigmoid')(x)  # Immagine finale 64x64x1
    decoder = Model(inputs=decoder_input, outputs=x)
    return decoder
```

```
# 3. Autoencoder
def build_autoencoder():
    encoder = build_encoder()

    # *Aggiungi qui il congelamento dei layer dell'encoder*
    #for layer in encoder.layers:
        #layer.trainable = False  # Congela tutti i layer dell'encoder

    # Se vuoi fare fine-tuning di alcuni strati dell'encoder (ad esempio gli ultimi):
    for layer in encoder.layers[:140]:  # Congela i primi 140 strati
        layer.trainable = False
    for layer in encoder.layers[140:]:  # Mantieni allenabili gli ultimi strati
        layer.trainable = True

    decoder = build_decoder()

    input_img = Input(shape=(64, 64, 1))
    encoded = encoder(input_img)
    decoded = decoder(encoded)

    autoencoder = Model(inputs=input_img, outputs=decoded)
    return autoencoder

autoencoder_model = build_autoencoder()
autoencoder_model.compile(optimizer=AdamW(learning_rate=1e-4), loss='binary_crossentropy')
autoencoder_model.summary()
```


Originale
Originale
Originale
Originale
Originale
Originale
Originale
Originale
Originale
Originale
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita
Ricostruita

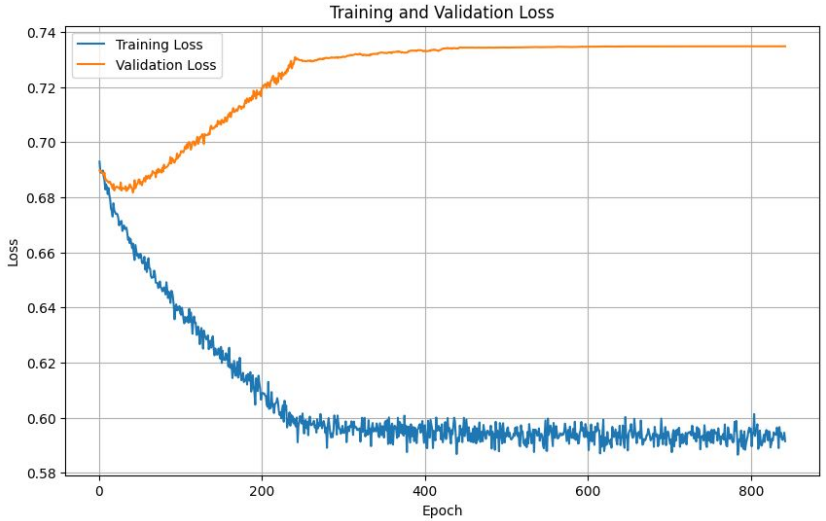
RISULTATI classificazione ottenuta attraverso una rete neurale semplice

```
def create_mlp(input_shape):
    model = models.Sequential()

    model.add(layers.Dense(64, input_shape=(input_shape,)))
    model.add(layers.LeakyReLU(alpha=0.01))
    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

osservazione: funziona meglio semplice piuttosto che con piu strati

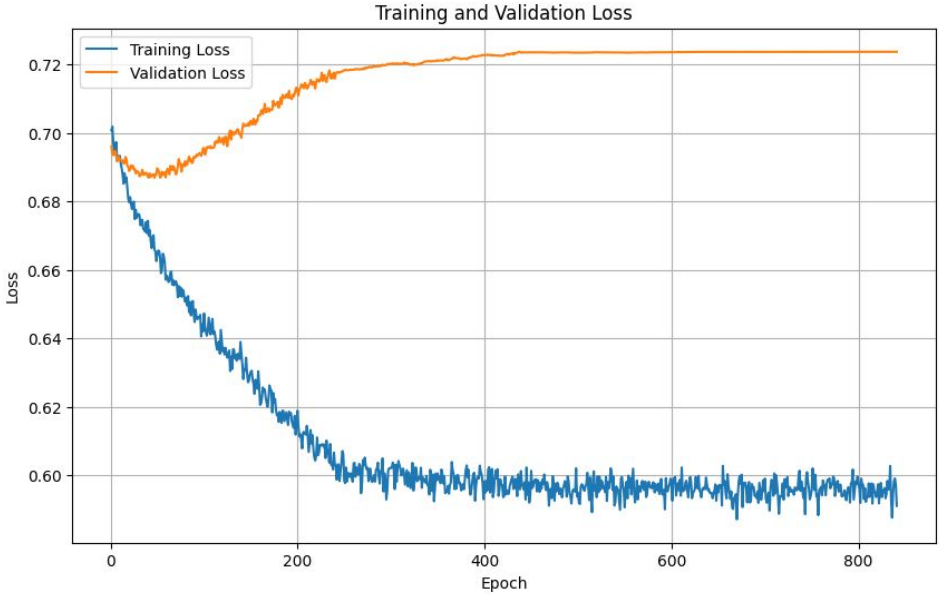
modello 1

```python
def encoder(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(inputs)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = GlobalAveragePooling2D()(x)  # Global Average Pooling
    x = Flatten()(x)
    encoded = Dense(128, activation='relu', kernel_regularizer=l1(10e-8))(x)
    encoder_model = Model(inputs, encoded, name='encoder')
    return encoder_model
```



F1 score: 0.23529411764705882
PR AUC: 0.3123673293081188
ROC AUC: 0.4779411764705882
Accuracy: 0.48
Confusion Matrix:
[[10 7]
[6 2]]
Best Precision: 0.2222222222222222
Best Recall: 0.25

modello 2 Resnet50




```
F1 score: 0.36363636363636365
PR AUC: 0.29774757406336355
ROC AUC: 0.4411764705882353
Accuracy: 0.44
Confusion Matrix:
[[ 7 10]
 [ 4  4]]
Best Precision: 0.2857142857142857
Best Recall: 0.5
```

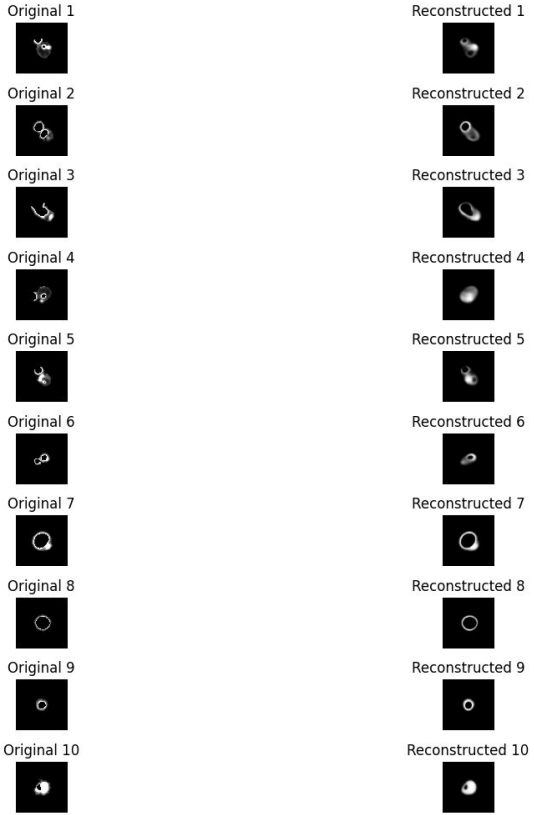
modello 3 trasformer

```
def combined_loss(y_true, y_pred):
    ssim_loss = 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=0))
    mse_loss = mean_squared_error(y_true, y_pred)
    bce_loss = tf.keras.losses.binary_crossentropy(y_true, y_pred)
    return bce_loss + 0.5 * mse_loss + 0.5 * ssim_loss

def transformer_block(x, num_heads, ff_dim, rate=2):
    attn_output = MultiHeadAttention(num_heads=num_heads, key_dim=x.shape[-1])(x, x)
    attn_output = Dropout(rate)(attn_output)
    out1 = Add()([x, attn_output])
    out1 = LayerNormalization(epsilon=1e-6)(out1)

    ff_output = Dense(ff_dim, activation='relu')(out1)
    ff_output = Dense(x.shape[-1])(ff_output)
    ff_output = Dropout(rate)(ff_output)


    out2 = Add()([out1, ff_output])
    return LayerNormalization(epsilon=1e-6)(out2)
```

```
def encoder(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(inputs)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(2, 2), padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    x = Flatten()(x)

    encoded = Dense(64, activation='relu', kernel_regularizer=l1(10e-6))(x)
    encoder_model = Model(inputs, encoded, name='encoder')
    return encoder_model

input_shape = (64, 64, 1)
encoder_model = encoder(input_shape)

encoded_input = encoder_model.output
x = Reshape((1, 64))(encoded_input)
x = transformer_block(x, num_heads=8, ff_dim=512)
x = Flatten()(x)
```

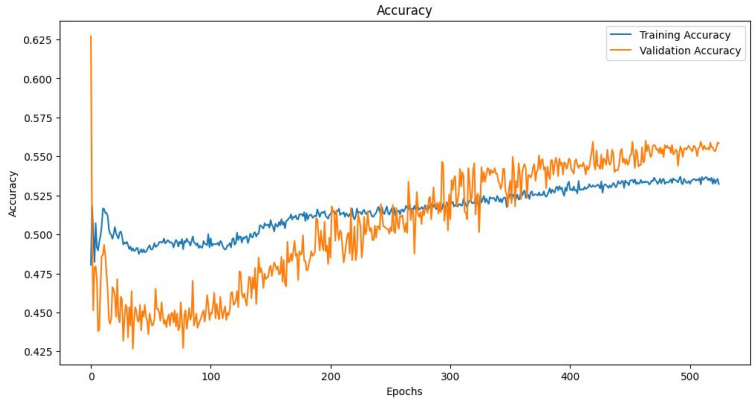
DIMENSIONI DATI PER IL TRAINING
Dimensione del vettore delle immagini aumentate TRAIN: (17165, 64, 64, 1) classe 0 9515 e classe 1 7650
Dimensione del vettore delle immagini aumentate VAL : (1952, 64, 64, 1) classe 0 1224 e classe 1 728

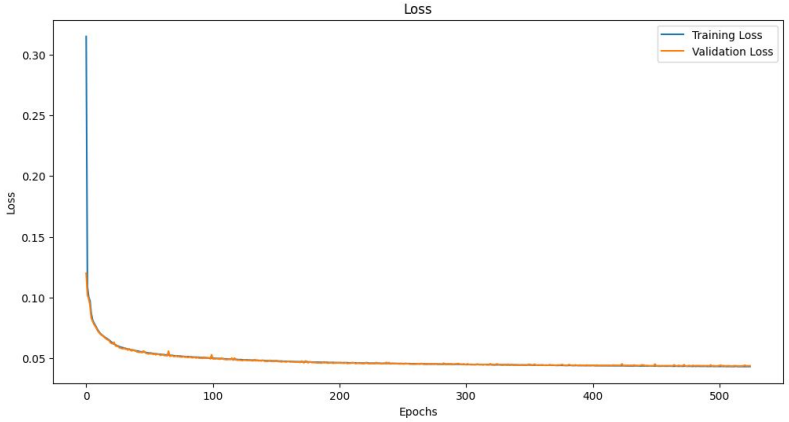
modello senza doppia loss

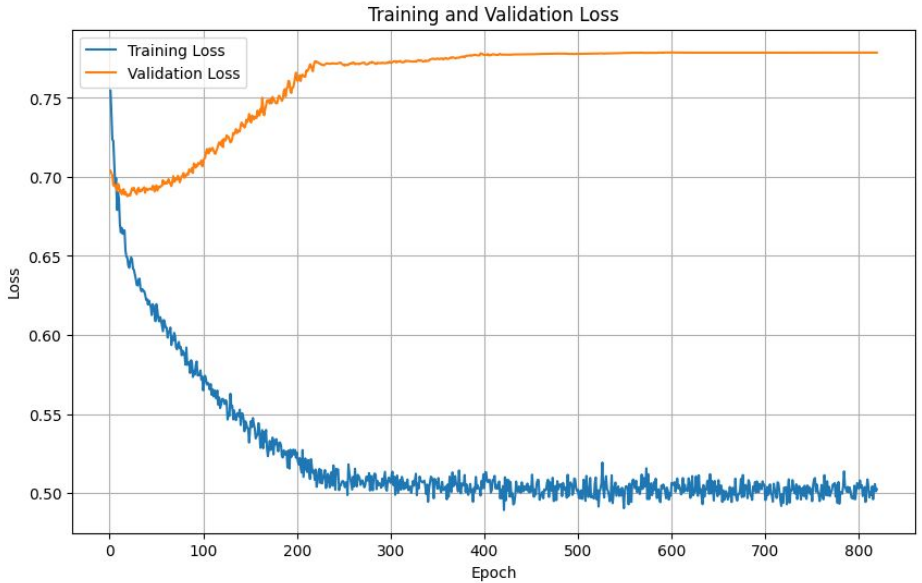


Valore medio di SSIM con maschera: 0.6680924245758035
Deviazione standard di SSIM: 0.152324523339985

Mean SSIM on Test Set: 0.93598668646
Std SSIM on Test Set: 0.0373785488401








```
F1 score: 0.38095238095238093
PR AUC: 0.31058455433455434
ROC AUC: 0.47058823529411764
Accuracy: 0.48
Confusion Matrix:
[[8 9]
 [4 4]]
Best Precision: 0.3076923076923077
Best Recall: 0.5
```

modello 4 con doppia loss (uguale al 3 ma doppia loss)

osservazioni: encoder estrae 64 features, facendo feature correlation con soglia a 0.9, lascia solo 9 features.

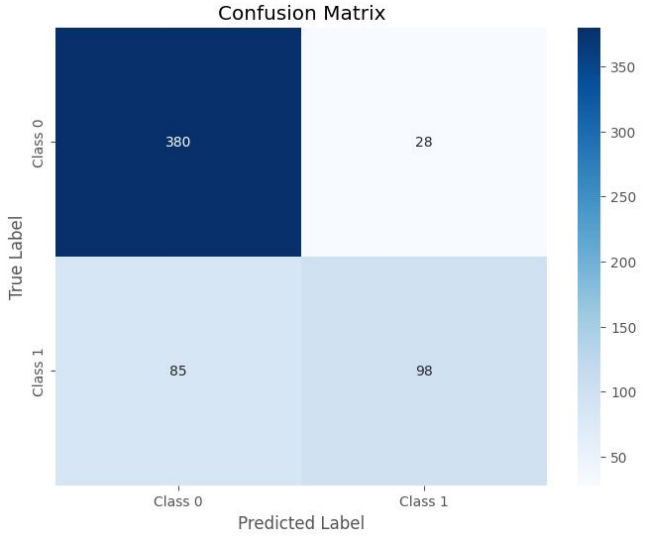
Double Loss

```
# Loss Classificazione
bce_loss = tf.keras.losses.BinaryCrossentropy()(stored_values['y_true_class'], stored_values['y_pred_class'])

# Loss  decoder

mse_loss = tf.keras.losses.MeanSquaredError()(stored_values['y_true_recon'], stored_values['y_pred_recon'])
bce_loss_re = tf.keras.losses.BinaryCrossentropy()(stored_values['y_true_recon'], stored_values['y_true_recon']), max_val=1.0))

dec_loss= lambdadec * mse_loss + (1-lambdadec) * bce_loss_re


combined_loss_value = lambdatot * dec_loss + (1-lambdatot) * bce_loss



    # Compilazione del modello
    autoencoder_classifier.compile(optimizer=Adam(learning_rate=lr_schedule),
loss=create_combined_loss(0.5, 0.4), metrics={'classifier': 'accuracy'})
```

più aggiungiamo un classificatore fatto di un solo strato denso

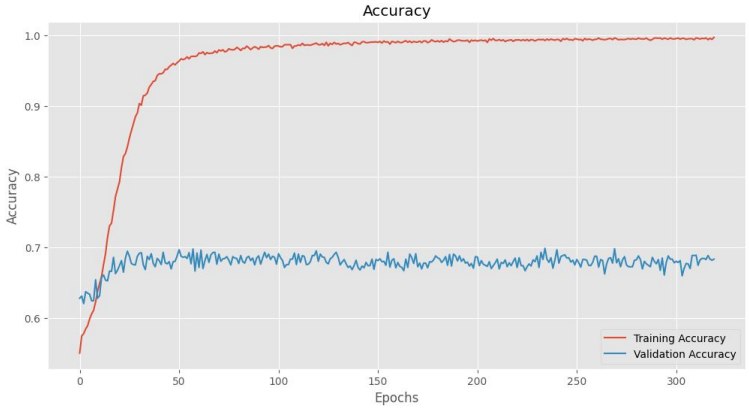
I risultati dei modelli 4, 5 e 6 sono ottenuti tutti con (0.5, 0.4)

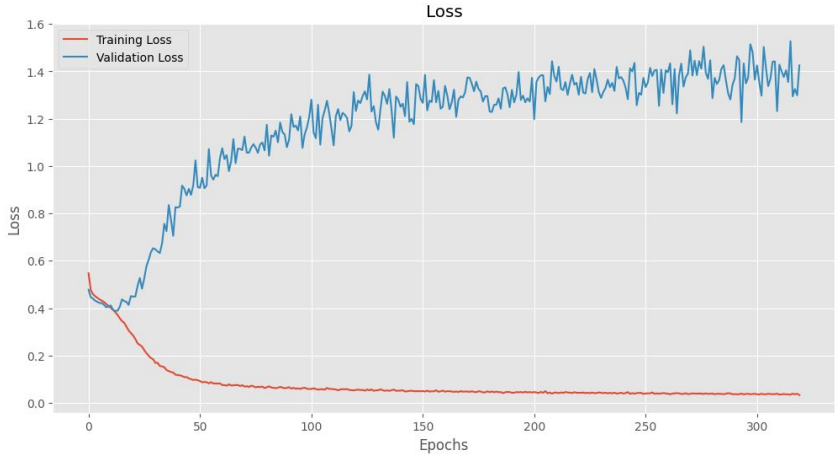
TRAINING AUTOENCODER MODELLO 4



Classification Report:

	precision	recall	f1-score	support
Class 0	0.82	0.93	0.87	408
Class 1	0.78	0.54	0.63	183
accuracy			0.81	591
macro avg	0.80	0.73	0.75	591
weighted avg	0.80	0.81	0.80	591





usando i modelli di machine learning sul modello 4
(senza effettuare feature correlation e selection via p-value)

Miglior risultato per il classificatore
XgBoost
Selector mrmr and mode mean

VALIDATION:
F1 score: 0.6956521739130435
PR AUC: 0.6158600077468395
ROC AUC: 0.7033492822966507
Accuracy: 0.7666666666666667
Numero di features: 2
Confusion Matrix:
[[15 4]
 [3 8]]
Best Precision: 0.6666666666666666
Best Recall: 0.7272727272727273
Best Threshold: 0.497
Selected Features: [0, 13]

TEST:
PR AUC: 0.3857477226720648
ROC AUC: 0.4779411764705882
F1: 0.25
Precision: 0.25
Recall: 0.25
Accuracy: 0.52
Confusion Matrix:
[[11 6]
 [6 2]]

ON TEST WITB TRAINING WITH VAL INCLUDED:
F1 score finale: 0.4
Precision finale: 0.42857142857142855
Recall finale: 0.375
Confusion Matrix finale:
[[13 4]
 [5 3]]
Accuracy finale: 0.64
threshold: 0.497
features: [0, 13] e sono 2

Original 1

Reconstructed 1

Original 2

Reconstructed 2

Original 3

Reconstructed 3

Original 4

Reconstructed 4

Original 5

Reconstructed 5

Mean SSIM on Test Set: 0.6604808242866078

Original 6

Reconstructed 6

Original 7

Reconstructed 7

Original 8

Reconstructed 8

Original 9

Reconstructed 9

Original 10

Reconstructed 10

usando i modelli di machine learning sul modello 4
(senza effettuare feature correlation e selection via p-value)

Miglior risultato per il classificatore
XgBoost
Selector mrmr and mode mean

VALIDATION:
F1 score: 0.6956521739130435
PR AUC: 0.6158600077468395
ROC AUC: 0.7033492822966507
Accuracy: 0.7666666666666667
Numero di features: 2
Confusion Matrix:
[[15 4]
 [3 8]]
Best Precision: 0.6666666666666666
Best Recall: 0.7272727272727273
Best Threshold: 0.497
Selected Features: [0, 13]

TEST:
PR AUC: 0.3857477226720648
ROC AUC: 0.4779411764705882
F1: 0.25
Precision: 0.25
Recall: 0.25
Accuracy: 0.52
Confusion Matrix:
[[11 6]
 [6 2]]

ON TEST WITB TRAINING WITH VAL INCLUDED:
F1 score finale: 0.4
Precision finale: 0.42857142857142855
Recall finale: 0.375
Confusion Matrix finale:
[[13 4]
 [5 3]]
Accuracy finale: 0.64
threshold: 0.497
features: [0, 13] e sono 2

```
def create_encoder(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.001))(inputs)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Flatten()(x)
    encoded = Dense(64, activation='relu', kernel_regularizer=l2(0.001))(x)
    return Model(inputs, encoded, name="encoder")

# Funzione per creare il decoder
def create_decoder(encoded_shape):
    inputs = Input(shape=encoded_shape)
    x = Dense(8 * 8 * 128, activation='relu', kernel_regularizer=l2(0.001))(inputs)
    x = Reshape((8, 8, 128))(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same',
kernel_regularizer=l2(0.001))(x)
    return Model(inputs, decoded, name="decoder")
```

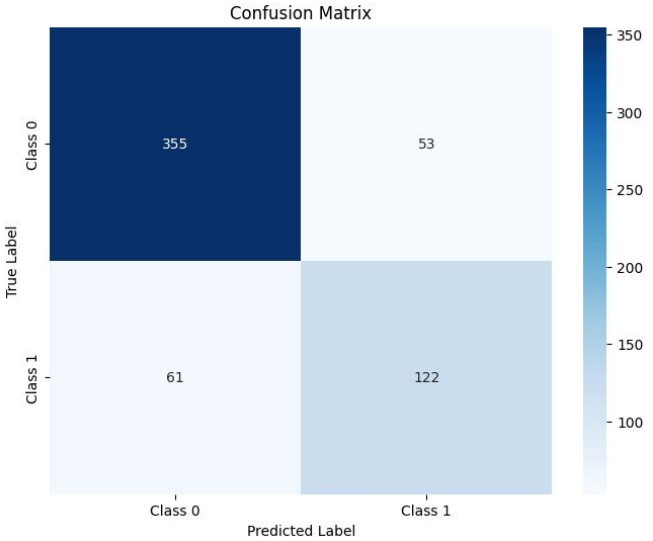
modello 5

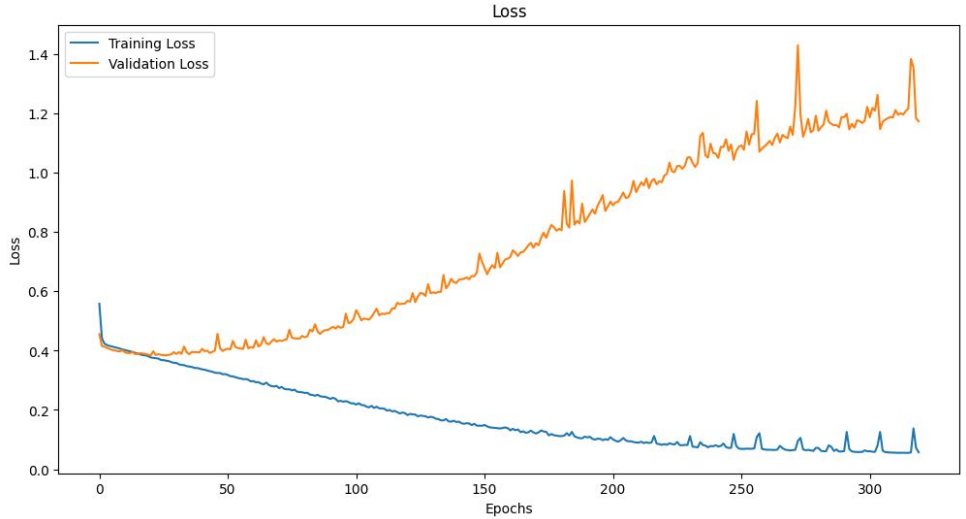
(stessa loss e stessi parametri del modello 4) - 64 features

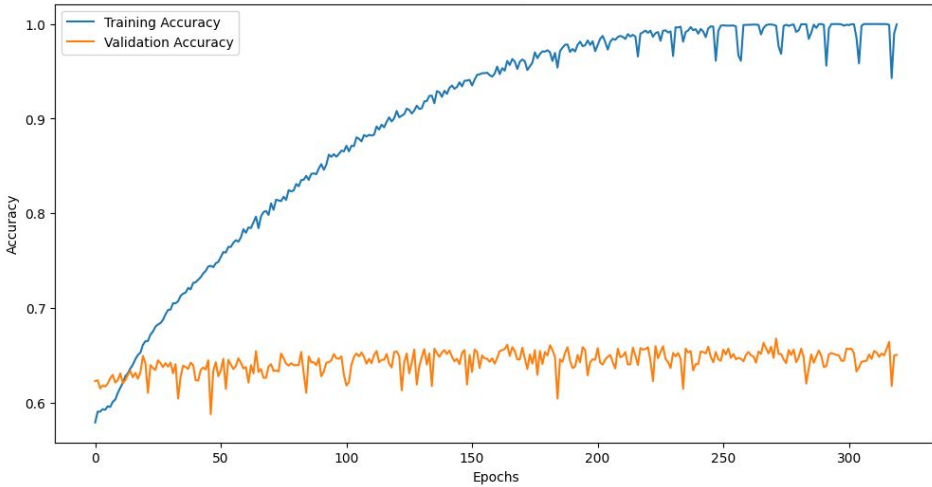
osservazioni: anche addestrando l'AE avendo lambda=1 per la classificazione, comunque classifica male

Classification Report:

	precision	recall	f1-score	support
Class 0	0.85	0.87	0.86	408
Class 1	0.70	0.67	0.68	183
accuracy			0.81	591
macro avg	0.78	0.77	0.77	591
weighted avg	0.80	0.81	0.81	591



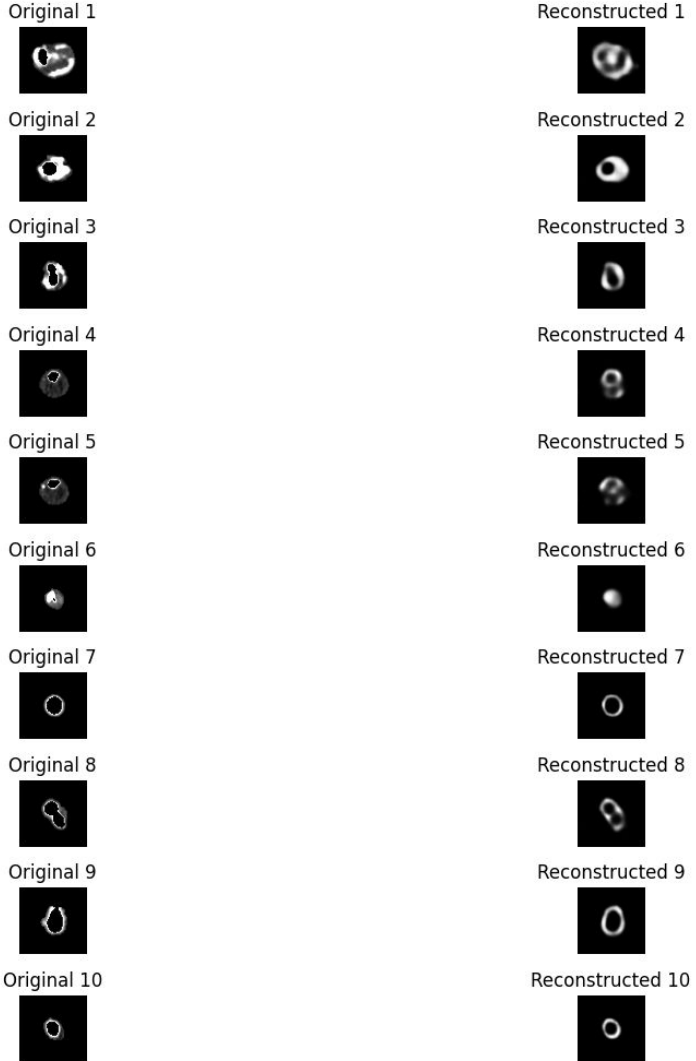




osservazioni: le features che trova sono meno correlate (0.9 lascia 58 features, poi p-value 22)
Ci sono tante features a 0

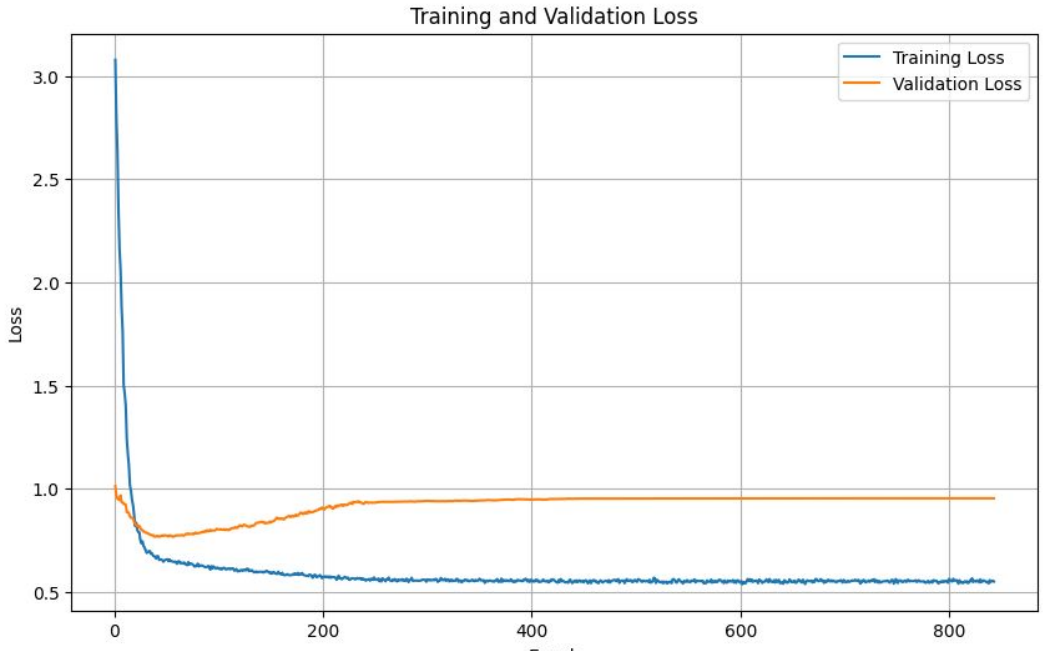
Mean SSIM on Test Set: 0.9009016712135512

Valore medio di SSIM con maschera: 0.4840349061795918
Deviazione standard di SSIM: 0.1298063562839805



classificazione con rete neurale

```
F1 score: 0.47058823529411764
PR AUC: 0.44659720075321097
ROC AUC: 0.6029411764705883
Accuracy: 0.64
Confusion Matrix:
[[12  5]
 [ 4  4]]
Best Precision: 0.4444444444444444
Best Recall: 0.5
```



classificazione con machine learning

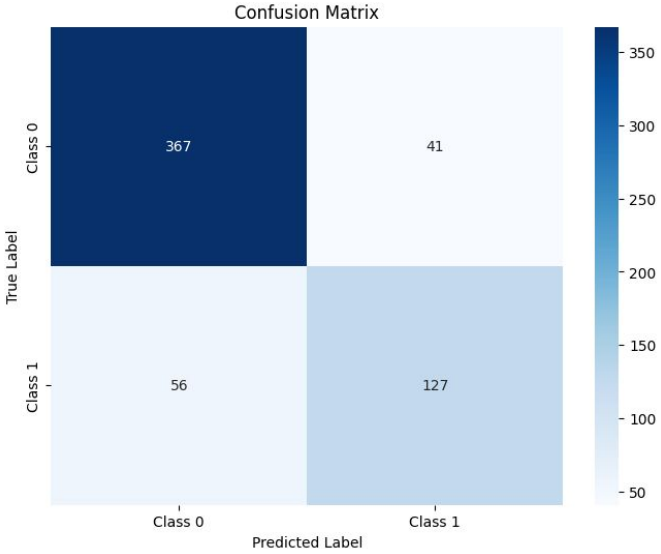
Miglior risultato per il classificatore MLP
Selector p_value and mode majority voting

VALIDATION:
F1 score: 0.6666666666666666
PR AUC: 0.5036510939266112
ROC AUC: 0.6220095693779903
Accuracy: 0.6666666666666666
Numero di features: 7
Confusion Matrix:
[[10 9]
 [1 10]]
Best Precision: 0.5263157894736842
Best Recall: 0.9090909090909091
Best Threshold: 0.47900000000000004
Selected Features: [0 1 2 3 4 5 6]

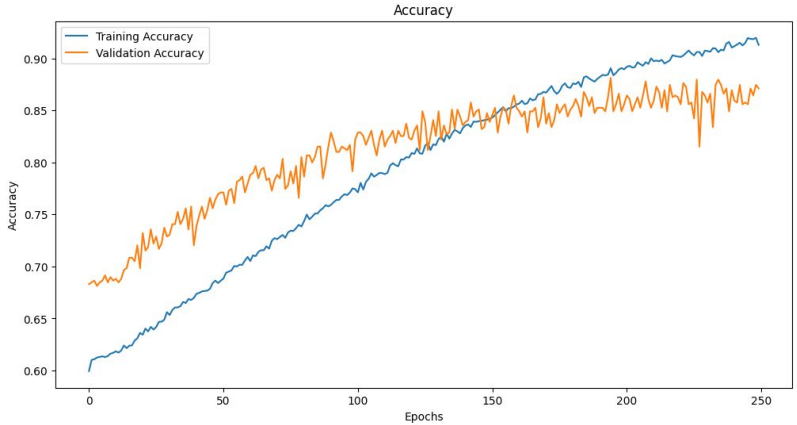
TEST:
PR AUC: 0.40555225713473797
ROC AUC: 0.4779411764705882
F1: 0.3157894736842105
Precision: 0.2727272727272727
Recall: 0.375
Accuracy: 0.48
Confusion Matrix:
[[9 8]
 [5 3]]

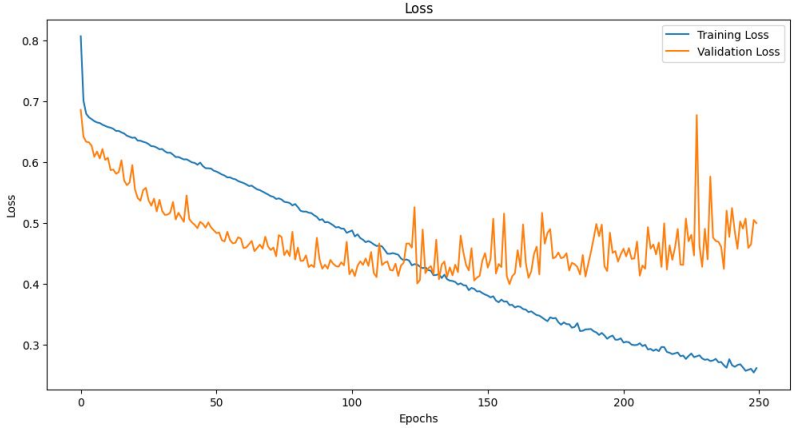
ON TEST WITB TRAINING WITH VAL INCLUDED:

F1 score finale: 0.36363636363636365
Precision finale: 0.2857142857142857
Recall finale: 0.5
Confusion Matrix finale:
[[7 10]
 [4 4]]
Accuracy finale: 0.44
threshold: 0.47900000000000004
features: [0 1 2 3 4 5 6] e sono 7



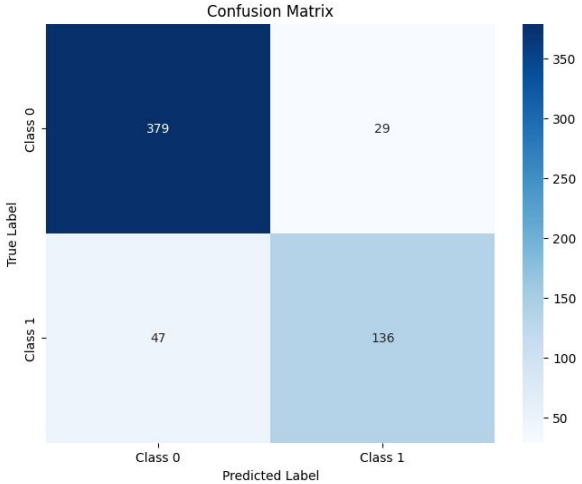
nb: al classificatore servono dei layer di **Dropout**.
questo infatti è il grafico dando lambda=1 al classificatore ma le ricostruzioni sono completamente nere





modello 5 + 3 strati di Dropout nell’encoder

(risultati senza aumentare il validation)

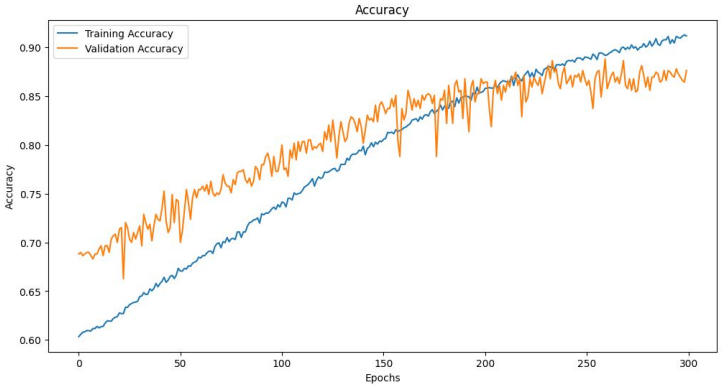


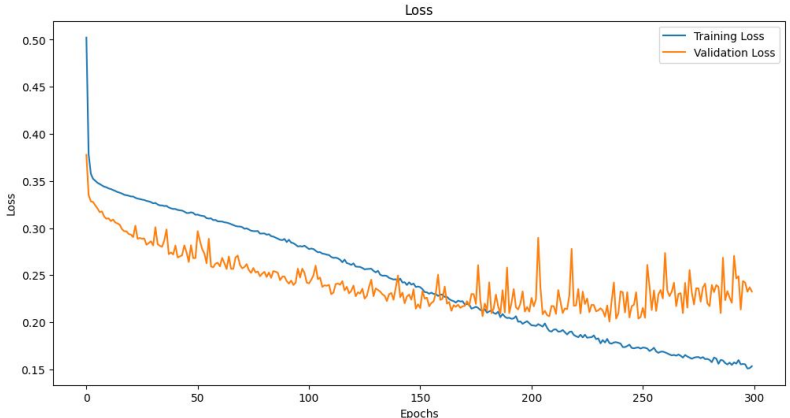
Test Loss: 0.035842977464199066, Test Accuracy: 0.8714044094085693

Validation Loss: 0.23270243406295776
Validation Accuracy: 0.8762711882591248

Classification Report:

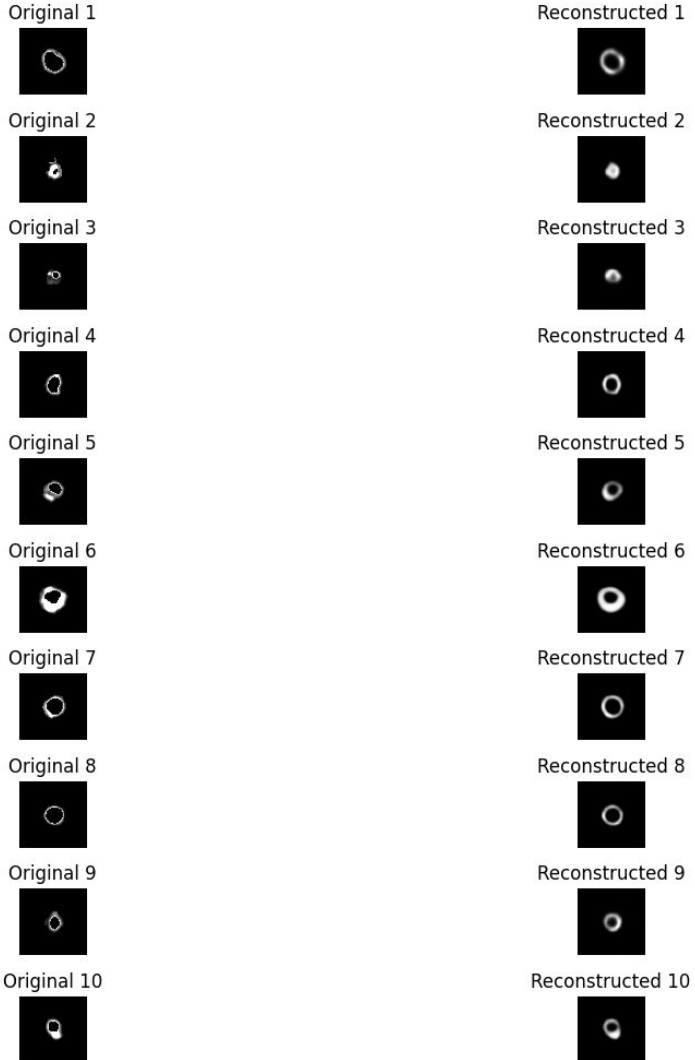
	precision	recall	f1-score	support
Class 0	0.89	0.93	0.91	408
Class 1	0.82	0.74	0.78	183
accuracy			0.87	591
macro avg	0.86	0.84	0.85	591
weighted avg	0.87	0.87	0.87	591



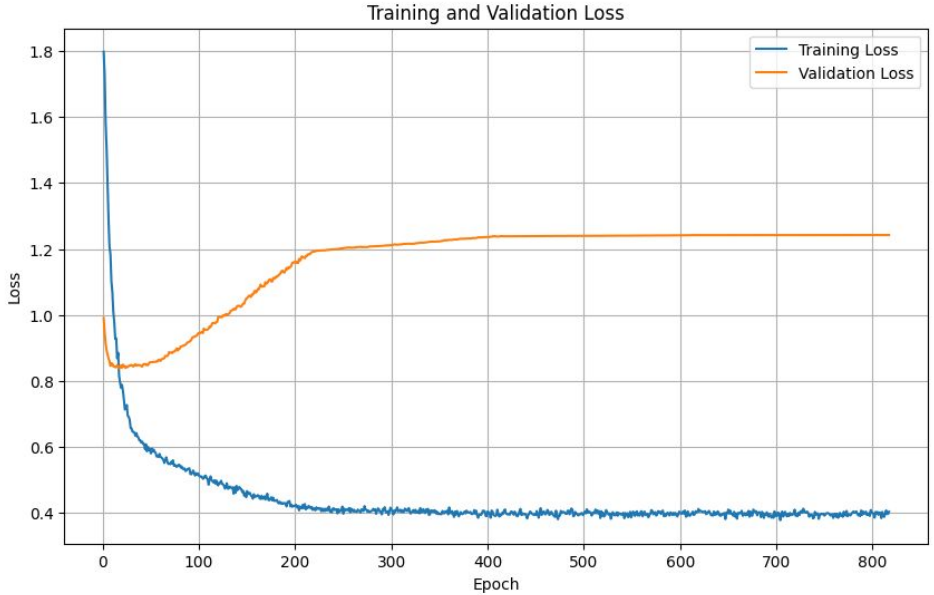


Mean SSIM on Test Set: 0.8894779384604957
Std SSIM on Test Set: 0.06045763699341788

Valore medio di SSIM con maschera: 0.4647321039619405
Deviazione standard di SSIM: 0.13591285240588807



```
F1 score: 0.42857142857142855
PR AUC: 0.4376989432136491
ROC AUC: 0.625
Accuracy: 0.68
Confusion Matrix:
[[14  3]
 [ 5  3]]
Best Precision: 0.5
Best Recall: 0.375
```

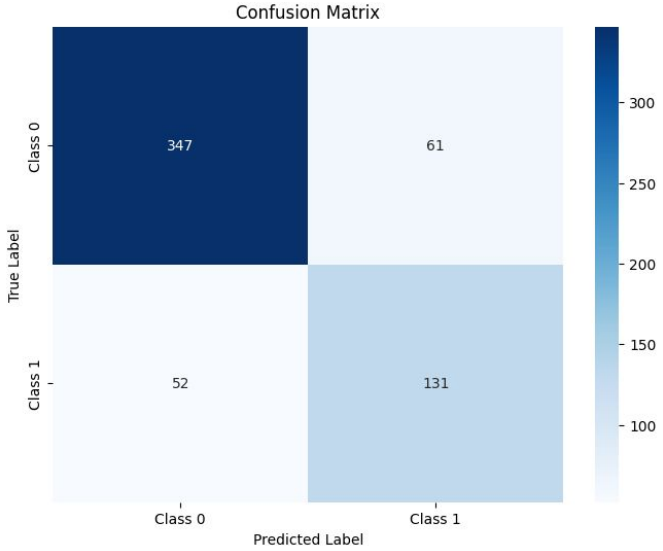


Ma i risultati non migliorano piu di tanto

```
def create_encoder(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(128, (3, 3),
activation='relu', padding='same',
kernel_regularizer=l2(0.001))(inputs)
    x = MaxPooling2D((2, 2),
padding='same')(x)
    x = Conv2D(64, (3, 3),
activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2),
padding='same')(x)
    x = Conv2D(32, (3, 3),
activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2),
padding='same')(x)
    x = Flatten()(x)
    encoded = Dense(128,
activation='relu',
kernel_regularizer=l2(0.001))(x)
    return Model(inputs, encoded,
name="encoder")
```

modello 6
128 features

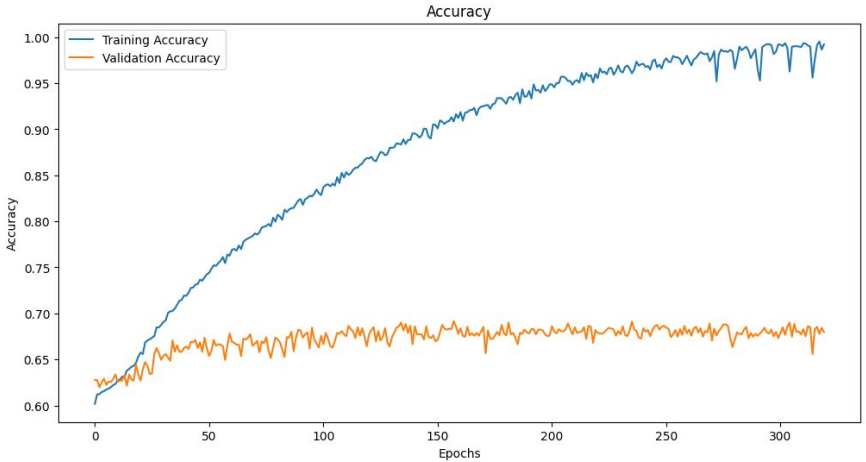
conclusione: classifica peggio del 64 features, guadagni solo nelle ricostruzioni
ci sono tante features a 0. usando correlation 0.9 e p-value ne lascia 65

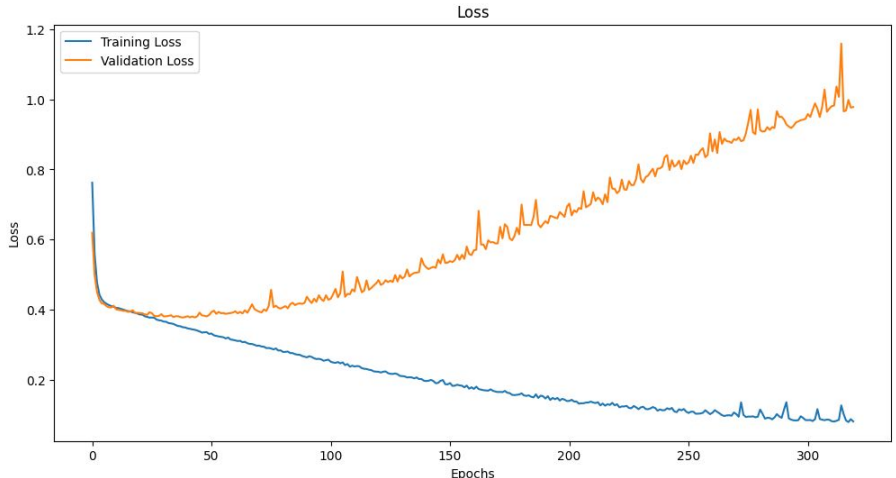


Test Loss: 0.04525649547576904, Test Accuracy: 0.8087986707687378

Classification Report:

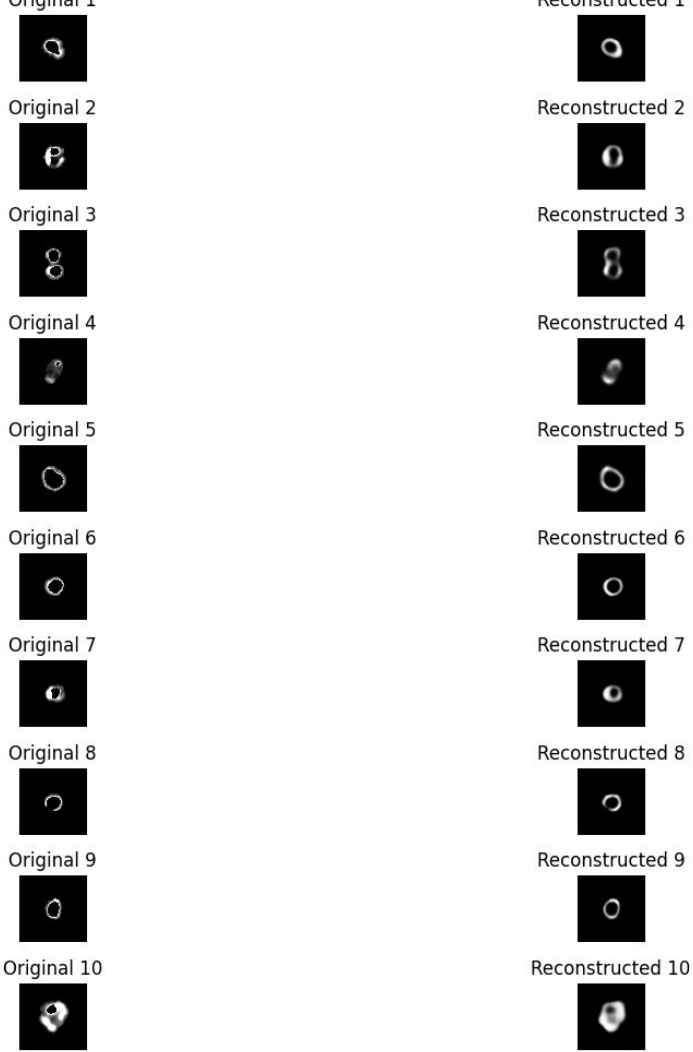
	precision	recall	f1-score	support
Class 0	0.87	0.85	0.86	408
Class 1	0.68	0.72	0.70	183
accuracy			0.81	591
macro avg	0.78	0.78	0.78	591
weighted avg	0.81	0.81	0.81	591





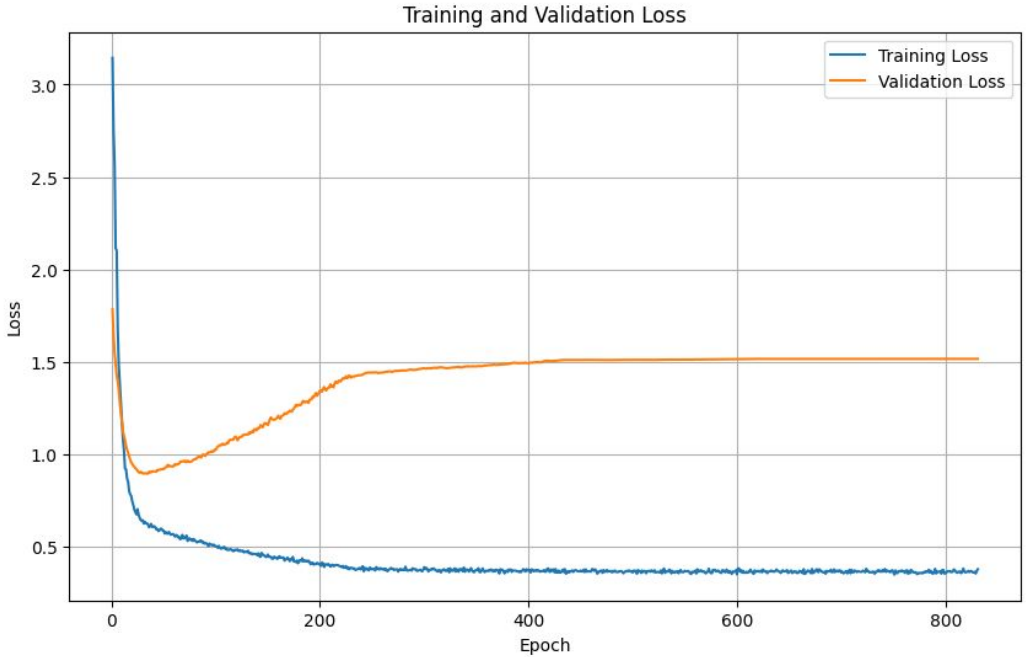
Mean SSIM on Test Set: 0.9121202218144985

Valore medio di SSIM con maschera:
0.5549922496779617
Deviazione standard di SSIM:
0.12946583059311578



classificazione con rete neurale

```
F1 score: 0.26666666666666666
PR AUC: 0.42829254079254075
ROC AUC: 0.6911764705882353
Accuracy: 0.56
Confusion Matrix:
[[12  5]
 [ 6  2]]
Best Precision: 0.2857142857142857
Best Recall: 0.25
```



classificazione con machine learning

Miglior risultato per il classificatore MLP
Selector p_value and mode mean

VALIDATION:
F1 score: 0.6923076923076923
PR AUC: 0.5365723357102667
ROC AUC: 0.6889952153110048
Accuracy: 0.7333333333333333
Numero di features: 31
Confusion Matrix:
[[13 6]
 [2 9]]
Best Precision: 0.6
Best Recall: 0.8181818181818182
Best Threshold: 0.487
Selected Features: [0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]

TEST:
Risultati per 31 funzionalità:
PR AUC: 0.5194708069708069
ROC AUC: 0.6985294117647058
F1: 0.47058823529411764
Precision: 0.4444444444444444
Recall: 0.5
Accuracy: 0.64
Confusion Matrix:
[[12 5]
 [4 4]]

ON TEST WITB TRAINING WITH VAL INCLUDED:

F1 score finale: 0.5555555555555556
Precision finale: 0.5
Recall finale: 0.625
Confusion Matrix finale:
[[12 5]
 [3 5]]
Accuracy finale: 0.68
threshold: 0.487
features: [0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30] e sono 31