Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

VALUTAZIONE DI UN SAFETY WRAPPER PER MACHINE LEARNERS NEL CONTESTO DELLA CLASSIFICAZIONE DI IMMAGINI

EVALUATING A SAFETY WRAPPER FOR MACHINE LEARNERS IN IMAGE CLASSIFICATION DOMAIN

LUIGI CENNINI

Relatore: *Tommaso Zoppi*

Anno Accademico 2021-2022

Luigi Cennini: *Valutazione di un Safety Wrapper per Machine Learners nel contesto della classificazione di immagini,* Corso di Laurea in Informatica, © Anno Accademico 2021-2022

INDICE

ELENCO DELLE FIGURE

1

INTRODUZIONE

La **classificazione di immagini** è una delle attività più importanti nell'ambito dell'intelligenza artificiale. Essa consiste nel processare un'immagine digitale e assegnarla ad una o più categorie prestabilite, in base alle caratteristiche e alle informazioni che l'immagine contiene [1].
Questa disciplina è in continua evoluzione, grazie alla sempre maggiore disponibilità di dati, hardware di supporto e alla continua ricerca di nuovi algoritmi e metodologie più efficaci ed efficienti.
Il suo sviluppo è in rapida crescita anche perché può essere applicata in diversi contesti, come ad esempio nella sorveglianza del territorio, nell'industria automobilistica, nell'analisi medica, nella rilevazione di oggetti in movimento, e in molti altri campi.
Le tecniche utilizzate per la classificazione di immagini sono basate su algoritmi di apprendimento automatico (machine learning) ed in particolare di deep learning, che permettono di identificare pattern e relazioni complesse tra i pixel dell'immagine e le categorie di appartenenza.
Il problema delle reti neurali utilizzate per svolgere questo compito è che occasionalmente possono commettere errori e questo non può essere accettabile nei sistemi critici, il cui fallimento può avere severe conseguenze per persone, ambiente ed infrastrutture [2].
Gli studi in passato si sono concentrati soprattutto su minimizzare gli errori, che però non possono essere eliminati. Negli ultimi 5 anni invece sono nati studi per cercare di capire perché e quando si verificano gli errori, in modo da prevederli. In questo contesto, i **safety wrapper** sono stati sviluppati per prevenire gli errori commessi dai classificatori, se il wrapper sospetta un errore blocca l'output del classificatore. In questo modo, invece di avere a volte un output giusto o sbagliato, si ha o un output giusto o nessun output. La Tesi presentata utilizza il transfer learning su reti neurali pre-addestrate per la classificazione di immagini e testa l'utilizzo del safety wrapper **SPROUT** (un wrapper di classificatori sviluppato dall'Università di Firenze) su questi modelli per predire gli

errori del classificatore e bloccare l'output di classificatori poco sicuri. In particolare nella Tesi si sono testate le reti neurali per immagini più popolari sui tre dataset per immagini più diffusi (MNIST, FASHION_MNIST e CIFAR10) utilizzando la tecnica del transfer learning. Alle tre reti più performanti è stato applicato il safety wrapper SPROUT. In questo modo è stato possibile confrontare i risultati ottenuti con e senza SPROUT.

La Tesi è strutturata come segue:

- il capitolo 2 fornisce una panoramica delle fondamenta teoriche delle reti neurali.
- nel capitolo 3 si analizzano le reti e dataset per immagini.
- il capitolo 4 spiega il concetto di transfer learning. Applicando questa tecnica sui dataset usati vengono individuate le migliori reti per immagini.
- nel capitolo 5 si applica il safety wrapper SPROUT sulle migliori reti trovate nel capitolo precedente.
- il capitolo 6 conclude la Tesi.

2

BASI DI RETI NEURALI

Il machine learning è un campo dell'informatica che si occupa di sviluppare algoritmi e modelli che permettono alle macchine di apprendere dai dati e di migliorare le loro prestazioni in modo autonomo. Si basa sulla capacità delle macchine di riconoscere pattern e relazioni nei dati, e di adattarsi a nuove situazioni e dati non visti in precedenza.
Il termine "machine learning" è stato coniato nel 1959 dallo scienziato **Arthur Samuel**, che lo ha definito come *l'abilità delle macchine di apprendere senza essere esplicitamente programmate*. Samuel stava cercando di applicare questi concetti al gioco della dama [3].
L'idea di creare macchine in grado di apprendere risale al 1958, quando lo psicologo **Frank Rosenblatt** ha creato il primo **percettrone**, l'elemento di base per creare una rete neurale artificiale, ispirato al funzionamento del cervello umano [4].
Il percettrone di Rosenblatt è stato il primo passo per creare una rete neurale, ovvero una struttura composta da nodi interconnessi. Tuttavia, fino a pochi anni fa non si sapeva come addestrare ed eseguire queste reti per ottenere risultati migliori rispetto agli approcci tradizionali, in particolare a causa del supporto hardware non adeguato.
Negli ultimi anni invece il machine learning è diventato sempre più utilizzato grazie all'aumento della disponibilità di dati [5], della potenza di calcolo delle macchine [6] e dei progressi nelle tecniche di apprendimento automatico. Con la diffusione di internet, dei social media e dei dispositivi connessi, è diventato sempre più facile raccogliere grandi quantità di dati da una vasta gamma di fonti. Ciò ha reso possibile l'addestramento di modelli di machine learning su insiemi di dati molto più grandi e diversi, aumentando la loro precisione e capacità di generalizzazione. Inoltre, la crescente potenza di calcolo delle macchine ha reso possibile l'elaborazione di grandi quantità di dati in modo molto più rapido e efficiente, consentendo di addestrare modelli di machine learning più complessi in meno tempo. Infine, ci sono stati importanti progressi nelle

tecniche di apprendimento automatico, come il **deep learning**, che ha permesso di creare modelli più precisi e sofisticati per una vasta gamma di applicazioni, tra cui l'elaborazione del linguaggio naturale, la robotica, il riconoscimento della voce e delle immagini.

Il deep learning è una tecnica di apprendimento automatico basata su reti neurali, composte da molteplici strati di nodi interconnessi. Queste reti possono apprendere da enormi quantità di dati, riconoscere pattern complessi e produrre risultati di alta qualità.
Oggi, grandi compagnie come Google [7], Tesla [8], Amazon [9] e Meta [10] utilizzano il machine learning per migliorare la loro produttività e creare prodotti innovativi. Inoltre, ci sono anche molte startup e imprese di piccole e medie dimensioni che utilizzano il machine learning per creare soluzioni personalizzate e innovare nei loro settori di attività.
In sostanza, il machine learning è una tecnologia in costante evoluzione, che offre numerose opportunità per migliorare le prestazioni delle macchine e aprire nuove possibilità in molte industrie. Grazie alle tecniche di apprendimento automatico, le macchine possono apprendere e migliorare le loro prestazioni in modo autonomo, aprendo nuovi scenari per l'innovazione tecnologica.

2.1 PERCETTRONE

Il modello di percettrone proposto da Rosenblatt nel 1958 fu poi rifinito da Minsky and Papert nel 1969 [11]. Un percettrone prende un numero finito di input booleani $x_1, x_2, ..., x_n$ e grazie a dei pesi reali $w_1, w_2, ..., w_n$ resitituisce un risultato booleano y .

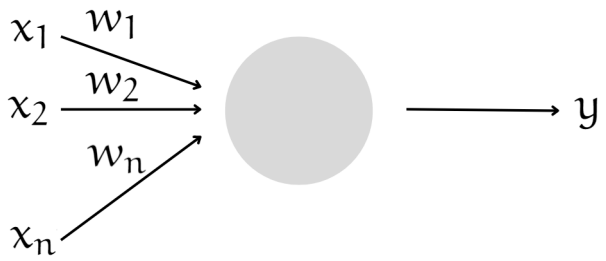


Figura 1.: percettrone

I pesi $w_1, w_2, ..., w_n$ descrivono l'importanza che ciascun input ha sul determinare l'output. La regola per calcolare l'output è la seguente:

$$y = \begin{cases} 0 \text{ se } \sum_i x_i w_i \leqslant \mathtt{threshold} \\ 1 \text{ se } \sum_i x_i w_i > \mathtt{threshold} \end{cases}$$

Dove *threshold* è un parametro del percettrone. Un modo più compatto e comune per descrivere il comportamento di un percettrone è il seguente:

$$y = \begin{cases} 0 \text{ se } \vec{w}\vec{x} + b \leqslant 0 \\ 1 \text{ se } \vec{w}\vec{x} + b > 0 \end{cases}$$

Dove b è chiamato bias ($b \equiv -\mathtt{threshold}$).
Dunque in pratica quello che un percettrone fa è pesare delle evidenze per predere una decisione. Ovviamente un singolo percettrone non basta per simulare il cervello o l'intelligenza umana. Per prendere decisioni più sofisticate è possibile costruire reti di percettroni, mettendo insieme più percettroni e raggruppandoli in strati (**layer**).

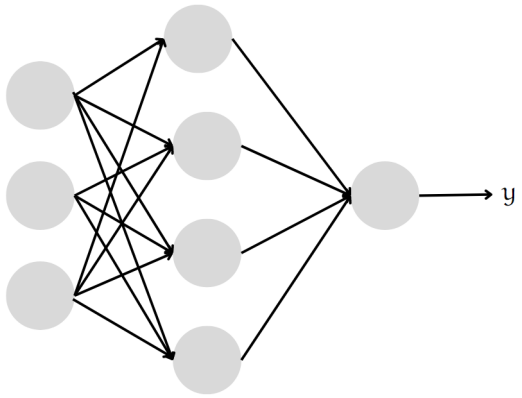


Figura 2.: rete di percettroni

Nell'immagine, il primo strato (chiamato input layer) è formato da 3 percettroni ed è quello che prende decisioni più semplici basandosi solo sull'input dato.
Il secondo strato (chiamato hidden layer) sarà invece quello che prende decisioni basate sulle decisioni prese dal primo layer. Quello in figura è anche chiamato layer **denso** o **completamente connesso**. L'ultimo strato della rete prende invece nome di output layer ed è quello che restituisce la risposta binaria $y \in [0, 1]$.
Dunque non è difficile immaginare che aumentando o diminuendo layer e neuroni è possibile ottenere reti più o meno complesse e sofisticate.

Il problema di una rete di percettroni è che cambiando anche di poco i bias è possibile che il comportamento della rete cambi drasticamente (es: che l'output cambi da 0 a 1 o da 1 a 0).

2.2 NEURONI ARTIFICIALI E RETI NEURALI

Per risolvere il problema dei percettroni si introduce un modello più complesso dove gli input e l'output non sono più booleani ma bensì numeri reali. Inoltre, all'output viene applicata una funzione detta funzione di attivazione. Un neurone artificiale può dunque essere rappresentato in questo modo:

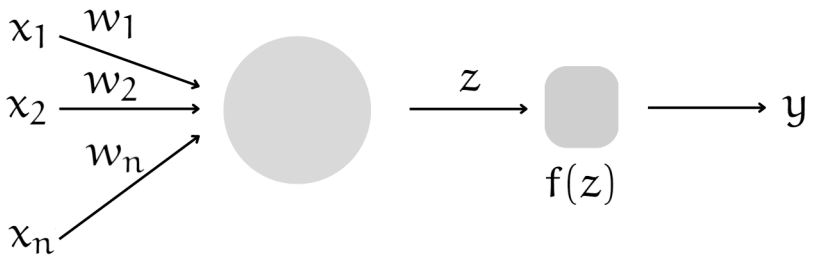


Figura 3.: neurone artificiale

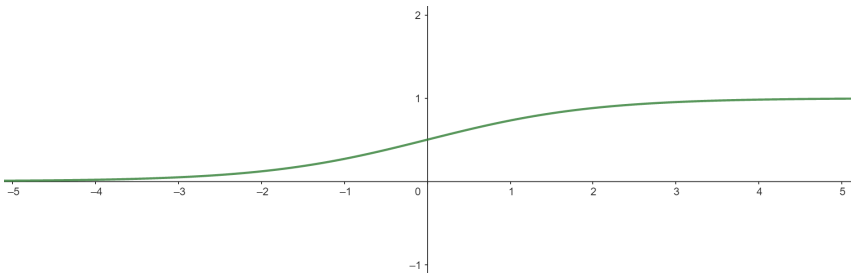
Nel caso dei neuroni artificiali, l'output della rete può essere descritto in questo modo:

$$y = f(\vec{w}\vec{x} + b)$$

Disponendo questi neuroni in strati come per i percettroni otteniamo quella che viene chiamata una **rete neurale**.
Le funzioni di attivazione possono essere diverse e devono essere scelte in base al tipo di problema che la rete deve risolvere [12]. Ad esempio eccone alcune elencate:

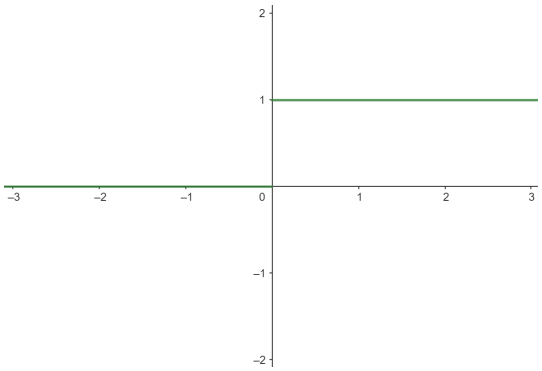
1. sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



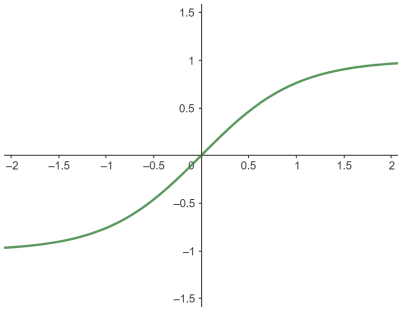
2. step o funzione a gradino

$$f(x) = \begin{cases} 0 \text{ se } x < 0 \\ 1 \text{ altrimenti} \end{cases}$$



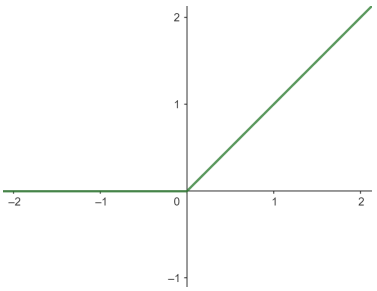
3. tangente iperbolica

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



4. ReLu (Rectified linear unit)

$$f(x) = \begin{cases} 0 \text{ se } x \leqslant 0 \\ x \text{ altrimenti} \end{cases}$$



Per esempio scegliendo come funzione di attivazione la sigmoide avremo che l'output di un singolo neurone sarà calcolato nel modo seguente:

$$y = \sigma(\vec{w}\vec{x} + b) = \frac{1}{1 + e^{-(\vec{w}\vec{x}+b)}}$$

Grazie dunque ai neuroni artificiali si risolve il problema dei percettroni; un piccolo cambiamento di pesi e bias provocherà adesso solo un piccolo cambiamento nell'output.

2.3 ADDESTRAMENTO DI RETI NEURALI

Avendo definito la struttura generale delle reti, quello di cui avremmo bisogno è un algoritmo che ci permette di trovare pesi e bias che minimizzano l'errore tra l'output della rete e l'output desiderato.
Ci sono 3 modi che è possibile utilizzare per fare apprendere gli algoritmi: [13] :

1. **apprendimento supervisionato** è una tecnica in cui un algoritmo viene addestrato su un insieme di dati etichettati.

2. **apprendimento non supervisionato** è una tecnica in cui una rete viene addestrata su un insieme di dati **non** etichettati. L'algoritmo esplora i dati per cercare di identificare eventuali regolarità o cluster di punti che possano aiutare a creare una rappresentazione più compatta e significativa del dataset.

3. **apprendimento per rinforzo** è una tecnica in cui un algoritmo apprende a compiere azioni ricevendo feedback in forma di ricompense o penalità a seconda delle azioni che compie, con l'obiettivo di massimizzare la ricompensa totale ottenuta.

Per lo scopo di questa Tesi tratteremo solo l'apprendimento supervisionato di reti neurali.

Per minimizzare l'errore della rete viene definita una **funzione di costo** (o **loss**) che viene calcolata in fase di addestramento quando l'input è passato dalla rete.
La funzione di costo indica quanto la rete sta sbagliando la predizione. Dunque addestrare la rete si traduce formalmente nel minimizzare questa funzione.
L'algoritmo più usato per fare questo è la **backpropagation**.

2.3.1 *backpropagation*

La backpropagation è un algoritmo utilizzato per addestrare reti neurali artificiali supervisionate. Consiste nell'aggiornare i pesi delle connessioni tra i neuroni in modo da minimizzare l'errore tra l'output della rete e l'output desiderato.
La backpropagation funziona in due fasi:

1. **forward propagation** dove l'input viene processato attraverso la rete.

2. **backward propagation** dove l'errore viene propagato all'indietro attraverso la rete per aggiornare i pesi.

Nella fase di **forward propagation** l'input viene passato attraverso la rete e ogni neurone calcola il suo output passandolo poi ai neuroni successivi. L'output finale della rete viene confrontato con l'output desiderato per calcolare l'errore.
Nella fase di **backward propagation**, l'errore viene propagato all'indietro attraverso la rete per calcolare la derivata dell'errore rispetto ai pesi delle connessioni. Questi valori vengono utilizzati per aggiornare i pesi delle connessioni in modo da minimizzare l'errore.
Questo processo viene ripetuto fino a quando l'errore non raggiunge una soglia di tolleranza o per un certo numero di epoche.

3

RETI PER IMMAGINI

Le reti neurali per immagini utilizzano una struttura di neuroni artificiali per analizzare e classificare le immagini. Esse sono composte da diverse strati di neuroni, ognuno dei quali elabora le informazioni dell'immagine in modo diverso. Questi strati lavorano insieme per identificare i pattern distintivi dell'immagine e classificarla in base alla sua somiglianza con le immagini di addestramento. Le reti neurali per immagini sono diventate molto popolari negli ultimi anni grazie ai loro eccellenti risultati nella classificazione di immagini in diversi settori, come la visione artificiale, la medicina [14] e l'industria automobilistica [15].
Per il momento, consideriamo la rete neurale come una "scatola nera", senza entrare nel dettaglio del funzionamento interno. Nella prossima sezione della Tesi, invece, verranno approfondite le reti neurali specificamente utilizzate per analizzare e classificare le immagini.

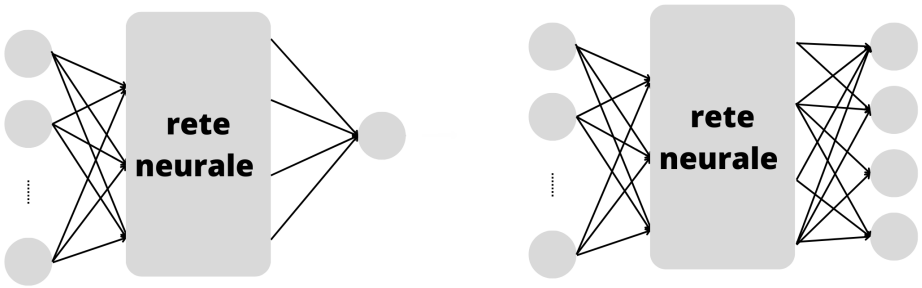


Figura 4.: a sinistra, classificazione binaria; a destra, classificazione multiclasse

Nell'immagine, a sinistra, possiamo vedere ad alto livello una rete neurale per la **classificazione binaria** delle immagini (con solo un neurone nel layer di output). Ovvero utilizzata per classificare le immagini in due categorie.
A destra invece, una rete neurale per la **classificazione multiclasse**. Ogni neurone nel layer di output rappresenta una classe del dominio del problema di classificazione. Nel caso specifico della figura, ci saranno 4 classi di immagini da classificare.

In ogni caso, l'output dei neuroni nel layer di output è la distribuzione di probabilità che l'input dato alla rete appartenga a quel neurone, cioè che rappresenti una certa classe.

3.1 RETI CONVOLUZIONALI

Adesso è necessario entrare nel dettaglio e spiegare la rete neurale che nella sezione precedente è stata tattata come una "scatola nera".
Le reti neurali più performanti per la classificazione di immagini sono generalmente le reti convoluzionali (CNN, Convolutional Neural Networks) [16], grazie alla loro capacità di estrarre automaticamente le caratteristiche salienti delle immagini. Le reti convoluzionali sono caratterizzate da strati di convoluzione, pooling e completamente connessi che le rendono in grado di catturare la struttura spaziale delle immagini e di costruire rappresentazioni sempre più complesse a partire dai dati di input. Tra le reti convoluzionali più note, ci sono ad esempio la VGG, la ResNet, la Inception (o GoogLeNet) e la MobileNet, che si differenziano per il numero di strati, la complessità e le prestazioni [25].

3.1.1 *convoluzione*

La convoluzione è una operazione matematica utilizzata nella teoria dei segnali [17] e che può essere applicata anche nel campo dell'intelligenza artificiale. Consiste nell'applicare un filtro ad una porzione dell'immagine di input, generando una mappa di attivazione che evidenzia la presenza di particolari pattern locali. Il filtro, chiamato anche kernel, scansiona l'immagine in modo sistematico, moltiplicando i valori dei pixel dell'immagine con i pesi del filtro e sommando i risultati, generando un valore di attivazione per ogni posizione. I layer convoluzionali sono costituiti da una serie di filtri che estraggono progressivamente caratteristiche sempre più complesse dall'immagine di input, generando un insieme di mappe di attivazione che costituiscono l'output del layer. Ogni filtro viene applicato durante l'addestramento del modello, in modo da massimizzare l'accuratezza della classificazione. Grazie all'uso di strati convoluzionali, le reti neurali sono in grado di rilevare automaticamente le caratteristiche salienti delle immagini, senza la necessità di definirle manualmente.

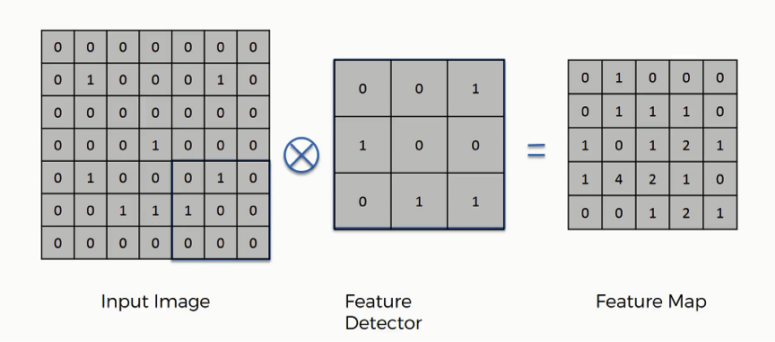


Figura 5.: operazione di convoluzione, fonte superdatascience.com

3.1.2 *pooling*

L'operazione di pooling consiste nell'aggregare le informazioni di un'area dell'immagine di input, riducendone la dimensione e mantenendo solo le informazioni più rilevanti. I layer di pooling sono costituiti da una serie di pool di diversi tipi (max-pooling, average-pooling, ecc.), che applicano l'operazione di pooling all'immagine di input in modo sistematico, suddividendola in regioni sovrapposte e mantenendo solo la caratteristica più importante di ciascuna regione.

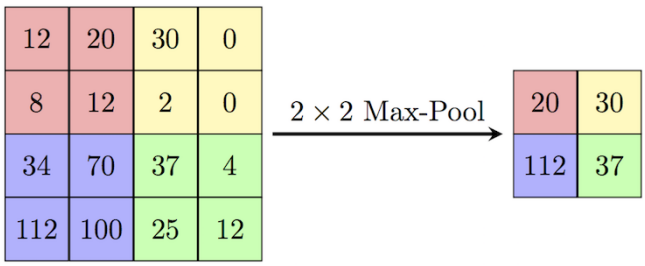


Figura 6.: operazione di max pooling 2x2, fonte computersciencewiki.org

3.2 VALUTAZIONE DI RETI PER IMMAGINI

Per valutare e quantificare le prestazioni di queste reti neurali, è necessario utilizzare alcune metriche.
La metrica più comune utilizzata per valutare la performance di una rete neurale per immagini è l'**accuracy**, che rappresenta la percentuale di immagini correttamente classificate rispetto al totale delle immagini valutate. Tuttavia, l'accuracy potrebbe non essere sufficiente per valutare la performance di una rete neurale in modo completo, specialmente se il dataset ha classi sbilanciate [18].

Altre metriche comunemente utilizzate per la classificazione binaria includono la **precision** e la **recall**. La precision rappresenta la percentuale di immagini correttamente classificate come appartenenti a una determinata classe rispetto al totale delle immagini classificate come tale classe. La recall rappresenta la percentuale di immagini appartenenti a una determinata classe che sono state correttamente classificate come tale rispetto al totale delle immagini appartenenti a quella classe.
Inoltre, sempre per la classificazione binaria, altre metriche utilizzate includono l'F1-score, che combina precision e recall per fornire una singola misura della performance, e il e Matthews correlation coefficient (MCC) [19].
Visualmente è possibile vedere come performa una rete neurale grazie alla **matrice di confusione** ovvero una tabella 2x2 (o nxn, nel caso di classificazione multiclasse) utilizzata per valutare le prestazioni di un modello di classificazione binaria o multiclasse [20].
In una matrice di confusione, le righe rappresentano le classi reali e le colonne rappresentano le classi predette dal modello.

3.3 DATASET

Un dataset è un insieme di dati strutturati che rappresentano informazioni su un determinato argomento. Per gli scopi di questa Tesi utilizzeremo 3 dataset di immagini per testare le reti neurali. È importante notare che tutti i dataset hanno 10 classi, dunque le reti neurali che costruiremo per classificarli saranno reti per la classificazione multiclasse con 10 neuroni nel layer di output.

1. **MNIST**

   Il dataset MNIST [21] è uno dei dataset più popolari e utilizzati nella comunità del machine learning per la classificazione di immagini. Esso contiene un insieme di immagini di cifre scritte a mano, ciascuna rappresentata da una matrice di 28x28 pixel in scala di grigi. Il dataset è composto da 60.000 immagini per l'insieme di addestramento e 10.000 per l'insieme di test. MNIST è stato introdotto nel 1998 come una versione modificata del database NIST originale ed è stato utilizzato per testare l'efficacia di diverse tecniche di riconoscimento di cifre scritte a mano, tra cui le reti neurali. Grazie alla sua semplicità e alla sua ampia disponibilità, il dataset MNIST è diventato uno standard di riferimento per valutare le prestazioni

di algoritmi di apprendimento automatico per la classificazione di immagini.

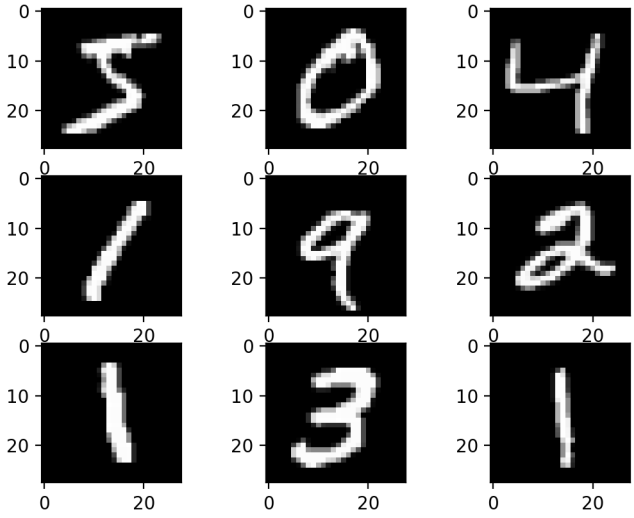


Figura 7.: sottoinsieme del dataset MNIST

2. **FASHION_MNIST**

   Il dataset FASHION_MNIST [22] è un dataset di immagini che è stato introdotto nel 2017 come alternativa a MNIST. È stato creato per fornire una sfida più impegnativa rispetto a MNIST (ritenuto facile da classificare) ed è composto da un insieme di immagini di capi d'abbigliamento appartenenti a 10 diverse categorie, ciascuna rappresentata da una matrice di 28x28 pixel in scala di grigi. Il dataset contiene 60.000 immagini per l'insieme di addestramento e 10.000 per l'insieme di test. L'obiettivo di FASHION_MNIST è quello di valutare le prestazioni degli algoritmi di apprendimento automatico per la classificazione di immagini di capi d'abbigliamento, come ad esempio t-shirt, pantaloni, camicie, abiti e scarpe.

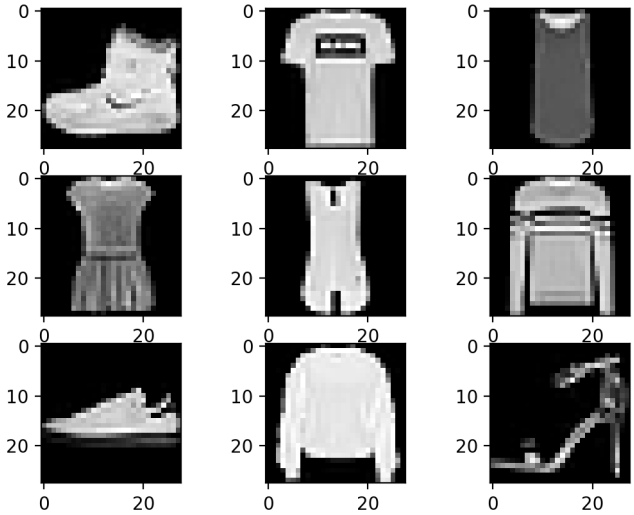


Figura 8.: sottoinsieme del dataset FASHION_MNIST

3. **CIFAR10**

   Il dataset CIFAR-10 [23] è un dataset di immagini che consiste in 60.000 immagini a colori di dimensione 32x32, divise in 10 classi diverse. Le classi includono oggetti comuni come aerei, automobili, uccelli, gatti, cervi, cani, rane, cavalli, navi e camion. Il dataset è diviso in un insieme di addestramento di 50.000 immagini e uno di test di 10.000 immagini, ed è stato uno dei primi dataset a colori ad essere ampiamente utilizzato nella comunità di visione artificiale. Grazie alla sua difficoltà maggiore rispetto a MNIST e FASHION_MNIST, CIFAR-10 è diventato uno dei dataset di riferimento per valutare l'efficacia di algoritmi di apprendimento automatico più avanzati per la classificazione di immagini.

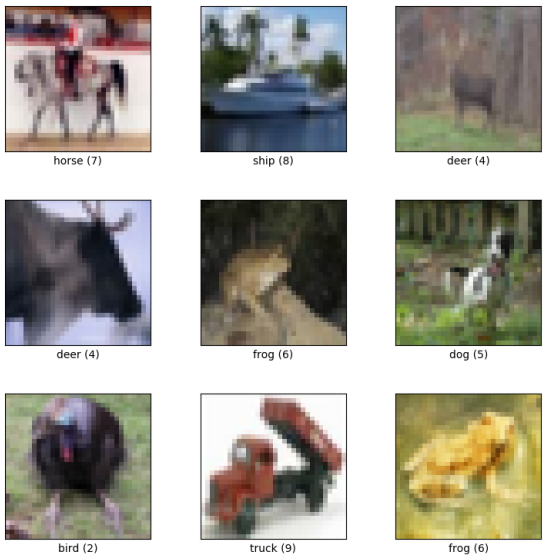


Figura 9.: sottoinsieme del dataset CIFAR10

3.4 RETE CONVOLUZIONALE CON PYTHON

È possibile utilizzare *python*[1] e la libreria *Keras*[2] per costruire una rete neurale convoluzionale per il riconoscimento delle immagini dei dataset menzionati in precedenza.
Nel codice sotto la costruzione e l'addestramento di una rete convoluzionale semplice per il dataset MNIST.

1 Python, `https://www.python.org/`
2 Keras, `https://keras.io/`

```
1  #load MNIST dataset
2  (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
3
4  #reshape x_train and x_test
5  x_train = x_train.reshape(60000, 28, 28, 1) / 255
6  x_test = x_test.reshape(10000, 28, 28, 1) / 255
7
8  #one hot encoding of y_train and y_test
9  y_train = to_categorical(y_train)
10 y_test = to_categorical(y_test)
11
12 #create convolutional neural network
13 model = Sequential()
14 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
15 model.add(MaxPooling2D(2, 2))
16 model.add(Conv2D(64, (3, 3), activation='relu'))
17 model.add(MaxPooling2D(2, 2))
18 model.add(Flatten())
19 model.add(Dense(128, activation='relu'))
20 model.add(Dense(10, activation='softmax'))
21 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
       accuracy'])
22
23 #train the model
24 model.fit(x_train, y_train, validation_split=0.2, epochs=20, batch_size=
       60, callbacks=[EarlyStopping(patience=4, restore_best_weights=True)])
25
26 accuracy=model.evaluate(x=x_test,y=y_test)[1]
```

La rete convoluzionale costruita ha una struttura semplice: 2 layer convoluzionali, 2 di maxpooling e 2 completamente connessi.

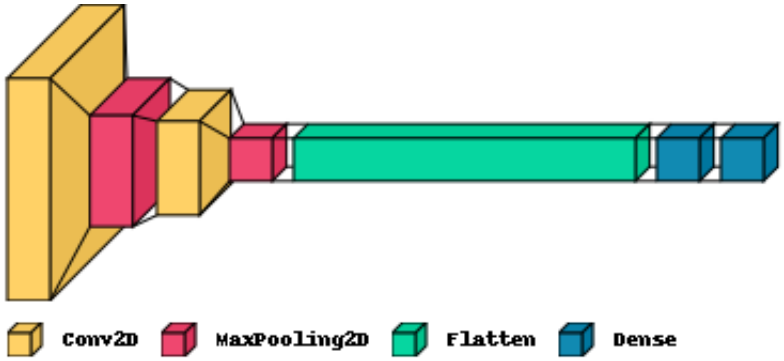


Figura 10.: rete convoluzionale semplice

Alla riga 24 del codice viene chiamata la funzione fit(...) [3] che addestra

3 https://keras.io/api/models/model_training_apis/

la rete sul dataset per 20 epoche.
Alla riga 29 viene calcolata l'accuracy del modello sul dataset usando l'insieme di dati di test.

Ripetendo questo codice e variando il dataset per FASHION_MNIST e CIFAR10, giungiamo ai seguenti risultati di accuracy:

	MNIST	**FASHION_MNIST**	**CIFAR10**
accuracy	0.9894	0.9085	0.7016

Tabella 1.: Accuracy rete CNN semplice sui 3 dataset MNIST, FASHION_MNIST e CIFAR10.

Si può vedere che con questo approccio si ha un accuracy del 98% su MNIST, del 90% su FASHION_MNIST e 70% su CIFAR10.
Questo piccolo esperimento servirà da base per costruire gli altri esperimenti della Tesi.

4

TRANSFER LEARNING

Come abbiamo visto nel capitolo precedente, una rete CNN semplice può ottenere buone performance su dataset facili come MNIST, ma su dataset più complessi come CIFAR10 l'accuracy si attesta solo al 70%. Per migliorare l'accuracy, ci sono due opzioni: aumentare la complessità della rete, i parametri di training, oppure utilizzare la tecnica del **transfer learning**.
In questo capitolo si utilizzeranno reti più complesse su cui si applicherà il transfer learning.

Il transfer learning è una tecnica introdotta nel 1976 che consiste nel prendere un modello già addestrato su un insieme di dati e riutilizzarlo su un diverso insieme di dati [24]. In pratica, il modello pre-addestrato viene utilizzato come punto di partenza per l'addestramento del nuovo modello.
Questo approccio consente di sfruttare le conoscenze acquisite dal modello pre-addestrato per velocizzare l'addestramento del nuovo modello e migliorarne le prestazioni, senza dover partire da zero. Ad esempio, se si ha a disposizione un modello addestrato su un insieme di immagini di oggetti, si può utilizzare questo modello per addestrare un nuovo modello per il riconoscimento di oggetti in un altro insieme di immagini. Il transfer learning è particolarmente utile in situazioni in cui il set di dati di addestramento è limitato o in cui l'addestramento di un modello da zero richiederebbe troppo tempo e risorse.

4.1 IMAGENET

ImageNet è un ampio database di immagini contenente oltre 14 milioni di immagini etichettate in oltre 20.000 categorie [35]. è stato creato nel 2009 con lo scopo di fornire un ampio set di dati utilizzabile come benchmark

per l'addestramento e la valutazione di modelli di riconoscimento di immagini e visione artificiale.

Keras mette a disposizione i pesi pre-addestrati di tutti i modelli elencati nella sezione precedente. Questi pesi possono essere utilizzati per inizializzare un nuovo modello, diminuendo dunque il tempo necessario per l'addestramento.

4.2 PRESENTAZIONE RETI PRE-ESISTENTI

La libreria di python *keras* intorodtta nel capitolo precedente mette a disposizione reti di vario tipo, come le ricorrenti (RNN), le feed forward (FFNN) le generative (GAN) e quelle che verranno usate agli scopi di questa Tesi: le convoluzionali (CNN) [25]. Qui sotto elencato un sottoinsieme delle reti offerte da *keras* insieme alla loro profondità, ovvero di quanti layer si compongono, il numero di parametri e la top-1 accuracy, ovvero l'accuracy del modello sull'insieme di valutazione ImageNet.

model	**depth**	**parameters**	**top-1 accuracy**
InceptionV3	189	23.9M	77.9%
VGG19	19	143.7M	71.3%
Xception	81	22.9M	79.0%
MobileNetV2	105	3.5M	71.3%
ResNet152V2	307	60.4M	78.0%
InceptionResNetV2	449	55.9M	80.3%
DenseNet201	402	20.2M	77.3%
EfficientNetV2S	-	350.1M	86.7%

Tabella 2.: reti keras

1. InceptionV3 (o GoogLeNet) è la terza evoluzione della rete Inception sviluppata da Google [26]. Ha una struttura a blocchi, dove ogni

blocco è composto da più strati convoluzionali con diversi filtri. Inception è nata specificatamente per ImageNet.

2. VGG19 è una rete neurale molto profonda, composta da diversi strati convoluzionali e di pooling [27]. Sono utilizzate principalmente per l'elaborazione di immagini ad alta risoluzione.

3. Xception è un'architettura di rete neurale convoluzionale (CNN) sviluppata da Google nel 2016 [28]. Utilizza un'idea innovativa chiamata "depthwise separable convolution" per ridurre il numero di parametri e migliorare l'efficienza computazionale della rete. Su ImageNet, Xception ha una accuracy leggermente migliore di InceptionV3.

4. MobileNet [29] è una rete neurale leggera e veloce, ideale per l'elaborazione di immagini su dispositivi mobili. La particolarità di questa rete è l'utilizzo di filtri di dimensione ridotta, che riducono il numero di parametri.

5. ResNet [30] è una rete progettata per supportare centinaia o migliaia di layer convoluzionali. La particolarità di questa rete è l'uso di "skip connection", ovvero il collegamento diretto tra due o più strati.

6. InceptionResNet [31] combina i vantaggi di due diverse architetture: Inception e ResNet. InceptionResNet è in grado di raggiungere un'elevata accuratezza di classificazione con un basso errore di test; tra tutte le reti qui presentate, InceptionResNet è quella che dopo EfficientNetV2S ha la migliore accuracy su ImageNet.

7. DenseNet [32] è una rete neurale con struttura a blocchi, dove ogni blocco riceve in input i dati del blocco precedente. In particolare, ogni strato riceve input da tutti i layer precedenti, creando una connessione densa tra tutti i livelli. Ciò consente a DenseNet di utilizzare in modo efficiente le informazioni provenienti da tutti i livelli, migliorando la capacità di apprendimento e l'accuratezza del modello.

8. EfficientNet [33] è una architettura di rete neurale profonda progettata per minimizzare la complessità computazionale. Utilizza una combinazione di metodi di riduzione dei parametri e di aumento dell'efficienza computazionale per creare una rete efficiente e molto precisa. Tra tutte le reti qui elencate, EfficientNetV2S [34] è quella che ha l'accuracy più alta su ImageNet.

Sfruttando ancora la libreria *keras* di *python* è possibile implementare una prima versione di rete che sfrutta il transfer learning. Nel codice seguente viene creata una rete a partire da InceptionV3 su cui poi viene applicato il transfer learning.

```
1  #load InceptionV3 model
2  base_model = InceptionV3(include_top=False,weights='imagenet',input_shape
       =(299, 299,3),classes=10)
3  base_model.trainable = False
4
5  #add custom classification layers
6  model = keras.models.Sequential()
7  model.add(Input(shape=(28, 28,3)))
8  model.add(Lambda(lambda image: preprocess_input(tf.image.resize(image,
       (299,299)))))
9  model.add(base_model)
10 model.add(Flatten())
11 model.add(Dense(128, activation='relu'))
12 model.add(Dropout(0.5))
13 model.add(Dense(10, activation='softmax'))
14 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
       accuracy'])
15 model.fit(x_train, y_train,
16                           validation_split=0.2,
17                           epochs=20, batch_size= 60,
18                           callbacks=[earlystopping_cb])
```

Alla riga 2 del codice viene costruito il modello di base InceptionV3 al quale vengono tolti i layer di classificazione (nel codice: $include\_top = False$) e vengono assegnati i pesi pre-addestrati di ImageNet.
Nelle righe successive vengono aggiunti degli strati di classificazione e poi come nella rete vista nel capitolo precedente viene chiamata la funzione $fit(...)$ per addestrare la rete.
È importante notare che la tecnica del transfer learning è individuabile nel fatto che gli unici pesi che cambieranno durante l'addestramento della rete saranno quelli dei nuovi strati di classificazione aggiunti e non quelli di ImageNet nell rete di base (riga 3). In questo modo utilizziamo la conoscenza già acquisita dalla rete senza andare a sovrascrivere nuovi pesi.

4.4 RISULTATI RETI KERAS CON TRANSFER LEARNING

A questo punto è possibile utilizzare il transfer learning per tutte le reti di *keras* viste in precedenza. Il codice completo per l'addestramento di queste reti sui 3 dataset MNIST, FASHION_MNIST e CIFAR10 è nell'appendice della Tesi.

modello	**MNIST**	**FASHION_MNIST**	**CIFAR10**
naïve	0.9894	0.9085	0.7016
InceptionV3	0.9843	0.8877	0.8222
VGG19	0.9894	0.9111	0.8594
Xception	0.9848	0.8992	0.8841
MobileNetV2	0.9858	0.9086	0.8199
ResNet152V2	0.9879	0.9023	0.8858
InceptionResNetV2	0.9879	0.8988	0.9021
DenseNet201	0.9901	0.9176	0.9006
EfficientNetV2S	0.9919	0.9289	0.9265

Tabella 3.: Transfer learning su reti keras. In evidenza le 3 migliori reti per ogni dataset.

Dal confronto con i risultati ottenuti nel capitolo precedente da una rete convoluzionale (tabella 1) possiamo vedere che per dataset facili come MNIST una rete convoluzionale con struttura semplice è molto efficace e riesce a convergere a una buona accuracy. Anche per reti più complesse usando transfer learning l'accuracy è sempre superiore al 98%. Per dataset più complessi come FASHION_MNIST non sempre utilizzare trnsfer learning è la soluzione ottimale. Infatti si vede che per alcune reti, come InceptionV2 o InceptionResNetV2, l'accuracy è minore di quella con la rete convoluzionale del capitolo precedente. Per CIFAR10 invece, tutte le reti su cui è stato utilizzato transfer learning hanno una accuracy

migliore dell'82% superando quindi di gran lunga la rete del capitolo precedente.
La rete che ha dato la migliore accuracy in ogni dataset è stata EfficientNetV2S dove utilizzando transfer learning, ha ottenuto un accuracy del 99% su MNIST e 92% sia su FASHION_MNIST che CIFAR10.

5

SAFETY WRAPPERS E SPROUT

In ambito di apprendimento automatico, un **safety wrapper** (o safety monitor) è una tecnica utilizzata per garantire che il modello di machine learning funzioni in modo sicuro, prevenendo possibili conseguenze negative per gli utenti, l'ambiente circostante o i dati stessi.
Il concetto di safety wrapper si basa sulla consapevolezza che i sistemi di machine learning possono essere soggetti a errori e comportamenti indesiderati [36], specialmente quando si tratta di applicazioni critiche come il controllo di veicoli autonomi, il monitoraggio medico, la sicurezza informatica, e così via [37].
Pertanto, l'obiettivo del safety wrapper è quello di creare una scatola nera intorno al modello di machine learning, che individui quando il modello commette errori durante la fase di previsione in modo che, una volta rilevati questi errori, il sistema possa decidere cosa fare.

Ad esempio, un safety wrapper potrebbe essere utilizzato per evitare che un sistema di diagnosi mediche classifichi in maniera errata un dato, o che un sistema di guida autonoma prenda decisioni che possano mettere a rischio la sicurezza degli occupanti o degli altri utenti della strada.
Le tecniche utilizzate per implementare un safety wrapper possono variare a seconda dell'applicazione e del contesto. Per esempio è possibile controllare che i dati in ingresso rispettino determinati vincoli o criteri di sicurezza, ad esempio evitando dati errati, incompleti o dannosi [38].
Altri modi possono essere calcolare delle **misure di incertezza** sui dati in ingresso e in uscita. Questo è l'approccio usato da SPROUT.

Un classificatore può essere visto ad alto livello nel modo seguente, dove α è l'accuracy e $\varepsilon = 1 - \alpha$ è invece la probabilità che il classificatore sbagli la sua previsione.

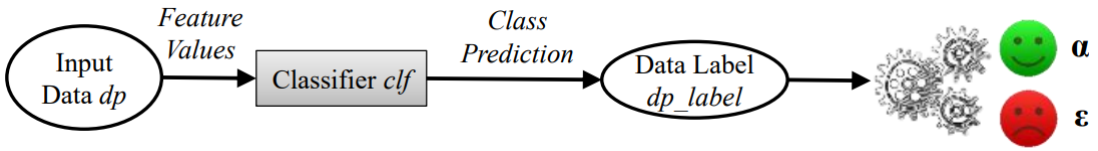


Figura 11.: classificatore supervisionato

Una volta che il classificatore viene avvolto con un safety wrapper può essere visto nel modo seguente:

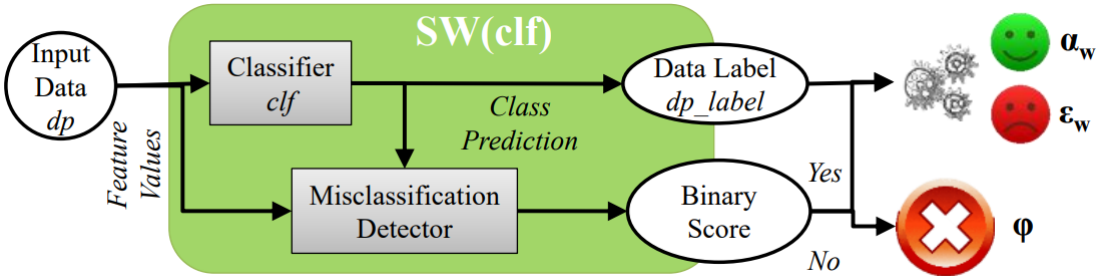


Figura 12.: classificatore avvolto con un safety wrapper

Indipendentemente da come è implementato, un safety wrapper trasforma un classificatore che ha una accuracy $0 \leqslant \alpha \leqslant 1$ e una probabilità di errore di $0 \leqslant \varepsilon = (1 - \alpha) < 1$ in un componente che ha

- accuracy $\alpha_w \leqslant \alpha$
- probabilità di omissione del risultato $0 \leqslant \varphi \leqslant 1$
- probabilità di errore residua $0 < \varepsilon_w \leqslant \varepsilon \leqslant 1$

Quello che è importante notare è che un classificatore avvolto da un safety wrapper non avrà mai una accuracy maggiore di α. L'obiettivo di questa tecnica è avere un componente quanto più possibile con la stessa accuracy $\alpha_w \approx \alpha$, un errore di classificazione molto minore $\varepsilon_w \ll \varepsilon$ e una probabilità di omissione dell'errore prossima a ε, ovvero $\varphi \approx \varepsilon$.

5.1 SPROUT

SPROUT (Safety wraPper thROugh ensembles of UncertainTy measures) è un safety wrapper che utilizza una combinazione di misure di incertezza per riuscire a individuare quando un classificatore commette degli errori. Quello che fa SPROUT è:

1. avvolgere un qualsiasi classificatore (in questa fase è visto come una "scatola nera").

2. computa delle misure di incertezza per ogni dato in input e il suo rispettivo risultato ottenuto dal classificatore.

3. produce un risultato binario (una misura di confidenza) che indica se la predizione del classificatore può essere usata in maniera sicura oppure se è un errore. Siccome è possibile vedere il monitor come un classificatore binario, tutte le metriche tradizionali di valutazione dei classificatori (precision, recall, ecc.) possono essere usate per valutare la sua performance.

Le misure di incertezza usate da SPROUT sono le seguenti:

UM#	**Name**	**Parameters**
UM1	Confidence Intervals	w: confidence level
UM2	Maximum Likelihood	-
UM3	Entropy of Probabilities	-
UM4	Bayesian Uncertainty	chk_c: classifier to check agreement with
UM5	Combined Uncertainty	CC: classifiers to check agreement with
UM6	Multi-Combined Uncertainty	bagC: classifier to build bagger set
UM7	Feature Bagging	k: number of relevant neighbors
UM8	Neighbourhood Agreement	
UM9	Reconstruction Loss	layers: structure of the AutoEncoder

Tabella 4.: misure incertezza SPROUT

Dunque sempre ad alto livello SPROUT per dataset etichettati può essere visto nel modo seguente:

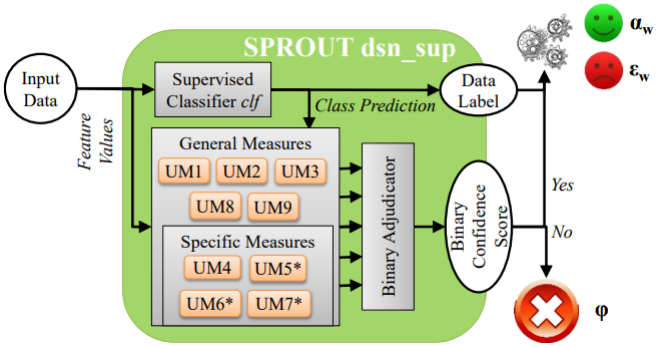


Figura 13.: schema SPROUT

A livello di implementazione, applicare SPROUT a un classificatore presuppone la sola esposizione di una funzione `predict_prob(...)` per generare `dp_label` da parte del classificatore. Si può notare come la maggior parte dei framework (scikit-learn, xgboost, pyod, ...) di machine learning espongano questa funzione. Per una prima applicazione di SPROUT possiamo vedere il codice qui sotto listato. L'implementazione di SPROUT invece è possibile trovarla nella repository GitHub [39].

```
1  import sklearn, numpy
2  from sprout.SPROUTObject import SPROUTObject
3  x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split
       (x, y, test_size=0.5)
4
5  sp_obj = SPROUTObject()
6  sp_obj.load_wrapper(model_tag="dsn_sup", x_train=x_train, y_train=y_train,
       label_names=label_tags)
7
8  classifier = sklearn.ensemble.RandomForestClassifier()
9  classifier.fit(x_train, y_train)
10
11 sp_df, bin_adj = sp_obj.predict_misclassifications(data_set=x_test,
       classifier=classifier)
12
13 sprout_pred = sprout_df["pred"].to_numpy()
14 phi = 100*numpy.count_nonzero(sprout_pred == 0) / len(sprout_pred)
15 alphaw = sum((1-sprout_pred)*(1-y_test)) / numpy.count_nonzero(sprout_pred
       == 0)
16 epsilonw = 1 - phi - alphaw
```

Nel codice, dopo aver caricato il dataset etichettato (riga 3), viene inizializzato un wrapper SPROUT (riga 5) e successivamente caricato un modello specifico per il binary adjudicator (riga 6). SPROUT ha 7 modelli per il binary adjudicator, divisi in modelli per la classificazione supervisionata (per esempio all_sup_fast, all_sup_fast_2 e dsn_sup_2) e quella non su-

pervisionata (per esempio dsn_uns). Nella riga 8 e 9 viene utilizzato il classificatore per avere una previsione sui dati in input. Questa viene poi passata a SPROUT (riga 11) per avere una previsione sulla bontà della classificazione. Nelle righe successive vengono calcolate α_w, ε_w e φ.

5.2 RISULTATI SPROUT SULLA CLASSIFICAZIONE DI IMMAGINI

Utilizzando sempre *keras* e SPROUT è ora possibile condurre degli esperimenti sull'utilizzo del wrapper applicato ai 3 dataset (MNIST, FASHION_MNIST e CIFAR10) e alle migliori reti trovate nel capitolo precedente.
Il codice completo per i seguenti esperimenti si trova nell'appendice della Tesi. Gli esperimenti sono condotti nel modo seguente:

- per ogni dataset sono state prese le 3 reti migliori.
  Per MNIST e FASHION_MNIST: EfficientNetV2S, DenseNet201, VGG19.
  Per CIFAR10: EfficientNetV2S, DenseNet201, InceptionResNetV2.
  - sono presentate le metriche di accuracy α_w, missclassification ε e prediction time sulla coppia [Rete,Dataset].
  - applico SPROUT sulla coppia [Rete,Dataset] usando 3 diversi modelli per SPROUT (all_sup_fast,all_sup_fast_2,dsn_sup_2). Per ognuno sono presentate le metriche di accuracy α_w, missclassification ε, omission rate φ e prediction time.

dunque

1. EfficientNetV2S,MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.992	0.960	0.979	0.969
missclassification	0.008	0.002	0.002	0.003
omission rate	-	0.038	0.019	0.029
prediction time	23.303ms	182.342ms	149.639ms	443.719ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 5.: EfficientNetV2S su MNIST, senza e con SPROUT

2. EfficientNetV2S,FASHION_MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.923	0.823	0.877	0.854
missclassification	0.077	0.023	0.029	0.030
omission rate	-	0.153	0.094	0.116
prediction time	24.744ms	150.885ms	177.513ms	530.625ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 6.: EfficientNetV2S su FASHION_MNIST, senza e con SPROUT

3. EfficientNetV2S,CIFAR10

	senza SPROUT	**con SPROUT**		
accuracy	0.924	0.204	0.405	0.274
missclassification	0.076	0.005	0.013	0.007
omission rate	-	0.790	0.582	0.719
prediction time	23.605ms	153.415ms	134.875ms	785.241ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 7.: EfficientNetV2S su CIFAR10, senza e con SPROUT

4. DenseNet201,MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.987	0.956	0.975	0.966
missclassification	0.013	0.002	0.003	0.003
omission rate	-	0.041	0.022	0.031
prediction time	64.311ms	246.065ms	361.150ms	567.721ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 8.: DenseNet201 su MNIST, senza e con SPROUT

5. DenseNet201,FASHION_MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.917	0.819	0.876	0.849
missclassification	0.083	0.023	0.030	0.033
omission rate	-	0.158	0.093	0.117
prediction time	37.028ms	196.171ms	237.653ms	981.724ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 9.: DenseNet201 su FASHION_MNIST, senza e con SPROUT

6. DenseNet201,CIFAR10

	senza SPROUT	**con SPROUT**		
accuracy	0.899	0.204	0.411	0.286
missclassification	0.101	0.007	0.020	0.011
omission rate	-	0.789	0.570	0.703
prediction time	37.073ms	189.181ms	214.714ms	842.306ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 10.: DenseNet201 su CIFAR10, senza e con SPROUT

7. VGG19,MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.988	0.957	0.975	0.967
missclassification	0.012	0.002	0.003	0.003
omission rate	-	0.041	0.022	0.030
prediction time	42.405ms	383.763ms	281.116ms	736.731ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 11.: VGG19 su MNIST, senza e con SPROUT

8. VGG19,FASHION_MNIST

	senza SPROUT	**con SPROUT**		
accuracy	0.915	0.821	0.868	0.847
missclassification	0.085	0.024	0.030	0.033
omission rate	-	0.154	0.102	0.120
prediction time	49.056ms	305.654ms	338.232ms	858.302
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 12.: VGG19 su FASHION_MNIST, senza e con SPROUT

9. InceptionResNetV2,CIFAR10

	senza SPROUT	**con SPROUT**		
accuracy	0.905	0.205	0.414	0.279
missclassification	0.095	0.006	0.017	0.009
omission rate	-	0.789	0.569	0.712
prediction time	65.462ms	354.656ms	252.969ms	1088.689ms
modello SPROUT	-	all_sup_fast	all_sup_fast_2	dsn_sup_2

Tabella 13.: InceptionResNetV2 su CIFAR10, senza e con SPROUT

Si può osservare come diminuisce l'accuracy con l'utilizzo di SPROUT: con MNIST diminuisce sempre di almeno lo 0,013 (pari all'1%). Analogamente, l'accuracy di FASHION_MNIST diminuisce di almeno 0,046 (pari al 4%) e quella di CIFAR10 di almeno 0,51 (pari al 50%). L'accuracy su CIFAR10 è la più bassa, questo per via della complessità di questo dataset, tuttavia la misclassification per questo dataset non sale mai sopra 0.020. Anche per gli altri dataset otteniamo dei buoni risultati per quanto riguarda omission rate e misclassification, con quest'ultima che non sale mai sopra 0.033. Bisogna anche tenere conto del fatto che utilizzando il safety wrapper non solo è possibile riscontrare una diminuzione dell'accuracy, ma anche un aumento del tempo di classificazione. In generale avremo sempre un aumento del tempo di classificazione di almento 111 ms ma l'utilizzo di modelli SPROUT più complessi, come ad esempio dsn_sup_2, può comportare un aumento significativo del tempo di classificazione senza necessariamente migliorare l'accuracy.

6

CONCLUSIONI

L'obiettivo del presente lavoro di Tesi era quello di valutare l'utilizzo del safety wrapper come tecnica di protezione delle reti neurali. In particolare, il lavoro si è concentrato sull'analisi delle prestazioni di diverse reti neurali con e senza il safety wrapper su tre dataset di immagini (MNIST, FASHION_MNIST e CIFAR10).
Nelle due tabelle successive sono riportati i risultati con accuracy più elevata ottenuti senza e con SPROUT nelle migliori reti con transfer learning:

	MNIST	**FASHION_MNIST**	**CIFAR10**
rete	EfficientNetV2S	EfficientNetV2S	EfficientNetV2S
accuracy	0.992	0.923	0.924
missclassification	0.077	0.029	0.076
prediction time	23.303ms	24.744ms	23.605ms

Tabella 14.: tabella riassuntiva migliori risultati senza SPROUT

	MNIST	**FASHION_MNIST**	**CIFAR10**
rete	EfficientNetV2S	EfficientNetV2S	InceptionResNetV2
accuracy	0.979	0.877	0.414
missclassification	0.002	0.029	0.017
omission rate	0.019	0.094	0.569
prediction time	149.639ms	177.513ms	252.969ms
modello SPROUT	all_sup_fast_2	all_sup_fast_2	all_sup_fast_2

Tabella 15.: tabella riassuntiva migliori risultati con SPROUT

Per quanto riguarda i risultati ottenuti con SPROUT, le migliori reti risultato essere EfficientNetV2S per MNIST e FASHION_MNIST e InceptionResNetV2 per CIFAR10 (usando per tutte il modello all_sup_fast_2 di SPROUT).
Si può vedere che per dataset come MNIST e FASHION_MNIST, SPROUT è molto efficace e riesce a prevedere la maggior parte degli errori di classificazione. Per dataset più complessi come CIFAR10, SPROUT invece fatica di più a capire se il risultato della classificazione è corretto o meno dunque il tasso di omissione φ è alto e l'accuracy bassa ($\alpha_w = 0.414$). In ogni caso, l'errore residuo è molto basso $\varepsilon_w \leqslant 0.029$.

Il lavoro svolto ha dimostrato l'efficacia dei safety wrapper nel campo del riconoscimento delle immagini, soprattutto in ambienti critici dove la sicurezza dei dati e dei sistemi è fondamentale. Tuttavia, è importante notare che l'uso del safety wrapper rallenta notevolmente il classificatore, quindi è necessario valutare caso per caso il suo utilizzo.

Nonostante ciò, la campagna sperimentale condotta ha provato che i safety wrapper rappresentano una soluzione promettente per garantire la sicurezza dei sistemi di apprendimento automatico, nonostante ci siano ancora margini di miglioramento per garantire un livello di protezione ancora maggiore e un tempo per la classificazione minore.

BIBLIOGRAFIA

[1] mathworks, "Che cos'è l'Image Recognition?". https://it.mathworks.com/discovery/image-recognition-matlab.html (Cited on page 5.)

[2] A. Avizienis et al. "Basic concepts and taxonomy of dependable and secure computing". https://ieeexplore.ieee.org/document/1335465 (Cited on page 5.)

[3] Arthur Lee Samuel, "Some Studies in Machine Learning Using the Game of Checkers". https://ieeexplore.ieee.org/document/5392560 (Cited on page 7.)

[4] Frank Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain". https://psycnet.apa.org/record/1959-09865-001 (Cited on page 7.)

[5] Doruk Sena, Melike Ozturkb, Ozalp Vayvay, "An Overview of Big Data for Growth in SMEs". https://www.researchgate.net/publication/311357755_An_Overview_of_Big_Data_for_Growth_in_SMEs (Cited on page 7.)

[6] Wikipedia, legge di Moore. https://it.wikipedia.org/wiki/Legge_di_Moore (Cited on page 7.)

[7] Google image classification in Google Photo. https://developers.google.com/machine-learning/practica/image-classification (Cited on page 8.)

[8] Tesla, AI & Robotics.https://www.tesla.com/AI (Cited on page 8.)

[9] Neal Karlinsky, "How artificial intelligence helps Amazon deliver". https://www.aboutamazon.com/news/innovation-at-amazon/how-artificial-intelligence-helps-amazon-deliver (Cited on page 8.)

[10] Facebook, "How Facebook is using AI to improve photo descriptions for people who are blind or visually impaired". https://ai.facebook.com/blog/how-facebook-is-using-ai-to-

improve-photo-descriptions-for-people-who-are-blind-or-visually-impaired/ (Cited on page 8.)

[11] Marvin Minsky and Seymour Papert, "A Step toward the Understanding of Information Processes: Perceptrons. An Introduction to Computational Geometry". https://www.science.org/doi/10.1126/science.165.3895.780 (Cited on page 8.)

[12] Wikipedia, funzioni di attivazione. https://en.wikipedia.org/wiki/Activation_function (Cited on page 10.)

[13] Wikipedia, rete neurale artificiale. https://it.wikipedia.org/wiki/Rete_neurale_artificiale (Cited on page 12.)

[14] ZhiFei Lai and HuiFang Deng, "Medical Image Classification Based on Deep Features Extracted by Deep Model and Statistic Feature Fusion with Multylayer Perceptron". https://www.hindawi.com/journals/cin/2018/2061516/ (Cited on page 14.)

[15] Majd Alqarqaz, Maram Bani Younes and Raneem Qaddoura, "An Object Classification Approach for Autonomous Vehicles Using Machine Learning Techniques". https://www.mdpi.com/2032-6653/14/2/41 (Cited on page 14.)

[16] Atul Sharma et al. "Image Classification Using CNN". https://ssrn.com/abstract=3833453 (Cited on page 15.)

[17] Wikipedia, operazione matematica di convoluzione. https://it.wikipedia.org/wiki/Convoluzione (Cited on page 15.)

[18] Martin Mayer-Krebs, "Why accuracy (might) not be a good measure". https://www.qmr.ai/why-accuracy-might-not-be-a-good-measure/ (Cited on page 16.)

[19] Davide Chicco e Giuseppe Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7 (Cited on page 17.)

[20] Wikipedia, confusion matrix. https://en.wikipedia.org/wiki/Confusion_matrix (Cited on page 17.)

[21] Yann LeCun, Courant Institute. http://yann.lecun.com/exdb/mnist/ (Cited on page 17.)

[22] Han Xiao, Kashif Rasul, Roland Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". https://arxiv.org/abs/1708.07747 (Cited on page 18.)

[23] cifar10, Toronto university. https://www.cs.toronto.edu/~kriz/cifar.html (Cited on page 19.)

[24] Stevo. Bozinovski and Ante Fulgosi, "The influence of pattern similarity and transfer learning upon the training of a base perceptron B2." .https://www.informatica.si/index.php/informatica/article/view/2828/1433

[25] reti keras. https://keras.io/api/applications/ (Cited on page 23.)

[26] Szegedy et al. (2015), "Going deeper with convolutions". https://arxiv.org/pdf/1409.4842.pdf (Cited on pages 15 and 24.)

[27] Karen Simonyan and Andrew Zisserman,"Very Deep Convolutional Networks for Large-Scale Image Recognition". https://arxiv.org/abs/1409.1556v6 (Cited on page 24.)

[28] François Chollet, "Xception: Deep Learning With Depthwise Separable Convolutions". https://arxiv.org/abs/1610.02357v3 (Cited on page 25.)
(Cited on page 25.)

[29] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". https://arxiv.org/abs/1704.04861 (Cited on page 25.)

[30] Kaiming He et al. "Deep Residual Learning for Image Recognition". https://arxiv.org/abs/1512.03385v1 (Cited on page 25.)

[31] Christian Szegedy et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". https://arxiv.org/abs/1602.07261v2 (Cited on page 25.)

[32] Gao Huang et al. "Densely Connected Convolutional Networks". https://arxiv.org/abs/1608.06993 (Cited on page 25.)

[33] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". https://arxiv.org/abs/1905.11946 (Cited on page 25.)

[34] Mingxing Tan, Quoc V. Le, "EfficientNetV2: Smaller Models and Faster Training". https://arxiv.org/abs/2104.00298 (Cited on page 25.)

[35] Imagenet. https://www.image-net.org/ (Cited on page 23.)

[36] Dan Hendrycks, Kevin Gimpel, "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". https://openreview.net/forum?id=Hkg4TI9xl (Cited on page 29.)

[37] Dario Amodei et al. "Concrete Problems in AI Safety". https://arxiv.org/abs/1606.06565 (Cited on page 29.)

[38] Bin Liang, Jiachun Li, Jianjun Huang, "We Can Always Catch You: Detecting Adversarial Patched Objects WITH or WITHOUT Signature". https://arxiv.org/abs/2106.05261 (Cited on page 29.)

[39] GitHub, SPROUT. https://github.com/tommyippoz/SPROUT (Cited on page 32.)

A

CODICE UTILIZZATO

Di seguito viene riportato il codice utilizzato nei capitoli 4 e 5.

A.1 TRANSFER LEARNING

Transfer learning dal modello InceptionV3 sui dataset MNIST,FASHION_MNIST e CIFAR10.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import numpy as np
from keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.inception_v3 import InceptionV3,
    preprocess_input
from keras.models import Sequential

EPOCHS=20
BATCH_SIZE=60

#load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

#expand x_train and x_test to 3 dimension (RGB)
x_train = np.expand_dims(x_train, axis=-1)
x_train = np.repeat(x_train, 3, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
x_test = np.repeat(x_test, 3, axis=-1)

#one hot encoding
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)

#load base model
```

```
base_model = InceptionV3(include_top=False,weights='imagenet',input_shape
    =(299, 299,3),classes=10)
base_model.trainable = False #freezing base model

#add custom classification layers
model = keras.models.Sequential()
model.add(Input(shape=(28, 28,3)))
model.add(Lambda(lambda image: preprocess_input(tf.image.resize(image,
    (299,299)))))
model.add(base_model)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

model.fit(x_train, y_train,
                           validation_split=0.2,
                           epochs=EPOCHS, batch_size= BATCH_SIZE,
                           callbacks=[earlystopping_cb])

mnist_iv3_val_accuracy=model.evaluate(x=x_test,y=y_test)[1]
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import numpy as np
from keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.inception_v3 import InceptionV3,
    preprocess_input
from keras.models import Sequential

EPOCHS=20
BATCH_SIZE=60

#load FASHION_MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.
    load_data()

#expand x_train and x_test to 3 dimension (RGB)
x_train = np.expand_dims(x_train, axis=-1)
x_train = np.repeat(x_train, 3, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
x_test = np.repeat(x_test, 3, axis=-1)

```

```
#one hot encoding
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)

#load base model
base_model = InceptionV3(include_top=False,weights='imagenet',input_shape
    =(299, 299,3),classes=10)
base_model.trainable = False

#add custom classification layers
model = keras.models.Sequential()
model.add(Input(shape=(28, 28,3)))
model.add(Lambda(lambda image: preprocess_input(tf.image.resize(image,
    (299,299)))))
model.add(base_model)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

model.fit(x_train, y_train,
                           validation_split=0.2,
                           epochs=EPOCHS, batch_size= BATCH_SIZE,
                           callbacks=[earlystopping_cb])

fashion_mnist_iv3_val_accuracy=model.evaluate(x=x_test,y=y_test)[1]
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import numpy as np
from keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.inception_v3 import InceptionV3,
    preprocess_input
from keras.models import Sequential

EPOCHS=20
BATCH_SIZE=60

#load CIFAR10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data
    ()

#one hot encoding
```

```
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)

#load base model
base_model = InceptionV3(include_top=False,weights='imagenet',input_shape
    =(299, 299,3),classes=10)
base_model.trainable = False

#add custom classification layers
model = keras.models.Sequential()
model.add(Input(shape=(32, 32,3)))
model.add(Lambda(lambda image: preprocess_input(tf.image.resize(image,
    (299,299)))))
model.add(base_model)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

model.fit(x_train, y_train,
                           validation_split=0.2,
                           epochs=EPOCHS, batch_size= BATCH_SIZE,
                           callbacks=[earlystopping_cb])

cifar10_val_accuracy=model.evaluate(x=x_test,y=y_test)[1]
```

A.2 SPROUT

Utilizzo di SPROUT avvolgendo InceptionV3 sui dataset MNIST,FASHION_MNIST e CIFAR10.

```
import numpy
import pandas
import sklearn

from sprout.SPROUTObject import SPROUTObject
from sprout.utils import sprout_utils
from sprout.utils.dataset_utils import load_MNIST
from sprout.utils.general_utils import current_ms
from sprout.utils.sprout_utils import correlations

import tensorflow as tf
from tensorflow import keras
from keras.layers import *
```

```
14 from tensorflow.keras.utils import to_categorical
15 from tensorflow.keras.applications.inception_v3 import InceptionV3,
       preprocess_input
16 from keras.models import Sequential
17
18 #custom wrapper for keras model (for predict and predict_proba functions)
19 class MyClassifier():
20     def __init__(self, classifier, original_image_size = 28, is_dataset_rgb
       = False):
21         self.classifier = classifier
22         self.is_dataset_rgb = is_dataset_rgb
23
24     def predict(self, data):
25         if len(data.shape) < 3:
26             if self.is_dataset_rgb:
27                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size, 3)).numpy()
28             else:
29                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size)).numpy()
30                 data = numpy.expand_dims(data, axis=-1)
31                 data = numpy.repeat(data, 3, axis=-1)
32         return numpy.argmax(self.classifier.predict(data), axis=1)
33
34     def predict_proba(self, data):
35         if len(data.shape) < 3:
36             if self.is_dataset_rgb:
37                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size, 3)).numpy()
38             else:
39                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size)).numpy()
40                 data = numpy.expand_dims(data, axis=-1)
41                 data = numpy.repeat(data, 3, axis=-1)
42         return self.classifier.predict(data)
43
44 MODELS_FOLDER = "../models/"
45 MODEL_TAG = "all_sup_fast"
46 EPOCHS = 10
47 BATCH_SIZE = 60
48
49 if __name__ == '__main__':
50
51     # Reading sample dataset (MNIST). Loads data as images
52     x_train, x_test, y_train, y_test, label_names, feature_names =
       load_MNIST(flatten=False, row_limit=5000)
53
54     is_dataset_rgb = True if len(x_train.shape) > 3 else False
```

```
55      original_image_size = x_train.shape[1]
56
57      # expand dimension of train set to feed the neural network (from
        (28,28) to (28,28,3))
58      x_train = numpy.expand_dims(x_train, axis=-1)
59      x_train = numpy.repeat(x_train, 3, axis=-1)
60      x_test = numpy.expand_dims(x_test, axis=-1)
61      x_test = numpy.repeat(x_test, 3, axis=-1)
62
63      # Reading sample dataset (MNIST). Loads data as images which are going
        to be linearized (flatten = true)
64      # It is required to feed data to SPROUT
65      x_train_lin, x_test_lin, y_train_lin, y_test_lin, label_names,
        feature_names = load_MNIST(flatten=True, row_limit=5000)
66
67      sprout_obj = SPROUTObject(models_folder=MODELS_FOLDER)
68      sprout_obj.load_model(model_tag=MODEL_TAG, x_train=x_train_lin, y_train
        =y_train_lin, label_names=label_names)
69
70      base_model = InceptionV3(include_top=False, weights='imagenet',
        input_shape=(299, 299, 3), classes=10)
71      base_model.trainable = False
72      classifier = keras.models.Sequential()
73      classifier.add(Input(shape=(original_image_size, original_image_size,3)
        ))
74      classifier.add(Lambda(lambda image: preprocess_input(tf.image.resize(
        image, (299, 299)))))
75      classifier.add(base_model)
76      classifier.add(Flatten())
77      classifier.add(Dense(128, activation='relu'))
78      classifier.add(Dropout(0.5))
79      classifier.add(Dense(10, activation='softmax'))
80      classifier.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])
81      classifier.fit(x_train, to_categorical(y_train),
82                     epochs=EPOCHS,
83                     batch_size=BATCH_SIZE,
84                     validation_split=0.4,
85                     callbacks=[keras.callbacks.EarlyStopping(patience=3,
        restore_best_weights=True)])
86
87      myclassifier = MyClassifier(classifier, original_image_size =
        original_image_size, is_dataset_rgb=is_dataset_rgb)
88
89      start_pred = current_ms()
90      y_pred = myclassifier.predict(x_test)
91      clf_time = (current_ms() - start_pred) / len(y_test)
92      y_proba = myclassifier.predict_proba(x_test)
```

```
    clf_misc = numpy.asarray(y_pred != y_test_lin)
    clf_acc = sklearn.metrics.accuracy_score(y_test_lin, y_pred)
    print("Fit and Prediction completed with Accuracy: " + str(clf_acc))

    # Calculating Trust Measures with SPROUT
    out_df = sprout_utils.build_SPROUT_dataset(y_proba, y_pred, y_test_lin,
     label_names)
    start_pred = current_ms()
    sp_df, adj = sprout_obj.predict_set_misclassifications(data_set=
    x_test_lin, y_proba=y_proba, classifier=myclassifier)
    sprout_time = (current_ms() - start_pred) / len(y_test)
    sprout_pred = sp_df["pred"].to_numpy()

    # Computing SPROUT Metrics for output
    # o_rate is phi
    # sp_acc is alpha_w
    # 1-sp_acc-o_rate is epsilon_w
    o_rate = numpy.average(sprout_pred)
    susp_misc = sum(sprout_pred*clf_misc) / sum(clf_misc)
    sp_acc = numpy.average((1-clf_misc)*(1-sprout_pred))

    print('\n------------  SPROUT Report  ------------------\n')
    print('Regular classifier has accuracy of %.3f and %.3f
    misclassifications' % (clf_acc, 1-clf_acc))
    print('SPROUT suspects %.3f of misclassifications' % (susp_misc))
    print('Classifier wrapped with SPROUT has %.3f accuracy, %.3f omission
    rate, '
          'and %.3f residual misclassifications' % (sp_acc, o_rate, 1-
    sp_acc-o_rate))
    print('Prediction Time of the regular classifier %.3f ms per item, with
     SPROUT: %.3f per item'
          % (clf_time, sprout_time))


    out_df = pandas.concat([out_df, sp_df.drop(columns=["pred"])], axis=1)
    out_df["clf_pred"] = y_pred
    out_df["sprout_omit (binary confidence score)"] = sp_df["pred"]
    out_df.to_csv('mnist.csv', index=False)
```

```
import numpy
import pandas
import sklearn

from sprout.SPROUTObject import SPROUTObject
from sprout.utils import sprout_utils
from sprout.utils.dataset_utils import load_FASHIONMNIST
from sprout.utils.general_utils import current_ms
from sprout.utils.sprout_utils import correlations
```

```
10
11 import tensorflow as tf
12 from tensorflow import keras
13 from keras.layers import *
14 from tensorflow.keras.utils import to_categorical
15 from tensorflow.keras.applications.inception_v3 import InceptionV3,
       preprocess_input
16 from keras.models import Sequential
17
18 #custom wrapper for keras model (for predict and predict_proba functions)
19 class MyClassifier():
20     def __init__(self, classifier, original_image_size = 28, is_dataset_rgb
       = False):
21         self.classifier = classifier
22         self.is_dataset_rgb=is_dataset_rgb
23
24     def predict(self, data):
25         if len(data.shape) < 3:
26             if self.is_dataset_rgb:
27                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size, 3)).numpy()
28             else:
29                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size)).numpy()
30                 data = numpy.expand_dims(data, axis=-1)
31                 data = numpy.repeat(data, 3, axis=-1)
32         return numpy.argmax(self.classifier.predict(data), axis=1)
33
34     def predict_proba(self, data):
35         if len(data.shape) < 3:
36             if self.is_dataset_rgb:
37                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size, 3)).numpy()
38             else:
39                 data = tf.reshape(data, (data.shape[0], original_image_size
       , original_image_size)).numpy()
40                 data = numpy.expand_dims(data, axis=-1)
41                 data = numpy.repeat(data, 3, axis=-1)
42         return self.classifier.predict(data)
43
44 MODELS_FOLDER = "../models/"
45 MODEL_TAG = "all_sup_fast"
46 EPOCHS = 10
47 BATCH_SIZE = 60
48
49 if __name__ == '__main__':
50     # Reading sample dataset (FASHION_MNIST). Loads data as images
51     x_train, x_test, y_train, y_test, label_names, feature_names =
```

```
    load_FASHIONMNIST(flatten=False, row_limit=5000)
52
53      is_dataset_rgb = True if len(x_train.shape) > 3 else False
54      original_image_size = x_train.shape[1]
55
56      # expand dimension of train set to feed the neural network (from
        (28,28) to (28,28,3))
57      x_train = numpy.expand_dims(x_train, axis=-1)
58      x_train = numpy.repeat(x_train, 3, axis=-1)
59      x_test = numpy.expand_dims(x_test, axis=-1)
60      x_test = numpy.repeat(x_test, 3, axis=-1)
61
62      # Reading sample dataset (FASHION_MNIST). Loads data as images which
        are going to be linearized (flatten = true)
63      # It is required to feed data to SPROUT
64      x_train_lin, x_test_lin, y_train_lin, y_test_lin, label_names,
        feature_names = load_FASHIONMNIST(flatten=True,
65
                                  row_limit=5000)
66
67      sprout_obj = SPROUTObject(models_folder=MODELS_FOLDER)
68      sprout_obj.load_model(model_tag=MODEL_TAG, x_train=x_train_lin, y_train
        =y_train_lin, label_names=label_names)
69
70      base_model = InceptionV3(include_top=False, weights='imagenet',
        input_shape=(299, 299, 3), classes=10)
71      base_model.trainable = False
72      classifier = keras.models.Sequential()
73      classifier.add(Input(shape=(original_image_size, original_image_size,
        3)))
74      classifier.add(Lambda(lambda image: preprocess_input(tf.image.resize(
        image, (299, 299)))))
75      classifier.add(base_model)
76      classifier.add(Flatten())
77      classifier.add(Dense(128, activation='relu'))
78      classifier.add(Dropout(0.5))
79      classifier.add(Dense(10, activation='softmax'))
80      classifier.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])
81      classifier.fit(x_train, to_categorical(y_train),
82                     epochs=EPOCHS,
83                     batch_size=BATCH_SIZE,
84                     validation_split=0.4,
85                     callbacks=[keras.callbacks.EarlyStopping(patience=3,
        restore_best_weights=True)])
86
87      myclassifier = MyClassifier(classifier,original_image_size=
        original_image_size, is_dataset_rgb=is_dataset_rgb)
```

```
88
89      start_pred = current_ms()
90      y_pred = myclassifier.predict(x_test)
91      clf_time = (current_ms() - start_pred) / len(y_test)
92      y_proba = myclassifier.predict_proba(x_test)
93      clf_misc = numpy.asarray(y_pred != y_test_lin)
94      clf_acc = sklearn.metrics.accuracy_score(y_test_lin, y_pred)
95      print("Fit and Prediction completed with Accuracy: " + str(clf_acc))
96
97      # Calculating Trust Measures with SPROUT
98      out_df = sprout_utils.build_SPROUT_dataset(y_proba, y_pred, y_test_lin,
         label_names)
99      start_pred = current_ms()
100     sp_df, adj = sprout_obj.predict_set_misclassifications(data_set=
        x_test_lin, y_proba=y_proba, classifier=myclassifier)
101     sprout_time = (current_ms() - start_pred) / len(y_test)
102     sprout_pred = sp_df["pred"].to_numpy()
103
104     # Computing SPROUT Metrics for output
105     # o_rate is phi
106     # sp_acc is alpha_w
107     # 1-sp_acc-o_rate is epsilon_w
108     o_rate = numpy.average(sprout_pred)
109     susp_misc = sum(sprout_pred*clf_misc) / sum(clf_misc)
110     sp_acc = numpy.average((1-clf_misc)*(1-sprout_pred))
111
112     print('\n------------  SPROUT Report  ------------------\n')
113     print('Regular classifier has accuracy of %.3f and %.3f
        misclassifications' % (clf_acc, 1-clf_acc))
114     print('SPROUT suspects %.3f of misclassifications' % (susp_misc))
115     print('Classifier wrapped with SPROUT has %.3f accuracy, %.3f omission
        rate, '
116           'and %.3f residual misclassifications' % (sp_acc, o_rate, 1-
        sp_acc-o_rate))
117     print('Prediction Time of the regular classifier %.3f ms per item, with
         SPROUT: %.3f per item'
118           % (clf_time, sprout_time))
119
120
121     out_df = pandas.concat([out_df, sp_df.drop(columns=["pred"])], axis=1)
122     out_df["clf_pred"] = y_pred
123     out_df["sprout_omit (binary confidence score)"] = sp_df["pred"]
124     out_df.to_csv('fashion_mnist.csv', index=False)
```

```
1   import numpy
2   import pandas
3   import sklearn
4
```

```
from sprout.SPROUTObject import SPROUTObject
from sprout.utils import sprout_utils
from sprout.utils.dataset_utils import load_CIFAR10
from sprout.utils.general_utils import current_ms
from sprout.utils.sprout_utils import correlations

import tensorflow as tf
from tensorflow import keras
from keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.inception_v3 import InceptionV3,
    preprocess_input
from keras.models import Sequential

#custom wrapper for keras model (for predict and predict_proba functions)
class MyClassifier():
    def __init__(self, classifier,original_image_size=28,is_dataset_rgb=
    False):
        self.classifier = classifier
        self.is_dataset_rgb=is_dataset_rgb

    def predict(self, data):
        if len(data.shape) < 3:
            if self.is_dataset_rgb:
                data = tf.reshape(data, (data.shape[0], original_image_size
    , original_image_size, 3)).numpy()
            else:
                data = tf.reshape(data, (data.shape[0], original_image_size
    , original_image_size)).numpy()
                data = numpy.expand_dims(data, axis=-1)
                data = numpy.repeat(data, 3, axis=-1)
        return numpy.argmax(self.classifier.predict(data), axis=1)

    def predict_proba(self, data):
        if len(data.shape) < 3:
            if self.is_dataset_rgb:
                data = tf.reshape(data, (data.shape[0], original_image_size
    , original_image_size, 3)).numpy()
            else:
                data = tf.reshape(data, (data.shape[0], original_image_size
    , original_image_size)).numpy()
                data = numpy.expand_dims(data, axis=-1)
                data = numpy.repeat(data, 3, axis=-1)
        return self.classifier.predict(data)

MODELS_FOLDER = "../models/"
MODEL_TAG = "all_sup_fast"
EPOCHS = 10
```

```
47 BATCH_SIZE = 60
48
49 if __name__ == '__main__':
50     # Reading sample dataset (MNIST). Loads data as images
51     x_train, x_test, y_train, y_test, label_names, feature_names =
       load_CIFAR10(flatten=False, row_limit=5000)
52
53     is_dataset_rgb = True if len(x_train.shape) > 3 else False
54     original_image_size = x_train.shape[1]
55
56     # Reading sample dataset (CIFAR10).
57     x_train_lin, x_test_lin, y_train_lin, y_test_lin, label_names,
       feature_names = load_CIFAR10(flatten=True,
58
                                    row_limit=5000)
59     sprout_obj = SPROUTObject(models_folder=MODELS_FOLDER)
60     sprout_obj.load_model(model_tag=MODEL_TAG, x_train=x_train_lin, y_train
       =y_train_lin, label_names=label_names)
61
62     base_model = InceptionV3(include_top=False, weights='imagenet',
       input_shape=(299, 299, 3), classes=10)
63     base_model.trainable = False
64     classifier = keras.models.Sequential()
65     classifier.add(Input(shape=(original_image_size, original_image_size,3)
       ))
66     classifier.add(Lambda(lambda image: preprocess_input(tf.image.resize(
       image, (299, 299)))))
67     classifier.add(base_model)
68     classifier.add(Flatten())
69     classifier.add(Dense(128, activation='relu'))
70     classifier.add(Dropout(0.5))
71     classifier.add(Dense(10, activation='softmax'))
72     classifier.compile(optimizer='adam', loss='categorical_crossentropy',
       metrics=['accuracy'])
73     classifier.fit(x_train, to_categorical(y_train),
74                    epochs=EPOCHS,
75                    batch_size=BATCH_SIZE,
76                    validation_split=0.4,
77                    callbacks=[keras.callbacks.EarlyStopping(patience=3,
       restore_best_weights=True)])
78
79     myclassifier=MyClassifier(classifier,original_image_size=
       original_image_size,is_dataset_rgb=True)
80
81     start_pred = current_ms()
82     y_pred = myclassifier.predict(x_test)
83     clf_time = (current_ms() - start_pred) / len(y_test)
84     y_proba = myclassifier.predict_proba(x_test)
```