Università degli Studi di Udine
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Corso di Laurea in Informatica

Tesi di Laurea

Deep learning per il rilevamento dei loghi nelle potenziali pagine di phishing

Candidato:
Jakob Husu

Relatore:
Prof. Claudio Piciarelli

Anno Accademico 2017/18

Sommario

Nella seguente tesi sarà illustrato il fenomeno del brand abuse e i vari tipi di abuso nella rete, tralasciando invece quelli tradizionali. Ci sarà una descrizione di come il significato del brand si è evoluto con il tempo, dal semplice marchio a qualcosa di più astratto.

Sarà quindi trattato un modello per risolvere il seguente problema: ricevuto l'url della pagina web, verificare se questa è una pagina di phishing o no. La soluzione trattata per questo problema sarà basata sull'object detection controllando la presenza di loghi nelle pagine web.

Siccome sarà utilizzato l'object detection, ci sarà una breve storia di questo, trattando la sua evoluzione fino allo stato attuale. L'object detection utilizzato per il modello utilizzerà il deep learning basato sulle reti neurali convoluzionali, per questo saranno illustrati il deep learning e, più in dettaglio, le reti neurali convoluzionali.

Illustrate queste, sarà discussa la scelta del algoritmo di detection Yolo spiegando le performance che questo offre rispetto agli altri riguardo la precisione e la velocità di detection.

Inoltre, sarà descritto come il modello è stato applicato per risolvere il problema, ovvero: scovare se l'elenco di url delle presunte pagine di phishing contiene effettivamente pagine di phishing.

Infine, verranno spiegate varie problematiche, quali l'overfitting, per le quali sarà illustrata la soluzione presa. Oltre alle soluzioni, verrà presentata una breve delineazione del motivo per il quale è stato scelto Yolo illustrando i vari test eseguiti per controllare che l'algoritmo funzioni. Test i quali sono stati eseguiti durante la fase di training è durante la fase di validazione.

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti critiche e osservazioni. A loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni possibile errore contenuto in questa tesi.

Ringrazio anzitutto il mio relatore professor Claudio Piciarelli. Senza i suoi consigli e la sua guida nella stesura, questa tesi non esisterebbe.

Proseguo con le persone che ho incontrato durante il mio tirocinio presso Emaze sempre pronte ad aiutarmi. In particolare vorrei ringraziare Giacomo Alzetta senza il quale questo progetto non avrebbe funzionato, Marco D'Orlando per la sua disponibilità e l'opportunità datami con questo progetto e infine Davide Varesano per avermi accettato come tirocinante.

Un ringraziamento va ai miei amici e compagni di corso che mi hanno incoraggiato, in particolare Omar che ha speso parte del suo tempo per leggere e discutere con me le bozze del lavoro.

Vorrei infine ringraziare i miei genitori per avermi supportato durante l'intero percorso di studio.

Jakob Husu,
Trieste, 22 Febbraio 2019

Indice

Elenco delle figure

Elenco delle tabelle

1
Brand Abuse

In questo capitolo verrà innanzitutto descritto cos'è il **brand**, descrivendo come il significato della parola cambio con il tempo. Poi sarà descritto cos'è **brand abuse**, il quale tende in molti casi tende a rovinare la reputazione del *brand*. Infine, saranno descritti i vari tipi di abuso del brand.

1.1 Cos'è un brand?

Per capire cos'è un *brand* o "marca", dovremmo innanzitutto osservare cosa significhi *brand* e come il suo significato del termine è cambiato con il tempo.

"La marca è il nome, il simbolo, il disegno, o una combinazione di questi elementi, con cui si identificano prodotti o servizi di uno o più venditori al fine di differenziarli da altri venditori sul mercato"

Con una definizione del genere ci si focalizza soltanto sul marchio. Pensando alla storia del marchio, basterebbe pensare, come una volta i bovini venivano venduti. Essi venivano marchiati e così facendo, per le macellerie era molto più semplice conoscerne la provenienza. In questo caso il *brand* era utilizzato puramente per specificare l'appartenenza.

Nel XIX secolo, i produttori iniziarono a marchiare sempre più prodotti. Verso la fine del secolo viene fondata la Coca-Cola, uno dei marchi più conosciuti al mondo e ancora oggi attivo. Al momento della fondazione, vi erano già abbastanza produttori di soda sul mercato. Il problema in cui la ditta si ritrovò ad affrontare era: come distinguersi dagli altri produttori? I fondatori del marchio, ovviamente, volevano che il consumatore comprasse la loro soda, quindi dovevano far sì che il loro prodotto si distinguesse dalla concorrenza ed emergesse sul mercato. Già in quel periodo, il brand stava assumendo un nuovo significato, ovvero non soltanto quello di marchiare

il prodotto, ma anche quello di dare un motivo ai consumatori di sceglierlo. Verso la fine del XX secolo, gli economisti iniziarono a comprendere che c'era qualcosa di più nel brand: non pensavano più che la funzione del marchio fosse più di distinguersi. Dunque, non bastava avere soltanto un marchio diverso dalla concorrenza. In quel periodo infatti David Ogilvy descrisse il significato che lui aveva intravisto nel *brand*:

"Il brand è la somma intangibile delle caratteristiche di un prodotto"[36]

Ogilvy non era l'unico: anche gli altri economisti avevano intravisto qualcosa di più nel *brand* e capito che potevano creare una rappresentazione delle qualità e degli attributi in ogni prodotto/servizio non generico, cosa che poi chiamarono "the brand". Colin Bates definì così la sua visione del brand:

"Il brand è un insieme di percezioni nella mente dei consumatori"[6]

Ripensando a questa definizione diventa chiaro che essa esprime un concetto chiave, ovvero che il brand è intangibile. Esso è l'insieme delle percezioni legate a ciascuna interazione tra l'azienda e i suoi clienti. Questo significato vale ancora oggi, dove rispecchia ancora di più la realtà. Il brand non è più solamente il logo, la campagna pubblicitaria, il prodotto o la scenografia del punto vendita. Questi sono soltanto dei mezzi attraverso i quali l'azienda interagisce con il mondo esterno. Il *brand* oramai è la percezione mentale, l'insieme delle idee, delle opinioni e delle esperienze che i consumatori hanno avuto ogni volta che sono entrati in contatto con l'azienda. Tale percezione, se positiva, viene rafforzata ogni volta che i clienti utilizzano l'oggetto oppure usufruiscono del servizio con un'esperienza positiva. L'idea del *brand* è oramai così astratta che ci sono professionisti che creano i loghi per le aziende con vari obbiettivi. Un noto *brand* designer americano, Walter Landor definì così il *brand*:

"il brand è una promessa. attraverso l'identificazione e l'autenticazione di un prodotto/servizio, il brand si impegna solennemente a fornire determinati standard di soddisfazione e qualità."

Al giorno d'oggi, il progresso della tecnologia ha cambiato tanto la società e, di conseguenza, anche il significato di brand. Seth Godin, noto impresario di un'azienda dot-com, spiega che per un brand:

"Fare delle promesse e mantenerle è un'ottima strada da percorrere verso la costruzione di un brand."

Finora il *brand* è stato descritto sotto luce positiva, mentre non è sempre il caso nella realtà.

"Cosa succede se la tanto aspettata esperienza positiva per il cliente, lo scopo delle varie aziende, non c'è?" "Cosa succede se il cliente che utilizza un prodotto si aspetta qualcosa che il brand non gli ha dato?"

Se si osservano gli aspetti negativi invece, il brand è diventato così astratto come idea, che i consumatori ormai pensano che abbia promesso loro qualcosa[23]. Qualora questa promessa non abbia luogo, si perde la fiducia nel brand. Secondo un sondaggio[38], parlando di promesse si può affermare che:

- Il 70% dei consumatori reputa che il *brand* abbia fatto a loro una promessa;
- Il 40% dei consumatori reputa che questa promessa sia stata infranta;
- Il 90% dei consumatori afferma che considererebbe di cambiare il *brand* per via di una promessa non mantenuta;
- L'80% dei consumatori ridarebbe un'altra possibilità dopo una promessa non mantenuta;

Quindi all'interno di un brand, il marchio c'è, ma non è l'unico aspetto fondamentale. Aspetti quali la storia dell'azienda o del prodotto, la *customer experience* e la *brand awareness e identity*. C'è un mondo di promesse e valori e i consumatori, gli azionisti, i dipendenti e la concorrenza percepiscono che non possono essere quantificate.

1.1.1 Customer experience

Con il termine *customer experience*[33] si intende il modo in cui il cliente percepisce l'iterazione con un determinato *brand* a livello conscio e inconscio. È un indice che indica come il cliente "si sente" dopo aver interagito con il marchio, quindi la customer experience è legata al "come" non al "cosa". Ogni cliente valuta in modo cosciente e incosciente la sua esperienza. Di conseguenza, le aziende stanno diventando sempre più "cliente-centriche", mettendo alla base delle loro strategie la *customer experience*.

1.1.2 Brand awareness

Il *brand awareness*, anche chiamato *notorietà di marca*, è un parametro che indica quanto il marchio e i suoi prodotti o servizi sono riconosciuti nella mente del consumatore. L'apice della brand awareness è il punto di arrivo auspicato da ogni brand e significa che esso è il primo brand a cui i consumatori pensano durante il processo d'acquisto di un certo bene o servizio. Ovviamente, seppur raggiungere l'apice è un obiettivo ideale raggiunto da pochi, la notorietà di marca è in ogni caso un obiettivo notevole che le marche cercano di massimizzare.

La notorietà di marca, secondo il noto economista statunitense David Aaker, può essere definita con le seguenti parole:

> "Un set di attività (o passività) collegate a un segno distintivo (marchio, nome, logo) che si aggiungono (o sottraggono) al valore generato da un prodotto o servizio" [10]

Oltre alla definizione che Aaker diede della notorietà di marca, egli elaborò la *piramide della brand awareness*, che rispecchia la brand awareness e il punto in cui si trovano varie aziende. Una illustrazione della piramide di Aaker si può vedere nella figura 1.1.

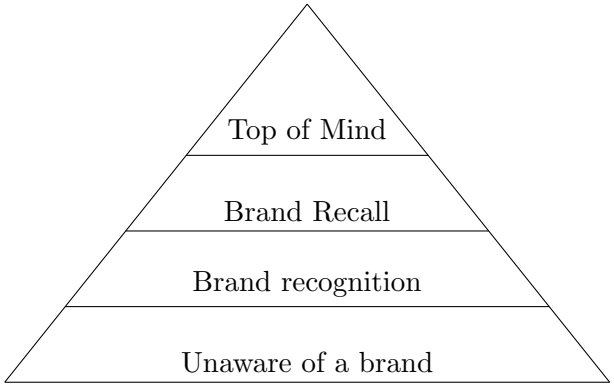

Figura 1.1: La piramide della *brand awareness*

Secondo il modello di Aaker, ogni marca parte dalla base della piramide. Partendo dal fondo si possono trovare i marchi che non sono ancora noti e che devono ancora emergere. L'obiettivo dei marchi è quello di vendere servizi/prodotti, ma finché non sono conosciuti non potranno trarne tanti profitti, avendo una clientela ristretta. Il marchio farà di tutto per farsi riconoscere e, come acquisisce notorietà, inizierà a scalare la piramide. Più il marchio si avvicina all'apice, più ne diventa spontaneo il

riconoscimento da parte del pubblico. Infine, quando il brand raggiungerà la cima, una meta tanto auspicata da tutti i marchi, il riconoscimento del marchio sarà così spontaneo, che il consumatore collegherà il bene/servizio al marchio. Scalando la piramide, il marchio, oltre a diventare più conosciuto, si creerà un'identità. Lo stesso Aaker definì l'identità, anche detta *brand identity*, come:

"una combinazione unica di associazioni che l'azienda ambisce a costruire e a mantenere nel tempo. Queste associazioni supportano la marca e rappresentano la promessa che l'azienda si impegna a mantenere nei confronti dei consumatori."[10]

1.2 Cos'è il brand abuse

Il **brand abuse** è un termine inglese utilizzato per riferirsi all'abuso di marca e per raggruppare i diversi tipi di attività dannose nei confronti della stessa. Il termine include diversi tipi di attività dannose per la marca, quali quelle "dolose", che sfruttano la popolarità di un brand per il loro beneficio e le attività che mirano a danneggiare direttamente la reputazione della marca.

1.3 Tipi di abuso frequenti

Dopo aver spiegato cos'è il brand, verranno spiegati i vari tipi di abuso del marchio. Tali tipi di abuso danneggiano l'esperienza del cliente e, siccome il brand è ormai molto astratta come idea, possono creare grossi danni all'azienda colpita. Per abusi, verranno intesi soltanto gli abusi perpetrati online, poiché questi verranno trattati nella tesi[35].

Un'attività illegale abbastanza diffusa, anche se grazie alle leggi[8] sta perdendo piede nei paesi sviluppati, è il **cybersquatting**. Questa attività illegale è intesa come l'atto di appropriarsi di domini corrispondenti a marchi commerciali o a nomi di personaggi famosi, al fine di realizzarne un ricavo rivendendo il dominio all'interessato, oppure di creare danno impedendone il suo utilizzo. Per esempio, un atto di cybersquatting potrebbe essere il seguente: un malintenzionato viene a conoscenza dell'esistenza di un'azienda emergente, oppure che non ha ancora un dominio registrato, quindi lo compra e ne registra il nome. Siccome un singolo nome non può essere registrato più volte, l'azienda in questione, se interessata ad esso, deve contattare il proprietario per farselo cedere. Il malintenzionato, ovviamente, punta a sfruttare questa situazione e farsi pagare per la cessione del dominio. Quello che

capita in questo caso è che il malintenzionato compra nomi di domini con all'interno nomi di aziende, oppure di personaggi famosi. Per questo esistono leggi[32] che fanno sì che oramai non sia più così facile per i malintenzionati lucrare con il cybersquatting.

Un'attività simile allo cybersquatting è il **domain grabbing**. Quest'ultimo consiste nel comprare il dominio di nome simile ad un'azienda rivale per trarne profitto. In tal caso, l'attività viene eseguita da un'azienda, la quale regista un dominio con un nome simile a quello di un'altra azienda conosciuta, per trarne profitto. Un esempio di questo potrebbe essere un azienda che ha un dominio "buymyshoes.com" e un altra che vuole trarne vantaggio con il dominio "buymyshirts.com". Lo scopo dell'azione in questo caso non è più vendere il dominio, ma trarne profitto, così che i clienti vadano sul sito dell'azienda malintenzionata per sbaglio o per via della similitudine nell'URL. Di conseguenza, la presente attività potrebbe anche essere definita come concorrenza sleale[53].

Un altro tipo di abuso è il **typosquatting**[3]. Per il typosquatting si intende un'attività in cui il malintenzionato registra un nome di dominio molto simile, a prima vista uguale a quello che vuole trarne vantaggio. Per esempio, per il dominio "google.com" il malintenzionato crea uno simile "gooogle.com". L'obiettivo della registrazione di un nome simile è quello di trarre in inganno l'utente, che non si accorge che sta visitando un altro sito. Di solito, il typosquatting viene effettuato per far sì che l'utente rimanga sul sito e il malintenzionato così guadagni con la pubblicità. I gestori di tali siti in genere imitano persino la grafica e i contenuti del sito originale, in modo che gli utenti vi rimangano più tempo possibile prima di accorgersene. In questo caso, il typosquatting viene chiamato **PPC Abuse (Pay-Per-Click)**. Oltre a guadagnare con la pubblicità sul sito, il malintenzionato potrebbe sfruttare la situazione per rubare dati sensibili, quali nomi utente e password, oppure dati di pagamento. In tal caso, il typosquatting si trasforma in un vero e proprio atto di phishing[14] il quale sarà trattato con il modello.

Un'altra attività che capita sul web è quella delle **associazioni fasulle**. Lo scopo di tale pratica è di far credere ai visitatori che si è associati con un brand, mentre in realtà non è così. Un esempio di associazione fasulla potrebbe essere un operatore di un sito web che utilizza varie icone o loghi di altri brand famosi, cosicché il cliente pensi che sia associato ad esso. Lo scopo di questo atto è far pensare che sia un sito web di qualità, perché se il cliente vede associazioni con marchi famosi la prima cosa che pensa è:

"Se il brand famoso è associato con il brand in questione allora sarà sicuramente un marchio meritevole, dopotutto un brand importante non fa associazioni con i brand di scarsa qualità."

Un altro modo in cui un utente malintenzionato potrebbe abusare di un brand, è con **l'intercettazione del traffico** per re-direzionarlo verso un altro sito web concorrente, oppure a un sito web illecito, per guadagno a danno del legittimo proprietario. Essendo questi veri e propri attacchi informatici, i quali possono essere risolti in parte utilizzando https invece di http. Questo tipo di abuso non sarà trattato nella presente tesi.

2

Lo stato dell'arte dell'object detection

Per l'object detection, esistono diversi approcci al problema. Generalmente, gli approcci sfruttano il machine learning tradizionale oppure il deep learning. Per quelli gli approcci al machine learning, invece, bisogna innanzitutto definirne le caratteristiche usando per esempio tecniche come la SVM[1] per la classificazione. Per quelli di deep learning, invece, si può fare l'"object detection" senza definirle, poiché vengono utilizzate le reti neurali convoluzionali, le quali saranno spiegate nella sezione 4.1.

I vari tipi di approcci:

- Machine Learning:
  - "Viola–Jones object detection framework based on Haar features"
  - "Scale-invariant feature transform (SIFT)"[30]
  - "Histogram of oriented gradients (HOG) features" [9]
- Deep Learning:
  - "Region Proposals" (R-CNN[18], "Fast R-CNN"[17], "Faster R-CNN" [45])
  - "Single Shot MultiBox Detector" (SSD)[31]
  - "You Only Look Once" (YOLO)[42, 43, 44]

Generalmente le soluzioni che utilizzano il deep learning sono più performanti di quelle del *machine learning*, come si può evincere dalla figura 2.1.

[1] support vector machine [37]

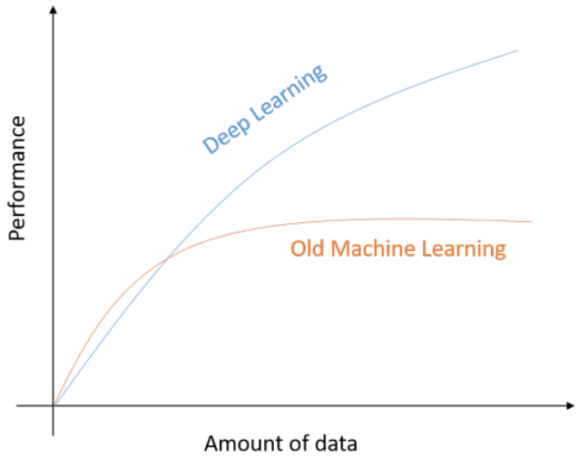

Figura 2.1: La *performance* del deep learning rispetto al machine learning

L'*object detection*, come viene pensato oggi con l'utilizzo delle reti neurali convoluzionali [52], nasce nel 2012 con AlexNet.

AlexNet[26] vinse ILSVRC[12] con un errore nella classifica "top-5" del 15.3%. Questo errore era di gran lunga più piccolo di quelli degli altri concorrenti, infatti AlexNet diede un distacco dal secondo concorrente di ben 10.8%, il che creò una grande notizia. La vittoria fu così strabiliante che anche le comunità al di fuori di quelli del AI iniziarono a interessarsi al deep learning. Parecchi articoli furono scritti, al di fuori delle comunità di AI[1, 2, 16] e di conseguenza il *deep learning* divenne un argomento molto discusso e i programmatori iniziarono ad utilizzarlo.

Dopo il successo di AlexNet, vennero create molte nuove reti, alcune di queste visibili nella figura 2.2.

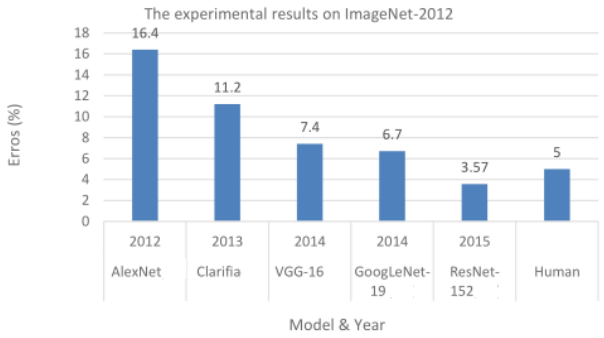

Figura 2.2: Precisione dei modelli su ImageNet: AlexNet[26], Clafira[58], VGG-16[49], GoogleNet-19[51], ResNet-152[21]

Attualmente, possiamo trovare diversi tipi di reti, dalle reti molto precise come la ResNet a quelle che puntano alla velocità. Un esempio di quest'ultime è la rete

neurale Darknet[41], la quale ha più versioni Darknet-19 e Darknet-53, composte da rispettivamente 19 e 53 strati.

Questa rete convoluzionale è stata creata per essere utilizzata da Yolo[2] [42, 43, 44]. Yolo è un algoritmo di detection che punta alla velocità e può essere utilizzato su un portatile di potenza media oppure addirittura su uno smartphone. Oltre a ciò, Yolo è così veloce da poter essere applicato su uno stream video in real time.

[2]You Only Look Once

3

Machine Learning e il Deep Learning

Essendo il deep learning un caso specifico del machine learning, quest'ultimo verrà descritto per primo. In seguito, ne saranno elencati i vari tipi e si proporrà la definizione dell'algoritmo, la quale sarà successivamente analizzata. Infine, verrà presentata la spiegazione del deep learning.

3.1 Il Machine Learning

Il "machine learning"[5], anche noto in italiano come apprendimento automatico (termine quasi mai usato), può essere definito come l'insieme dei metodi utilizzati per trovare automaticamente relazioni tra i dati, nonché usufruire delle informazioni scoperte per predire gli esiti futuri dei dati[46].

Il termine "machine learning" venne coniato da Arthur L. Samuel nel 1959[47]. Sempre nello stesso anno, venne pubblicato dall'"IBM JOURNAL OF RESEARCH AND DEVELOPMENT" il primo articolo nel quale veniva trattato il "machine learning". Per la prima volta fu pubblicata una soluzione al problema, che, nel caso di Arthur L. Samuel era il gioco della "dama", utilizzando un approccio differente rispetto al passato. Arthur scrisse un programma, il quale **imparava** giocando e migliorava dopo ogni partita, diventando così più forte. Questo è l'approccio del machine learning: cercare di insegnare ad un programma cosa deve fare e "dirgli" di imparare a risolverlo. A. L. Samuel, oltre alla descrizione del programma, descrisse anche due possibili approcci al "machine learning":

1. Il primo approccio consiste in un modello di machine learning per l'impiego generale, il quale apprende il comportamento con una rete di commutazione connessa casualmente. L'apprendimento viene basato sulla base della ricompensa e della punizione (apprendimento per rinforzo).

2. Il secondo approccio consiste invece nel progettare una rete organizzata, la quale è pensata per imparare soltanto alcune specifiche.

La differenza tra i due approcci è, che il secondo tipo necessita di supervisione, quindi la rete deve essere riprogrammata ad ogni applicazione. Anche se questo tipo di approccio richiede la riprogrammazione per ogni problema, dal punto di vista computazionale è più efficiente. Più tardi, appena nel 1997, il "machine learning" viene formalmente definito da Tom M. Mitchel nel suo libro.

3.1.1 Algoritmi di apprendimento

Un algoritmo di apprendimento è un algoritmo capace di imparare dai dati, come viene definito da Tom M. Mitchell:

> "Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, misurate da P migliorano con l'esperienza E." [34]

Il Compito: T

Un compito (in inglese "task") è descritto in termini di come un sistema di machine learning dovrebbe processare un *esempio*. Quest'ultimo potrebbe, per esempio, essere una collezione di *funzionalità* che sono state misurate quantitativamente, da un oggetto o evento, le quali si vuole che il sistema di machine learning processi.

Per la definizione di "compito", invece, è necessario innanzitutto specificare che il processo di apprendimento stesso non è il compito: l'apprendimento è il mezzo per raggiungere la capacità necessaria per svolgere il compito. Dunque, se si volesse costruire una macchina con la guida automatizzata, "il programma in grado di guidare una macchina" sarebbe il "compito". Al fine di risolvere questo compito, è possibile utilizzare un approccio classico e creare un programma che sappia come guidare direttamente. Tuttavia, se si desiderasse utilizzare l'approccio del machine learning, allora sarebbe più opportuno creare un programma che impari a guidare.

Il machine learning è adatto a diversi tipi di *compiti*, ma le sue attività comuni includono le seguenti:

- **Classificazione**: Per la classificazione, il calcolatore conosce un insieme di categorie, dove il suo obiettivo è quello di scoprire e riportare a che categoria

appartengono, quando gli vengono passati dei dati in input. Si prenda la funzione dell'equazione 3.1, dove x sono i dati in input che la funzione riceve e dove z è il risultato, cioè la categoria che la funzione ci ha restituito.

$$z = f(x) \tag{3.1}$$

Inoltre, analizzando la funzione dell'equazione 3.2 come da esempio, si può notare che la funzione calcola la categoria corrispondente, dall'insieme di categorie che la funzione conosce.

$$f : \mathbb{R}^n \to \{1, .., k\} \tag{3.2}$$

- **Regressione**: Per questo compito al programma viene dato un input e chiesto di prevedere un valore numerico. Tale compito assomiglia a quello di classificazione, ma il formato dell'output è diverso. La seguente equazione 3.3 esprime la funzione di regressione:

  $$f : \mathbb{R}^n \to \mathbb{R} \tag{3.3}$$

  Di solito la regressione viene utilizzata dagli algoritmi per prevedere movimenti in borsa.

- **Trascrizione**: Per questo compito al programma viene chiesto di osservare una rappresentazione di dati e trascriverli in forma testuale. Per esempio:

  "al programma viene data una foto di un testo, il programma restituisce un file di testo (caratteri ASCII, UTF-8 o simili)."

  Ciò aiuta l'utente, dandogli la possibilità di automatizzare la trascrizione del testo da un'immagine.

- **Traduzione di macchina**: per questo compito, il programma prende una sequenza di lettere/simboli di un linguaggio e li traduce. Ciò che il programma ritorna è il testo tradotto in un altro linguaggio.

- **Scoperta delle anomalie**: il programma per scoprire le anomalie controlla una serie di eventi od oggetti e se trova qualcosa di strano in essi lo segnala. Un esempio di questo potrebbe essere i casi in cui vengono rubati i dati di accesso di un utente: un sito potrebbe infatti monitorare da che IP si accede e, qualora notasse che si fatto accede da un IP improbabile, potrebbe chiedere informazioni aggiuntive per verificare che sia l'utente legittimo ad accedere.

- **Sintesi e campionamento**: L'algoritmo in questo caso genera nuovi esempi simili a quelli dei dati di training. Il compito viene utilizzato nei task dove la generazione manuale è costosa oppure troppo lunga.
- **Togliere il rumore**: il seguente compito, chiamato in inglese *denoising*, consiste nel correggere i dati corrotti e prevederne quelli che dovrebbero essere ricevuti. Al programma viene dato in input un dato corrotto $\tilde{x} \in \mathbb{R}^n$, il suo scopo è di prevedere quello che doveva essere x, dove $x \in \mathbb{R}^n$ è appartenente all'insieme dei dati corretti.

Le prestazioni: P

Per stimare le abilità dell'algoritmo di machine learning, si deve dare una misura quantitativa delle sue prestazioni, le quali vengono misurate rispetto a uno specifico task che il sistema deve svolgere. Per i task come la classificazione, basta misurare la precisione del modello che si basa sulla proporzione di quante volte il modello produce un output corretto rispetto alle volte che il modello sbaglia. Per gli altri task, i quali non vengono trattati in questa tesi, invece non si può semplicemente stimare quante volte l'algoritmo riesce a trovare un output giusto ma bisogna trovare altri modi.

L'esperienza: E

Gli algoritmi di machine learning possono essere categorizzati dal tipo di apprendimento a cui vengono sottoposti. Le due più grandi categorie sono rispettivamente gli algoritmi di machine learning **non supervisionati** e quelli **supervisionati**. Esiste anche una terza categoria, quella dell'**apprendimento per rinforzo**.

- L'**apprendimento supervisionato**[25]:

  è una tecnica che mira ad istruire un sistema informatico, cosicché esso possa elaborare automaticamente previsioni sui valori d'uscita di un sistema datogli un input sulla base di esempi forniti inizialmente. Al sistema vengono date inizialmente varie coppie di input e output e da queste coppie inizierà a cercare le varie caratteristiche le quali gli permetteranno di trovare da solo gli output giusti in futuro, quando al sistema saranno forniti vari dati in ingresso e ci si aspetterà un risultato da lui calcolato. Esempio di un insieme di coppie fornite al sistema durante la fase iniziale.

$$D = \{(x_i, y_i)_{i=1}^N\} \tag{3.4}$$

Dove **N** è il numero di coppie, **x** l'input e **y** l'output, **D** invece è il *training set*.

Per esempio, si prenda un modello di dataset nel quale per ogni esempio ci sia un label associato. Datogli un dataset di foto di animali, questi saranno denotati nelle immagini. Facendo questo il modello saprà come classificare gli animali che ha imparato durante l'apprendimento, così quando gli saranno sottoposte immagini saprà trovare quelli che già conosce.

- L'**apprendimento non supervisionato**[20] è una tecnica che consiste nel fornire al sistema una serie di input, dai quali il sistema organizzerà e classificherà nelle varie classi, sulla base delle relazioni che ha trovato tra gli esempi. L'obiettivo è scoprire se nei dati ci sono "pattern interessanti". Siccome questo algoritmo non riceverà suggerimenti, l'insieme dei dati di ingresso, saranno solo i vari input forniti.

$$D = \{(x_i)_{i=1}^{N}\} \tag{3.5}$$

  Qui gli input non sono etichettati in nessun modo e lo scopo del modello è capire cosa cercare. Questi algoritmi sono più utilizzati per il deep learning che per il machine learning, poiché possono imparare intere distribuzioni di probabilità che ha generato il dataset.

- Infine, c'è l'approccio dell'**apprendimento per rinforzo**, dove il modello interagisce con un ambiente dinamico e, avendo un insegnante che lo premia e punisce, viene indirizzato verso l'obiettivo voluto. L'idea è di far capire al sistema, con premi e punizioni, se sta imparando bene o no e di fargli scartare così le cose sbagliate che ha imparato.

3.2 Il Deep Learning

Il *deep learning*[27] è quel campo di ricerca del "Machine Learning" e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso livello.

L'apprendimento profondo è definito come una classe di algoritmi di machine learning basati sull'apprendimento[13], il quale:

- "Usa vari livelli a cascata per svolgere compiti di estrazione di caratteristiche e di trasformazione."

- "Ciascun livello successivo riceve in input i dati di output del livello precedente."
- "Gli algoritmi in caso di apprendimento non supervisionato includono l'analisi di pattern, nel caso di apprendimento supervisionato invece di classificazione."

Il *Deep learning* si suddivide in tre grandi categorie[19]:

- **Reti profonde per l'apprendimento non supervisionato o reti generative**[7], che hanno lo scopo di trovare correlazioni di ordine elevato nei dati osservati per l'analisi del modello, quando non sono disponibili informazioni sulle etichette delle classi di destinazione. Quando si parla di reti non supervisionate, normalmente ci si riferisce a questa categoria di reti profonde.
- **Reti profonde per l'apprendimento supervisionato o reti discriminative**, queste reti sono programmate per riconoscere i vari pattern e così classificarli. L'etichettizzazione finale voluta è sempre disponibile in forma diretta o indiretta alla rete.
- **Reti profonde Ibride**[24], queste reti hanno lo scopo di discriminare, in modo assistito, ma con i risultati delle reti profonde non supervisionate o generative.

Nei seguenti capitoli le reti utilizzate e spiegate saranno quelle di **apprendimento supervisionato**. Gli altri tipi di reti non saranno trattate in questa tesi.

Il deep learning viene utilizzato principalmente per i seguenti[4]:

1. **approccio di apprendimento universale**: L'approccio, anche chiamato apprendimento universale, è chiamato così perché può essere applicato a quasi ogni ambito.
2. **robustezza**: Gli approcci di *deep learning* non necessitano design per le caratteristiche prima dell'inizio, siccome queste vengono automaticamente dal modello. Le caratteristiche imparate sono quelle ottimali per il tipo di problema senza studi, per questo la robustezza alle variazioni nei dati è imparata completamente dal modello.
3. **generalizzazione**: Lo stesso approccio può essere usato per diverse applicazioni, oppure con dati diversi. Questa proprietà è anche chiamata *transfer learning* [39] ed è utile quando non si ha abbastanza dati.

4. **scalabilità**: L'approccio del *deep learning* è fortemente scalabile. Nel 2015 venne rilasciato ResNet[21], il quale contiene 1202 layer e quindi è utilizzato soltanto da supercomputer. Oltre a questa rete, vi è una grande iniziativa per sviluppare anche questo tipo di reti a LLNL[1] Quindi, il *deep learning* può essere implementato sia per calcolatori "normali" che per supercomputer.

[1]Lavrence Livermore National Laboratory [55]

4

Deep learning per l'object detection

In questo capitolo saranno spiegati i vari dettagli del deep learning. Verranno analizzate le reti neurali convoluzionali, il motivo del loro nome, e le varie operazioni che vengono fatte durante il loro uso. Infine, si descriverà Yolo[42].

4.1 Rete neurale convoluzionale

Le reti neurali convoluzionali[52], chiamate *Convolutional Network* oppure *CNN*, sono reti neurali specializzate per processare le matrici. Un esempio di questo tipo di dato sono le immagini, le quali sono matrici bi-dimensionali di pixel[26]. Le reti vengono chiamate convoluzionali per convenzione, anche se in realtà utilizzano l'operazione di **cross-correlation**, la quale ha una forte somiglianza a quella di **convoluzione**[56].

Una rete neurale consiste di un layer di *input* e uno di *output* e vari stati intermedi. Gli stati intermedi della rete, che verranno successivamente analizzati, sono:

- layer convoluzionale
- *RELU layer*
- layer di "pooling"
- layer completamente connesso
- layer di perdita

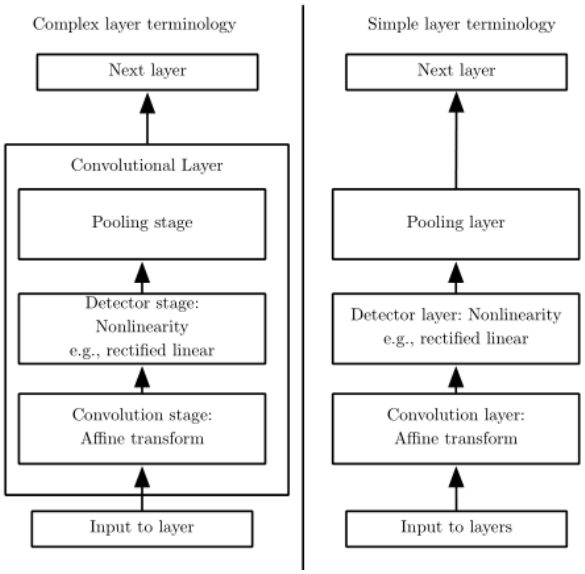

Figura 4.1: Vari step presente nel layer convoluzionale. Immagine presa dal libro Deep Learning[19]

4.1.1 Convolutional layer

Il convolutional layer applica la funzione di convoluzione al dato ricevuto in input e restituisce il risultato. La convoluzione simula la risposta di un neurone alla stimolazione visiva.[28]

Un layer convoluzionale, esempio in figura 4.8, è composto da tre processi, di seguito elencati:

- Il primo processo effettua diverse convoluzioni in parallelo per produrre un insieme lineare di attivazioni.
- Nel secondo processo, invece, ogni attivazione lineare viene eseguita da una funzione non lineare, per esempio la funzione lineare del rettificatore. Il secondo processo viene talvolta chiamato **detector stage**.
- Diversamente, nel terzo processo viene utilizzata una **funzione di pooling** per modificare l'output del layer.

L'operazione di convoluzione

Una convoluzione è un'operazione su due funzioni di un argomento con valori reali. Si prenda il seguente esempio per descrivere l'operazione. Presa una persona che

stia camminando, si cerchi di tenerne la traccia del suo spostamento con un sensore. Il sensore fornisce un singolo output $x(t)$. Considerando t una variabile che indica il tempo, la funzione restituisce la posizione della persona. Siccome x e t sono valori reali, il risultato sarà diverso ogni volta.

"Tutto questo potrebbe bastare, ma cosa si può fare quando il nostro sensore non riesce a calcolare sempre il risultato giusto per colpa del rumore?"

Per filtrare il rumore si può calcolare più volte la sua posizione ed effettuare la media con la funzione $w(x)$ delle varie posizioni. Ma se questo viene applicato a ogni istante, ci si ritrova con la seguente funzione di *convoluzione*, la quale dà una stima lisciata della posizione della persona:

$$s(t) = \int x(a)w(t-a)\ da \tag{4.1}$$

Questa viene di solito indicata con un asterisco:

$$s(t) = (x * w)(t) \tag{4.2}$$

Presa la funzione w come una valida funzione di densità di probabilità, per l'esempio dato, questa ritornerà 0 per tutti i valori negativi.

Nella terminologia delle reti convoluzionali, il primo argomento viene chiamato **input**, mentre il secondo argomento viene chiamato **kernel** oppure **feature detector**. Infine, l'output viene chiamato **feature map**. Nell'esempio precedente x è considerato l'input e w è il kernel.

L'operazione di convoluzione calcolata dal calcolatore

Si prenda l'equazione 4.1 che indica la funzione precedentemente utilizzata come esempio. Il problema della seguente funzione è che purtroppo il calcolatore non può calcolare la posizione in ogni istante. Per questo motivo, il variare del tempo sul calcolatore viene discretizzato. Difatti, è abbastanza ragionevole aspettarsi da un sensore che fornisca la sua posizione per esempio ogni secondo invece che fornirlo costantemente. Ciò sarebbe appunto impossibile per un calcolatore, quindi quest'ultimo utilizza la convoluzione discreta.

Se si ritorna all'esempio di prima, della persona, utilizzando un sensore che calcola la sua posizione ad intervalli di un secondo, si può fare in modo che l'indice t

assuma solo valori interi. Presupponendo che x e w siano definiti soltanto sull'indice t, si può formulare la seguente convoluzione discreta:

$$s(t) = (x * w)(t) = \sum_{y=-\infty}^{\infty} x(y)w(t-y) \tag{4.3}$$

Prese le applicazioni di machine learning reali, l'*input* è di solito una matrice di dati multidimensionale, mentre il *kernel* è una matrice di parametri che sono adattati dall'algoritmo di machine learning. Queste matrici multidimensionali verranno chiamate *tensor*. Siccome ogni elemento dell'*input* e del *kernel* viene salvato separatamente, si assume che queste funzioni siano nulle in tutti i punti tranne che nel set finito di punti in cui vengono salvati i valori. Assumendo questo, si può implementare la sommatoria infinita come una sommatoria su un insieme finito di elementi della matrice.

Di seguito, viene mostrato un esempio di una funzione di convoluzione che utilizza più di un'asse. Per essere più specifici, si prenda l'equazione 4.4 che rappresenta la funzione a due assi, dove I è un'immagine e K è il kernel. Entrambi i valori sono matrici bi-dimensionali.

$$s(x,y) = (I * K)(x,y) = \sum_m \sum_n I(m,n)K(x-m, y-n) \tag{4.4}$$

Sfruttando la commutatività si può riscrivere l'equazione come:

$$s(x,y) = (I * K)(x,y) = \sum_m \sum_n I(x-m, y-n)K(m,n) \tag{4.5}$$

L'equazione 4.4 rappresenta la stessa formula, ma formulata in maniera più semplice da implementare in una libreria di machine learning, poiché c'è una minore variazione dei valori che m e n possono assumere. Il motivo di questa formula è di sfruttare la commutatività **invertendo** il kernel relativo all'input. Facendo ciò, con l'incrementare di m, l'input dell'indice incrementa mentre quello del kernel decrementa.

Anche se questa formula è più efficiente nelle varie librerie delle reti neurali, viene utilizzata la funzione di **cross-correlation**, dove viene fatta la stessa operazione, ma senza invertire il kernel. L'operazione di cross-correlation è visibile nell'equazione 4.6:

$$s(x,y) = (I * K)(x,y) = \sum_m \sum_n I(x+m, y+n)K(m,n) \tag{4.6}$$

In questa figura 4.3 si può notare che il kernel rimane nel range dell'immagine.

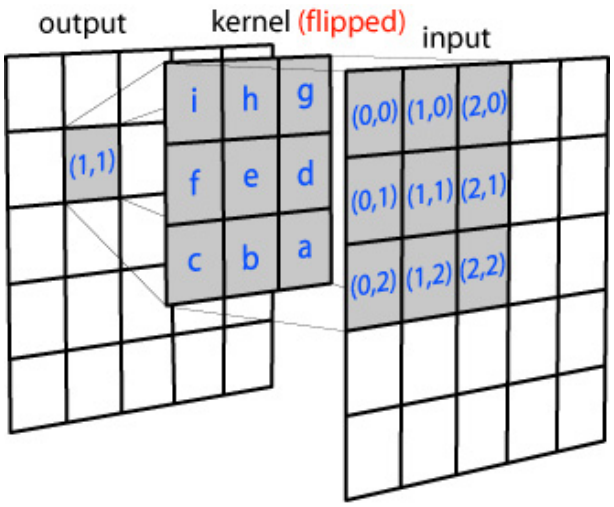

Figura 4.2: Esempio di convoluzione 2-D con l'inversione del kernel

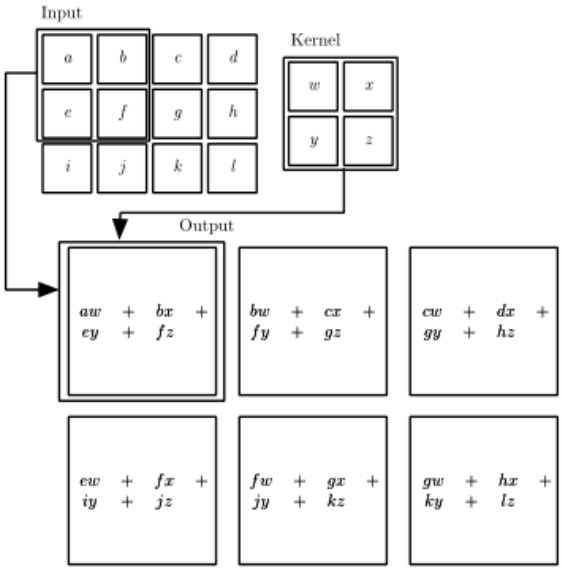

Figura 4.3: Esempio di convoluzione 2-D senza l'invertire il kernel. Immagine presa dal libro Deep Learning[19]

Motivo dell'utilizzo delle convoluzioni

Nel machine learning, le convoluzioni vengono utilizzate per migliorare le operazioni di machine learning. Infatti, vengono migliorate le **connesioni sparse**, **la condivisione dei parametri** e le **rappresentazioni di equivarianti**.

Gli strati delle reti neurali tradizionali usano la moltiplicazione delle matrici da una matrice di parametri con parametri separati, descrivendo l'iterazione di ogni input con ogni output. Questo non accade invece nelle reti convoluzionali, le quali

di solito usano **connessioni sparse**.

Le connessioni sparse permettono di avere il kernel più piccolo dell'input e ciò permette a sua volta al modello di salvare meno parametri: migliorando l'utilizzo della memoria e la sua efficienza. In tal modo il modello sarà più performante per il calcolo.

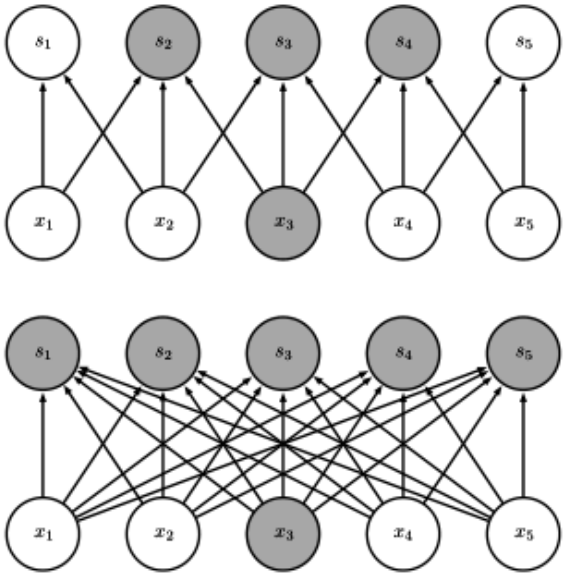

Figura 4.4: Connettività sparsa vista dall'alto. Immagine presa dal libro Deep Learning [19]

Si prenda la figura 4.4 come esempio, dove le varie variabili x_i sono valori in input e i vari s_i sono gli output. Si prenda, inoltre, un kernel di grandezza 3, x_3 e i vari s_i evidenziati. Si può vedere, nell'immagine superiore, che un kernel di grandezza 3 darà tre valori in output. Nell'immagine di sotto invece gli s vengono formati da una moltiplicazione di matrici e quindi le connessioni non sono più sparse, per questo gli output non sono più dati soltanto da x_3.

Nella figura 4.4, si può vedere lo stesso grafo dal basso. Qui il problema è inverso: se si prende il nodo s_3 evidenziato nel grafo superiore, si può vedere che questo dipende dai nodi di input evidenziati con le varie x_i. Anche per questa figura è stato preso un kernel di grandezza 3. Nel grafo inferiore invece, la connessione tra i nodi non è più sparsa è si può vedere che tutti gli x_i influenzano s_3. Le unità di x evidenziate vengono chiamate **campi ricettivi**.

Nella figura 4.6, vengono mostrati i collegamenti di più strati della rete convoluzionale. Più si va in profondità, più grandi sono i campi ricettivi. Ciò significa che i nodi più profondi dipendono da più nodi di input.

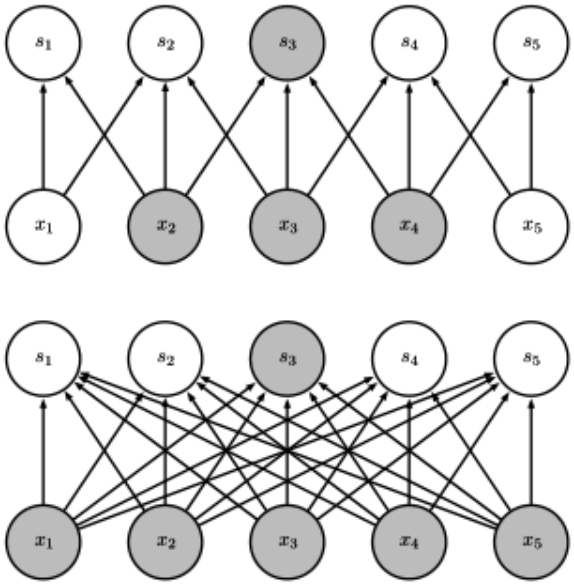

Figura 4.5: Connettività sparsa vista dal basso. Immagine presa dal libro Deep Learning[19]

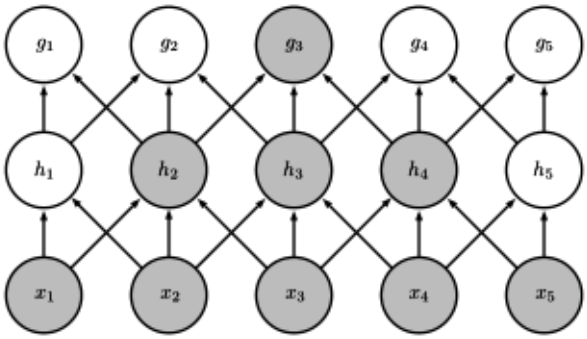

Figura 4.6: Differenza tra i vari livelli di strati convoluzionali. Immagine presa dal libro Deep Learning[19]

Lo scopo della convoluzione con i campi condivisi è l'efficienza rispetto alla moltiplicazione di matrici dense.

Nella figura 4.7, l'immagine alla destra viene formata cosicché a ogni pixel venga sottratto il valore del pixel alla sua sinistra. Questo fa si che si "annullino" i pixel vicini in cui non c'è un bordo essendo questi di un colore simile, mentre rimangono i valori nei pixel in cui ci sono i bordi di un oggetto.

Figura 4.7: Efficienza nello scovare i bordi di un oggetto. Immagine presa dal libro Deep Learning[19]

Equi-varianza

La condivisione dei parametri nella convoluzione fa sì che il layer assuma la proprietà di **equi-varianza** rispetto alla traslazione. Una funzione equi-variante è caratterizzata dal fatto che se le vengono dati input diversi, gli output che restituirà cambieranno nello stesso modo. Questa proprietà può essere espressa dall'equazione 4.7:

$$f \text{ è equivalente a } f(g(x)) = g(f(x)) \tag{4.7}$$

Presa una funzione g, come una funzione che trasla l'input, si può vedere che questa operazione permette di shiftare i pixel. Presa un'immagine Img e g una funzione equi-variante si può creare la seguente equazione:

$$Img' = g(Img) \Rightarrow Img'(x, y) = Img(x - 1, y) \tag{4.8}$$

Considerata l'equazione 4.8, si può notare che Img' è praticamente uguale a Img, ma con ogni pixel shiftato di un'unità a destra. Se viene applicata questa trasformazione ad Img e poi viene applicata la convoluzione, il risultato è uguale all'applicazione inversa. Ciò accade perché, come è stato detto, la funzione di convoluzione è una funzione equi-variante.

Ciò ci viene utile nel caso conosciamo già una funzione con un numero piccolo di pixel vicini, la quale se applicata più volte in diverse posizioni anche lì fa quello che ci serve.

Di norma, quando le reti convoluzionali processano immagini, scoprono i bordi nel primo layer di una rete convoluzionale e si pensa gli stessi bordi, infatti possiamo trovarli in più parti dell'immagine.

Certi dati, infatti, non possono essere processati dalle reti neurali normali, le quali fanno le moltiplicazioni di matrici normali. In questi casi, si può usare le reti convoluzionali per processare questi dati.

Esempio di convoluzione applicata alle immagini

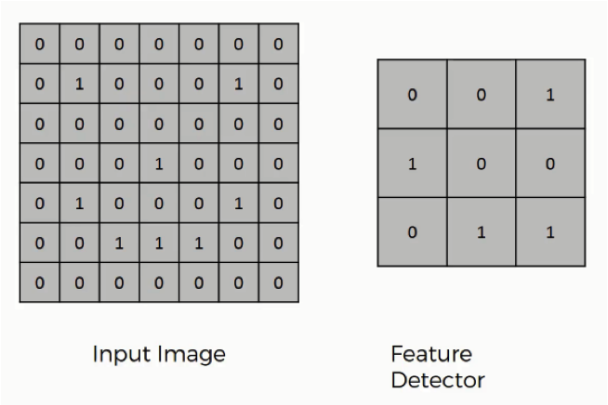

Figura 4.8: L'immagine di input e il feature detector scelto

Si prenda come esempio un'immagine di grandezza 9×9 e un feature detector di grandezza 3×3 per mostrare come funziona.

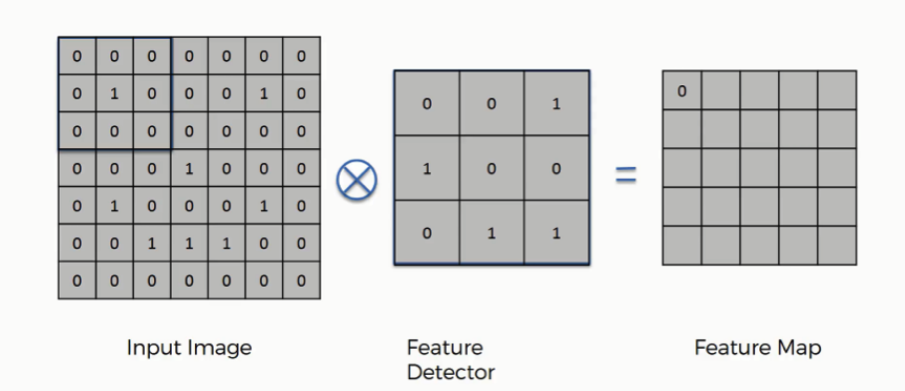

Figura 4.9: Inizio del calcolo della feature map

Come si sa, grazie alle convoluzioni da una matrice di $N \times M$ si può ricavare un'immagine più piccola, senza perdita di troppe informazioni. Si prenda come esempio l'immagine 4.9. Per calcolare la feature map si svolgono le seguenti operazioni:

1. Si prenda il kernel e lo si applichi nella posizione in alto a sinistra come mostrato in figura 4.9

2. Una volta scelta la posizione, si utilizzi la nuova matrice scelta, contando il numero di posizioni in cui entrambe le celle contengono il numero 1.

3. Il numero calcolato viene inserito nella cella della feature map.

4. Per calcolare le restanti celle, la matrice 3×3 viene spostata sulla matrice di input. Per ottenere il valore della cella a destra, di quella calcolata nella feature map, viene spostata la matrice a destra. Per ricavare invece il valore della cella in basso, la matrice viene spostata in basso di una posizione.

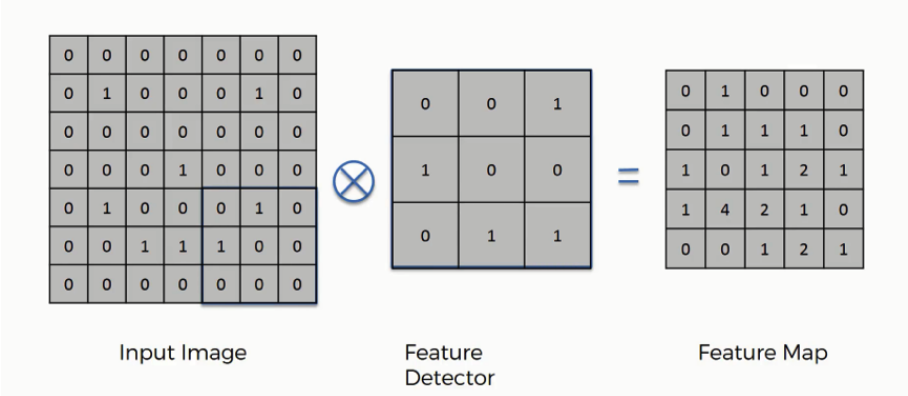

Figura 4.10: Completamento del calcolo della feature map

Così facendo, si possono calcolare i valori della feature map, come mostrato nella figura 4.10. Dopo questa operazione, si realizza una nuova matrice, la quale sarà di grandezza minore della matrice di input. In questo esempio, è stata semplificata l'operazione rispetto a quello che fanno le reti neurali convoluzionali, le quali applicano più kernel per sviluppare più feature map, le quali vengono chiamate *convolutional layer*.

Un esempio di quello che le reti fanno si può vedere nella figura 4.11. La cosa interessante è che i vari kernel vengono trovati, in base a ciò che la rete convoluzionale crede sia importante. La rete di solito riesce a trovare informazioni che all'occhio umano non sono visibili ed è questo uno dei motivi per cui le reti convoluzionali sono così utili.

Il feature detector, chiamato anche kernel oppure filter, potrebbe svolgere ancora un altro ruolo importante. Potrebbe fungere da vero e proprio filtro, motivo per cui viene chiamato anche filter. Per esempio, con dei feature detector particolari, si possono trasformare le immagini come nelle figure 4.12, 4.13, 4.14

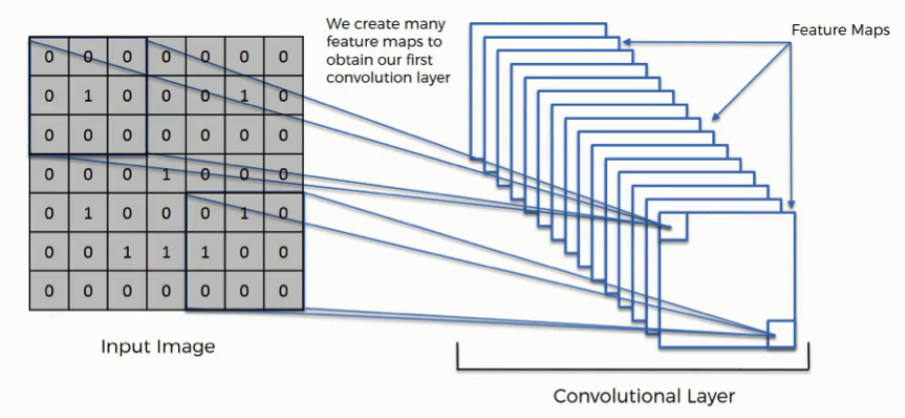

Figura 4.11: Esempio di più feature map applicate in serie

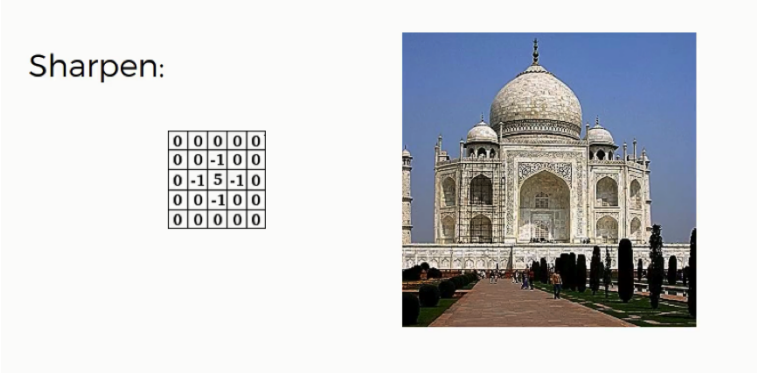

Figura 4.12: Messa a fuoco

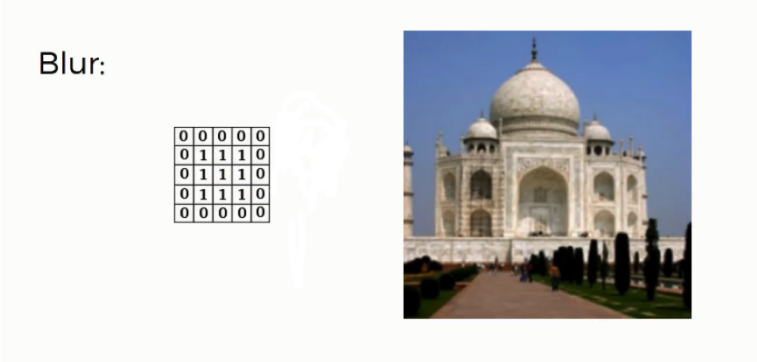

Figura 4.13: Sfocamento

4.1.2 Rectified linear unit (RELU)

Il rectified linear unit è un'operazione supplementare all'operazione di convoluzione. L'obiettivo della funzione rettificatore è quella di aumentare la non-linearità nelle immagini, poiché queste naturalmente sono lineari. Esse sono lineari, poiché

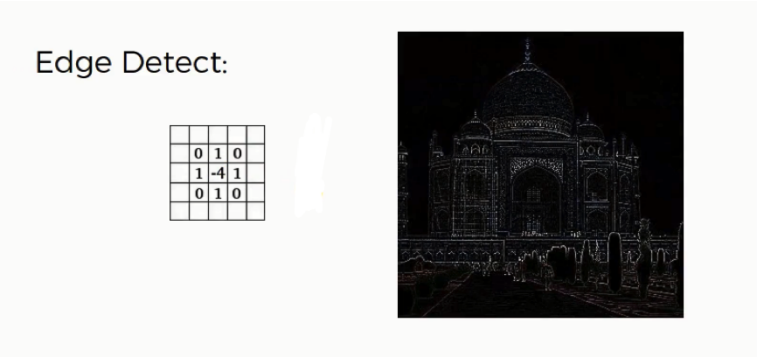

Figura 4.14: Trovare i bordi

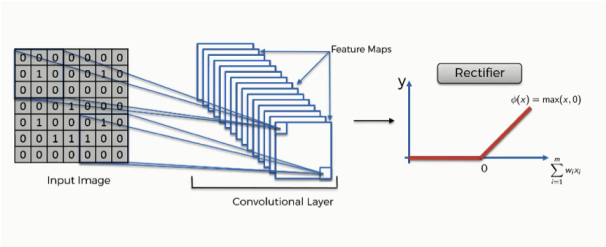

Figura 4.15: Funzione rettificatore

contengono tante caratteristiche come la transizione tra pixel, i bordi, i colori, etc. Per questo motivo, si applica la funzione all'immagine, perché la non-linearità viene imposta durante l'operazione di convoluzione.

Come esempio di questa operazione, si prenda la figura 4.16 come immagine data in input. Essendo questa un'immagine in scala di grigi, verrà data alla funzione

Figura 4.16: Immagine data in pasto alla funzione rettificatore

rettificatore la quale ritornerà una nuova immagine, come si può vedere in figura 4.17, che avrà perso la non-linearità.

Figura 4.17: L'immagine calcolata dalla figura 4.16

In questa nuova immagine, si può notare che i colori cambiano molto più bruscamente. Siccome il cambio graduale dei colori è lineare e quindi è stato tolto, i colori non sono più come nella prima immagine, come di vede in figura 4.16.

4.1.3 Pooling

L'obiettivo della funzione di pooling[48] è rimpiazzare l'output della rete in una certa locazione con un sommario di statistiche degli output vicini. Ci sono diverse funzioni di pooling, tra cui le più usate sono:

- Mean pooling
- Max pooling
- Sum pooling

In questa tesi si prenderà come esempio la funzione **max pooling**, perché questa è la funzione che viene utilizzata di solito per il rilevamento delle immagini, ed è usata anche da Darknet[41].

Max Pooling

L'obiettivo del *max pooling* è quello di trovare l'oggetto voluto nell'immagine, quando l'immagine può assumere varie forme e dimensioni. Di seguito il max pooling sarà spiegato con varie immagini. Presa la figura 4.18, si può vedere che le tre immagini hanno lo stesso animale, anche se sono di varie forme. L'obiettivo del max pooling è quello di capire che in tutte e tre le immagini c'è un ghepardo. Presa la figura 4.19, si può notare che il ghepardo nelle immagini può assumere varie posizioni e

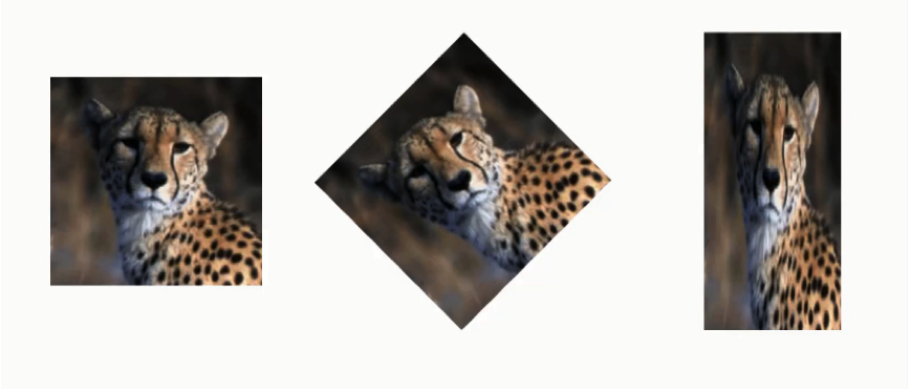

Figura 4.18: Immagine di un ghepardo trasformata in varie forme

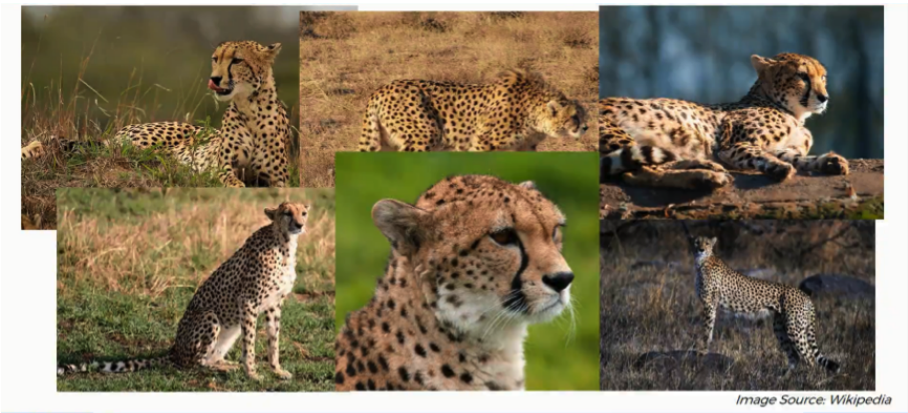

Figura 4.19: Diverse immagini di ghepardo

forme. Il max pooling dovrebbe permettere di capire che anche queste immagini contengono un ghepardo. Siccome utilizziamo il max pooling dopo aver eseguito la convoluzione, si ha già una feature map pronta.

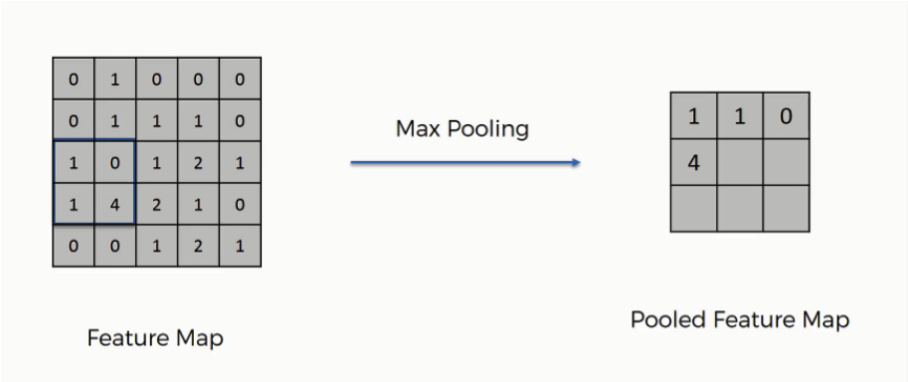

Figura 4.20: Max pooling feature map

Per spiegare come ottenere la pooled feature map verrà presa la figura 4.20,

dove sarà applicata la funzione con una matrice di grandezza 2×2. L'approccio in questione sarà simile all'approccio del passo precedente, il quale ha permesso di calcolare la feature map.

- Applicando la matrice 2×2 sull'angolo superiore sinistro della feature map, con la funzione di max viene trovato il valore massimo. Il seguente valore andrà nella nuova matrice nella posizione in alto a sinistra.
- Rispetto al passo precedente, per trovare gli altri valori la matrice sarà sempre spostata di due posizioni.
- Per trovare il valore nella posizione a destra nella nuova matrice, la matrice verrà spostata a destra.
- Stessa cosa vale quando si calcola la posizione in basso, semplicemente si sposta il quadrato in basso invece che a destra.
- In caso, la grandezza della matrice di feature map non è un multiplo della matrice di max pooling, semplicemente ci spostiamo calcolando i valori anche nel caso in cui la matrice di max pooling esce dalla matrice di feature map. In quel caso saranno calcolati i valori presenti nella matrice, anche se non sarà una matrice intera.

Anche nel caso di max pooling, ci saranno perdite di informazioni. Ciò accade perché vengono selezionati i bit dalla funzione, mentre il resto viene scartato. Il motivo di questa estrazione di pixel è di prendere in considerazione le varie distorsioni che si vuole rimuovere in questo passaggio. Con la rimozione delle distorsioni, vengono rimosse le informazioni che favoriscono il fenomeno dell'overfitting, fenomeno che verrà spiegato in seguito nella sezione 5.6.1

4.1.4 Flattening

Lo step di *flattening* (appiattimento) è uno step che viene effettuato dopo aver effettuato gli step precedenti. Questo step serve ad appiattire una feature map dove si è precedentemente applicato il pooling, come mostrato in figura 4.21.

Il motivo di questa operazione è, che i dati calcolati dopo l'operazione di appiattimento saranno quelli, che verranno utilizzati per darli in pasto alla rete neurale.

Nella figura 4.22 viene mostrato come si applica questa operazione alle multiple feature map calcolate dagli step precedenti. Questa operazione permette di trovare

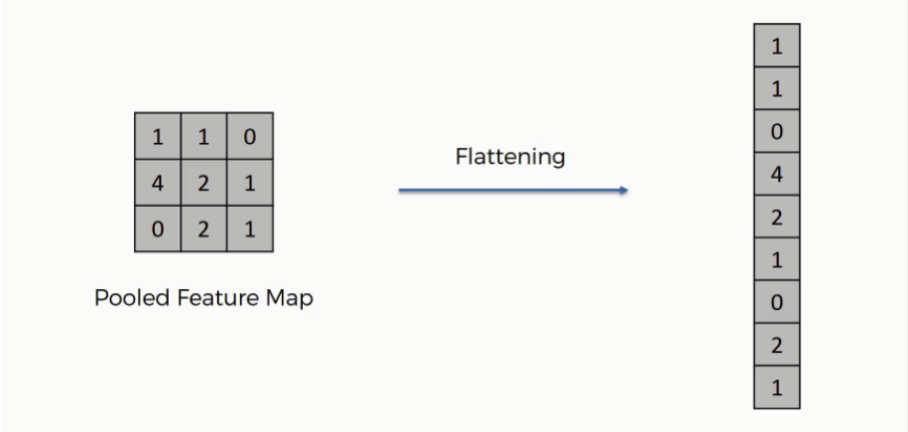

Figura 4.21: Funzione di flattening sulla feature map

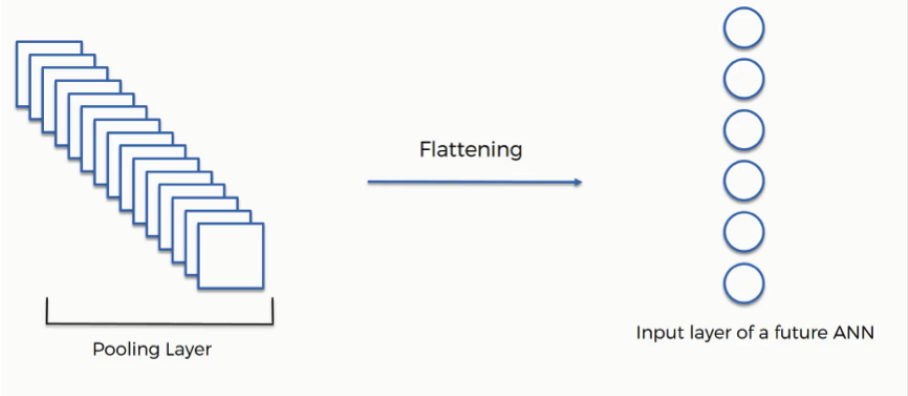

Figura 4.22: Funzione di flattening applicata sui layer

un lungo vettore di dati in input, che viene poi passato alla rete neurale per ulteriori processi.

4.1.5 Connessione totale

A questo punto, si può predire la classe di oggetti che si sta processando. La connessione totale prende in input il vettore di dati creati dallo step precedente, lo step di appiattimento.

Lo scopo della rete neurale è di prendere i dati e combinarli con le caratteristiche di una classe di attributi, i quali faranno sì che la rete sia capace di classificare le immagini.

Si consideri adesso la figura 4.24, sapendo che all'inizio è stata fatta la convoluzione, il pooling, e l'appiattimento adesso verrà eseguito lo step della connessione totale. Questo è lo step che farà le previsioni è poi le darà in output. Nella figura 4.24,

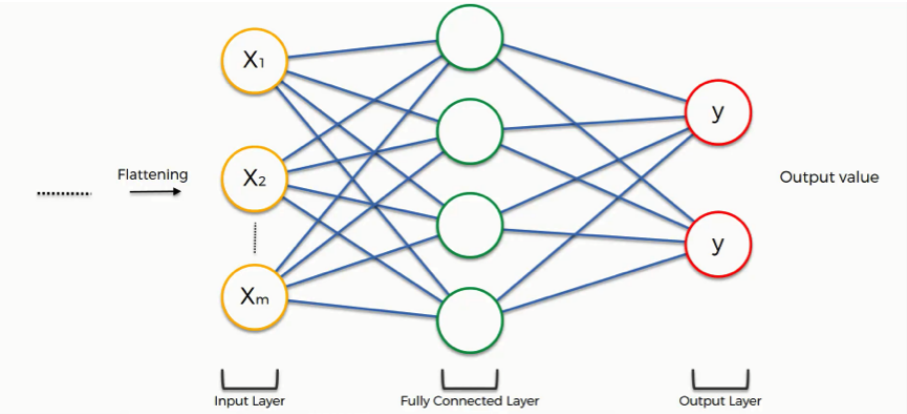

Figura 4.23: Esempio del layer di connessione totale come ultimo layer

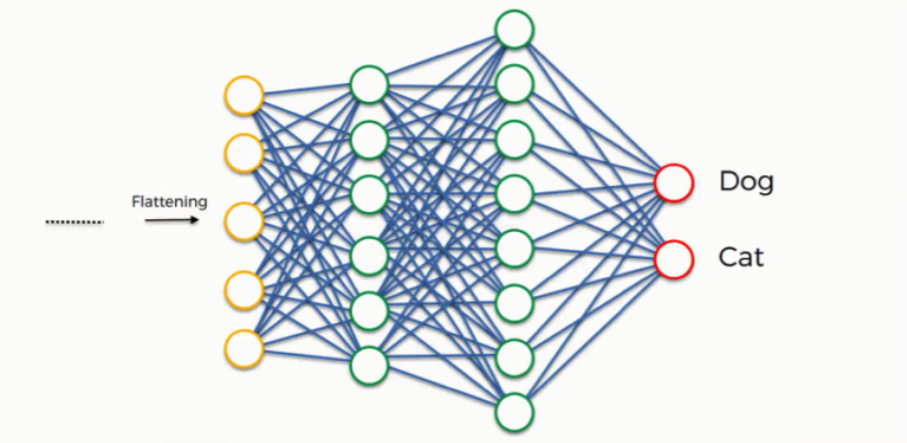

Figura 4.24: Esempio del layer più realistico, non semplificato

si può vedere che la conessione totale elenca le classi che ha trovato, che in questo esempio sono il cane e il gatto.

In ogni caso, la rete può sbagliare. Quindi, se questa rete trova un gatto in un'immagine dove non c'è, ovviamente non è quello che ci si aspetta. Per questi casi, bisogna calcolare un errore, il quale è l'errore medio quadratico chiamato anche "funzione di perdita" (*loss function*) nel contesto di reti neurali convoluzionali. Ciò viene fatto con la *cross-entropy*, la quale dice quanto è precisa la rete nel trovare le classi cercate. La funzione viene inoltre utilizzata per far sì che la rete migliori. Un errore significa che la rete sta sbagliando e che quindi qualcosa deve essere migliorato.

Per la scelta della classe, come mostrato in figura 4.24 l'obiettivo adesso è di trovare la classe più simile all'oggetto in questione. Questo viene fatto controllando i nodi che la indicano. I nodi che indicano una classe sono le caratteristiche che la rete ha trovato. Ogni classe ha una probabilità di appartenenza per ogni caratteristica.

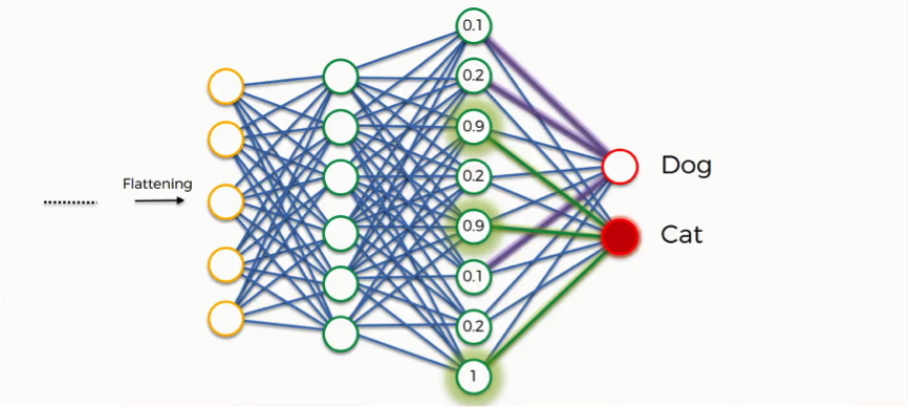

Figura 4.25: Varie probabilità delle caratteristiche calcolate nel layer

Quindi, le classi come prima cosa iniziano a chiedersi, per ogni caratteristica, se questa le appartiene. Una volta che conoscono la probabilità della caratteristica, la riferiscono alle altre. In tal modo la classe che ha reputato con la maggiore probabilità una certa caratteristica la reclamerà come sua, mentre le altre impareranno che quella caratteristica non è loro. Questo fa si che la connessione tra caratteristiche e le classi diventi sempre migliore.

Per esempio, una delle caratteristiche potrebbe essere le orecchie che sono state identificate. Una volta che queste sono identificate, possono assomigliare alle orecchie di un cane o di un gatto, quindi la rete analizza a quale classe potrebbero appartenere (figura 4.25). Se per la classe del cane queste appartengono con il 40% mentre per quella del gatto con il 90%, la classe del gatto prenderà come buona la caratteristica e lo riferirà all'altra classe, in questo caso quella del cane, che imparerà che quelle non sono orecchie di cane.

In questo passo, l'unica cosa che viene mostrata è se una caratteristica appartiene ad una classe e con quale probabilità. Il resto viene eseguito dalla funzione di softmax.

4.1.6 Softmax

Dopo il passo della connessione totale, il passo di softmax permette di coordinare le varie probabilità delle classi, così che sommate diano una probabilità di 1.

Quello che permette di fare la funzione di softmax è calcolare la probabilità di ogni classe. Prima dell'applicazione della funzione, ogni classe sa con quale probabilità ogni caratteristica appartiene ad essa. Quello che non si sa prima dell'applicazione di softmax per esempio prendendo una foto con una persona e un gatto, è quanto

sia probabile che l'oggetto analizzato sia il gatto o la persona. Per calcolare questa probabilità viene utilizzata la funzione nell'equazione 4.9, la funzione di softmax:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{4.9}$$

Con l'applicazione di softmax, si evita che nel risultato venga dato come risposta, per esempio, che l'oggetto in questione ha una probabilità dell'80% di essere una persona e del 50% di essere un gatto. Infatti, questo risultato è abbastanza inutile, mentre grazie alla funzione di softmax, questo problema viene risolto. Softmax, infatti, normalizza gli output in modo che sommino ad uno e possano essere interpretati come probabilità. Di conseguenza, in caso la probabilità che sia una persona fosse dell'80%, la probabilità di essere un gatto sarebbe del 20%, questo supponendo che ci sia solo la classe gatto, oppure che le restanti siano dello 0%.

4.1.7 Cross-entropy

Una funzione che viene utilizzata per sapere quanto è precisa la rete è la funzione di cross-entropy[11], che ha le seguenti equazioni:

$$L_i = -log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{4.10}$$

$$H(p,q) = -\sum_i p_i \log q_i \tag{4.11}$$

Quando viene utilizzata la funzione nell'equazione 4.11, q rappresenta la probabilità della classe e p la presenza (valore 1) o l'assenza (valore 0).

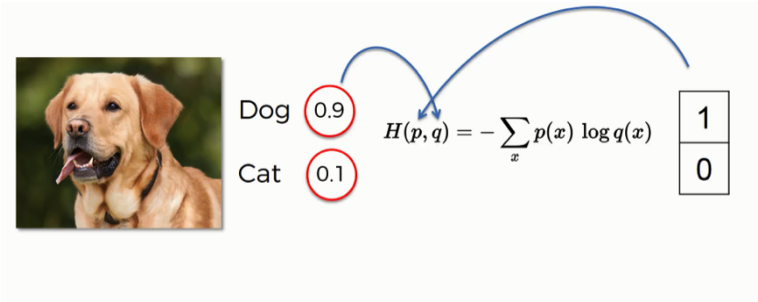

Figura 4.26: Dimostrazione della funzione di cross-entropia

Si prenda, come esempio, la figura 4.26. Si vede che la rete ha generato una probabilità del 90% per il cane e 10% per il gatto, e si nota la presenza/assenza a destra della funzione nell'equazione 4.11.

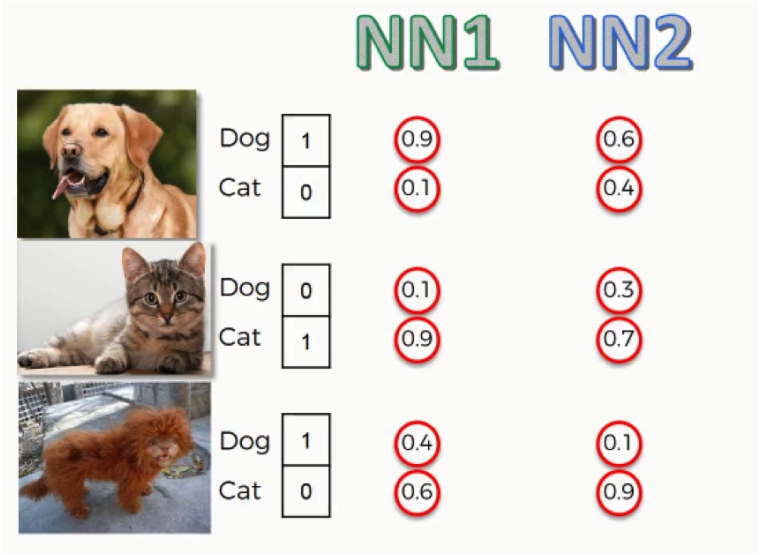

Figura 4.27: Risultati di due reti neurali su varie immagini

Un'altra informazione interessante che possiamo sapere utilizzando questa formula è di quanto sbagli la rete. Ovviamente, la gravità dell'errore deve essere quantificata. Quindi, come esempio verrà presa la figura 4.27. Si può evincere che le due reti in questione hanno prodotto gli stessi risultati, però le probabilità delle classi sono diverse. Guardando le probabilità, si nota che la rete NN1 è più precisa della NN2 nei primi due casi, mentre nel terzo caso, questa sbaglia sì, ma con un errore minore.

$$J(w) = \frac{1}{N} \sum_{n=1}^{N} H(p_n, q_n) \tag{4.12}$$

L'equazione 4.12 mostra come si utilizza la cross-entropy quando si ha più immagini per calcolare la funzione che comprende tutte e farsi un'idea della rete con più test.

- L'*errore di classificazione* è un parametro che dice quante volte ha sbagliato la rete neurale a scegliere la classe appropriata. In questo caso, si noti che entrambi le reti sbagliano una volta, quindi in ambo i casi si ha una percentuale del 33%.

- Lo *errore quadratico medio* è derivato dagli errori che la rete fa e poi calcola la media di questi. Infatti, si può notare che la prima rete è più precisa della seconda.

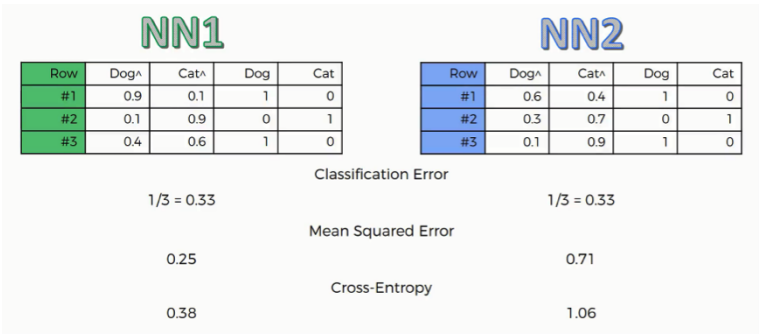

Figura 4.28: Probabilità delle due reti neurali con i label corretti

- Come esempio si prenda la figura 4.28 e si applichi la funzione dell'equazione 4.12:

$$J(w) = \frac{1}{3}(H(p_1, q_1) + H(p_2, q_2) + H(p_3, q_3)) \tag{4.13}$$

$$J(w) = \frac{1}{3}(0.10536 + 0.10536 + 0.91629) = 0.38 \tag{4.14}$$

$$J(w) = \frac{1}{3}(0.51083 + 0.35667 + 2.30259) = 1.06 \tag{4.15}$$

  Così facendo si ricava il valore di *cross-entropy*.

Applicata la funzione su entrambe le reti neurali, si può ottenere la funzione dell'equazione 4.14 e la funzione dell'equazione 4.15.

4.2 Binary Cross-entropy

La binary cross-entropy di solito viene utilizzata per i classificatori binari, cioè nei casi in cui si decide se una caratteristica appartiene oppure no.

Si prenda la funzione dell'equazione 4.16 che ha 10 elementi come esempio per mostrare il funzionamento della cross-entropy binaria.

$$x = [-2.2, -1.4, -0.8, 0.2, 0.4, 0.8, 1.2, 2.2, 2.9, 4.6] \tag{4.16}$$

Disegnando la funzione precedente, si ritrova la figura 4.29.

Una volta ottenuto questo, per esempio, si colorino certi punti in verde e i restanti punti in rosso, così come nella figura 4.30.

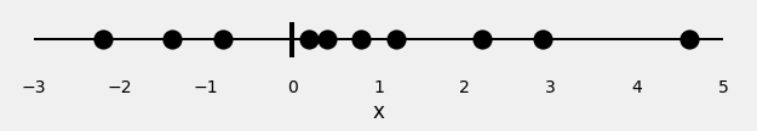

Figura 4.29: La funzione 4.16 con 10 elementi

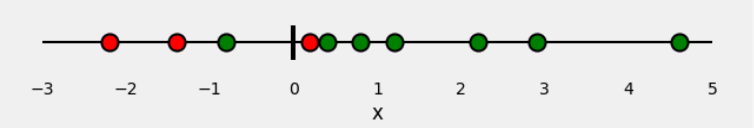

Figura 4.30: Funzione dell'equazione 4.16 con 10 elementi colorata

Così facendo, viene ottenuta la funzione alla quale verrà applicata la cross-entropy binaria. Questa infatti permette di rispondere, per esempio, alle seguenti domande:

"Il punto è verde?"

"Qual è la probabilità che un punto sia verde?"

Idealmente i *punti verdi* hanno probabilità 1.0 di essere verdi ed i *punti rossi*, hanno probabilità 0.0 di essere verdi.

La funzione è chiamata binaria, in quanto risponde al quesito di appartenenza. In questo esempio, i nodi verdi appartengono alla classe positiva, mentre i nodi rossi, appartengono alla classe negativa. Questo perché si cercavano i nodi verdi. Preso un modello per effettuare la classificazione, si ha un modello che predice la probabilità di essere verde per ogni nodo. In caso si sappia di che colore siano effettivamente i nodi si può valutare quanto sbaglia il modello.

Per questa domanda, ovvero valutare quanto sbaglia il modello, si utilizza la "funzione di perdita"[54] (loss function).

Loss function

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i)) \tag{4.17}$$

La seguente funzione dell'equazione 4.17 è la funzione di perdita. Si ha y come *label*, cioè il valore 1 per i punti verdi e il valore 0 per i punti rossi. Continuando con l'esempio della sezione 4.2, dell'entropia incrociata, la funzione p(y) invece è la probabilità che il sistema predica che il punto è verde per tutti i N punti.

Se viene applicata la formula all'esempio dei punti verdi e rossi di prima, dove vengono presi come positivi i punti verdi. Quello che viene fatto è: per ogni punto

verde, dove $(y = 1)$, si aggiunge $\log(p(y))$, cioè la *probabilità logaritmica che il punto sia verde*. Inoltre, viene aggiunto $\log(1 - p(y))$, che è la probabilità logaritmica che il punto sia rosso per ogni punto *rosso* dove si ha $y = 0$.

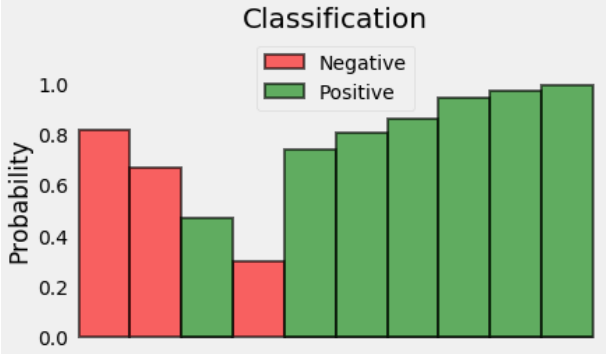

Figura 4.31: Grafico con le probabilità degli elementi positivi e quelli negativi

Si prendano adesso tutte le probabilità che sono state calcolate con le probabilità logaritmiche appena spiegate e per comodità le si riposizioni su un grafico. (figura 4.31). Quello che si vuole, fare conoscendo le probabilità, è penalizzare le decisioni sbagliate. Conoscendo le *probabilità*, è possibile vedere che se la probabilità classe positiva è 1 la perdita deve essere 0. Quindi, ogni volta che la probabilità che un nodo sia positivo è alta, la perdita sarà il contrario, quindi bassa. Queste probabilità saranno utilizzate nella funzione dell'equazione 4.17 e, siccome i valori dei logaritmi tra 0 è 1 sono negativi verrà negato come nella formula il risultato. Nella figura4.32 si possono vedere i valori di x ai quali e stata applicata $-\log(p(x))$.

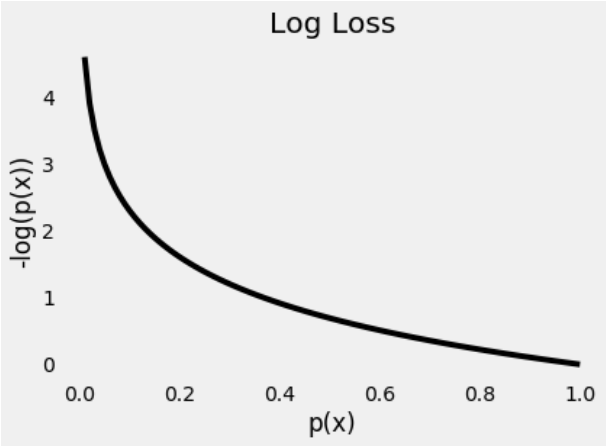

Figura 4.32: Funzione logaritmica applicata alle probabilità della funzione

Se adesso applichiamo $-\log(p(x))$ ad ogni probabilità, ci ritroviamo con le seguenti probabilità, mostrate nella figura 4.33 per comodità.

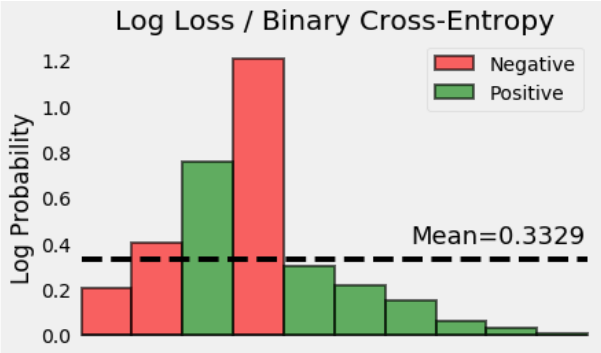

Figura 4.33: Grafico delle probabilità una volta applicata la funzione di perdita

4.3 Yolo

In questa sezione verrà presentata l'ultima versione di Yolo, ovvero la terza, che utilizza la rete neurale Darknet-53[41] come classificatore. Di conseguenza, quando ci si riferisce a Yolo ci si riferisce alla sua terza versione e non alla prima.

4.3.1 Predizione delle classi

Per la predizione delle classi, Yolo rispetto alla maggior parte dei detector assume che le etichette di output non siano mutualmente esclusive. Per l'esclusività delle etichette di output si intende che un box non può appartenere a due classi diverse. Per esempio, se il detector assume che i box sono mutualmente esclusivi, una box etichettata con *pedone* e *bambino* non è accettabile. Per Yolo, invece, questa predizione è legittima. Per tale motivo non è più inclusa la funzione di softmax, ma lo è invece quella del independent logistic classifier, la quale permette una classificazione multi-label.

Calcolare la perdita della classificazione utilizza la binary cross-entropy per ogni label. La quale riduce la complessità evitando così la funzione e soprattutto softmax.

4.3.2 Bounding box prediction

Yolo predice un punteggio per ogni "bounding box", utilizzando la logistic regression. Oltre a ciò, il modo in cui calcola la funzione di perdita dipende da:

- se la bounding box trovata sovrappone il *ground truth* object più delle altre bounding box allora il punteggio dell'oggetto vale a 1;
- per le restanti con sovrapposizione maggiore di una soglia prestabilita, Yolo ha come default 0.5 e non vengono penalizzate;

- Ogni ground truth object è associato soltanto con la bounding box precedente;
- Nel caso un bounding box precedente non è assegnato, non viene persa la classificazione e della localizzazione, quello che succede e il perdersi della confidenza sull'oggetto;
- per commutare la perdita, si utilizza la t_x e la t_y (invece di) b_x e b_y

Per annotare gli oggetti, il sistema predice quattro coordinate per ogni "box": t_x, t_y, t_w, t_h. Queste coordinate hanno un offset, indicato con (c_x, c_y), dall'angolo superiore sinistro dell'immagine. Mentre p_w, p_h corrispondono alla larghezza e all'altezza della predizione del precedente "box".

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

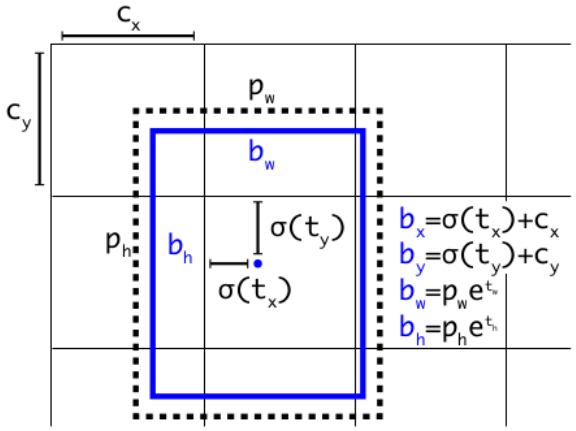

Figura 4.34: Annotazione di un box

4.3.3 Feature extractor

Per estrarre le caratteristiche, il modello utilizza Darknet-53, il quale è l'implementazione di Darknet con 53 strati. Darknet è composto principalmente da filtri 3×3 e 1×1 con skip connections come nella residual network in ResNet utilizzando così

il residual layer. Darknet-53 è molto ottimizzato, ha meno BFLOP[1] [57] di ResNet-152 ma ha una precisione alla pari. Questo gli permette di classificare le classi nella metà del tempo rispetto a ResNet-152.

4.3.4 Performance di Yolo

Come esempio, verrà presa la figura 4.36 dove ci sono più modelli i quali vengono testate le performance sul database di immagini COCO[29]. Come si può vedere, Yolo ha una AP (average precision) simile al modello SSD[31], sebbene sia 3× più veloce. Un'altra cosa che si può vedere è che Yolo non ha una precisione alla pari di RetinaNet, specialmente nel caso di AP@IoU=.75, dove RetinaNet dimostra di essere il più preciso dei modelli. Ciò che si può notare, però, è che Yolo è migliorato notevolmente rispetto alla versione precedente. Inoltre, è migliorato moltissimo nella detection di oggetti piccoli.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figura 4.35: Grafico con le differenze dei vari detector con i rispettivi classificatori

Non considerando soltanto la precisione e quindi analizzando la velocità, si può osservare il grafo nella figura 4.36, da cui si può dedurre che Yolo è molto più veloce degli altri modelli. Questo fa sì che Yolo è un buon modello, dove la velocità di detection è importante.

Inoltre, essendo Yolo basato su Darknet-53, ciò si può vedere nella tabella 4.2 che è molto performante.

[1] Bilion Floating Point Operations

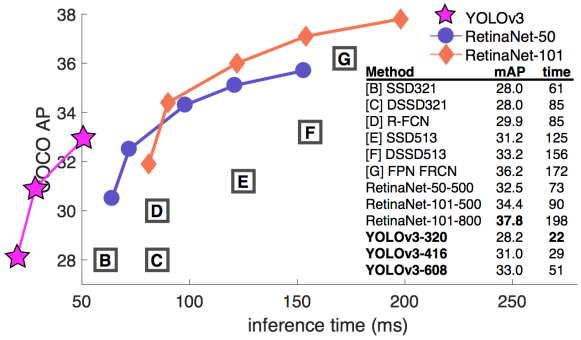

Figura 4.36: La velocità di Yolo rispetto ad altri modelli

Tabella 4.1: Performance sul dataset COCO

model	train	mAP	FLOPS	FPS
SSD300	COCO	41.2	-	46
SSD500	COCO	46.5	-	19
YOLOv2 608x608	COCO	48.1	62.94 Bn	40
Tiny YOLO	COCO	23.7	5.41 Bn	**244**
SSD321	COCO	45.4	-	16
DSSD321	COCO	46.1	-	12
R-FCN	COCO	51.9	-	12
SSD513	COCO	50.4	-	8
DSSD513	COCO	53.3	-	6
FPN FRCN	COCO	59.1	-	6
Retinanet-50-500	COCO	50.9	-	14
Retinanet-101-500	COCO	53.1	-	11
Retinanet-101-800	COCO	57.5	-	5
YOLOv3-320	COCO	51.5	38.97 Bn	45
YOLOv3-416	COCO	55.3	65.86 Bn	35
YOLOv3-608	COCO	57.9	140.69 Bn	20
YOLOv3-tiny	COCO	33.1	5.56 Bn	220
YOLOv3-spp	COCO	**60.6**	**141.45 Bn**	20

Tabella 4.2: La precisione calcolata su Imagenet

Model	top-1	top-5	Ops	GPU	CPU
Alexnet	57.0	80.3	2.27 Bn	3.1 ms	0.29s
Darknet Reference	61.1	83.0	**0.96 Bn**	**2.9 ms**	**0.14 s**
VGG-16	70.5	90.0	30.94 Bn	9.4 ms	4.36s
Extraction	72.5	90.8	8.52 Bn	4.8 ms	0.97s
Darknet19	72.9	91.2	7.29 Bn	6.2 ms	0.87s
Darknet19 448x448	76.4	93.5	22.33 Bn	11.0 ms	2.96s
Resnet 18	70.7	89.9	4.69 Bn	4.6 ms	0.57s
Resnet 34	72.4	91.1	9.52 Bn	7.1 ms	1.11s
Resnet 50	75.8	92.9	9.74 Bn	11.4 ms	1.13s
Resnet 101	77.1	93.7	19.70 Bn	20.0 ms	2.23s
Resnet 152	77.6	93.8	29.39 Bn	28.6 ms	3.31s
ResNeXt 50	77.8	94.2	10.11 Bn	24.2 ms	1.20s
ResNeXt 101 (32x4d)	77.7	94.1	18.92 Bn	58.7 ms	2.24s
ResNeXt 152 (32x4d)	77.6	94.1	28.20 Bn	73.8 ms	3.31s
Densenet 201	77.0	93.7	10.85 Bn	32.6 ms	1.38s
Darknet53	77.2	93.8	18.57 Bn	13.7 ms	2.11s
Darknet53 448x448	**78.5**	**94.7**	56.87 Bn	26.3 ms	7.21s

5

Deep Learning per la detection dei siti di phishing

In questo capitolo sarà presentato il vero scopo di questa tesi. Verrà spiegato come è stato applicato il deep learning al problema riscontrato. Oltre alla sua applicazione, verrà spiegato il motivo di questa scelta rispetto agli altri approcci. Infine, siccome durante la risoluzione del problema sono stati riscontrati vari problemi, ne saranno spiegate le soluzioni e i motivi di queste soluzioni.

5.1 La problematica alla quale è stato applicato il deep learning per il rilevamento degli oggetti

Dato un elenco di url di sospetti siti di phishing, questi ultimi dovevano essere controllati per verificare che fossero veri siti di phishing o soltanto falsi positivi. L'elenco in questione veniva aggiornato periodicamente, con aggiornamenti che aggiungevano migliaia di nuovi url, e per tale motivo non poteva essere controllato manualmente. Era necessario, invece, trovare un modo per automatizzare, o almeno semi-automatizzare, il processo di controllo. Come soluzione al problema, è stato pensato un approccio basato sul *deep learning*, sfruttando gli algoritmi di object detection per risolverlo. Il presente approccio è stato reputato il più adatto perché esso può imparare dalle pagine dategli come esempi e poi utilizzarle per prendere le decisioni sulle pagine future. Senza l'ausilio del *deep learning*, sarebbe stato difficile definire parametri di controllo per le pagine future, le quali non si sa neanche come sono strutturate, ma si sa soltanto che sono simili alle originali.

5.2 Come è stato risolto il problema

Una volta deciso l'utilizzo del *deep learning*, sono emerse le seguenti domande:

- *Quale algoritmo sarà utilizzato per la risoluzione del problema?*
- *Cosa analizzerà l'algoritmo nelle pagine di Deep learning?*
- *Come gli verranno passate le immagini delle pagine?*
- *Siccome l'algoritmo avrà bisogno di essere addestrato, cosa gli si darà come esempi?*
- *Quanto grande dovrà essere il dataset?*
- *Quanta potenza di calcola servirà all'algoritmo?*

Oltre a queste domande, ne vennero prese in questione altre. Nelle seguenti sezioni ci saranno le risposte e le soluzioni ai vari problemi riscontrati.

5.3 Decisione dell'algoritmo

Innanzitutto, per la risoluzione del problema doveva essere scelto un algoritmo appropriato, che doveva essere:

- **veloce**: veloce perché doveva finire in tempi ragionevoli;
- **abbastanza preciso**: preciso perché altrimenti sarebbe stato inutile, ma evitando di puntare soltanto alla precisione a discapito della velocità.

Questi ragionamenti sono stati fatti siccome il lavoro era un aspetto che andava combinato con altri, quindi la precisione assoluta non era necessaria, essendo il risultato finale tratto dalle varie parti.

Dopo aver letto i vari test è stato scelto Yolo[44], perché Yolo è uno degli algoritmi più veloci, e oltre alla velocità è molto preciso, come si evince nella tabella 4.1. La terza versione di Yolo è stata pubblicata pochi mesi prima di questo lavoro e l'algoritmo è migliorato per la *detection* su oggetti di piccole dimensioni, il che ha reso Yolo molto più preciso.

Inoltre Yolo utilizza la rete convoluzione darknet-53[41], e come possiamo vedere anch'essa è molto performante.

A parte le questioni di performance, tali scelte sono state fatte anche per via delle dimensioni delle immagini. Darknet, infatti, è abbastanza buona per lavorare con immagini di grosse dimensioni.

5.4 Dataset

Per ogni immagine si sono identificate le coordinate di ogni logo in considerazione. Quest'informazione assieme all'informazione su "quale logo è" è stata salvata nei file di label utilizzati durante il training per determinare gli errori. Per capire il motivo della creazione del dataset per gli algoritmi supervisionati è possibile consultare la sezione 3.1.

5.5 Training

Una volta scelto un dataset iniziale per far imparare il modello, doveva incominciare la fase di training. Inizialmente il training è stato effettuato usando la CPU. Dopo i primi tentativi si è stimato che il training sul dataset usando la CPU si sarebbe completato in maniera completa nell'ordine di mesi. Il problema fu poi risolto utilizzando una macchina dotata di GPU per il training, problematica che verrà spiegata nella sezione 5.5.1. Grazie a questa macchina, il modello poteva imparare più velocemente e poteva essere addestrato in un paio di giorni.

Inoltre, durante la fase di training il modello è stato osservato 5.5.3, per valutare la sua precisione nella localizzazione con l'indice di Jaccard 5.5.2.

5.5.1 La potenza di calcolo richiesta dall'apprendimento

Uno dei problemi del *machine learning*, e ancora di più del *deep learning*, è la potenza di calcolo richiesta per far sì che il programma impari ad eseguire il compito in tempi ragionevoli. Questo è uno dei problemi che ha rallentato lo sviluppo del machine learning e di conseguenza del *deep learning*. Anche se i processori sono sempre più potenti, non sono abbastanza potenti e già dall'inizio dello scorso decennio le gpu (schede video), le quali diventarono abbastanza comuni, stanno prendendo piede nel machine learning. Già nell'anno 2001[15], le persone iniziavano a programmare cose diverse dalla grafica sfruttando le schede video.

Le schede video aiutano il machine learning rendendolo più veloce, perché le schede video sono progettate per fare bene i calcoli in parallelo. Possono infatti eseguire calcoli di matrici in tempi brevissimi. Basti pensare che le schede video, se usate per la grafica, devono calcolare i vari valori dei pixel in tempi brevissimi. Per questo motivo già nel 2005 iniziarono a utilizzare le gpu per il machine learning[50].

Anche se la potenza di calcolo con il passare del tempo aumenta, il problema è che gli algoritmi diventano sempre più complessi, richiedendo quindi più potenza di calcolo.

5.5.2 Indice di Jaccard

L'*indice di Jaccard*[40], noto anche come *coefficiente di similarità di Jaccard*, è un indice statistico utilizzato per confrontare la similarità e la diversità di insiemi campionari, originalmente chiamato *coefficient de communauté* da Paul Jaccard nell'articolo del 1901[22].

Il coefficiente misura la similarità tra insiemi campionari. È definito come la dimensione dell'*intersezione* divisa per la dimensione dell'*unione* degli insiemi campionari:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

L'*indice di Jaccard* nel machine learning viene utilizzato per misurare l'accuratezza di un algoritmo di "object detection" su un dataset. Questo "indice" viene utilizzato per misurare la performance degli algoritmi di rilevazione delle *reti neurali convoluzionali* (per esempio R-CNN, Faster R-CNN, Yolo, etc.). La performance dell'algoritmo viene calcolata con l'indice di Jaccard (5.1), dove come A e B sono il "box reale" e il "box predetto" rispettivamente.

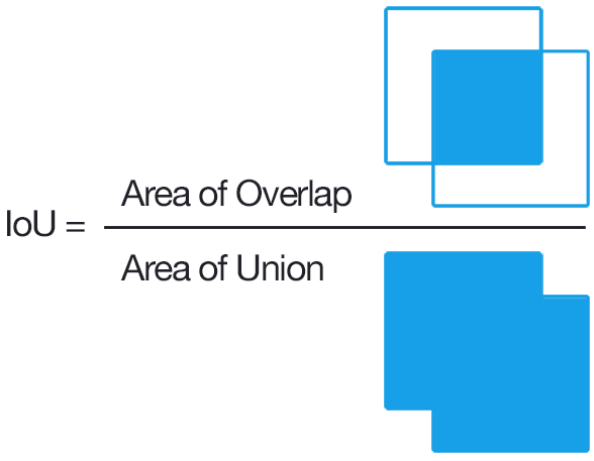

Figura 5.1: L'indice di Jaccard pure chiamato IoU (Intersection over Union) nel *deep learning*

La figura 5.1 rappresenta l'indice di Jaccard, con i due bounding box. Per il calcolo dell'indice di Jaccard non è importante l'algoritmo che genera le predizioni,

l'indice infatti ci dice soltanto la precisione, la quale rappresenta la percentuale di quanto i box si sovrappongono. Per questo motivo l'indice viene utilizzato per scoprire quale algoritmo di rivelazione è più preciso.

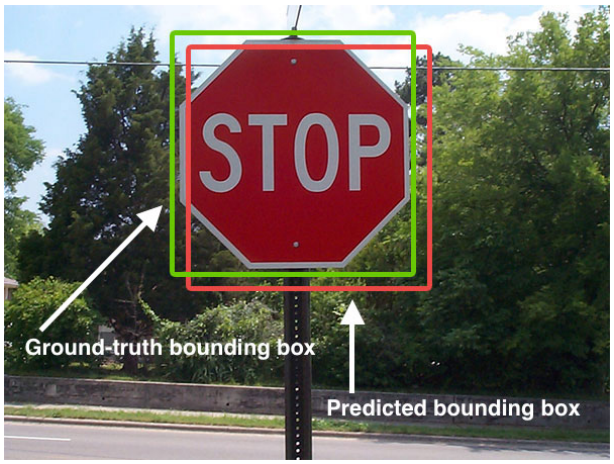

Figura 5.2: Esempio reale di utilizzo dell'indice di Jaccard

Per fare un esempio reale di utilizzo si prenda l'immagine nella figura 5.2. Qui il box verde chiamato "ground-truth bounding box" indica il box reale, mentre quello rosso chiamato "predicted bounding box" indica il box predetto dal modello. Questi sono i box ai quali si applica l'indice di Jaccard.

Figura 5.3: Esempio di calcolo dell'indice con la rispettiva valutazione

L'indice di Jaccard ci permette di conoscere la precisione di ogni predizione degli oggetti nell'immagine. Siccome nel *deep learning* ogni oggetto sarà predetto più volte dal modello con "bounding box" in diverse posizioni, l'indice di Jaccard aiuterà a scartare i box con una bassa precisione. Questo fa sì che il sistema troverà soltanto il box nella posizione più corretta possibile.

5.5.3 Controllo della localizzazione

Il comando usato per eseguire il training con il programma è il seguente:

```
$ logo-darknet train --config yolov3.cfg \
  --data-config loghi.data --weights yolov3-final.weights
```

Durante il training il modello ritornava informazioni utili nello stdio

```
progress of learning

Loading weights from yolov3-final.weights...Done!
Learning Rate: 0.0001, Momentum: 0.9, Decay: 0.0005
Loaded: 0.588000 seconds
Region Avg IOU: 0.131223, Class: 0.372192, Obj: 0.612942, No Obj: 0.512942,
    Avg Recall: 0.000000, count: 10
Region Avg IOU: 0.241324, Class: 0.432192, Obj: 0.532192, No Obj: 0.514123,
    Avg Recall: 0.061293, count: 33
Region Avg IOU: 0.192344, Class: 0.512352, Obj: 0.572319, No Obj: 0.513006,
    Avg Recall: 0.178888, count: 14
Region Avg IOU: 0.191452, Class: 0.321482, Obj: 0.642192, No Obj: 0.514102,
    Avg Recall: 0.000000, count: 21
Region Avg IOU: 0.182402, Class: 0.532148, Obj: 0.572312, No Obj: 0.514012,
    Avg Recall: 0.084333, count: 26
Region Avg IOU: 0.145242, Class: 0.572142, Obj: 0.532129, No Obj: 0.513219,
    Avg Recall: 0.000000, count: 12
Region Avg IOU: 0.145214, Class: 0.424299, Obj: 0.492381, No Obj: 0.512931,
    Avg Recall: 0.000000, count: 17
Region Avg IOU: 0.155258, Class: 0.429921, Obj: 0.523192, No Obj: 0.513492,
    Avg Recall: 0.000000, count: 12
1: 1525.245972 seconds, 76 images
....
....
Region Avg IOU: 0.782192, Class: 0.999982, Obj: 0.879291, No Obj: 0.008729,
    Avg Recall: 1.000000, count: 18
Region Avg IOU: 0.784929, Class: 0.997991, Obj: 0.803823, No Obj: 0.009992,
    Avg Recall: 1.000000, count: 33
Region Avg IOU: 0.720021, Class: 0.996832, Obj: 0.759232, No Obj: 0.007929,
    Avg Recall: 0.872911, count: 21
Region Avg IOU: 0.802192, Class: 0.999598, Obj: 0.842991, No Obj: 0.009233,
    Avg Recall: 1.000000, count: 20
Region Avg IOU: 0.780121, Class: 0.998092, Obj: 0.880122, No Obj: 0.005992,
    Avg Recall: 1.000000, count: 10
Region Avg IOU: 0.759912, Class: 0.999321, Obj: 0.719932, No Obj: 0.012042,
    Avg Recall: 0.970123, count: 35
Region Avg IOU: 0.759829, Class: 0.997212, Obj: 0.749238, No Obj: 0.015021,
    Avg Recall: 0.982102, count: 45
Region Avg IOU: 0.760021, Class: 0.999923, Obj: 0.823942, No Obj: 0.005354,
    Avg Recall: 1.000000, count: 12
207: 1602.0022 seconds, 14921 images
```

Durante l'esecuzione del precedente comando si può vedere che l'indice di Jaccard oppure IOU, cresce. Ciò significa che i box trovati dal modello sono più precisi. Per il presente esperimento, l'indice di Jaccard non era tanto importante, in quanto lo scopo del programma era quello di trovare i loghi nelle immagini e non localizzarli con precisione. Per questo motivo, non sono stati fatti troppi test per la precisione.

5.6 Validazione e test

Durante la validazione è stato osservato se capitano fenomeni di overfitting e di underfitting, come spiegati nella sezione 5.6.1.

Per i calcoli della precisione, invece, sono state utilizzate pagine con loghi per i quali il modello è stato addestrato, ovviamente utilizzando immagini non presenti nel training. Per le immagini vengono presi esempi di oggetti diversi per rispettare

la privacy delle aziende, in quanto il modello è utilizzato con clienti veri. I test completi si possono vedere nella sezione 5.6.2.

5.6.1 L'overfitting e l'underfitting

Quando si parla di underfitting, si parla delle reti neurali convoluzionali che sono state addestrate. Quello che si può dire dell'overfitting e dell'underfitting è che questi problemi sono opposti.

Presa una rete convoluzionale addestrata, l'underfitting è il fenomeno che capita quando alla rete vengono date le immagini per la rilevazione, e:

- su queste immagini è più preciso, per rilevare gli oggetti, rispetto alle immagini del training set, oppure
- il modello non è preciso.

Se per esempio viene utilizzato un modello lineare per le immagini, il fenomeno di underfitting si verificherà sempre. Di solito, il fenomeno in questione succede per colpa dell'abbandono degli attivatori ("dropout"), i quali sono introdotti appositamente per evitare il fenomeno opposto, l'overfitting. Gli attivatori non sono presenti durante la validazione e per la detection di immagini, mentre ci sono durante la fase di test. Questo è il motivo per il quale la detection funziona meglio rispetto al training.

Quando si verifica l'underfitting, ciò che di solito si fa è ridurre il numero di attivatori, oppure rimuoverli del tutto. In caso vengano rimossi troppi attivatori, si verificherebbe il fenomeno opposto, ma il fenomeno dell'overfitting è più facilmente gestibile. In seguito, sarà spiegato come eliminare il fenomeno di overfitting.

L'*overfitting* si verifica quando il modello è molto preciso con il training set, ma inizia a diventare sempre peggiore con altri insiemi di dati. Questo si può osservare nella figura 5.4, dove gli errori, dopo che il modello è stato addestrato troppo, iniziano ad aumentare sempre di più. L'overfitting si riscontra invece quando il modello inizia a imparare le immagini del training set "a memoria" e quindi non le analizza più analizzandone le caratteristiche. Per risolvere l'overfitting si può procedere nei seguenti modi:

1. Aggiungere dati;
2. Applicare il data augmentation;

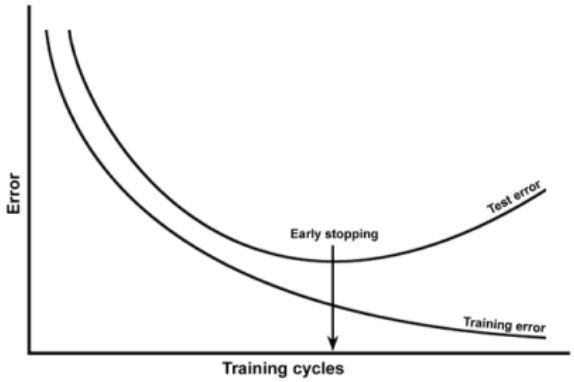

Figura 5.4: Grafo della precisione

3. Utilizzare architetture che generalizzano bene;
4. Introdurre regole di "dropout";
5. Ridurre la complessità dell'architettura.

Si assuma che per un modello si verifichi l'overfitting. Sarà spiegato come risolverlo utilizzando le tecniche appena elencate e per ogni passo sarà considerato che sono già stati applicati i passi precedenti, dato che è questo l'ordine in cui di solito vengono applicati.

1. Come primo passo, si può aggiungere più dati al modello. Così facendo per il modello sarà più difficile "impararlo a memoria". Il problema di questa soluzione è che per svariati motivi, di solito, non si può aggiungere abbastanza dati;
2. Alternativamente all'aggiunta di dati, si può utilizzare il data augmentation. Questa tecnica ci permette di aumentare l'insieme dei dati per il training senza raccoglierne di nuovi. Quello che fa il data augmentation è prendere il dataset e creare per ogni immagine delle copie, dove la copia di ogni immagine sarà l'immagine ruotata, zoomata oppure a cui è stato applicato un filtro di colori. Questa operazione è visibile in figura 5.5. Dopo l'applicazione del data augmentation, è abbastanza comune vedere che il dataset diventi molto più grande. Per il data augmentation non bisogna mai esagerare con le modifiche. Per esempio, un'immagine troppo zoomata è inutile, perché in questo caso le caratteristiche che la rete dovrebbe trovare potrebbero non esserci essendo stati tagliati fuori dall'immagine i pezzi in cui c'erano le caratteristiche;

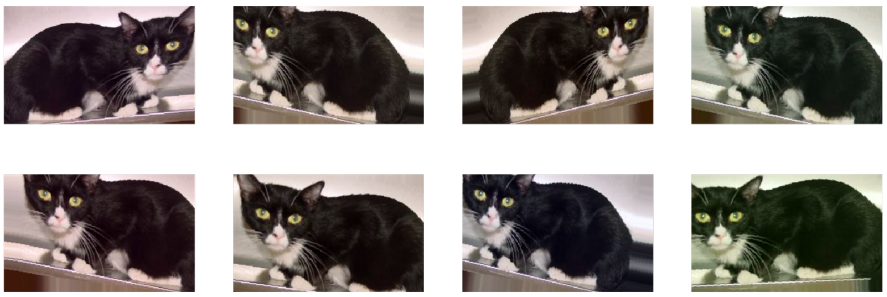

Figura 5.5: Immagine del gatto alla quale è stato applicato il data augmentation

3. Un'altra opzione sarebbe quella di cambiare l'architettura e utilizzarne una che generalizza meglio. Questo punto non è così fondamentale, infatti è probabile che venga utilizzata una rete neurale che generalizza già bene e i problemi siano del dataset e quindi risolvibili con i passi precedenti;

4. Nel caso il fenomeno sia ancora presente, si potrebbe iniziare ad aggiungere le regole di *dropout*, dove si esegue l'opposto di quanto descritto prima per la soluzione dell'underfitting, in cui sono state tolte le regole di dropout. Un dropout in machine learning rimuove un campione casuale di attivazione (resettandolo a zero) durante il training. Il problema di questa tecnica è che ogni dropout causa perdite di informazioni. Quindi, se si perde informazioni nel primo layer queste saranno perse per tutti quelli successivi. Di conseguenza, è necessario evitare di esagerare e iniziare l'aggiunta di pochi dropout nel primo layer e in caso procedere con l'aggiunta di altri;

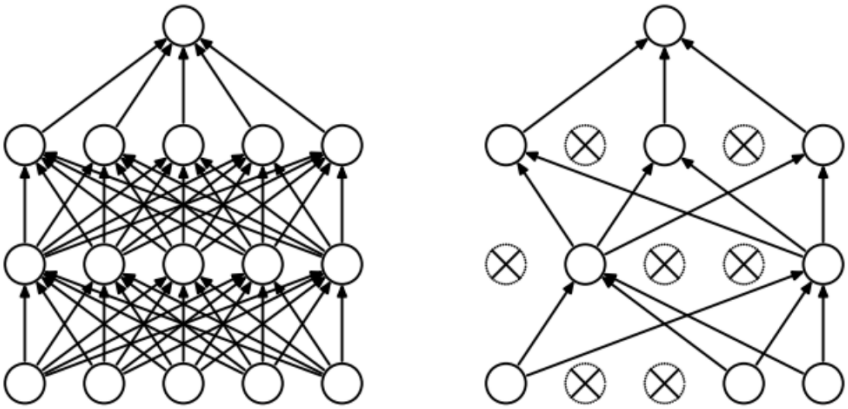

Figura 5.6: Le connessioni dopo aver applicato le regole di dropout

5. Come ultima opzione, si può ridurre la complessità della rete. Questa opzione però viene usata di rado, in quanto la sua necessità è abbastanza rara. Di solito le opzioni precedenti bastano.

5.6.2 Validazione

Per la validazione con il dataset iniziale è stato riscontrato l'overfitting subito dopo pochissime epoche e la precisione del modello era pessima.Questo problema è tuttavia stato risolto con il data augmentation. Per il data augmentation, sono state prese le immagini del primo dataset e sono state fatte le seguenti operazioni sulle immagini:

- ritagliate;
- presi i loghi e incollati nelle altre immagini;
- tolti i loghi nell'immagine;
- distorte le immagini;
- sfocati leggermente i loghi;
- messe in scala di grigi.

Una volta addestrato con il nuovo dataset, il processo di validazione procede come nella figura 5.7: al modello viene data un'immagine e lui ritorna l'immagine etichettata con gli oggetti che ha trovato.

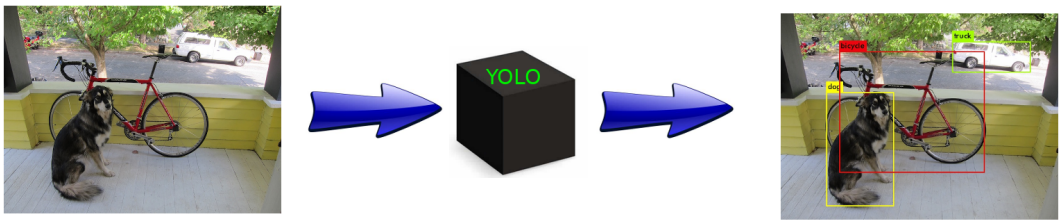

Figura 5.7: Esempio di immagine dopo la processazione di yolo

Per giunta, oltre all'immagine Yolo ci dà una risposta sullo stdout degli oggetti che ha trovato:

```
dog: 97.9221%
truck: 98.3342%
bicycle: 99.017%
```

Per il test di precisione e al fine di capire quando capita l'overfitting, sono state prese le seguenti immagini:

A 20 immagini con 1 logo;
B 5 immagini con 3 loghi;
C 1 immagine con 1 logo normale 1 logo in bianco e nero;
D 1 immagine con 1 logo in bianco e nero 4 tagliati a metà;
E 1 immagine con 2 loghi in bianco e nero;
F l'immagine con 7 loghi normali e 2 in bianco nero;
G l'immagine con 7 loghi normali e 2 in bianco nero oscurata del 15%;
H l'immagine con 7 loghi normali e 2 in bianco nero oscurata del 30%;
I l'immagine con 7 loghi normali e 2 in bianco nero oscurata del 45%;
J l'immagine con 7 loghi normali e 2 in bianco nero oscurata del 60%;
K l'immagine con 7 loghi normali e 2 in bianco nero oscurata del 75%;
L 249 immagini prese a caso senza nessun logo.

Come immagini, sono state prese anche quelle scattate durante le conferenze, oppure, foto di loghi trovati in strada. Questo venne fatto per vedere come si comporta Yolo, essendo questi casi più difficili per trovare gli oggetti.

Le seguenti immagini sono state utilizzate per misurate la precisione e così vedere come procedeva l'addestramento. Questo si può vedere nella tabella 5.1. La tabella mostra il livello dei loghi trovati, rispetto a quelli totali del/delle immagine/i. Si può inoltre vedere che la precisione sale, per poi diminuire quando il modello raggiunge l'overfitting. Nella scelta dell'epoca migliore per il modello è stata scelta quella evidenziata nella tabella.

Nelle seguenti figure 5.8 5.9 5.10 5.11 si possono vedere i grafici della tabella precedente, nei quali si può vedere che la precisione sale e poi diminuisce.

Tabella 5.1: Risultati della validazione con le immagini elencate

n	epoche	A	B	C	D	E	F	G	H	I	J	K	L
1	1000	0/20	0/15	0/1 0/1	0/2 0/4	0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0
2	10000	5/20	4/15	0/1 0/1	0/2 0/4	0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	2
3	20000	9/20	6/15	0/1 0/1	0/2 2/4	0/2	2/7 0/2	2/7 0/2	1/7 0/7	1/7 0/2	0/7 0/2	0/7 0/2	0
4	30000	6/20	7/15	1/1 0/1	1/2 2/4	1/2	2/7 0/2	2/7 0/2	2/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	0
5	40000	9/20	6/15	0/1 0/1	1/2 2/4	1/2	2/7 0/2	2/7 0/2	2/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	1
6	50000	9/20	7/15	1/1 0/1	1/2 2/4	1/2	3/7 0/2	2/7 0/2	2/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	1
7	60000	9/20	8/15	1/1 1/1	1/2 2/4	1/2	2/7 0/2	2/7 0/2	2/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	1
8	70000	7/20	8/15	1/1 1/1	1/2 1/4	1/2	2/7 1/2	2/7 1/2	2/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	4
9	80000	11/20	6/15	1/1 1/1	1/2 2/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 0/2	1/7 0/2	0/7 0/2	1
10	90000	11/20	7/15	1/1 0/1	1/2 2/4	1/2	2/7 1/2	2/7 0/2	2/7 0/2	2/7 0/2	1/7 0/2	0/7 0/2	1
11	100000	11/20	8/15	1/1 0/1	1/2 2/4	1/2	4/7 0/2	4/7 0/2	3/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	1
12	110000	11/20	8/15	1/1 0/1	0/2 2/4	1/2	4/7 0/2	4/7 0/2	4/7 0/2	4/7 0/2	1/7 0/2	0/7 0/2	2
13	120000	13/20	7/15	1/1 1/1	1/2 2/4	1/2	4/7 1/2	4/7 1/2	4/7 0/2	2/7 0/2	1/7 0/2	0/7 0/2	0
14	130000	12/20	7/15	1/1 1/1	0/2 2/4	1/2	4/7 1/2	4/7 1/2	4/7 1/2	3/7 0/2	1/7 0/2	0/7 0/2	1
15	140000	11/20	8/15	1/1 1/1	0/2 2/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 0/2	1/7 0/2	0/7 0/2	0
16	150000	9/20	7/15	1/1 1/1	0/2 2/4	1/2	3/7 1/2	3/7 1/2	2/7 0/2	2/7 0/2	0/7 0/2	0/7 0/2	1
17	160000	13/20	7/15	1/1 0/1	1/2 2/4	1/2	3/7 1/2	3/7 1/2	3/7 0/2	2/7 0/2	2/7 0/2	0/7 0/2	0
18	170000	12/20	8/15	1/1 1/1	1/2 2/4	2/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 1/2	1/7 1/2	0/7 0/2	4
19	180000	10/20	7/15	1/1 1/1	0/2 2/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 0/2	2/7 0/2	0/7 0/2	2
20	190000	11/20	7/15	1/1 1/1	0/2 1/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 0/2	2/7 0/2	0/7 0/2	1
21	200000	12/20	7/15	1/1 0/1	0/2 2/4	0/2	2/7 0/2	2/7 0/2	2/7 0/2	2/7 0/2	1/7 0/2	0/7 0/2	1
22	210000	12/20	7/15	1/1 1/1	1/2 2/4	1/2	3/7 1/2	2/7 1/2	2/7 1/2	2/7 1/2	1/7 0/2	0/7 0/2	2
23	220000	12/20	8/15	1/1 1/1	0/2 2/4	2/2	3/7 1/2	3/7 1/2	3/7 1/2	3/7 0/2	1/7 0/2	0/7 0/2	2
24	230000	11/20	8/15	1/1 1/1	0/2 2/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 0/2	0/7 0/2	0/7 0/2	1
25	240000	11/20	7/15	1/1 1/1	0/2 2/4	1/2	2/7 1/2	2/7 1/2	2/7 1/2	2/7 1/2	1/7 0/2	0/7 0/2	0
26	250000	12/20	6/15	1/1 0/1	0/2 1/4	1/2	2/7 1/2	2/7 0/2	1/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	1
27	250000	10/20	6/15	0/1 0/1	0/2 1/4	1/2	1/7 0/2	1/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0
28	250000	8/20	4/15	1/1 0/1	0/2 0/4	0/2	1/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0
29	260000	6/20	5/15	0/1 0/1	0/2 0/4	0/2	0/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0
30	260000	3/20	2/15	0/1 0/1	0/2 0/4	0/2	0/7 0/2	1/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0/7 0/2	0

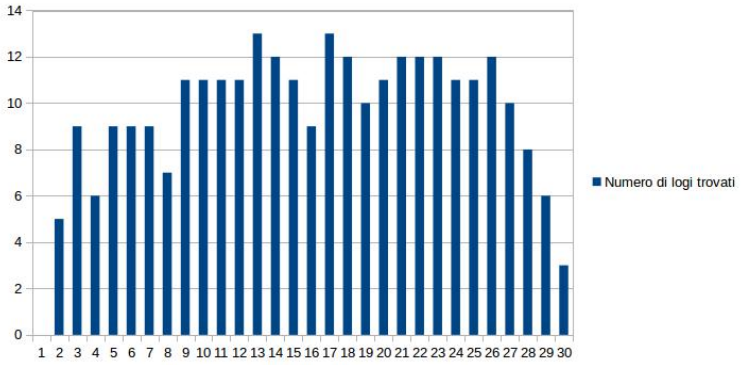

Figura 5.8: Precisione con il procedere dell'apprendimento per le 20 immagini con un singolo logo

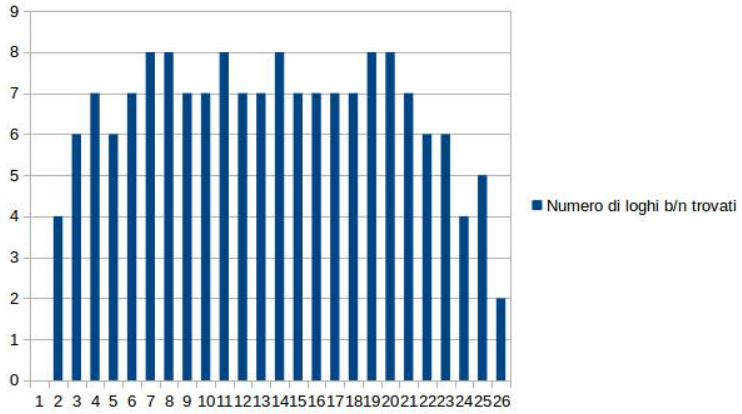

Figura 5.9: Precisione con il procedere del training per l'immagine in bianco/nero

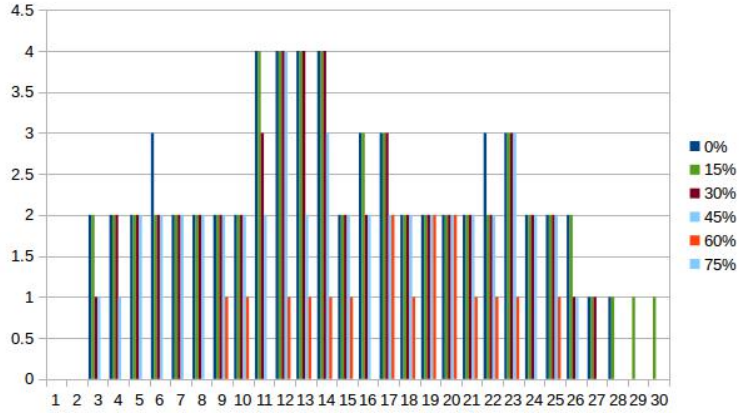

Figura 5.10: Precisione con il procedere del training per l'immagine con 7 loghi normali e 2 in bianco/nero, prendendo i loghi soltanto i loghi normali, con i vari oscuramenti

Figura 5.11: Precisione con il procedere del training per l'immagine con 7 loghi normali e 2 in bianco/nero, prendendo i loghi in bianco/nero con i vari oscuramenti

Conclusioni

Nella seguente tesi è stato spiegato il brand abuse e i vari tipi di abusi nella rete, tralasciando quelli tradizionali. È stata descritta l'evoluzione del significato di brand, di come questo non è più soltanto il logo ma qualcosa di più astratto.

Spiegata l'evoluzione, sono state introdotte le reti neurali convoluzionali, le quali sono la base dell'algoritmo di classificazione Darknet. Vi è stata un breve descrizione dell'algoritmo di detection Yolo con la motivazione della sua scelta e i vari test eseguiti sul progetto svolto.

I test fatti erano quelli di precisione della localizzazione dei loghi e quelli di detection, fatte per scegliere il livello di addestramento adeguato.

Il modello è stato creato utilizzando algoritmi di detection reputati attualmente come i migliori per la soluzione del problema, ma questi potrebbero cambiare con il tempo. In futuro, questi potrebbero essere reputati obsoleti, rendendo la scelta di altri più opportuna.

L'approccio presentato per la soluzione del progetto è attualmente utilizzato in produzione da un'azienda. Questa soluzione semiautomatica semplifica l'individuazione dei siti di phishing da parte degli addetti, ma potrebbe essere migliorata automatizzandola maggiormente.

Questo progetto, ha affrontato una delle problematiche del brand abuse, il quale è sempre più diffuso nel mondo digitale. Siccome ci sono sempre nuovi tipi di abuso, il modello descritto potrebbe essere utilizzato come base per la soluzione di altri problemi simili, come per esempio:

- Controllare se una pagina espone i loghi degli sponsor, oppure
- Controllare periodicamente se una pagina web mostra la pubblicità di un azienda, e con che frequenza,

Bibliografia

[1] From not working to neural networking. *The Economist*, 2016. 2

[2] Machines 'beat humans' for a growing number of tasks. *Financial Times*, 2016. 2

[3] Pieter Agten, Wouter Joosen, Frank Piessens, and Nick Nikiforakis. Seven months' worth of mistakes: A longitudinal study of typosquatting abuse. In *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS 2015)*. Internet Society, 2015. 1.3

[4] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018. 3.2

[5] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009. 3.1

[6] Colin Bates. What is a brand. *C./Davison. P./HUI. N*, 2006. 1.1

[7] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012. 3.2

[8] Steven R Borgman. The new federal cybersquatting laws. *Tex. Intell. Prop. LJ*, 8:265, 1999. 1.3

[9] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 2

[10] Aaker David. Brand equity. la gestione del valore della marca, 1997. 1.1.2, 1.1.2

[11] Pieter-Tjerk de Boer, Dirk Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 1 2005. Imported from research group DACS (ID number 277). 4.1.7

[12] J Deng, A Berg, S Satheesh, H Su, A Khosla, and L Fei-Fei. Ilsvrc-2012, 2012. *URL http://www. image-net. org/challenges/LSVRC*, 2012. 2

[13] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. 3.2

[14] Rachna Dhamija and J Doug Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88. ACM, 2005. 1.3

[15] Randima Fernando, Eric Haines, and Tim Sweeney. Gpu gems: programming techniques, tips and tricks for real-time graphics. *Dimensions*, 7(4):816, 2001. 5.5.1

[16] Dave Gershgorn. The inside story of how ai got good enough to dominate silicon valley. *Quartz*, 2016. 2

[17] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2

[18] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 2

[19] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. (document), 3.2, 4.1, 4.3, 4.4, 4.5, 4.6, 4.7

[20] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009. 3.1.1

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (document), 2.2, 4

[22] Paul Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bull Soc Vaudoise Sci Nat*, 37:241–272, 1901. 5.5.2

[23] Paul Jankowski. Brand promises are very valuable in the new heartland. *Forbes*, 2013. 1.1

[24] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014. 3.2

[25] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007. 3.1.1

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. (document), 2, 2.2, 4.1

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. 3.2

[28] LenNet. Convolutional neural networks — lenet. `http://deeplearning.net/tutorial/lenet.html`, 2013. [Online; in data 11-febbraio-2019]. 4.1.1

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 4.3.4

[30] Tony Lindeberg. Scale invariant feature transform. 2012. 2

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2, 4.3.4

[32] Silvia Martinelli. La disciplina dei nomi a dominio ei rimedi esperibili in caso di cybersquatting. 2015. 1.3

[33] Christopher Meyer and Andre Schwager. Customer experience. *Harvard business review*, 85(2):116–126, 2007. 1.1.1

[34] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013. 3.1.1

[35] Brian Murray. *Defending the brand: aggressive strategies for protecting your brand in the online arena*. Amacom, 2003. 1.3

[36] David Ogilvy and Isa Surci. *La pubblicità*. Illustrati Mondadori, 1990. 1.1

[37] Edgar Osuna, Robert Freund, and Federico Girosit. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*, pages 130–136. IEEE, 1997. 1

[38] Dawn Palmer. The power of keeping a promise. *MediaPost*, 2013. 1.1

[39] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. 3

[40] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, 45(3):380–385, 1996. 5.5.2

[41] Joseph Redmon. Darknet: Open source neural networks in c. *http://pjreddie.com/darknet*, 2016, 2013. 2, 4.1.3, 4.3, 5.3

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2, 2, 4

[43] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017. 2, 2

[44] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 2, 2, 5.3

[45] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2

[46] Christian Robert. Machine learning, a probabilistic perspective, 2014. 3.1

[47] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959. 3.1

[48] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010. 4.1.3

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. (document), 2.2

[50] Dave Steinkraus, Ian Buck, and PY Simard. Using gpus for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1115–1120. IEEE, 2005. 5.5.1

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. (document), 2.2

[52] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013. 2, 4.1

[53] E Tosi. Contraffazione di marchio e concorrenza sleale in internet: dal classico "domain grabbing" all'innovativo "key-word" marketing confusorio. *Rivista di diritto industriale*, 58(4-5):387–403, 2009. 1.3

[54] Treccani. perdita, funzione di — treccani - dizionario di economia e finanza, 2012. [Online; in data 11-febbraio-2019]. 4.2

[55] Brian Van Essen, Hyojin Kim, Roger Pearce, Kofi Boakye, and Barry Chen. Lbann: Livermore big artificial neural network hpc toolkit. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 5. ACM, 2015. 1

[56] Wikipedia. Convoluzione — wikipedia, l'enciclopedia libera, 2018. [Online; in data 11-febbraio-2019]. 4.1

[57] Wikipedia contributors. Flops — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=FLOPS&oldid=880439143`, 2019. [Online; accessed 11-February-2019]. 4.3.3

[58] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. (document), 2.2