Segnali Stradali

Composizione Gruppo	Matricola	Email
Marco Longo	05121-05945	m.longo36@studenti.unisa.it
Antonio Lamberti	05121-09103	a.lamberti72@studenti.unisa.it
Riccardo Polidoro	05121-07801	r.polidoro1@studenti.unisa.it

Link Github	https://github.com/marcolongo00/Segnali-Stradali

Sommario

1. Introduzione: scopo del progetto

Negli ultimi anni, il numero di veicoli sulle strade è aumentato esponenzialmente, e con questo anche il numero di incidenti dovuti alla distrazione da parte dei conducenti.

La tecnologia negli anni ha fatto passi da gigante portando alla costruzione di strumenti in grado di migliorare la qualità della vita.

Parte di questa tecnologia è l'intelligenza artificiale, con la quale è possibile costruire strumenti che possano riconoscere i segnali in modo da facilitare la guida e renderla più sicura.

Con il nostro progetto intendiamo implementare un classificatore in grado di riconoscere ben 43 diverse categorie di segnali.

2. Descrizione dell'agente

2.1 Obiettivi

Lo scopo del progetto è quello di realizzare un modello di deep learning che sia in grado di riconoscere e classificare i diversi segnali stradali, ed eventualmente segnalare la presenza di tali segnali all'autista del veicolo.

2.2 Specifica PEAS

PEAS	
Performance	La misura di performance dell'agente corrisponde al numero di segnali stradali correttamente classificati.
Environment	L'ambiente in cui opera l'agente è il dataset che prende in input per l'apprendimento e la collezione di immagini creata al termine della sua applicazione.
Actuators	Gli attutatori dell'agente consistono nell'aggiornamento del bias e neuroni e nel classificatore dell'immagine.
Sensors	I sensori dell'agente corrispondono allo stato corrente del modello e all'interfaccia per il caricamento delle immagini.

2.3 Analisi del problema

Il problema poteva essere affrontato implementando un classificatore tramite uno degli algoritmi visti a lezione, come ad esempio Naive Bayes o Decision Tree. Tuttavia, abbiamo deciso di utilizzare un algoritmo di deep learning che utilizza una rete convoluzionale in quanto più adatta alla classificazione di immagini per il nostro problema.
Le CNN sono progettate per riconoscere pattern visivi in modo diretto senza molto preprocessing. Dalle immagini vengono estratte le features usate per calcolare i match tra i punti caratteristici in fase di apprendimento. Si cercano di individuare dei pattern, come ad esempio curve, angoli, circonferenze o quadrati.

3. Dataset

Per il dataset necessario per la creazione del modello di deep learning le possibilità erano due:

1. **Creare** un dataset personale, dove venivano effettuate una serie di fotografie ai diversi segnali stradali
2. **Cercare** sul web un dataset già formato, andandolo ad analizzare e a selezionare le immagini che ci interessavano.

La prima soluzione richiedeva molto tempo per la raccolta delle immagini; inoltre, le foto potevano avere problemi di sfocatura o scarsa visibilità.

Per questo abbiamo deciso di prendere in considerazione la **seconda opzione** cercando sulla rete un dataset già formato per le nostre richieste.

3.1 Analisi e scrematura del dataset

Il dataset in questione è reperibile al seguente link: esso è un database pubblico disponibile sulla piattaforma Kaggle; le sue dimensioni sono di 642 MB e contiene più di 50000 immagini di segnali stradali e 43 classi di segnali stradali.

Tali classi sono:

- Limite di velocità 20km
- Limite di velocità 30km
- Limite di velocità 50km
- Limite di velocità 60km
- Limite di velocità 70km
- Limite di velocità 80km
- Fine limite di velocità 80km
- Limite di velocità 100km
- Limite di velocità 120km
- Divieto di sorpasso
- Divieto di sorpasso per i veicoli oltre 3.5 tonnellate
- Intersezione con diritto di precedenza

- Diritto di precedenza
- Dare precedenza
- Stop
- Divieto di transito
- Divieto di transito per autocarri con peso maggiore di 3.5 tonnellate
- Senso vietato
- Segnale di pericolo generico
- Curva pericolosa a sinistra
- Curva pericolosa a destra
- Doppia curva pericolosa
- Strada deformata
- Banchina cedevole
- Strettoia asimmetrica a destra
- Lavori in corso
- Semaforo verticale
- Attraversamento pedonale
- Attraversamento di bambini
- Attraversamento di biciclette
- Pericolo di ghiaccio/neve
- Attraversamento animali selvatici
- Fine divieto
- Obbligo di svoltare a destra
- Obbligo di svoltare a sinistra
- Obbligo di proseguire dirittto
- Obbligo di proseguire diritto o svoltare a destra
- Obbligo di proseguire diritto o svoltare a sinistra
- Obbligo di sorpasso a destra
- Obbligo di sorpasso a sinistra
- Rotatoria

- Fine divieto di sorpasso
- Fine divieto di sorpasso per autocarri con peso maggiore di 3.5 tonnellate

3.2 Formattazione dei dati

Dopo aver selezionato il dataset, abbiamo effettuato le seguenti operazioni per preparare i dati.

3.2.1 Importazione delle librerie e struttura dei dati

Abbiamo importato le librerie dei tool utilizzate dal modello come **Numpy** (per l'utilizzo degli array)**, Pandas (**per la gestione del dataset e la visualizzazione dei dati), **Tensorflow** e **Keras** (per la definizione del modello)**.**

Per quanto riguarda il dataset, esso è costituito da due cartelle: la cartella **Train**, che contiene le immagini (39.209) usate per l'addestramento del modello, e la cartella **Test** che contiene le immagini (12.631) che fanno parte del test set per validare le prestazioni del modello. Inoltre, per ciascuna cartella è presente un file .csv (**Train.csv e Test.csv**) con lo stesso nome, che contiene informazioni riguardo ciascuna immagine presente nella rispettiva cartella. Tali informazioni sono:

- **Width:** larghezza dell'immagine
- **Height:** altezza dell'immagine
- **Roi.X1,Roi.Y1,Roi.X2,Roi.Y2:** sono le coordinate che ci permettono di delimitare la parte di immagine corrispondente al segnale vero e proprio.
- **ClassID:** indica a quale classe di segnali stradali appartiene l'immagine (es 0= limite di velocità 20km, 1= limite di velocità 30km, …)
- **Path:** indica il path in cui trovare l'immagine

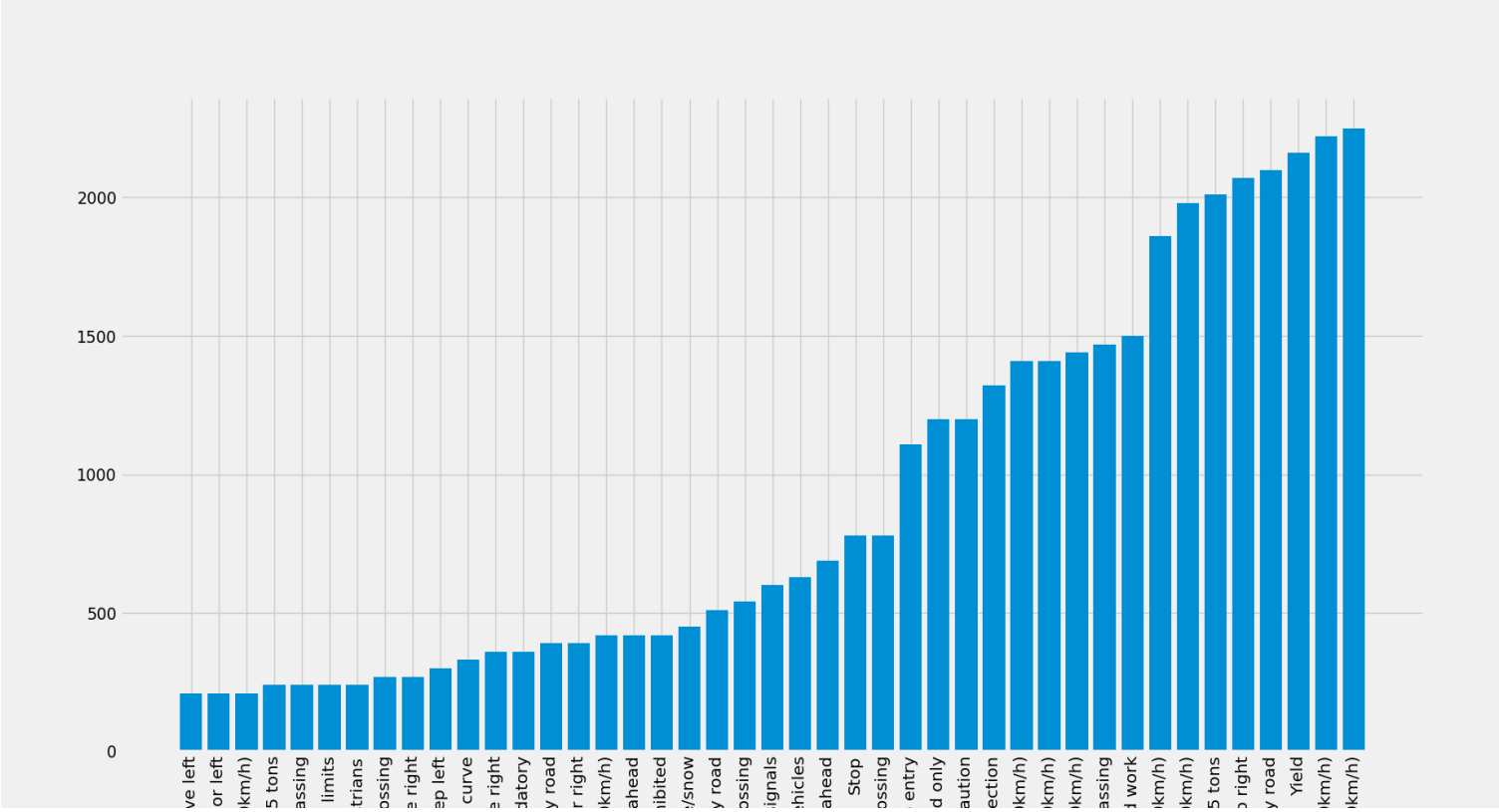
Inoltre, usiamo un dizionario per associare a ciascuna valore dell'etichetta la classe di appartenza.

```
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from PIL import Image
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from sklearn.metrics import accuracy_score
import random
from matplotlib.image import imread
from matplotlib import style
style.use('fivethirtyeight')
np.random.seed(42)

tf.random.set_seed(100)

data_dir = '../Segnali-Stradali/input'
train_path = '../Segnali-Stradali/input/Train'
test_path = '../Segnali-Stradali/input/'

# Le immagini vengono ridimensionate con dimensioni 30x30x3
IMG_HEIGHT = 30
IMG_WIDTH = 30
channels = 3

NUM_CATEGORIES = len(os.listdir(train_path))
print(NUM_CATEGORIES)

# Label per le varie classi
classes = {0: 'Speed limit (20km/h)',
           1: 'Speed limit (30km/h)',
           2: 'Speed limit (50km/h)',
           3: 'Speed limit (60km/h)',
```

	A	B	C	D	E	F	G
1	Width,Height,Roi.X1,Roi.Y1,Roi.X2,Roi.Y2,ClassId,Path						
2	27,26,5,5,22,20,20,Train/20/00020_00000_00000.png						
3	28,27,5,6,23,22,20,Train/20/00020_00000_00001.png						
4	29,26,6,5,24,21,20,Train/20/00020_00000_00002.png						
5	28,27,5,6,23,22,20,Train/20/00020_00000_00003.png						
6	28,26,5,5,23,21,20,Train/20/00020_00000_00004.png						
7	31,27,6,5,26,22,20,Train/20/00020_00000_00005.png						
8	31,28,6,6,26,23,20,Train/20/00020_00000_00006.png						
9	31,28,6,6,26,23,20,Train/20/00020_00000_00007.png						
10	31,29,5,6,26,24,20,Train/20/00020_00000_00008.png						
11	34,32,6,6,29,26,20,Train/20/00020_00000_00009.png						
12	36,33,5,6,31,28,20,Train/20/00020_00000_00010.png						
13	37,34,5,6,32,29,20,Train/20/00020_00000_00011.png						
14	38,34,5,6,32,29,20,Train/20/00020_00000_00012.png						
15	40,34,6,6,34,29,20,Train/20/00020_00000_00013.png						
16	39,34,5,5,34,29,20,Train/20/00020_00000_00014.png						
17	42,36,6,5,37,31,20,Train/20/00020_00000_00015.png						
18	45,39,6,5,40,34,20,Train/20/00020_00000_00016.png						
19	47,42,5,5,41,36,20,Train/20/00020_00000_00017.png						
20	50,45,5,5,45,40,20,Train/20/00020_00000_00018.png						
21	55,49,6,5,49,43,20,Train/20/00020_00000_00019.png						
22	56,51,6,6,51,46,20,Train/20/00020_00000_00020.png						
23	59,54,5,5,54,49,20,Train/20/00020_00000_00021.png						
24	64,57,6,5,59,52,20,Train/20/00020_00000_00022.png						
25	70,61,6,5,64,56,20,Train/20/00020_00000_00023.png						
26	76,69,6,6,70,63,20,Train/20/00020_00000_00024.png						
27	86,75,8,6,79,69,20,Train/20/00020_00000_00025.png						
28	97,87,8,7,89,80,20,Train/20/00020_00000_00026.png						
29	111,100,9,8,102,92,20,Train/20/00020_00000_00027.png						
30	131,119,12,11,120,109,20,Train/20/00020_00000_00028.png						
31	166,152,15,14,152,139,20,Train/20/00020_00000_00029.png						
32	29,31,5,6,24,26,20,Train/20/00020_00001_00000.png						
33	31,30,6,5,25,25,20,Train/20/00020_00001_00001.png						

3.2.2 Costruzione del DataFrame e visualizzazione dei dati

Per costruire il DataFrame, ogni tipologia di immagine è contenuta all'interno di una cartella numerata col numero corrispondente a ciascuna classe di segnali. Nel fare ciò, visualizziamo sia il numero di istanze per ciascuna classe di segnali nel training set e visualizziamo in maniera casuale 25 immagini prese dal test set.


2000
1500
1000
500
0
ve left
or left
km/h)
5 tons
assing
limits
trians
ossing
e right
ep left
curve
e right
datory
y road
r right
km/h)
ahead
ibited
y/snow
y road
ossing
ignals
hicles
ahead
Stop
ossing
entry
d only
aution
ection
km/h)
km/h)
km/h)
assing
l work
km/h)
km/h)
5 tons
o right
y road
Yield
km/h)
km/h)


36
40
31
70
29

3.2.3 Training e Test Set

Una volta raccolto i dati dalla cartella **Train,** mischiamo i dati di training e dividiamo il dataset in 70% train set e 30% test set, tramite la funzione **train_test_split**, che prende in input le immagini (**X**) e la classe da predire (**Y**), dividendo casualmente il dataset in train e test set.

```
image_data = []
image_labels = []

for i in range(NUM_CATEGORIES):
    path = data_dir + '/Train/' + str(i)
    images = os.listdir(path)

    for img in images:
        try:
            image = cv2.imread(path + '/' + img)
            image_fromarray = Image.fromarray(image, 'RGB')
            resize_image = image_fromarray.resize((IMG_HEIGHT, IMG_WIDTH))
            image_data.append(np.array(resize_image))
            image_labels.append(i)
        except:
            print("Error in " + img)


image_data = np.array(image_data)
image_labels = np.array(image_labels)

print(image_data.shape, image_labels.shape)

# Mischiamo i dati di training
shuffle_indexes = np.arange(image_data.shape[0])
np.random.shuffle(shuffle_indexes)
image_data = image_data[shuffle_indexes]
image_labels = image_labels[shuffle_indexes]



X_train, X_val, y_train, y_val = train_test_split(image_data, image_labels, test_size=0.3, random_state=42, shuffle=True)
```

4. Algoritmo di classificazione

4.1 Scelta dell'algoritmo di classificazione

Dopo un'analisi tra tutti gli algoritmi di classificazione che potevano scegliere abbiamo deciso di utilizzare un algoritmo di classificazione basato sulla rete neurale convoluzionale, o CNN. La scelta è motivata poiché la CNN è adatta alla classificazione di immagini relativa al nostro problema.

4.2 Definizione dei layers

TensorFlow offre una serie di librerie e tool per costruire il modello di CNN andando a definire i vari layer della rete neurale convoluzionale.
Le librerie utilizzate nel codice sono:

- **Conv2D**: layer di convoluzione che si occupa di costruire un kernel, ossia una matrice con cui viene convoluto l'input utilizzato per ottenere una Feature Map. Questa mappa rappresenta la parte di immagine (feature) che la rete userà per l'addestramento.
- **MaxPool2D:** si occupa di calcolare il valore massimo o più grande per ciascuna Feature Map creando una rappresentazione che comprende i valori più presenti. Ciò permette di ridurre la dimensione dell'immagine, diminuendo così anche il rischio di overfitting, in quanto permette di evitare che una modifica all'immagine sia interpretata con una feature map diversa, ed è utile anche a livello computazionale poiché il numero di parametri usato per l'apprendimento è minore e quindi abbiamo meno tempo computazionale.
- **Flatten:** layer che si occupa di appiattire i dati ottenuti dalla matrice di convoluzione formata, rendendola un vettore unidimensionale.
- **Dropout:** layer che si occupa di selezionare in maniera casuale un insieme di neuroni (sia del livello di input che del livello nascosto)che verranno ignorati durante la fase di addestramento; questo viene fatto sempre per ridurre il rischio di overfitting

- **Dense:** layer che si occupa selezionare un numero di neuroni che prenderanno a loro volta in input tutti i dati formattati da **Flatten** andando ad effettuare una somma pesata e, assieme al bias, vengono trasferiti attraverso una funzione di attivazione.
- **Batch Normalization:** layer che permette di normalizzare l'input di ciascun layer, in modo da ridurre il rischio di overfitting

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu',
                        input_shape=(IMG_HEIGHT, IMG_WIDTH, channels)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

    keras.layers.Dense(43, activation='softmax')
```

4.3 Generazione di immagini casuali

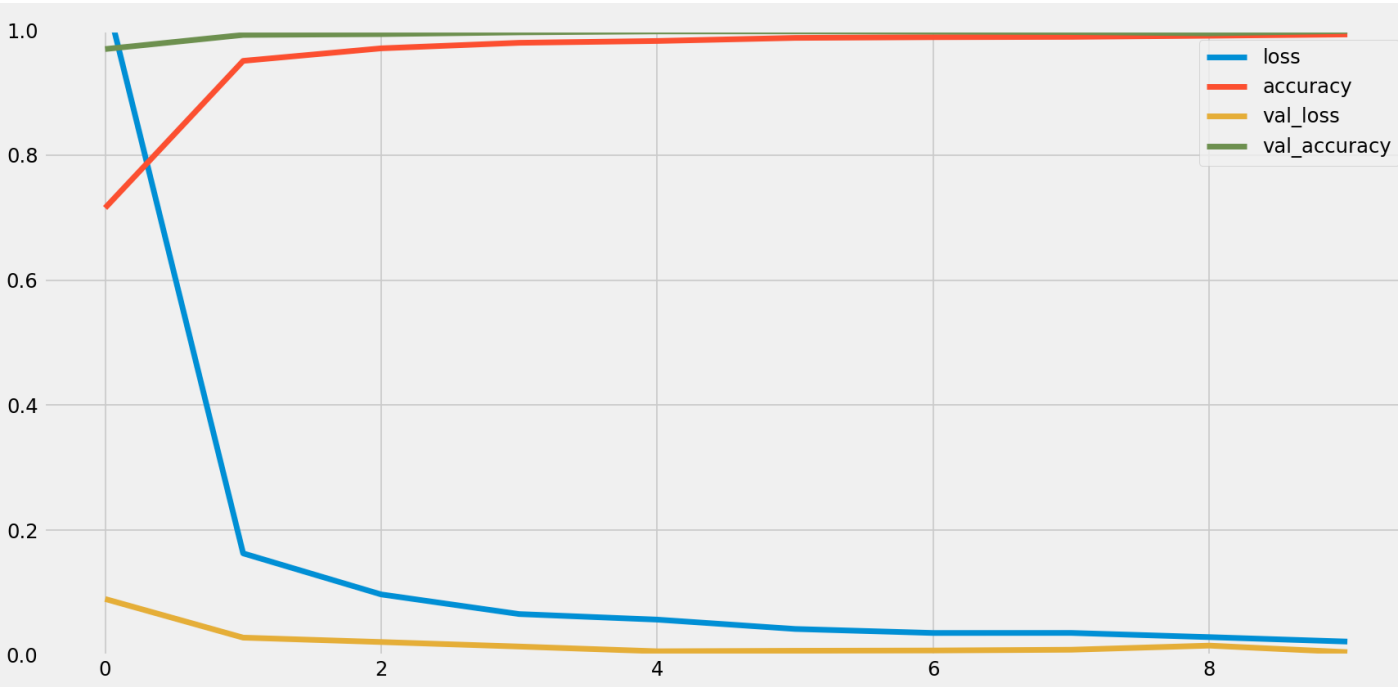
Per ridurre ulteriormente il rischio di overfitting e per ottenere de risultati di accuratezza migliori, abbiamo deciso di generare casualmente delle immagini, usando la classe **Image Data Generator**, una classe utile per generare le immagini casuali a partire da quelle già a disposizione, effettuando diverse operazioni sulle immagini, come ad esempio la rotazione, zoom, flip verticale, flip orizzontale.

4.4 Compilazione e valutazione del modello

Una volta definiti i layer compiliamo il modello di CNN utilizzando la funzione **compile** che configura il modello per l'addestramento e la validazione. Il metodo prende in input:

1. Il parametro **loss,** che indica una funzione di perdita, ossia una misura della differenza tra il valore reale della variabile target e il valore predetto che vogliamo minimizzare
2. **Optimizer,** ottimizzatore che serve a ridurre il valore della funzione di perdita; in questo caso, usiamo l'ottimizzatore **Adam.**
3. **Metrics,** ossia la metrica usata per valutare le prestazioni del modello; in questo caso abbiamo usato **l'accuratezza**.

Effettuiamo, tramite il metodo **fit,** l'addestramento del modello per un numero fissato di epoche che nel nostro caso sono date dal parametro **epochs.** Il metodo prende in input i dati per il training, **X_train,** e i valori per la classe target per i dati di training, **y_train,** con un **batch_size** di 32 campioni per aggiornamento; **validation_data** sono i dati su cui valutare il modello, calcolando l'accuratezza ed eventuali perdite del modello al termine di ogni epoca. Tutto questo viene riassunto da un grafico alla fine dell'addestramento.
Infine, tramite la libreria **Matplotlib,** usiamo una sua interfaccia **Pyplot** che ha il metodo **plot** utilizzato per la costruzione di un grafico che mostra il grado di accuratezza per ogni epoca.



5. Glossario

- **PEAS:** acronimo inglese di Performance Environment Actuators Sensors. È utilizzato nello studio dell'intelligenza artificiale per raggruppare in un unico termine l'ambiente operativo.
- **CNN:** sono un tipo di rete neurale artificiale utilizzata per analizzare le immagini visive
- **Naive Bayes:** è un algoritmo per risolvere problemi di classificazione e apprendimento automatico (machine learning) che utilizza il teorema di Bayes.
- **Decision Tree:** è un classificatore con struttura ad albero (alberi di decisione), in cui ogni nodo può essere o foglia o nodo interno: se foglia, indica il valore della classe assegnata all'istanza; se nodo interno, specifica il test effettuato su un attributo.
- **Dataset:** collezione di dati strutturati in forma relazionale secondo righe e colonne o matrice.
- **Kaggle:** è una comunità online di data scientist e professionisti dell'apprendimento automatico.
- **Numpy:** libreria open source per il linguaggio di programmazione Python, che aggiunge supporto a grandi matrici e array multidimensionali insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati.
- **Keras:** libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python.
- **Pandas:** è una libreria software scritta per il linguaggio di programmazione Python per la manipolazione e l'analisi dei dati.
- **Python:** linguaggio di programmazione di "alto livello", orientato a oggetti.
- **Tensorflow:** ibreria software open source per l'apprendimento automatico che fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio
- **Machine Learning:** branca dell'intelligenza artificiale che raccoglie metodi per definire algoritmi in grado di apprendere dai dati.
- **DataFrame:** struttura dati della libreria Pandas che permette di gestire dati nella forma tabulare.
- **Feature Map:** rappresenta una caratteristica univoca estrapolata da un'immagine, utilizzata dalla CNN per l'apprendimento.