Road-Signs-Classification

|  |
| --- |
| **Composizione Gruppo** |
| Marco Longo | 05121-05945 |
| Antonio Lamberti | 05121-09103 |
| Riccardo Polidoro | 05121-07801 |

Sommario

[**1. Introduzione: scopo del progetto 3**](#_Toc106809600)

[**2. Descrizione dell’agente 3**](#_Toc106809601)

[**2.1 Obiettivi 3**](#_Toc106809602)

[**2.2 Specifica PEAS 3**](#_Toc106809603)

[**2.3 Analisi del problema 4**](#_Toc106809604)

[**3. Dataset 4**](#_Toc106809605)

[**3.1 Analisi e scrematura del dataset 5**](#_Toc106809606)

[**3.2 Formattazione dei dati 5**](#_Toc106809607)

[**3.2.1 Importazione delle librerie ed estrazione delle feature 5**](#_Toc106809608)

[**3.2.2 Costruzione del DataFrame 7**](#_Toc106809609)

[**3.2.3 Training e Test Set 9**](#_Toc106809610)

[**4. Algoritmo di classificazione 10**](#_Toc106809611)

[**4.1 Scelta dell’algoritmo di classificazione 10**](#_Toc106809612)

[**4.2 Definizione dei layers 10**](#_Toc106809613)

[**4.3 Compilazione e valutazione del modello 11**](#_Toc106809614)

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 diverse categorie di segnali (stop,limiti di velocità, divieti, segnali di obbligo, …)

2. Descrizione dell’agente

**2.1 Obiettivi**

Lo scopo del progetto è quello di realizzare un agente intelligente 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](https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign): 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.

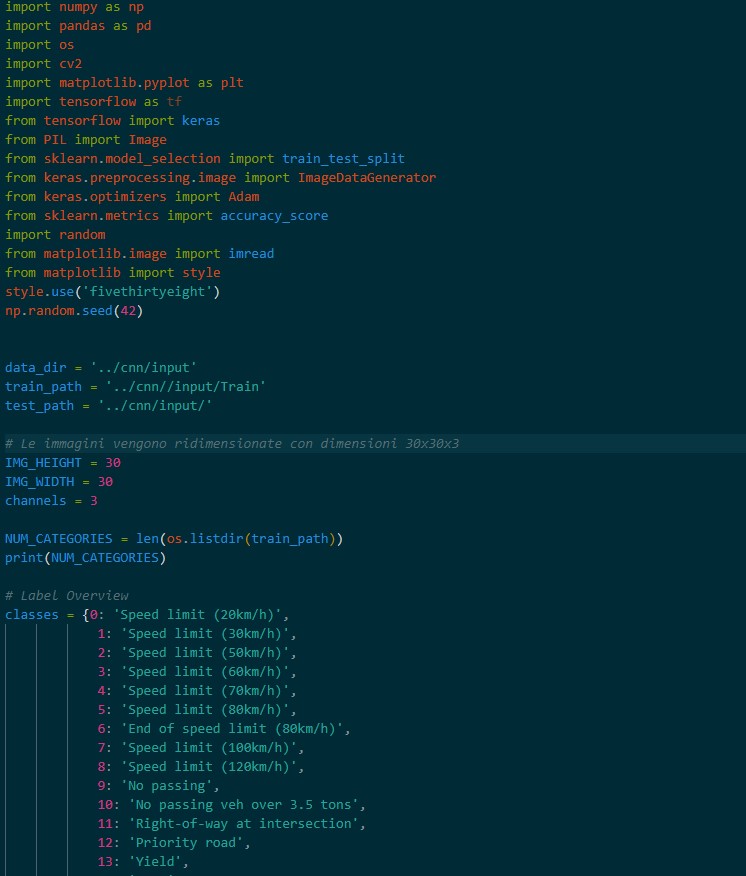
**3.2.1 Importazione delle librerie e struttura dei dati**

Abbiamo importato le librerie dei tool utilizzate dal modello come **Numpy** (per l’utilizzo degli array)**, Pandas (**per la gestione del dataset, la visualizzazione dei dati, ecc…), **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

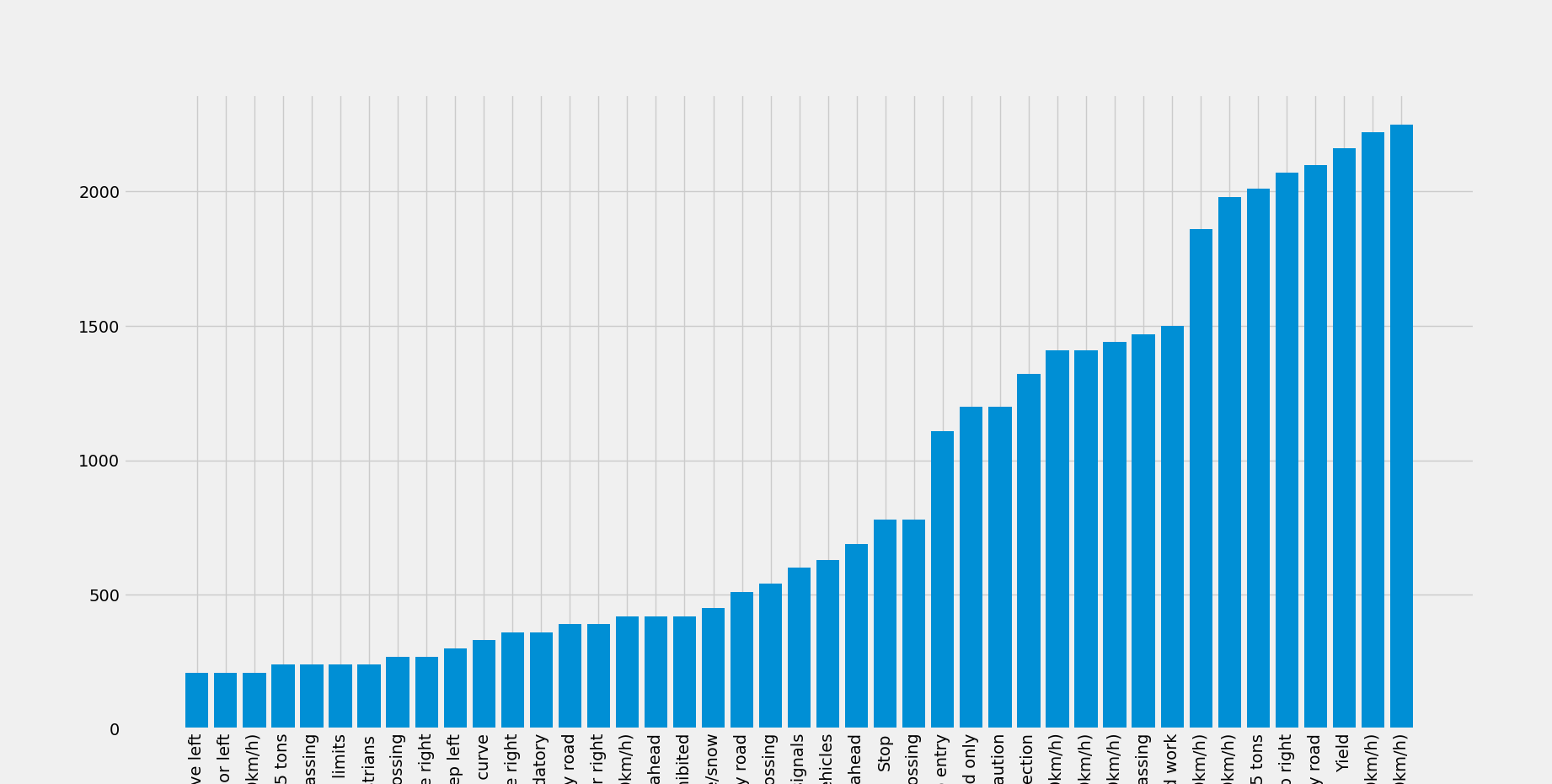
Inoltre, usiamo un dizionario per associare a ciascuna valore dell’etichetta la classe di appartenza.





**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, come detto precedentemente. 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.





**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, traimte 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.

Immagine che contiene testo

Descrizione generata automaticamente

1. 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 puntare tutto sulla rete neurale convoluzionale, o CNN. La scelta è motivata poiché la CNN è adatta alla classificazione di immagini relativa al nostro problema.

Le CNN sono progettare per riconoscere dei pattern visivi in modo diretto, senza molto pre-processing.

**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 serve agli altri layer per apprendere in maniera più indipendente. Questo layer permette di normalizzare l’input di ciascun layer, in modo da ridurre il rischio di overfitting

Immagine che contiene testo

Descrizione generata automaticamente

**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 quali rotazione, zoom, flip verticale, flip orizzontale, ecc…

**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 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, **optimizer,** ottimizzatore che serve a ridurre il valore della funzione di perdita; in questo caso, usiamo l’ottimizzatore **Adam,** e **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.

La funzione **evaluate** fornisce come output il valore della perdita e il valore della metrica del modello durante la fase di testing.

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, sia nella fase di addestramento che nella fasse di test. Immagine che contiene testo

Descrizione generata automaticamente

1. Glossario