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,semafori,limiti di velocità, attraversamenti pedonali, …)

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, nello specifico, segnali di stop, di attraversamento pedonale,limiti di velocità e semafori, 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 originali sono di 642 MB e contiene oltre 12500 immagini di segnali stradali.

Il dataset risultava troppo grande ai fini del progetto e con molte features non utili allo scopo finale. Per questo sono stati effettuate delle operazioni di taglio verticale e orizzontale andando a ridurre il numero a 876 immagini di segnali stradali.

**3.2 Formattazione dei dati**

Dopo aver costruito il nostro dataset andando a selezionare le immagini dai diversi dataset abbiamo iniziato con la stesura del modello. Esso prevede una serie di passaggi da noi descritti.

**3.2.1 Importazione delle librerie ed estrazione delle feature**

Abbiamo importato le librerie dei tool utilizzate dal modello come **Numpy, Pandas** e **Tensorflow.**

Abbiamo strutturato il dataset andando ad associare ad ogni immagine un file .xml che conteneva all’interno dei tag tutte le informazioni di ogni singola immagine.

I tag presenti e di maggior rilievo sono:

* **Folder,** indica il nome della cartella contenente le immagini
* **Filename,** indica il nome dell’immagine associata .png.
* **Size,** indica i campi riguardanti la dimensione dell’immagine
* **Object,** indica diverse feature dell’immagine tra cui è presente il tag **name** che ne indica la tipologia di segnale stradale
* **BndBox,** indica la zona dell’intera immagine in cui è presente il segnale stradale.

Grazie all’utilizzo del pacchetto BeautifulSoup (**bs4**) abbiamo estratto i tag dei file .xml associati alle immagini andando a memorizzare i valori all’interno delle variabili **class\_name, name, width, height, depth.**

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

**3.2.2 Costruzione del DataFrame**

Associamo ad ogni file .xml la corrispettiva immagine memorizzata all’interno del tag **filename**.

Una volta associata controlliamo se **class\_name** è uno **speedlimit,** ovvero un segnale di limite di velocità: in caso di risposta affermativa utilizziamo un counter **speedcounter** per ridurre a 75 il numero di immagini di tipo **speedlimit** da dare al modello; abbiamo quindi applicato una strategia di **undersampling** andando a diminuire il numero di immagini della classe di maggioranza.

In ogni caso estraiamo le dimensioni delle immagini costruendo una tabella memorizzata in **content.**

Grazie alla libreria **Pandas,** libreria utilizzata per la manipolazione e l’analisi dei dati in Python, abbiamo creato tramite il costruttore **DataFrame** una struttua bidimensionale formata da righe e colonne, dove le righe sono i dati estratti e le colonne le feature corrispondenti. Inoltre abbiamo fatto un mapping delle 4 classifiche di immagini con il valore:

* **1,** per i semafori
* **2,** per i limiti di velocità
* **3,** per i attraversamenti pedonali
* **4,** per i stop.

Immagine che contiene testo

Descrizione generata automaticamente

**3.2.3 Training e Test Set**

Abbiamo aperto ogni singola immagine e le abbiamo convertite nel formato .RGB; effettuiamo un resize delle immagini così da facilitare l’apprendimento avendo tutte la stessa taglia. Il 30% delle immagini presenti verranno utilizzate durante la fase di testing e il restante per l’apprendimento.

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.
* **MaxPool2D,** si occupa di calcolare il valore massimo o più grande per ciascuna Feature Map creando una rappresentazione che comprende i valori più presenti.
* **Flatten,** layer che si occupa di rimuovere tutte le dimensioni dopo Conv2D e MaxPool2D. Appiattisce i dati ottenuti dalla matrice di convoluzione formata.
* **Dropout,** layer che si occupa di selezionare in maniera casuale un insieme di neuroni che verranno ignorati durante la fase di addestramento.
* **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.

Immagine che contiene testo

Descrizione generata automaticamente

**4.3 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. Il metodo prende in input il parametro **loss,** una funzione di perdita, **optimizer,** ottimizzatore che implementa l’algoritmo **Adam,** e **metrics,** metrica che il modello deve utilizzare durante la fase di addestramento e di test.

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 in ingresso, **X\_train,** e i dati in uscita, **y\_train,** con un **batch\_size** di 64 campioni per aggiornamento; **validation\_data** sono i dati su cui valutare la metrica ed eventuali perdite del modello al termine di ogni epoca.

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