Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Detenzione di Esempi Fuori Distribuzione in una Rete Neurale Addestrata

Candidato
Alice Casali

Relatore
Prof. Andrew Bagdanov

Anno Accademico 2017/2018

Se puoi sognarlo, puoi farlo.
Ricorda sempre che questa meravigliosa avventura
è partita da un topolino.
Walt Disney

Indice

Introduzione

Negli ultimi anni, molti miglioramenti sono stati fatti nel campo del Machine Learning. Infatti, è aumentato notevolmente l'impiego di tecnologie per la classificazione immagini nel mondo reale tramite reti neurali. Il problema sta nel fatto che questi classificatori tendono a fallire quando le immagini da classificare nel test, appartengono a classi diverse da quelle usate durante l'addestramento.

Queste immagini fuori distribuzione vengono comunque classificate. Quello che invece ci si dovrebbe aspettare dalla rete è una completa incertezza.

L'importanza di risolvere questo problema sta nella enorme utilità che questa tecnologia può offrire. Infatti, il riconoscimento delle immagini viene sempre più utilizzato nelle nuove tecnologie, basti pensare al riconoscimento facciale che tutti i nuovi smartphone hanno impiegato.

Uno dei motivi per cui ancora non vengono messe in commercio le auto self-driving è proprio il problema della classificazione delle immagini fuori distribuzione. Infatti, quando si presenta un caso limite il classificatore si comporta in maniera inaspettata e questo può portare anche a fatali collisioni [15]. Come l'incidente fatale della Tesla del 23 marzo 2018, dove l'auto in modalità self-driving si è schiantata contro lo spartitraffico in autostrada, a causa delle righe bianche di segnalazione non ben marcate.

E' stato riscontrato anche un uso scorretto del problema l'Adversarial

Network Attacck; studi recenti hanno mostrato che le reti neurali sono vulnerabili alla perturbazione delle immagini [1]. Infatti un mutamento dell'immagine può portare una rete ben addestrata a mal classificarla. [8] Questo mutamento può essere reso invisibile all'occhio umano ma, la rete classificherà l'immagine in maniera errata. Questo è un punto critico per la sicurezza.

La tesi si concentra sull'aspetto di detenzione della classificazione sbagliata di un elemento fuori distribuzione. Il documento sarà così strutturato:

- Il capitolo 1 introduce il concetto di Multilayer Perceptron e Rete Neurale Convoluzionale. Inoltre descrive il lavoro relativo al problema che è già stato effettuato.
- Il capitolo 2 spiega l'approccio che è stato utilizzato per effettuare le sperimentazioni e i metodi matematici applicati.
- Il capitolo 3 mostra gli esperimenti fatti tramite il codice e i risultati ottenuti.
- Il capitolo 4, infine, analizza i risultati traendo conclusioni e idee per progetti futuri.

Capitolo 1

Lavoro Relativo

In questo articolo troviamo la panoramica degli studi che sono stati fatti sulla classificazioni di immagini Out of Distribution (OOD).

Prima però è necessario introdurre il Multilayer Perceptron e le Reti Neurali Convoluzionali, al fine di comprendere meglio gli esperimenti esposti all'interno della tesi.

1.1 Tecnologie Utilizzate

1.1.1 Multilayer Perceptron

Il Multilayer Perceptron (MLP) è una rete neurale di apprendimento supervisionato. Questa rete è composta da più di un perceptron. Ciascuno è formato da un layer di input per ricevere dati, e un layer di output che fa predizioni sull'input ricevuto, in mezzo a questi c'è un numero arbitrario di layer nascosti che sono la vera forza computazionale del MLP.

Abbiamo utilizzato MLP per processare il dataset MNIST, insieme di immagini di numeri scritti con varie calligrafie.

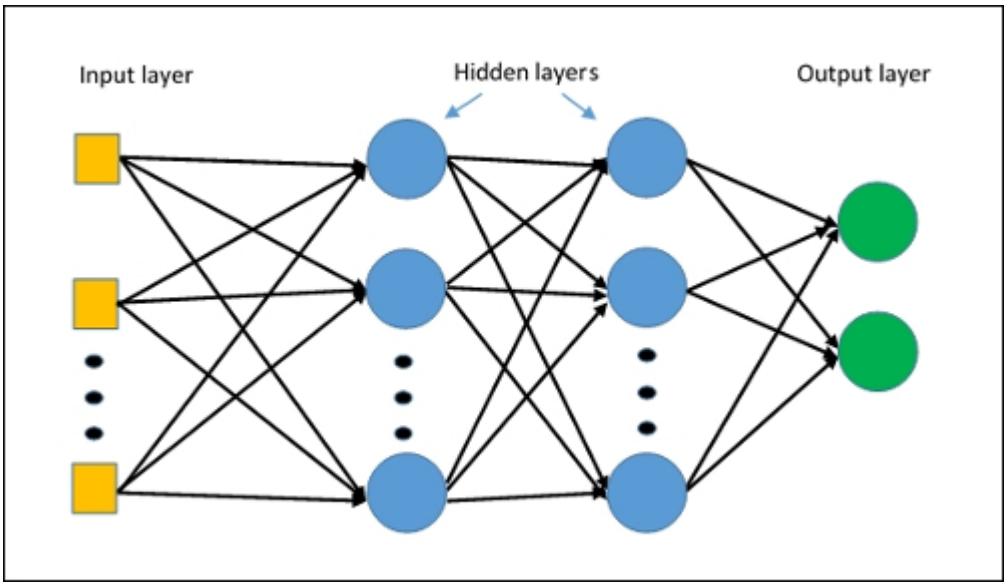

Figura 1.1: Esempio di Multilayer Perceptron

1.1.2 Reti Neurali Convoluzionali

Le Reti Neurali Convoluzionali (CNN) sono una categoria delle reti neurali che si sono dimostrate davvero efficaci nel campo del riconoscimento immagini e classificazione.

Il tradizionale MLP viene eseguito con successo per il riconoscimento delle immagini. Quando c'è completa connessione tra i nodi e in input ci sono immagini con alta risoluzione, MLP non è in grado di fare un'elaborazione efficiente. Ad esempio nel dataset CIFAR-10 dove ci sono 10 classi di immagini $32x32$.

Per questo sono state introdotte le CNN, esse sono una variante del MLP ispirate al comportamento della corteccia celebrale degli esseri umani. Le CNN hanno le seguenti funzionalità, in più rispetto a MLP [13]:

- Le CNN hanno neuroni con 3 dimensioni: altezza larghezza e profondità. I neuroni di un layer sono connessi solo con una piccola regione

di layer che lo precede: detto receptive field. Tipi distinti di layer sono sia locali sia completamente connessi per fromare l'architettura CNN.

- Connettività Locale: seguendo il concetto del receptive field, le CNN implementano una connettività locale tra neuroni di layer adiacenti. L'architettura garantisce così che i "filtri" appresi producano una risposta più forte a un modello di input spazialmente locale. L'impilamento di molti di questi livelli porta a filtri non lineari che diventano sempre più globali, così la rete inizialmente fa una rappresentazione di piccole parti dell'input, poi da queste assembla una rappresentazione dell'intera area.

- Pesi Condivisi: nelle CNN ciascun filtro è replicato attraverso l'intero campo visivo. Queste repliche riunite condividono la stessa parametrizzazione (vettore dei pesi e bias) e formano una feature map. Questo significa che tutti i neuroni in un dato layer convoluzionale rispondono la stessa feature senza il loro specifico campo di risposte. La replica delle unità in questo modo permette per le feature, di essere rilevate senza la loro posizione nel campo visivo, questo costituisce la proprieta del translation invariance.

Insieme queste proprietà permettono alle CNN di raggiungere al meglio la generalizzazione nei problemi visivi, la rete può fare un addestramento più ampio e risultare più potente.

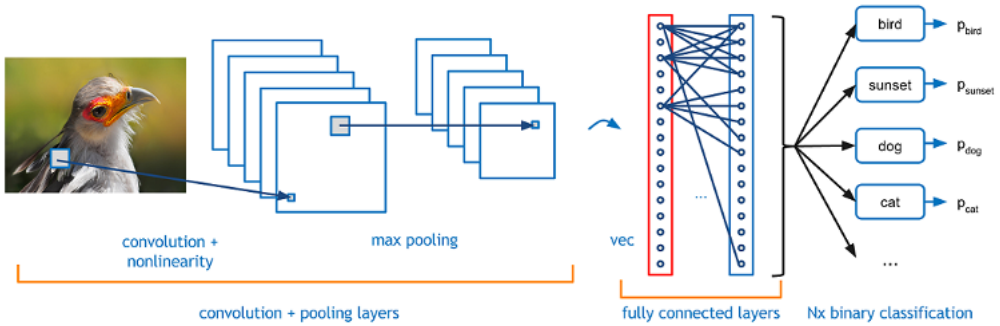

Figura 1.2: Esempio di classificazione in una Rete Neurale Convoluzionale

1.2 Panoramica del lavoro svolto

Nel 2017 [7] è stato dimostrata la baseline di predizioni sulle probabilità ricavate tramite softmax e la detenzione di esempi OOD su diverse architetture e numerosi dataset. Successivamente hanno presentato un modulo dei comportamenti con dati fuori distribuzione, che prevede punteggi superiori per la discriminazione tra esempi in distribuzione e fuori distribuzione nel test. Questo modulo dimostra che la baseline può essere battuta in qualche caso e questo implica che c'è spazio per una futura ricerca.

Nel 2018 [9] è stato proposto un modo semplice per la detenzione di immagini OOD nelle reti neurali. Non è necessario in questo metodo riaddestrare la rete e migliorare la baseline su architetture neurali differenti su differenti dataset. Bensì si analizza il metodo sotto differenti parametri impostati e si provvede a alcuni accorgimenti dietro l'approccio.

Capitolo 2

Il nostro approccio

Questo capitolo descrive tramite quali algoritmi abbiamo affrontato il problema della detenzione di immagini OOD.

In particolare si differenziano tre particolari approcci per l'estrazione delle predizioni dalle probabilità restituite dalla rete durante i test: probabilità massima, entropia minima o incertezza bayesiana.

2.1 Probabilità massima come indicatore di In-Distribution

Una volta che la rete è stata addestrata su un determinato dataset, si eseguono dei test con immagini nuove.

Queste immagini entrano in input nella rete e vengono classificate tramite dei punteggi, questi punteggi vengono convertiti in probabilità.

Per ogni tupla di probabilità, il cui numero n di elementi sarà uguale al numero di classi, si estrae la probabilità massima:

$$P_{max} = \max \{P_1, \ldots, P_n\}$$

Questa probabilità indica la classe che la rete ha scelto di assegnare all'immagine in input. Questo procedimento avviene per ogni immagine in input durante il test. La raccolta di queste predizioni ci porterà a concludere con quanta precisione la rete classifica le immagini.

Nel caso di immagini OOD ci dovremmo aspettare una variazione minima tra le probabilità in uscita, anche se non c'è mai completa incertezza come vedremo negli esperimenti.

2.2 Entropia minima come indicatore di In-Distribution

In questo caso, quando si esegue il test, invece che estrarre semplicemente la massima probabilità risultante si calcola l'entropia minima per ciascuna tupla.

Vediamo adesso nel particolare perché si utilizza l'entropia in questo specifico frangente.

In generale, nell'informatica teorica, l'entropia di una sorgente di messaggi è l'informazione media contenuta in ogni messaggio emesso. L'informazione contenuta in un messaggio è tanto più grande quanto meno probabile era. Un messaggio scontato, che ha un'alta probabilità di essere emesso dalla sorgente contiene poca informazione, mentre un messaggio inaspettato, poco probabile contiene una grande quantità di informazione.

- Informazione contenuta in un evento: l'informazione contenuta in un evento x emesso da una sorgente X, detta anche autoinformazione si calcola come:

$$I(x) = -\log_b \mathbb{P}(x)$$

Dove $P(x)$ è la probabilità che l'evento x accada. Il logaritmo nasce dal fatto che attraverso la notazione posizionale è possibile distinguere N eventi equiprobabili con l'utilizzo di sole $\log_b N$ cifre, dove b è la base di numerazione.

Significa quindi che l'informazione di un evento può essere vista come la quantità di cifre in base b da utilizzare per distinguere l'evento accaduto da tutti gli altri eventi possibili. Il logaritmo diventa indispensabile se considerando due eventi indipendenti la cui probabilità è il prodotto delle singole probabilità si vuole che l'entropia totale sia la somma delle entropie dei singoli eventi.

Dato che spesso in informatica si usano i bit per codificare è convenzione comune utilizzare il logaritmo in base 2.

- Entropia di una sorgente: l'entropia di una sorgente è definita come il valore atteso dell'autoinformazione, ovvero l'informazione media contenuta in ogni messaggio emesso:

$$\mathbb{H}[X] = \mathbb{E}[I(X)] = \mathbb{E}[-\log_b \mathbb{P}(X)]$$

Si può intendere come il numero di cifre in base b da utilizzare in media per memorizzare un singolo evento prodotto dalla sorgente.

Se la sorgente X è una variabile aleatoria discreta il valore atteso si riduce ad una media dell'informazione di ogni evento x_i pesata con la propria probabilità $\mathbb{P}(x_i)$

$$\mathbb{H}[X] = -\sum_{i=1}^{N} \mathbb{P}(x_i) \cdot \log_b \mathbb{P}(x_i)$$

L'entropia di una variabile aleatoria continua non condivide tutte le proprietà di quella per variabili discrete, ad esempio $\mathbb{H}[X]$ può risultare negativa.

La probabilità degli eventi emessi può dipendere dallo stato della sorgente o dagli eventi emessi in precedenza, in tal caso si dice che la sorgente ha memoria. L'entropia delle sorgenti con memoria è ragionevolmente minore dell'entropia di una sorgente senza memoria. Infatti gli eventi emessi dipendono in una certa misura da quelli emessi precedentemente, il che li rende più prevedibili.

Quello che ci aspettiamo dunque sarà un'entropia bassa quando il classificatore sarà certo della sua classificazione, e un'entropia alta quando ci sarà maggior incertezza. Le immagini OD dovrebbero sempre avere entropia alta, cosa che invece non si verifica come vedremo negli esperimenti.

2.2.1 Temperature scaling

Prima di parlare del Temperature Scaling dobbiamo necessariamente introdurre la funzione Softmax.

In matematica, una funzione Softmax, o funzione esponenziale normalizzata, è una generalizzazione di una funzione logistica che comprime un vettore k-dimensionale z di valori reali arbitrari in un vettore k-dimensionale $\sigma(z)$ di valori compresi in un intervallo $(0, 1)$ la cui somma è 1. La funzione è data da:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{per } j = 1, \ldots, K.$$

La funzione softmax è spesso usata nello strato finale dei classificatori basati su reti neurali.

Nel campo dell'apprendimento per rinforzo, una funzione softmax può essere usata per convertire valori in probabilità, è qui che entra in gioco il Temperature Scaling.

La temperatura è un parametro usato per controllare la randomicità delle predizioni scalando i punteggi (scores) del test prima di applicare il softmax.

Quando la temperatura è 1, si esegue il softmax direttamente sui punteggi, usando la temperatura di 0.8 il modello esegue il softmax su *scores* 0.8. Facendo eseguire il softmax su temperature più alte produce una distribuzione di probabilità più smorzata tra le classi.

La rete rende le probabilità delle classi con un vettore di punteggi $z = (z_1, \ldots, z_n)$ eseguendo la funzione softmax per ottenere il vettore probabilità $q = (q_1, \ldots, q_n)$ confrontando z_i (il punteggio corrente) con gli altri punteggi.

$$\sigma(z)_j = \frac{e^{\frac{z_j}{\tau}}}{\sum_{k=1}^{K} e^{\frac{z_k}{\tau}}} \quad \text{per } j = 1, \ldots, K.$$

2.3 Incertezza Bayesiana via dropout

Per rendere lo studio più completo dobbiamo utilizzare un modello che calcoli l'incertezza in maniera più specifica. [14] La probabilità Bayesiana ci offre un modello matematico per poter considerare l'incertezza, questo solitamente a livello computazionale è molto costoso.

Possiamo interpretare il dropout all'interno della sequenza dei livelli della rete come modello di approssimazione Bayesiana per l'incertezza.

Per poter comprendere al meglio il concetto è bene fare chiarezza sul significato del Dropout. [2] Esso consiste nell'ignorare alcuni nodi della rete all'interno di un certo insieme durante la fase di addestramento. Essi sono scelti casualmente. Per ignorati si intende che non vengono considerati durante una particolare fase di forward o backward.

Questa tecnica viene impiegata per evitare il problema dell'over-fitting. [4] Si parla di overfitting quando un modello statistico molto complesso si adatta

ai dati osservati (il campione) perché ha un numero eccessivo di parametri rispetto al numero di osservazioni.

Nel caso in cui l'apprendimento sia stato effettuato senza dropout, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; perciò, in presenza di overfitting, le prestazioni (cioè la capacità di adattarsi/prevedere) sui dati di allenamento aumenteranno, mentre le prestazioni sui dati non visionati saranno peggiori.

2.4 Avarage Precision Score

Per raggiungere il nostro obiettivo dobbiamo dunque classificare delle immagini assegnandole alla classe corrispondente (multilabel classification). Per fare ciò servirà implementare l'Avarage Precision Score (AP).

Prima di spiegare il concetto dell'Avarage Precision Score è necessario introdurre l'Information Retrieval. L'IR è l'insieme delle tecniche utilizzate per gestire la rappresentazione, la memorizzazione, l'organizzazione e l'accesso ad oggetti contenenti informazioni quali documenti, pagine web, cataloghi online e oggetti multimediali.

Lo scopo dell'IR è di soddisfare il cosiddetto "bisogno informativo dell'utente", ovvero garantire a quest'ultimo, in seguito ad una sua ricerca, i documenti e le informazioni che rispondono alla sua richiesta.

Due concetti sono di fondamentale importanza per analizzare un sistema di IR: query ed oggetto.

Le query ("interrogazioni") sono stringhe di parole-chiavi rappresentanti l'informazione richiesta. Vengono digitate dall'utente in un sistema IR (per

esempio, un motore di ricerca) e sono la concretizzazione del reale bisogno informativo dell'utente.

Un oggetto è un'entità che possiede informazioni le quali potrebbero essere risposta dell'interrogazione dell'utente. Un documento di testo, per esempio, è un oggetto di dati.

La valutazione di un sistema IR è il processo che decide quanto bene il sistema provvede alle informazioni necessarie ai suoi utenti. La valutazione metrica tradizionale, strutturata per i Boolean retrieval, include il precision e recall.

- *Precision*: è la proporzione di documenti pertinenti fra quelli recuperati:

  $$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

  Nella classificazione binaria la precisione è analoga al valore positivo di previsione. La precisione può anche essere valutata rispetto a un certo valore soglia, indicato con $P@n$, piuttosto che relativamente a tutti i documenti recuperati: in questo modo, si può valutare quanti fra i primi n documenti recuperati sono rilevanti per la query.

- *Recall*: è la proporzione fra il numero di documenti rilevanti recuperati e il numero di tutti i documenti rilevanti disponibili nella collezione considerata:

  $$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

  Nella classificazione binaria, questo valore è chiamato sensitività.

Dunque a questo punto possiamo introdurre il concetto di AP. [3] Nel calcolo dell'Average Precision l'idea è di combinare assieme il valore di Precision e l'ordinamento fatto dall'RI. [5] Sia $n = |Retr|$ il numero di documenti reperiti e $h[i]$ l'i-esimo documento reperito. Definiamo $Rel[i]$ uguale a 1 se $h[i]$ è

pertinente e 0 altrimenti. Infine, sia $|Rel|$ il numero totale di documenti pertinenti rispetto all'interrogazione. Possiamo, con questa notazione, formulare la definizione di Precision al documento j come:

$$P@j = \sum_{k=1}^{j} \frac{Rel[k]}{j}$$

Possiamo quindi definire l'Average Precision:

$$AP = \sum_{j=1}^{n} \frac{P@j \cdot Rel[j]}{|Rel|}$$

Average Precision risulta quindi essere la somma dei valori di Precision calcolati quando ogni documento pertinente è reperito. La somma è normalizzata per il numero totale di documenti pertinenti presenti nella collezione.

Dunque per esempio, se tutte le immagini processate dalla rete avessero probabilità 1.0 di appartenere a una classe: si avrebbe una linea retta nel plot.

Quello che ci dovremmo aspettare è una buona percentuale di classificazione per immagini ID è una cattiva percentuale di classificazione per immagini OD.

Invece come vedremo nelle sperimentazioni, non averrà una differenziazione netta tra immagini ID e OD.

Capitolo 3

Esperimenti condotti

In questo capitolo viene illustrata la sperimentazione svolta al fine di rilevare gli esempi *out of distribution* tramite reti neurali.

Di seguito si trovano i dettagli dell'implementazione, i problemi riscontrati (con relative soluzioni adottate) e i risultati dei test effettuati. Vengono inoltre presentati tutti i tools adoperati nel progetto, infatti, abbiamo bisogno di librerie specifiche per l'implementazione della rete.

3.1 Aspettative

Ci aspettiamo che con MNIST essendo un dataset semplice la rete dia punteggi più alti per immagini ID ma che comunque in casi limite classificherà immagini OD come se fossero ID dando punteggi più alti e meno incerti.

Per CIFAR10 ci aspettiamo che il classificatore sia meno preciso essendo le immagini più complesse. Dunque darà punteggi alti alle immagini OD non solo in casi limite, classificandole come ID.

Per evidenziare questo errore di classificazione ricorriamo all'Avarage Precision Score. Più il valore dell'AP sarà basso più il classificatore sarà incerto

nella classificazione, dunque, più immagini OD avranno punteggi simili a immagini ID.

3.2 Ambiente di esecuzione e Tools

Il primo importante compito in qualsiasi progetto implementativo è quello di trovare, e soprattutto scegliere, gli strumenti su cui lavorare. La scelta deve tenere conto delle funzionalità offerte, della frequenza degli aggiornamenti e della dimensione della community, a livello mondiale, relativa.

3.2.1 Pytorch

Il tools utilizzato principalmente è la libreria Pytorch [10]. E' una libreria open-source per il machine learning, basata su Torch, usata per le applicazioni come il natural language processing. Provvede due features di alto livello:

- Computazione coi tensori con una forte accelerazione tramite GPU.
- Reti Neurali costruite su sistemi tape-based autodiff.

3.2.2 Scikit-Learn

Un'altra libreria che ho utilizzato è Scikit-Learn [6]. E' una libreria open-source di apprendimento automatico per il linguaggio di programmazione Python.

Contiene algoritmi di classificazione, regressione e clustering (raggruppamento) e macchine a vettori di supporto, regressione logistica, classificatore bayesiano, k-mean e DBSCAN, ed è progettato per operare con le librerie NumPy e SciPy.

3.2.3 Jupyter Notebook

Jupyter Notebook è una applicazione web open-source che permette di creare e condividere documenti che contengono codice, equazioni, visualizzazioni e testo narrativo.

Gli usi inclusi sono: il data cleaning e la trasformazione, simulazione numerica, modelli statistici, visualizzazione dati e machine learning. Supporta più di 40 linguaggi tra cui Python.

L'analisi ed elaborazione di dati è sempre stato il punto di forza (oltre al motivo della sua recente ascesa) di Python ed essendo il linguaggio più versatile e ampiamente utilizzato in machine learning, la scelta è ricaduta su di esso.

3.3 Dataset Utilizzati

La scelta dei dataset per effettuare gli esperimenti si è rivelata più difficile del previsto. Sono stati selezionati in base alla quantità di classi presenti e alla quantità di immagini per ciascuna classe.

Inoltre è risultato importante fare una distinzione nella sperimentazione tra immagini piccole in bianco e nero, e immagini leggermente più grandi a colori.

Il motivo è la complessità rilevante di un dataset rispetto a un altro che ha reso necessario l'uso di reti diverse e più complesse.

3.3.1 MNIST

Il dataset più semplice a cui si può ricorrere per la classificazione di immagini è MNIST (Modified National Institute of Standards and Technology database).

Esso è un grande database di caratteri numerici scritti a mano che è comunemente usato per addestramento e testing nel campo del machine learning. E' un sottoinsieme dell'originale dataset NIST che non era adatto per gli esperimenti sul machine learning.

Il dataset MNIST è composto da immagini da 28x28 pixel. Contiene 60,000 immagini per il training e 10,000 immagini per il testing.

Figura 3.1: Insieme di immagini tratte da MNIST

3.3.2 CIFAR10

Per utilizzare un dataset più complesso ricorriamo a CIFAR10. Abbiamo escluso CIFAR100 perchè richiedeva troppa potenza di calcolo.

CIFAR10 consiste in 60,000 immagini a colori 32x32 divise in 10 classi con 6,000 immagini ciascuna. Ci sono 50,000 immagini dedicate all'addestramento della rete e 10,000 immagini per i test *in distribution*. Saremo noi a

modificare gli split al fine di valutare cosa avviene con test di immagini *out of distribution*.

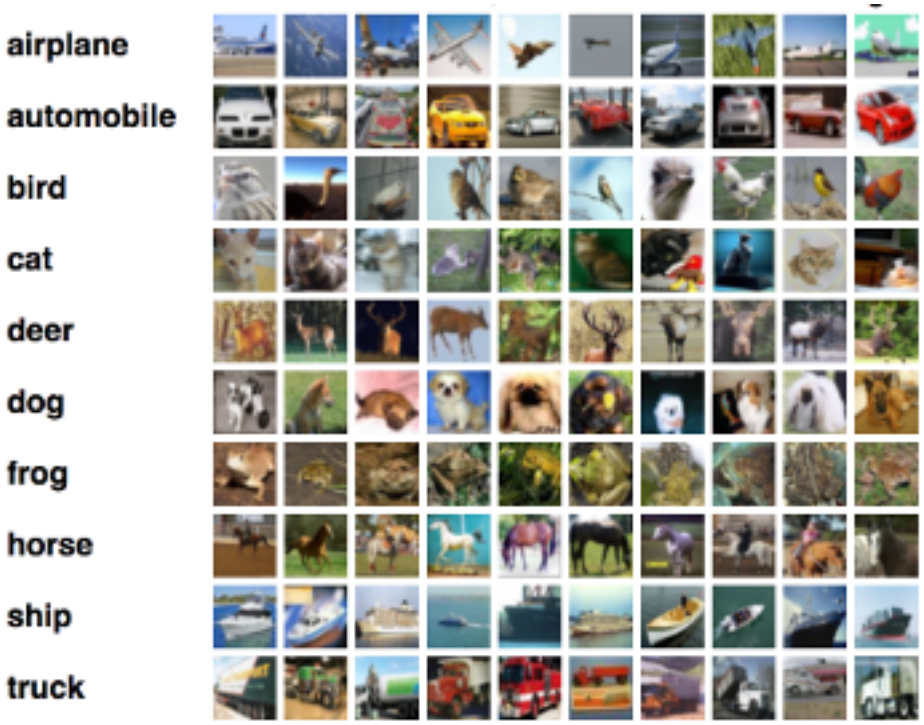

Figura 3.2: In figura l'elenco delle classi di CIFAR10 con relativi esempi

3.4 Protocollo Sperimentale

Le sperimentazioni svolte consistono in due studi principalmente. In uno si esegue l'addestramento e i test della rete Multilayer Perceptron con il dataset MNIST. Nell'altro si utilizza una rete Neurale Convoluzionale con dataset CIFAR 10.

In entrambi l'iter è lo stesso: si addestra la rete su una porzione del dataset, si eseguono test su dati della stessa parte di dataset (dati *in distribution*), poi si eseguono test con la restante parte di dati (dati *out of distribution*).

Nell'esecuzione dei test in entrambi si estraggono le predizioni sia con il calcolo della probabilità massima, sia con l'entropia di Shannon.

Essendo la rete che analizza CIFAR10 più complessa, si esegue anche un test con incertezza bayesiana come punteggio.

3.4.1 Split dei Dataset

Per poter fare il nostro studio è stato necessario effettuare uno split dei dataset in modo da eseguire test su immagini *in distribution* (ID) e *out of distribution* (OD).

Lo split è necessario perché una volta addestrata la rete su un determinato insieme di immagini, facendo il test ID, si avrà una percentuale di riconoscimento alta. Quando andremo a fare il test OD, che quindi prenderà in considerazione immagini appartenenti a classi mai considerate, valuteremo come reagirà il classificatore. Questo è appunto l'obiettivo di questa sperimentazione.

Vediamo come sono stati divisi entrambi i dataset:

- **MNIST**: il train e i test ID sono stati eseguiti sulle classi di immagini da 0 a 6. I test OD sono stati eseguiti invece sulle classi da 7 a 9.
- **CIFAR10**: il train e i test ID sono stati eseguiti sulle classi di immagini da 0 a 4. I test OD sono stati eseguiti invece sulle classi da 5 a 9.

3.4.2 Struttura dei classificatori

Vediamo adesso come sono stati costruiti i classificatori, mettendo in evidenza la semplicità del primo rispetto al secondo.

- Per analizzare MNIST è stato sufficiente utilizzare un Multilayer Perceptron con un unico layer nascosto.

Listing 3.1: MLP

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, 128)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(128, 7)


    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        return F.log_softmax(self.fc3(x), 1)
```

ReLu è la funzione di attivazione, FC è il layer completamente connesso (fully connected) e il SoftMax è la funzione di attivazione dell'output layer.

- Per CIFAR10 è stato necessario utilizzare una rete più complessa: una Rete Neurale Convoluzionale con due layer convoluzionali.

Listing 3.2: CNN

```
class Net(nn.Module):
    def __init__(self, dropout=0.0):
        super(Net, self).__init__()
```

```
        self.conv1_1 = nn.Conv2d(3, 64, 3)
        self.conv1_2 = nn.Conv2d(64, 64, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2_1 = nn.Conv2d(64, 64, 3)
        self.conv2_2 = nn.Conv2d(64, 64, 3)
        self.drop = nn.Dropout(0.5)
        self.fc1 = nn.Linear(64 * 5 * 5, 256)
        self.fc2 = nn.Linear(256, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = F.relu(self.conv1_1(x))
        x = self.pool(F.relu(self.conv1_2(x)))
        x = F.relu(self.conv2_1(x))
        x = self.pool(F.relu(self.conv2_2(x)))
        x = x.view(-1, 64 * 5 * 5)
        x = self.drop(F.relu(self.fc1(x)))
        x = self.drop(F.relu(self.fc2(x)))
        z = self.fc3(x)
        x = F.log_softmax(z, 1)
        return (x, z)
```

ReLu è una funzione di attivazione, MaxPool è un pooling lyer, FC è un layer completamente connesso e il SoftMax è la funzione di attivazione del layer di output.

3.4.3 Addestramento Rete

L'addestramento della rete viene fatto in entrambi i casi con funzioni analoghe, ne vediamo il codice sia del training che del validation test.

Listing 3.3: Funzione di training

```
def train(epoch, loader, optimizer):
    model.train()
    print('Training epoch {}...'.format(epoch))
    total_loss = 0.0
    for (data, target) in loader:
        target = torch.tensor(target)
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        (output, _) = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        total_loss += loss
    print('Average loss: {}'.format(total_loss / len(loader)))
```

Listing 3.4: Validation test

```
def validate(loader):
    model.eval()
    val_loss, correct = 0, 0
    for data, target in loader:
        target = torch.tensor(target)
        if cuda:
```

```
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        (output, _) = model(data)
        val_loss += F.nll_loss(output, target).item()
        pred = output.data.max(1)[1]
        correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(loader)
    accuracy = 100. * correct / len(loader.dataset)
    print('Validation set: Average loss: {:.4f},
        Accuracy: {}/{} ({:.0f}%)\n'.format(
        val_loss, correct, len(loader.dataset), accuracy))
```

3.4.4 Test effettuati

I test vengono eseguiti nello stesso modo in MNIST e CIFAR10 in modo da rendere i risultati coerenti. Su CIFAR10 si esegue anche un test in più utilizzando l'incertezza bayesiana.

Listing 3.5: Funzione di Test

```
def max_test(epoch, loader):
    model.eval()
    test_loss = 0
    correct = 0
    preds_list = []
    for data, target in loader:
        target = target.type(torch.long)
        if cuda:
```

```
            data, target = data.cuda(), target.cuda()
        data = Variable(data)
        target = Variable(target)
        output = model(data)
        preds_list.append(output)
        test_loss += F.nll_loss (output, target).item ()
        pred = output.max (1, keepdim=True)[1]
        correct += pred.eq (target.view_as
        (pred)).sum ().item ()
    max_pred = new_predict(preds_list)
    test_loss /= len (loader.dataset)
    print ('\nTest set: Average loss: {:.4f},
        Accuracy: {}/{} ({:.0f}%)\n'.format (
        test_loss, correct, len (loader.dataset),
        100. * correct / len (loader.dataset)))
    return max_pred
```

- In un caso si calcola la predizione massima e si colleziona gli output, dopo averli convertiti in probabilità. Poi si estrae il maggiore. La predizione maggiore indica la classe di appartenenza dell'immagine.

```
idness = np.max(preds_list, 1)
```

- Nell'altro caso si calcola l'entropia minima su tutta la tupla degli output. Più è bassa l'entropià più la rete è certa della sua classificazione. Per poter calcolare l'AP è necessario mettere un meno davanti al calcolo dell'entropia minima.

```
entropy = (-entr(preds_list)).sum(1)
```

- Inizialmente era stato utilizzato anche il Temperature Scaling su CIFAR10 ma i risultati erano molto simili a quelli dati dall'entropia, quindi abbiamo deciso di non considerarne i risultati.

- Il test con incertezza bayesiana via dropout viene eseguito solamente su CIFAR10, su MNIST non serve perché è un dataset molto più semplice. Ogni immagine viene processata dalla rete 50 volte, si calcola la varianza sugli output e infine si fa la media.

$$score = \frac{1}{N} \sum_{i=1}^{N} [\sigma^2 \sigma^2 \dots \sigma^2]$$

Listing 3.6: Test con bayesian uncertainty via dropout

```
num = 50
probs_ID = np.stack([predict(model, test_ID,
dropout=True)
for i in range(num)])
id_bayes = probs_ID.var(0).mean(1)
probs_OOD = np.stack([predict(model, test_OOD,
dropout=True)
for i in range(num)])
ood_bayes = probs_OOD.var(0).mean(1)
```

3.4.5 Valutazione con Avarage Precision Score

Per effettuare il calcolo dell' AP [12] inseriamo in y_true gli indicatori binari ovvero tanti 1 quante sono le predizioni ID, tanti 0 quante sono le predizionei OD. In y_score inseriamo la lista delle predizioni ricavate dai test sia ID che OD.

Listing 3.7: Valutazione AP

```
## Raccolgo le predizioni max o con entropia in y_scores
y_scores =np.concatenate((resID, resOOD), axis = None)

## Raccolgo le etichette, agli ID assegno 1,
##agli OD assegno 0
y_true = np.concatenate((pred_ID, pred_OOD), axis = None)

AP = average_precision_score(y_true, y_scores)
plot_func(y_true, y_scores, AP)
```

3.5 Risultati

I risultati che mettiamo in evidenza sono l'accuratezza del classificatore e il grafico dell'Avarage Precision Score.

3.5.1 MNIST tramite MLP

Eseguiamo un addestramento di 10 epoche, l'accuratezza del classificatore è del 98% Prima analizziamo i risultati calcolati raccogliendo le predizioni tramite il calcolo della *predizione massima*.

L'Avarage Precision Score risulta essere: $AP = 0.9670$

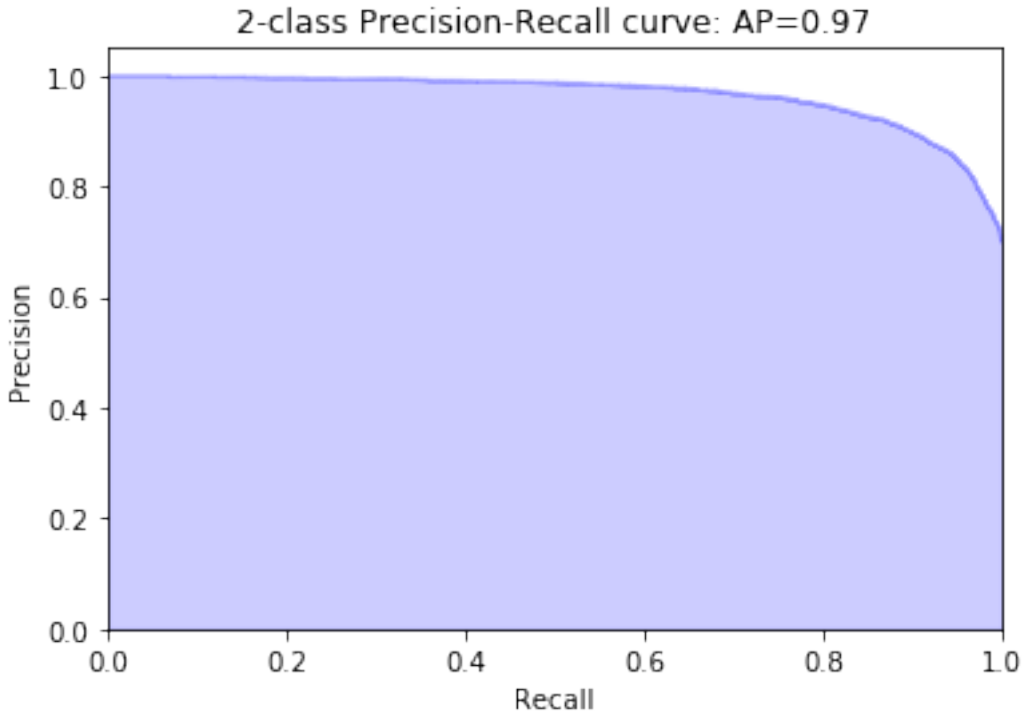

Figura 3.3: AP con predizione massima

Adesso analizziamo i risultati calcolati raccogliendo le predizioni tramite il calcolo della *entropia minima*.

L'Avarage Precision Score risulta essere: $AP = 0.9680$

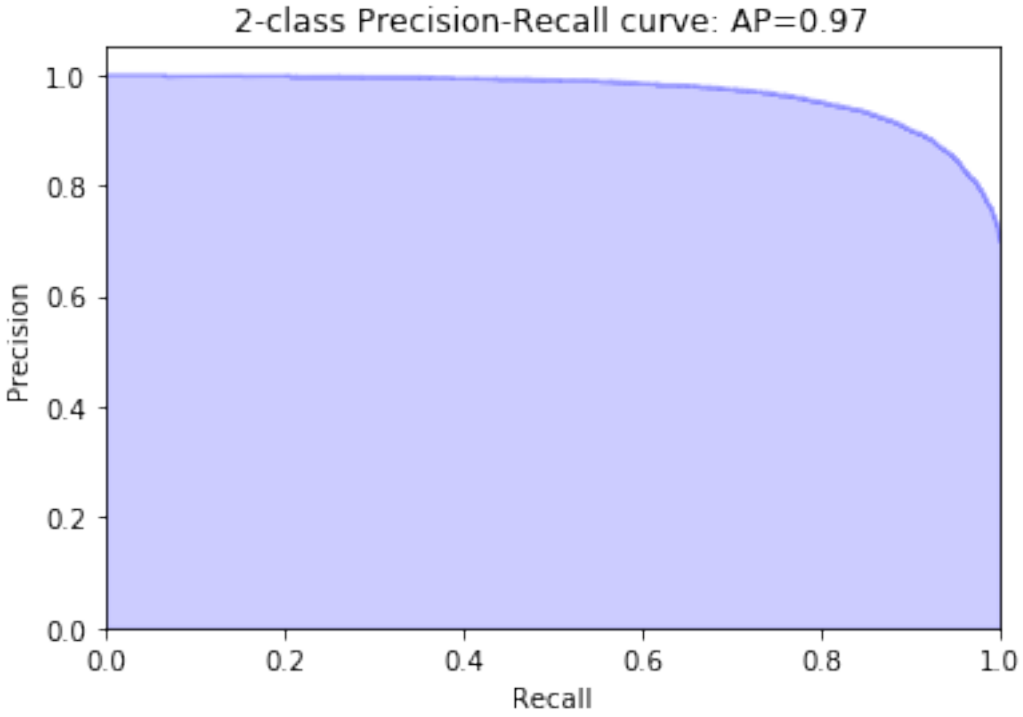

Figura 3.4: AP con entropia

Anche se minima, c'è una differenza tra gli AP:

$AP_{en} - AP_{max} = 0.0010$

3.5.2 CIFAR10 tramite CNN

Eseguiamo un addestramento di 20 epoche, l'accuratezza del classificatre è del 84% Prima analizziamo i risultati calcolati raccogliendo le predizioni tramite il calcolo della *predizione massima*.

L'Avarage Precision Score risulta essere: $AP = 0.60$

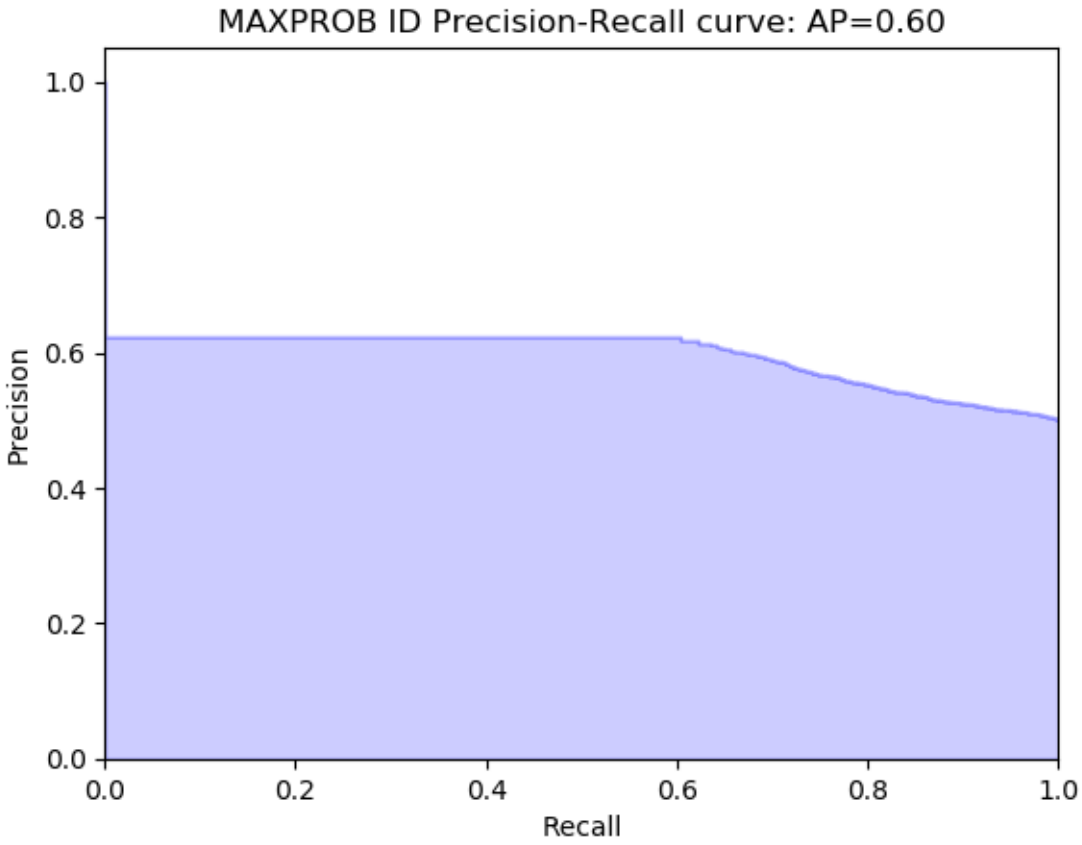

Figura 3.5: AP con predizione massima

Adesso analizziamo i risultati calcolati raccogliendo le predizioni tramite il calcolo della *entropia*.

L'Avarage Precision Score risulta essere: $AP = 0.65$

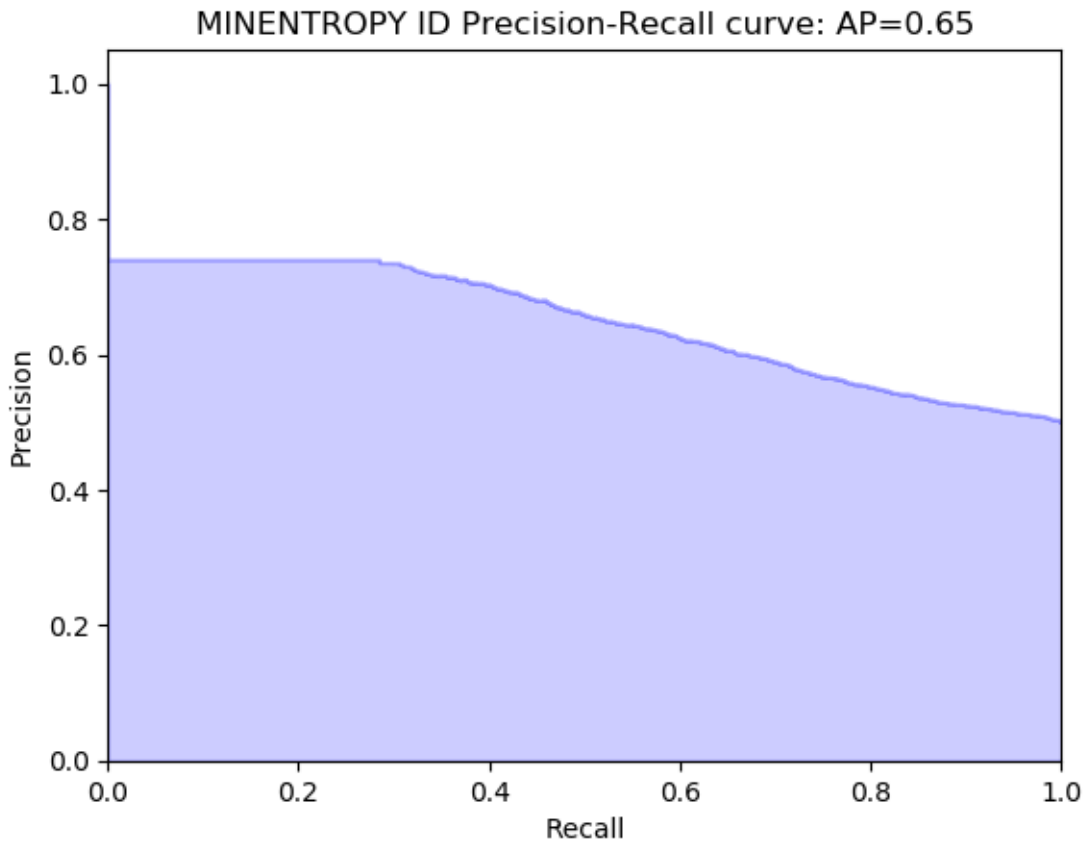

Figura 3.6: AP con entropia

Adesso analizziamo i risultati calcolati raccogliendo le predizioni tramite il calcolo della *bayesian uncertainty via dropout*.

L'Avarage Precision Score risulta essere: $AP = 0.67$

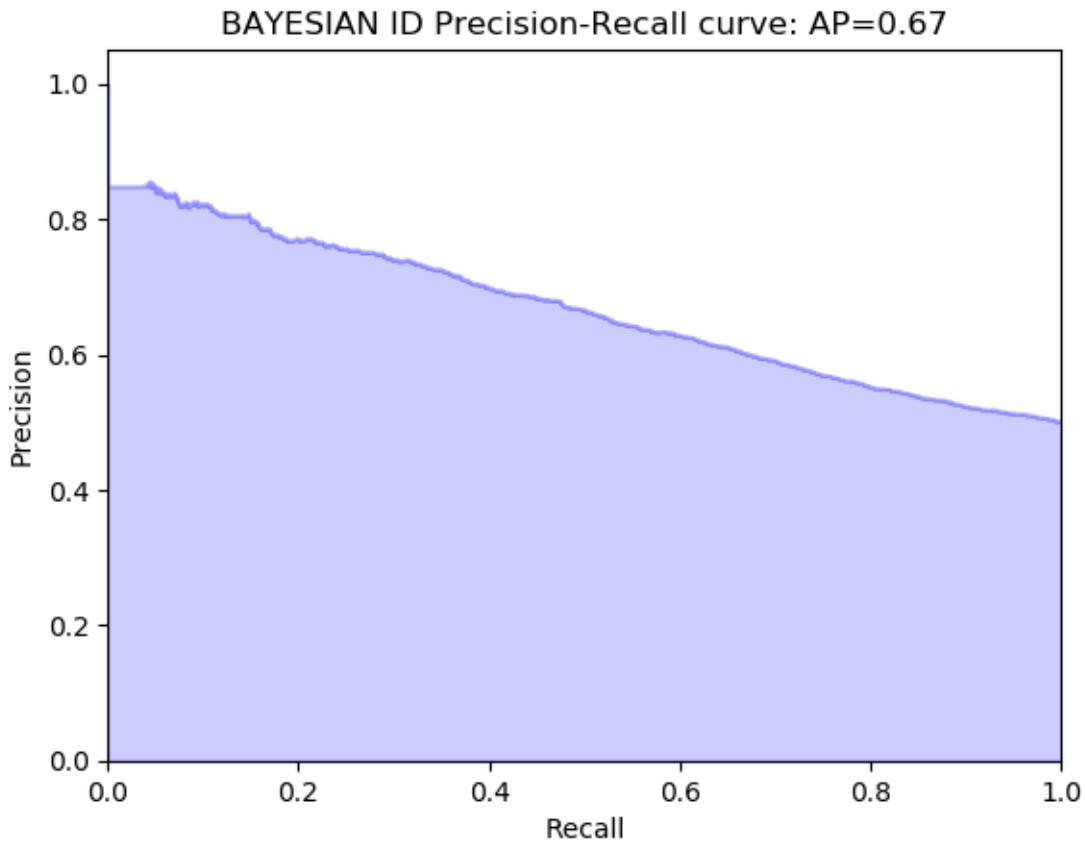

Figura 3.7: AP con bayesian

Capitolo 4

Conclusioni e Lavori Futuri

In quest'ultimo capitolo vengono presentate le conclusioni dello studio effettuato; un riassunto di quello che è stato scritto, e quello che è stato dimostrato.

Inoltre si vanno ad indicare possibili sviluppi e progetti futuri relativi al lavoro di questa tesi.

4.1 Conclusioni

In questa tesi è stato introdotto il problema della detenzione di immagini Out of Distribution all'interno di reti neurali addestrate. Ne è stato spiegato l'importanza e le motivazioni della risoluzione.

Sono stati presentati vari esperimenti in modo da cercare di descrivere più accuratamente il comportamento delle reti, in presenza di dati OD.

Come preannunciato nelle aspettative i risultati su MNIST sono migliori di quelli su CIFAR10.

Essendo un dataset molto semplice il classificatore riesce a rilevare con punteggi più alti le immagini ID rispetto a quelle OD. In realtà essendo

così semplice non dovrebbe esserci nessuna immagine mal classificata invece notiamo dal grafico dell'AP, sia con l'estrazione della probabilità massima (grafico 3.3) sia con l'entropia (grafico 3.4), che circa un 20% delle immagini ID ha un punteggio simile, se non minore di immagini OD.

La rete che processa CIFAR10 ha difficoltà a classificare le immagini, essendo più complesse. Dai grafici dell'AP si nota un miglioramento progressivo nella classificazione rispettivamente tramite: estrazione di probabilità massima (grafico 3.5), entropia minima (grafico 3.6), bayesian uncertanty (grafico 3.7).

Prendendo il caso migliore ovvero il test eseguito con bayesian uncertainty, neanche il 10% delle immagini ID ha un punteggio più alto delle immagini OD.

Come ci aspettavamo la rilevazione dei dati OD diventa più complessa aumentando la complessità delle immagini.

4.2 Lavori Futuri

Il lavoro riguardo all'argomento trattato può essere sviluppato maggiormente. I possibili sviluppi futuri sono molteplici, prima fra tutti sarà rendere la detenzione ancora più accurata. In questo modo sarà possibile rendere sensibile la rete di apprendimento nel caso di immagini OD. [11] Se così fosse la realizzazione e la messa in pratica delle auto self-driving non sarebbe poi così lontana.

Ringraziamenti

Ormai anche questo ciclo è finito. E‘ stato intenso e soddisfacente per me, ed è necessario ringraziare le persone che lo hanno reso tale.

Primo fra tutti il Professor Andrew Bagdanov che non ha solo confermato le mie aspettative ma le ha superate. Ci sono tanti tipi di professori all'università, ma sono pochi quelli che si dedicano con tutto il cuore a essere insegnanti. Lei è uno di questi, è stato un punto sicuro in mezzo a un mare di incertezze e per questo la ringrazio con tutta me stessa.

Adesso passiamo agli amici che mi hanno reso le infinite ore di studio molto più leggere e quasi piacevoli, senza di loro probabilmente non sarei qui.

Prima fra tutti la Michi, fedele compagna di studio (e di ansia) con ore e ore di chiamate su Skype a notte fonda. Grazie anche a Ema e Stiv che hanno sempre risposto ai miei mille dubbi e domande, sopportando anche quelle più sceme.

Grazie alla Chiara, a Cg, alla Giuli, alla Boch e alla Rache che nonostante sia sparita per 3 anni sono ancora al mio fianco a sostenermi.

Passiamo adesso alla mia famiglia che mi ha sempre sostenuto, anche quando ero insopportabile.

Grazie alla Nonna e alla Zia Candida che per ogni esame mi hanno fedelmente acceso una candelina per buon auspicio.

Grazie alla Mamma, al Babbo e a Elisa che mi hanno aiutato a tenere i piedi per terra e a dare una giusta importanza alle cose, soprattutto quando andavano male.

E infine un immenso grazie a te, Simo, che sei riuscito a farmi vedere la luce del sole (letteralmente) nei momenti più bui. A farmi respirare quando mi mancava l'aria. Ad asciugarmi le lacrime e a festeggiare con me quando ne avevo bisogno.

Grazie a tutti per aver reso possibile tutto ciò, vi voglio bene.

Alice Casali, 21 Febbraio 2019

Bibliografia

[1] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. 2018.

[2] Kyunghyun Cho. Understanding dropout: Training multi-layer perceptrons with auxiliary independent stochastic neurons. *CoRR*, abs/1306.2801, 2013.

[3] Gianluca Demartini. *Una metrica di valutazione per l'Information Retrivial: analisi critica e sperimentazioni.* Tesi di laurea, Università degli Studi di Udine, 2004/2005.

[4] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995.

[5] R. Baeza-Yates e R. Neto. *Modern Information Retrieval.* ACM Press, 1999.

[6] Alexandre Gramfort Vincent Michel Bertrand Thirion Olivier Grisel Mathieu Blondel Peter Prettenhofer Ron Weiss Vincent Dubourg Jake Vanderplas Alexandre Passos David Cournapeau Matthieu Brucher Matthieu Perrot Édouard Duchesnay Fabian Pedregosa, Gaël Varoquaux. Scikit-learn: Machine learning in python. 2011.

[7] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, abs/1610.02136, 2016.

[8] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7167–7177. Curran Associates, Inc., 2018.

[9] Shiyu Liang, Yixuan Li, and R. Srikant. Principled detection of out-of-distribution examples in neural networks. *CoRR*, abs/1706.02690, 2017.

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[11] Bjarke Kristian Maigaard Kjær Pedersen, Kamilla Egedal Andersen, Simon Köslich, Bente Charlotte Weigelin, and Kati Kuusinen. Simulations and self-driving cars: A study of trust and consequences. In *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, pages 205–206, New York, NY, USA, 2018. ACM.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia, 2019. [Online; accessed 30-January-2019].

[14] Zoubin Ghahramani Yarin Gal. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. 2015.

[15] Suman Jana Baishakhi Ray Yuchi Tian, Kexin Pei. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. 2018.