UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Corso di Laurea Magistrale in Ingegneria Informatica

Relazione Progetto Esame:

Machine & Deep Learning

Candidato:

Ivan Scuderi 216635

Relatori:

Prof. Fabrizio Angiulli

Prof. Fabio Fassetti

Prof. Luca Ferragina

Anno Accademico 2021-2022

Indice

Introduzione: Descrizione Progetto e Struttura della Relazione

Il progetto assegnato ha come obiettivo lo sviluppo di un sistema di apprendimento automatico tramite linguaggio *Python*, che permetta la risoluzione di due *task* su due dataset differenti.

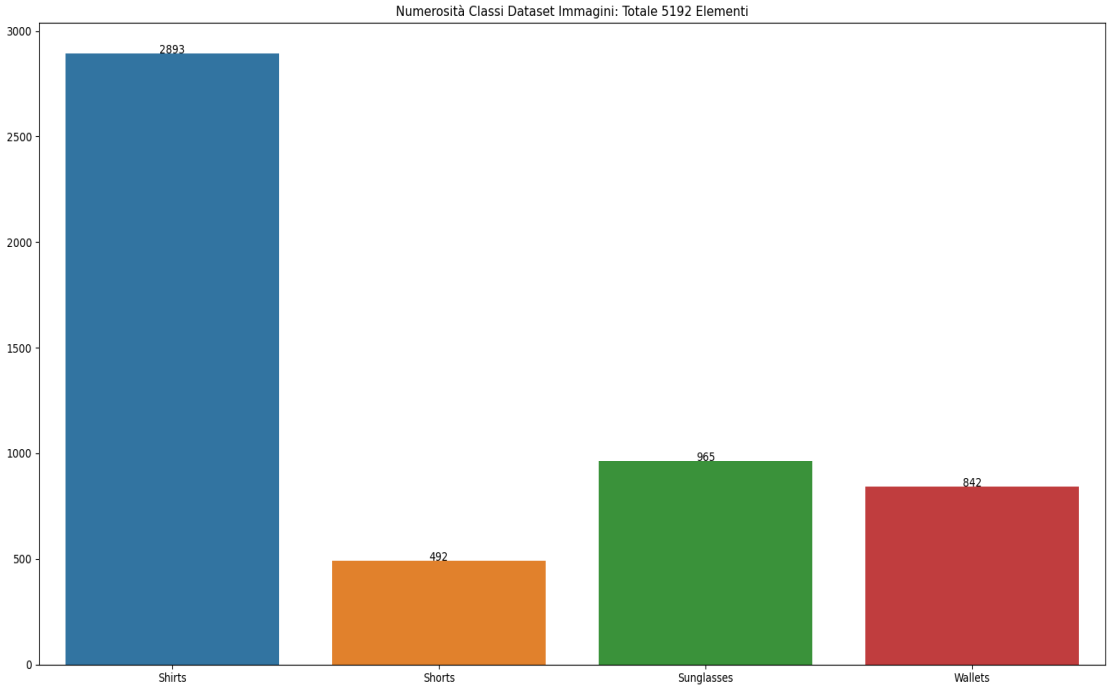
I *task* in questione comprendono: un problema di classificazione multi-classe, un problema di *anomaly detection* tramite tecniche di apprendimento semi-supervisionato, in cui gli esempi della classe più numerosa vanno a formale i dati normali mentre i restanti vengono considerati anomalie.

Per quanto riguarda i dati, oltre all'impiego di un primo *dataset* di immagini, si è scelto un secondo *dataset* riguardante dati testuali, la struttura della seguente relazione segue l'analisi di entrambi i *task* per entrambi i dataset impiegati. Si inizierà andando a descrivere i dati (numerosità, caratteristiche, numero di classi ed altro) e le operazioni di *preprocessing* effettuate, per poi andare ad analizzare modelli e scelte per la risoluzione dei due problemi.

Capitolo 1: Dataset di Immagini

1.1: Descrizione e Preprocessing dei Dati

Il dataset di immagini comprende un totale di 5192 elementi divisi in 4 classi distinte: '*Shirts*', '*Shorts*', '*Sunglasses*', '*Wallets*'. Sostanzialmente ogni immagine fa riferimento ad un capo di abbigliamento, si può inoltre osservare dai grafici sottostanti come tale dataset sia leggermente sbilanciato, con la classe più numerosa che risulta essere '*Shirts*', i cui elementi verranno quindi considerati facenti parte della classe normale quando si dovrà risolvere il task di *anomaly detection*.

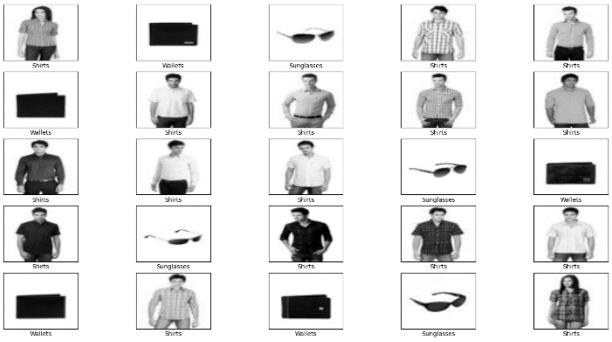


Le immagini sono di dimensione 60x80 a colori, il che significa che in fase di sviluppo dei modelli di apprendimento si dovranno tenere in considerazione i 3 canali colore. Per quanto riguarda la fase di *preprocessing* non sono stati svolti passaggi troppo complessi, ho sviluppato uno *script* che mi permettesse di leggere dal disco le varie immagini al fine di generare un *numpy ndarray* insieme al rispettivo vettore delle etichette di classe, ovviamente i dati sono stati normalizzati portando ogni elemento dell'*array* nell'intervallo [0, 1]. Inoltre in tale *script* vado ad eliminare eventuali elementi corrotti, sono state infatti eliminati un totale di 45 elementi su cui sono stati riscontrati errori di lettura, di cui: 4 facenti parte della classe '*Shirts*', 5 della classe

'*Sunglasses'* e 36 per la classe '*Wallets'*. In particolare è stato impossibile ottenere i 3 canali colore perché la lettura restituiva un unico canale, scelta progettuale è stata quindi quella di escludere tali dati, senza provare ad ottenere i valori mancanti adoperando tecniche di elaborazione delle immagini , essendo comunque il numero di elementi corrotti trascurabile rispetto alla dimensione del *dataset*.

Scelta progettuale è stata inoltre quella di andare ad impiegare tutti i canali colore solo per lo sviluppo dei modelli basati su reti neurali, mentre si è preferito impiegare la conversione in scala di grigi per i dati sottomessi ai modelli di *machine learning* non basati su reti. Questo è stato dettato dal fatto che, essendo le *ANN* molto potenti dal punto di vista dell'analisi delle immagini (in particolare le *CNN*), ho pensato di andare a sfruttare tutte le informazioni fornite dai vari canali per sviluppare modelli robusti e precisi, mentre la scelta dell'utilizzo di un unico canale per lo sviluppo degli altri modelli è stata fatta per evitare di appesantire troppo gli algoritmi. In definitiva si andranno quindi ad impiegare tensori di dimensione (60, 80, 3) per lo sviluppo delle reti neurali, mentre di dimensione (60, 80) per i modelli di *machine learning*. Infine per quanto riguarda la divisione in *training set* e *test set*, si è scelto di impiegare il 20% degli elementi campionati per il *testing*; ottenendo quindi un totale di 4117 immagini per l'addestramento e 1030 immagini per il *test*.

Di seguito si può vedere un esempio di immagini a colori ed in scala di grigi provenienti dalle varie classi.



1.2: Task di Classificazione

Nel proseguo della trattazione verranno valutati e descritti i modelli sviluppati per la risoluzione del *task* di classificazione per il *dataset* di immagini, ed analizzate eventuali scelte progettuali sulle architetture impiegate. Per quanto riguarda il *testing*, oltre all'impiego dell'accuratezza come metrica principale, ho calcolato *precision*, *recall* e *fi-score* per ognuna delle 4 classi, andando opportunamente a visualizzare i risultati prodotti.

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}}$$

$$\text{F}_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Inoltre come specificato nella consegna, è stata anche impiegata *10-fold cross-validation* per ognuno dei modelli valutati. Le operazioni di *parameter tuning* sono state svolte mediante l'impiego della tecnica '*GridSearchCV*' con un numero di *fold* sviluppati che variano a seconda del classificatore impiegato. Questa scelta è stata dettata semplicemente per un fattore temporale, poiché per molti classificatori anche pochi *fold* rendevano il processo estremamente lento, essendo questo tempo legato anche al numero di parametri valutati, ho preferito sviluppare un numero inferiore di *fold* ad ogni iterazione, andando però ad analizzare più parametri. Di seguito verranno discussi nel dettaglio i modelli adoperati, relativi alle tecniche: *AdaBoost*, *SVM, Reti Neurali, Stima di Densità*.

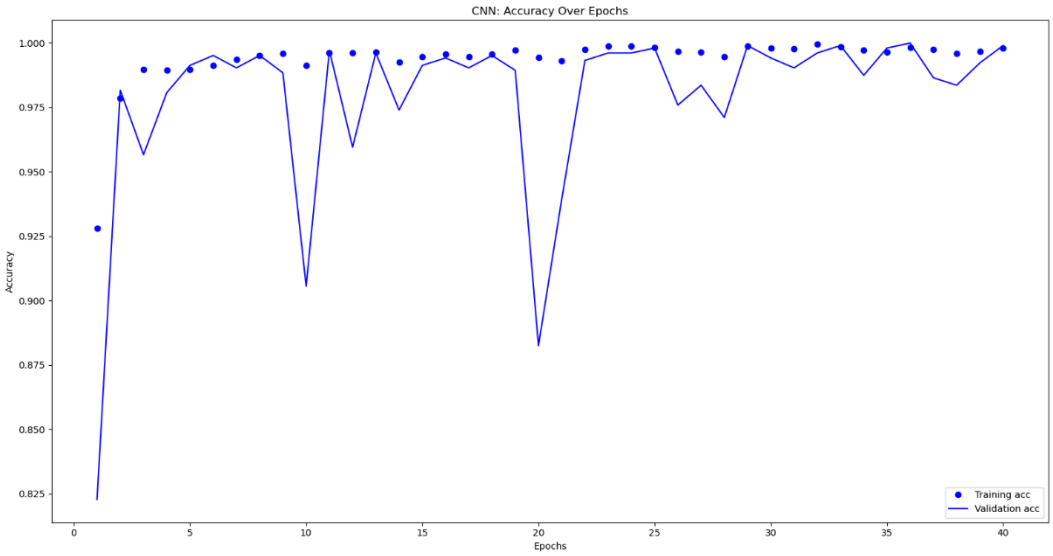
1.2.1: Reti Neurali

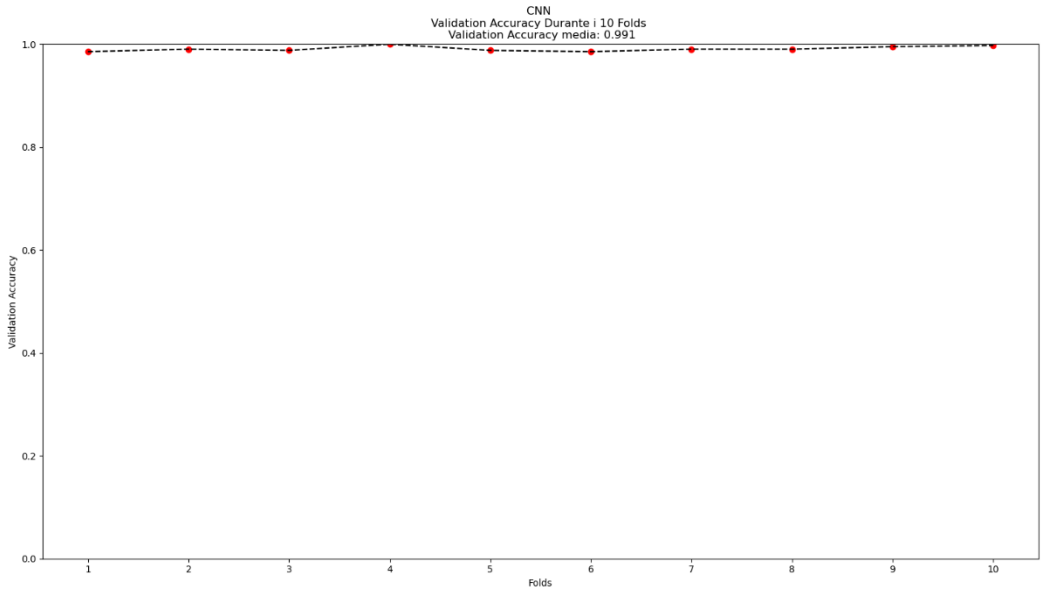
L'analisi condotta ha messo a confronto due differenti architetture ossia una ***rete convoluzionale*** ed una ***reta densa***, come detto precedentemente per tali modelli sono stati mantenuti i canali colore delle immagini, mentre le operazioni di *cross-validation* sono state implementate mediante l'utilizzo della libreria *scikit-learn* andando ad utilizzare l'oggetto '*KerasClassifier';* che sostanzialmente funge da *wrapper* implementando l'interfaccia degli *estimator* di *scikit-learn* tramite l'utilizzo dei modelli *keras*. Per quanto riguarda la rete convoluzionale sono partito dall'architettura *Lenet* andandola a modificare inserendo un

numero maggiore di livelli. Il modello impiegato aggiunge sostanzialmente un *layer* convoluzionale in più alla *lenet-5* andando

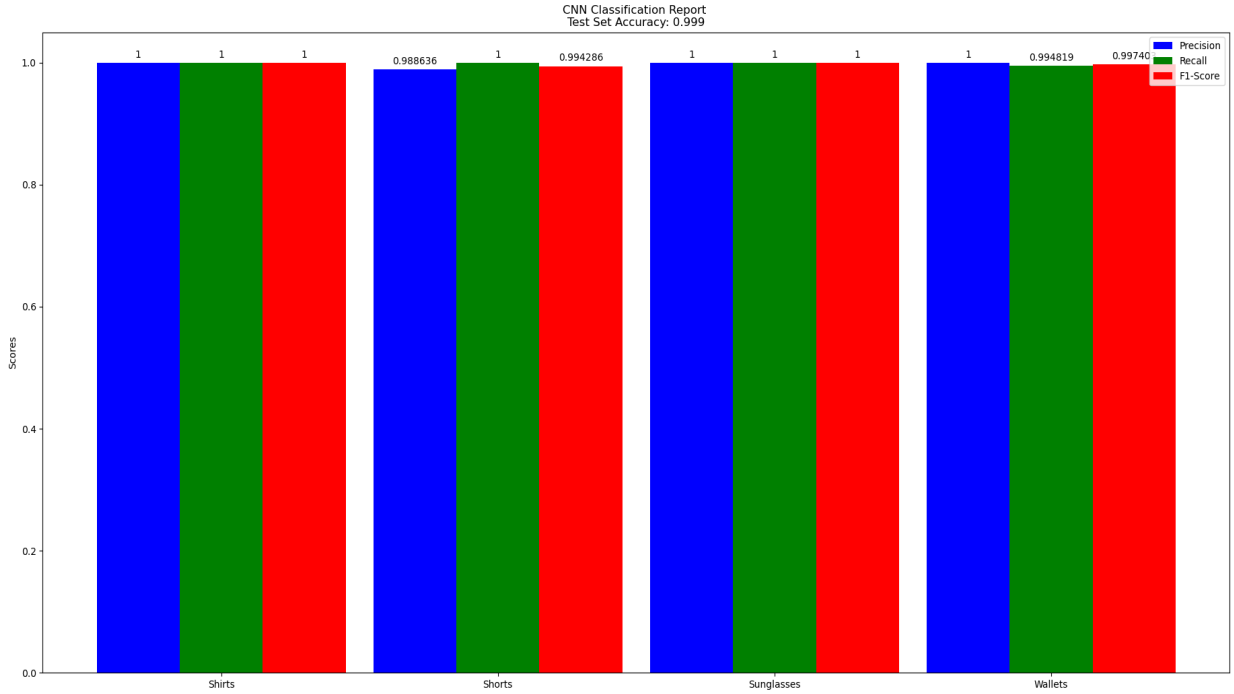
```
x = Conv2D(filters=32, kernel_size=(5, 5), padding='same')(x)
x = keras.layers.BatchNormalization()(x)
x = keras.activations.relu(x)
x = MaxPooling2D(strides=2)(x)
x = Conv2D(filters=48, kernel_size=(5, 5), padding='valid')(x)
x = keras.layers.BatchNormalization()(x)
x = keras.activations.relu(x)
x = MaxPooling2D(strides=2)(x)
x = Conv2D(filters=48, kernel_size=(5, 5), padding='valid')(x)
x = keras.layers.BatchNormalization()(x)
x = keras.activations.relu(x)
x = MaxPooling2D(strides=2)(x)
x = Flatten()(x)
x = Dense(1344, activation='relu', kernel_regularizer=keras.regularizers.l2())(x)
x = Dropout(0.3)(x)
x = Dense(672, activation='relu', kernel_regularizer=keras.regularizers.l2())(x)
x = Dropout(0.3)(x)
x = Dense(560, activation='relu', kernel_regularizer=keras.regularizers.l2())(x)
x = Dropout(0.3)(x)
x = Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2())(x)
x = Dropout(0.3)(x)
y = Dense(4, activation='softmax')(x)
```

ad impiegare i medesimi valori per numero di filtri, dimensione del *kernel* e tipologia di *padding*. Si vanno ad alternare quindi operazioni di convoluzione, *batchnormalization* e *maxpooling*, andando ad impiegare come funzione di attivazione per tali livelli nascosti la *ReLU*. Infine si aggiungono 4 livelli nascosti *fully-connected,* in cui si va ad impiegare *dropout* con valore 0.3 e si effettua regolarizzazione tramite metrica *l2*, il tutto per evitare di incorrere in *overfitting*. In una prima soluzione sono state impiegate anche tecniche di *data augmentation*, ma si sono riscontrati risultati peggiori, rinunciando quindi al loro utilizzo. Scelta progettuale è stata anche l'impiego di '*softmax'* come funzione di attivazione del livello di output, in modo tale da avere come risultati le probabilità di appartenenza di un esempio alle varie classi, la cui somma va ad 1. Il modello è stato compilato andando ad impiegare l'algoritmo *Adam* come ottimizzatore (con un *learning rate* iniziale di 10^-3), e come funzione di *loss* '*sparse categorical cross-entropy'* (avendo come etichette di classe dei valori interi nell'intervallo 0-3). Infine per quanto riguarda la fase di addestramento sono state sviluppate un totale di 40 epoche, impiegando come dimensione del *batch* in input un valore pari a 32 immagini. Come si può vedere dai grafici sottostanti, nonostante l'andamento altalenante dell'accuratezza durante l'addestramento, si riescono a raggiungere valori molti elevati vicini all'1. Inoltre a confermare la bontà dell'architettura è anche il valore di *validation accuracy* durante i 10 fold sviluppati per la cross-validation, che si assesta a 0.991.





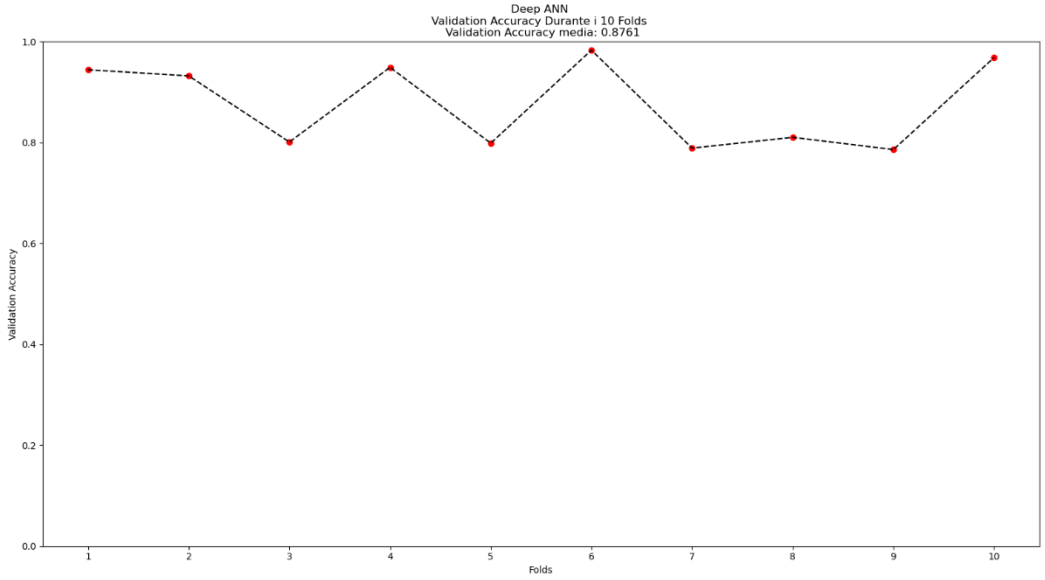
La valutazione sull'insieme di immagini di test ha dato risultati in linea con quelli ottenuti nel processo di validazione, come si può notare dal grafico a barre successivo il classificatore ottenuto è quasi ottimale, andando difficilmente a commettere errori.



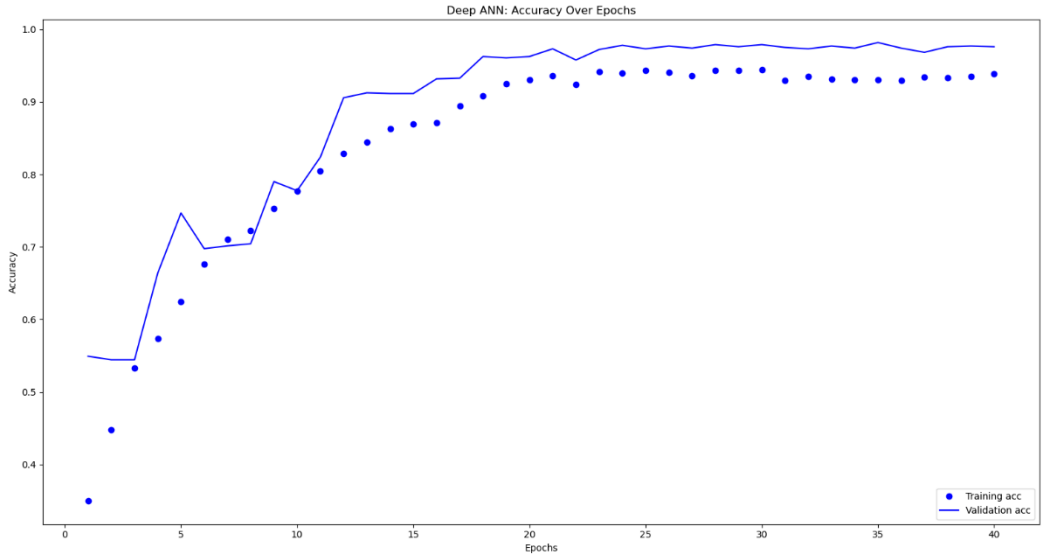
L'accuratezza ottenuta è circa pari ad 1 e, come si può osservare, si vanno a commettere errori sono per le classi '*Shirts'* e '*Wallets'*, in sostanza la rete va a riscontrare un basso valore di falsi positivi per quanto riguarda la prima etichetta, mentre si hanno falsi negativi per la seconda. Questo molto probabilmente è riconducibile al fatto che la numerosità degli esempi della classe '*Shorts'* è la più bassa tra le quattro, e probabilmente anche al fatto che le immagini sono di bassa qualità e con dimensioni molto piccole. Questo fa perdere di dettaglio durante le varie operazioni di convoluzione e quindi, essendo la forma degli elementi delle due classi abbastanza simile per qualche particolare esempio, può portare ad eventuali errori della rete.

Per quanto riguarda l'architettura densa, l'obiettivo era semplicemente andare a valutare come potevano variare i risultati analizzati precedentemente, partendo dal presupposto che difficilmente avrebbe potuto performare meglio, essendo le reti convoluzionali molto potenti e specifiche per l'elaborazione di immagini. Avendo un tensore in input di dimensione (60, 80, 3) opportunamente portato ad un'unica dimensione per sottometterlo alla rete, ho sviluppato 6 livelli densi intermedi partendo da un numero di neuroni pari a 2800 in quello più esterno, andando circa a dimezzare tale valore arrivando infine a 64 neuroni sull'ultimo livello intermedio. Ovviamente è stato impiegato dropout, in questo caso con parametro 0.5, mentre le ulteriori scelte progettuali per quanto riguarda ottimizzatore, numero di epoche, dimensione del batch, funzione di *loss* e funzione di attivazione per l'output, sono le medesime dell'architettura convoluzionale. Uno dei problemi principali di questa soluzione è senza dubbio l'elevato numero di parametri, si hanno circa 44 milioni totali di cui 40 milioni provengono solamente dal

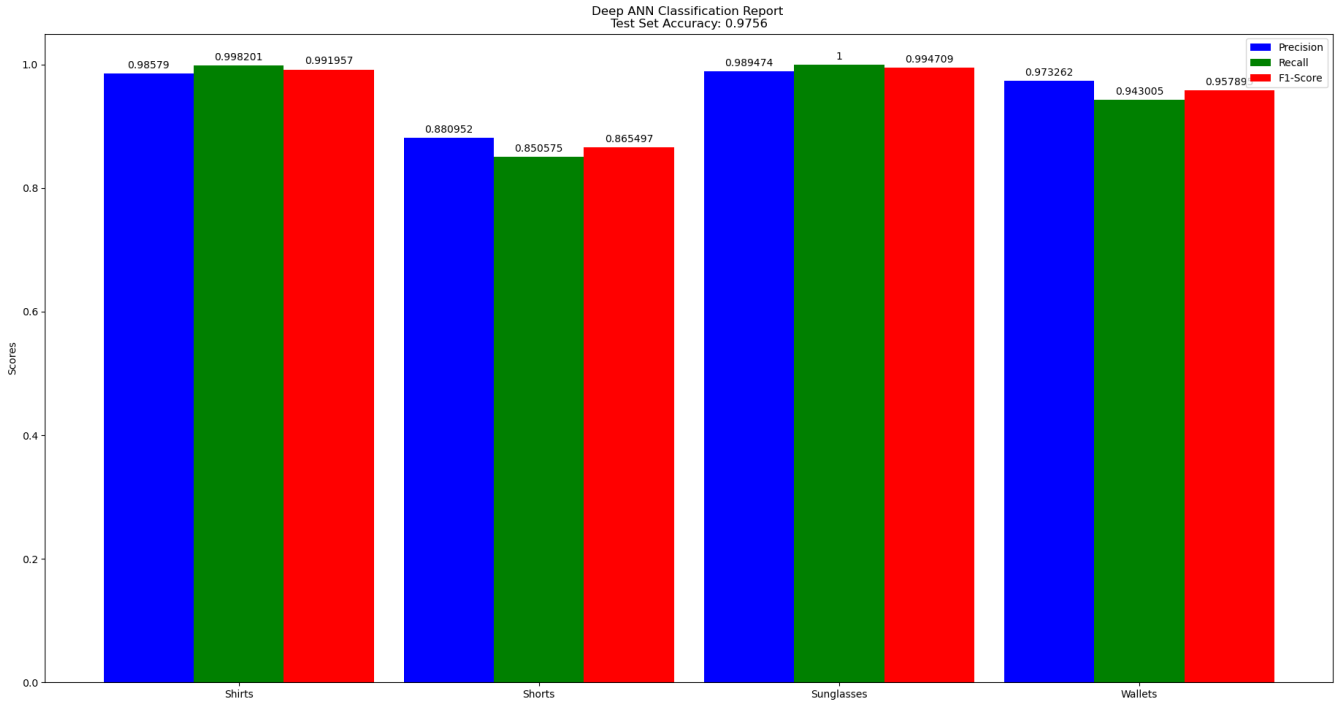
passaggio dal livello di input al primo livello intermedio, questo ha reso quindi lenta ed onerosa la fase di *training*. I risultati sono comunque molto buoni, si deve però sottolineare lo strano andamento dell'accuratezza calcolata sul *validation set* durante i vari *fold*, questo potrebbe far pensare che il risultato finale sia in qualche modo legato alla struttura dei vari *batch* durante le fasi di addestramento.



Si deve invece sottolineare come la curva di addestramento mostri un andamento più stabile rispetto all'architettura convoluzionale, molto probabilmente per via dell'impiego di regolarizzazione in ogni livello (mentre nella *CNN* i livelli convoluzionali ne erano esclusi).



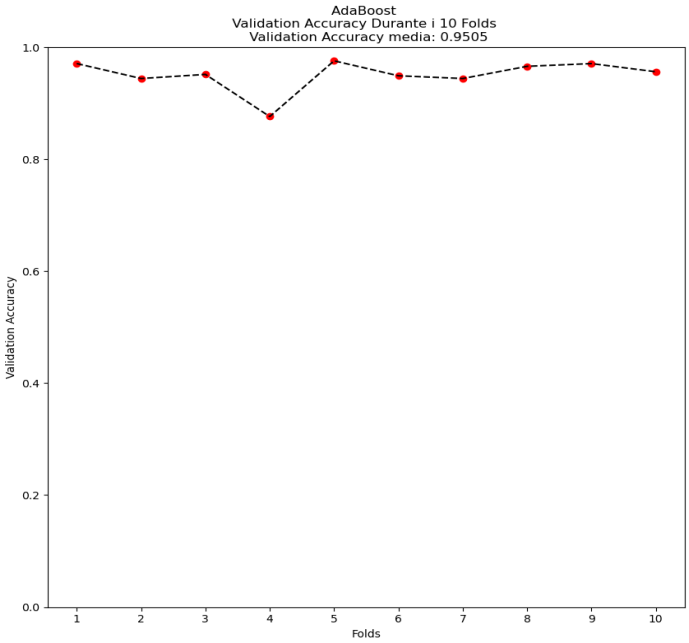
Nonostante si ottenga come accuratezza media durante il processo di *cross-validation* un valore pari circa a 0.87, sul dataset di test si hanno risultati molto buoni con 0.97 di accuratezza raggiunta. Questo valore è penalizzato molto soprattutto dagli errori commessi sulla classe '*Shorts*', dovuti al fatto che è quella che presenta meno esempi.



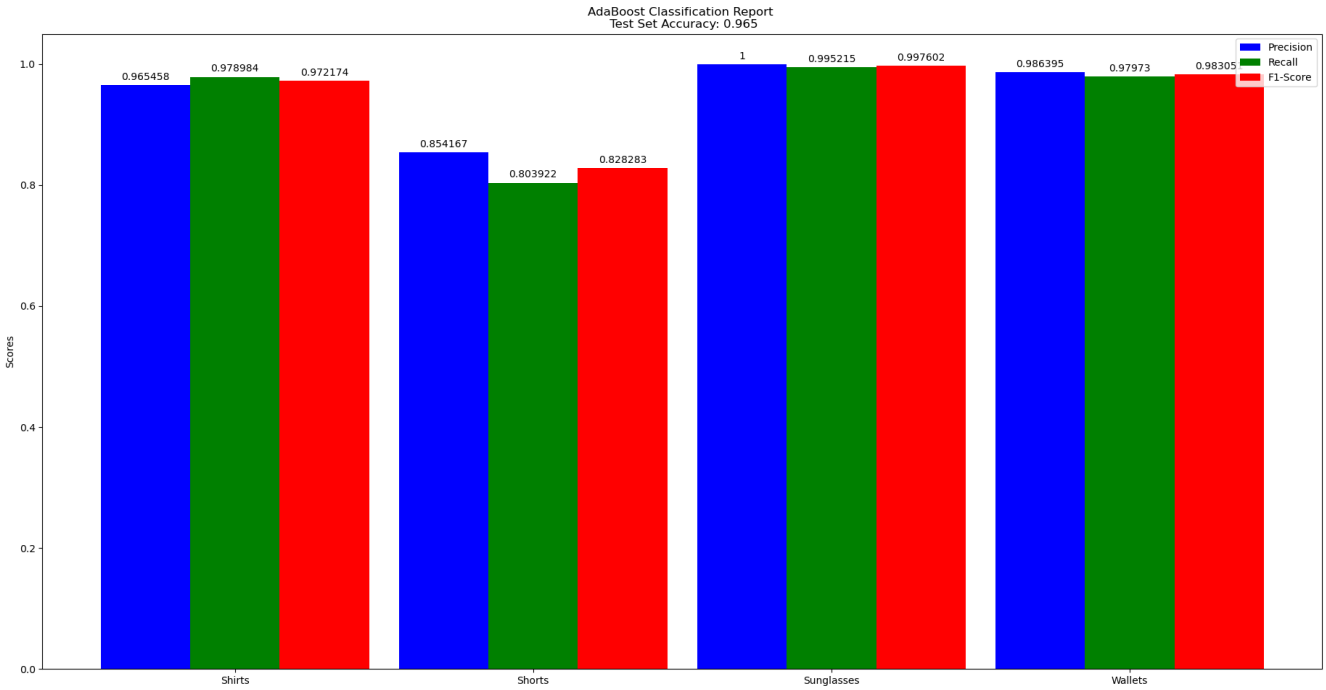
1.2.2: AdaBoost

Per quanto riguarda l'impiego della tecnica di *boosting AdaBoost*, scelta progettuale è stata quella di utilizzare il *weak learner* di base dell'implementazione di *scikit-learn* che risulta essere un albero decisionale, questo anche per andare ad analizzare i risultati restituiti sulle immagini da classificatori non basati su semi-spazi. Gli iper-parametri di cui si è effettuato il *tuning* in questo caso sono stati il numero di classificatori da produrre durante le varie iterazioni ed il *learning rate*, parametro impiegato per andare a controllare il contributo di ogni classificatore sul risultato finale. Il risultato migliore, pari a 0.96 di *validation accuracy*, è stato ottenuto dalla configurazione avente numero di *estimator* pari a 300 ed un *learning rate* pari a 2, la ricerca è stata effettuata tra le varie combinazioni di valori [0.7, 1, 1.5, 2, 3, 5] per il learning rate, e [50, 100, 150, 200, 250, 300] per il numero di estimator. La scelta di tali liste è stata svolta puramente in via sperimentale. Per via dell'elevato numero di iterazioni della *GridSearch* e anche per la lentezza di addestramento di tali modelli, si è optato per sviluppare solo 5 *fold* per la fase di

parameter-tuning; nel grafico sottostante si vanno a riportare i risultati dell'operazione di *10-fold cross-validation* sul miglior *estimator* che mostra un valore di accuratezza media sul *validation set* pari a 0.95 . (in linea con i risultati della *GridSearch*)



Anche in questo caso il modello si va a comportare meglio sull'insieme di test guadagnando un'accuratezza di circa 0.96. Come precedentemente riscontrato, anche per la tecnica *AdaBoost* la maggior parte degli errori provengono dalla classe meno numerosa, comunque il modello prodotto risulta essere molto buono.



1.2.3: SVM

Prima di continuare con la trattazione si deve specificare che la versione delle *Support Vector Machine* implementate in *scikit-learn* va ad impiegare come strategie di classificazione multi-classe "*One-vs-One*" (che corrisponde in letteratura ad "*All Pairs*"). Si va sostanzialmente ad addestrare un classificatore per ogni coppia di classi ed al momento della previsione si restituisce per l'esempio in input il valore di classe che ottiene più voti dai vari classificatori costruiti.

Il modello SVM ha dato risultati molto buoni superando in accuratezza la rete densa, lo studio degli iper-parametri è stato condotto andando ad effettuare *GridSearch* sui valori di:

```
'C': [.3, .5, .7, 1, 2, 5],
'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
'degree': [2, 3, 5, 7],
'gamma': ['scale', 'auto']
```

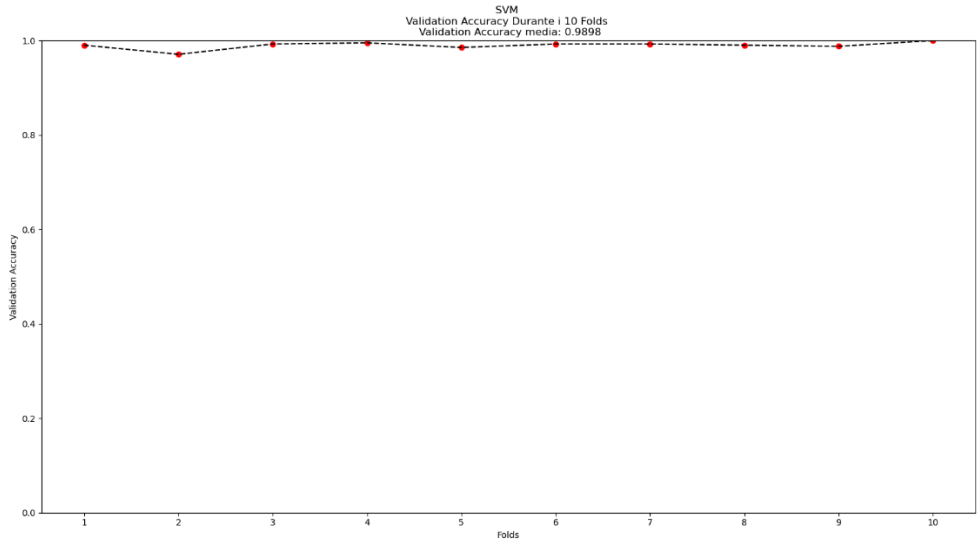
- "**C**", parametro legato in maniera inversamente proporzionale alla 'forza' della regolarizzazione (applicata nell'implementazione *scikit-learn* di *SVC* tramite norma *l2*), diminuendone il valore si va ad applicare regolarizzazione in maniera più marcata.
- "***kernel***", le funzioni kernel valutate sono state: *kernel lineare*(che corrisponde al non utilizzare nessun *kernel*), *kernel polinomiale*, *kernel rbf* (che corrisponde al kernel gaussiano), *kernel sigmoid.*

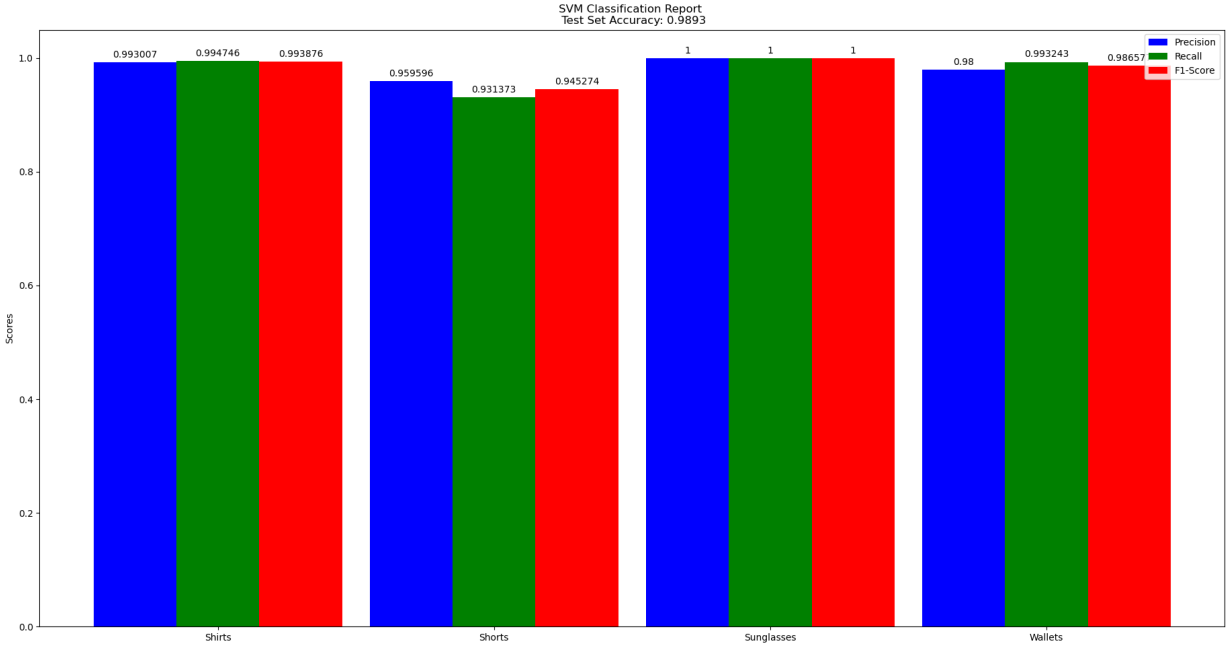
$$\begin{aligned}&\text{linear: } \langle x, x' \rangle \\ &\text{polynomial: } (\gamma \langle x, x' \rangle + r)^d \\ &\text{rbf: } \exp(-\gamma \|x - x'\|^2) \\ &\text{sigmoid } \tanh(\gamma \langle x, x' \rangle + r)\end{aligned}$$

- "***degree***", parametro legato all'utilizzo del *kernel polinomiale* che va a specificare il grado del polinomio impiegato per la trasformazione dello spazio.
- "***gamma***", parametro che va a specificare il coefficiente gamma per i *kernel rbf, polinomiale* e *sigmoid*; può assumere due valori ossia "*scale*": gamma = 1/(n_features * X.var()); oppure "*auto*": gamma = 1/n_features .

Sono stati svilupparti 5 fold per la tecnica GridSearch, andando ad ottenere come configurazione migliore la seguente: *C=0.5, kernel='poly' , gamma='scale', degree=2* . Inoltre tale classificatore raggiunge ottimi risultati sia mediante *10-fold*

cross-validation, con 0.98 di accuratezza media sull'insieme di validazione, che andando ad effettuare la valutazione sul *test set*, su cui ottiene sostanzialmente lo stesso risultato.





L'andamento dell'accuratezza durante i vari *fold* è molto regolare, e in generale anche per quanto riguarda le valutazioni effettua sulle singole classi, il classificatore *SVM* è quello che ricorda maggiormente i risultati della rete convoluzionale,

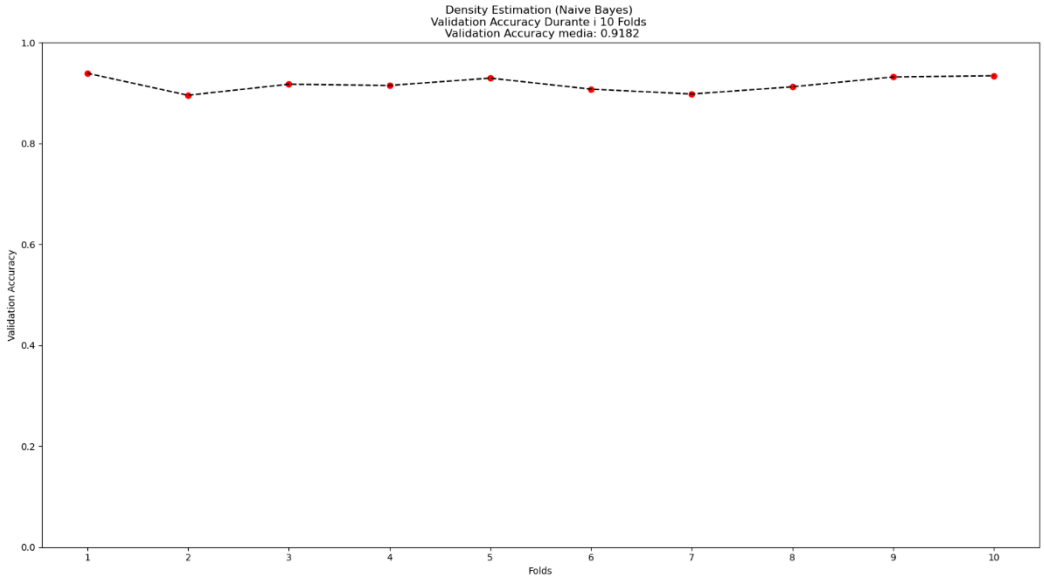
andando anche ad ottenere buoni valori per la classe meno numerosa, ossia '*Shorts',* segno che il *kernel* impiegato funziona bene nel discriminare gli esempi.

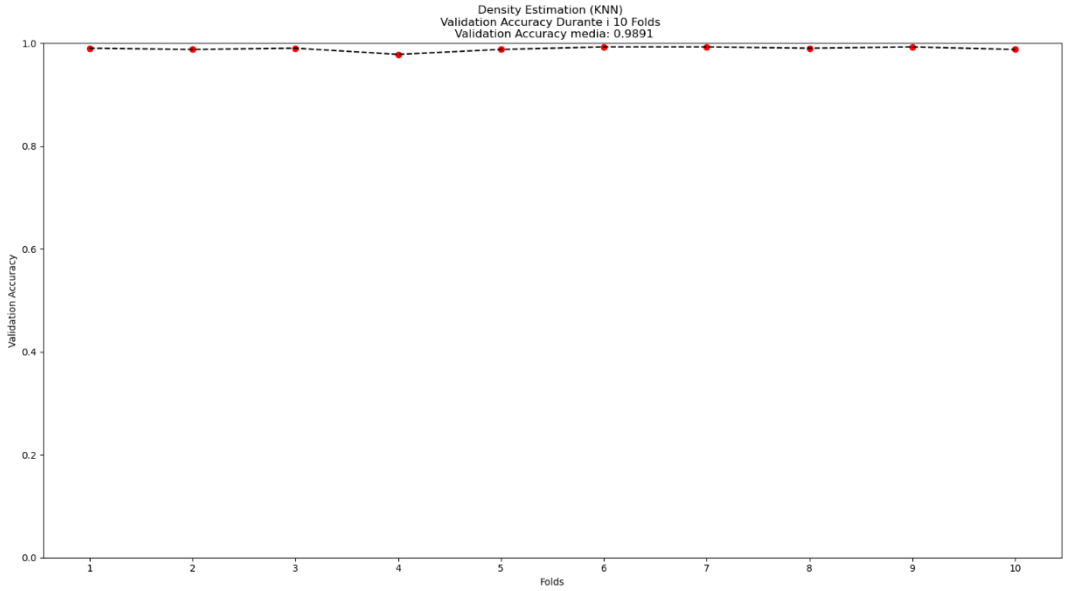
1.2.4: Stima di Densità

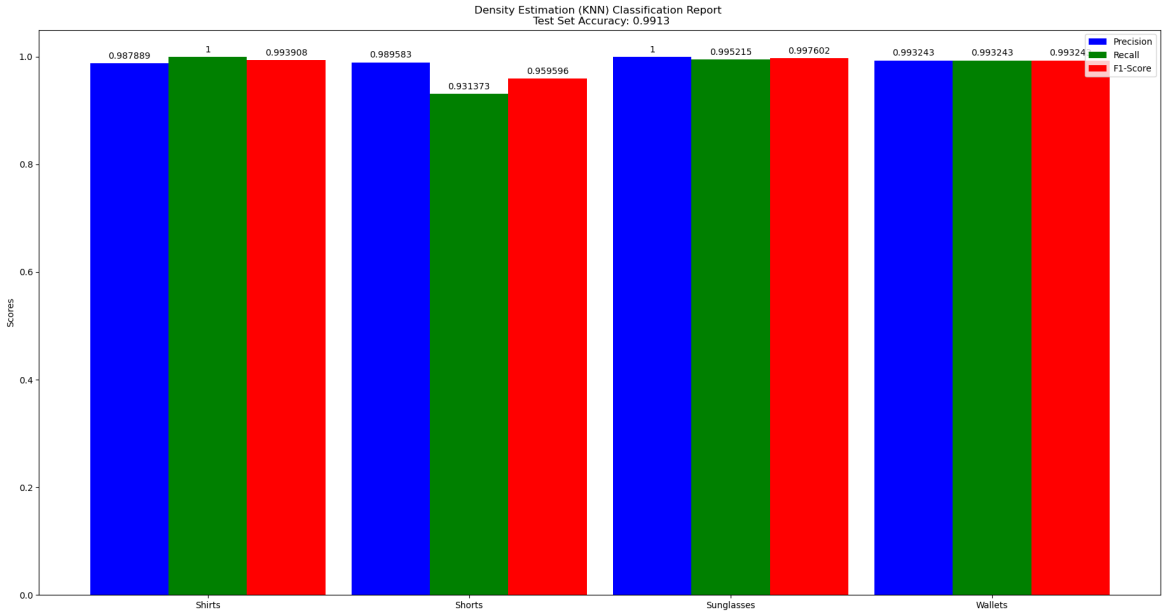
In tale ambito l'analisi condotta ha messo a paragone due differenti tecniche per la stima di densità*: Naïve Bayes Cassiere* (stima di densità parametrica), *K-NN Classifier* (stima di densità non parametrica). Per entrambi i modelli sono stati ottenuti i migliori parametri sviluppando 10 *fold* per la *GridSearch*. Per quanto riguarda *K-NN* si sono osservati i risultati migliori impiegando il valore di *K* pari a 3 (numero di vicini da tenere in considerazione), ed utilizzando per la classificazione la strategia di voto pesato, andando a pesare i vicini in maniera inversamente proporzionale alla distanza dell'esempio da classificare, la metrica di distanza impiegata è stata quella euclidea (*l2*).

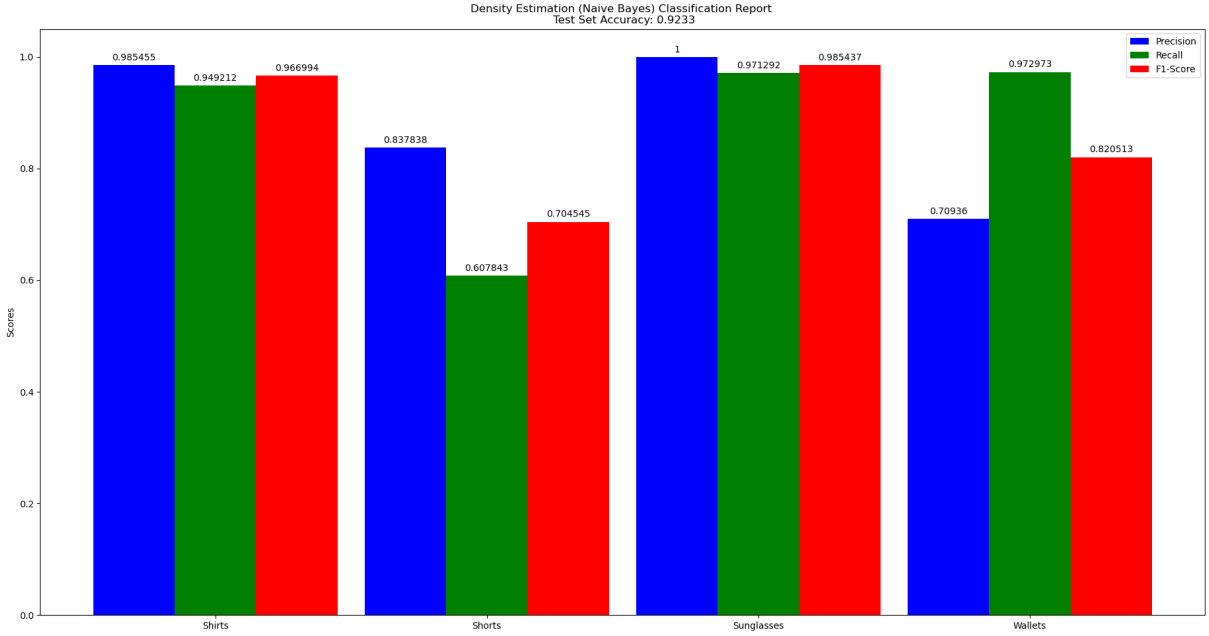
Per quanto riguarda la tecnica *Naïve Bayes*, i risultati migliori sono stati ottenuti andando ad utilizzare il modello che impiega la funzione gaussiana per il calcolo della probabilità, gli iper-parametri analizzati in questo caso sono stati: la probabilità apriori delle classi e la *'var_smoothing'*, ossia un termine additivo sulla varianza del dataset, aggiunto al fine di rendere il modello stabile dal punto di vista del calcolo. Per quanto riguarda il primo parametro la scelta migliore è stata quella di far calibrare al modello la probabilità nel corso dell'algoritmo, in base alle caratteristiche dei dati (si è anche valutata una probabilità proporzionale al numero di elementi delle varie classi ma i risultati non sono stati buoni); mentre il valore migliore di 'smussamento' della varianza è risultato essere pari a 1^-5.

Nel confronto è risultata vincente la tecnica di stima di densità non parametrica tramite *K-NN Classifier*, con risultati superiori sia per quanto riguarda la valutazione tramite cross-validation che sul test set, riuscendo addirittura ad ottenere risultati migliori rispetto ad SVM, è stato infatti raggiunto un valore di accuratezza sul test set circa pari ad 1, andando ad effettuare pochi errori sulle classi meno rumorose ed ottenendo un andamento dell'accuracy durante i vari fold molto stabile .

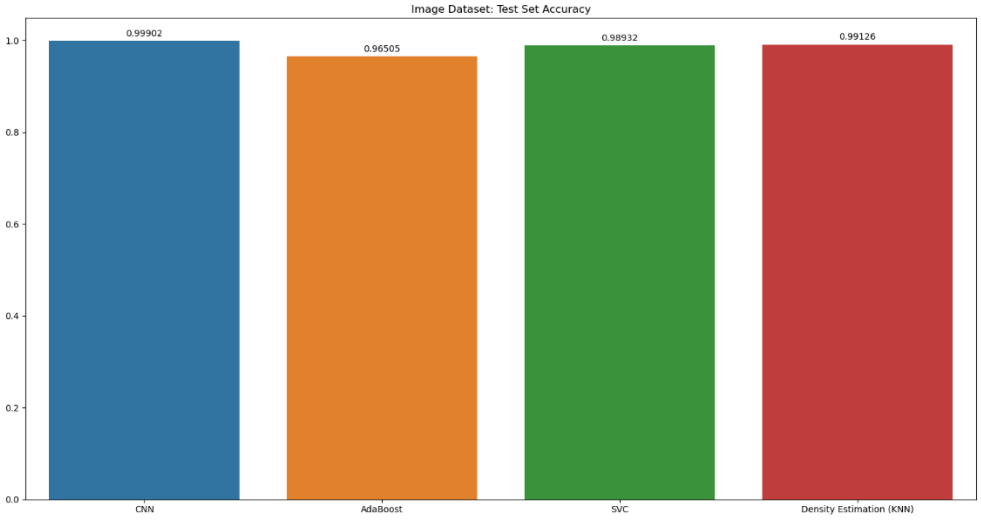

Density Estimation (Naive Bayes)
Validation Accuracy Durante i 10 Folds
Validation Accuracy media: 0.9182
Validation Accuracy
Folds


Density Estimation (KNN)
Validation Accuracy Durante i 10 Folds
Validation Accuracy media: 0.9891
Validation Accuracy
Folds


Density Estimation (KNN) Classification Report
Test Set Accuracy: 0.9913
Precision
Recall
F1-Score
Scores
0.987889
1
0.993908
0.989583
0.931373
0.959596
1
0.995215
0.997602
0.993243
0.993243
0.99324
Shirts
Shorts
Sunglasses
Wallets

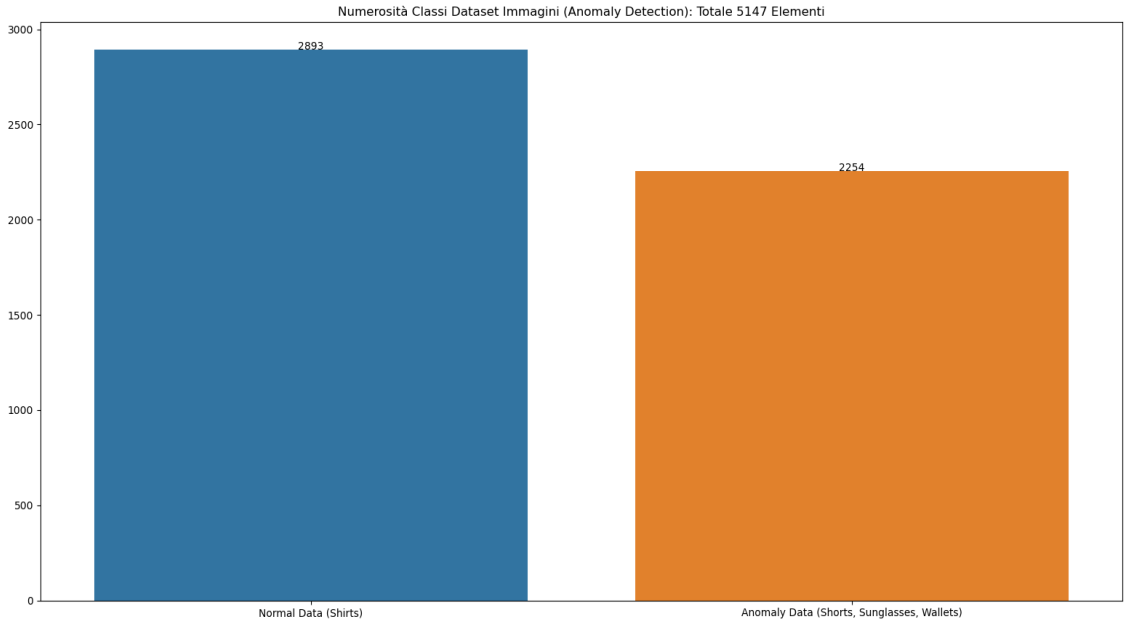


Per terminare, nel grafico sottostante sono stati messi a confronto i diversi modelli analizzati nelle varie sezioni del capitolo, il modello in assoluto migliore è risultato essere la *rete convoluzionale* (come era presumibile), ma ottimi risultati sono stati ottenuti anche tramite stima di densità con *K-NN*. Più in generale si può osservare come ognuno dei classificatori sviluppati, presi singolarmente, vadano a risolvere molto bene il *task* in questione, questo molto probabilmente per via della semplicità del dataset impiegato, composto da classi sbilanciate ma comunque con elementi a bassa dimensionalità, appartenenti ad un dominio applicativo molto semplice. Non è stato riscontrato *overfitting* dei vari modelli e le varie curve rappresentati l'andamento durante i *fold* di *cross-validation* risultano essere quasi tutte molto stabili, segno che i predittori risultano poco sensibili alla composizione degli elementi del dataset e riescono a generalizzare abbastanza bene.



1.3: Task di Semi-Supervised Anomaly Detection

Come detto precedentemente, per il seguente task il dataset è stato rimodellato come da consegna considerando come esempi della classe normale quelli della classe più numerosa e tutti i restanti esempi come anomalie.



Questo ha reso il dataset molto più bilanciato rispetto alla situazione precedente, con un numero di dati anomali pari a 2254 su un totale di 5147 elementi. Al fine di risolvere il problema, si è optato per l'utilizzo di due **reti convoluzionali** (essendo questo modello quello che ha performato meglio per la classificazione multi-classe) combinate al fine di produrre un'architettura di tipo ***autoencoder***. L'addestramento ha seguito le specifiche della traccia, andando ad impiegare *learning semi-supervisionato*, è stato infatti addestrato l'*autoencoder* sui soli esempi di classe normale, andando a discriminare le due classi in base al valore di *outlierness score* restituito dalla rete (*loss* ottenuta sul singolo esempio). La valutazione e la scelta della soglia per la classificazione è avvenuta tramite l'impiego della curva *ROC* e della misura *AUC*. Di seguito verrà descritta la strutture della rete ***encoder*** e ***decoder*** sviluppata.

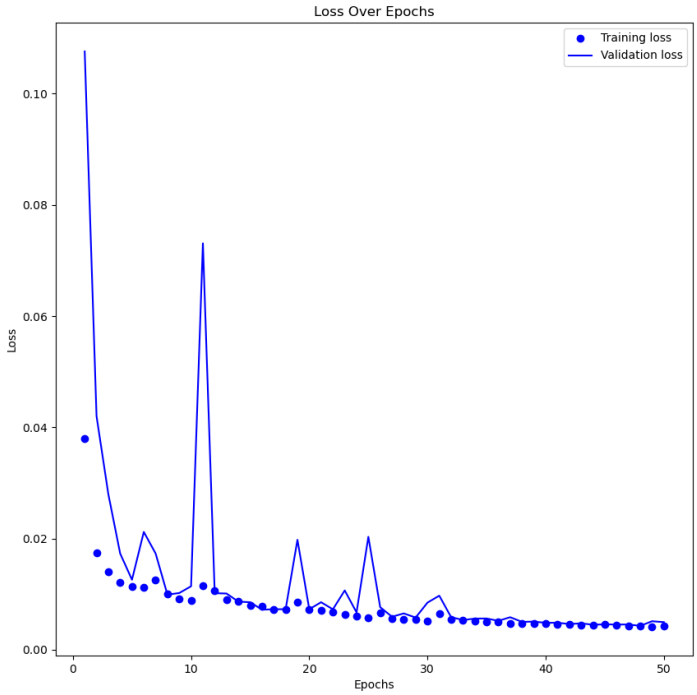
```
# ENCODER:
inp = keras.Input(shape=(64, 80, 3))
x = Conv2D(filters=64, kernel_size=(9, 9), padding='same', activation='relu')(inp)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
encoded = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
```

```
# DECODER:
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(encoded)
x = BatchNormalization()(x)
x = UpSampling2D(size=(2, 2))(x)
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling2D(size=(2, 2))(x)
x = Conv2D(filters=32, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling2D(size=(2, 2))(x)
x = Conv2D(filters=64, kernel_size=(9, 9), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling2D(size=(2, 2))(x)
decoded = Conv2D(filters=3, kernel_size=(3, 3), padding='same', activation='sigmoid')(x)
```

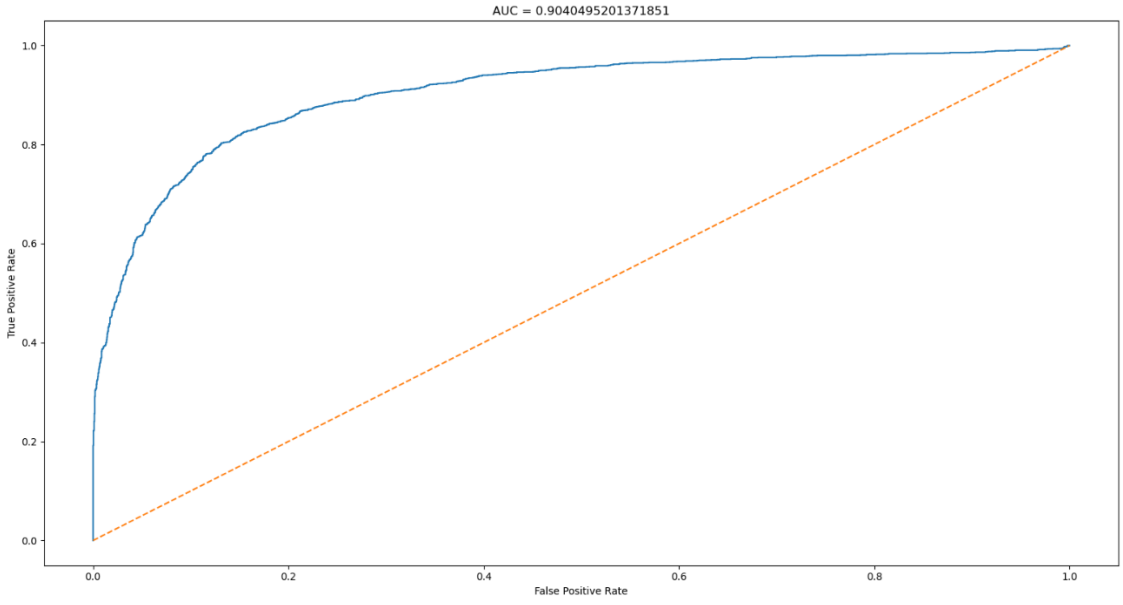
Partendo dal tensore in input su cui è stato effettuato un *reshape* per portarlo ad una dimensione pari a (64, 80, 3), sono stati applicati in sequenza 4 *blocchi Conv2D-BatchNormalization-MaxPooling2D* al fine di codificare l'esempio in uno spazio latente di dimensione (4, 5, 32). Questo perché non si va a diminuire la dimensione del tensore, per quanto riguarda altezza e larghezza, durante la convoluzione (impiego "*zero padding*"), ma si va a dimezzare durante le operazioni di *pooling*. La scelta di effettuare *reshape* sulla dimensione delle immagini è stata fatta allo scopo di ottenere, durante le varie operazioni di dimezzamento, un valore per altezza e larghezza che sia sempre divisibile per 2, potendo quindi ritornare in maniera corretta alla dimensione iniziale tramite il componente **decoder**. Di fatto questa rete va a riprendere interamente la struttura già descritta però in ordine inverso, andando man mano ad aumentare le dimensioni ed il numero di canali tramite l'impiego di *UpSampling2D*. La funzione d'attivazione impiegata per il livello di output è stata la sigmoide, mentre come ottimizzatore e funzione di *loss* rispettivamente *Adam* e *"mean squared error"* .

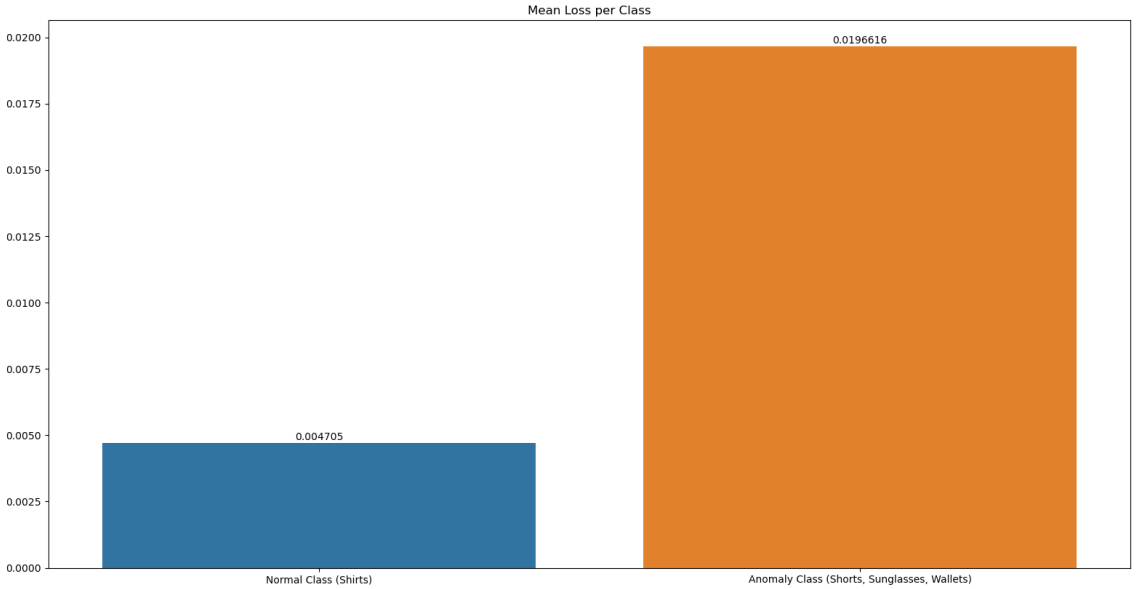
L'architettura sviluppate è risultata molto efficiente sia dal punto di vista del tempo di *training* (si impiegano meno di 800k parametri), che per i risultati ottenuti; altre soluzioni provate, o troppo profonde o meno profonde di questa, risultavano invece avere problemi di *overfitting* o *underfitting* e per tale motivo sono state scartate.

Per l'addestramento sono state sviluppate un totale di 50 epoche con una dimensione del *batch* pari a 32 immagini, a confermare la bontà della soluzione si può notare come l'andamento della *loss* durante il *training* sia quasi sempre decrescente, a meno di alcuni picchi riscontrati sull'insieme di validazione, ma soprattutto si raggiungono valori molto bassi.

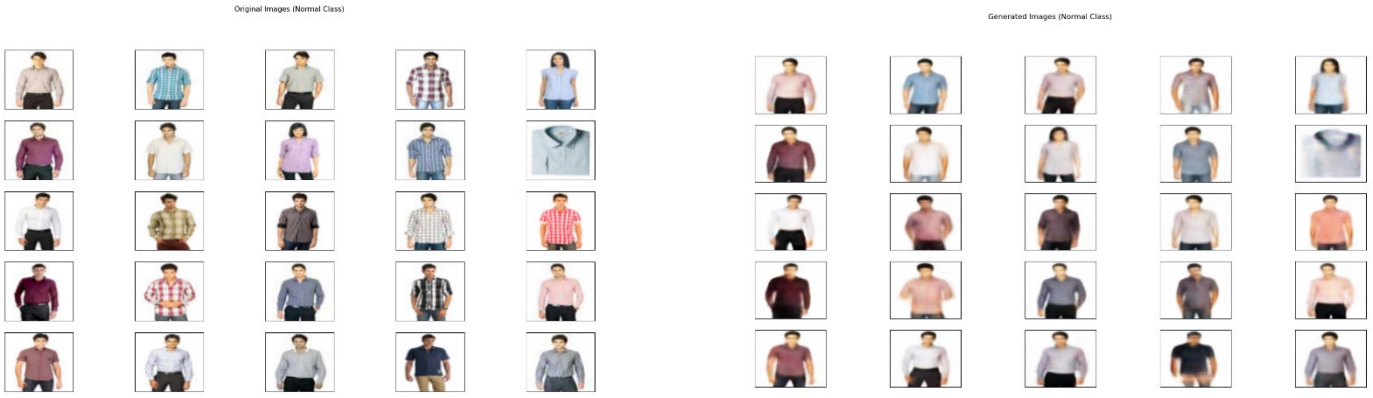


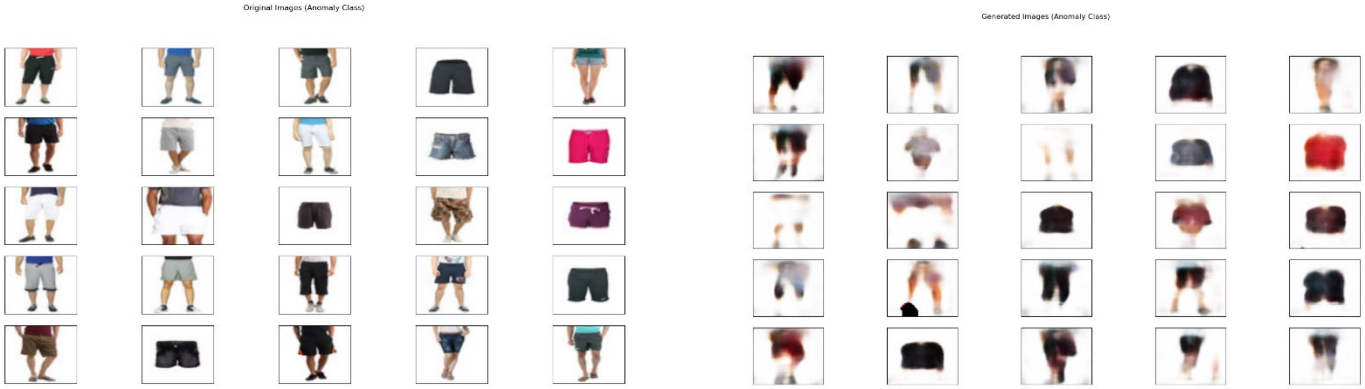
La valutazione sul dataset completo (anomalie e dati normali), ha messo in luce l'effettiva bontà del modello, è stato infatti ottenuto un valore di *AUC* pari a 0.9. Si può inoltre giustificare tale valore elevato andando a notare come effettivamente la rete riesca a ricostruire in maniera molto più corretta (con meno errori), gli esempi visti durante il *training*. Difatti, la media di *outlierness score* per le due classi è nettamente più bassa nel caso della classe normale, e risulta essere oltre il quadruplo per le anomalie.



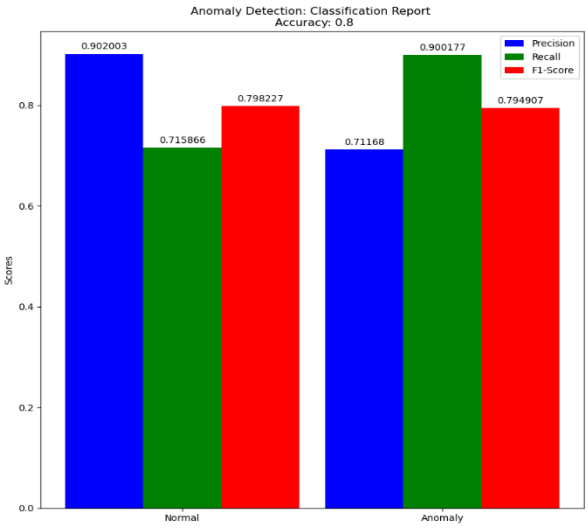


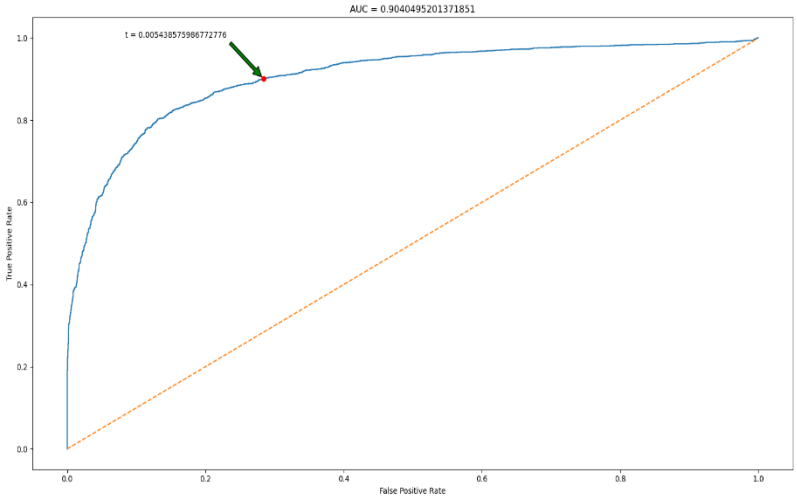
L'idea è quindi che, essendo questi due valori molto distanti tra loro, sia possibile trovare una buona soglia per andare a discriminare gli esempi, inoltre questa discrepanza si poteva già intuire andando a visualizzare il risultato in termini di immagini prodotte dalla rete. Come si può notare dalla figura sottostante, gli elementi della classe normale vengono riprodotti leggermente sfocati ma comunque molto vicini agli originali, mentre per quanto riguarda la classe anomala (sconosciuta alla rete), le riproduzioni sono sostanzialmente macchie indistinte di colori molto tendenti in alcuni casi alla forma delle magliette. Questo probabilmente perché avendo visto unicamente gli esempi '*Shirts*', la rete va ad abbozzare le immagini nelle forme apprese.

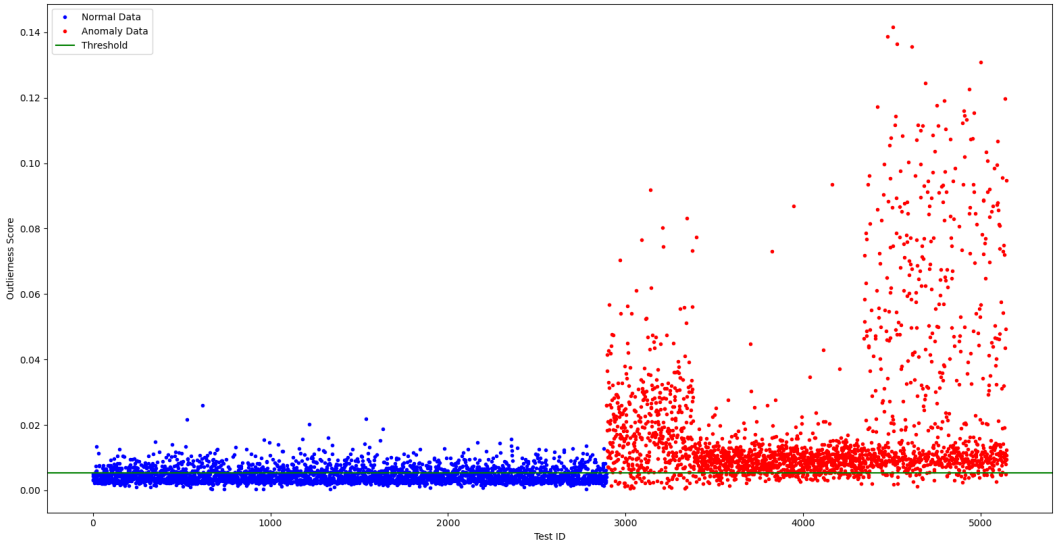




A corredo dell'analisi ho effettuato ulteriori valutazioni andando a completare il processo di classificazione, è stato infatti selezionato come valore di *threshold* il primo valore sulla curva per cui si ottiene tasso dei reali positivi pari a 0.9. Tale valore mi ha permesso di discriminare in maniera abbastanza corretta gli esempi, ottenendo un accuratezza pari a 0.8!





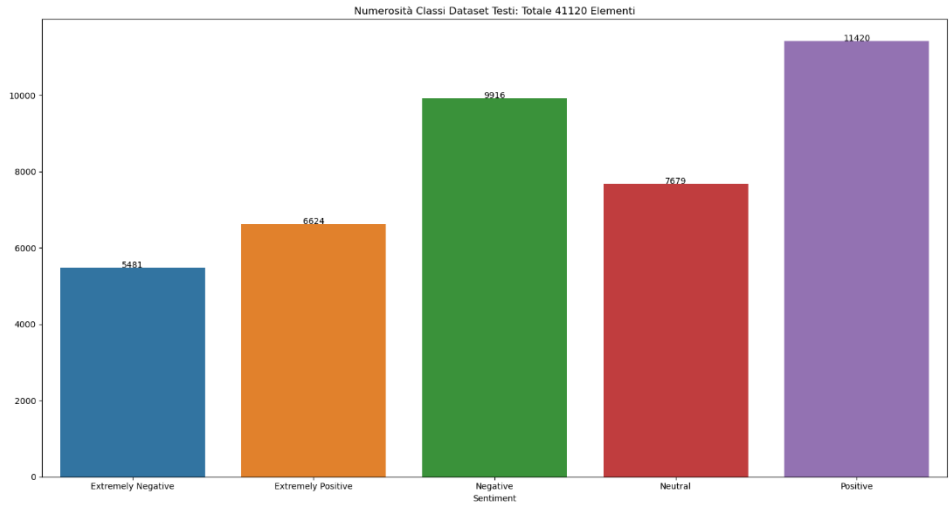


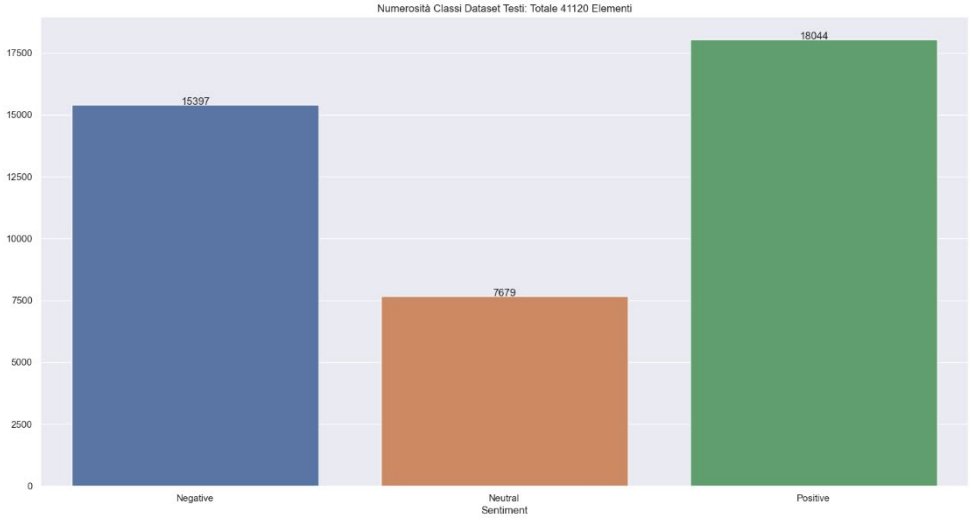
NOTA: Avendo impiegato uno script da me sviluppato per il caricamento del dataset, l'ordine degli elementi dipenderà da vari fattori, quale ad esempio il sistema operativo della macchina, per tale motivo durante l'esecuzione dello script per la risoluzione del *task* di *anomaly detection* si potrebbero avere risultati in termini di *ROC curve* e *AUC* differenti. Questo perché il calcolo ed il *dump* dei valori di *outlierness score* è stato fatto secondo l'ordine delle immagini che si ha sulla macchina su cui ho sviluppato il progetto. Questo fa sì che, impiegando un ordinamento degli elementi differente, potrebbe succedere che lo score dell'immagine i-esima non faccia riferimento all'effettivo dato che si trova in tale posizione nel *dataset*. Andando ad eliminare il file '*outlierness'* contenuto nella cartella '*drive'*, e ricalcolando lo score, si dovrebbero ottenere i risultati sopra esposti.

Capitolo 2: Dataset di Testi

2.1: Descrizione e Preprocessing dei Dati

Il dataset di testi si colloca sostanzialmente nell'ambito della *sentiment analysis*, è infatti composto da 2 sole colonne e contiene una collezione di tweet estratti dalla piattaforma Twitter, a cui sono associati delle *label* che possono essere: '*Extremely Negative*', '*Extremely Positive*', '*Negative*', '*Positive*' e '*Neutral*'. Come si può vedere dal grafico sottostante, il numero totale di elementi supera i 40 mila record e risulta inoltre essere abbastanza sbilanciato; al fine di andare a migliorare le prestazioni dei modelli e bilanciare leggermente il dataset, scelta progettuale è stata quella di ridurre il numero di classi andando ad accorpare i dati considerando solamente le etichetta '*Positive*', '*Negative*' e '*Neutral*'.

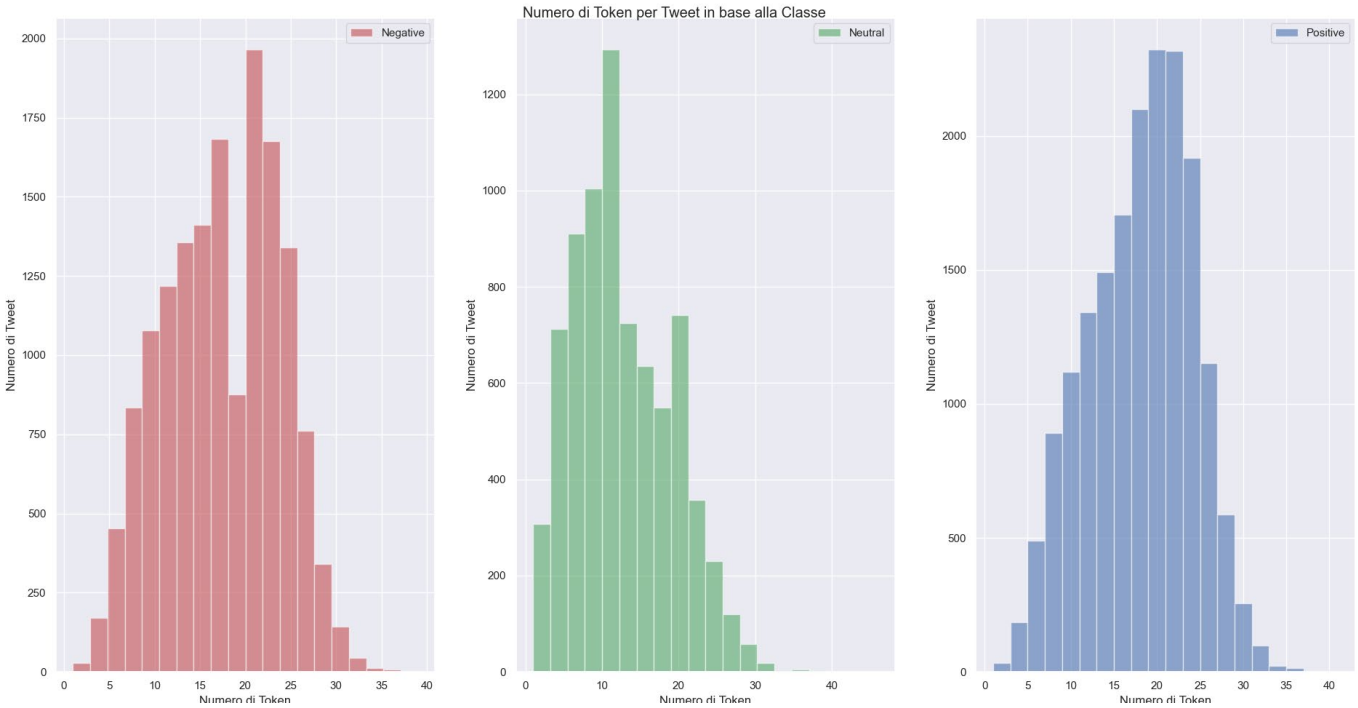




Sono state quindi applicate varie operazioni di rimozione dei duplicati e pulizia dei record: come prima cosa sono stati portati tutti i tweet in lettera minuscola, sono stati quindi rimossi i caratteri speciali (emoticon, '@', '#', ed altri simili), sono stati rimossi eventuali link *html*, indirizzi *e-mail* e *username* di utenti, infine è stata rimossa la punteggiatura, caratteri numerici e le *stopwords* (per la lingua inglese). Andando inoltre ad analizzare per via grafica le '*wordcloud*' relative alle 5 classi (sono stati considerato i top 200 termini la cui dimensione indica un'elevata presenza nei record di tale classe), si può notare come i termini '*covid*' e '*coronavirus*' siano presenti con grande frequenza in ogni tipologia di tweet.

Scelta progettuale è stata quella di rimuovere tali parole considerandole come delle *stopwords*, inoltre si può anche notare la somiglianza tra le varie classi per quanto riguarda la frequenza di alcuni termini, cosa che ha avvalorato la scelta di ridurre il numero di etichette. Bisogna inoltre precisare la scelta di non andare a rimuovere eventuali *hashtag* (solo il carattere '#'), perché questi spesso potrebbero essere legati al *sentiment* della frase e inoltre corrispondono sostanzialmente a termini significativi per il contesto di riferimento del tweet.

Sono state poi effettuate operazioni di lemmatizzazione e *stemming* delle varie parole che compongono i tweet, lo scopo è stato quello di andare a ridurre il più possibile la dimensione del vocabolario (insieme di tutti i termini che appaiono nel dataset), in modo tale da ridurre anche la dimensionalità del dataset finale una volta applicate eventuali tecniche di vettorizzazione. Infine è stata effettuala la *tokenizzazione* dei record. Come si può notare dal grafico a barre sottostante, il numero medio di *token* che costituiscono i record distinti per classe, è abbastanza simile per tutte e tre, con la maggioranza dei dati che si aggirano nell'intervallo 10-20, con una media complessiva pari a 16.5 ed un massimo pari a 45 *token*, questa informazione è stata utile nel proseguo dell'analisi per la scelta di alcuni parametri. Inoltre si può osservare come i tweet con il numero inferiore di *token* facciano parte della classe '*Neutral*'.



Dopo l'applicazione delle varie procedure di processamento dei dati, la numerosità del vocabolario è risultata essere pari a 32909 termini, per quanto riguarda il numero di parole più frequenti del vocabolario da utilizzare per la costruzione dei dataset di training e testing del modello, è stato trattato questo valore sostanzialmente come un iper-parametro dei modelli. Per quanto riguarda le tecniche non basate su reti neurali, la scelta è stata per l'impiego dei top 500 termini per numero di occorrenze. Questa è stata dettata da varie prove in via sperimentale, partendo dall'osservazione che impiegare anche poche migliaia di parole rendeva le procedure di training proibitive dal punto di vista temporale, e che molte parole vengono comunque osservate poche volte, sono state fatte diverse prove pratiche ed è stato notato sostanzialmente un netto peggioramento delle prestazioni dei modelli quando si vanno ad impiegare meno di 400 termini. Mentre i risultati erano pressoché uguali con un valore maggiore o uguale a 600, al costo di un aumento netto nel tempo di addestramento. Per quanto riguarda i modelli basati su reti, è stato possibile andare a considerare un numero nettamente maggiore di termini, arrivando ad impiegarne ben 20 mila per la costruzione del dataset.

Le tecniche di vettorizzazione impiegate sono state principalmente '*tfidf*', per i modelli di machine learning ed alcune architetture di rete, mentre sempre nell'ambito di analisi di architetture di rete si è scelto anche l'utilizzo di un dataset numerico basato su liste di indici: ogni record testuale originale veniva sostituito con una lista numerica di indici, dove ogni valore corrisponde all'indice che assume la relativa parola che compare nel tweet all'interno del vocabolario. Eventuali considerazioni aggiuntive sulle operazioni di vettorizzazione verranno trattate nel proseguo durante l'analisi dei vari modelli sviluppati.

2.2: Task di Classificazione

Le considerazioni fatte precedentemente per la risoluzione del problema di classificazione sul dataset di immagini, continuano a valere anche in questo caso. Verrà effettuata la ricerca dei migliori iper-parametri dei modelli mediante l'impiego di *GridSearchCV* e le metriche utilizzate sono sostanzialmente le stesse. Anche in questo caso, i *fold* sviluppati per il processo di *parameter tuning* variano in base alla complessità ed al tempo di esecuzione dei vari algoritmi di *learning*. L'analisi dei modelli si è inoltre concentrata anche sull'utilizzo e la comparazione di diverse tecniche di vettorizzazione dei record testuali, la scelta di queste tipologie di tecniche è stata infatti molto influente per quanto riguarda i risultati prodotti dai

classificatori. Inoltre, è stato impiegato il valore di 0.15 come percentuale di elementi da impiegare per la costruzione del dataset di test, andando quindi ad ottenere un totale di circa 35 mila elementi per il *training set* e poco oltre i 6000 per il *test set*. Questa scelta è stata dettata dall'idea di utilizzare un maggior numero di esempi durante la fase di addestramento, in modo da migliorare le capacità di generalizzazione dei modelli, ed anche per non appesantire tale processo rendendolo troppo lento.

Si andranno quindi a descrivere più dettagliatamente le soluzioni applicate.

2.2.1: Reti Neurali

L'analisi svolta per quanto riguarda i modelli basati su reti ha lo scopo di mettere a paragone differenti architetture al fine di trovare quella che performa meglio per questa tipologia specifica di dataset. Si è scelto l'utilizzo della tecnica di vettorizzazione che si basa sulla trasformazione dei record in *liste di indici*, questo perché l'utilizzo di *'tfidf'* implicava andare a ridurre di molto il numero delle parole considerate, per via dell'eccessiva dimensione dei tensori dei pesi delle reti (spesso la dimensione rendeva proibitivo il *training* poiché eccedeva la memoria utilizzabile). Per questo motivo si è optato per la tecnica citata, che ha permesso l'analisi di un numero molto elevato di termini, ossia i 20 mila più frequenti. Complessivamente sono state analizzate 4 differenti architetture di rete, rispettivamente: ***LSTM***, ***RNN***, **C*onvoluzione1D*** e ***Reti Dense***. Inoltre, è risultato molto influente nella bontà del processo di classificazione sia il numero di epoche di *train*, che la dimensione del *batch* di record in input, per questo motivo i vari modelli sviluppati sono stati provati preventivamente in via sperimentale al fine di ricercare le migliori combinazioni di tali valori (sono stati sostanzialmente considerati come iper-parametri). Altro valore influente per la tecnica di vettorizzazione impiegata è stata la scelta del valore di '*sequence_len*', una volta trasformati i record nelle rispettive liste degli indici che assumono le parole all'interno del vocabolario, è stato necessario andare ad uniformare la lunghezza di tali liste in modo tale da ottenere una dimensione unica condivisa. A questo scopo è stato utilizzato il parametro precedentemente citato, ed inoltre è stata anche molto utile l'analisi condotta sul numero medio di *token* che compongono i *record*, che ha permesso di evitare di impiegare un valore troppo alto, che avrebbe comportato l'eccessiva presenza di zeri nei *record* (perché si effettua *zero padding* nel caso in cui il numero di token sia inferiore a '*sequence_len*'), o troppo basso e che quindi poteva comportare la perdita di parole per alcuni elementi del dataset. Siccome è stato osservato essere 45 il numero massimo di token nei vari record, si è scelto il valore di 50 come dimensionalità del dataset trasformato.

Esempio record del dataset: [109, 108, 457, 621, 50, 621, 23, 74, 717, 791, 1181, 25, 750, 8774, 5790, 48, 1113, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

Come si può vedere dall'esempio, i differenti valori non ricadono in un intervallo preciso, o meglio sono contenuti nell'intervallo [0, numero di token considerati], sarebbe perciò opportuno andare a normalizzare i vari esempi, questa operazione non è stata però effettuato per via dell'impiego del *layer* di *Embedding* (addestrabile) su tutte le architetture di rete sviluppate. Il compito di tale livello è infatti quello di andare a mappare lo spazio dei valori del vocabolario in uno spazio a dimensione ridotta, per la risoluzione del problema è stata scelta la dimensione dell'output pari a 128, inoltre nel fare ciò i risultati prodotti saranno tutti nell'intervallo [0, 1].

L'architettura basata su ***LSTM*** consta di 3 livelli *lstm* ognuno con un numero di neuroni pari a 128, inoltre tali livelli sono stati impiegati in maniera bidirezionale, sostanzialmente è come se si utilizzassero due moduli *lstm* in ogni singolo livello, di cui uno viene addestrato sulla sequenza originale in input, mentre il secondo viene addestrato usando la sequenza inversa (il numero di neuroni è infatti raddoppiato rispetto a quello specificato). I tensori in output vengono poi passati ad una serie di 3 livelli che implementano la convoluzione monodimensionale con filtri di dimensione 7, restituendo ognuno 128 canali in uscita (vengono usati *kernel* con un'unica dimensione rispetto alla convoluzione2d in cui si impiegano matrici). Infine, si termina il processo di classificazione impiegando 3 livelli densi con funzione di attivazione del livello di output *'softmax'* (per avere un vettore di probabilità di appartenenza di un elemento ad una classe). Ovviamente vengono anche effettuate operazioni di *dropout* (sia sui livelli densi che *lstm*), *batchnormalization* e *spatialdropout* (si vanno ad inibire non i singoli neuroni ma bensì interamente porzioni delle *feature map* generate), al fine di evitare *overfitting*.

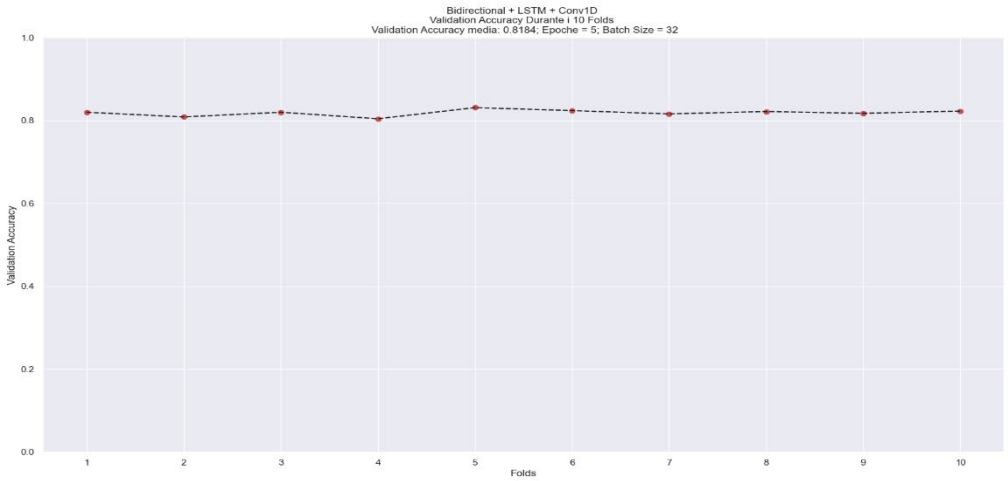
```
model = Sequential()
model.add(Embedding(input_dim=max_tokens + 1, output_dim=128, input_length=sequence_len, trainable=True))
model.add(SpatialDropout1D(0.3))
model.add(Bidirectional(LSTM(128, dropout=0.25, return_sequences=True)))
model.add(Bidirectional(LSTM(128, dropout=0.25, return_sequences=True)))
model.add(Bidirectional(LSTM(128, dropout=0.25, return_sequences=True)))
model.add(SpatialDropout1D(0.3))
model.add(Conv1D(128, 7, padding='same', activation='relu'))
model.add(Conv1D(128, 7, padding='same', activation='relu'))
model.add(Conv1D(128, 7, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(GlobalMaxPool1D())
model.add(Dense(64, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer))
model.add(Dropout(0.3))
model.add(Dense(len(classes), activation='softmax'))
```

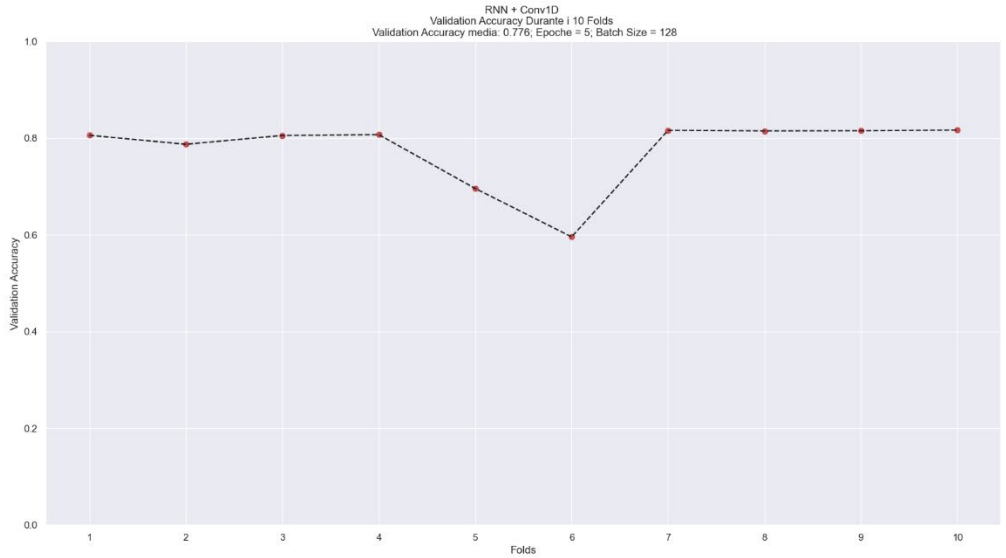
L'architettura basata su ***RNN*** va sostanzialmente a riprendere quanto descritto precedentemente, le differenze sono l'uso di meno livelli *ricorrenti* e di *convoluzione1d* (solo 2 rispetto ai 3 livelli impiegati nel modello precedente), inoltre la dimensione

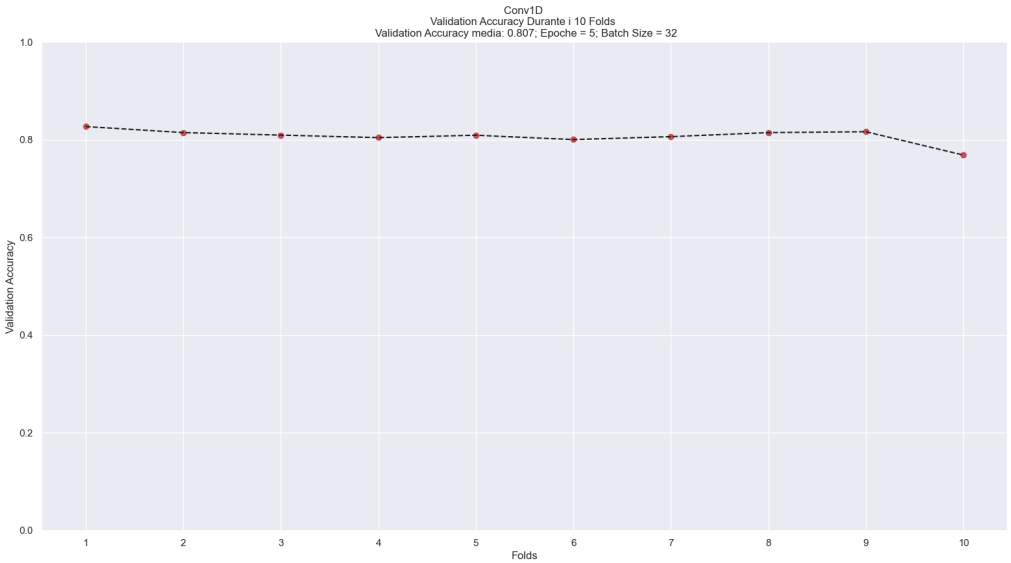
dei kernel convoluzionali impiegati è pari a 5 e si va ad impiegare una maggiore percentuale di dropout sui neuroni dei livelli *RNN*. In questo caso, scelta progettuale è stata quella di non impiegare operazioni bidirezionali per evitare di aumentare troppo il numero di parametri e magari rendere la rete maggiormente soggetta ad *overfitting,* ma anche per valutarne l'effettiva utilità.

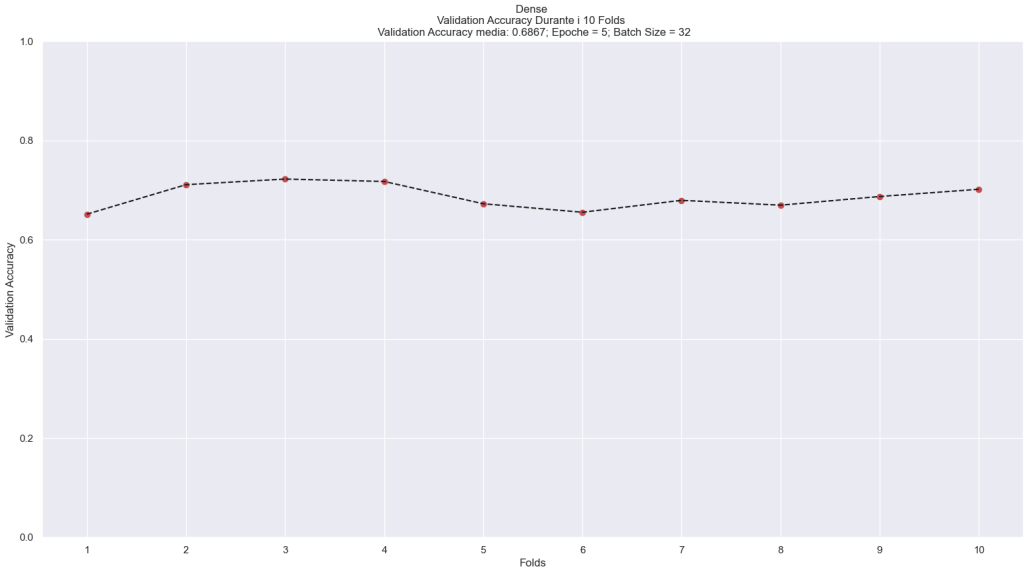
Per quanto riguarda l'architettura basata unicamente sull'uso di ***Convoluzione1d***, è composta da tre livelli convoluzionali con kernel di dimensione 7 e 128 canali prodotti, seguiti da livelli densi. Mentre la rete ***Fully Connected*** è stata sviluppata impiegando in sequenza 3 *layer* nascosti, per portare la dimensione del tensore in ingresso da 6400 (output '*flattenizzato'* del *layer* di *Embedding*), 1250, 256, 64, fino ad ottenere sull'ultimo livello della rete dimensione pari al numero di classi.

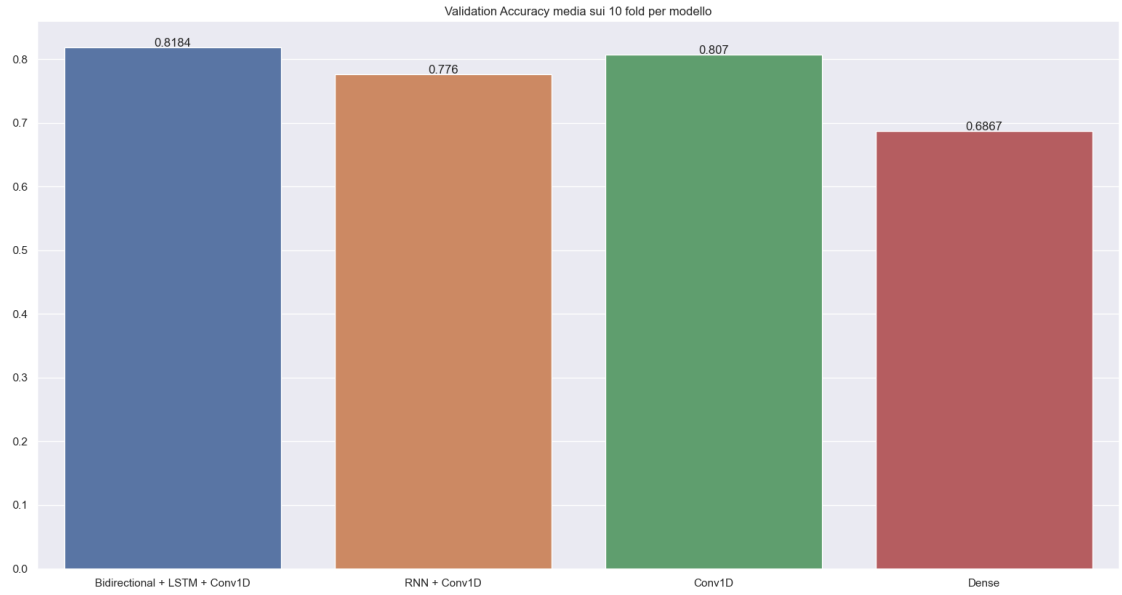
Tutti i modelli trattati vanno ad impiegare come funzione d'attivazione finale la *'softmax'*, inoltre anche le specifiche relative alla compilazione di tali modelli sono le stesse per ognuno: è stato impiegato *Adam* come ottimizzatore con un *learning rate* iniziale pari a 10^-3, come funzione di *loss* da minimizzare è stata utilizzata '*sparse categorical cross-entropy'*. I parametri che possono invece variare sono numero di epoche di addestramento e dimensione del *batch*, come detto la selezione di tali valori è avvenuta in via sperimentale provando i risultati dei vari classificatori al variare di questi. Per le epoche si è utilizzato per tutti i modelli il valore 5, mentre come *batch* sono stati impiegati 128 record nel caso di rete basata su *RNN*, mentre 32 per gli altri casi. Nei grafici successivi si può osservare l'andamento dell'accuratezza durante i vari *fold* del processo di *cross-validation* e il valore medio ottenuto per modello considerato. Si può notare come, a meno della *rete densa*, tutti i modelli si vanno ad assestare su un valore vicino a 0.8 di *accuracy* media, sfiorato dall'architettura *RNN* per via di un calo di prestazioni avuto in corrispondenza delle epoche centrali. Non ci si deve però stupire del fatto che sia il modello di rete *LSTM* quello che ottiene le prestazioni migliori, siccome tale tecnologia riesce a performare molto bene con questo tipo di vettorizzazione essendo che mantiene intatte le relazioni di precedenza e successione dei termini nel testo.


Bidirectional + LSTM + Conv1D
Validation Accuracy Durante i 10 Folds
Validation Accuracy media: 0.8184; Epoche = 5; Batch Size = 32
Validation Accuracy
Folds


RNN + Conv1D
Validation Accuracy Durante i 10 Folds
Validation Accuracy media: 0.776; Epoche = 5; Batch Size = 128
Validation Accuracy
Folds

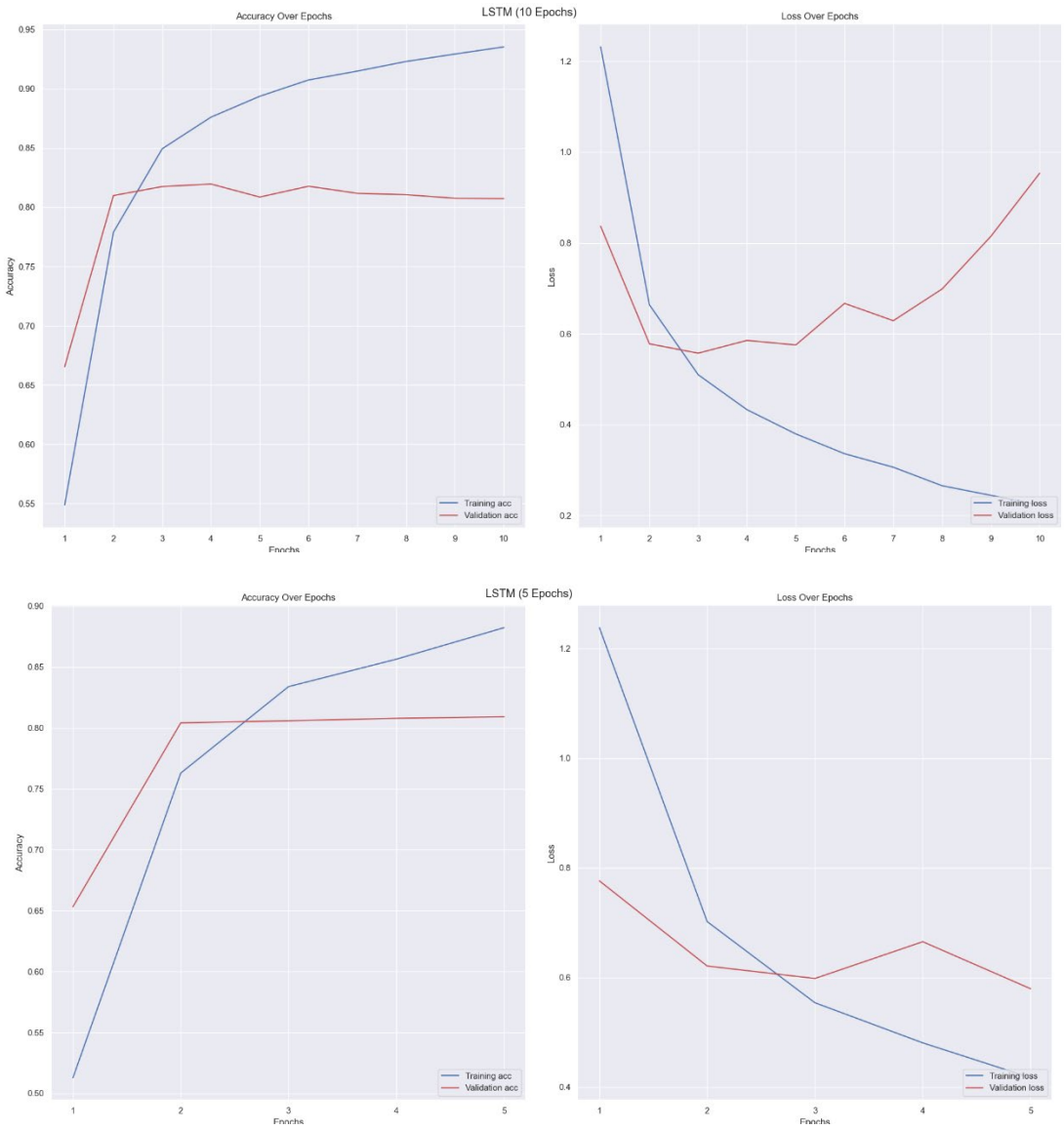

Conv1D
Validation Accuracy Durante i 10 Folds
Validation Accuracy media: 0.807; Epoche = 5; Batch Size = 32
Validation Accuracy
Folds



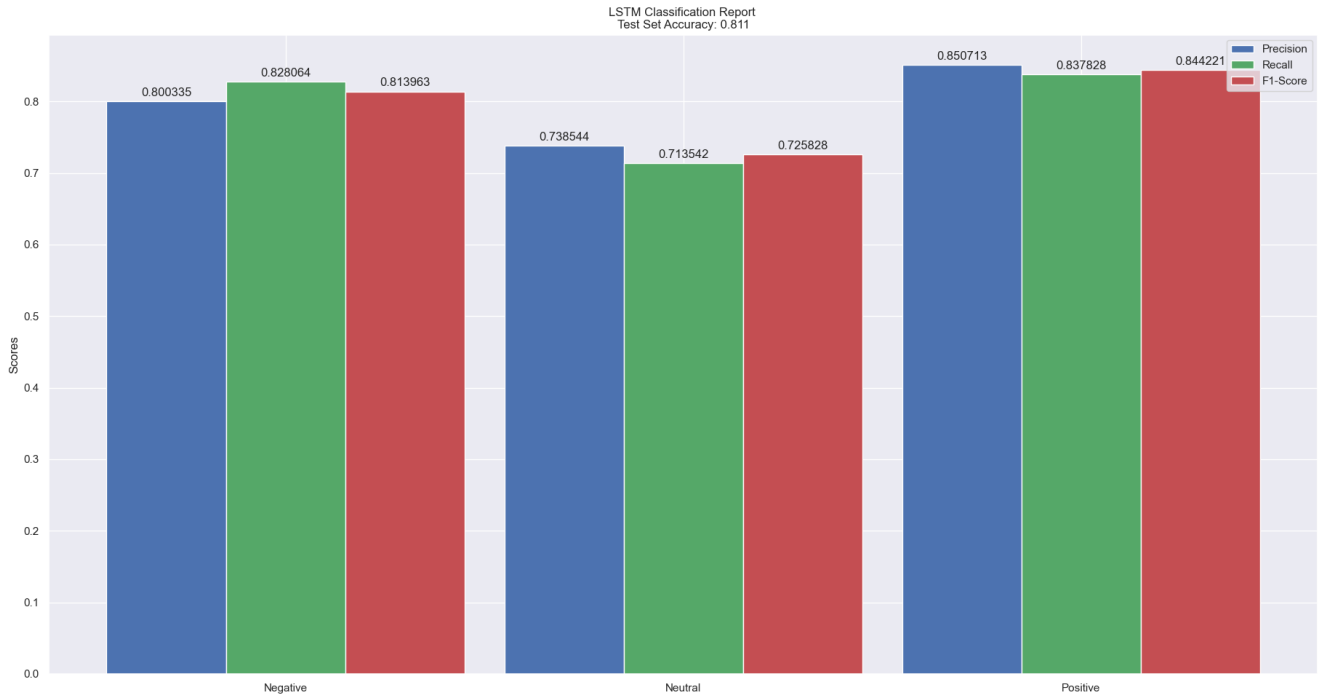


Per concludere l'analisi, è stato sviluppato nel dettaglio il processo di *training* della rete che ha performato meglio durante la fase di *model selection,* in questo caso si può effettivamente andare a giustificare la scelta del valore 5 come numero di epoche. Dal grafico dell'accuratezza e della *loss* ottenute sviluppando 10 epoche è chiaro come in corrispondenza della quinta epoca il valore di *validation accuracy* si vada ad assestare su un minimo e resti sostanzialmente invariato, quello che invece va sempre di più ad aumentare è l'accuratezza sul *dataset* di *training*. Inoltre, analizzando l'andamento della *loss*, si può vedere come da tale soglia in poi vada drasticamente ad aumentale la *validation loss* e diminuire la *training loss.* Si può quindi

concludere appurando come effettivamente il modello risulti soggetto ad *overfitting* andando ad impiegare un numero di epoche superiore a 5.

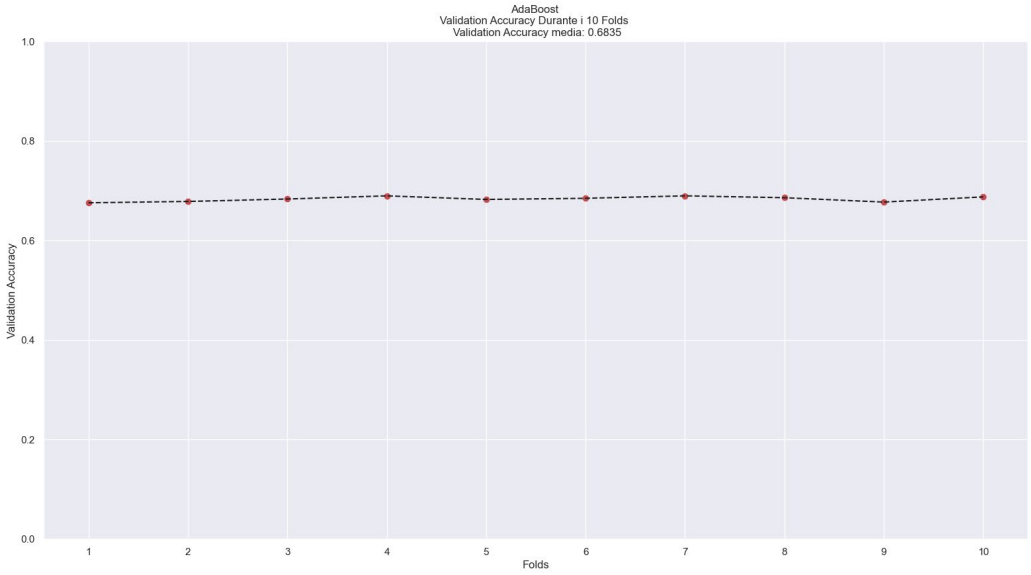


Infine, è stata testata l'architettura una volta addestrata correttamente sul dataset di test, ottenendo risultati simili a quanto già osservato, per quanto riguarda l'accuratezza si raggiunge un valore di 0.81 e, come era prevedibile, i risultati peggiori si hanno nella classificazione della classe meno numerosa, ossia *'Neutral'*, sia per quanto riguarda il numero di falsi positivi che di falsi negativi. Per tale label il valore più basso risulta essere la metrica *recall*, ad indicare molte classificazioni erronee di tali esempi in altre classi, probabilmente dovuto al fatto che i termini utilizzati nei vari record di questa tipologia sono abbastanza comuni anche per le altre label e magari anche privi di un forte significato semantico.

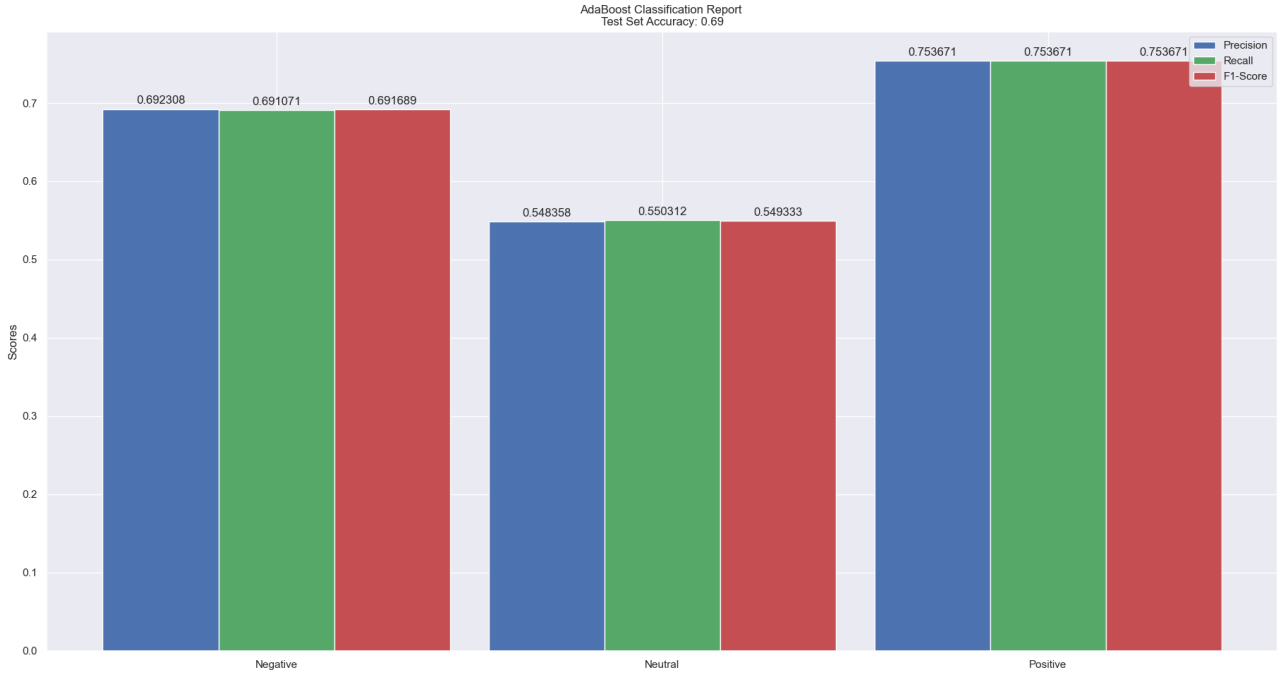


2.2.2: AdaBoost

Come è avvenuto per l'analisi del dataset di immagini, anche in questo caso si è scelto l'impiego di *AdaBoost* con *weak learner* di base, è stata utilizzata come tecnica di vettorizzazione *'tfidf'* andando a mantenere solo i primi 500 termini del vocabolario per numero di occorrenze. La ricerca dei migliori iper-parametri è stata svolta come detto tramite tecnica *GridSearchCV* sviluppando un totale di 5 *fold* per combinazione, anche in questo caso sono stati valutati diversi valori per i parametri: numero di *estimator* e *learning rate*. La migliore configurazione è stata raggiunta impiegando un totale di 300 *weak learner* e *learning rate* pari a 0.5, ottenendo accuratezza media sui vari step pari a 0.68 che, come si può notare dal grafico sottostante, è stata raggiunta anche per il processo di valutazione tramite *10-fold cross-validation.*



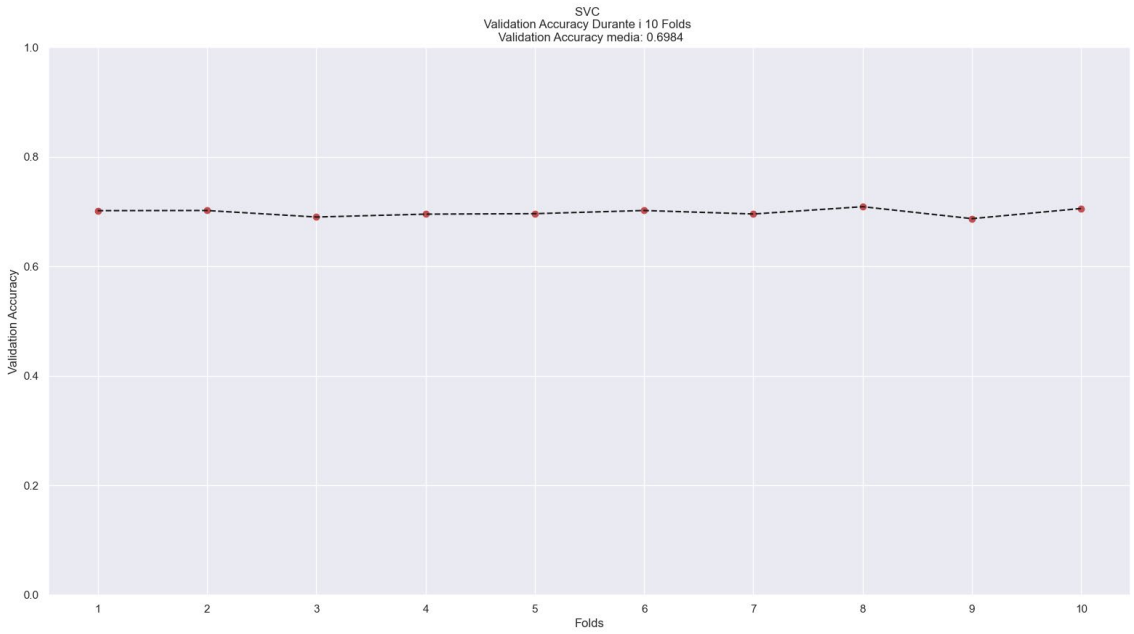
Infine è stato provato il modello sul dataset di test, come si può notare i risultati ottenuti, oltre ad essere sostanzialmente in linea con quelli precedenti, vanno a seguire quella che è la distribuzione degli elementi nelle varie classi. Sull'etichetta *'Positive'* sono stati raggiunti buoni risultati nel processo di classificazione, essendo questa la più numerosa, non buoni sono invece i risultati ottenuti per quanto riguarda la classe meno numerosa *'Neutral'*, con le metriche di *precision* e *recall* che si assestano entrambe su un valore circa pari a 0.55, questo indica che il numero di reali positivi è di poco più alto rispetto a falsi positivi e falsi negativi.



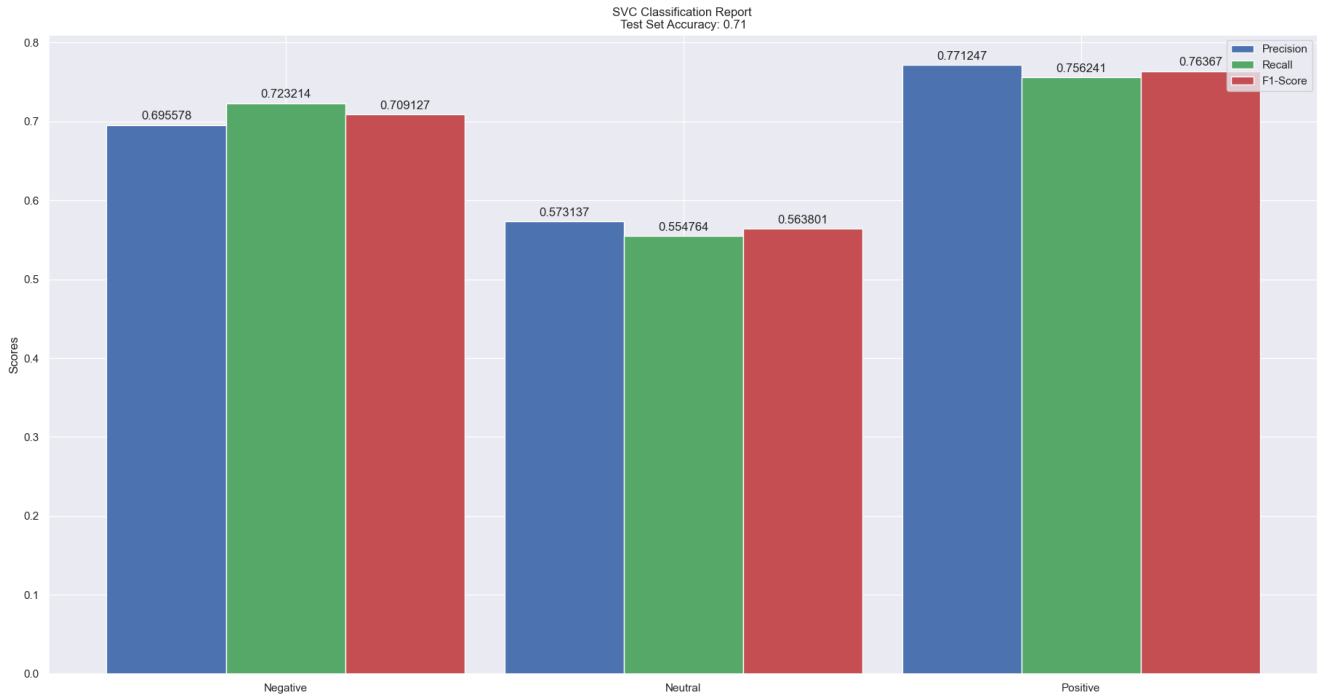
2.2.3: SVM

Lo sviluppo di classificatori basati su *SVM* è stato forse il lavoro più oneroso di tutta l'analisi condotta, non per quanto riguarda la complessità ma bensì il tempo, questo ha reso quasi intrattabile andare ad utilizzare un numero superiore a 500 termini del vocabolario. Per lo stesso motivo è stato scelto il valore 3 come numero di *fold* da sviluppare nei vari step di *GridSearch,* in cui sono stati analizzati diversi valori per gli iper-parametri: '*C*', *'kernel'*, e *'gamma'*.

La configurazione migliore è stata ottenuta per i valori: C*=2, kernel='rbf' , gamma='scale';* raggiungendo circa 0.70 come accuratezza. Come era presumibile il *kernel* ottimale è risultato essere quello gaussiano che, andando potenzialmente a trasformare lo spazio in uno a dimensione tendente ad infinito, permette la risoluzione di problemi anche molto difficili, aventi distribuzioni di elementi delle varie classi che possono assumere forme molto complesse. Si può osservare un andamento molto stabile dei risultati ottenuti durante il processo di valutazione mediante *10-fold cross-validation,* con accuratezza media pari al valore precedentemente ottenuto durante la fase di *parameter-tuning.*



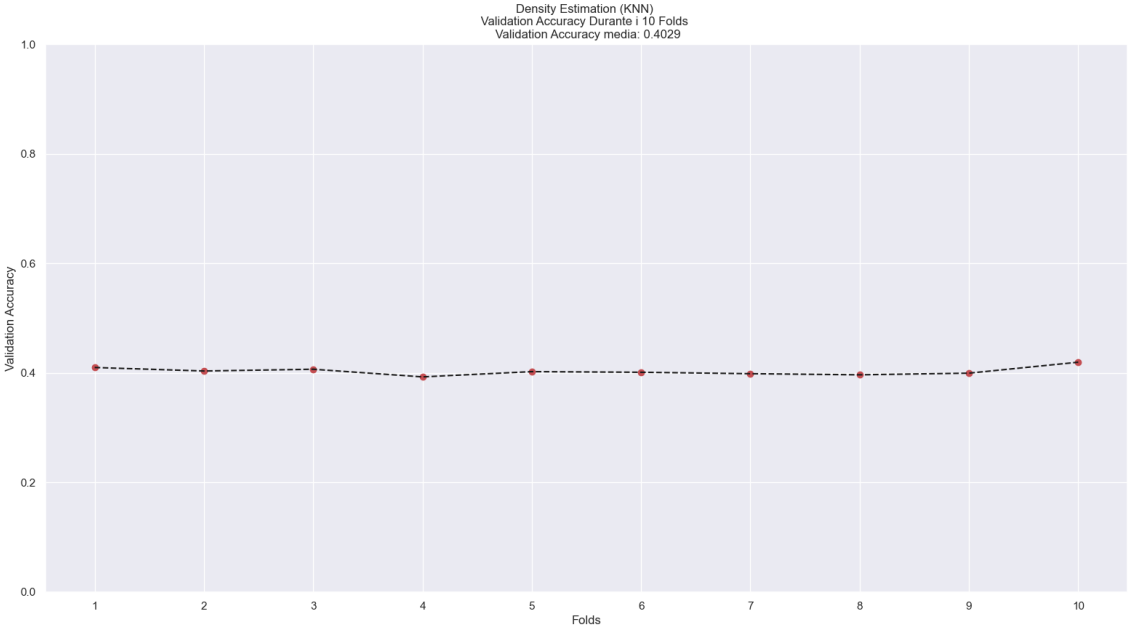
Per quanto concerne le prestazioni del modello sul *test set,* si ottengono performance superiori, raggiungendo 0.71 di accuratezza. Esattamente come successo nei casi precedenti, anche per *SVC* il peggioramento si ottiene sulla classe '*Neutral*', ottenendo invece risultati leggermente migliori rispetto *AdaBoost* per quanto riguarda le restanti due etichette, si riduce infatti il numero di falsi negativi rilevati per la label *'Negative'*, e rispettivamente anche il numero di falsi positivi per *'Positive'*.



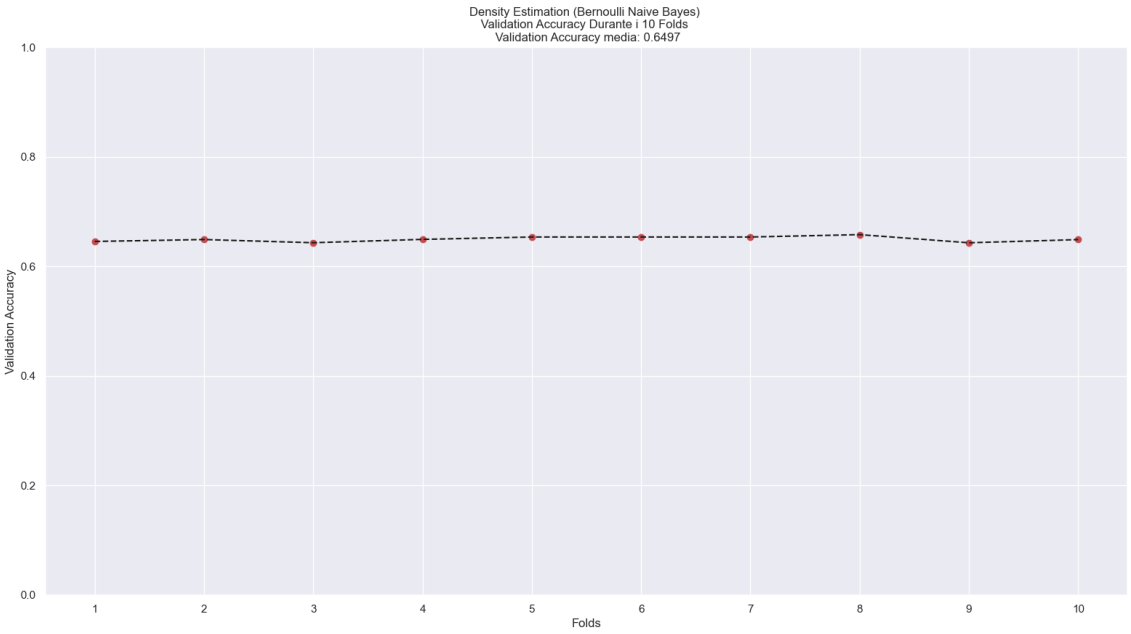
2.2.4: Stima di Densità

Anche in questo caso l'analisi condotta per questa classe è stata svolta mettendo a confronto ***K-NN***, per la famiglia delle tecniche non parametriche, e ***Naïve Bayes***, per quelle parametriche. Questa volta i risultati sono stati sostanzialmente opposti rispetto al dataset di immagini, con il secondo modello che ha raggiunto prestazioni non elevatissime, ma comunque migliori rispetto al primo.

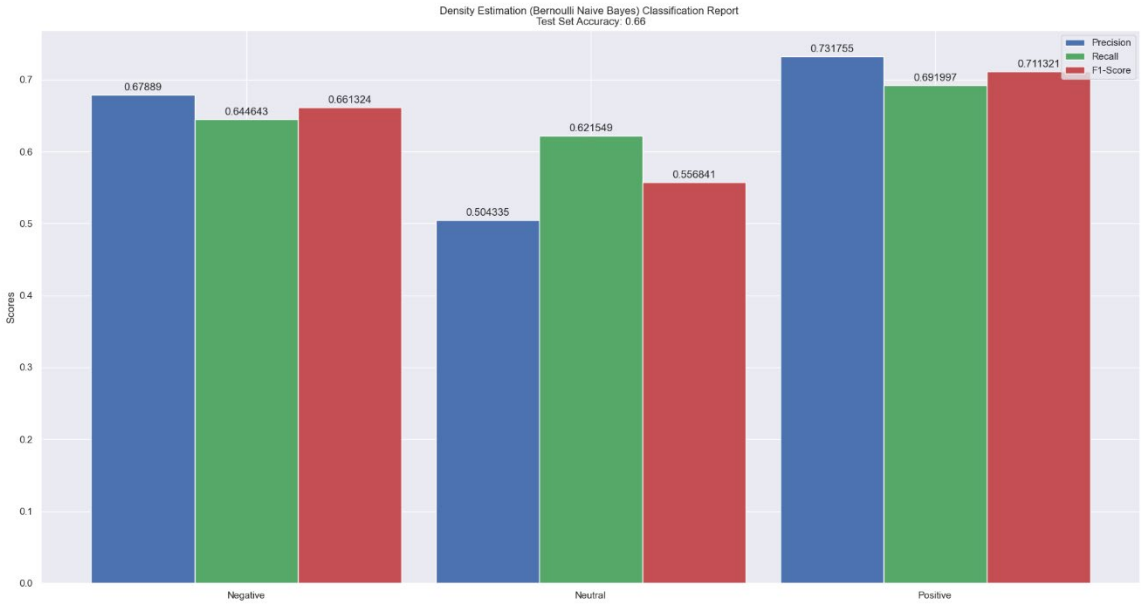
Per quanto riguarda ***K-NN***, l'operazione di *parameter tuning* ha permesso di selezionare il miglior valore di K pari a 5, mentre il peso da assegnare agli esempi vicini per il processo di classificazione è stato scelto in maniera inversamente proporzionale alla distanza (l_2). L'accuratezza come detto è molto bassa, la tecnica *GridSearchCV* ha restituito per il miglior estimator un valore pari a 0.39, in linea con il risultato di 0.40 ottenuto mediante *10-fold cross-validation*, è stato perciò scartato tale classificatore nella valutazione finale.



L'utilizzo della tecnica parametrica *Naïve Bayes*, ha permesso il raggiungimento di buone prestazioni circa in linea con i precedenti modelli analizzati. Bisogna però specificare come sia stata impiegata come funzione di distribuzione non la *Gaussiana*, con la quali si andava a perdere di molto in performance ottenendo lo stesso risultato di *K-NN*, ma bensì è stata impiegata la distribuzione di *Bernoulli* multivariata per approssimare la distribuzione delle varie classi. Per quanto riguarda le operazioni di *parameter tuning*, è stato selezionato il miglior valore di *'alpha'* pari a 0.0001 (rappresenta uno *smoothing* applicato per evitare il caso di ottenere probabilità di appartenenza di un esempio ad una data classe pari a 0). Più in generale è stato possibile notare come, andando a diminuirne il valore, veniva sempre scelto il coefficiente *'alpha'* inferiore, nonostante ciò, le prestazioni variavano di pochi millesimi al costo di dover effettuare ripetutamente gli onerosi step di *GridSearchCV*, per questo motivo è stato selezionato tale parametro *'alpha'*. Inoltre, la scelta migliore è risultata essere andare a calcolare le probabilità *apriori* delle varie classi, piuttosto che impiegare un valore uniforme. I processi di selezione dei migliori parametri e di valutazione mediante *10-fold cross-validation* hanno entrambi restituito un valore di accuratezza media pari a 0.65, con un andamento di tale misura quasi identico per ognuno dei *fold* sviluppati.

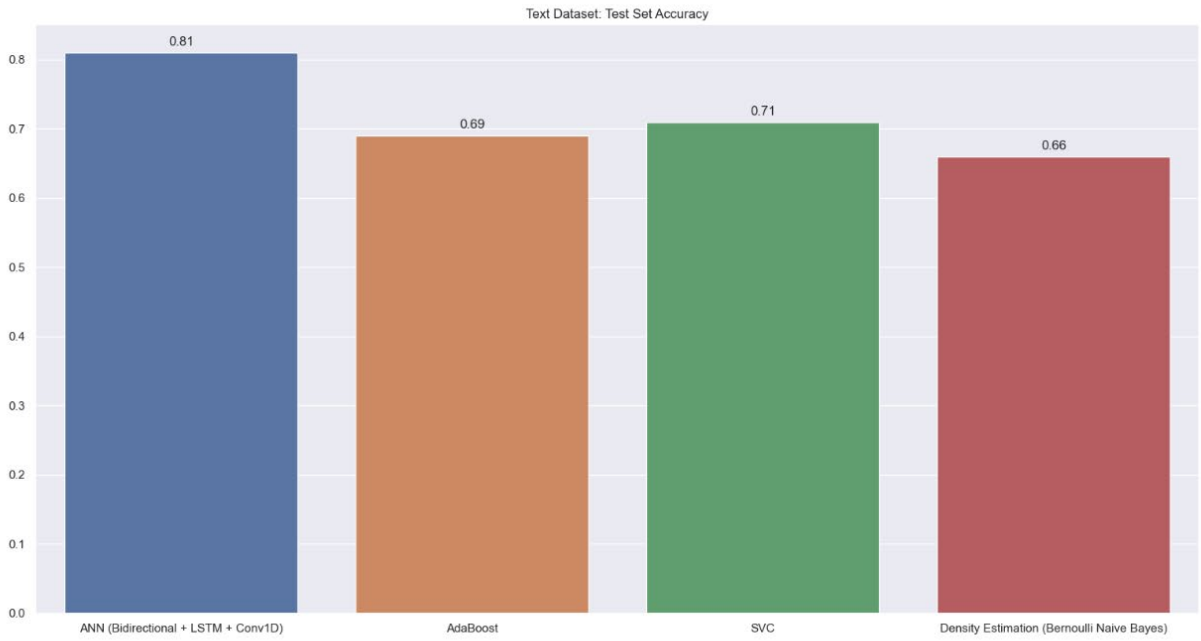


La fase finale, come detto precedentemente, ha richiesto la classificazione ed il calcolo delle varie metriche sul test set mediante l'utilizzo del *classificatore Bayesiano* ottenuto. In questo caso si può osservare come, nonostante il valore 0.66 di accuratezza, tale modello performi leggermente meglio rispetto *SVC* e *AdaBoost* per quanto riguarda la classificazione dell'etichetta *'Neutral'*, ottenendo complessivamente un numero inferiore di falsi negativi.



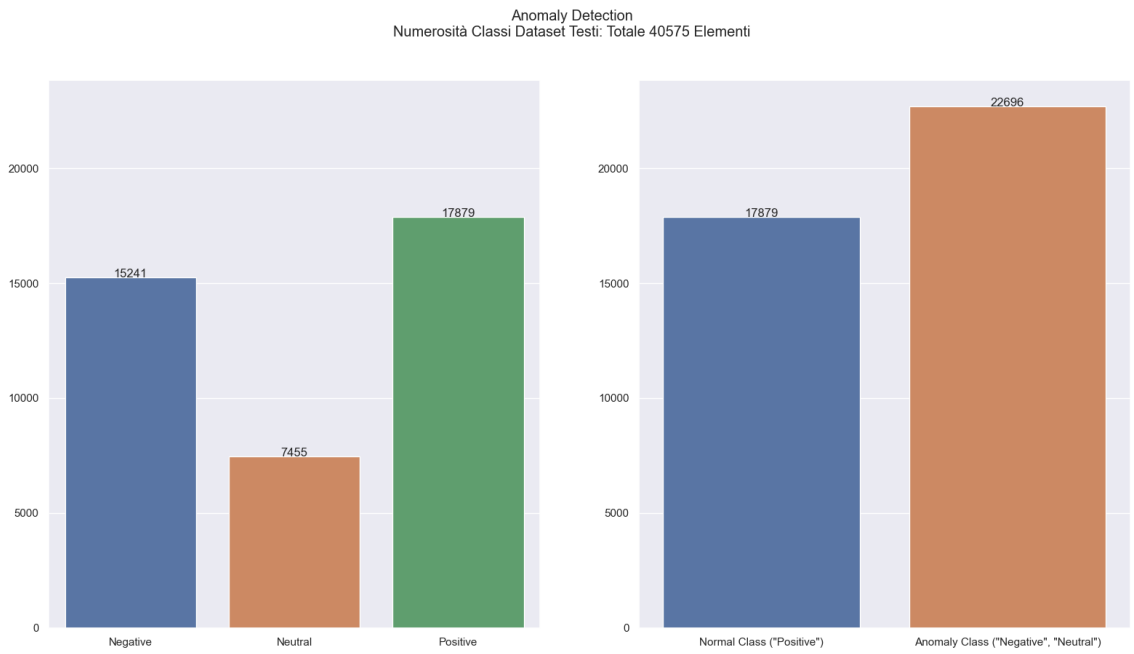
A chiusura del capitolo si riporta il grafico in cui si vanno a mettere a confronto le prestazioni sul *test set* dei migliori modelli sviluppati, si può notare come i risultati più alti si raggiungano in corrispondenza dell'architettura di rete neurale basata su *LSTM*. In generale non sono stati ottenuti valori elevatissimi, segno che i modelli sul dataset in questione pecchino dal punto

di vista della capacità di generalizzazione. Grazie all'analisi dei dati svolta durante la fase di pulizia e *preprocessing*, si può supporre che la loro stessa struttura potrebbe causare difficoltà nella classificazione, ogni tweet è infatti costituito da un numero limitato e molto basso di parole, sostanzialmente della stessa lunghezza di una sola frase, questo appunto potrebbe rendere difficile il processo di apprendimento dei modelli. Con testi di dimensione maggiore probabilmente sarebbe stato possibile ottenere risultati migliori.



2.3: Task di Semi-Supervised Anomaly Detection

Prima di procedere con lo sviluppo dei modelli in grado di risolvere efficacemente tale problema, è stato raffinato il processo di *preprocessing* al fine di ottenere il miglior risultato possibile. Come prima cosa si deve specificare che, essendo la classe dei tweet *'Positive'* la più numerosa, tali elementi sono stati considerati come dati normali, mentre tutti i restanti sono invece considerati anomalie. Come si può osservare nel grafico sottostante, tale modifica ha prodotto un dataset leggermente sbilanciato, con gli elementi che verranno impiegati per l'addestramento *semi-supervisionato* che risultano essere in numero inferiore rispetto ai dati della classe anomala.



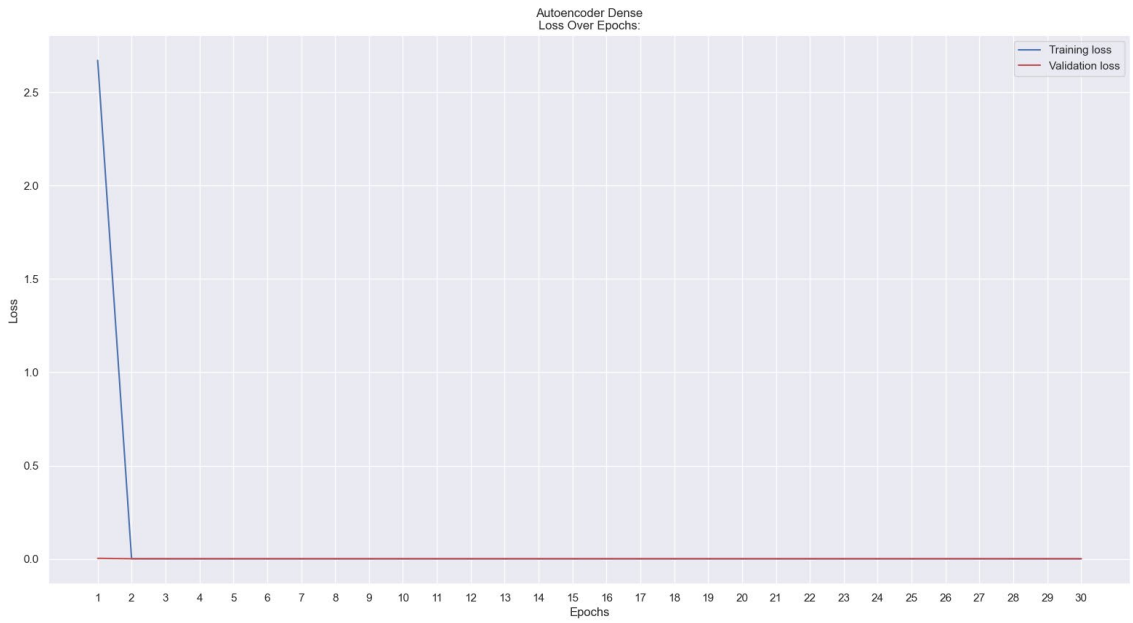
Inoltre, scelta progettuale è stata quella di andare ad eliminare i record composti da un numero di token minore o uguale a 2, non è stata invece considerata questa operazione per il precedente *task* perché anche il numero di parole per tweet poteva essere una caratteristica discriminativa per la classe d'appartenenza dei dati. In questo caso però, dovendo andare sostanzialmente a ricostruire l'input, avere molti elementi dei record con la maggior parte dei valori pari a 0, avrebbe potuto causare una perdita in performance.

L'analisi condotta ha avuto lo scopo di mettere a confronto due diverse architetture ***autoencoder***, addestrate sugli esempi della classe normale, andando ad impiegare le due tecniche di vettorizzazione discusse nella precedente sezione.

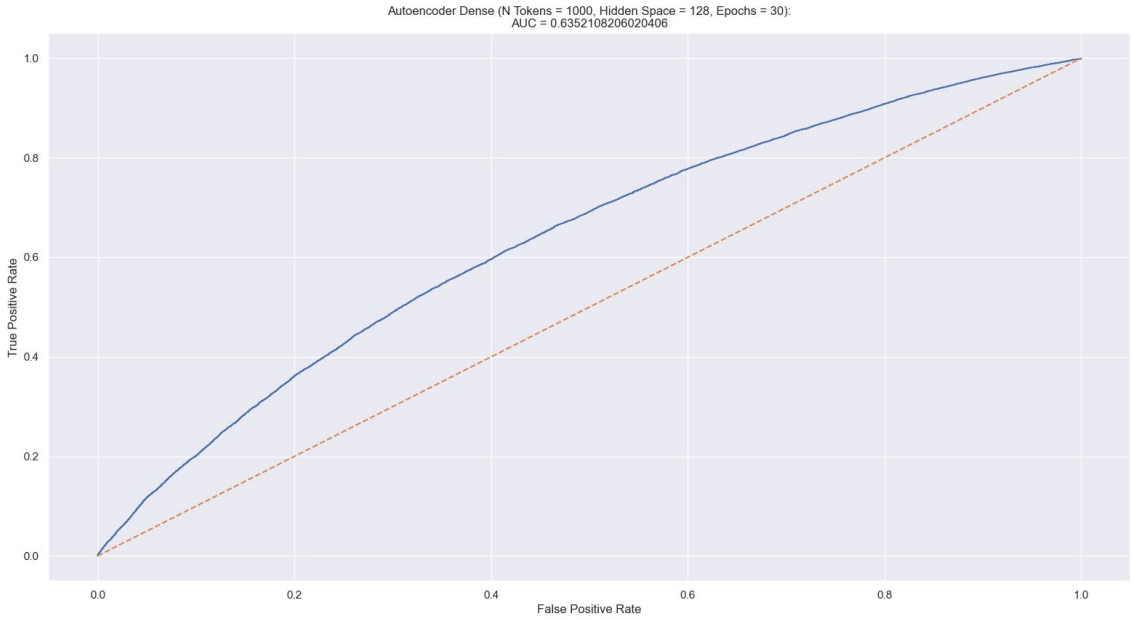
Il primo modello analizzato si basa sull'impiego di un *autoencoder denso* applicato su dati vettorizzati tramite la tecnica *'tfidf'*, è stata ridotta opportunamente la dimensionalità del dataset andando a considerare solo le top 1000 parole per la costruzione della matrice, inoltre tale architettura andava a mappare l'input in uno spazio latente di dimensione 128. Per ottenere tale riduzione sono stati impiegati 4 livelli densi, partendo da un numero di *features* pari a 1000 per ottenere 512, 256 fino a 128 *features* nello spazio latente. Operazioni inverse sono state invece applicate nella porzione *decoder,* che ha permesso di ricostruire la dimensione originale dell'input. Scelta progettuale è stata l'utilizzo della *sigmoide* come funzione di attivazione del *layer* di output, mentre è stato impiegato *Adam* come ottimizzatore e '*mean squared error'* come funzione di *loss*.

```
inp = Input(shape=(max_tokens,))
x = Dense(1000, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(inp)
x = Dropout(0.3)(x)
x = Dense(512, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(x)
x = Dropout(0.3)(x)
x = Dense(hidden_space, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(x)
encoded = Dropout(0.3)(x)
# DECODER:
x = Dense(hidden_space, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(encoded)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(x)
x = Dropout(0.3)(x)
x = Dense(512, activation='relu', kernel_initializer=kernel_initializer, kernel_regularizer=kernel_regularizer)(x)
x = Dropout(0.3)(x)
decoded = Dense(max_tokens, activation='sigmoid')(x)
```

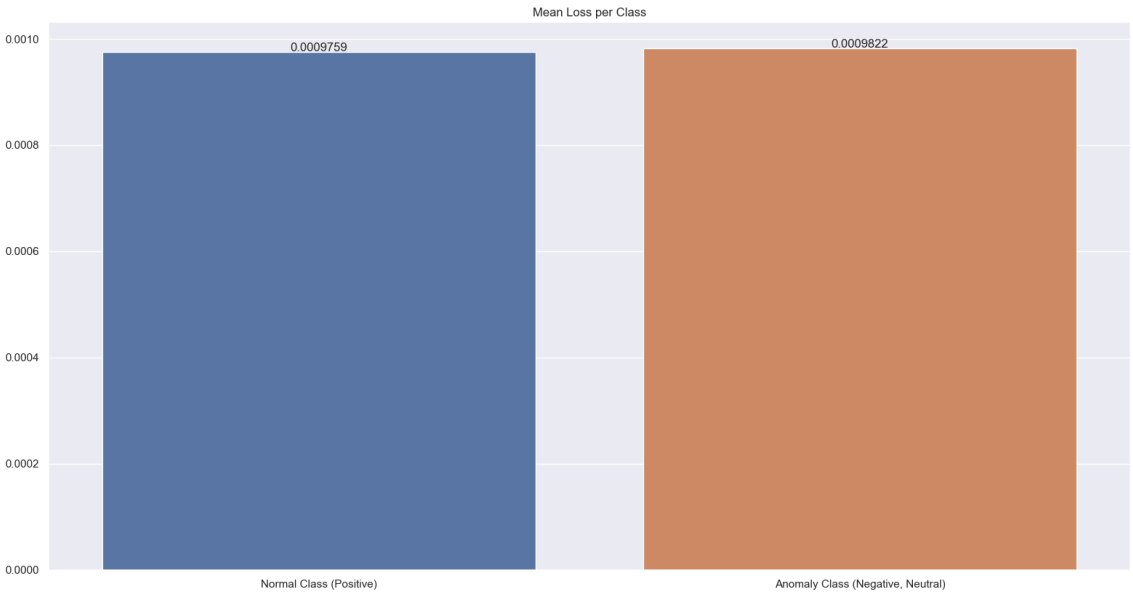
Il modello in questione è stato addestrato per un totale di 30 epoche, andando ad ottenere un basso valore di *loss*. Come si può vedere nel grafico sottostante, la diminuzione maggiore si ha una volta terminata la prima epoca, passando da un valore superiore a 2 a poco più di 0.



Nonostante l'addestramento abbia prodotto buoni risultati, la valutazione mediante curva *ROC* ha restituito un valore di *AUC* di 0.63.



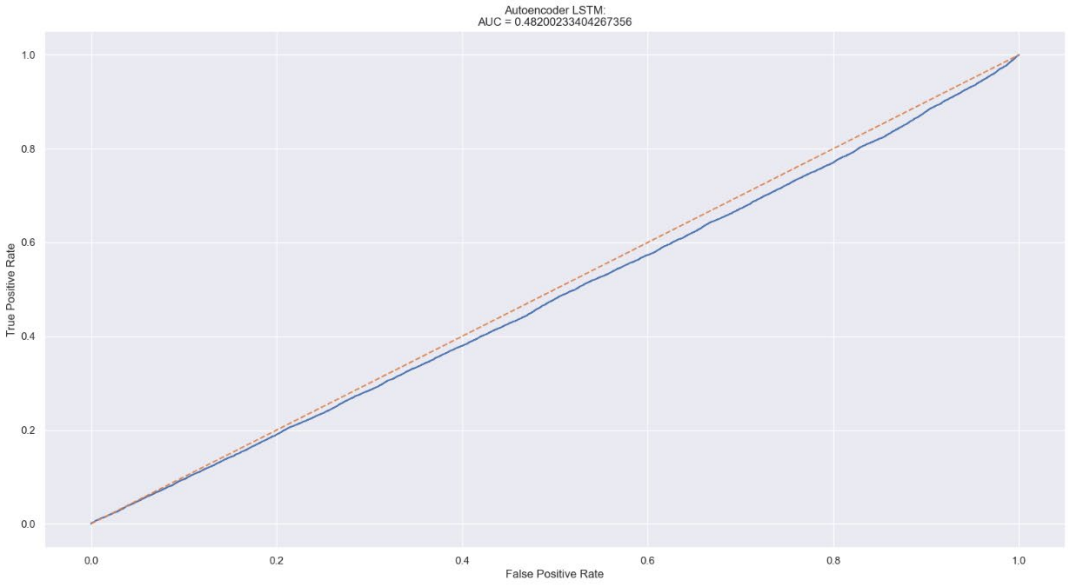
Analizzando l'*outlierness score* medio per label si può notare come, nonostante la rete sia stata addestrata solamente mediante l'utilizzo della classe normale, non vi è una discrepanza marcata tra i valori restituiti, si può quindi andare a giustificare il risultato ottenuto nella fase di valutazione. Essendo la struttura dei dati molto simili, nonostante appartengano a classi differenti (poiché i termini provengono dallo stesso vocabolario), la rete anche se non ha mai osservato gli esempi della classe anomala, riesce comunque a ricostruirli commettendo un errore abbastanza basso. Per tale motivo, non essendo elevata la differenza tra gli *outlierness score*, non si riesce a trovare un buon valore di *threshold* che permetta di discriminare correttamente gli esempi.



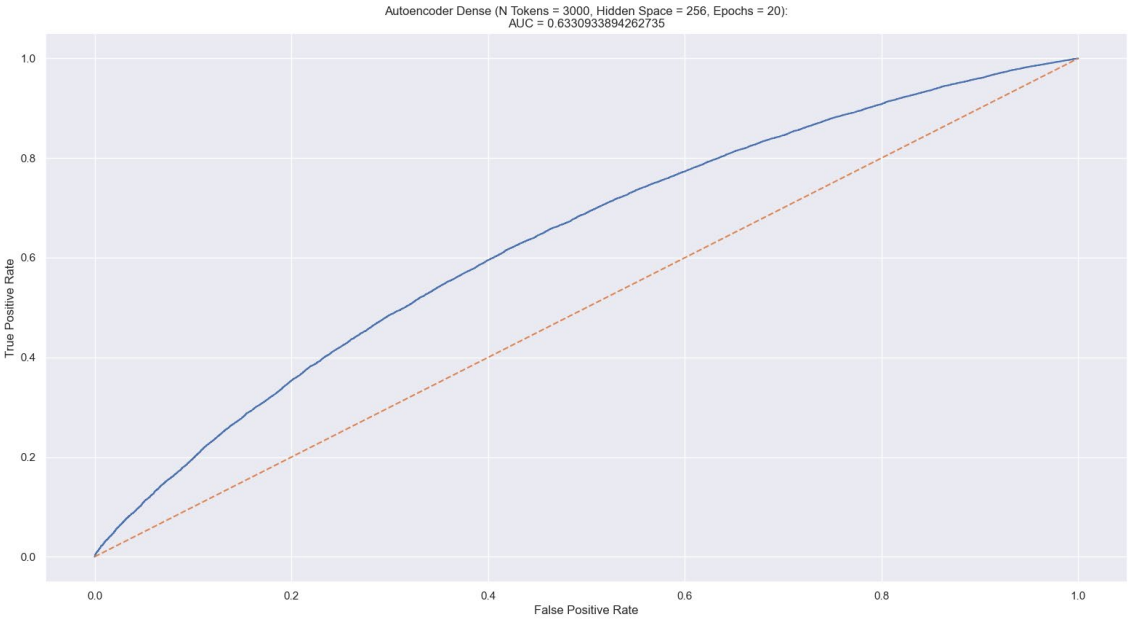
```
model.add(Bidirectional(LSTM(256, dropout=0.25, return_sequences=True)))
model.add(Bidirectional(LSTM(256, dropout=0.25, return_sequences=True)))
model.add(BatchNormalization())
model.add(SpatialDropout1D(0.3))
model.add(Conv1D(256, 3, padding='same', activation='relu'))
model.add(Conv1D(128, 3, padding='same', activation='relu'))
model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(SpatialDropout1D(0.3))
# DECODER
model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(Conv1D(128, 3, padding='same', activation='relu'))
model.add(Conv1D(256, 3, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(GlobalMaxPool1D())
model.add(Dropout(.3))
model.add(Dense(256, activation='relu'))
model.add(Dropout(.3))
model.add(Dense(sequence_len, activation='sigmoid'))
```

La seconda architettura sviluppata impiega sempre un *autoencoder*, ma questa volta la rete si basa sull'utilizzo di layer *LSTM bidirezionali*; è stata quindi impiegata la trasformazione in lista di indici dei vari record del dataset, utilizzando un numero di parole pari a 30000, ed un primo *layer* iniziale di *Embedding*. L'architettura ripresa per la porzione *encoder* è simile a quella impiegata per il problema di classificazione, troviamo 2 livelli *lstm* in cui sono presenti 256 neuroni, succeduti da 3 livelli di convoluzione monodimensionale. Scelta differente è stata invece adottata sulla rete *decoder*, in cui non vengono impiegati i *layer lstm* ma unicamente convoluzionali e densi, per riportare il tensore in input alla dimensione iniziale pari a *'sequence_len'* (per cui è stato utilizzato il valore 50 come nel *task* precedente).

Nonostante l'utilizzo di questa tipologia di modello ha permesso di raggiungere buoni risultati per la classificazione degli esempi, lo stesso non lo si può dire per la loro ricostruzione. La valutazione mediante curva *ROC* e valore di *AUC* ha infatti sottolineato come l'impiego di questa tipologia di rete ricorrente sia sconsigliabile per la risoluzione *semi-supervisionata* del *task*, è stato infatti raggiunto un valore pari a 0.48 di *AUC*, nettamente inferiore a quello ottenuto mediante architettura *densa*.



Ultimo passaggio del lavoro svolto è stato l'analizzare come vadano a variare i risultati se si vanno ad effettuare modifiche sull'autoencoder denso sviluppato, complessivamente sono stati impiegati un minore numero di livelli, andando ad aumentare il numero di token considerati in 'tfidf' (3000) e la dimensione dello spazio latente (portata a 256), mentre è stato diminuito il numero di epoche di addestramento portandolo a 20.



Si può osservare come tali modifiche non siano andate ad apportare dei cambiamenti così marcati, anzi risulta preferibile la prima versione dell'*autoencoder* sviluppato, che arriva ad ottenere risultati leggermente più alti, seppur in termini di millesimi.