UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Studi di meccanica classica con metodi di machine learning

Relatore

Prof.Marco Baiesi

Laureando

Salvatore Lampitelli

Anno Accademico 2016/2017

Indice

abstract

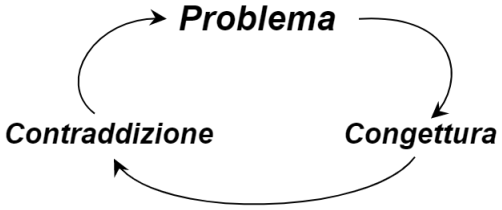
Tramite l'uso di metodi di machine learning si cercano metodi alterantivi per avere predizioni utili senza aver bisogno di ricorrere a teorie che modellizzano gli eventi. Nella tesi in particolare verrà analizzata la traiettoria di una pallina per predirne l'ultimo punto.

Introduzione

Hypothesis non fingo

Isaac Newton

L'ipotesi scientifica si basa sulla consapevolezza che essa può essere confutata[6]. Se non si vuole diventare un "tacchino induttivista"[1] bisogna ricordarsi che tutte le teorie scientifiche sono congetture che possono essere smentite in ogni momento e mai accettate completamente. Infatti, il modo di fare scienza moderno non è basato su un accumulo di *verità assolute*, ma sullo scarto delle *congetture falsificate*: il processo di analisi che si segue è:



Il "problema" è rappresentato da un fenomeno da spiegare, o ad un insieme di fenomeni a cui è stato attribuito un nesso di causa e effetto; la "congettura" è l'insieme di ipotesi e modelli logico/-fisico/matematici, che permette di spiegare e quantificare questa relazione di causa ed effetto. La "contraddizione" avviene quando, partendo dai risultati della congettura, si possono fare una serie di predizioni che però non vengono confermate dagli esperimenti. In questa continua osservazione ed analisi, ciò che produce il progresso scientifico, il "sapere", è contenuto in larga parte nella congettura. Possiamo dividere il sapere in due macro categorie: "sapere il come" e "sapere il perché". Il primo tipo di sapere si basa sul quantificare i nessi di causa ed effetto e ci dà il potere di utilizzare la natura a nostro piacimento: "Non mi importa sapere *perché* funziona un transistor, ma se conosco *come* funziona posso costruire un computer."
Sapere il perché, invece, è più delicato e difficile da quantificare. Il tipo principale di sapere a cui una congettura dovrebbe rispondere è cambiato nel corso dei secoli. All'inizio del pensiero umano "fare scienza" significava chiedersi esclusivamente il *perché*. La prima domanda che Newton si pose fu: "perché i corpi cadono?". Il genio di Newton fu capire che modellizare "come i corpi cadono" avrebbe portato luce anche sul perché. Da Newton in poi, non ci sono state tecniche per fare predizioni senza l'uso di congetture e ipotesi "ad hoc". Dopo l'avvento dei computer e dei "Big Data" tutto questo è cambiato. Adesso abbiamo uno strumento unico nella storia dell'umanità con cui poter fare predizioni basate esclusivamente sui dati: il Machine Learning. Questo strumento è capace di estrapolare e

[1]Fin dal primo giorno, questo tacchino osservò che, nell'allevamento in cui era stato portato, gli veniva dato il cibo alle 9 del mattino. Da buon induttivista non fu precipitoso nel trarre conclusioni dalle sue osservazioni e ne eseguì altre in una vasta gamma di circostanze: di mercoledì e di giovedì, nei giorni caldi e nei giorni freddi, sia che piovesse sia che splendesse il sole. Così arricchiva ogni giorno il suo elenco di una proposizione osservativa in condizioni più disparate. Finché la sua coscienza induttivista non fu soddisfatta ed elaborò un'inferenza induttiva come questa: "Mi danno sempre il cibo alle 9 del mattino". Questa concezione si rivelò incontestabilmente falsa alla vigilia di Natale, quando, invece di venir nutrito, fu sgozzato. (cit. Bertrand Russel)

analizzare le caratteristiche di un problema sulla base di dati grezzi, senza avere bisogno di ipotesi o di *intuizioni fenomenali*[2]. La velocità e la facilità con cui si possono analizzare vastissime quantità di dati aprirà nuove frontiere della conoscenza. Lo scopo di questa tesi è applicare questo nuovo metodo alla fisica classica, mostrando come sia possibile fare predizioni utili che si possano essere usate per creare, più che nuova conoscenza, nuova tecnologia.

[2]Enrico Fermi su come scoprì il neutrino lento

Visione e organizzazione

> Si può vedere in molti modi, come si può esser ciechi in molti modi.
>
> Paul Muad'Dib

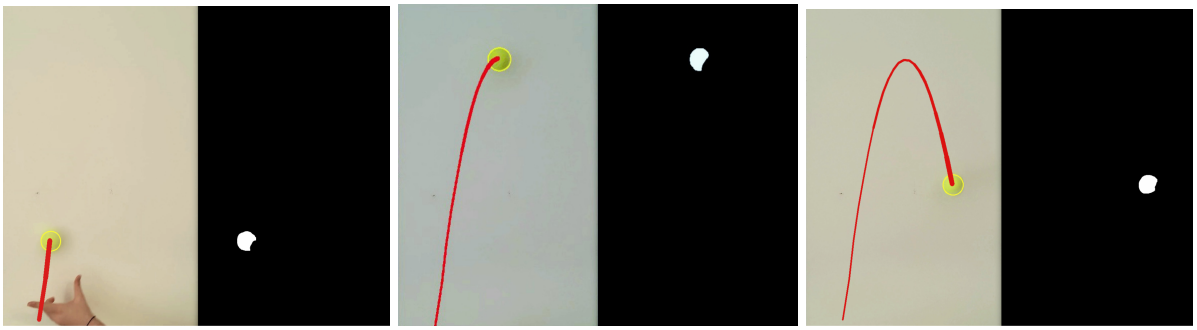
1.1 Computer Vision

La vista è uno dei sensi più sviluppati negli esseri umani. Gran parte della nostra esperienza quotidiana si basa sul *vedere* un evento per poi analizzare e comprendere cosa stiamo guardando. I computer, invece, "vedono" in maniera differente da noi. Ciò che negli uomini è un meccanismo istantaneo e ancestrale, per i calcolatori è un tenetativo di adattamento e automatizzazione di quella che è la nostra esperienza personale. Il processo di "visione" di un computer può essere schematizzato come segue. Per primo si ha l'evento reale. In questo evento si cerca una qualche correlazione tra i dati. Una volta scovata una caratteristica (una "feature") interessante dell'evento, la si estrapola e si cerca di creare un modello semplificato che le esalti. Nel particolare di questa tesi ci si è posto l'obiettivo di **predire l'ultimo punto del moto parabolico di una palla**. Quindi: dapprima si lancia una pallina (evento reale); in ogni lancio, la pallina compirà traiettorie simili (feature); tramite tecniche di computer vision, si creerà un modello che esalta solo la traiettoria del nostro oggetto, ignorando quindi le caratteristiche di contorno quali il colore o la grandezza della pallina. Come si vedrà, i metodi di analisi e di acquisizione dati sono così generali che è possibile, tramite qualche modifica, analizzare tutta la fisica classica.
Gli algoritmi che permettono di analizzare il movimento di oggetti sono molto recenti[7] e negli ultimi anni si sono compiuti progressi strepitosi. Il programma usato in questa tesi è scritto interamente in Python in meno di 100 linee di codice e permette di analizzare la traiettoria di diversi oggetti. Il programma può ricevere sia video in tempo reale da qualsiasi fonte che permetta streaming video, sia frammenti di video in formato mp4. Il programma apre il video e per ogni frame vengono applicati dei filtri che isolano un range di colore (scelto in anticipo). In figura 1.1 all'algoritmo è stato chiesto di analizzare il range di colore del giallo e quindi eliminerà ogni altro colore nel frame per potersi "concentrare" sulla pallina. Fatto ciò il programma analizza il contorno dell'oggetto grazie a delle librerie OpenCv che restituiscono la forma della pallina trovando il punto centrale. Ogni punto centrale di ogni frame viene salvato automaticamente in un file che sarà poi utilizzato dal machine learning per fare l'analisi.

C'è da tener presente che questo programma è stato realizzato per tenere traccia di un solo oggetto per frame; se nel video da analizzare fossero presenti colori simili a quello dichiarato dell'oggetto, potrebbero esserci problemi nel riconoscerlo. Vari miglioramenti possono essere implementati. Ad esempio, si potrebbero definire tanti range di colori diversi ed utilizzare una funzione non globale ma locale. In rete si trovano progetti già terminati su questa linea di pensiero[5].

1.2 Organizzazione dei dati

Nell'ambito di questo lavoro di tesi sono stati girati 50 video con risoluzione 512x1024p di circa 1 secondo ciascuno, eliminando le possibili fonti di disturbo: ad esempio si è fatto uso di un background



(a) (b) (c)

Figura 1.1: Output del programma per analizzare la traiettoria di una pallina

omogeneo e bianco, in cui il colore della pallina scelta potesse risaltare e si è cercato di fare tutti i video con una fonte di luce costante e omogenea. In principio ogni set di dati conteneva in media 24 punti(ogni punto corrisponde ad un frame) con un minimo di 20 punti per asse. Si sono tagliati i dati al minimo di 20 punti per asse, scegliendo come punto centrale quello più vicino allo zero. In figura **??** sono stati riportati i primi 9 set di dati, con una interpolazione parabolica e coefficiente R^2:

$$Err(y, \hat{y}, N) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{N}} \tag{1.1}$$

Err è la media delle varianze dalla parabola originale, minore è Err maggiore è l'accuratezza del fit. Per far si che l'algoritmo converga più in fretta, i dati sono stati riscalati in un intervallo $[a, b] = [-1, 1]$ seguendo la trasformazione:

$$x' = ((b - a) * (x - m))/(M - m) + a \tag{1.2}$$

in cui x rappresenta i dati originali, x' i dati riscalati.

L'interpolazione parabolica risulta ottima siccome sappiamo a posteriori che, nella teoria della fisica classica, il moto è rappresentato da un'equazione del tipo:

$$y(x) = x \tan\theta - \frac{g}{2v_0^2 \cos^2\theta} \cdot x^2 \tag{1.3}$$

in cui θ rappresenta l'angolo che forma il vettore velocità con l'asse x, g è l'accelerazione di gravità e v_0 è il modulo della velocità.

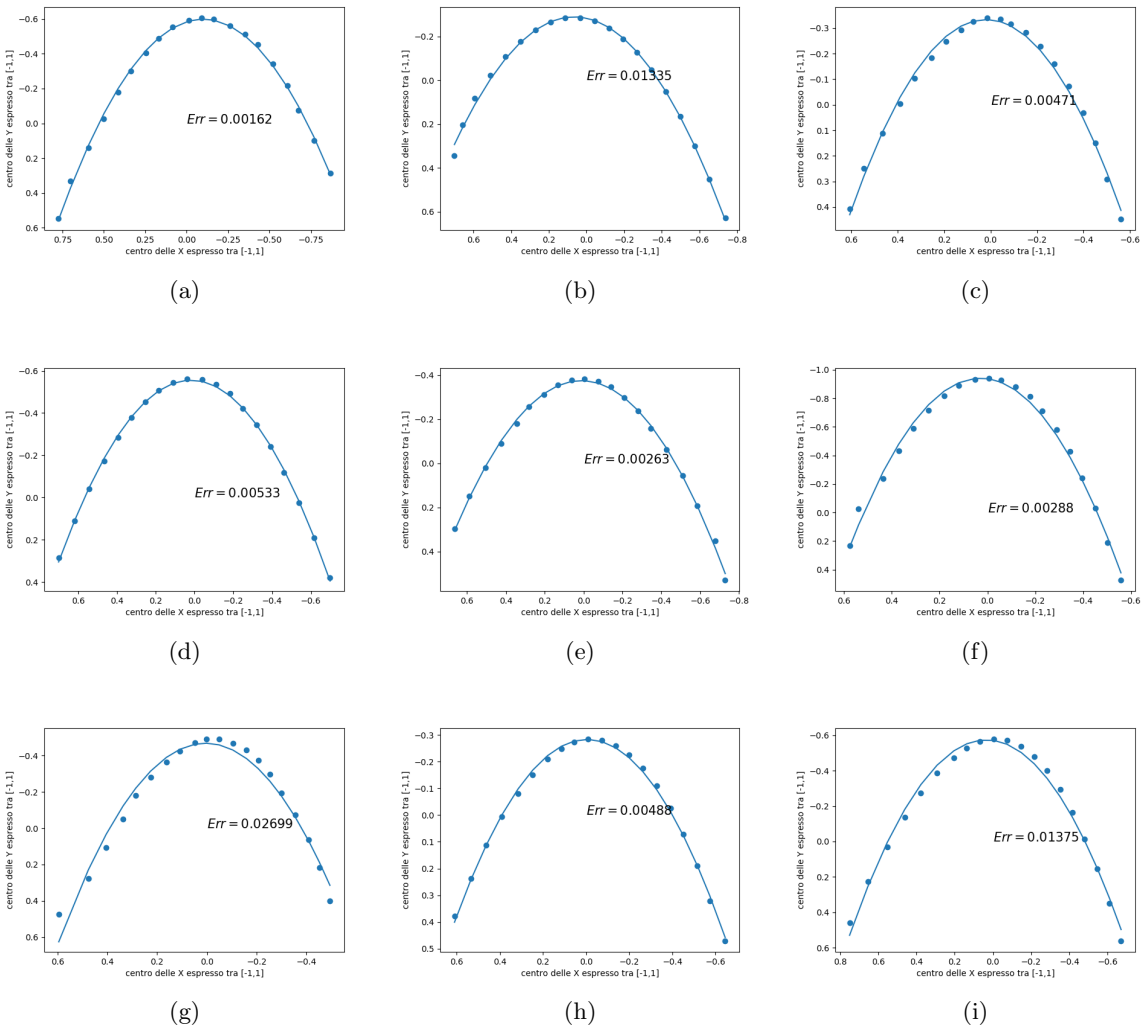


Figura 1.2: 9 set di dati con interpolazione $ax^2 + bx + c$ e coefficiente R^2, riscalare i dati li rende adimensionali

Machine Learning

> The electric things have their life too.
> Paltry as those lives are.
>
> Philip K. Dick, Do Androids Dream of Electric Sheep

2.1 Un po' di storia

Nell'esperienza comune si ha un' idea istintiva della traiettoria che compie un oggetto. Un giocatore di basket, ad esempio, potrebbe non saper calcolare analiticamente la traiettoria della palla che lancia, ma essere comunque molto preciso nel tiro. Il cervello umano riesce a fare una previsione qualitativa basata sui dati che ha raccolto in tutti i lanci a cui ha assistito. Una predizione, però, per essere utile nella creazione di nuova tecnologia deve essere quantitativa. Il machine learning simula questa "istintività" umana basata sull'esperienza per fare predizioni quantitative.
L'idea che un computer possa *imparare* risale già agli anni '50, quando Arthur Samuel[8] coniò il termine Machine Learning. Egli scrisse un programma per giocare a dama capace di *migliorare* dopo ogni partita, grazie a delle funzioni di scoring che assegnavano un punteggio ad ogni mossa. In quello stesso periodo, si facero anche i primi passi verso la modellizzazione del cervello umano, cercando di far luce su i suoi processi interni. Fu naturale per i programmatori dell'epoca, cercare ispirazione proprio in questi modelli del cervello umano, e nel '57 Frank Rosenblatt[3] combinò le teorie di Donald Hebb[2] sul funzionamento dei neuroni con l'algoritmo di Samuel, creando il primo neuro-computer: "Mark I Perceptron". Pochi anni dopo David Hubel e Torsten Wiesel descrissero una teoria[3] sul funzionamento delle *simple cell* e *complex cell* dei neuroni della corteccia visiva, responsabili del riconoscimento di *pattern* e degli oggetti. Le *simple cell* rispondono a bordi e griglie con particolari orientazioni, le *complex cell* rispondono sempre a bordi e griglie orientate, ma hanno un certo grado di invarianza spaziale e hanno un grande campo ricettivo. Questa invarianza è raggiunta grazie alla somma pesata di varie simple cell in un processo che usa la somma di semplici stimoli per valutare pattern più complessi. Ispirandosi a questa idea il Dr. Kunihiko Fukushima propose il modello Neocognitron[1]. L'idea principale fu di riuscire a modelizzare il comportamento simple-to-complex della corteccia visiva. Già era presente l'idea di base di una Convolutional Neural Network(C.N.N.), ispirarsi alla corteccia visiva umana per creare algoritmi che riconoscessero pattern. Solo negli anni '90 si ebbe il primo paper[4] moderno sulle Convolutional Neural Network. Il paper dimostra con successo che è possibile usare una CNN che aggrega caratteristiche semplici in caratteristiche progressivamente più complicate per il riconoscimento dei caratteri scritti a mano[4]. Per molto tempo le CNN sono state considerate "black box" in cui non era possibile capire cosa stesse succedendo al loro interno. Recentemente si sono trovati dei metodi per avere un'idea sul loro funzionamento. Ad esempio, *DeepDream* di Google[5](Figura 2.3). DeepDream è una CNN che funziona "al contrario". Prima l'algoritmo viene addestrato a riconoscere determinate categorie (nel caso preso in considerazione in figura, il network è addestrato a riconoscere cani); successivamente una immagine qualsiasi viene passata al network, che vede il pattern su cui è stato addestrato. Nel programma per le predizioni

[3] Manuale pdf di come operare il perceptron, declassificato nel 1960:https://apps.dtic.mil/dtic/tr/fulltext/u2/236965.pdf
[4] usando keras e python è possibile, con poche linee di codice, ricreare il paper con quasi accuratezza maggiore
[5] per un video del funzionamento live: Deep Visualization toolbox

scritto per questo lavoro di tesi, usiamo una CNN la cui struttura verrà spiegata in dettaglio nel paragrafo 2.2.

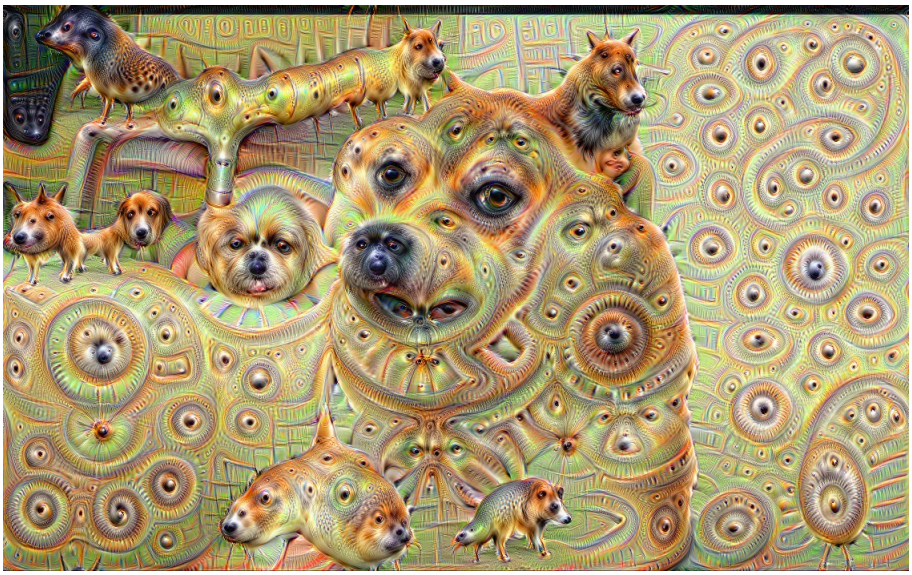


Figura 2.3: L'immagine è costruita sulla base del training che ha avuto la CNN e su quello che si *aspetta* di vedere dall'immagine. Dall'immagine è chiaro perché il programma è chiamato DeepDream. Basta andare in http://deepdream.psychic-vr-lab.com/deepdream/ e caricare qualsiasi foto per vedere l'effetto

2.2 Struttura CNN

Una Convolutional Neural Network è costituita da 2 parti principali: Feature extraction e Classification (figura 2.3).

La parte di **Feature Extraction** usa strumenti di convoluzione per estrapolare le features importanti secondo il principio di simple-to-complex. La parte di **Classification** usa un classico MLP(Multi-Layer Perceptron) che riceve i dati dalla prima parte e riesce classificarli in base alle features.
L'algoritmo è diviso ulteriormente in layer, ognuno di questi lavora in modo autonomo con una funzione prestabilita. Anche se esiste una ricetta generale per costruire una CNN i layer funzionano in qualsiasi ordine e a volte costruire l'algoritmo perfetto per un determinato problema dipende da come vengono impilati i vari layer.

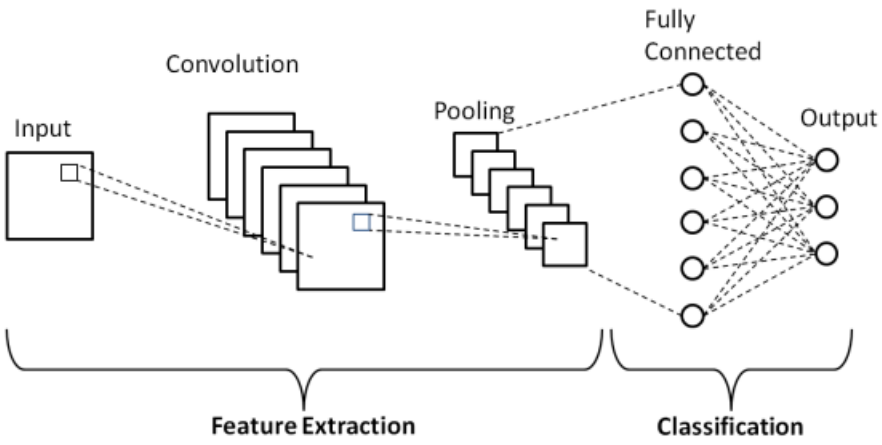


Figura 2.4: Struttura generale per la classificazione di immagini, source url: Binary image classifier

2.2.1 Input

Le CNN godono di fama per i recentissimi avanzamenti nel campo del riconoscimento di oggetti. La loro struttura è però completamente generalizzabile ad ogni tipo di input. Nella maggior parte dei casi, l'input di una CNN è una immagine, ovvero una matrice "height$x$width" in cui ogni punto è un pixel che può essere rosso, blu o verde (o solo bianco o nero). L'input della CNN sarà allora una matrice tridimensionale del tipo Height$x$Width$x$3 (oppure Height$x$Width$x$1). Il tipo di input specifica anche il tipo di convoluzione che si andrà a eseguire sui dati.

2.2.2 Convolution

Con la convoluzione si riescono a filtrare i dati, distillandone le caratteristiche intrinseche. Si crea una maschera NxN, la cui dimensione N deve essere compatibile con il tipo di input. La si fa scorrere sull'input stesso (figura 2.4). Gli iperparametri da considerare quando si utilizza il layer sono:

- numero di filtri: regolano la profondità dell'output;
- stride (passo): sceglie il numero di passi ad ogni iterazione della maschera;
- zero-padding: se le dimensioni dei filtri non si adattano all'input, questo viene riempito da zeri per avere un output della stessa dimensione dell'input;
- activation function: stabilisce l'output di un nodo dato un input o un set di input.

Nello zero-padding posso poi avere tre possibilità di scelta:

- valid padding: dove non si aggiungono zero e la dimensione potrebbe cambiare se si sceglie una maschera non adatta
- same padding; per forzare l'output ad avere la stessa dimensione dell'input;
- full padding: l'ouput ha dimensioni maggiori dell'input.

I vari tipi di activation function posso essere:

- linear: $y(x) = ax + b$
- ReLu: $\max\{0, y(x) = ax + b\}$
- exponential: $R(z) = z$ se $z > 0$ Oppure $(e^z – 1)$ se $z <= 0$
- softmax: $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
- ... E molte altre, per una lista completa: Activation Function[10].

I vari tipi di activation function regolano la risposta del network agli input e di solito hanno una risposta simile a quella dei neuroni, che possono essere "accesi" o "spenti".

2.2.3 Pooling

Il layer di Pooling[6], anche chiamato *downsampling*, è primariamente usato per abbassare il costo computazionale della rete neurale riducendo il numero di parametri dell'input, abbassandone la dimensione. Similmente al layer convoluzionale, l'operazione di raggruppamento spazza un filtro sull'intero input, con la differenza è che questo filtro non ha pesi. Il kernel applica una funzione di aggregazione ai valori all'interno del campo ricettivo, popolando l'array di output. Esistono due tipi principali di pooling:

- max pooling, che seleziona il pixel con il valore maggiore e lo scrive nell'output;
- average Pooling, che fa una media su tutti i pixel della maschera e li scrive nell'output

Il pooling oltre a ridimensionare l'output ha una funzione di pulizia di feature non necessarie e riesce a evidenziare la caratteristica più forte dei dati.

[6]Trad. *aggregazione*

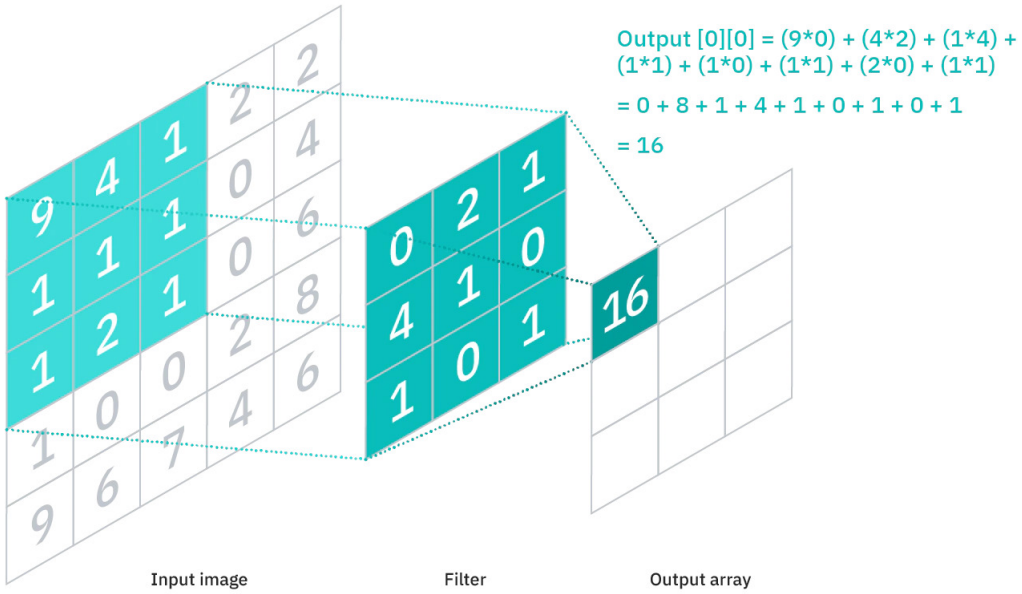


Figura 2.5: source url: IBM Convolutional Neural Network

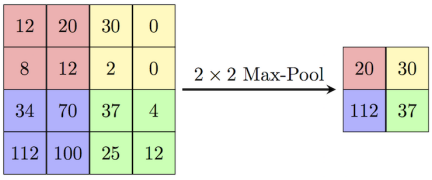


Figura 2.6: Esempio di Max Pooling

2.2.4 Dense

Il layer Dense è uno strato di rete neurale che è connesso in modo profondo: ogni neurone nello strato denso riceve input da tutti i neuroni del suo strato precedente. Il Dense layer è lo strato più comunemente usato nei modelli di machine learning. Il layer implementa l'operazione: output = activation (dot (input, kernel) + bias). I valori utilizzati nella matrice sono in realtà parametri che possono essere addestrati e aggiornati con l'aiuto della backpropagation. L'output generato dallo strato denso è un vettore dimensionale "m". Pertanto, lo strato denso viene fondamentalmente utilizzato per modificare le dimensioni del vettore. I livelli densi applicano anche operazioni come rotazione, ridimensionamento, traslazione sul vettore.

2.2.5 Dropout

Il layer Dropout imposta casualmente alcuni dati di input su 0 con una frequenza che aumenta ad ogni passaggio durante il tempo di allenamento. Questo livello aiuta a prevenire l'overfitting dato che se il programma viene allenato per troppo tempo potrebbe produrre troppi parametri che lo rendono poco generalizzabile. Gli ingressi non impostati su 0 vengono scalati di 1 / (1 - rate) in modo che la somma di tutti gli ingressi rimanga invariata.

2.2.6 Flatten

Il flatten layer converte i dati in un array unidimensionale per poi passarli nel layer finale. Si appiattisce l'output dei layer convoluzionali per creare un unico vettore di caratteristiche. Ed è collegato al modello di classificazione finale, chiamato *fully connected layer*. In altre parole, mettiamo tutti i dati dei pixel in una riga e creiamo i collegamenti con il livello finale.

2.2.7 Fully Connected

Un Fully Connected (FC) layer è una funzione $\mathbb{R}^m \to \mathbb{R}^n$. Ogni dimensione di output dipende da ciascuna dimensione di input. Questo layer svolge l'attività di classificazione in base alle caratteristiche estratte attraverso i layer precedenti e ai loro diversi filtri. Negli usi di classificazione di solito si usa come funzione di attivazione il softmax che produce un output di probabilità sulle diverse classi. Come vedremo, per avere predizioni di output continui converrà usare Linear o ReLu. In figura 2.7 un esempio di come Flatten e Fully connected lavorano insieme.

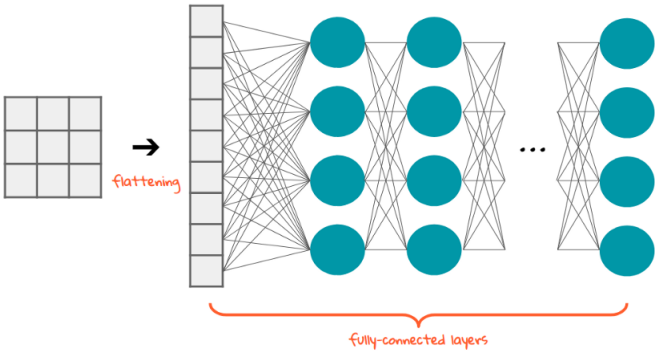


Figura 2.7: L'input di una F.C. è un array unidimensionale e quindi si deve appiattire l'immagine

2.2.8 Epochs e Batch

Altri iperparametri per il corretto funzionamento dell'algoritmo sono il "sample Batch"[7] e l'Epochs[8]. La dimensione del Batch definisce il numero di campioni su cui lavorare prima di aggiornare i parametri del modello interno. Il programma compie un ciclo su una porzione scelta in anticipo e, alla fine, compie delle previsioni che vengono confrontate con le variabili di output previste e viene calcolato un errore. Da questo errore, l'algoritmo si aggiorna e migliora il modello. Il numero di epoche definisce quante di volte l'algoritmo funzionerà attraverso l'intero set di dati di addestramento. Sono due parametri molto difficili da settare e da questi spesso dipende se l'algoritmo riuscirà a convergere ad una soluzione oppure no.

2.3 Perché le reti convoluzionali?

2.3.1 Teoria delle convoluzioni

La convoluzione è un operatore matematico che prende due funzioni, f e g, producendone una terza, $f * g$ descritta da

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{2.4}$$

Per iniziare ad avere una qualche intuizione fisica sul perché la convoluzione sia importante per il processing di immagini, si possono studiare le onde. Se si lascia cadere un sasso in un lago, questo creerà delle increspature sulla superfice che chiamiamo "onde". Possiamo dire che c'è una correlazione tra $Sasso \to Onda$. Esiste cioè una certa funzione $w : Onda = w[Sasso]$. Si può osservare anche che l'intensità dell'onda dipende dalla grandezza del sasso, e che c'è una qualche relazione lineare per cui $c \cdot Sasso \to c \cdot Onda$. Si può quindi studiare il caso speciale del sasso con grandezza unitaria; con un riscalamento si possono poi studiare gli altri casi. Se assumiamo che il sasso unitario sia piccolo rispetto l'onda generata e che la perturbazione causata dal sasso avviene in tempi piccoli rispetto a quella di propagazione dell'onda, possiamo definire il sasso come un impulso δ che crea una risposta d'impulso h, cioè $\delta \to h$. Si può notare anche che traslando la posizione in cui si lancia il sasso trasla anche l'onda associata, cioè $\delta(r - r_0) \to h(r - r_0)$ e che se si lanciano due o più sassi

[7] trad. porzione/frazione
[8] trad. era/età

Figura 2.8: Stella δ in (u,v) del cielo in 2D e rispettivo segnale sfocato h in (u,v) sull'immagine 2D
source: Convolution[9]

la risposta d'impulso è la somma delle varie risposte, cioè $\sum_{n=1}^{N} \delta_n(r - r_0^n) \rightarrow \sum_{n=1}^{N} h_n(r - r_0^n)$. Da qui si può fare una predizione chiedendosi "Cosa succede se lancio N sassi di diversa grandezza nel laghetto?". Avrò $impatto = \sum_{n=1}^{N} Sasso_n$, cioè $impatto = \sum_{n=1}^{N} c_n \cdot \delta(r - r_0^n)$. Si ha allora $Onda = w[\sum_{n=1}^{N} c_n \cdot \delta(r - r_0^n)] = \sum_{n=1}^{N} c_n \cdot w[\delta(r - r_0^n)]$ e sapendo che $w[\delta(r - r_0)] = h(r - r_0)$. Si può quindi capire una situazione abbastanza complessa, come la forma di un'onda dopo tanti piccoli impulsi, conoscendo solo la forma funzionale dell'onda unitaria. Volendo fare un altro esempio: l'obiettivo di una fotocamera[9] è quello di catturare la luce originale e depositarla sulle lenti. Se si chiama *Source* la fonte di luce e *Foto* l'immagine finale si può scrivere una relazione simile a quelle delle onde: $S \xrightarrow{C} F$, cioè $F = C[S]$. Cavalcando l'onda[10] dell'analogia di prima se si volesse fotografare un cielo stellato, l'input della fotocamera sarebbe $input = \sum_{i=1}^{N} stelle_i = \sum_{i=1}^{N} m_i(\delta(x - u, y - v)$, poiché una stella molto lontana può essere considerata un punto in un piano (x,y) avente una posizione (u,v) con una intesità m_i. Se la camera non è in focus, come nel caso delle onde, ad un impulso corrisponde un segnale(figura 2.8), cioè quello che vedrò nell'immagine sarà $m_i \cdot h_i(x-u, y-v) = m_i \cdot C[\delta_i(x-u, y-v)$. Nel caso reale m_i sarà in realtà una funzione di (u,v), poiché la volta celeste è coperta di stella più o meno luminose. E siccome nel caso discreto si ha $Foto = \sum m_i h_i(x-u, y-v)$ il passaggio al continuo è dato da $F = \int_{\mathbb{R}} m(u,v)h(x-u, y-v)dudv$. Se si considera allora una foto come una somma di impulsi pesati si ha subito che $foto = m(x,y) * h(x,y)$.

2.3.2 Filtri

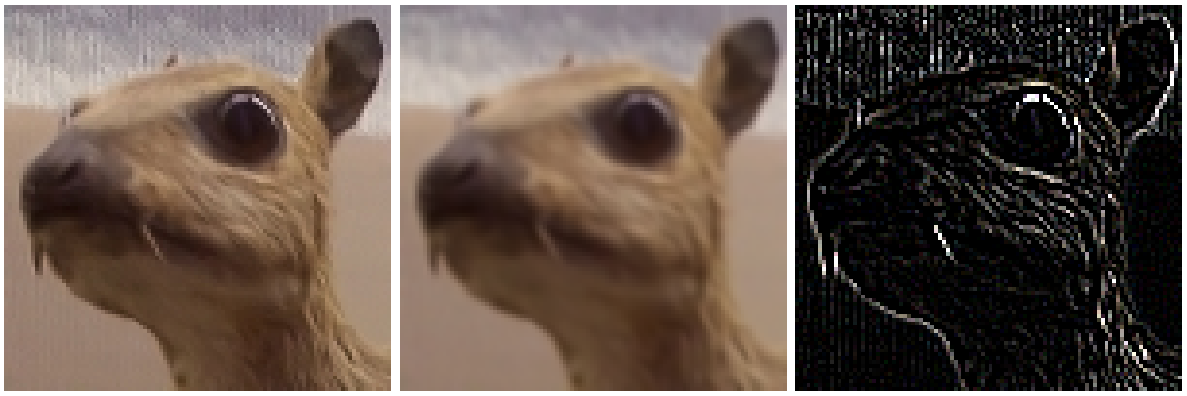
Allora è chiaro cosa rappresenta una matrice di convoluzione nel caso del machine learning. Matematicamente ho

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy) \tag{2.5}$$

con $g(x,y)$ l'immagine processata, $f(x,y)$ l'immagine originale e ω il filtro kernel che passa sull'immagine. Con questa tecnica si possono avere semplici filtri gaussiani che sfocano l'immagine o filtri che isolano i bordi.

I filtri in figura 2.9 sono matrici 3x3. Ad esempio:
il gaussian blur(b) è dato da: $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.
Mentre l'edge detection è dato da: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$.

[9] gioco di parole non voluto
[10] questa volta è voluto



(a) (b) (c)

Figura 2.9: a) immagine originale b) gaussian blur c)edge detection filter
source: Kernel[11]

La maggior parte dei filtri per immagine (anche quelli di instagram) è applicato con questo metodo. I filtri che però sono utilizzati dal machine learning sono quelli che riescono ad estrapolare informazioni dall'immagine. L'esempio dell'edge detection è che il filtro ignora tutti gli input che non sono bordi. La procedura è la stessa, cambia solo la forma dell'input che vogliamo analizzare con il passaggio dei filtri. L'algoritmo troverà sempre modi per estrapolare delle caratteristiche interessanti per fare predizioni significative.

Predizioni

> Deep in the human unconscious is a pervasive need for a logical universe that makes sense. But the real universe is always one step beyond logic.
>
> Frank Herbert, Dune Messiah

3.1 Predizioni classiche

3.1.1 Teoria delle predizioni classiche

La teoria classica del moto dei corpi predice con grande accuratezza[11] una quantità incredibile di fenomeni. La meccanica classica ha però un problema che hanno tutte le teorie, un problema che può essere evidenziato dalla barzelletta della mucca sferica [12].La teoria è solo una approssimazione molto scarna della realtà. Il rischio di *dimenticare* qualche parametro importante per avere predizioni accurate, come l'attrito o l'aria, è molto alto. Nel caso specifico del lancio di una pallina, la predizione data dalla fisica classica è ottima e solo se si vogliono predizione eccezionalemente accurate su lunghe distante c'è bisogno di considerare altri parametri. Avendo qualche punto iniziale del centro di massa dell'oggetto, si può interpolare una parabola e computare la posizione delle y in modo preciso e continuo. Basta sapere che l'equazione è parabolica e interpolando i primi dati è possibile avere con grande precisione il dato successivo. La considerazione che la traiettoria sia una parabola si basa sulle assunzioni della meccanica classica e perfino l'affermazione "tutte le traiettorie sono parabole" è evidentemente falsa. La meccanica classica si basa su assunzioni che sono o incomplete (come $F = ma$) o false (come la fissità del tempo e dello spazio) e va usata con la massima cautela quando cerchiamo la massima precisione possibile.
Va chiesto se esiste un metodo per avere delle predizioni che non richieda l'uso di nessuna teoria e che si basi solo su evidenze sperimentali reali e innegabili. Questo metodo va cercato nel machine learning e nelle sue future applicazioni.

3.2 Predizioni con machine learning

3.2.1 Teoria delle predizioni con machine learning

Grazie ai metodi del machine learning mostrati in questa tesi è possibile fare predizioni su una gran quantità di fenomeni. In questo lavoro si vuole prevedere la posizione finale di una pallina lanciata in aria senza usare conoscenze pregresse date dalla teoria della meccanica classica. Una predizione di questo genere si basa solamente sui dati che forniamo all'algoritmo. Essendo la rete convoluzione definita supervised learning tutto quello che dobbiamo fare per avere una predizione è raccogliere dei dati: la precisione e il modello emergeranno in modo autonomo.

[11] in relazione alla scala di grandezza umana

3.2.2 Set up dati

Per fare questa predizione si è usata una rete convoluzionale con le caratteristiche mostrate in figura 3.10. Conta solamente 200 linee di codice ed è in grado di processare qualsiasi gruppo di dati in successione temporale, riuscendone a capire la struttura.

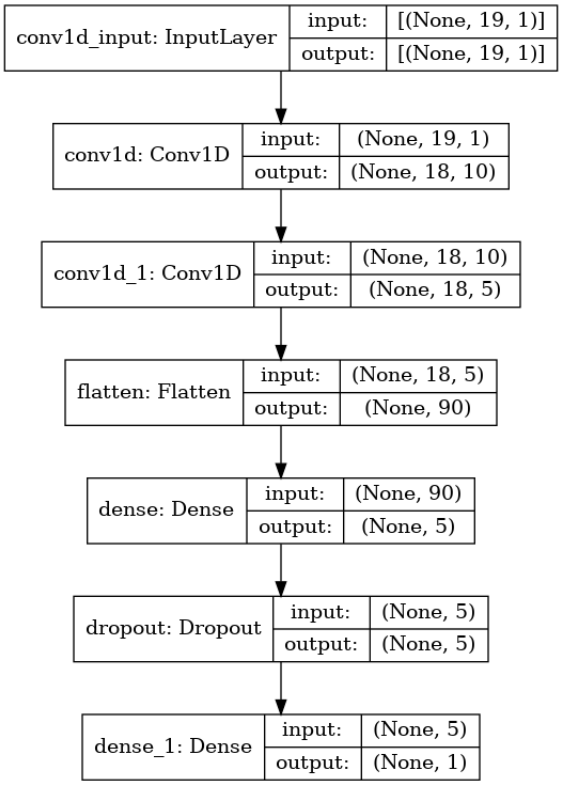
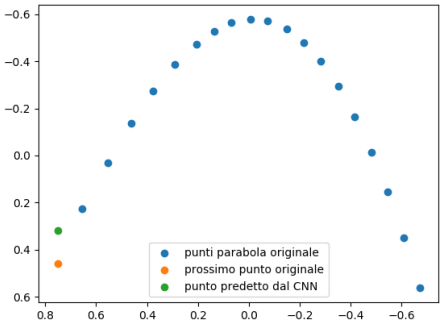


Figura 3.10: Modello usato per la predizione

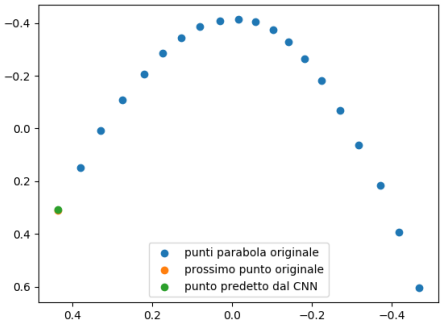
Il programma inizialmente riceve un array tridimensionale che ha come lunghezza 50x2x20. Sono stati registrati 50 lanci,con un due che rappresenta se il dato è una x o una y e con 20 coppie totali di x e y. In pratica si ha che la scrittura DATI[3][0][12] rappresenta il dodicesimo punto delle x del terzo lancio. Da questo set di dati si tagliano le x e si creano due set di dati separati. Uno lungo 50x19 con i 19 punti delle y e un altro lungo 50x1 riempito con gli ultimi punti delle y dei 50 lanci. A questo punto si dividono ancora i dati per un totale di 4 set:

- Un set con dimensione 45x19 chiamato "training features"
- un set con dimensione 45x1 chiamato "training labels"
- Un set con dimensione 5x19 chiamato "test features"
- Un set con dimensione 5x1 chiamato "test labels"

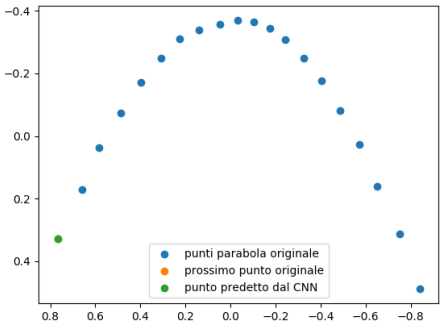
Non importa l'ordine con cui vengono riempiti i dati ma si deve fare molta attenzione a riempire con ordine uguale i set di training e i set di test. Cioè è fondamentale che se il terzo posto del training features è riempito con il quinto lancio anche il terzo posto del training labels deve essere l'ultima y del quinto lancio. I nomi rappresentano molto bene il ruolo all'interno del programma. Il set di training viene usato per insegnare le correlazioni al programma. Su questo set viene fatto scorrere il kernel che riuscirà a estrapolare le informazioni per la predizioni. In questo caso nel network convoluzione ci sono due convoluzioni con filtri di dimension 1x2. Quando la fase di training termina il modello ha imparato a riconoscere i dati e se è stato addestrato in maniera corretta darà predizioni corrette. Il secondo set chiamato test serve appunto a testare quanto queste predizioni sono corrette. Si fa partire il programma dandogli solo le 19 y del lancio e verrà restituito un solo dato y_{20} che sarà la predizione del modello su dove deve trovarsi il prossimo punto. Di seguito in Figura 3.11 sono riportate le predizioni dell'algoritmo.



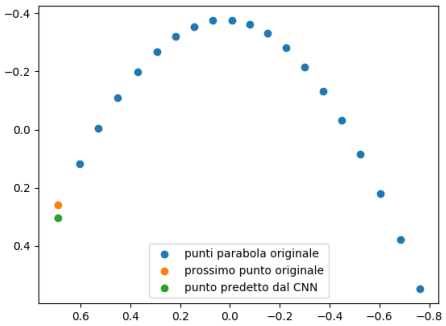
(a)



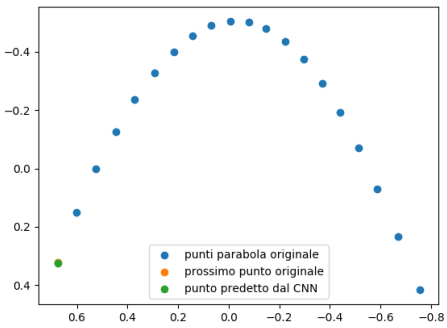
(b)



(c)



(d)



(e)

Figura 3.11: In figura le 5 predizioni del set di test con il punto originale registrato

Conclusioni

La distanza media dal punto originale al punto predetto è 0.10 ± 0.06. Ricordando che l'errore sul fit di figura 1.2 si aggira sui 0.01-0.001 potrebbe sembrare una misura imprecisa. La differenza fondamentale tra i due approcci è che i fit di figura 1.2 si basano sulla ipotesi che i dati seguano delle parabole. Se noi non sapessimo la fisica sottostante(e se fosse un problema più complesso) saremmo completamete in imbarazzo sul tipo di fit che dovremmo usare. Il metodo del machine learning risulta quello più diretto e pulito e non usa nessuna ipotesi che potrebbe essere falsa.
Per la precisione della predizione bisogna considerare che essa dipende fortemente dalla precisione dei dati raccolti e quindi sarebbe da ripetere tutto l'esperimento cercando di avere una precisione maggiore. Gli iperparametri del modello, inoltre, sono stati scelti a posteriori ad hoc per migliorare le predizioni e bisognerebbe testarli con misure quantitative, magari associando una ulteriore CNN per trovare la migliore configurazione possibile. Sicuramente un numero maggiore di dati con cui addestrare la CNN le predizioni avrebbe reso le predizioni ancora più accurate e generali. Possibili migliorie sono attuabili e le performance del programma potrebbero anche permettere di fare predizioni in tempo reale. Nonostante questo il modello proposto è riuscito a fare predizioni compatibili con i dati che si sono raccolti.

Bibliografia

[1] Kunihiko Fukushima. ≪Neocognitron: A hierarchical neural network capable of visual pattern recognition≫. In: *Neural Networks* 1.2 (1988), pp. 119–130. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90014-7. URL: https://www.sciencedirect.com/science/article/pii/0893608088900147.

[2] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, 2008. ISBN: 978-0805843002.

[3] D. H. Hubel e T. N. Wiesel. ≪Receptive fields, binocular interaction and functional architecture in the cat's visual cortex≫. In: *The Journal of Physiology* 160.1 (1962), pp. 106–154. DOI: https://doi.org/10.1113/jphysiol.1962.sp006837. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1962.sp006837. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1962.sp006837.

[4] Yann Lecun et al. ≪Gradient-Based Learning Applied to Document Recognition≫. In: *Proceedings of the IEEE* 86 (dic. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[5] Nathaniel-Guy. *Ball-Tracking*. 2016. URL: https://github.com/NattyBumppo/Ball-Tracking.

[6] Karl M. Popper. *la logica della scoperta scientifica*. Collana Piccola Biblioteca N.s. Einaudi, 2010. ISBN: 978-88-06-20392-4.

[7] Adrian Rosebrock. *Ball Tracking with OpenCV*. 2015. URL: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/.

[8] A. L. Samuel. ≪Some Studies in Machine Learning Using the Game of Checkers≫. In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[9] Steve Trettel. *Convolution*. URL: https://sites.google.com/site/butwhymath/m/convolution.

[10] Wikipedia. *Activation Function*. URL: https://en.wikipedia.org/wiki/Activation_function.

[11] Wikipedia. *Kernel*. URL: https://en.wikipedia.org/wiki/Kernel_(image_processing).

[12] Wikipedia. *Mucca sferica*. URL: https://it.wikipedia.org/wiki/Mucca_sferica.