UNIVERSITÀ
DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA (DIMES)

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

RELAZIONE PROGETTO MACHINE E DEEP LEARNING

a cura di:

GIORGIO SIMONE 214575

A.A. 2020/2021

Sommario

Introduzione

L’obiettivo del progetto è quello di effettuare l’analisi per due categorie di dati:

- Immagini
- File di testo

Per entrambe le tipologie di dati, dopo aver effettuato una prima fase di pre-processing per far si che questi si prestino meglio all’elaborazione, verranno risolti due tipi di task:

- Classificazione
- Semi-Supervised Anomaly Detection

Nello specifico, per quanto riguarda la Classificazione, verranno utilizzati i seguenti modelli:

- Adaboost
- SVM
- Reti Neurali
- Naive Bayes per Stima di Densità

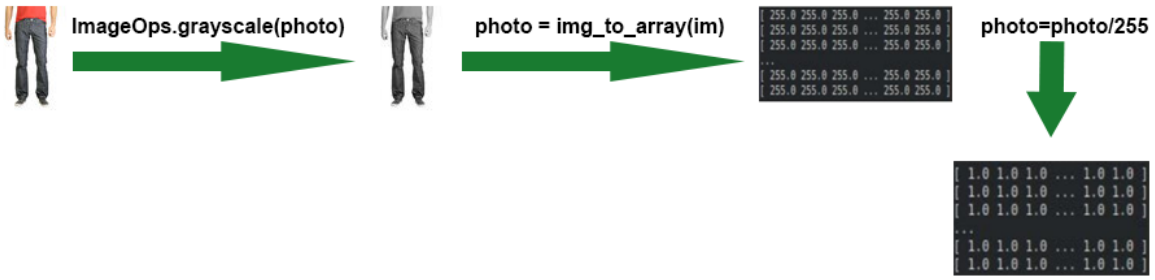
Per quanto riguarda la Anomaly Detection si è scelto di optare per AutoEncoder.

Classificazione Immagini

Caricamento e Pre-processing

La prima fase è stata quella della scelta delle librerie e delle funzioni per il caricamento e la manipolazione delle immagini.
Ad una prima esplorazione del dataset sono state osservate 6204 immagini rgb 60x80, appartenenti a 4 tipologie di prodotti di abbigliamento:
jeans, shirts, trousers, watches.

Dal pacchetto *keras.preprocessing.image* è stata utilizzata la funzione *load_img,* che ha permesso il caricamento e la manipolazione dell'immagine rendendola 60x60.
Subito dopo il caricamento le immagini sono state convertite in scala di grigi e, dopo essere state convertite in numpy array, sono state normalizzate e salvate in un array "photos".
Nello stesso tempo è stato creato un array "labels" con le relative etichette.



Ai fini della classificazione molto utile è stata la funzione *train_test_split* del pacchetto model_selection di sklearn, che ha permesso in un primo momento di suddividere i dati in train e test set, rispettivamente assegnandogli l'80% e il 20% del totale.
Un'altra funzione molto utile da segnalare, è stata la Grid Search di sklearn utilizzata per svolgere l'ottimizzazione dei parametri in modo rapido e semplice, andando ad utilizzare inoltre una cross validation=10.

SVM

Nel suo caso più semplice SVM è applicato sulla classificazione binaria, andando a dividere i punti in 0-1. Per la classificazione multiclasse, si utilizza lo stesso principio.
Nella funzione SVM di sklearn "One-vs-rest" è impostato di default.
Banalmente divide i punti del dataset in una classe "contro" tutte le altre.

```
model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['linear','rbf']
        }
    },
}
```

```
for model_name, mp in model_params.items():
    clf = GridSearchCV(mp['model'], mp['params'], cv=10)
    clf.fit(d2xtrain, ytrain)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
            'best_params': clf.best_params_
    })
```

Il modello migliore restituito è ottenuto con i seguenti parametri:

```
   model  best_score                    best_params
0    svm    0.964536  {'C': 1, 'kernel': 'linear'}
```

Già con questo modello si ottengono ottimi risultati, come possiamo vedere anche dal Classification Report e la Confusion Matrix allegate a seguire, ottenute andando a valutare il classificatore sul test set che era stato separato inizialmente.

Classification Report

	precision	recall	f1-score	support
jeans	0.88	0.84	0.86	100
shirts	0.99	1.00	1.00	572
trousers	0.82	0.88	0.85	88
watches	1.00	0.99	0.99	481
accuracy			0.97	1241
macro avg	0.92	0.93	0.92	1241
weighted avg	0.97	0.97	0.97	1241

Confusion Matrix

```
[[ 84   0  16   0]
 [  0 571   0   1]
 [ 11   0  77   0]
 [  0   4   1 476]]
```

Per completezza allego il significato dei parametri calcolati:

precision= tp/(tp+fp)
abilità del classificatore di non etichettare come positivi campioni che sono negativi

recall= tp/(tp+fn) abilità di trovare tutti i campioni positivi

f-beta score è una media armonia della precision e recall.
beta pesa recall e precision (beta=1 li rende equi-importanti)

support è il numero di occorrenze di ogni classe in y_true

ADABOOST

Per questo classificatore, sono stati provati i seguenti parametri:

```
'adaboost': {
    'model': AdaBoostClassifier(base_estimator= DTC),
    'params': {
        'base_estimator__criterion' : ['gini', 'entropy'],
        'base_estimator__splitter' : ['best', 'random'],
        'n_estimators' : [1,5,10,20,30,40,50]
    }
},
```

Il modello migliore restituito è ottenuto con i seguenti:

,model	,best_score	,best_params
0,adaboost,	0.9447913621154533,	"{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 5}"

Classification Report

	precision	recall	f1-score	support
jeans	0.71	0.73	0.72	130
shirts	0.99	0.99	0.99	564
trousers	0.72	0.71	0.71	104
watches	0.98	0.98	0.98	443
accuracy			0.93	1241
macro avg	0.85	0.85	0.85	1241
weighted avg	0.94	0.93	0.94	1241

Confusion Matrix

```
[[ 95   2  29   4]
 [  3 556   0   5]
 [ 30   0  74   0]
 [  6   2   0 435]]
```

Reti Neurali

Per quanto riguarda le reti neurali, si è deciso di utilizzare sia una ANN che una CNN, per poter osservare e valutare l'effettivo miglioramento delle prestazioni.

ANN

```
ann = models.Sequential([
        layers.Flatten(input_shape=(60,60,1)),
        layers.Dense(360, activation='relu'),
        layers.Dense(400, activation='relu'),
        layers.Dense(60, activation='relu'),
        layers.Dense(4, activation='softmax')
    ])
```

```
Epoch 1/5
156/156 [==============================] - 1s 4ms/step - loss: 0.8525 - accuracy: 0.6367
Epoch 2/5
156/156 [==============================] - 1s 4ms/step - loss: 0.3243 - accuracy: 0.8725
Epoch 3/5
156/156 [==============================] - 1s 4ms/step - loss: 0.2412 - accuracy: 0.8993
Epoch 4/5
156/156 [==============================] - 1s 4ms/step - loss: 0.2520 - accuracy: 0.8998
Epoch 5/5
156/156 [==============================] - 1s 4ms/step - loss: 0.1903 - accuracy: 0.9231
```

Classification Report *Confusion Matrix*

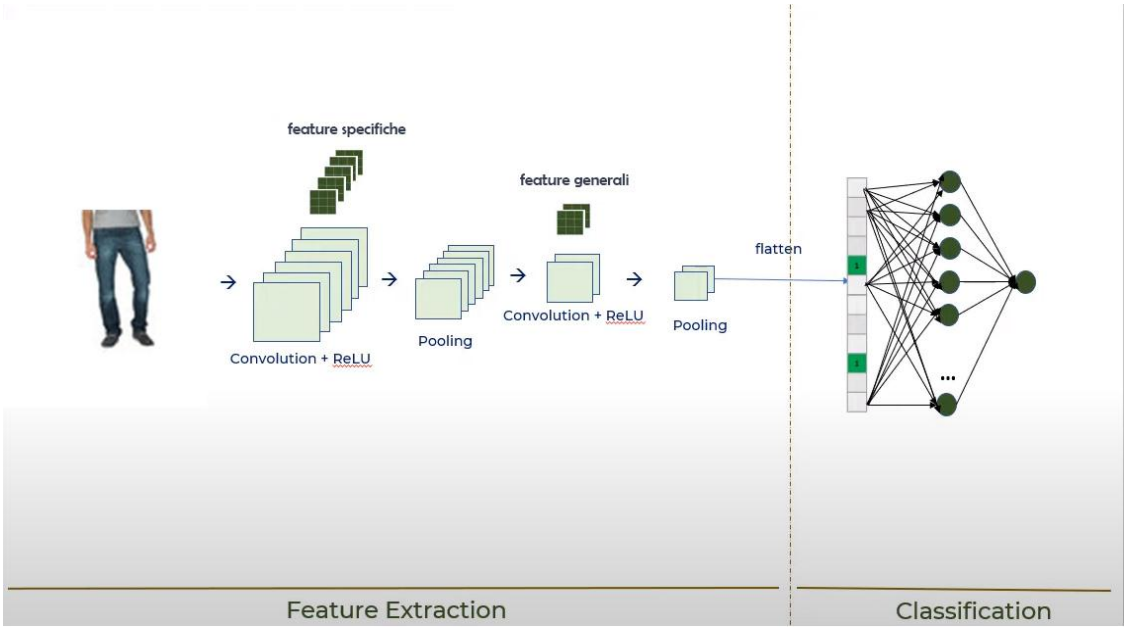
```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       108
           1       1.00      0.94      0.97       582
           2       0.78      0.63      0.70        75
           3       0.93      1.00      0.96       476

    accuracy                           0.94      1241
   macro avg       0.87      0.86      0.86      1241
weighted avg       0.94      0.94      0.94      1241
```

```
[[ 93   0  13   2]
 [  0 548   0  34]
 [ 27   0  47   1]
 [  0   2   0 474]]
```

CNN




```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(60, 60, 1)),
   layers.MaxPooling2D((2, 2)),

   layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
   layers.MaxPooling2D((2, 2)),

   layers.Flatten(),
   layers.Dense(64, activation='relu'),
   layers.Dense(4, activation='softmax')
])
```

```
Epoch 1/10
156/156 [==============================] - 9s 53ms/step - loss: 0.5471 - accuracy: 0.7839
Epoch 2/10
156/156 [==============================] - 8s 52ms/step - loss: 0.1160 - accuracy: 0.9561
Epoch 3/10
156/156 [==============================] - 8s 50ms/step - loss: 0.0820 - accuracy: 0.9703
Epoch 4/10
156/156 [==============================] - 8s 53ms/step - loss: 0.0716 - accuracy: 0.9723
Epoch 5/10
156/156 [==============================] - 9s 57ms/step - loss: 0.0552 - accuracy: 0.9763
Epoch 6/10
156/156 [==============================] - 8s 50ms/step - loss: 0.0439 - accuracy: 0.9819
Epoch 7/10
156/156 [==============================] - 8s 52ms/step - loss: 0.0394 - accuracy: 0.9835
Epoch 8/10
156/156 [==============================] - 8s 49ms/step - loss: 0.0335 - accuracy: 0.9879
Epoch 9/10
156/156 [==============================] - 8s 49ms/step - loss: 0.0376 - accuracy: 0.9842
Epoch 10/10
156/156 [==============================] - 8s 49ms/step - loss: 0.0225 - accuracy: 0.9910
```

Già dopo 5 epoche in questo caso abbiamo raggiunto il 97% di accuracy, contro il 92% della ANN.

Classification Report

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.89      0.91       108
           1       1.00      1.00      1.00       588
           2       0.89      0.91      0.90        99
           3       0.99      1.00      1.00       446

    accuracy                           0.98      1241
   macro avg       0.95      0.95      0.95      1241
weighted avg       0.98      0.98      0.98      1241
```

Confusion Matrix

```
[[ 96   0  11   1]
 [  0 587   0   1]
 [  8   0  90   1]
 [  0   1   0 445]]
```

Con questa CNN si raggiungono i migliori risultati rispetto a tutti i classificatori analizzati.

Stima di densità (Naive Bayes)

Per quanto riguarda la stima di densità si è scelto di optare per il Naive Bayes, algoritmo basato sull'applicazione del teorema di bayes con l'assunzione "naive" di indipendenza tra le variabili.
Tutto sommato anche questo algoritmo si comporta abbastanza bene raggiungendo un accuracy dell' 89%.

Classification Report

```
              precision    recall  f1-score   support

       jeans       0.72      0.39      0.51       119
      shirts       0.99      0.94      0.96       590
    trousers       0.54      0.82      0.65        83
     watches       0.90      0.98      0.94       449

    accuracy                           0.89      1241
   macro avg       0.79      0.78      0.77      1241
weighted avg       0.90      0.89      0.89      1241
```

Confusion Matrix

```
[[ 47   0  58  14]
 [  4 555   0  31]
 [ 13   0  68   2]
 [  1   8   0 440]]
```

Anomaly Detection Immagini

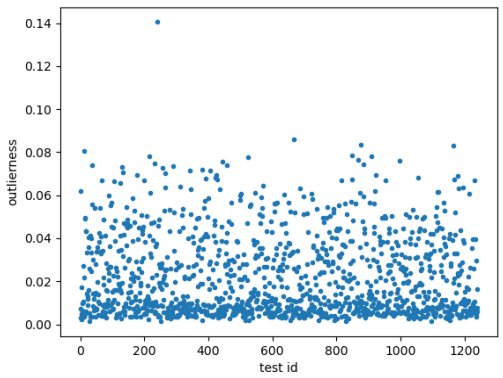
Per l'anomaly detection la classe più numerosa del dataset è stata utilizzata come classe principale, e gli esempi di tutte le altre classi come anomalie. Per far ciò è stato creato il metodo *dataset,* che semplicemente prende in input l'etichetta della classe che deve diventare inlayer, e restituisce le nuove labels con valore 0 (inlayer) e 1 (outlayer), ed inoltre l'insieme degli elementi della classe inlier per effettuare il training del modello.
Come modello è stato utilizzato l'Autoencoder a seguire, con l'idea che, una volta addestrato, vada ad assegnare un errore di ricostruzione maggiore agli elementi del test set che sono outlier.

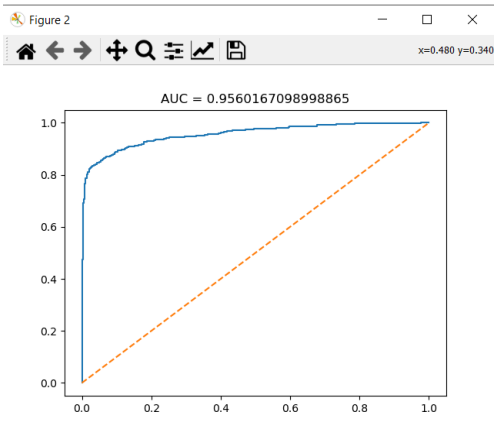
```
# SIZE OF LAYERS
n = 3600
l2 = 128
l3 = 64

# MODEL'S STRUCTURE
input = keras.Input(shape=(n,))
x = keras.layers.Dense(l2, activation ='relu')(input)
x = keras.layers.Dense(l3,activation = 'relu')(x)
encoder = keras.Model(input, x)
x = keras.layers.Dense(l2, activation='relu')(x)
x = keras.layers.Dense(n, activation='sigmoid')(x)
autoencoder = keras.Model(input, x)

autoencoder.compile(optimizer='adam', loss='mse')
```

Per la valutazione della qualità del detector si utilizza la ROC curve lavorando sull'intero dataset (classe principale più anomalie), ottenendo un ottimo risultato.





Classificazione Testi

Caricamento e Pre-processing

L'insieme di file di testo da classificare è formato da 31702 email, suddivise tra ham e spam.
Dopo aver caricato i file con la funzione *load_files* di sklearn, particolarmente utile è stato il pacchetto nltk, che ha permesso il processing del testo caricato, andando a ripulirlo da tutti i caratteri superflui e standandizzarlo per le successive operazioni. esempio:

```
b"Subject: key dates and impact of upcoming sap implementation\r\nover the next few weeks , project apollo and beyond will conduct its final sap\r\nimplementation
```

```
document = nltk.re.sub(r'\\n',' ', str(mail_data[d]))       #rimuovo \n
document = nltk.re.sub(r'\W', ' ', document)                #rimuovo caratteri speciali
document = nltk.re.sub(r'\s+[a-zA-Z]\s+', ' ', document)    #caratteri singoli
document = nltk.re.sub(r'\^[a-zA-Z]\s+', ' ', document)     #rimuovo singoli car dall'inizio
document = nltk.re.sub(r'\s+', ' ', document, flags=nltk.re.I)#tolgo spazi multipli
document = nltk.re.sub(r'^b\s+', '', document)              # rimuovo prefixed 'b'
document = document.lower()                                 # converto lowercase
document = document.split()                                 #lemmatizzazione: sostituisce con radice
document = [stemmer.lemmatize(word) for word in document]
document = ' '.join(document)
mail_data[d]=document
```

```
subject key date and impact of upcoming sap implementation over the next few week project apollo and beyond will conduct it final sap implementation
```

Successivamente, è stata creata la matrice tf-idf, la quale determina quanto è importante una parola pesando la sua frequenza di occorrenze nel documento e calcolando quanto spesso la stessa parola viene trovata negli altri documenti. Se una parola viene trovata molte volte all'interno del documento ma non negli altri, potrebbe essere rilevante per quel documento e le viene quindi assegnata un'importanza maggiore.
Questa tecnica è più accurata rispetto al semplice BAG OF WORDS, il quale crea un vocabolario contando semplicemente il numero di occorrenze delle parole.
Si è scelto di selezionare un massimo di 1500 parole, che devono essere presenti in almeno 5 documenti ed in al più il 70% dei documenti considerati, andando inoltre a rimuovere quelle che sono le stop words della lingua inglese.

```
tfidfconverter = TfidfVectorizer(max_features=1500, min_df=5, max_df=0.7,stop_words=stopwords.words('english'))
```

SVM

```
model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['linear','rbf']
        }
    },
}
```

Il modello migliore restituito è ottenuto con i seguenti parametri:

```
   model  best_score                    best_params
0    svm    0.983321  {'C': 1, 'kernel': 'linear'}
```

Matrice confusione

```
[[3044   69]
 [  30 3198]]
```

Classification Report

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      3113
           1       0.98      0.99      0.98      3228

    accuracy                           0.98      6341
   macro avg       0.98      0.98      0.98      6341
weighted avg       0.98      0.98      0.98      6341
```

ADABOOST

Per questo classificatore, sono stati provati i seguenti parametri:

```
model_params = {
    'adaboost': {
        'model': AdaBoostClassifier(base_estimator= DTC),
        'params': {
            'base_estimator__criterion' : ['gini','entropy'],
            'base_estimator__splitter' : ['best','random'],
            'n_estimators' : [5,10,20,30,40,50]
        }
    },
}
```

Il modello migliore restituito è ottenuto con i seguenti parametri:

```
{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}
```

Classification Report

	precision	recall	f1-score	support
0	0.96	0.97	0.96	3113
1	0.97	0.96	0.96	3228
accuracy			0.96	6341
macro avg	0.96	0.96	0.96	6341
weighted avg	0.96	0.96	0.96	6341

Confusion Matrix

```
[[3010  103]
 [ 138 3090]]
```

Reti Neurali

ANN

```
ann = models.Sequential([
        layers.Dense(1500, activation='relu'),
        layers.Dense(150, activation='relu'),
        layers.Dense(2, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

```
Epoch 1/10
793/793 [==============================] - 6s 8ms/step - loss: 0.6814 - accuracy: 0.6369
Epoch 2/10
793/793 [==============================] - 6s 8ms/step - loss: 0.5587 - accuracy: 0.9151
Epoch 3/10
793/793 [==============================] - 6s 8ms/step - loss: 0.2776 - accuracy: 0.9603
Epoch 4/10
793/793 [==============================] - 6s 8ms/step - loss: 0.1392 - accuracy: 0.9719
Epoch 5/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0983 - accuracy: 0.9767
Epoch 6/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0796 - accuracy: 0.9792
Epoch 7/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0669 - accuracy: 0.9821
Epoch 8/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0589 - accuracy: 0.9839
Epoch 9/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0543 - accuracy: 0.9853
Epoch 10/10
793/793 [==============================] - 6s 8ms/step - loss: 0.0517 - accuracy: 0.9854
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      3113
           1       0.97      0.99      0.98      3228

    accuracy                           0.98      6341
   macro avg       0.98      0.98      0.98      6341
weighted avg       0.98      0.98      0.98      6341
```

```
[[3031   82]
 [  34 3194]]
```

Stima di densità (Naive Bayes)

Anche in questo caso come nella classificazione delle immagini si è optato per l'algoritmo Naive Bayes, il quale in questo caso ha ottenuto ottimi risultati.

Classification Report

	precision	recall	f1-score	support
ham	0.97	0.97	0.97	3113
spam	0.97	0.97	0.97	3228
accuracy			0.97	6341
macro avg	0.97	0.97	0.97	6341
weighted avg	0.97	0.97	0.97	6341

Confusion Matrix

```
[[3031   82]
 [  89 3139]]
```

Anomaly Detection Testi

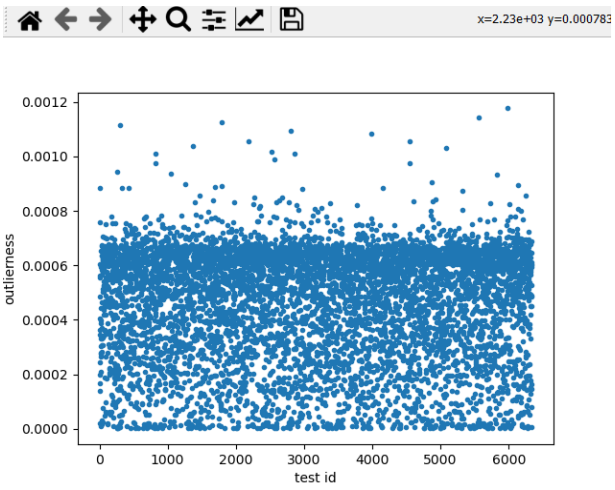
Anche qui vale quanto detto per l'Anomaly Detection delle immagini, con la semplificazione che avendo solo due classi, non c'è stato bisogno di manipolazioni particolari come nel caso precedente.

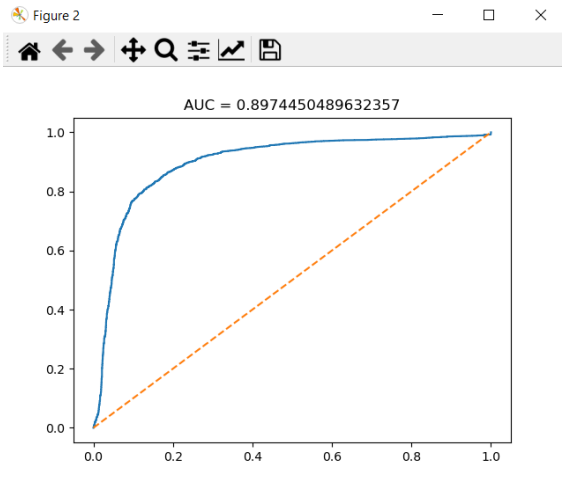
```
# SIZE OF LAYERS
n = 1500
l2 = 128
l3 = 64

# MODEL'S STRUCTURE
input = keras.Input(shape=(n,))
x = keras.layers.Dense(l2, activation ='relu')(input)
x = keras.layers.Dense(l3,activation = 'relu')(x)
encoder = keras.Model(input, x)
x = keras.layers.Dense(l2, activation='relu')(x)
x = keras.layers.Dense(n, activation='sigmoid')(x)
autoencoder = keras.Model(input, x)

autoencoder.compile(optimizer='adam', loss='mse')
```

Concludo l'elaborato allegando la ROC curve:





Grazie per l'attenzione.