Relazione Progetto Machine e Deep Learning

A.A. 2019/2020

Francesco Pasceri 204963

Introduzione

Il progetto tratta l'applicazione di tecniche di machine e deep learning a diversi formati di dati in linguaggio Python.

In particolare l'attività richiede l'utilizzo di tecniche volte a due task: classificazione e anomaly detection. Entrambe i task di learning sono stati svolti sulle tipologie di dati a disposizione, ovvero immagini e testi.

Il primo passo dell'attività progettuale è lo studio e la preparazione dei dati, durante la quale sono utilizzate librerie per la modellazione delle immagini come *"PIL"* (Python Image Library) e alcuni metodi di *"keras"* e altre per la manipolazione dei testi prelevati da file.

Il secondo passo è volto all'esecuzione di alcune tecniche e lo studio delle loro curve di learning per valutare la loro qualità.

L'ultimo è il task di anomaly detection nel quale l'obiettivo è apprendere, seguendo un approccio *semi-supervised*, quale immagini e quali testi siano di una categoria interessante e quale invece risultano essere anomalie (o *outliers*).

La relazione si snoda su queste tre fasi del progetto ponendo attenzione alle scelte dedicate per ogni tipologia di dato e degli iperparametri che regolano le diverse tecniche di learning utilizzate.

1. Pre-processing dei dati

La fase di pre-processing dei dati è la fase più importante in quanto si studiano e si analizzano i dati che dovranno essere sottoposti ad analisi mediante diverse tecniche. In particolare, si prelevano i dati nel loro formato originale e li si trasformano in un dataset, uno per ogni tipologia di dato, da dare in input agli algoritmi.

1.1. Immagini

La prima tipologia di dati da analizzare è quella delle immagini.
Nel nostro caso di studio le immagini sono delle piccole figure in formato JPG di dimensione 80 x 60 pixel. Queste sono di quattro (4) classi: *belts*, *casual-shoes*, *sportive-shoes* e *wallets*. Ognuna delle precedenti classi rappresenta un indumento diverso di abbigliamento e sono caratterizzate da diverse forme:

Si può notare come, in realtà, le informazioni riguardanti il colore non sono fondamentali alla distinzione delle diverse tipologie di immagini. Inoltre, la differenza tra la classe delle scarpe casual e quella delle scarpe sportive, in alcuni casi, è molto sottile anche per una classificazione umana tanto da poter scambiare alcuni esempi tra le due classi.
La presenza di immagini difficilmente distinguibili già ad un intervento umano implica una certa difficoltà nel raggiungere livelli di accuratezza vicini il 100% sia nel caso del task di classificazione, sia in quello di anomaly detection (dove la difficoltà è evidenziata anche in termini di analisi delle curve).

A sinistra una scarpa rientrante nella tipologia "casual". A destra quella della classe "sport"

Com'è normale, l'immagine è prelevata in questo formato mediante il metodo *Image.open(file)* del modulo Image della libreria PIL; inoltre è utilizzato anche il metodo *Image.convert('L')* per indicare al programma la trasformazione delle immagini da colori a scala di grigi. In termini algebrici, si passa quindi da una immagini come tensore (larghezza, altezza, 3 canali rgb) ad una semplice matrice bidimensionale (larghezza, altezza, livello di luminosità).

A completare il pre-processing dell'immagini vi sono il ridimensionamento dell'immagine in formato 80 x 64 elementi, ottenuta con un semplice *resize(shape=(width, height))*, e la normalizzazione dei valori (dividendo per 255.0) con conversione a numeri *float* per evitare problematiche di differenze molto evidenti tra intervalli di valori.
Al termine delle operazioni, ogni immagini è resa *flatten* ovvero ridotta ad un vettore di dimensione 80 x 64 posizioni ed inserita come riga del dataset delle immagini, mentre un vettore di etichette (o *label*) è mantenuto per ricordare ogni riga del vettore a quale classe appartiene seconda la seguente tabella:

Belt	Casual-shoes	Sport-shoes	Wallet
0	1	2	3

1.2. Testi

Il pre-processing dei dati utilizza più funzioni poiché a differenza delle immagini, le parole caratterizzanti i testi sono molte di più nella totalità dei documenti.
Nel nostro caso i file testuali sono in formato TXT, quindi testo semplice, e suddivisi in 2 categorie, ognuna delle quali ha altre sottoclassi:

1. Ham (ovvero email interessanti per l'utente che le riceve)
    a. farmer
    b. kaminski
    c. beck
    d. lokay
    e. kitchen
    f. williams
2. Spam (email non interessanti)
    a. SA and HP
    b. GP
    c. BG

Ognuna delle precedenti classi, codificate con numeri interi da 0 ad 8 nel medesimo ordine dell'elenco, è caratterizzata da diversi file con diversi contenuti anche se alcune parole sono riprese in file della stessa tipologia (es. nei file di *farmer* sono quasi sempre presenti le parole *power* e *energy* anche se a volte scritte in maniera errata o in forma derivata).

Le operazioni svolte sui testi sono state le seguenti:

- Trasformazione in minuscolo: si trasformano tutte le lettere delle parole in lower case per normalizzarle ad uno stesso formato;
- Rimozione dei numeri e degli spazi: si eliminano dalla sequenza del testo tutti i numeri e gli spazi bianchi;
- Tokenizzazione: questo passo è interessante in quanto consente di eseguire la tokenizzazione del testo, ovvero di fare lo *splitting* delle parole in singoli elementi e modellare il testo come una lista di elementi, o meglio di parole;
- Rimozione delle stopwords: in quest'unico passo, il più costoso, si analizzano tutte le parole e si eliminano tutte quelle che rappresentano le cosiddette *stopword*, ovvero parole di contenuto semantico nullo come articoli, preposizioni, soggetti, ecc. ecc.;
- Lemmatizzazione: per ogni parola che non è stata identificata come *stopword* si esegue la lemmatizzazione, ovvero la trasformazione della parola nella sua forma base (*energizer -> energy*, *feeding -> feed*, …);

Per concludere le operazioni di pre-processing si definisce un vettore di stringhe in cui ogni stringa corrisponde ad un testo trasformato e ridotto.
Prima però di dare in input il dataset delle stringhe agli algoritmi di machine e deep learning, si procede alla *Tfid-vectorization* mediante l'oggetto Python omonimo che trasforma la stringa del testo in un vettore di un numero massimo di elementi (secondo il parametro *max features*) ordinato a seconda del valore *tf-idf* (ovvero $tf_{i,j} = \frac{n_{i,j}}{|d_j|}$, dove il numeratore è il numero di occorrenze del termine i nel documento j, mentre il denominatore è la lunghezza del documento j usato per normalizzare).

2. Classificazione di immagini

Una volta composto il dataset delle immagini, vettorizzate in 80 x 64 elementi, si danno in input ai diversi algoritmi di learning. I diversi modelli di learning sono dapprima addestrati e successivamente valutati ognuno sulla base dei risultati della *confusion matrix* e dei parametri di *precision*, *recall* e *f1-score* (il quale mette in relazione le precedenti due misure di qualità).
In particolare: la *confusion matrix* mette in evidenza il numero di elementi ben classificati e misclassificati; la *precision* rappresenta il rapporto tra numero di elementi classificati correttamente sul numero totale di osservati, ovvero più è alta, più è bassa la possibilità di ottenere un falso positivo; la *recall* è il rate di classificati correttamente sulla base di tutti quelli della classe stessa; infine, la *f1 score* è la media pesata tra le due precedenti misure che raccoglie le informazioni e ne dà una stima che rappresenta meglio la qualità dell'algoritmo stesso.

2.1. Adaboost

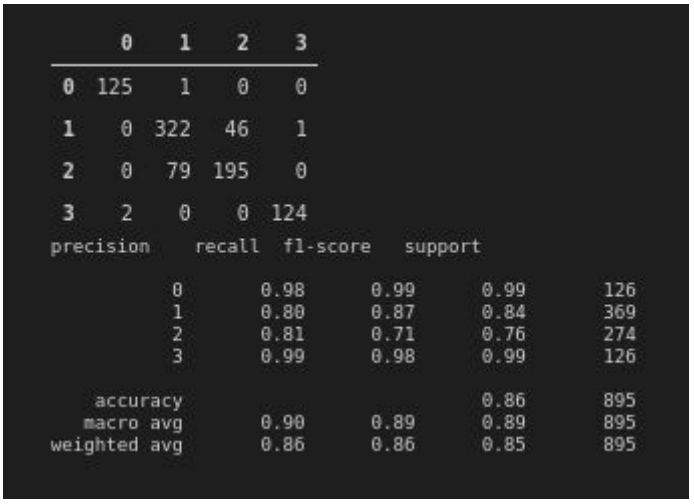
Il primo classificatore è l'Adaboost, una tecnica Ensemble Boosting, che combina diversi weak learner (diversi Decision Tree) per definire un learner più sensato. L'idea alla base è di addestrare in maniera sequenziale i weak learner: ognuno di essi parte dal risultato del precedente cercandone di correggere gli errori.

Il modello Adaboost lo si crea effettuando una ricerca di *fine-tuning*, la *GridSearchCV*, sugli iperparametri dello stesso modello che sono:

- *n_estimators:* rappresenta il numero di weak learner che vengono addestrati, nel caso di studio il valore viene fatto variare nell'insieme {5, 10, 20} poiché già con questo numero di learner si va a configurare un modello Adaboost che raggiunge l'85% di accuratezza ed una buona classificazione;
- *learning_rate:* rappresenta il contributo che ogni weak learner dà sui pesi e diminuendo questo indice, il modello è più lento nell'addestramento. Di default il valore è pari ad 1, ma per ottenere risultati più attenti la *GridSearchCV* varia anche con i valori 10^{-3} e 10^{-2};
- *base_estimator:* è la tipologia di weak learner lasciata al suo valore default, ovvero i *DecisionTree*. Di questa tipologia varia il valore della massima profondità, ovvero quante features imparano per classificare. Il set di valori è {5, 10, 20, 30}.

```
Best params:  AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=10,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1, n_estimators=60, random_state=None)
```

Il risultato del modello è un Adaboost con 20 weak learner di profondità massima pari a 20 (quindi già vuol dire che con DecisionTree di profondità 30 il modello va in *overfitting*). Il learning rate migliore rimane quello di default con questa tecnica per effettuare *fine-tuning*.
Dalla matrice di confusione possiamo concludere che in generale questo modello performa in maniera soddisfacente sulla maggior parte delle classi; come prevedibile, il modella ha qualche incertezza sugli esempi delle due classi di scarpe.



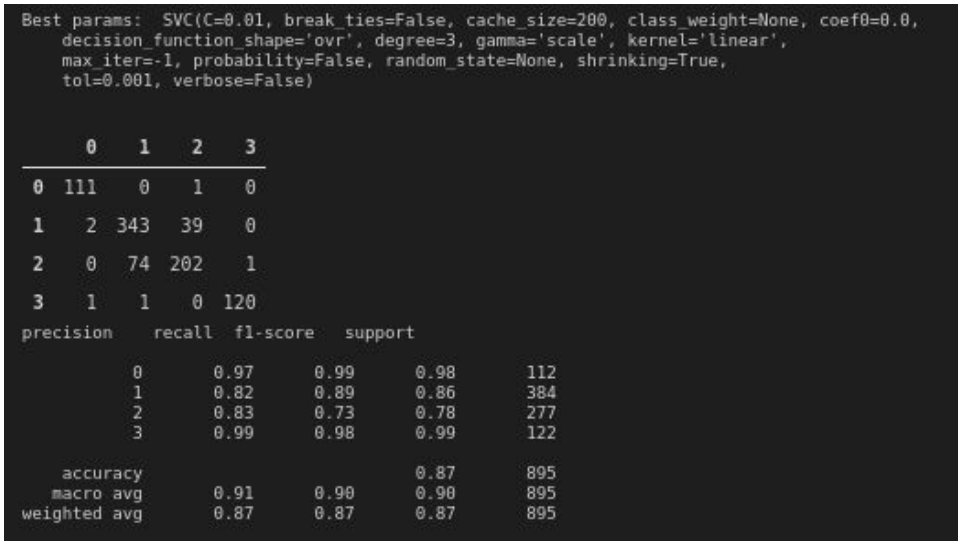
	0	1	2	3
0	125	1	0	0
1	0	322	46	1
2	0	79	195	0
3	2	0	0	124

	precision	recall	f1-score	support
0	0.98	0.99	0.99	126
1	0.80	0.87	0.84	369
2	0.81	0.71	0.76	274
3	0.99	0.98	0.99	126
accuracy			0.86	895
macro avg	0.90	0.89	0.89	895
weighted avg	0.86	0.86	0.85	895

2.2. Support Vector Machine (SVM)

Il secondo algoritmo di machine learning è il *Support Vector Machine*, alla cui base sono utilizzate tre tipologie di kernel: *linear*, *polynomial* e *rbf* (*radial basis function*).

Nella prima tipologia di kernel, quello *linear*, di SVM si effettua una *GridSearchCV* sull'iper-parametro C di regolarizzazione, che rappresenta un trade off tra una corretta classificazione e la massimizzazione dei margini in maniera tale da evitare l'*overfitting* e una buon valore di precisione di classificazione.



```
Best params:  SVC(C=0.01, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

	0	1	2	3
0	111	0	1	0
1	2	343	39	0
2	0	74	202	1
3	1	1	0	120

	precision	recall	f1-score	support
0	0.97	0.99	0.98	112
1	0.82	0.89	0.86	384
2	0.83	0.73	0.78	277
3	0.99	0.98	0.99	122
accuracy			0.87	895
macro avg	0.91	0.90	0.90	895
weighted avg	0.87	0.87	0.87	895

Il valore migliore del parametro *degree* per il secondo kernel è ricercato tra valori {1,2,3,4,5} in maniera tale da definire il polinomio più interessante. Quello che performa in maniera più interessante è il grado 2 che risulta anche essere il più utilizzato in vari ambiti con SVM; gradi più elevati non sono ottimali in quanto incrementano il *fitting* al training set e quindi in fase di predizione risulta avere risultati peggiori.

```
Best params:  SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)


     0    1    2    3
 0 118    0    0    0
 1   1  318   51    0
 2   0   69  211    0
 3   2    0    0  125
precision    recall  f1-score   support

           0       0.98      1.00      0.99       118
           1       0.82      0.86      0.84       370
           2       0.81      0.75      0.78       280
           3       1.00      0.98      0.99       127

    accuracy                           0.86       895
   macro avg       0.90      0.90      0.90       895
weighted avg       0.86      0.86      0.86       895
```

Infine, l'*rbf* è l'ultima tipologia di kernel utilizzato e, lasciato con i parametri di default, si comporta complessivamente meglio dei precedenti due kernel.

```
Best params:  SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)


     0    1    2    3
 0  91    0    1    0
 1   1  345   45    0
 2   0   71  216    0
 3   2    1    0  122
precision    recall  f1-score   support

           0       0.97      0.99      0.98        92
           1       0.83      0.88      0.85       391
           2       0.82      0.75      0.79       287
           3       1.00      0.98      0.99       125

    accuracy                           0.86       895
   macro avg       0.90      0.90      0.90       895
weighted avg       0.86      0.86      0.86       895
```

2.3. Density Estimation

La terza tipologia di classificatori è quella della classe degli stimatori di densità, ovvero tecniche che sfruttano gli esempi del training set per definire la densità negli intorni dei punti.

Il primo classificatore è quello *kNearestNeighbor* che utilizza l'omonimo algoritmo per fare classificazione. La *GridSearchCV* è lanciata su diversi parametri tra cui i più interessanti sono due:

- *n_neighbors*: il numero di vicini che deve essere considerato per definire il raggio dell'intorno del punto che necessita l'etichettatura. In particolare, variando il valore tra 1 e 50 vicini, il miglior numero risulta essere il 4 in quanto i successivi portano ad un eccessivo *overfitting*;
- *algorithm:* definisce l'algoritmo utilizzato per la ricerca dei *nearest neighbor*. Nel nostro caso il migliore risulta essere il *KD-Tree* che consente di velocizzare il calcolo dei punti vicini sfruttando una struttura ad albero.

```
Best params:  KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=1,
                     weights='distance')
```

	0	1	2	3
0	106	1	0	0
1	0	368	36	0
2	0	102	168	1
3	0	1	0	112

	precision	recall	f1-score	support
0	1.00	0.99	1.00	107
1	0.78	0.91	0.84	404
2	0.82	0.62	0.71	271
3	0.99	0.99	0.99	113
accuracy			0.84	895
macro avg	0.90	0.88	0.88	895
weighted avg	0.85	0.84	0.84	895

Il secondo e terzo stimatore di densità sono due versione del Naive Bayes in versione gaussiana e complementare: *GaussianNB* e *ComplementNB*. Entrambi, presentano i due stimatori di densità più "scarsi" dal punto di vista dell'accuratezza anche se presentano molte difficoltà principalmente nell'apprendere la differenza tra le due classi di scarpe.

```
Best params:  ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
```

	0	1	2	3
0	97	1	7	2
1	2	301	93	8
2	5	119	143	4
3	1	1	2	109

	precision	recall	f1-score	support
0	0.92	0.91	0.92	107
1	0.71	0.75	0.73	404
2	0.58	0.53	0.55	271
3	0.89	0.96	0.92	113
accuracy			0.73	895
macro avg	0.78	0.79	0.78	895
weighted avg	0.72	0.73	0.72	895

```
Best params:  GaussianNB(priors=None, var_smoothing=1e-09)
```

	0	1	2	3
0	102	0	0	5
1	52	180	15	157
2	17	144	21	89
3	2	0	5	106

	precision	recall	f1-score	support
0	0.59	0.95	0.73	107
1	0.56	0.45	0.49	404
2	0.51	0.08	0.13	271
3	0.30	0.94	0.45	113
accuracy			0.46	895
macro avg	0.49	0.60	0.45	895
weighted avg	0.51	0.46	0.41	895

2.4. Neural Network e Convolutional Neural Network (CNN)

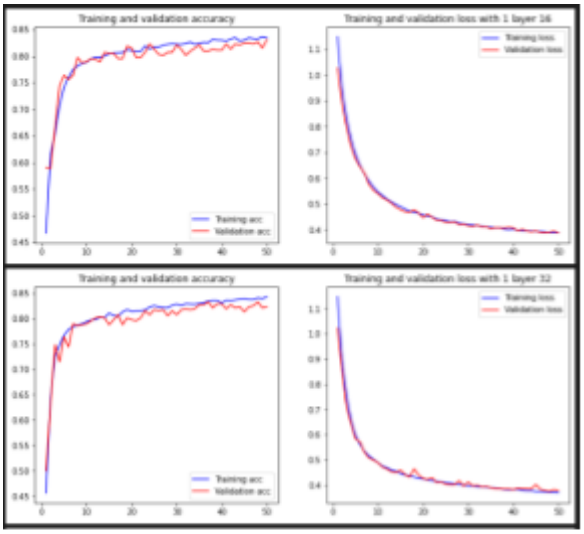
Le ultime due classi di classificatori per le immagini sono rappresentate da due versioni di reti neurali: quelle Dense e quelle Convoluzionali. Nelle prime si danno in input le immagini sotto forma di vettori di dimensione

80 x 64 mentre le seconde utilizzano direttamente le immagini come matrici di pixel, in realtà come matrici di float dei livelli di luminosità.

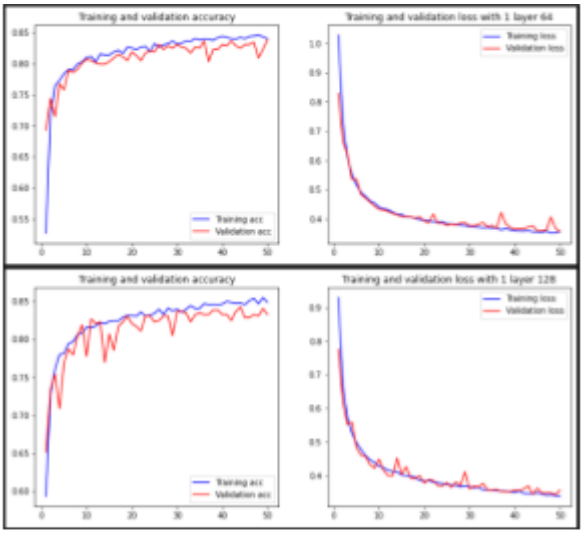
Per le reti neurali dense sono testate diverse architetture:

- 1 layer con 16 neuroni;
- 1 layer con 32 neuroni;
- 1 layer con 64 neuroni;
- 1 layer con 128 neuroni
- 1 layer con 256 neuroni;

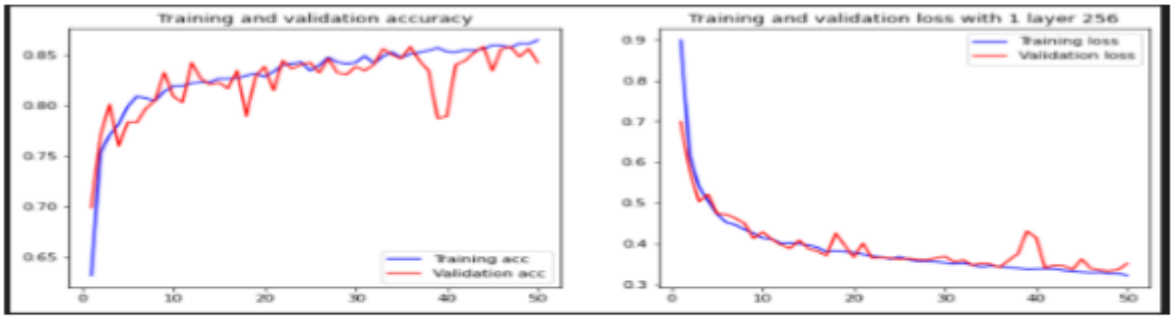
Tutti i test effettuati sono eseguiti definendo una loss di tipo *categorical_crossentropy* per avere classificazione multiclasse e con ottimizzatore *Adam* il cui learning rate è stato sceso a 10^{-5} per avere un learning più lento ma più accurato. Inoltre, con un *learning rate* più basso diminuiscono le fluttuazioni delle due curve di *accuracy* e di *loss*.



16 (sopra) e 32 (sotto) neuroni per 1 layer



64 (sopra) e 128 (sotto) neuroni per 1 layer

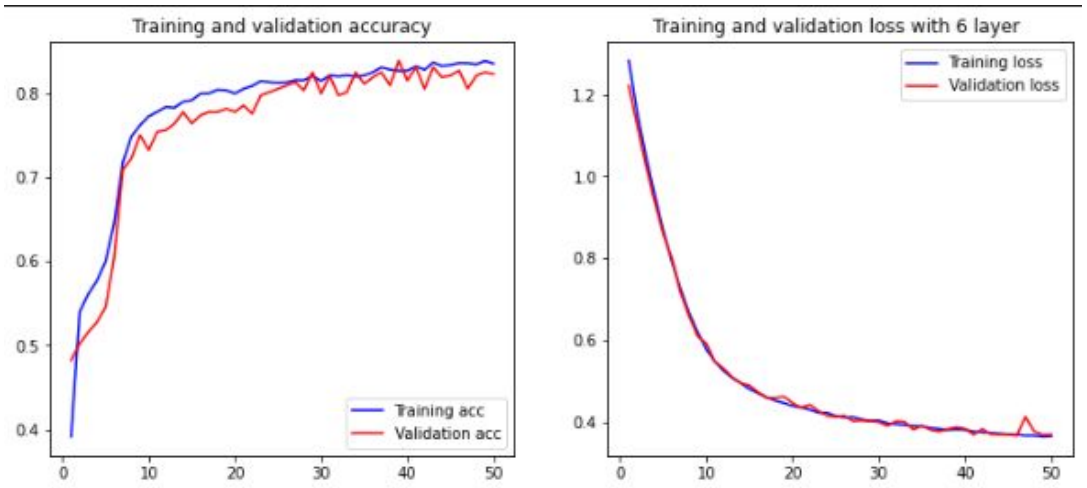


256 neuroni per 1 layer

Un secondo step di analisi delle reti dense è svolto per approfondire l'architettura studiando una eventuale aggiunta di layer Dense. Nello specifico il numero di layer varia nell'insieme {2, 4, 6, 10, 20, 50} e i risultati, come si evince dai grafici sottostanti, sono in piccolo ma costante miglioramento fino a 6 layer; rendendo più profonda la rete i risultati sono degradanti e con maggiori fluttuazioni.

La scelta progettuale del modello più interessante per la predizione ricade tra l'architettura con 4 hidden layer con 128 neuroni e quella con 6 layer formati nel medesimo modo. Il primo modello di rete neurale densa però

risulta più soddisfacente in termini di velocità di apprendimento e di semplicità della struttura in pari termini di accuratezza e valore di loss.



	0	1	2	3
0	114	0	1	1
1	0	329	55	1
2	5	65	204	0
3	1	0	0	119

	precision	recall	f1-score	support
0	0.95	0.98	0.97	116
1	0.84	0.85	0.84	385
2	0.78	0.74	0.76	274
3	0.98	0.99	0.99	120
accuracy			0.86	895
macro avg	0.89	0.89	0.89	895
weighted avg	0.85	0.86	0.85	895

La seconda tipologia di rete neurale è quella della rete convoluzionale (CNN) che sfrutta l'estrazione di diverse feature delle immagini per la distinzione delle varie classi di appartenenza. Sono qui utilizzati i layer convoluzionali (*Conv2D*), i quali a partire da porzioni di immagini estraggono delle informazioni legando i pixel della sezione interessata; l'altra tipologia di layer usata è quella *MaxPool2D*, la quale estrae dai pixel considerati il valore maggiore e dimezza la dimensione dell'immagine per ottenere più feature estratte.
Sono costruite due reti le quali prevedono l'utilizzo dei dati sia in *grayscale* sia che con RGB ed è proprio nel secondo caso in cui si ottengono risultati più interessanti vicino al 90%.

	0	1	2	3
0	133	1	0	0
1	0	318	42	1
2	0	56	219	1
3	0	0	0	124

	precision	recall	f1-score	support
0	1.00	0.99	1.00	134
1	0.85	0.88	0.86	361
2	0.84	0.79	0.82	276
3	0.98	1.00	0.99	124
accuracy			0.89	895
macro avg	0.92	0.92	0.92	895
weighted avg	0.89	0.89	0.89	895

CNN su scala di grigi

	0	1	2	3
0	107	0	0	0
1	1	345	48	1
2	0	34	244	1
3	1	1	0	112

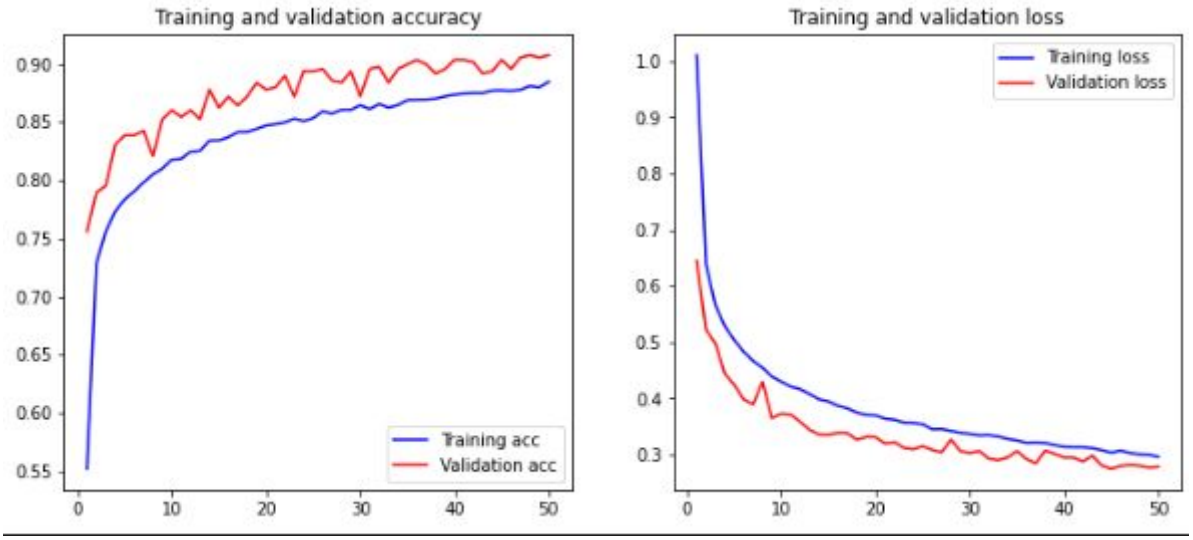
	precision	recall	f1-score	support
0	0.98	1.00	0.99	107
1	0.91	0.87	0.89	395
2	0.84	0.87	0.85	279
3	0.98	0.98	0.98	114
accuracy			0.90	895
macro avg	0.93	0.93	0.93	895
weighted avg	0.90	0.90	0.90	895

CNN su scala di grigi

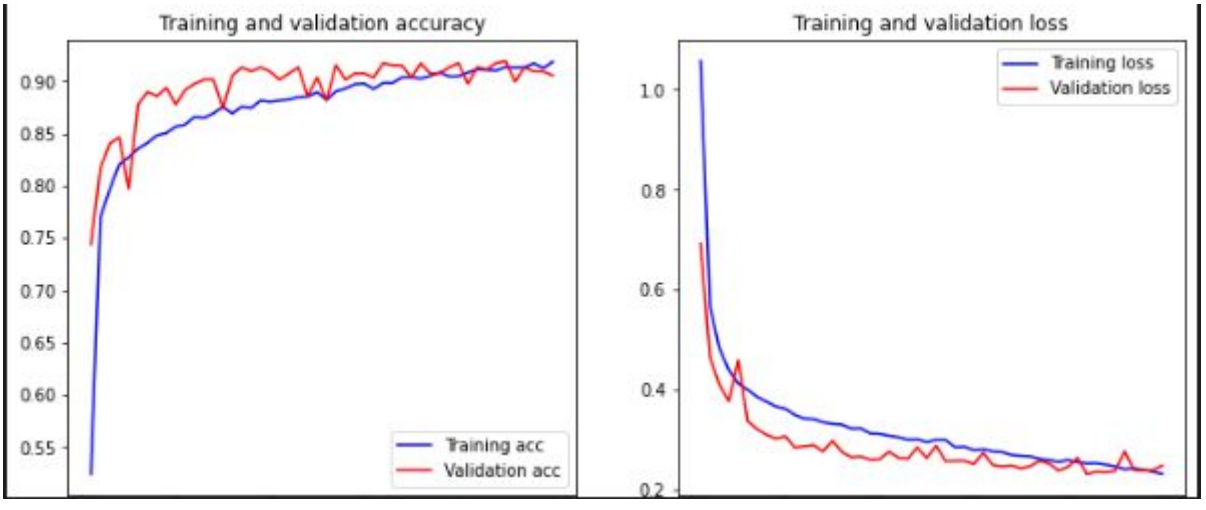
La CNN è studiata a partire da una configurazione base: rete convoluzionale per l'estrazione delle feature e un ultimo segmento di rete denso per la classificazione.

Un tentativo volto ad incrementare il risultato della CNN su scala di grigi è la *Data Augmentation* in maniera tale da fornire alla rete stessa un numero maggiore di dati in input su cui essere addestrata. Mediante il modulo Python *PIL.Image* si sono prese le singole immagini del training set e le si sono ruotate e specchiate riportandole sempre ad una dimensione di 80 x 64 pixel.

Il risultato dell'addestramento induce un leggero distaccamento tra la curva di training e quella di validation, quindi si introduce della regolarizzazione negli *hidden layers* per evitare migliorare la curva della rete. In particolare, la regolarizzazione è basata sul tipo *l2* con un fattore di 10^{-3}. Ciò che vien fuori da rete convoluzionale con regolarizzazione e data augmentation è la curva sottostante:



Il risultato finale della è quello che riesce a raggiungere un livello di accuratezza interessante e molto simile a quello della CNN su immagini RGB. A quest'ultima è inserita anche della regolarizzazione sui neuroni per controllare il principio di overfitting in maniera analoga a quanto effettuato con la versione precedente. E quindi graficando accuracy e loss si ottiene il seguente grafico.



Sia nel caso di CNN che delle reti dense, la funzione di attivazione utilizzata dai neuroni è la *ReLu*:

$$ReLu(x) = max\{0,\ x\}$$

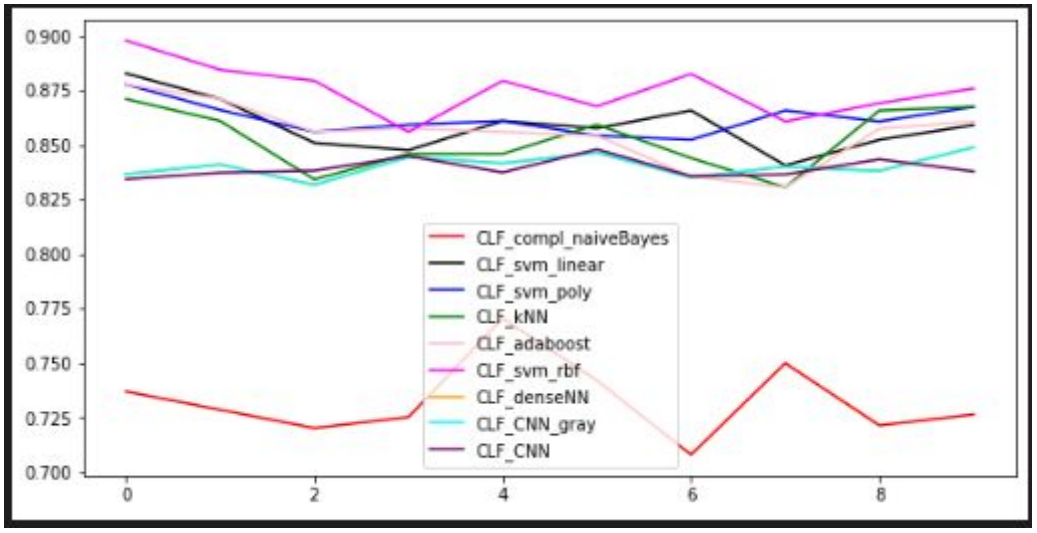
Questa funzione di attivazione è utile in quanto valori negativi di x sono mappati nel valore 0 della funzione, inoltre, è utile nella *backpropagation* in quanto consente di evitare l'effetto di *vanishing gradient*.

2.5. Cross Validation

Una volta costruiti tutti i classificatori è eseguita la *k-fold cross validation*. Questa tecnica consente di controllare l'andamento dell'accuratezza dei test andando a limitare eventuali problematiche di overfitting o ad evidenziarle nel caso in cui il modello sia errato.

L'esecuzione di tale controllo è avvenuto mediante il modulo *"sklearn.model_selection.cross_val_score"* che consente di eseguire appunto tale tecnica e si è impostato un valore di k pari a 10. In tal modo, il training set è diviso in 10 partizioni una delle quali a turno assume il ruolo di validation set mentre le restanti nove (9) svolgono il ruolo di training set.

Di seguito il risultato dell'esecuzione con tutti i classificatori vicini tra loro in fase di validation.



3. Classificazione di testi

In questo terzo capitolo affrontiamo la classificazione dei testi che varia leggermente da quella delle immagini e per lo più questo avviene nel caso delle reti neurali. Infatti, come le immagini anche i testi, dopo il pre-processing, si sono ridotti ad essere dei vettori in cui le parole non sono più lettere ma sono diventate dei numeri che esprimono il loro valore *tfidf* a seguito della *Tfidf-Vectorization*. Essendo quindi dei vettori numerici, sono introdotti direttamente negli algoritmi di machine e deep learning come i precedenti vettori delle immagini. Anche nel caso della classificazione testuale, i parametri studiati sono accuracy, precision, recall e f1-score mentre per la visualizzazione dei risultati entra in gioco la *confusion matrix*.

A differenza del capitolo precedente, si limita la spiegazione dei singoli algoritmi qualora questi risultino uguali al caso precedente e si introducono solo delle brevi giustificazioni progettuali ai vari iper-parametri.

3.1. Adaboost

Come nel caso della classificazione di immagini, il modello di Adaboost si basa su una GridSearchCV svolta sui parametri: numero di weak learner, learning rate e tipologia di weak learner. Proprio per quest'ultimo, poiché la differenziazione dei testi necessita di maggiore attenzione al dettaglio, sono stati inseriti DecisionTreeClassifier più profondi per consentire un migliore apprendimento dei parametri ma, grazie al *fine-tuning* si evitano alberi troppo profondi che generano overfitting.
Al termine della ricerca, si evince che la migliore configurazione risulta essere quella con 20 weak learner ognuno a *max_depth* cinquanta (50) e quindi si è scelto tale modello per portare a termine la classificazione ottenendo i seguenti risultati.

```
Best params:  AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=50,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=0.01, n_estimators=20, random_state=None)
```

Modello risultante da GridSearchCV con Adaboost

	0	1	2	3	4	5	6	7	8
0	809	32	27	31	64	13	12	56	16
1	25	1013	35	29	58	3	19	32	21
2	32	35	240	18	55	7	7	20	16
3	38	41	20	186	61	11	10	25	16
4	47	57	52	46	852	21	11	38	29
5	21	16	3	9	50	321	10	17	6
6	8	6	2	1	6	0	1087	139	152
7	11	12	3	5	21	2	144	1110	419
8	10	7	2	2	7	4	170	431	1011

Confusion matrix Adaboost

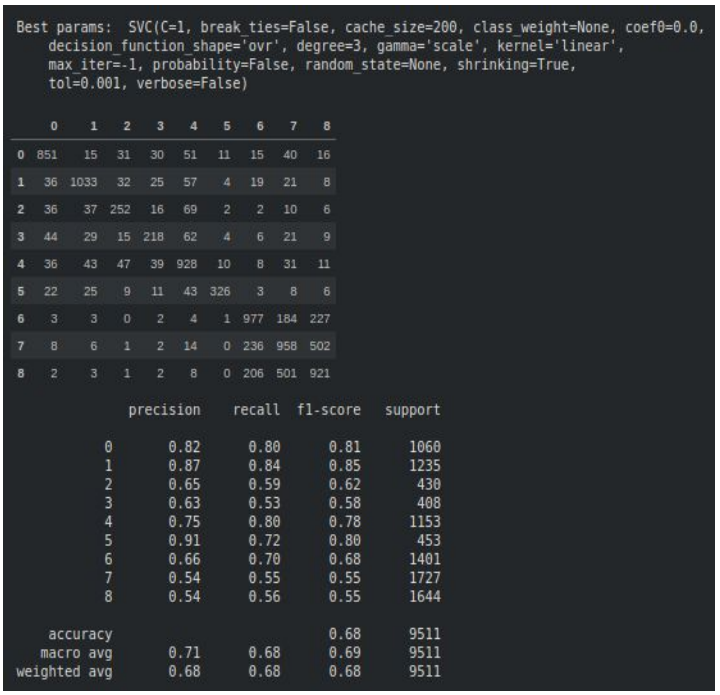
	precision	recall	f1-score	support
0	0.81	0.76	0.79	1060
1	0.83	0.82	0.83	1235
2	0.62	0.56	0.59	430
3	0.57	0.46	0.51	408
4	0.73	0.74	0.73	1153
5	0.84	0.71	0.77	453
6	0.74	0.78	0.76	1401
7	0.59	0.64	0.62	1727
8	0.60	0.61	0.61	1644
accuracy			0.70	9511
macro avg	0.70	0.68	0.69	9511
weighted avg	0.70	0.70	0.70	9511

Report delle misure

3.2. SVM

Anche nel caso del Support Vector Machine adattato alla classificazione testuale, sono utilizzati i tre tipi di kernel precedenti: *linear*, *polynomial* e *rbf*.

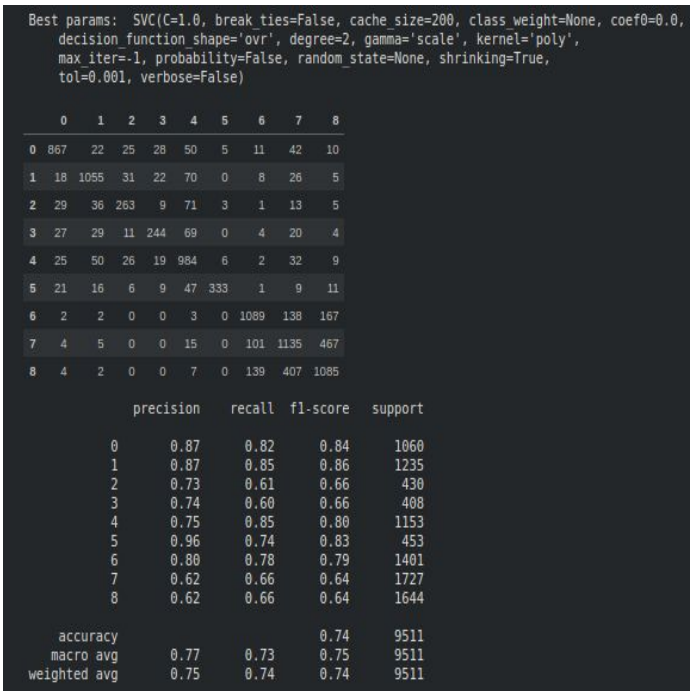
La differenza rispetto il caso della classificazione di immagini è quasi nullo, parametri e valori degli stessi sono fatti variare negli stessi range e si ottengono dei risultati superiori al 75% di accuratezza totale. Per chiarezza, di seguito si possono osservare i risultati ottenuti dai diversi modelli di SVM.



```
Best params:  SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

	0	1	2	3	4	5	6	7	8
0	851	15	31	30	51	11	15	40	16
1	36	1033	32	25	57	4	19	21	8
2	36	37	252	16	69	2	2	10	6
3	44	29	15	218	62	4	6	21	9
4	36	43	47	39	928	10	8	31	11
5	22	25	9	11	43	326	3	8	6
6	3	3	0	2	4	1	977	184	227
7	8	6	1	2	14	0	236	958	502
8	2	3	1	2	8	0	206	501	921

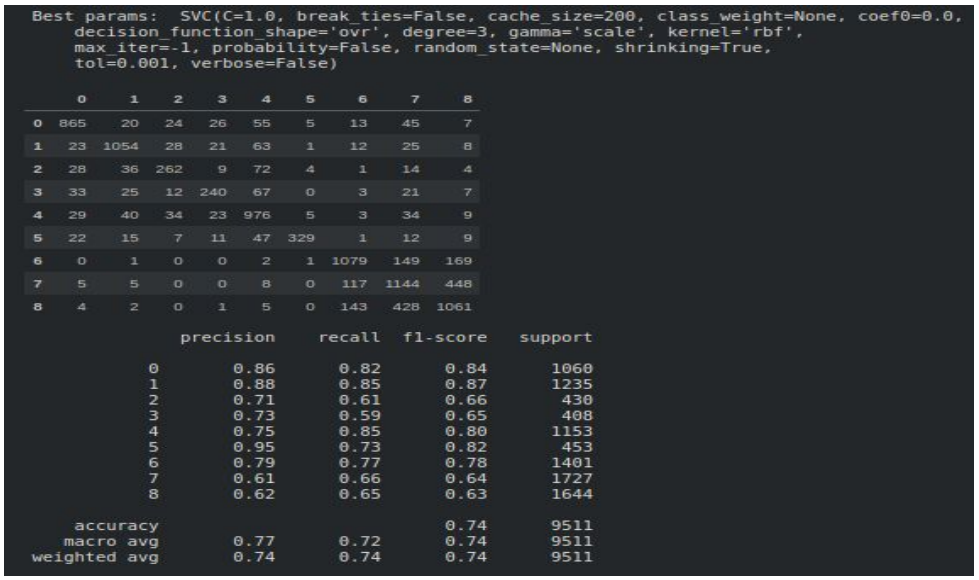
	precision	recall	f1-score	support
0	0.82	0.80	0.81	1060
1	0.87	0.84	0.85	1235
2	0.65	0.59	0.62	430
3	0.63	0.53	0.58	408
4	0.75	0.80	0.78	1153
5	0.91	0.72	0.80	453
6	0.66	0.70	0.68	1401
7	0.54	0.55	0.55	1727
8	0.54	0.56	0.55	1644
accuracy			0.68	9511
macro avg	0.71	0.68	0.69	9511
weighted avg	0.68	0.68	0.68	9511



```
Best params:  SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

	0	1	2	3	4	5	6	7	8
0	867	22	25	28	50	5	11	42	10
1	18	1055	31	22	70	0	8	26	5
2	29	36	263	9	71	3	1	13	5
3	27	29	11	244	69	0	4	20	4
4	25	50	26	19	984	6	2	32	9
5	21	16	6	9	47	333	1	9	11
6	2	2	0	0	3	0	1089	138	167
7	4	5	0	0	15	0	101	1135	467
8	4	2	0	0	7	0	139	407	1085

	precision	recall	f1-score	support
0	0.87	0.82	0.84	1060
1	0.87	0.85	0.86	1235
2	0.73	0.61	0.66	430
3	0.74	0.60	0.66	408
4	0.75	0.85	0.80	1153
5	0.96	0.74	0.83	453
6	0.80	0.78	0.79	1401
7	0.62	0.66	0.64	1727
8	0.62	0.66	0.64	1644
accuracy			0.74	9511
macro avg	0.77	0.73	0.75	9511
weighted avg	0.75	0.74	0.74	9511



```
Best params:  SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

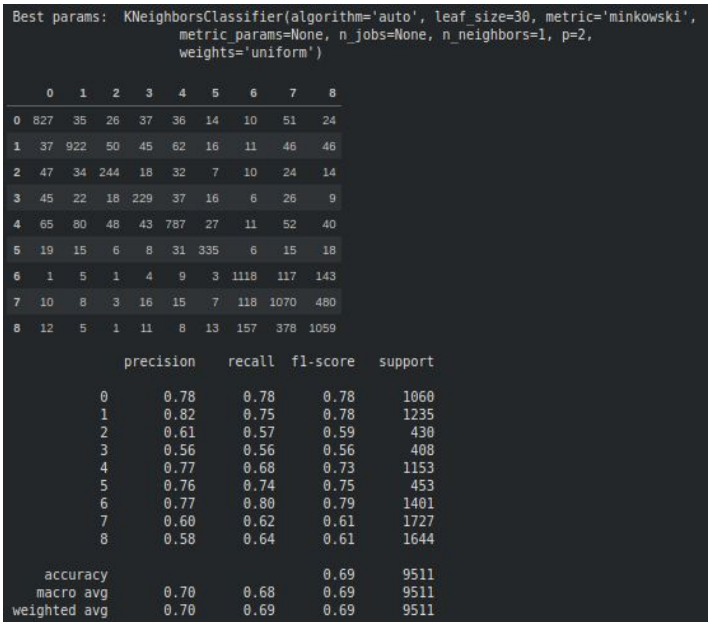
	0	1	2	3	4	5	6	7	8
0	865	20	24	26	55	5	13	45	7
1	23	1054	28	21	63	1	12	25	8
2	28	36	262	9	72	4	1	14	4
3	33	25	12	240	67	0	3	21	7
4	29	40	34	23	976	5	3	34	9
5	22	15	7	11	47	329	1	12	9
6	0	1	0	0	2	1	1079	149	169
7	5	5	0	0	8	0	117	1144	448
8	4	2	0	1	5	0	143	428	1061

	precision	recall	f1-score	support
0	0.86	0.82	0.84	1060
1	0.88	0.85	0.87	1235
2	0.71	0.61	0.66	430
3	0.73	0.59	0.65	408
4	0.75	0.85	0.80	1153
5	0.95	0.73	0.82	453
6	0.79	0.77	0.78	1401
7	0.61	0.66	0.64	1727
8	0.62	0.65	0.63	1644
accuracy			0.74	9511
macro avg	0.77	0.72	0.74	9511
weighted avg	0.74	0.74	0.74	9511

3.3. Density Estimation

Per quanto riguarda la classificazione avanzata con dei modelli di stima di densità si è utilizzato un'altra volta il k Nearest Neighbor, mentre come varianti del Naive Bayes questa volta abbiamo confrontato tutte le soluzioni: *Bernoulli*, *Categorical*, *Complement*, *Gaussian* e *Multinomial*.

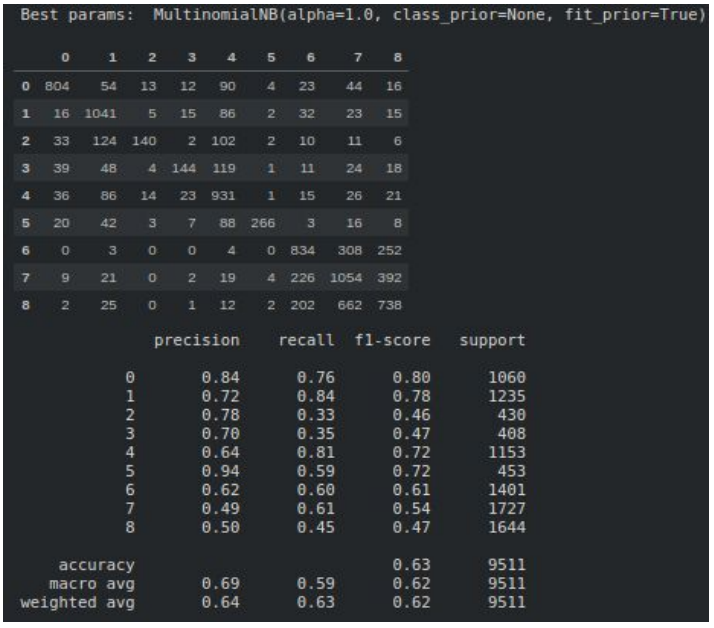
Se per il kNN, la ricerca del *fine-tuning* segue la medesima strada indicata nel paragrafo 2.3, l'approccio al Naive Bayes è cambiato in base alla natura stessa dei dati.
Come prevedibile i peggiori sono i Bernoulli e Categorical in quanto: il primo è più utile nel caso di feature di tipo binario o booleano; il secondo modello, invece, lavora bene su distribuzioni abbastanza categoriche nel distinguere i valori e ciò non rientra nel nostro caso.
La variante MultinomialNB risulta essere la migliore e la più performante nel classificare anche grazie al fatto che riesce a lavorare molto bene su feature che esprimono le *term frequencies* (ciò che è ottenuto nel capitolo 1 di pre-processing dei testi).



Best params: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=1, p=2, weights='uniform')

	0	1	2	3	4	5	6	7	8
0	827	35	26	37	36	14	10	51	24
1	37	922	50	45	62	16	11	46	46
2	47	34	244	18	32	7	10	24	14
3	45	22	18	229	37	16	6	26	9
4	65	80	48	43	787	27	11	52	40
5	19	15	6	8	31	335	6	15	18
6	1	5	1	4	9	3	1118	117	143
7	10	8	3	16	15	7	118	1070	480
8	12	5	1	11	8	13	157	378	1059

	precision	recall	f1-score	support
0	0.78	0.78	0.78	1060
1	0.82	0.75	0.78	1235
2	0.61	0.57	0.59	430
3	0.56	0.56	0.56	408
4	0.77	0.68	0.73	1153
5	0.76	0.74	0.75	453
6	0.77	0.80	0.79	1401
7	0.60	0.62	0.61	1727
8	0.58	0.64	0.61	1644
accuracy			0.69	9511
macro avg	0.70	0.68	0.69	9511
weighted avg	0.70	0.69	0.69	9511



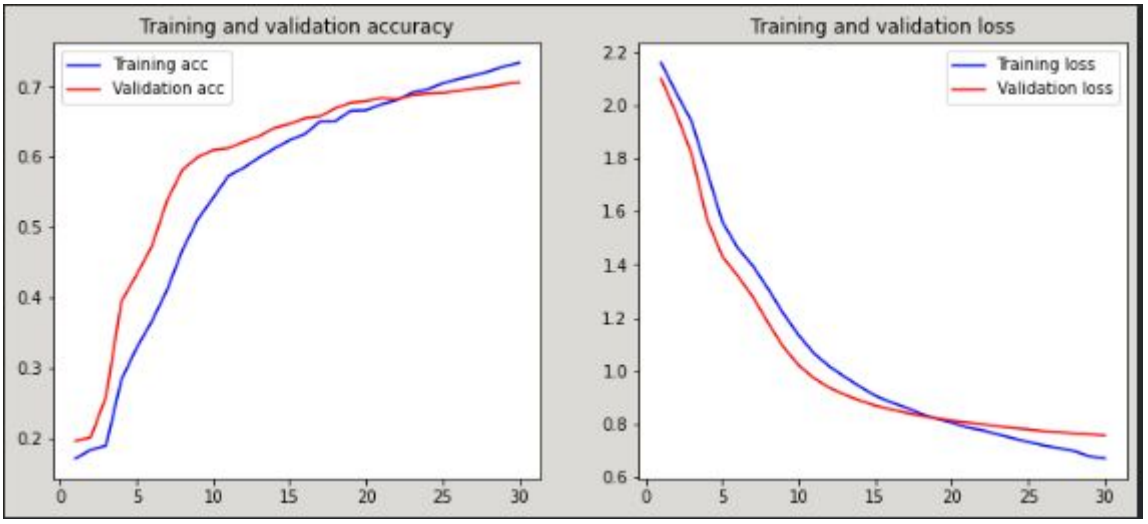
Best params: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

	0	1	2	3	4	5	6	7	8
0	804	54	13	12	90	4	23	44	16
1	16	1041	5	15	86	2	32	23	15
2	33	124	140	2	102	2	10	11	6
3	39	48	4	144	119	1	11	24	18
4	36	86	14	23	931	1	15	26	21
5	20	42	3	7	88	266	3	16	8
6	0	3	0	0	4	0	834	308	252
7	9	21	0	2	19	4	226	1054	392
8	2	25	0	1	12	2	202	662	738

	precision	recall	f1-score	support
0	0.84	0.76	0.80	1060
1	0.72	0.84	0.78	1235
2	0.78	0.33	0.46	430
3	0.70	0.35	0.47	408
4	0.64	0.81	0.72	1153
5	0.94	0.59	0.72	453
6	0.62	0.60	0.61	1401
7	0.49	0.61	0.54	1727
8	0.50	0.45	0.47	1644
accuracy			0.63	9511
macro avg	0.69	0.59	0.62	9511
weighted avg	0.64	0.63	0.62	9511

3.4. Reti neurali

Per l'attività di classificazione del testo con tecniche di deep learning, la scelta è ricaduta sull'adozione di: una DNN (*Deep Neural Network*) e di una LSTM.
Per l'architettura della DNN è composta da 4 layer di 512 neuroni ognuno. In questa forma base, si evidenzia fortemente il fenomeno dell'overfitting. A tal motivo si introduce dapprima il layer di *Dropout* e successivamente il *kernel regularized l2* nei nodi degli hidden layer in maniera tale da far attenuare il divario tra le curve di training e di validation.



	0	1	2	3	4	5	6	7	8
0	884	21	47	19	46	6	11	19	7
1	17	1068	47	16	58	3	11	14	1
2	27	52	268	7	68	3	0	5	0
3	55	39	32	163	89	6	5	13	6
4	23	49	43	23	985	11	2	16	1
5	19	11	30	6	44	332	1	9	1
6	0	1	0	3	2	4	1080	145	166
7	9	12	1	4	10	3	214	976	498
8	6	5	0	0	12	2	226	472	921

	precision	recall	f1-score	support
0	0.85	0.83	0.84	1060
1	0.85	0.86	0.86	1235
2	0.57	0.62	0.60	430
3	0.68	0.40	0.50	408
4	0.75	0.85	0.80	1153
5	0.90	0.73	0.81	453
6	0.70	0.77	0.73	1401
7	0.58	0.57	0.57	1727
8	0.58	0.56	0.57	1644
accuracy			0.70	9511
macro avg	0.72	0.69	0.70	9511
weighted avg	0.70	0.70	0.70	9511

Nonostante il valore di loss sia elevato, sulla classe 2 e 7 commetta diversi errori, il risultato è simile a quello degli altri classificatori. Purtroppo però, gli altri modelli riescono a raggiungere un livello di affidabilità (guardando recall e precision) più interessante.

La seconda tipologia di rete è la LSTM, una rete della famiglia RNN. La struttura prevede un layer *Embedding* che consente di trasformare i testi, sottoposti a pre-processing e tokenizzazione, da numeri interi in valori float più adatti agli strati sottostanti.

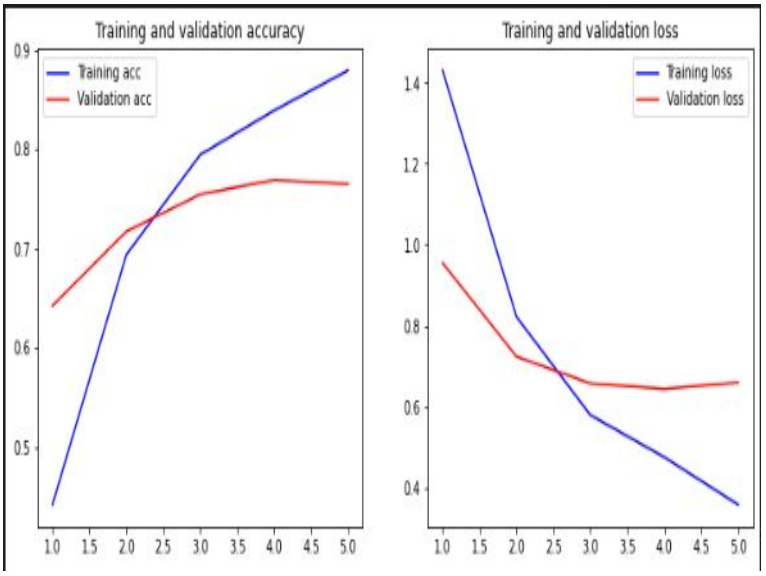
Poi un layer *LSTM* con 100 neuroni è utilizzato per analizzare il testo ed infine per la classificazione si ha un layer Dense con *softmax*. Tra il layer primo e secondo, è inserito un Dropout per ridurre il fenomeno dell'overfitting insieme al dropout ricorsivo del layer LSTM.

```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=x_train.shape[1]))
model.add(SpatialDropout1D(0.5))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(9, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

#history = model.fit(x_train, y_train, epochs=5, validation_data=(x_valid, y_valid), verbose=1, batch_size=64)
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 500, 100)          5000000
_________________________________________________________________
spatial_dropout1d_4 (Spatial (None, 500, 100)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_5 (Dense)              (None, 9)                 909
=================================================================
Total params: 5,081,309
Trainable params: 5,081,309
Non-trainable params: 0
```

Il modello è poi addestrato ma, come dai grafici sottostanti delle curve di loss e accuracy, si può evidenziare un certo livello di overfitting. Questo è principalmente dovuto al basso numero di esempi che sono consegnati alla rete per l'addestramento, in quanto, eseguendo diversi test con un numero maggiore di epoche non si notano differenze. Tuttavia, il risultato in termini di accuratezza ottenuto da quest'ultima rete neurale è accettabile anche se il livello di loss risulta alto. Di seguito le immagini delle curve e della confusion matrix.

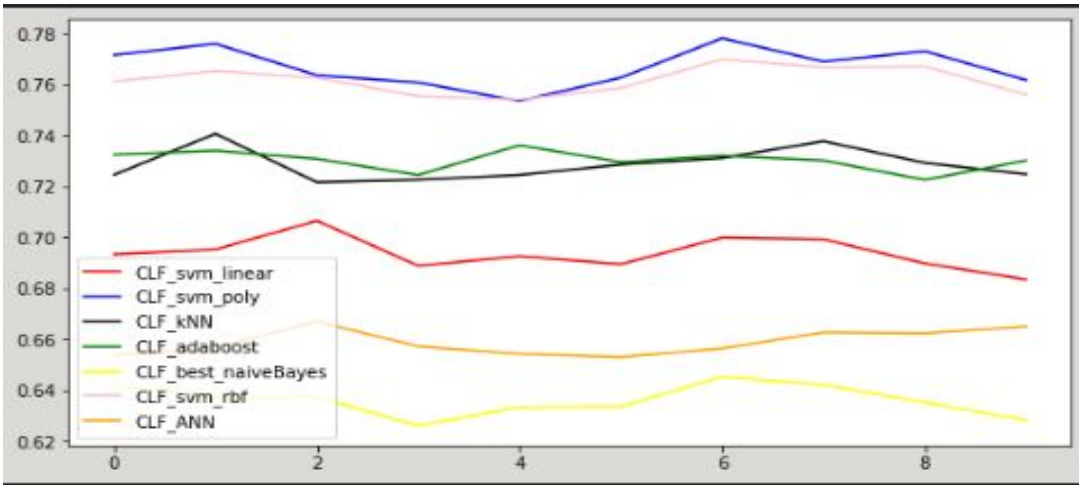


	0	1	2	3	4	5	6	7	8
0	882	21	45	55	30	2	3	16	6
1	5	1097	44	32	34	4	7	6	6
2	19	46	279	33	46	1	1	3	2
3	19	15	27	290	26	10	5	8	8
4	16	47	61	34	950	20	5	13	7
5	7	2	8	23	27	374	3	7	2
6	1	1	0	1	4	0	1190	77	127
7	1	4	1	4	1	1	121	952	642
8	4	5	0	1	6	0	180	274	1174

3.5. Cross Validation

Anche nel caso della classificazione dei testi, i risultati sono sottoposti a 10-fold cross validation e il risultato può essere analizzato nel grafico sottostante. Si noti come il risultato migliore lo si ottiene mediante SVM con kernel polinomiale (grado 2) ed rbf.

Tra i risultati non è stato possibile inserire la rete neurale con LSTM in quanto ogni step d'addestramento richiede all'incirca 35 minuti (condotto su solo 5 epoche) sulla piattaforma Google Colab e valutarlo per una 10 fold cross validation andrebbe a superare il limite messo a disposizione. Per mancanza hardware di scheda video non è stato possibile effettuare la simulazione la quale risulta temporalmente non adatta al processore della macchina usata.



4. Anomaly Detection

Il task di anomaly detection è l'ultima fase del progetto e consta nell'addestrare tre classificatori, due autoencoder e un OneClassSVM in maniera tale da renderli capaci di stabilire quale dei campioni sottoposti a predizione siano o meno della classe definita come "normale" e quali, invece, siano "anomalie".
L'approccio adottato è quello di una anomaly detection *semi-supervised*, ovvero sola una parte dei miei dati è con etichetta mentre la maggior parte risulta non avere informazioni circa la sua natura. Questa piccola parte di dati con etichetta (*labelled data*) non saranno utilizzati durante il training ma bensì durante la fase di valutazione delle predizioni di elemento *outlier* o meno.

I due autoencoder si differenziano per la loro architettura, in quanto, uno è una rete neurale densa mentre il secondo è convoluzionale. Le architetture specifiche verranno spiegate in seguito a seconda se ci si trovi davanti ad un rilevamento per immagini o testi, mentre ora si può dire che i due modelli sono basati sull'idea di ricevere il dato in input e tentare di ricostruirlo a partire dalla sua versione *latente*. In più, la funzione di *loss* utilizzata è una *mse*, ovvero lo scarto quadratico medio (*mean squared error*) mentre come ottimizzatore per il gradiente è utilizzato, come nel caso della classificazione, *Adam* ma con il valore di learning rate di default.

Il secondo modello per la scoperta di anomalie è il OneClassSVM della libreria scikit-learn e per quanto riguarda la scelta del modello si è partiti dal caso della classificazione. Infatti, il kernel scelto per l'adozione di tale modello è quello *radial basis function* (*rbf*) il quale, durante la classificazione, è stato il più promettente.

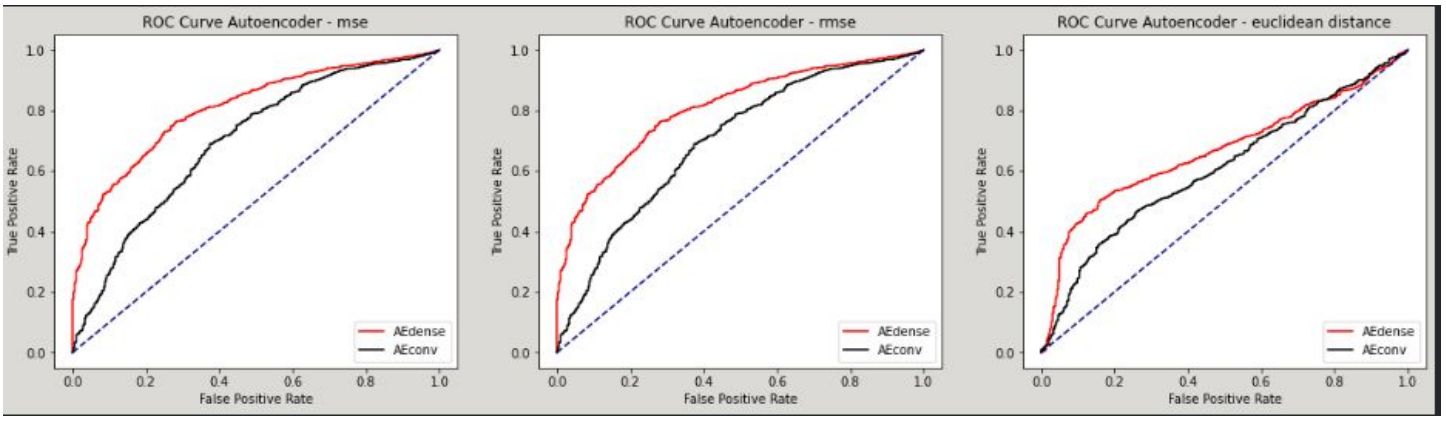
Come metro di valutazione si è scelta la curva ROC (*Receiver Operating Characteristic*) nella quale si confrontano le curve di predizione dei detector nei confronti del *Random Detector* che risulta nel diagramma dalla linea tratteggiata diagonale. Per leggere i risultati della curva, bisogna sapere che mette in rapporto il tasso di predizioni corrette ed errate facendo variare una soglia (*threshold*); in termina di andamento della curva, questa risulta essere più interessante quanto più adiacente all'angolo in alto a sinistra del grafico, ove si raggiunge la curva del detector perfetto.

4.1. Confronto OneClassSVM e Autoencoder per immagini

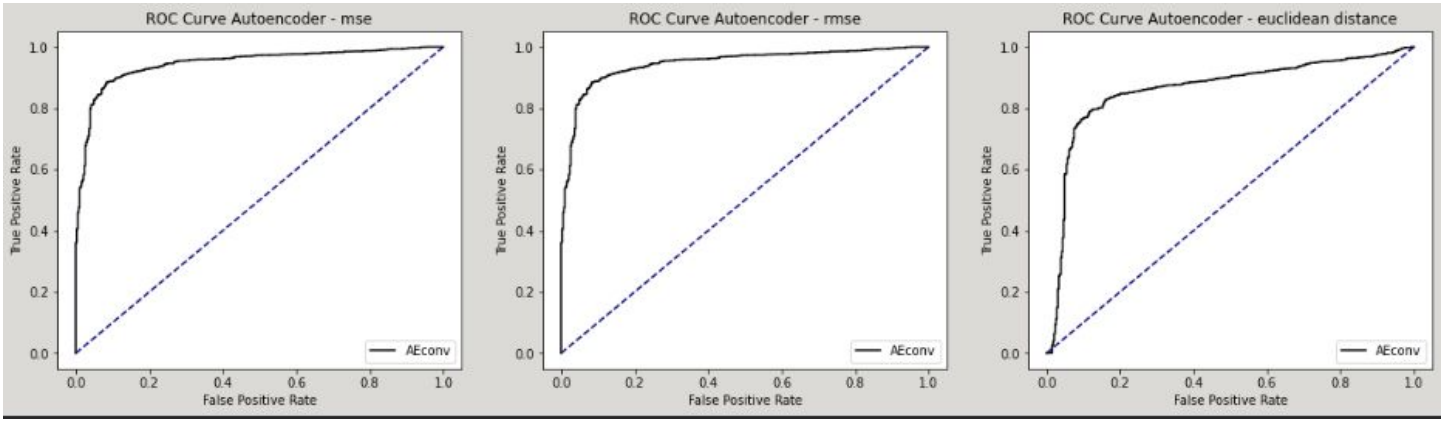
In questo paragrafo si descrive le scelte per ottenere i due modelli di autoencoder e li si confrontano con il caso in cui come detector è usato il OneClassSVM.

Analizzando l'autoencoder convoluzionale possiamo dire che, oltre ad ottimizzatore Adam e loss function "mse", la struttura è ricreata mediante l'utilizzo di 3 tipologie di layer: *Conv2D*, *MaxPooling2D* e *UpSampling2D*. Quello convoluzionale è usato per estrarre delle feature dalle immagini originali e per ricostruire da queste stesse l'immagine in fase di *decoding*; il MaxPooling e l'UpSampling, invece, consentono la restrizione e l'ingrandimento delle immagini durante le due fasi di *encoding* e *decoding*.

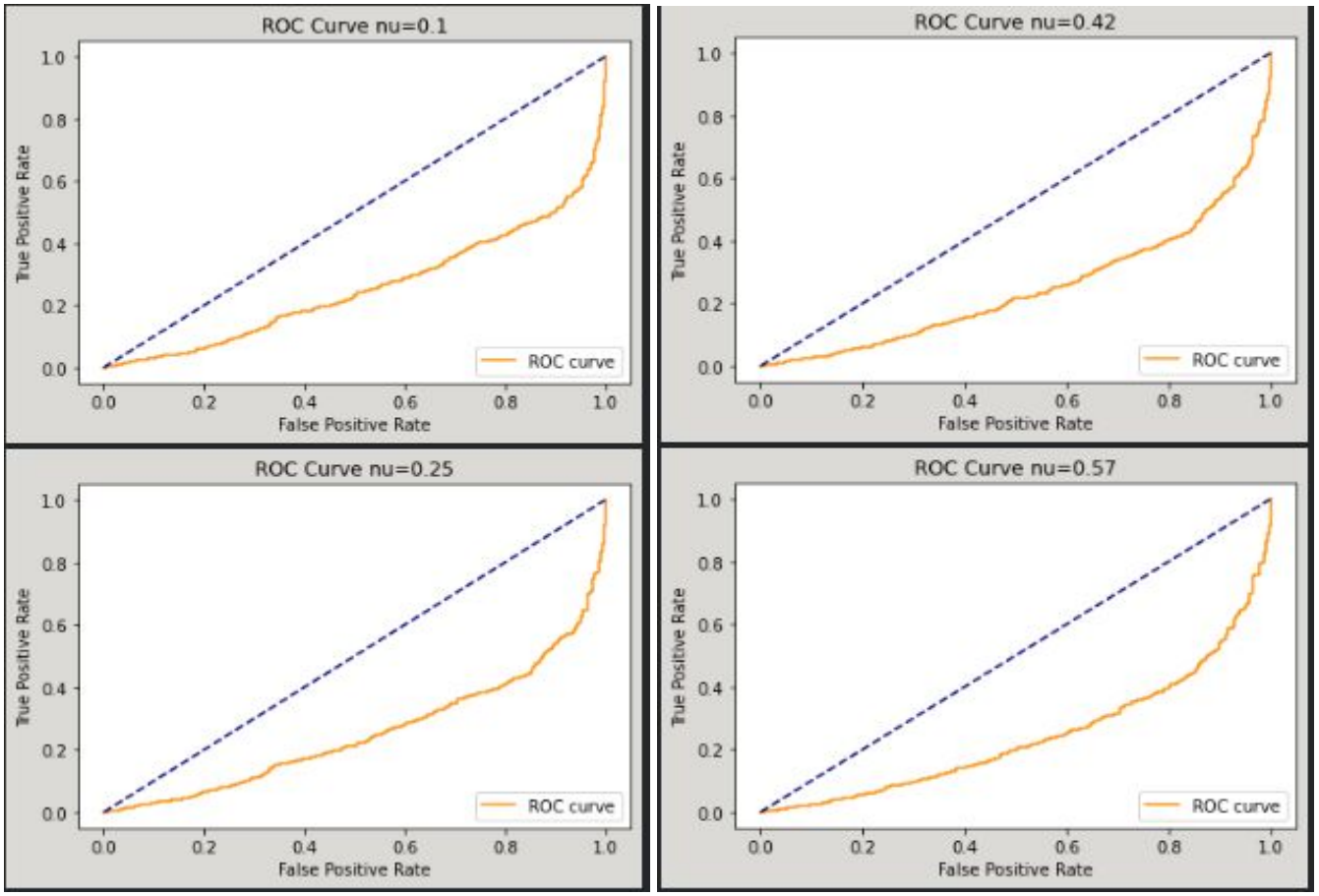
Nel caso dell'autoencoder completamente denso è adottata una struttura a 6 layer (escluso quello di input) di tipo Dense, quindi con collegamenti da ogni neurone ad ogni altro del livello successivo.



Come si evince dalle curve, entrambi risultano discernere in maniera sufficientemente buona le istanze normali e anomale. In realtà le curve sono leggermente alterate dalla presenza delle due classi molto simili di "scarpe" (*capitolo 1 per riferimento alle classi*) che degradano le reali prestazioni della rete.
Infatti, una ulteriore analisi dell'autoencoder convoluzionale eseguita su un dataset contenente solo immagini relative alle classi 0,1 e 3 (*riferimento a capitolo 1 per l'interpretazione di classi e immagini*) in cui le immagini di classe 1 sono le normali e le altre definite anomalie, si ottiene una curva che risulta essere ancora più spigolata nell'angolo sinistro e quindi sinonimo di buon rilevamento.



Infine, l'applicazione del modello di OneClassSVM delude le aspettative in quanto, oltre ad essere più lento nell'analisi rispetto l'autoencoder, riporta anche una curva ROC di bassa qualità anche rispetto la risposta random.



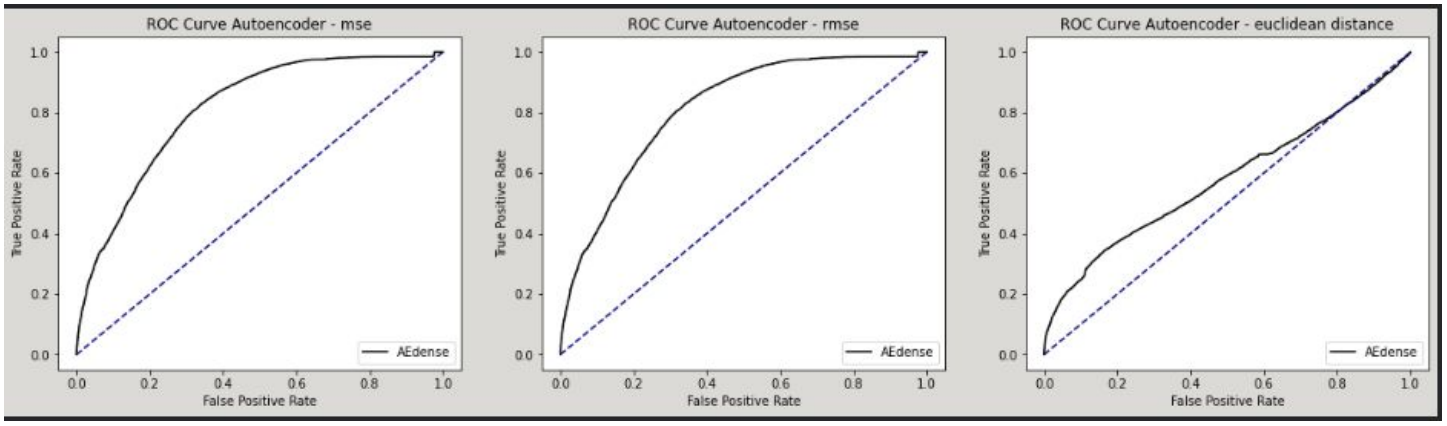
4.2. Confronto OneClassSVM e Autoencoder per testi

La fase di anomaly detection per i testi è svolta mediante un Autoencoder *Fully-Connected* e un modello di OneClassSVM. In questo secondo caso, la classe definita normale è rappresentata da tutti i file rientranti nella categoria *"hams"* (si veda capitolo 1 per collegare testi e classi) mentre la classe di cui le anomalie fanno parte è quella *"spam"*. Si tratta quindi di un problema di distinzioni tra quali email sono ritenute spam e quali invece sono interessanti.

Il modello di autoencoder è un'architettura di rete da 5 livelli, inclusi input ed output. Tutti i layer sono di tipo *Dense* e la configurazione più adatta a massimizzare il rilevamento di eventuali outliers è quella seguente:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 400)               400400
_________________________________________________________________
dense_2 (Dense)              (None, 150)               60150
_________________________________________________________________
dense_3 (Dense)              (None, 400)               60400
_________________________________________________________________
dense_4 (Dense)              (None, 1000)              401000
=================================================================
Total params: 921,950
Trainable params: 921,950
Non-trainable params: 0
```

Per approfondire inoltre l'apprendimento e renderlo più affidabile si è deciso di abbassare fino a 10^{-5} il learning rate dell'ottimizzatore Adam di aggiornamento del gradiente. Come funzione di loss si è scelta la *mse* per andare a verificare l'errore di ricostruzione del dato in input.
Il risultato ottenuto dall'Autoencoder nel riconoscere *"hams and spams"* è abbastanza soddisfacente come si può osservare dalla curva ROC sottostante che si avvicina a quella della curva perfetta.



La seconda versione di anomaly detector è quella della OneClassSVM che però ottiene dei risultati, come nel caso delle immagini, molto insufficienti essendo al di sotto della curva del rilevatore random. Anche facendo variare l'iper-parametro *nu* (che rappresenta il rapporto tra classe normale e classe delle anomalie) la situazione rimane immutata, a meno di piccolissimi cambiamenti puntuali della curva.