Progetto Esame Machine e Deep Learning

Serafino Salatino

March 2021

Sommario

1 Introduzione

Il progetto, proposto in questo pdf, riguarda l'apprendimento automatico di vari algoritmi di machine learning per 2 tipi di dataset che, sia per un task di classificazione, sia per un task di anomaly detection, saranno:

- dataset di immagini
- dataset di file audio

Gli algoritmi usati per lo studio e le sperimentazioni sono:

- SVM
- Adaboost
- Reti neurali
- Stima di densità

Ad ognuno di essi verrà dedicato una specifica sotto-sezione nei capitoli successivi.

2 Preprocessing

2.1 Preprocessing Immagini

Il dataset di input, contenente le immagini, è una cartella in cui si trovano diverse immagini in formato jpg (5192 per l'esattezza). Gli algoritmi di machine learning lavorano con array quindi la prima fase consiste nel convertire queste immagini in matrici di interi (pixel). Per facilitare questo preprocessing si utilizzeranno i package OpenCV e Numpy che permettono, tramite alcune funzioni, di leggere l'immagine e scriverle sotto forma di *ndarray* in un file binario. Prima di scrivere, l'immagine viene convertita in una scala di grigi per poter lavorare con un unico array per immagine. A questo metodo viene aggiunta un'altra procedura che legge il file binario e, per ogni immagine, inserisce in un array denominato **X** la matrice che identifica l'immagine e in un array **Y** la classe a cui appartiene, ottenuta facendo uno split della stringa che identifica il nome del file.

2.2 Preprocessing Audio

Per l'analisi dei file audio in formato *wav* è stato utilizzato il package **librosa**. I file si presentano in 4 cartelle ben distinte che rappresentano la classe a cui quel file appartiene: Happy, Neutral, Sad e Surprised; la ricostruzione delle etichette è quindi banale. La creazione del dataset prevede, invece, delle scelte riguardo le feature che esso dovrà contenere. Prima di vedere le varie feature è necessaria un'introduzione al concetto di audio e alla sua digitalizzazione. Un suono non è altro che un'onda che si propaga per mezzo dell'aria. Quindi esso può essere espresso su un calcolatore come un'onda ma, più nello specifico, dai valori assunti da quell'onda corrispondenti alla sua ampiezza in diversi punti. Infatti l'onda viene discretizzata in una serie di valori reali e viene definita **frequenza di campionamento** il numero di campioni presi in un intervallo di tempo. Più la frequenza è alta, più informazioni otteniamo dall'onda. I parametri che definiscono un'onda sono principalmente tre: ampiezza, lunghezza d'onda e fase. Dopo questa breve introduzione verranno elencate le feature scelte per la creazione del dataset.

- Ampiezza massima dell'onda
- Media dei valori di RMS per ogni frame
- Zero Crossing (numero di volte in cui l'onda passa per lo zero)

- Zero Crossing rate (frequenza di zero crossing)
- Media dei valori dei centroidi spettrali per ogni frame (feature collegata alla chiarezza di un suono)
- Media dei valori di attenuazione spettrale per ogni frame (spectral rolloff)
- Media dei valori di banda per ogni frame
- Media dei valori del cromagramma per ogni frame ottenuto dalla trasformata di fourier del segnale
- Media dei valori Delta per ogni frame

A queste sono state aggiunte altre 40 feature relative ai coefficienti di frequenza Mel, più comunemente chiamati **MFCC**. In generale, si tende a scegliere un numero di coefficienti compreso tra 13 e 40, e sono molto utili per particolari task come ad esempio il riconoscimento di un genere musicale. Il dataset così formato sarà composto da 50 feature.

3 Classificazione Dataset Immagini

3.1 Support Vector Machines

Le Support Vector Machines sono classi di ipotesi che creano un'iperpiano in uno spazio *d*. In questo contesto, lo spazio è pari al numero di pixel delle immagini, cioè 4800 (80x60).Il comportamento dei vari kernel viene influenzato dall'alta dimensionalità e si dovrebbe notare qualche divergenza. Per questo specifico dataset,però, non è molto evidente questa differenza. Questa sottosezione è divisa in 3 parti:

- Accuratezza dell'algoritmo al variare del fattore di regolarizzazione
- Accuratezza dell'algoritmo al variare della dimensionalità
- Accuratezza ottenuta con 10-fold Cross Validation

Nelle prime due, si utilizzerà un test set relativo al 20 % dell'intero dataset, ovviamente pesato sulla cardinalità delle varie classi.

Nella prima sperimentazione, si è voluta analizzare la differenza di performance tra i 3 tipo di kernel (Lineare, Polinomiale di grado 3 e Gaussiano) al variare del fattore di regolarizzazione. Dalla teoria delle Hard SVM si sa che il fattore C è uguale a

$$1/\lambda$$

e regola il tradeoff tra la dimensione del margine e gli errori ottenuti dai punti misclassificati. Come si può notare dalle 2 tabelle sottostanti, si ottengono delle performance altissime che crescono di poco all'aumentare del valore C.

Valore C	Lineare	Polinomiale	Gaussiano
0.1	1.0	1.0	0.974
0.3	1.0	1.0	0.989
0.6	1.0	1.0	0.991
0.8	1.0	1.0	0.993
1	1.0	1.0	0.994

Table 1: Risultati Accuratezza SVM Training set

Valore C	Lineare	Polinomiale	Gaussiano
0.1	0.992	0.994	0.977
0.3	0.989	0.992	0.986
0.6	0.989	0.991	0.991
0.8	0.989	0.991	0.991
1	0.989	0.991	0.991

Table 2: Risultati Accuratezza SVM Test set

Nella seconda parte, si è deciso di visionare le performance al variare del numero di feature, ridotte tramite la tecnica di **dimensionality reduction** denominata PCA. Come si può notare, benchè si veda un calo di accuratezza nel training set dovuto alla perdita di informazione, l'errore resta comunque basso e il tempo di learning si riduce di tanto.Il fattore di regolarizzazione verrà fissato ad 1, che ci dà delle performance migliori come mostrato nelle tabelle precedenti.

Valore PCA	Lineare	Polinomiale	Gaussiano
10	0.976	0.976	0.983
30	0.985	0.988	0.990
80	0.996	0.993	0.994
200	0.999	0.995	0.995
400	1	0.995	0.995

Table 3: Risultati Accuratezza SVM Training set

Valore PCA	Lineare	Polinomiale	Gaussiano
10	0.968	0.975	0.981
30	0.988	0.989	0.988
80	0.991	0.991	0.989
200	0.989	0.991	0.990
400	0.992	0.990	0.991

Table 4: Risultati Accuratezza SVM Test set

Se nelle prime 2 si è utilizzato l'accuracy score come misura di successo che si trova nel package *sklearn.metrics*, in questa sezione verranno visualizzate le performance con la k-fold cross validatio al variare di k. Il valore che si trova in ogni cella della tabella sottostante è ricavato dalla **media** delle accuratezze di ogni fold.

N° Fold	Lineare	Polinomiale	Gaussiano
5	0.991	0.992	0.989
10	0.990	0.992	0.990
15	0.991	0.992	0.990
20	0.991	0.992	0.990

Table 5: Risultati k-fold Cross Validation SVM

Alla luce dei risultati ottenuti, si è scelto di salvare il modello polinomiale di grado 3 con fattore di regolarizzazione pari ad 1.

3.2 Adaboost

In questa sottosezione si sperimenterà l'algoritmo di boosting più diffuso, per la creazione di uno strong learner basato sul weak learner **Decision Tree**. Questo weak learner viene fornito di default dall'algoritmo implementato in scikit-learn. Gli iperparametri di questo algoritmo sono:

- numero di stimatori
- learning rate

Come già visto in precedenza, si vedranno le differenze di accuratezza ottenute tramite uno split del dataset in training set e validation set e tramite l'uso della k-fold cross validation. Si eviterà di implementare la PCA visto che, come mostrato poc'anzi, la riduzione di feature mantiene all'incirca le stesse performance. Sulle righe delle tabelle sottostanti si avranno i valori che assume il learning rate, sulle colonne il numero di stimatori utilizzati.

Learning rate	5 est	10 est	20 est	40 est	60 est	80 est
0.1	0.963	0.970	0.980	0.986	0.990	0.992
0.3	0.968	0.982	0.991	0.995	0.998	0.999
0.6	0.979	0.987	0.997	1	1	1
0.8	0.979	0.990	0.997	1	1	1
1	0.981	0.991	0.998	1	1	1

Table 6: Risultati Accuratezza Adaboost Training set

Learning rate	5 est	10 est	20 est	40 est	60 est	80 est
0.1	0.956	0.963	0.979	0.987	0.987	0.988
0.3	0.964	0.978	0.986	0.986	0.986	0.988
0.6	0.976	0.982	0.986	0.989	0.988	0.988
0.8	0.973	0.979	0.985	0.987	0.989	0.988
1	0.973	0.985	0.988	0.990	0.989	0.991

Table 7: Risultati Accuratezza Adaboost Test set

Nelle tabelle appena mostrate si può notare che, all'aumentare delle ipotesi (stimatori) create, le performance aumentano. Nel training set si raggiunge anche un'accuratezza del 100 % con un numero di stimatori maggiori o uguali a 40. Questo potrebbe essere causato da overfitting, che sappiamo essere regolato dal numero di iterazioni nell'adaboost. In questo caso però, si ottengono ottime performance anche nella valutazione del validation set quindi possiamo scartare l'idea che il modello vada in overfitting. L'accuratezza migliore nel validation set si ha quindi con 80 stimatori ed un learning rate pari ad 1.

Ora si analizzeranno le performance con l'utilizzo della k-fold cross validation per vedere se questi risultati rimarranno invariati.

N° Fold	5 est	10 est	20 est	40 est	60 est	80 est
5	0.973	0.984	0.988	0.989	0.990	0.991
10	0.976	0.984	0.989	0.990	0.990	0.991
15	0.978	0.983	0.988	0.991	0.991	0.991
20	0.978	0.985	0.988	0.991	0.991	0.991

Table 8: Risultati k-fold Cross Validation SVM

Le performance dell'adaboost date dall'accuracy score sono infatti simili a quelle date dalla cross validation. Anche qui, come nelle SVM, si ottengono valori maggiori del 99%. Il modello salvato sarà quello con learning rate pari ad 1 e numero di stimatori pari a 40, dal momento che sul validation set l'errore rimane costante anche aumentandoli.

3.3 Reti Neurali

In questa sezione si analizzerà un approccio basato sulle reti neurali. A differenza degli approcci precedenti, non c'è bisogno di utilizzare la tecnica **OneVsRest** visto che viene implementata semplicemente utilizzando n neuroni di output, dove n è il numero di classi.Questa analisi è stata divisa in due parti, in base alla rete utilizzata:

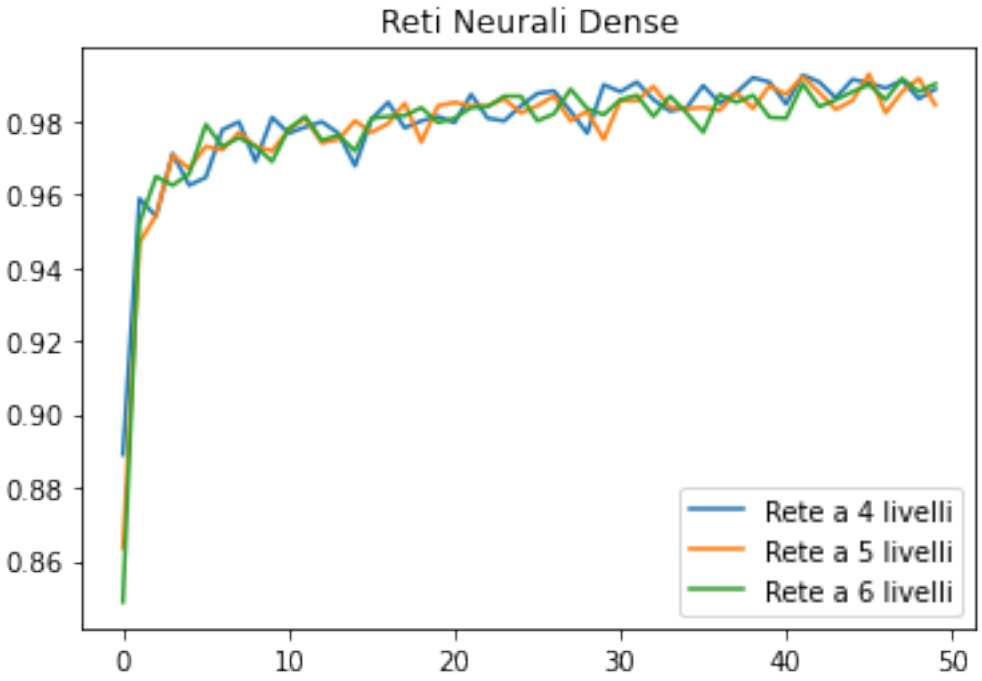
- Approccio con Rete Densa
- Approccio con Rete Convoluzionale

3.3.1 Rete Densa

La struttura di questa rete è molto semplice:

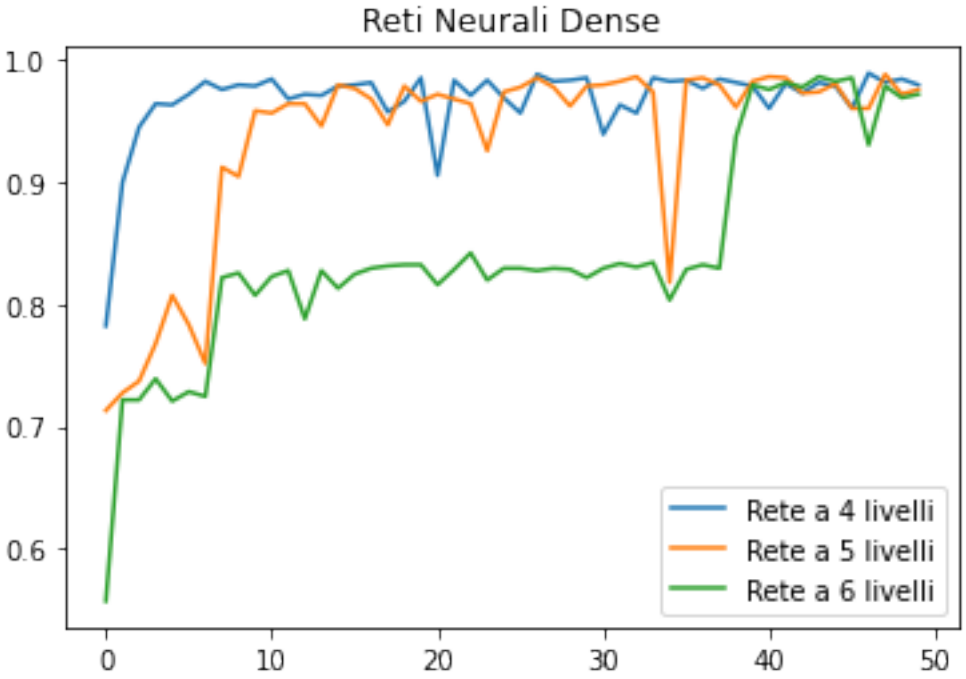
- 1 livello Flatten (che trasforma la matrice 80x60 in un array da passare direttamente alla rete)
- 1 o più livelli Densi
- 1 livello di output di 4 neuroni (uno per ogni classe)

Come mostrato nella figura sottostante, sono state create 3 reti neurali con funzione di attivazione **relu**. Una rete a 4 livelli che ha 2 livelli densi intermedi di dimensione 128 e 64 neuroni; una rete a 5 livelli che ha 3 livelli densi di dimensione 128,64 e 32 neuroni; l'ultima rete ha rispettivamente 128,64,32 e 10 neuroni. Dopo aver creato la struttura delle reti, il dataset è stato diviso in un training set ed in un validation set. La figura mostra l'errore sul validation set con un learning di 50 epoche.



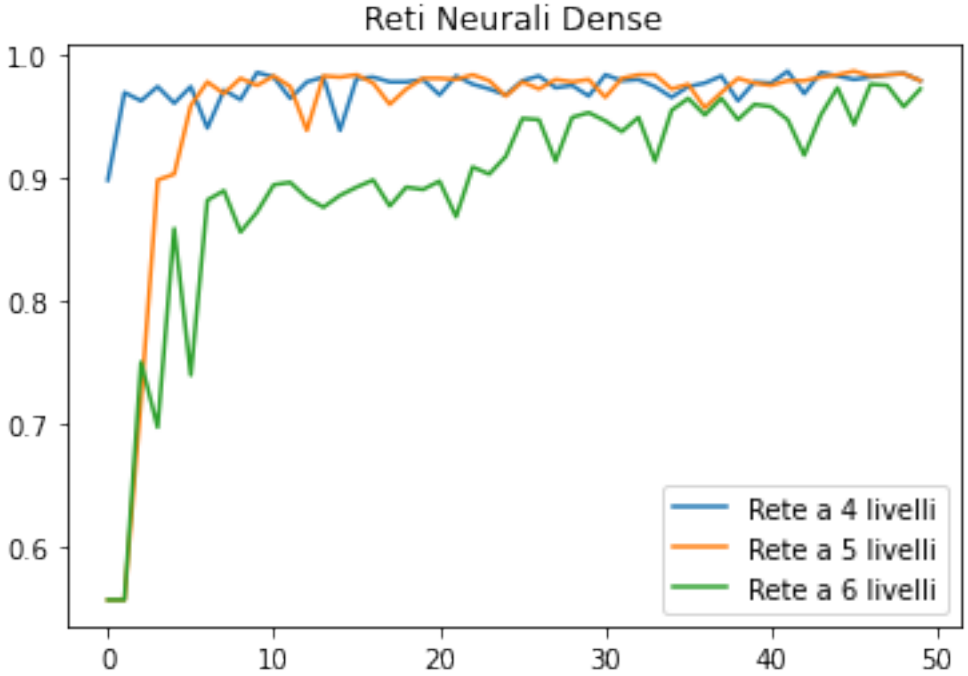
Come si può notare tutte e 3 le reti, nonostante siano di diverse taglie, hanno un'accuratezza molto elevata, come quella riscontrata nelle SVM e nell'algoritmo di boosting.

Interessante è il grafico ottenuto dalle stesse reti ma con funzione di attivazione la **sigmoide** per ogni neurone della reta.



E' facile notare che le reti convergono ad un'accuratezza molto elevata sul validation set, ma a differenza della precedente figura queste reti hanno un errore molto più elevato. Emblematico è il caso della rete a 6 livelli che per le prime 40 epoche ha un'accuratezza massima dell'80% circa. Nei primi due test, non è stata specificata nessuna funzione

di attivazione per il livello di output.Introducendo nel livello di output la funzione di attivazione **softmax** (con tutti gli altri livelli che usano la funzione sigmoide) si ottengono delle reti più stabili come si nota dalla figura sottostante.



Alla luce dei risultati ottenuti, la configurazione migliore per una rete densa è l'uso della funzione relu per ogni neurone e nessuna funzione di attivazione per i nodi di output. Al fine di non rendere lunga la relazione, non vengono riportate altre modifiche alla struttura delle reti come ad esempio:

- Stesse reti con sigmoide come funzione di attivazione per i nodi di output (accuratezza tra il 90% e il 99%)
- Introduzione di un ulteriore livello denso con 256 nodi (accuratezza tra il 92% e il 99% ma molto instabile)
- Uso dell'ottimizzatore **Stochastic gradient descent** (performance simili a quelle dell'ottimizzatore **Adam**)

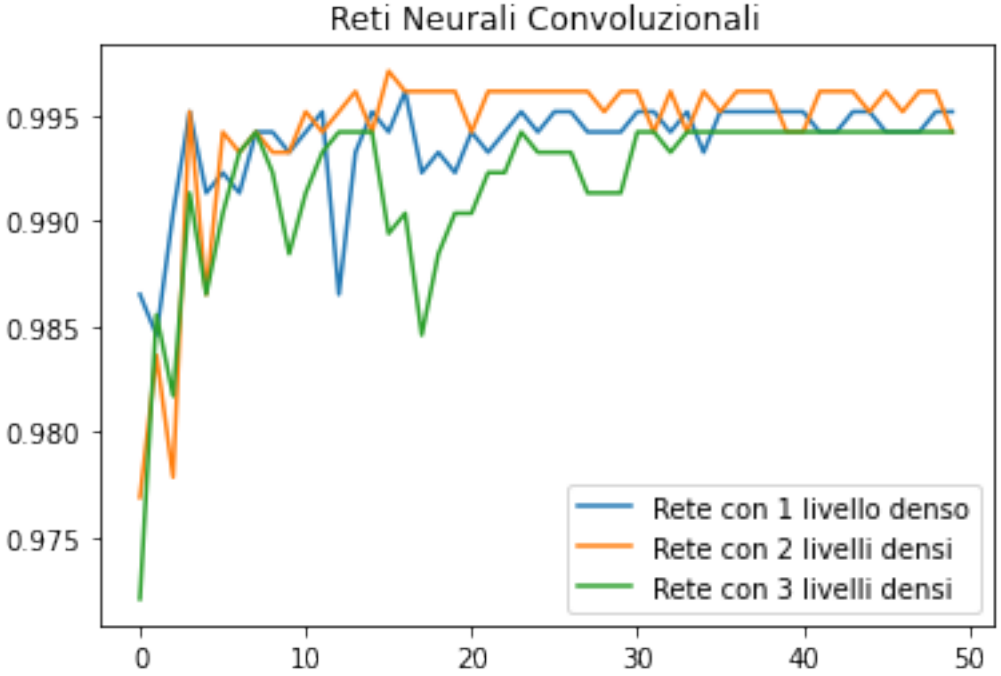
Il modello scelto per l'implementazione è quello a 6 livelli, con ottimizzatore **Adam** e **Relu** come funzione di attivazione per tutti i neuroni. Per questo modello è stata calcolata l'accuratezza tramite la 10-fold cross-validation che è pari a **98.35%**.

3.3.2 Rete Convoluzionale

Per la creazione della rete convoluzione, usiamo i layers **Conv2D** forniti dal package di keras. Come prima analisi viene creata una rete neurale fatta in questo modo:

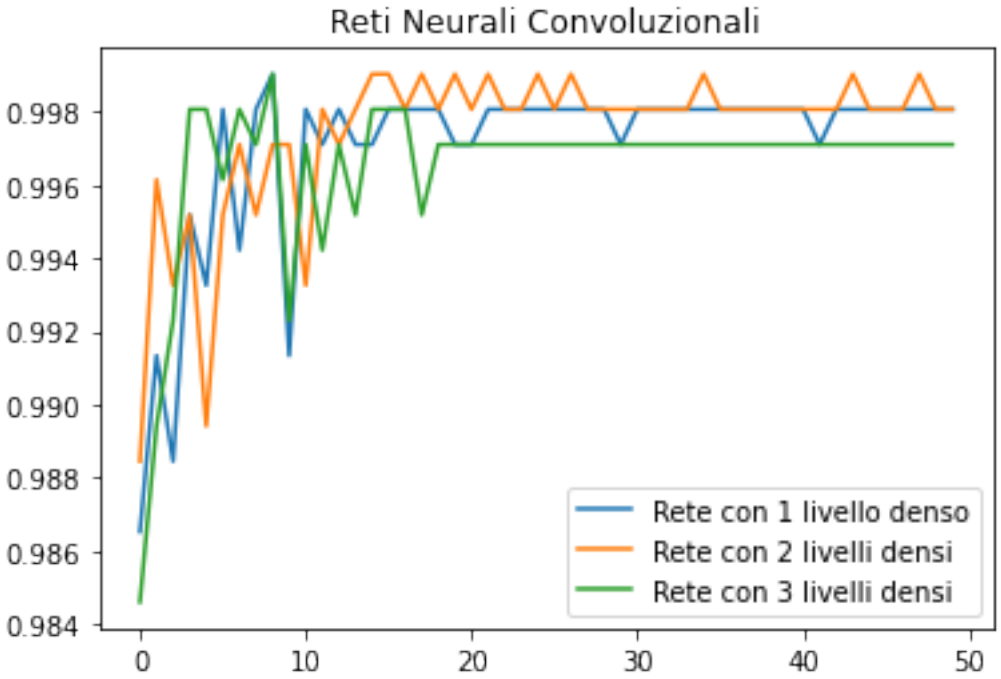
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 58, 64)	640
max_pooling2d (MaxPooling2D)	(None, 39, 29, 64)	0
conv2d_1 (Conv2D)	(None, 37, 27, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 18, 13, 64)	0
conv2d_2 (Conv2D)	(None, 16, 11, 64)	36928

Ad ogni livello di convoluzione, segue un livello di pooling. Nei 3 livelli applichiamo 64 filtri di dimensione 3x3. Ora si confronteranno 3 reti neurali che differiscono dal numero di livelli densi introdotti dopo la parte di convoluzione e pooling.



Si nota facilmente che queste reti hanno un'efficacia quasi ottimale. Le 3 reti create, a regime, raggiungono un'accuratezza sul validation set superiore al 99,2%.

Un'ulteriore modifica fatta è quella di far differire il numero di filtri per livello convoluzionale. Nella figura sottostante verrà mostrata l'accuratezza delle reti a cui vengono applicate meno filtri nel primo livello : la prima rete ne avrà 40, la seconda rete ne avrà 32.



La rete, non solo dovrà gestire meno parametri, ma raggiunge un'accuratezza sul validation set superiore al 99,6%. Il modello implementato sarà la rete neurale convoluzionale con 2 livelli densi che raggiunge un'accuratezza del 99,8%.

3.4 Stima di Densità

In questa sezione ci si è occupati della classificazione tramite stime di densità. Sono stati provati diversi stimatori che non sono stati riportati qui come ad esempio il GMM a causa delle pessime performance che si ottenevano. Sono stati scelti i migliori sia per la famiglia delle funzioni parametriche che non parametriche.

3.4.1 Parametrica

Per la versione parametrica è stato scelto il classificatore Naive Bayes che ci dà buone performance anche se non ottimali come le precedenti classi di ipotesi. Sono stati ottenuti questi risultati dopo la fase di learning tramite **accuracy score**:

- 92% per il training set
- 90% per il test set

I risultati ottenuti dalla k-fold cross validation sono i seguenti:

5 fold	10 fold	15 fold	20 fold
0.919	0.920	0.919	0.919

Table 9: Risultati K-fold cross validation

Al fine di comprendere meglio questa tecnica è stata applicata la PCA per generare 2 componenti e poter visionare meglio dove queste classi si collocano nello spazio.

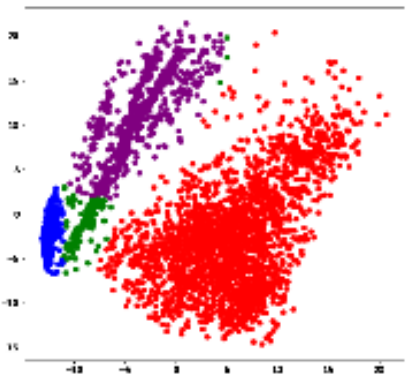


Figure 1: Classi predette del Training Set

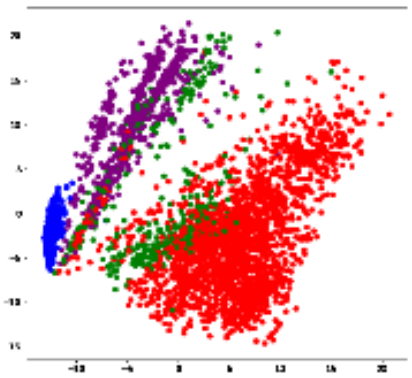


Figure 2: Classi reali del Training Set

- colore rosso = 'shirts'
- colore verde = 'shorts'
- colore blu = 'sunglasses'
- colore viola = 'wallets'

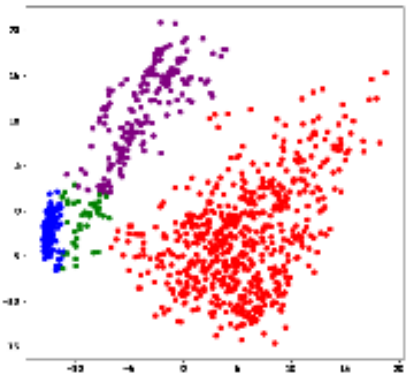


Figure 3: Classi predette del Test Set

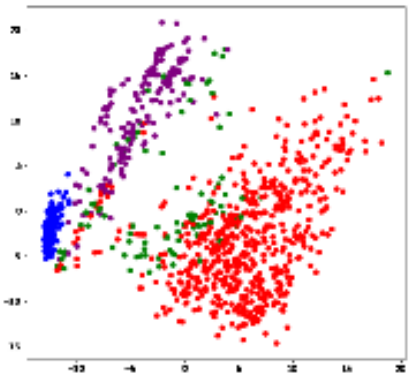


Figure 4: Classi reali del Test Set

Utilizzando la PCA con 2 componenti, l'accuratezza sia per il training set che per il test set risulta essere 87%. Come si può notare, le varie classi occupano uno spazio abbastanza specifico del piano, anche se tra le 4, la classe di color verde (shorts) presenta la maggior parte dei punti misclassificati. Utilizzare la PCA riduce il numero di dimensioni quindi ci si può aspettare che dei punti di una classe finiscano nella "regione" occupata da un'altra.

3.4.2 Non Parametrica

Per la stima di densità non parametrica si è deciso di implementare l'algoritmo di K-nearest neighbor. Sono state fatte diverse analisi variando il numero di vicini K, e il tipo di funzione pesata. La funzione che garantisce performance migliori è la **distance** che assegna il peso come l'inverso della distanza tra i punti. Più 2 punti sono vicini, più sarà il peso assegnato. La funzione **uniform** garantisce anche ottime prestazioni ma leggermente più piccole (1-2% in meno di accuratezza).

I risultati ottenuti dalla k-fold cross validation sono i seguenti:

5 fold	10 fold	15 fold	20 fold
0.990	0.991	0.991	0.992

Table 10: Risultati K-fold cross validation

Per quanto riguarda il numero di vicini, le prove hanno mostrato che non serve selezionare troppi punti, quindi è stato scelto un valore pari a 4.

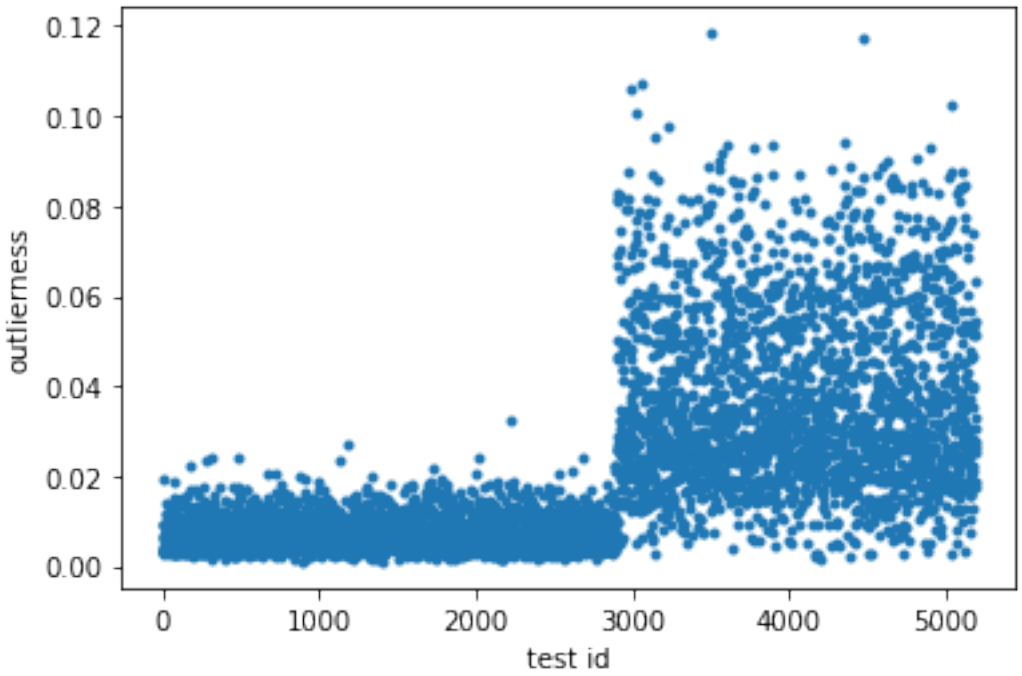
4 Anomaly Detection Dataset Immagini

Il task di anomaly detection ha l'obiettivo di creare un modello che venga addestrato sulla classe più numerosa permettendo ad esso di riconoscere più facilmente gli oggetti di quella classe (con una loss bassa) e di riconoscere con più difficolta gli oggetti delle altre classi. Per questo dataset, la classe prevalente era quella delle magliette (shirts). Per la risoluzione di questo task è stato creato un AutoEncoder fatto in questo modo:

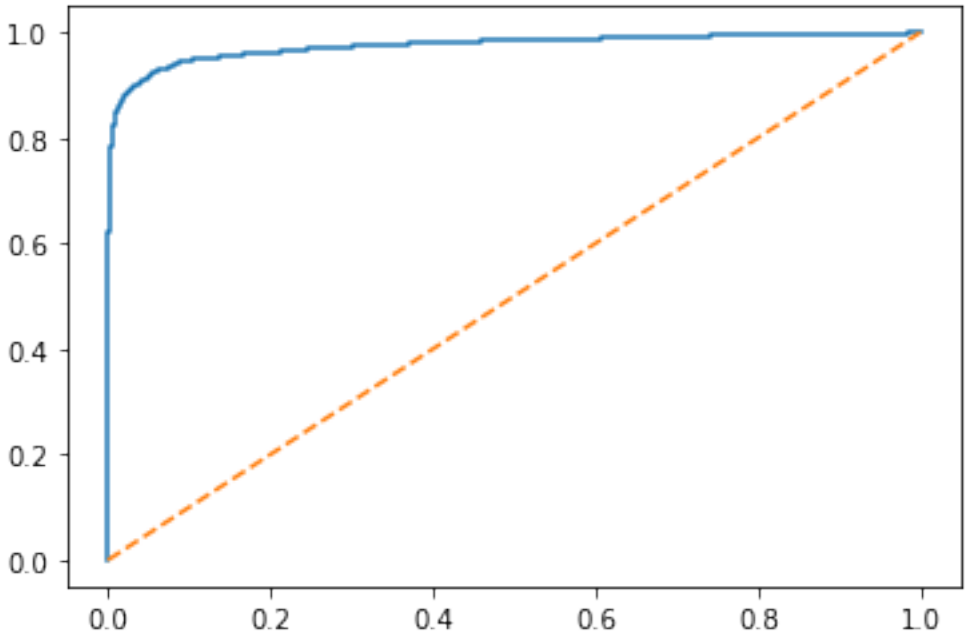
- un livello di input fatto da 4800 neuroni
- un encoder fatto da 3 livelli densi di taglia rispettivamente 512,256 e 128 neuroni.
- livello di spazio latente fatto da 64 neuroni
- decoder fatto da 3 livelli densi di taglia rispettivamente 128,256 e 512
- un livello di output fatto da 4800 neuroni

. Tutti i livelli utilizzano come funzione di attivazione la **relu** ma il livello di output viene implementato con la sigmoide (scelta basata sulle sperimentazioni). Prima di stampare i risultati è interessante vedere come l'autoencoder abbia ricostruito alcune delle immagini in input.

Anche se la qualità non è perfetta, la ricostruzione è quasi fedele. Nelle immagini successive verrà mostrata l'outlierness delle immagini, basate sulle loss **MSE** e la curva di ROC.



Come si nota dal grafico, le magliette (che sono i primi 2893 punti) vengono ricostruite con una loss inferiore a 0.02 mentre tutti gli oggetti delle altre classi invece con una loss superiore a 0.02.



La curva di ROC è molto significativa in questo caso, infatti ha un andamento simile a quello ideale. Il valore AUC restituito è pari a **0.974**, che indica l'area sotto la curva.

5 Classificazione Dataset Audio

Prima di procedere con i vari algoritmi utilizzati è necessario specificare che le prestazioni ottenute nel task di classificazione per le immagini non saranno replicate nelle sezioni

successive. In realtà, come visto e letto in vari paper, è difficile ottenere prestazioni sopra l'85% per questo tipo di problema.

5.1 Support Vector Machines

Come nella sezione riguardante la classificazione delle immagini, anche qui dovrà essere generato un iperpiano che cercherà di separare i diversi oggetti delle varie classi. La tecnica, utilizzata anche qui, è la **OneRestVSAll** che permette di eseguire una classificazione multiclasse. I risultati che verranno mostrati riguarderanno le performance ottenute sia con l'accuracy score, dividendo il dataset in training set e validation set, sia con la k-fold cross validation.

Valore C	Lineare	Polinomiale	Gaussiano
0.1	0.66	0.77	0.70
0.3	0.67	0.80	0.73
0.6	0.68	0.83	0.77
0.8	0.68	0.83	0.76
1	0.68	0.84	0.77

Table 11: Risultati Accuratezza SVM Training set

Valore C	Lineare	Polinomiale	Gaussiano
0.1	0.66	0.75	0.70
0.3	0.68	0.78	0.73
0.6	0.69	0.80	0.74
0.8	0.70	0.80	0.75
1	0.70	0.81	0.75

Table 12: Risultati Accuratezza SVM Test set

N° Fold	Lineare	Polinomiale	Gaussiano
5	0.684	0.803	0.758
10	0.685	0.807	0.762
15	0.686	0.808	0.761
20	0.686	0.808	0.762

Table 13: Risultati SVM K-Fold Cross Validation

Come si evince dai risultati si ottiene un'accuratezza dell'80% in corrispondenza di un kernel polinomiale di grado 3. Nelle varie tabelle si può notare come il kernel lineare abbia più difficoltà nel classificare gli oggetti rispetto a quello gaussiano e polinomiale. Oltre a questi risultati sono stati testati anche diversi kernel polinomiali di grado diverso da 3:

- con grado minore di 3 si ottengono performance basse
- con grado maggiore di 3 si va in overfitting

5.2 Adaboost

In questa sezione l'algoritmo di boosting ottiene delle performance peggiori rispetto alle SVM con kernel polinomiale. Esso si basa sempre sul weak learner **Decision Tree** e le tabelle sottostanti mostrano le performance ottenute nel training set, nel validation

set, e nell'utilizzo della k-fold cross validation al variare degli iperparametri che sono il learning rate e il numero di stimatori.

Learning rate	5 est	10 est	20 est	40 est	60 est	80 est
0.1	0.584	0.595	0.624	0.673	0.703	0.720
0.3	0.620	0.660	0.702	0.735	0.748	0.759
0.6	0.633	0.687	0.724	0.748	0.770	0.787
0.8	0.643	0.698	0.727	0.764	0.786	0.801
1	0.651	0.691	0.729	0.767	0.794	0.809

Table 14: Risultati Accuratezza Adaboost Training set

Learning rate	5 est	10 est	20 est	40 est	60 est	80 est
0.1	0.576	0.591	0.615	0.660	0.691	0.699
0.3	0.601	0.637	0.679	0.713	0.716	0.729
0.6	0.613	0.660	0.692	0.714	0.726	0.738
0.8	0.628	0.677	0.694	0.715	0.727	0.733
1	0.633	0.676	0.701	0.718	0.731	0.744

Table 15: Risultati Accuratezza Adaboost Test set

N° Fold	5 est	10 est	20 est	40 est	60 est	80 est
5	0.637	0.684	0.704	0.724	0.737	0.745
10	0.633	0.677	0.702	0.722	0.737	0.744
15	0.635	0.681	0.705	0.729	0.742	0.750
20	0.634	0.679	0.707	0.729	0.743	0.748

Table 16: Risultati k-fold Cross Validation Adaboost

L'andamento delle performance segue quello del task di classificazione delle immagini: all'aumentare del learning rate e del numero di stimatori le prestazioni crescono, anche se di poco. Se si confrontano le prime due tabelle, si vede che con un numero di stimatori maggiore di 40, l'algoritmo inizia ad andare in overfitting (la differenza di accuratezza sul training set e sul test set aumenta sempre di più).

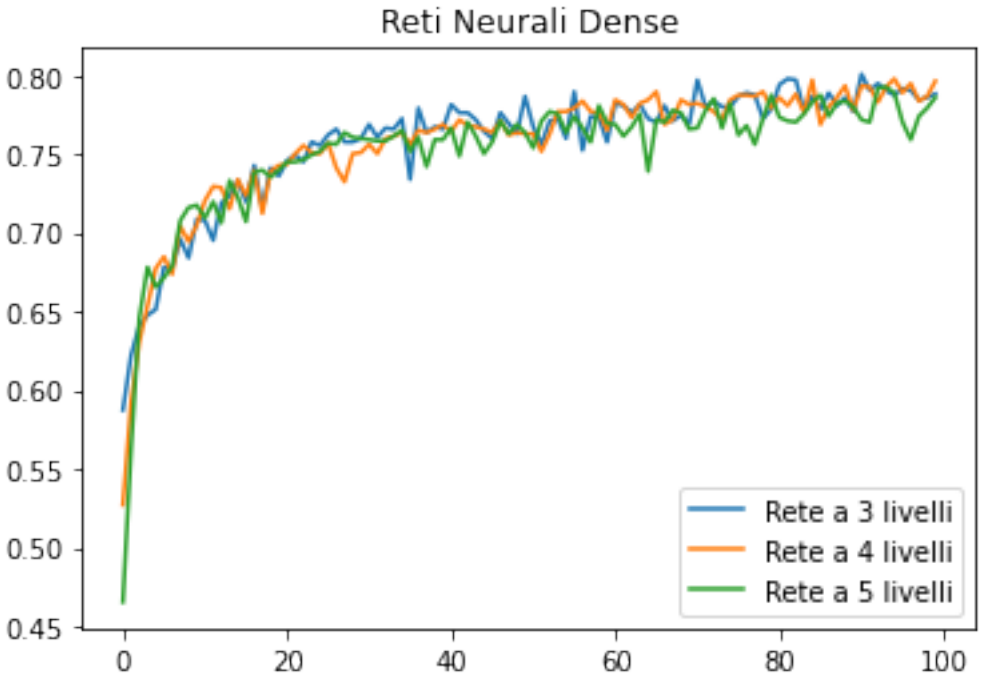
5.3 Reti Neurali

5.3.1 Reti Dense

Le varie prove, svolte per l'individuazione di una buona rete neurale, hanno portato alla costruzione e all'implementazione di una rete a 4 livelli fatta in questo modo:

- un livello denso con 64 neuroni
- un livello denso con 32 neuroni
- un livello denso con 16 neuroni
- un livello di output con 4 neuroni

Ogni livello utilizza la funzione di attivazione *relu* e l'ottimizzatore scelto è *Adam*. Come nel task delle immagini, la funzione di loss è la *cross entropy*. Il grafico riportato mostra l'andamento dell'accuratezza delle reti su un validation set al variare dei livelli utilizzati.



Come si può notare la rete a 4 livelli sembra la più stabile, mentre le altre due hanno un andamento più irregolare. Sono state eseguite altre prove riportate qui in elenco:

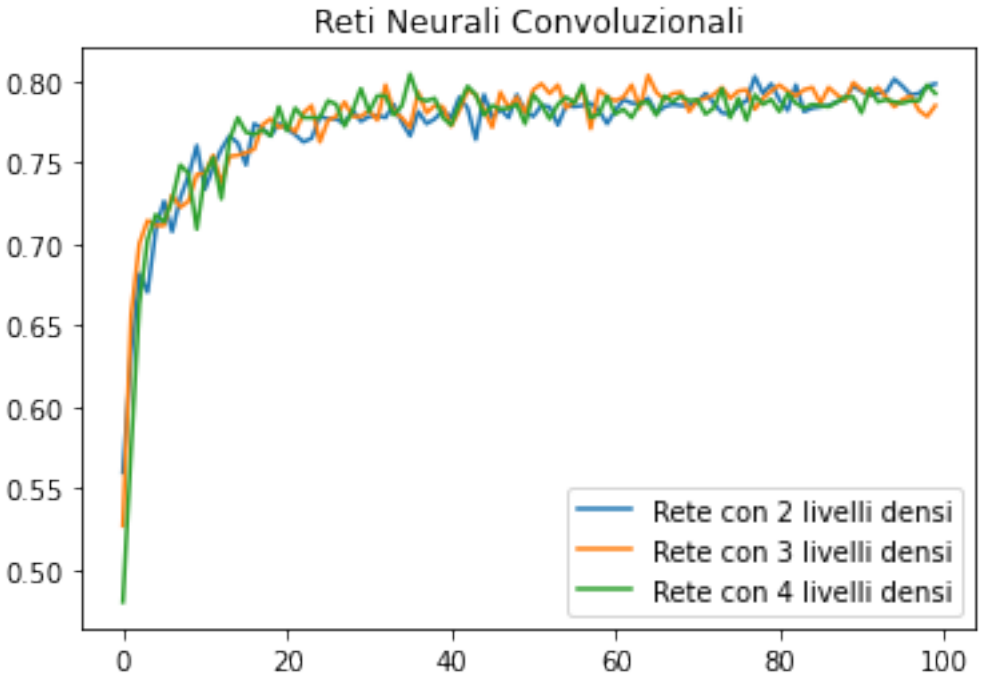
- sostituzione dell'ottimizzatore Adam con SGD (l'accuratezza scende di 5-6 punti percentuali)
- l'utilizzo di un livello iniziale con 128 o 256 neuroni (errore simile a quello attuale)
- sostituzione della funzione di attivazione relu con la sigmoide (l'accuratezza scende al 75%)

La rete a 4 livelli ha un'accuratezza, calcolata con la 10-fold cross validation, pari a **79,63%**

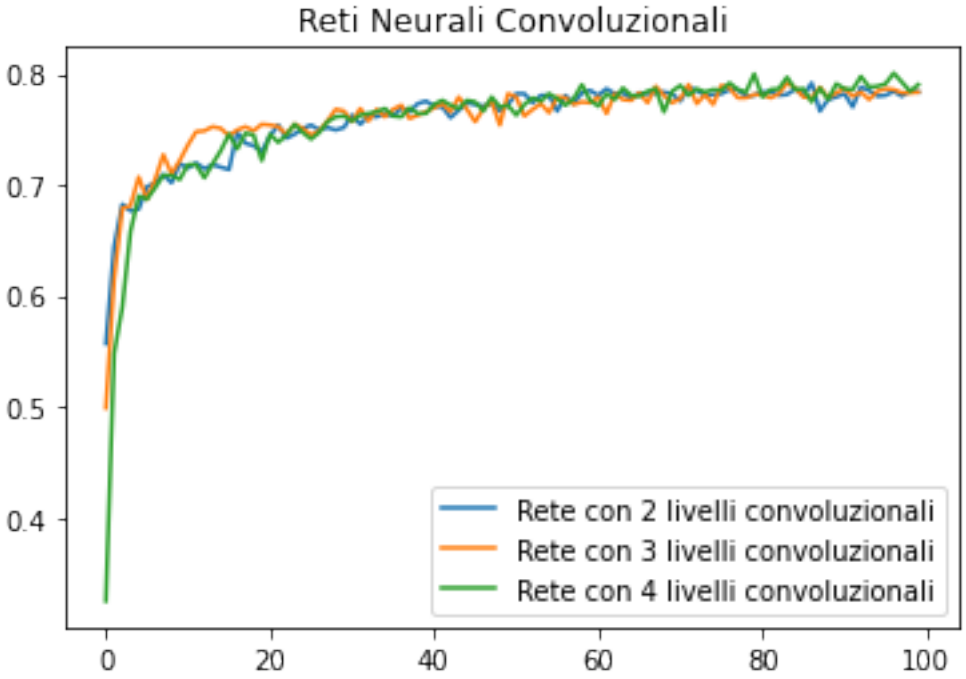
5.3.2 Reti Convoluzionali

Implementando una rete convoluzionale è stato osservato che le prestazioni non migliorano ma al contrario rimangono costanti, segno che gli esempi siano difficili da classificare in toto. I risultati della rete sono mostrati qui sotto e si nota come l'accuratezza rimanga tra il 75% e il 80%. La struttura della rete è la seguente:

- N livelli convoluzionali (con N che varia da 2 a 4)
- N-1 livelli di pooling
- un livello di Dropout
- un livello Flatten
- N livelli densi (con N che varia da 2 a 4)



Come si nota,fissando il numero di livelli convoluzionali pari a 2, al variare dei livelli densi l'accuratezza rimane fissa nel range 75-80% con l'uso di 100 epoche. L'ottimizzatore scelto è sempre *Adam* con i vari livelli che implementano la funzione di attivazione *relu* eccetto il livello di ouput che usa la **softmax**. Per completezza, si mostra come, anche cambiando il numero di livelli convoluzionali, le prestazioni rimangono identiche. Fissato quindi il numero di livelli densi a 4 si ottiene questo grafico:



Ad occhio si nota che, benchè le prestazioni non cambino, la velocità con cui la curva va a regime cresce all'aumentare dei livelli convoluzionali. La curva più stabile si ottiene con 4 livelli convoluzionali (rispettivamente di 16,32,64 e 128 neuroni) e 4 liv-

elli densi(rispettivamente di 32,16,8 e 4 neuroni). Come nelle sezioni precedenti, anche in questa fase sono stati eseguiti dei test modificando la funzione di attivazione, cambiando l'ottimizzatore in SGD, inserendo il regolarizzatore L2 negli ouput dei livelli ma l'accuratezza non subiva concreti miglioramenti. L'accuratezza ottentua dalla 10-fold cross validation è pari a **79,85%**.

5.4 Stima di densità

5.4.1 Parametrica

In questa sezione si procede nella definizione di un modello di stima di densità parametrica usando il package *sklearn.naive_bayes*. I risultati, a differenza dei precedenti e dei successivi, sono bassi. Sono stati provati diversi tipi di algoritmi e nella tabella sottostante sono riportate le performance.

Algoritmo	5 fold	10 fold	15 fold	20 fold
GaussianNB	0.532	0.532	0.531	0.533
MultinomialNB	0.511	0.511	0.512	0.513
ComplementNB	0.473	0.472	0.473	0.474
BernoulliNB	0.533	0.533	0.534	0.533

Table 17: Risultati K-Fold cross validation

Il modello migliore può essere ottenuto eseguendo gli algoritmi **GaussianNB** e **BernoulliNB**. Il secondo ha un'accuratezza che varia in base all'iperparametro *binarize*. Infatti esso esegue un'operazione di binarizzazione delle feature dipendentemente dal threshold definito.

5.4.2 Non Parametrica

Per la stima di densità non parametrica è stato scelto, come nelle immagini, l'algoritmo K-Nearest Neighbor. Dopo aver diviso il dataset in 2 parti (un training set ed un validation set), l'accuratezza ottenuta è pari a:

- 99.98% per il training set
- 74.6% per il test set

Come si nota dai risultati ottenuti, l'algoritmo ha creato un modello che si adatta quasi perfettamente ai dati con cui viene addestrato ma subisce un calo delle performance su esempi non visti. In realtà le prestazioni sono simili a quelle ottenute nei precedenti algoritmi utilizzati, infatti utilizzando k-fold cross validation si ottiene questo errore:

5 fold	10 fold	15 fold	20 fold
0.755	0.759	0.764	0.765

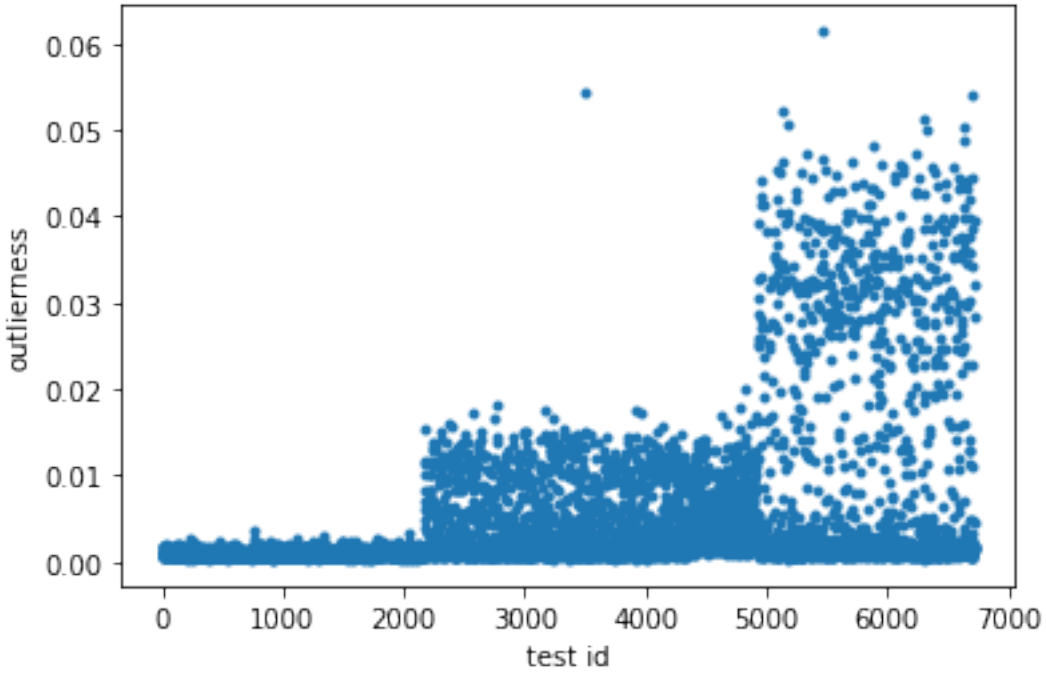
Table 18: Risultati K-fold cross validation

Il numero di vicini scelti per la classificazione è pari a 5 (si è notato infatti che al variare dei vicini, le performance rimangono pressochè simili). Stessa situazione avviene modificando la funzione che assegna i pesi: scegliendo **distance** o **uniform** si ottengono risultati simili.

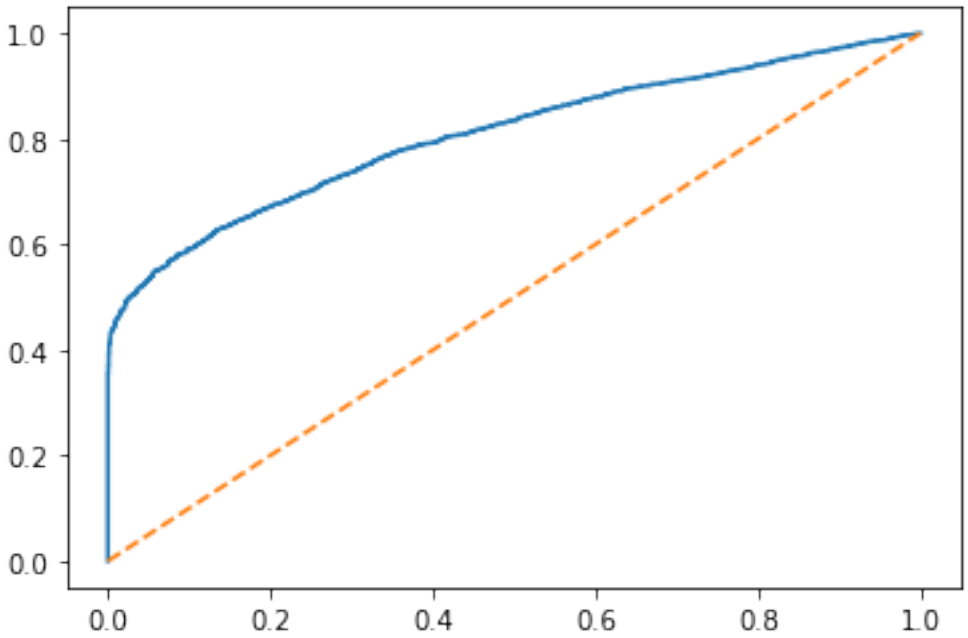
6 Anomaly Detection Dataset Audio

Per il task di anomaly detection si individua la classe più popolosa. In questo caso la dimensione degli esempi "happy" e "sad" sono in egual numero quindi si è scelto di procedere con la classe "happy". La tecnica utilizzata in questo caso è quella dell'AutoEncoder,

quindi una struttura neurale per ricostruire gli esempi della classe normale. Visti i risultati precendenti, anche qui non ci aspettiamo grosse sorprese. Prima di vedere la curva di ROC, è interessate stampare l'outlierness dei vari punti.



A differenza del dataset delle immagini, qui non c'è una vera differenza tra i vari punti. Si può notare che, siccome la classe "happy" è fatta di 2200 esempi circa e l'autoencoder è stato addestrato su di essa,l'errore relativo alla classe normale non supera il 0,003 (questo è un ottimo risultato). Di contro però, questo errore si ottiene anche nei punti outlier, anche se molti di essi hanno una loss che supera quel 0,003. Questo grafico anticipa il fatto che la curva di ROC non sarà perfetta e nella figura sottostante lo si può osservare.



Il valore di **AUC** restituito dalla curva è pari a 0.807. L'Autoencoder usato per

questo task ha una struttura definita come segue:

- un livello di input di dimensione 50
- un encoder fatto da 4 livelli di rispettivamente 512,256,128 e 64 neuroni
- un livello denso di 32 neuroni che rappresenta i dati nello spazio latente
- un decoder fatto da 4 livelli di rispettivamente 64,128,256 e 512
- un livello di output di 50 neuroni

Tutti i livelli utilizzando come funzione di attivazione la *relu* tranne quello di output che implementa la *sigmoide*. L'AutoEncoder è stato addestrato su 100 epoche con l'utilizzo della loss **MSE**. La forma dell'AutoEncoder non è casuale ma è stata scelta dopo varie sperimentazioni, infatti riducendo il numero di livelli o di neuroni per livello, l'accuratezza scendeva di molto. Con gli stessi livelli ma con un numero di neuroni per encoder e decoder pari a 32,16,8 e 4 si otteneva un pessimo valore di AUC pari a 0.54.