



SAPIENZA  
UNIVERSITÀ DI ROMA

## Detection e Recognition di cartelli stradali: combinazione di rete neurale e approcci geometrici

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Luca Polenta  
Matricola 1794787

Relatore

Prof. Daniele Nardi

Anno Accademico 2019/2020

---

**Detection e Recognition di cartelli stradali: combinazione di rete neurale e approssimi geometrici**

Tesi di Laurea. Sapienza – Università di Roma

© 2020 Luca Polenta. Tutti i diritti riservati

Questa tesi è stata composta con L<sup>A</sup>T<sub>E</sub>X e la classe Sapthesis.

Email dell'autore: polenta.1794787@studenti.uniroma1.it

## Sommario

Questo progetto ha come obiettivo la realizzazione di una pipeline volta a effettuare il rilevamento e riconoscimento di oggetti (Object Detection and Recognition) in un'immagine non pre-processata. Tale pipeline è ottenuta coniugando tra loro varie metodologie più semplici, fra cui vi sono un algoritmo di Classification, utilizzato per il riconoscimento degli oggetti, e dei filtri geometrici, utilizzati per il rilevamento di questi ultimi. Infine, in questo particolare progetto, i soggetti presi in esame sono varie tipologie di cartelli stradali.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Strumenti Utilizzati</b>	<b>2</b>
2.1	Librerie Importate . . . . .	2
2.2	Funzioni Utilizzate dalla Rete Convoluzionale . . . . .	3
2.3	Training Dataset della Rete Convoluzionale . . . . .	6
2.4	Pre-Processamento del Training Dataset . . . . .	8
2.5	Test Set della Pipeline di Detection e Recognition . . . . .	9
<b>3</b>	<b>Metodologia</b>	<b>12</b>
3.1	Struttura della Rete Neurale Convoluzionale . . . . .	12
3.2	Filtri Geometrici . . . . .	13
3.2.1	Filtro Circolare . . . . .	13
3.2.2	Filtro Triangolare . . . . .	14
3.3	Pipeline dell'Object Detection and Recognition . . . . .	17
<b>4</b>	<b>Risultati</b>	<b>18</b>
4.1	Addestramento e Ottimizzazione della PersonalCNN . . . . .	18
4.1.1	Ricerca del Numero Ottimale di Nodi per i Layers . . . . .	19
4.1.2	Analisi dell'Early Stopping . . . . .	20
4.1.3	Confronto tra le funzioni di attivazione "ReLU" e "LeakyReLU"	20
4.1.4	Analisi del Pre-Processing per il Training Dataset . . . . .	21
4.1.5	Valutazione Finale dei Risultati Ottenuti . . . . .	21
4.2	Analisi e Miglioramento della Pipeline Completa . . . . .	26
4.2.1	Ottimizzazione del Rilevamento dei Cartelli Circolari . . . . .	26
4.2.2	Ottimizzazione del Rilevamento dei Cartelli Triangolari . . . . .	28
4.2.3	Analisi dei Canali di Colore . . . . .	30
4.3	Valutazione e Conferma dei Risultati Finali . . . . .	31
<b>5</b>	<b>Conclusioni</b>	<b>34</b>
	<b>Bibliografia</b>	<b>36</b>

# Capitolo 1

## Introduzione

Una pietra miliare nel campo delle scienze informatiche è il "machine learning". Esso è divenuto un caposaldo per quanto riguarda la creazione di algoritmi in grado di prendere decisioni complesse in autonomia. Tale risultato è ottenuto utilizzando metodi statistici per migliorare le performance di un algoritmo nell'identificare pattern nei dati disponibili in input. Nell'attuale progetto i dati in input sono delle immagini e i metodi statistici precedentemente citati cercano e sfruttano pattern comuni nelle forme e nei colori per stabilire quali sono le caratteristiche che contraddistinguono le immagini tra le varie classi. In particolare, tale tipo di procedura viene definita "classification", che per l'appunto si occupa di classificare e distinguere in sottoinsiemi, definiti "classi", determinati oggetti appartenenti ad un macro-insieme generico. Un limite che però si presenta con gli algoritmi di Classification riguarda l'impossibilità di analizzare immagini complesse e non pre-processate. Un algoritmo di questo tipo, infatti, richiede che in input vengano usate delle immagini in cui vi è principalmente l'oggetto che si cerca di classificare. Questo rende tale approccio poco utile ad un'applicazione reale, in quanto non esistono nel mondo fisico dei casi in cui gli oggetti siano isolati, distinti e ben distinguibili dagli altri. Più in particolare, prendendo in esame i cartelli stradali, non esistono automobili che si muovano in strade prive di oggetti diversi da questi ultimi e in cui sia possibile acquisire immagini che presentano esclusivamente cartelli stradali. Questo problema ha quindi portato alla creazione di nuovi algoritmi, i quali sono in grado di elaborare delle immagini non pre-processate e ricche di oggetti, al fine di individuare prima il soggetto d'interesse su cui svolgere solo in un secondo momento la classificazione. Una delle prime tecniche che fu sviluppata per portare a termine tale compito sfruttava i filtri geometrici per la rilevazione degli oggetti in esame. Tramite questi ultimi, infatti, è possibile individuare vari tipi di forme all'interno di immagini generiche, e pertanto estrarre i contenuti che presentino le forme degli oggetti desiderati. Sarà quindi con questo approccio che, dopo aver creato ed ottimizzato un algoritmo di Classification per immagini pre-processate, verrà espanso in un algoritmo di "Object Detection and Recognition" per immagini complesse e non precedentemente elaborate.

## Capitolo 2

# Strumenti Utilizzati

In questo capitolo sono esplicitati diversi strumenti che sono stati adoperati per portare a termine l'obiettivo preposto. Di fondamentale importanza saranno le librerie importate, e le funzioni da esse carpite, per creare ed ottimizzare la rete di classificazione e per strutturare i filtri geometrici dediti all'individuazione delle forme geometriche d'interesse. Inoltre verranno introdotti il dataset utilizzato nell'addestramento della rete convoluzionale, con le rispettive tecniche di augmentation per aumentare la realistica dei soggetti, e il dataset da cui sono state prese le immagini per i successivi test di object detection.

### 2.1 Librerie Importate

Al fine di testare sperimentalmente il funzionamento della pipeline, è stato creato un codice nel quale sono state importate svariate librerie, le quali a loro volta hanno messo a disposizione strumenti, risorse e funzioni utili alla creazione, allo sviluppo e alla fase testing di tale codice. Due delle librerie più importanti e fondamentali alla creazione del suddetto codice sono "Tensorflow"<sup>[1]</sup> e "OpenCV"<sup>[2]</sup>.

La prima è una libreria per l'apprendimento automatico che mette a disposizione un ecosistema completo, modulare e affidabile di risorse e strumenti, ognuno dei quali è stato già precedentemente testato e ottimizzato per portare a termine il proprio compito in maniera eccelsa. Specificatamente è stata utilizzata l'API di alto livello chiamata "Keras"<sup>[3]</sup>, la quale facilita l'interazione con il modello e il suo debug. Da quest'ultima API sono stati importati diversi elementi, tra cui i principali sono: i modelli, che occorrono per strutturare la rete e per caricarla una volta addestrata e salvata; i layers, i quali definiscono gli strati attraversati dalle immagini durante l'addestramento e che caratterizzano la rete in quanto tale; e le funzioni di callbacks da applicare alla rete, come ad esempio l'earlyStopping.

Invece la libreria "OpenCV" (acronimo di “Open Source Computer Vision Library”) è una libreria utilizzata nell’ambito dell’apprendimento automatico (machine learning) legato alle immagini, o più in generale alla “Artificial Computer Vision”. La libreria è molto popolare in quanto incorpora più di 2500 algoritmi ottimizzati e facilmente personalizzabili rispetto alle esigenze poste dal problema in esame. Questi algoritmi possono essere utilizzati per svariati scopi che sfociano in molteplici ambiti, ma ai fini di questo progetto saranno utilizzati gli algoritmi dediti principalmente

all'identificazione degli oggetti partendo dall'analisi delle loro forme.

Infine sono state utilizzate ulteriori librerie, tra cui "matplotlib", "sys", "numpy", "csv", "os" e "math", le quali hanno permesso di interagire con il sistema, visualizzare grafici e immagini, leggere file in formati terzi e svolgere computazioni matematico-scientifiche di varia difficoltà.

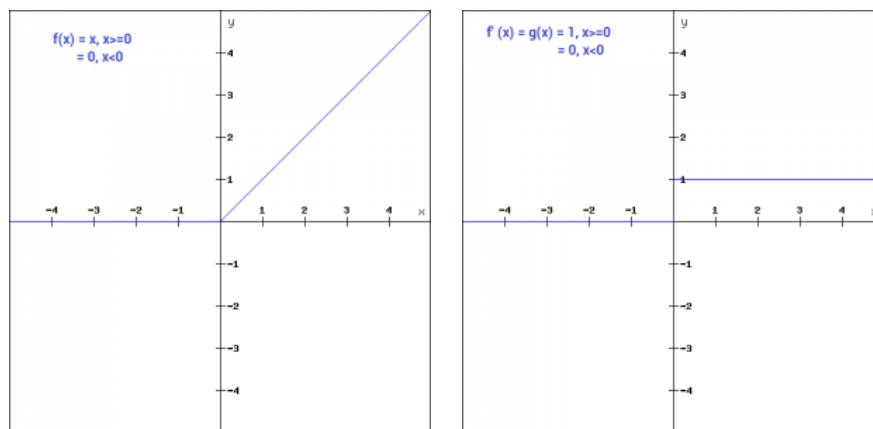
## 2.2 Funzioni Utilizzate dalla Rete Convoluzionale

Le funzioni presenti nella rete convoluzionale si dividono in due categorie: le macro-funzioni, o metodi, che caratterizzano la rete in quanto tale, e le funzioni che non derivano direttamente dalla libreria Tensorflow o da Keras, ma che si presenta come opzioni per i parametri delle macro-funzioni precedentemente citate. Le prime funzioni menzionate verranno analizzate successivamente, mentre in questo paragrafo verrà esplicitata la seconda categoria di funzioni. Si è deciso di trattare queste ultime in quanto esse sono di fondamentale importanza nel determinare l'efficienza e il comportamento della rete durante lo svolgimento del suo compito.

Prime fra esse vi sono le funzioni di attivazione. Tali funzioni definiscono l'uscita di ogni nodo dato l'input che ognuno di essi riceve. Inoltre, mappano tale valore risultante all'interno di uno specifico intervallo, il quale è per l'appunto determinato a seconda della funzione di attivazione scelta. Le funzioni di attivazione usate<sup>[4, 5, 6]</sup> nell'attuale rete neurale convoluzionale sono la ReLU (Rectified Linear Unit), la LeakyReLU e la softmax. Quest'ultima in particolare è stata usata come funzione di attivazione nel layer finale fully-connected della rete e verrà poi anche usata dalla funzione loss, come illustrato successivamente.

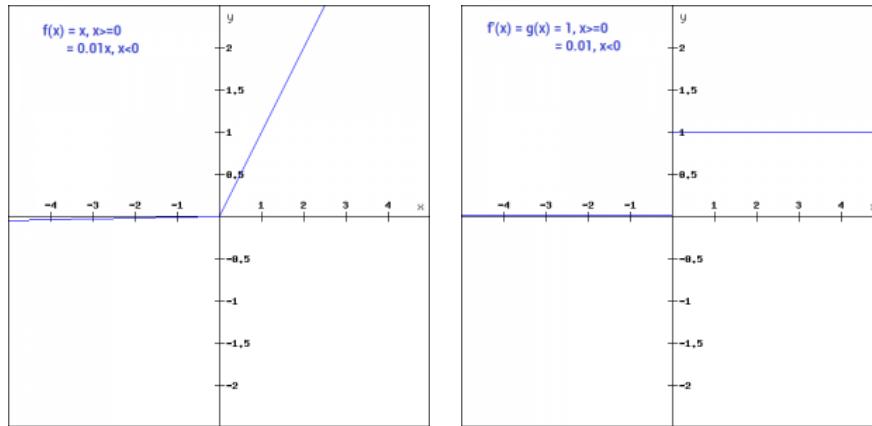
La formula matematica che esprime la funzione di attivazione "ReLU" è:

$$f(x) = \begin{cases} 0 & \text{when } x \leq 0 \\ x & \text{when } x > 0 \end{cases} \quad f'(x) = \begin{cases} 0 & \text{when } x \leq 0 \\ 1 & \text{when } x > 0 \end{cases}$$



Invece la formula matematica che esprime la funzione di attivazione "LeakyReLU" è:

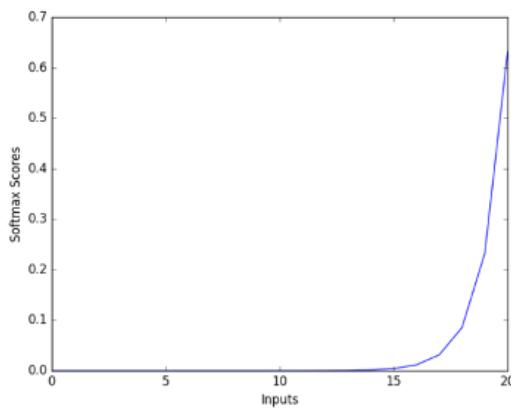
$$f(x) = \begin{cases} 0.001 * x & \text{when } x \leq 0 \\ x & \text{when } x > 0 \end{cases} \quad f'(x) = \begin{cases} 0.001 & \text{when } x \leq 0 \\ 1 & \text{when } x > 0 \end{cases}$$



Infine la formula matematica che esprime la funzione di attivazione "Softmax" è:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Dove  $s_i$  sono i punteggi inferiti dalla rete per la classe  $i$ -esima appartenente all'insieme delle classi C. Il rispettivo andamento grafico è il seguente:



Si è scelto di utilizzare le funzioni di attivazione ReLU e LeakyReLU sia per la loro leggerezza computazionale, in quanto permettono di non appesantire eccessivamente la rete, sia per l'eccellente efficienza nel computare risultati ottimali. In particolare,

si è deciso di utilizzarle entrambe per confrontare il caso della ReLU, la quale può portare alla completa disattivazione dei nodi più inefficienti, con la LeakyReLU, la quale invece non disattiva mai completamente tali nodi, lasciando che questi si addestrino senza influenzare negativamente il risultato finale. Quest'ultima casistica è rilevante in quanto a volte permette ai nodi inizialmente inutili di poter dare un contributo positivo alla rete in fasi successive dell'addestramento. Invece, per quanto riguarda la funzione di attivazione softMax, essa viene utilizzata nel layer finale ed è stata scelta in quanto rielabora egregiamente i logits che si ottengono come output dell'ultimo livello, trasformandoli in distribuzioni di probabilità normalizzati tra 0 e 1.

In aggiunta, un'altra funzione rilevante per la rete è quella utilizzata per il calcolo della loss. La funzione selezionata in questo caso è la "Categorical Cross-Entropy"<sup>[7]</sup>. Il calcolo della loss si rende necessario sia durante l'addestramento che nei test successivi per comprendere quanto i risultati ottenuti si discostino dall'effettivo risultato che si vorrebbe ottenere. La "Categorical Cross-Entropy" spesso è anche definita "Softmax Loss" in quanto è composta da una funzione di attivazione Softmax sommata alla loss Cross-Entropy. La rispettiva formula matematica è:

$$\text{Cross - Entropy} : \quad CE = - \sum_i^C t_i * \log(f(s)_i)$$

Dove:  $f(s)_i$  è la funzione di attivazione Softmax descritta dalla formula precedentemente esposta;  $s_i$  sono i punteggi inferiti dalla rete per la classe  $i$ -esima appartenente all'insieme delle classi C;  $t_i$  è il vettore Target (vettore che contiene i veri output).

Infine, come funzione di ottimizzazione (optimizer) della rete è stato utilizzato "Adam"<sup>[8]</sup>. L'optimizer permette di minimizzare la funzione di costo, individuando durante l'addestramento i valori ottimizzati da assegnare ai pesi (weights) relativi ad ogni nodo. È stato scelto "Adam" in quanto combina esaurientemente l'optimizer Momentum, caratterizzato da una veloce e ottima ricerca del minimo, e l'optimizer RMSProp, che impedisce eccessive oscillazioni durante tale ricerca. La rispettiva formula matematica è:

$$\begin{aligned} \text{Per ogni valore in } \omega_j : \quad \omega_{t+1} &= \omega_t + \Delta\omega_t \\ \Delta\omega_t &= -\eta * \frac{v_t}{\sqrt{s_t + \epsilon}} * g_t \\ v_t &= \beta_1 * v_{t-1} - (1 - \beta_1) * g_t \\ s_t &= \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \end{aligned}$$

Dove:  $\eta$  è il learning rate iniziale;  $v_t$  è la media esponenziale del gradiente lungo  $\omega_j$ ;  $s_t$  è la media esponenziale quadratica del gradiente lungo  $\omega_j$ ;  $g_t$  è il gradiente al tempo  $t$  di  $\omega_j$ ;  $\beta_1$  e  $\beta_2$  sono degli iperparametri che permettono di regolare la dipendenza di Adam rispettivamente dall'approccio del Momentum e dall'approccio dell'RMSProp;  $\epsilon$  è una costante molto piccola per evitare di avere divisioni e radici per zero.

## 2.3 Training Dataset della Rete Convoluzionale

Il dataset di cartelli stradali che verrà utilizzato per addestrare la rete neurale si chiama "Traffic Signs Classification"<sup>[9]</sup> ed è costituito da un totale di 43 classi, ciascuna contenente un numero variabile di samples. Il dataset presenta un totale di 54658 samples per il train set, di 9623 samples per il validation set e di 8858 samples per il test set. Inoltre i samples di ogni classe hanno tutti la medesima dimensione di 32x32 pixels. Una caratteristica fondamentale di tale dataset consiste nella diversificazione all'interno di una stessa classe: segnali stradali simili sono stati acquisiti in momenti diversi della giornata, in modo da avere alcuni samples ritratti di notte, in condizioni di illuminazione variabile o in condizioni atmosferiche avverse ad un loro immediato riconoscimento. In aggiunta, sono presenti dei samples per ogni classe in cui è stata particolarmente accentuata la forma e le caratteristiche peculiari tramite l'applicazione di una colorazione a contrasto blu/bianco e l'equalizzazione di ogni bordo dei soggetti in esame.

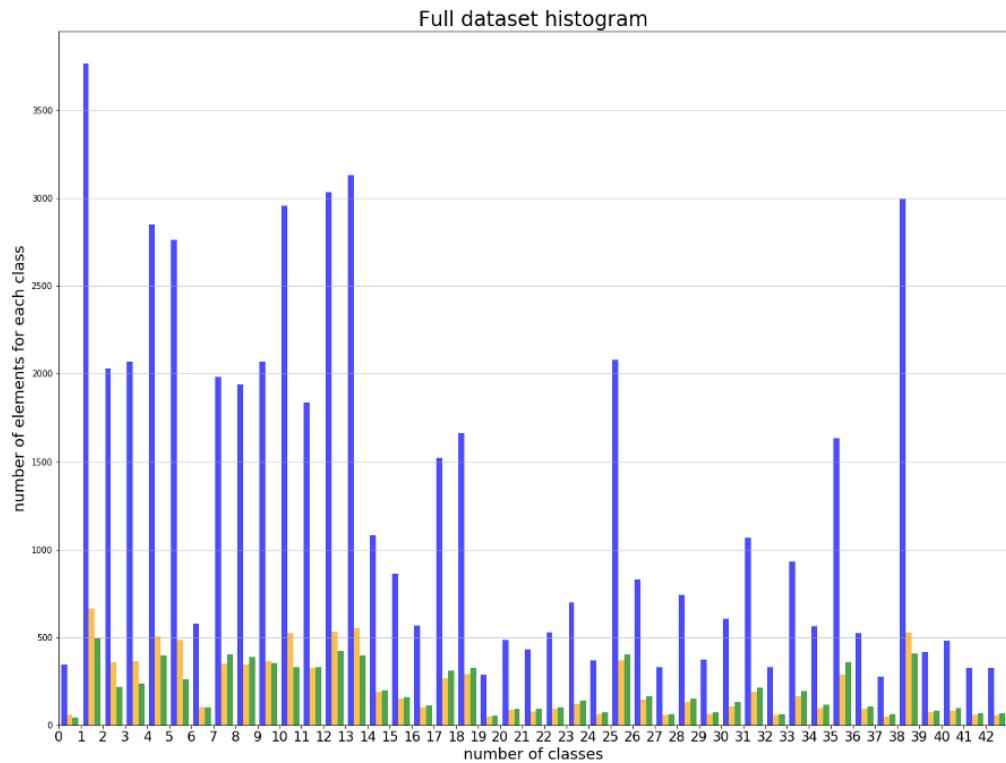
In aggiunta a tale dataset sono stati valutati il dataset italiano DITS-Classification, subset del dataset "DITS"<sup>[10]</sup> (Dataset of Italian Traffic Signs) e il dataset tedesco "GTSDB"<sup>[11]</sup> (The German Traffic Sign Detection Benchmark). Questi ultimi presentano una diversa varietà di classi, in numero o in tipo di segnali stradali, ma entrambi sono risultati inefficienti. Essi presentano un minor numero di samples per ogni classe e la rete utilizzata non è stata in grado di ottenere dei risultati soddisfacenti, nemmeno con l'ausilio di un'augmentation che tentasse di colmare le lacune per quanto riguarda la numerosità o la variabilità dei samples.

Inoltre, al fine di specificare con maggior precisione la densità di ogni classe del dataset, è stata stilata una tabella che racchiude le seguenti informazioni per ogni classe: il numero associato alla classe, il rispettivo nome comune e il numero di samples presenti nel training set, nel validation set e nel test set. Il numero di samples presenti nei validation sets di ogni classe è stato calcolato come il 15% del numero di elementi presenti nei training sets di ogni classe (arrotondato per difetto), mentre il numero di samples del training set corrisponde al restante 85% (arrotondato per eccesso):

NUM	NOME CLASSE	TRAIN SET	VALID SET	TEST SET
0	Speed limit (20km/h)	346	60	46
1	Speed limit (30km/h)	3765	664	492
2	Speed limit (50km/h)	2034	358	219
3	Speed limit (60km/h)	2070	365	237
4	Speed limit (70km/h)	2850	502	400
5	Speed limit (80km/h)	2993	487	260
6	End of speed limit (80km/h)	578	101	103
7	Speed limit (90km/h)	1982	349	407
8	Speed limit (100km/h)	1939	342	391
9	No passing	2071	365	355

NUM	NOME CLASSE	TRAIN SET	VALID SET	TEST SET
10	No passing for vechiles over 3.5 metric tons	2109	372	330
11	Right-of-way at the next intersection	1837	324	330
12	Priority road	3036	535	420
13	Yield	2994	528	399
14	Stop	1080	190	201
15	No vehicles	862	151	158
16	Vehicles over 3.5 metric tons prohibited	568	100	114
17	No Entry	1523	268	310
18	General caution	1665	293	324
19	Dangerous curve to the left	288	50	53
20	Dangerous curve to the right	486	85	91
21	Double curve	432	76	93
22	Bumpy road	526	92	102
23	Slippery road	699	123	140
24	Road narrows on the right	370	65	76
25	Road work	2080	367	405
26	Traffic signals	830	146	166
27	Pedestrians	386	57	66
28	Children crossing	714	130	150
29	Bicycles crossing	373	65	72
30	Beware of ice/snow	605	106	129
31	Wild animals crossing	1069	188	214
32	End of all speed and passing limits	329	57	66
33	Turn right ahead	932	164	194
34	Turn left ahead	564	99	119
35	Ahead only	1636	288	357
36	Go straight or right	522	92	107
37	Go straight or left	278	49	65
38	Keep right	2994	528	410
39	Keep left	416	73	83
40	Roundabout mandatory	480	84	98
41	End of no passing	327	57	67
42	End of no passing by vehicles over 3.5 metric tons	328	57	67

Infine è possibile visionare anche l'istogramma delle densità di ogni classe. In blu è riportata la densità del training set, in giallo la densità del validation set e in verde la densità del test set:



## 2.4 Pre-Processamento del Training Dataset

Per addestrare la rete a riconoscere i cartelli stradali in situazioni che siano il più possibile realistiche è di fondamentale importanza svolgere un buon pre-processing delle immagini. In realtà il dataset presenta già una parziale augmentation, sia a livello quantitativo che per quanto riguarda il contenuto di ogni sample, come illustrato nel paragrafo precedente. Oltre a tali augmentation però si è deciso di eseguire anche le ulteriori modifiche:

- shear\_range: applica una distorsione angolare per deformare leggermente l'immagine, simulando situazioni in cui i cartelli non sono paralleli alla fotocamera, ma si presentano lievemente ruotati in profondità. Quest'ultimo può rappresentare il caso realistico in cui i ganci allentati dei cartelli hanno permesso una lieve rotazione di quest'ultimi, per esempio per via del vento. È stata impostato a 0.1 nel test e a 0.15 nel train/validation;

- zoom\_range: applico uno zoom alle immagini. Aiuta la rete a riconoscere cartelli ravvicinati e a volte ritagliati. È stata impostato a 0.1 nel test e a 0.15 nel train/validation;
- width\_shift\_range: trasla l'immagine orizzontalmente, facendo valutare casi in cui l'immagine stessa venga tagliata orizzontalmente. È stata impostato a 0.1 nel test e a 0.15 nel train/validation;
- height\_shift\_range: sposta l'immagine verticalmente, facendo valutare casi in cui l'immagine stessa venga tagliata verticalmente. È stata impostato a 0.1 nel test e a 0.15 nel train/validation;
- fill\_mode: è stata impostata a constant cosicché, una volta eseguite le precedenti distorsioni, le aree esterne vengano colmate di colore nero. È sembrato il caso più realistico in quanto con l'opzione di default “nearest” gli spazi in eccesso vengono colmati con parti dell'immagine stessa specchiata, rappresentando dunque un caso poco realistico.

Sono state invece disattivate le seguenti augmentation:

- vertical\_flip: è impostata a false in quanto in cartelli non possono trovarsi invertiti verticalmente;
- horizontal\_flip: è impostata a false in quanto i cartelli non possono trovarsi invertiti orizzontalmente e, in alcuni casi, possono persino cambiare significato, e quindi classe, come nel caso dell'esempio riportato successivamente;
- rotation\_range: è stata disattivata in quanto non sono presenti cartelli roteati. Essi infatti generalmente sono fissati da due o più ganci ad un palo verticale, che quindi ne potrebbe permettere al massimo solo uno scorrimento dall'alto verso il verso. Inoltre, ruotare i cartelli potrebbe far cambiare loro significato, e di conseguenza anche classe. Un esempio di tale problema è visibile negli indicatori di corsia:



## 2.5 Test Set della Pipeline di Detection e Recognition

Per quanto riguarda le immagini di ambienti realistici e ricchi di oggetti, sono stati estratti dei campioni dal dataset "DITS-Detection", il quale è un subset del dataset "DITS"<sup>[10]</sup>. Su tali immagini verrà effettuata una preliminare ricerca dei cartelli triangolari o circolari, per poi applicare l'algoritmo di classificazione solo ai cartelli individuati. Un dei motivi per cui si è deciso di ricorrere a tale dataset è perché le immagini contenute in esso presentano rilevazioni stradali ottime e in differenti momenti della giornata. Questa caratteristica permette di avere samples ritratti di

notte con illuminazione scarsa o artificiale, di giorno in condizioni di illuminazione variabile o in condizioni atmosferiche avverse ad un loro immediato riconoscimento, come nel caso di foschia o nebbia. Ciò ha permesso, pertanto, di verificare il corretto funzionamento del rilevamento e poi il successivo riconoscimento dei cartelli stradali in casi molto realistici. Un’ulteriore caratteristica delle immagini di questo dataset è che tutte presentano una dimensione fissa di 1280x720 pixels, la quale ha facilitato l’individuazione di pochi valori di threshold ottimi per tutte le immagini.

In aggiunta al suddetto dataset, è stato valutato anche l’utilizzo delle immagini di test contenute nel dataset tedesco "GTSDB"<sup>[11]</sup>, ma i filtri applicati a tali immagini hanno riportato diverse problematiche: prima fra tutte l’eccessivo numero di false rilevazioni di possibili cartelli. È complesso determinare la causa specifica che ha portato al verificarsi di tale fenomeno, ma in parte può essere ricondotto a delle caratteristiche intrinseche nelle immagini in sé. La maggior parte di esse, infatti, presenta eccessiva sovrabbondanza di illuminazione, la quale molto spesso ha reso persino il cielo bianco. Quest’ultimo dettaglio è di particolare rilevanza in quanto sembra ingannare i filtri, facendo individuare proprio in esso molte false rilevazioni. Data tale osservazione, si è valutato di analizzare solo una porzione dell’immagine, che per l’appunto non comprenda la zona superiore dove vi è minor probabilità di trovare cartelli stradali, ma ad ogni modo non ci sono stati miglioramenti degni di nota. In aggiunta, le immagini del dataset "GTSDB" presentano quasi sempre cartelli stradali molto piccoli, in quanto lontani dal punto di rilevazione della foto. Questa caratteristica è svantaggiosa ai fini del progetto attuale, in quanto cartelli eccessivamente remoti potrebbero riferirsi a punti o incroci stradali lontani e non immediatamente rilevanti. L’intento di questo progetto, infatti, è quello di individuare cartelli che non siano né troppo vicini e né troppo lontani dal punto di rilevazione delle immagini, il quale generalmente è associato ad una telecamera posizionata sul tetto o sul parabrezza dell’automobile. Qui di seguito sono inoltre riportate due esempi di immagini provenienti dal dataset "GTSDB" per illustrare visivamente le situazioni e le problematiche appena citate:



**Figura 2.1.** Esempio n.1 delle immagini di test del dataset "GTSDB"



**Figura 2.2.** Esempio n.2 delle immagini di test del dataset "GTSDB"

Pertanto, date tutte le considerazioni espresse finora, si è deciso di scartare l’analisi sulle immagini di test contenute nel dataset "GTSDB", a favore invece dell’utilizzo del dataset "DITS-Detection" per tale compito.

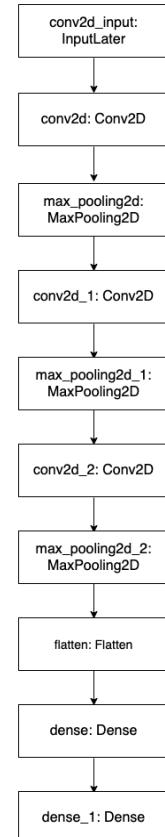
# Capitolo 3

## Metodologia

In questo capitolo verranno illustrate le metodologie utilizzate nella creazione della rete convoluzionale, nella strutturazione dei filtri geometrici e nell'organizzazione della pipeline finale, la quale combina le suddette metodologie per fornire un algoritmo di Detection e Recognition dei cartelli stradali.

### 3.1 Struttura della Rete Neurale Convoluzionale

La rete neurale convoluzionale, chiamata PersonalCNN per semplicità, è composta da tre layers convoluzionali bidimensionali (Conv2D) alternati da tre layers di max pooling bidimensionali (MaxPooling2D) di dimensione 2x2. Nei layers Conv2D saranno valutati diversi valori per il numero di nodi e le funzioni di attivazione “ReLU” (Rectified Linear Unit) e “LeakyReLU”. Il primo layer Conv2D ha un kernel di dimensione 7x7, il secondo di dimensione 5x5 e il terzo di dimensione 3x3. Le dimensioni dei kernel nei layers diminuiscono gradualmente poiché è stato notato empiricamente come tale struttura tenda ad ottenere risultati migliori rispetto ad altre combinazioni. Inoltre, il primo layer specifica la dimensione delle immagini che riceve in input: le immagini devono essere 32x32 pixels e presentare i tre canali di colore (RGB). Dopo il terzo layer di MaxPooling2D vi è un layer Flatten che permette di passare dalla struttura tridimensionale della matrice nello strato precedente ad un vettore monodimensionale che rappresenti ugualmente tutti i risultati ottenuti. Il primo dei due è costituito da 512 nodi e nei test che verranno svolti successivamente potrà presentare come funzione di attivazione sia la “ReLU” che la “LeakyReLU”. Infine, l'ultimo layer è formato da un numero di nodi pari al numero di classi e come funzione di attivazione utilizza la Softmax, la quale predice la classe di appartenenza trasformando i logits del layer precedente in una distribuzione di probabilità con valori normalizzati tra 0 e 1 e scegliendo il valore più elevato.

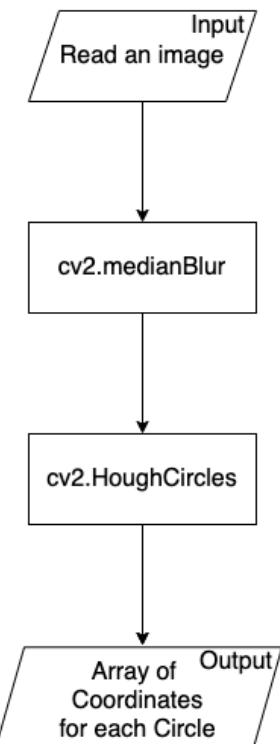


## 3.2 Filtri Geometrici

I filtri geometrici sono essenziali per individuare i cartelli stradali all'interno di un'immagine non pre-processata in cui vi sono una moltitudine di oggetti. Si è scelto di utilizzare tale approccio in quanto è uno dei metodi che coniuga meglio l'efficienza prestazionale e la leggerezza computazionale desiderata, ma esistono anche altri metodi più complessi, efficienti e dal maggior costo computazionale. Ad ogni modo, le funzioni utilizzate in tali filtri sono importate dalla libreria OpenCV<sup>[2]</sup>. I filtri in esame hanno la caratteristica di rilevare la presenza di forme poligonali o ellisoidali, le quali successivamente vengono filtrate per eliminare il più possibile le false rilevazioni che non corrispondono a dei cartelli stradali. Il filtraggio dei primi risultati ottenuti può essere di due tipologie: di dimensione o di forma. Una scrematura per quanto riguarda la dimensione delle rilevazioni si rende necessaria per eliminare i riscontri eccessivamente grandi o eccessivamente piccoli, in quanto rappresentano casi limite improbabili o di poca rilevanza, come è stato espresso anche nel capitolo precedentemente. Invece una scrematura per quanto riguarda la forma si rende necessaria perché i cartelli stradali non assumono tutte le forme ellisoidali o poligonali esistenti. Principalmente la maggior parte dei essi presenta una forma triangolare o circolare, pertanto i filtri trattati in questo progetto cercheranno di individuare queste due forme geometriche.

### 3.2.1 Filtro Circolare

Il filtro che ricerca la presenza di cartelli circolari all'interno di un'immagine è composto principalmente da due step: dopo aver caricato l'immagine tramite la funzione "imread" di OpenCV, essa subisce una prima elaborazione tramite la funzione "MedianBlur" e poi il risultato di tale elaborazione viene fatto analizzare dalla funzione "HoughCircles" per individuare forme circolari. La funzione MedianBlur sostituisce l'elemento centrale dell'immagine con la mediana di tutti i pixel nell'area del kernel, il quale è stato settato di dimensione 5x5 pixels. Questa operazione rielabora i bordi rimuovendo il rumore. Successivamente, per l'effettiva ricerca dei cerchi, la funzione HoughCircles implementa il metodo del gradiente di Hough (cv2.HOUGH\_GRADIENT), che si compone di due fasi principali: la prima fase prevede il rilevamento dei bordi e l'individuazione dei possibili centri del cerchio; la seconda fase trova il raggio migliore per ogni centro candidato. Al termine di questa operazione, la funzione restituisce un array di coordinate che indicano la posizione e la grandezza dei cerchi individuati tramite le coordinate del centro (x, y) e del raggio. Nell'immagine a destra è possibile visionare una pipeline delle operazioni svolte sulle immagini di test.



La funzione HoughCircles, inoltre, è caratterizzata da ulteriori parametri che sono stati settati come segue per ottenere i migliori riscontri possibili:

- dp: corrisponde al rapporto inverso tra la risoluzione dell'accumulatore e la risoluzione dell'immagine. Nell'attuale casistica è stato settato a 2, quindi l'accumulatore ha larghezza e altezza pari alla metà della risoluzione dell'immagine;
- minDist: corrisponde alla distanza minima tra i centri dei cerchi rilevati. Se il parametro è troppo piccolo, più cerchi vicini potrebbero essere rilevati erroneamente oltre a quello più probabile. Se invece è troppo grande, alcuni cerchi potrebbero non essere riportati. Empiricamente, date le immagini di test a disposizione, si sono notati risultati migliori con il parametro settato a 20;
- param1: corrisponde al valore più alto di threshold che viene passato alla funzione "Canny Edge Detector". Il minimo valore di threshold è due volte più piccolo. Empiricamente è stato individuato che il valore ottimo per le immagini a disposizione sia 300;
- param2: corrisponde alla soglia dell'accumulatore per individuare i centri del cerchio in fase di rilevamento. Più il valore è piccolo, maggiore è la probabilità di avere false rilevazioni di cerchi. Empiricamente è stato individuato che il valore ottimo per le immagini a disposizione sia 50;
- minRadius: corrisponde alla lunghezza minima del raggio, calcolata in pixel. Data una stima della dimensione e della forma dei cartelli che stiamo cercando, empiricamente è stato individuato che il valore ottimo per le immagini a disposizione è 25;
- maxRadius: corrisponde alla lunghezza massima del raggio, calcolata in pixel. Data una stima della dimensione e della forma dei cartelli che stiamo cercando, empiricamente è stato individuato che il valore ottimo per le immagini a disposizione è 350.

### 3.2.2 Filtro Triangolare

Il filtro per la ricerca di forme triangolari presenta una struttura più complessa del filtro precedente. Dopo aver importato l'immagine con la funzione “imread” di OpenCV, essa subisce una prima fase di trasformazione in cui viene privata dei tre canali di colori per passare ad una colorazione in scala di grigi. Questa operazione alleggerisce il calcolo computazionale successivo e viene svolta dalla funzione “cvtColor” di OpenCV. Successivamente essa viene processata e modificata sequenzialmente da due funzioni: la “dilate” e la “GaussianBlur”, entrambe appartenenti alla libreria di OpenCV. La prima funzione dilata l'immagine sorgente utilizzando l'elemento di strutturazione specificato, che nel nostro caso corrisponde ad un kernel quadrato di dimensione 4x4 riempito di valori “1”. Quest'ultimo determina il valore di ogni pixel come il valore massimo tra quelli dei vicini inclusi nel kernel. La funzione “GaussianBlur” invece applica una sfocatura usando un filtro gaussiano con un kernel di dimensione 5x5. Quest'ultima operazione attenua il rumore nell'immagine.Terminate poi le suddette operazioni, l'immagine ottenuta viene passata alla funzione “adaptiveThreshold” di OpenCV che la trasformerà in un'immagine binaria, rispetto

a come viene impostato il parametro `thresholdType`. Quest'ultimo è stato impostato a "THRESH\_BINARY\_INV" in quanto ha riportato il risultati migliori e presenta la seguente formula:

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > T(x, y) \\ maxValue & \text{otherwise} \end{cases}$$

Dove: `maxValue` è un parametro della funzione `adaptiveThreshold`;  $T(x,y)$  è una soglia calcolata individualmente per ogni pixel in base al valore del parametro "adaptiveMethod" della funzione in esame.

Occorre inoltre osservare che "maxValue" è stato impostato a 255 e il parametro "adaptiveMethod" è stato impostato a "ADAPTIVE\_THRESH\_GAUSSIAN\_C", pertanto  $T(x,y)$  è calcolato come la differenza tra una costante C e la somma pesata dell'intorno di ogni pixel in posizione  $(x,y)$  di dimensione `blockSize*blockSize`. Il parametro `blockSize` è fondamentale nella funzione `adaptiveThreshold` perché regola l'approccio di riconoscimento. Esso può assumere come valori solo numeri primi o loro potenze. Sono state valutate varie opzioni ed empiricamente i valori che hanno ottenuto i migliori risultati sono stati "1331", che corrisponde ad  $11^3$ , e "14641", che corrisponde ad  $11^4$ . La costante C, invece, è stata impostata a 2.

Terminata poi l'esecuzione di quest'ultima funzione, l'immagine binaria viene passata alla funzione "findContours" di OpenCV, la quale determina, per l'appunto, i contorni degli oggetti presenti nell'immagine. Gli ulteriori parametri di questa funzione sono stati impostati come segue:

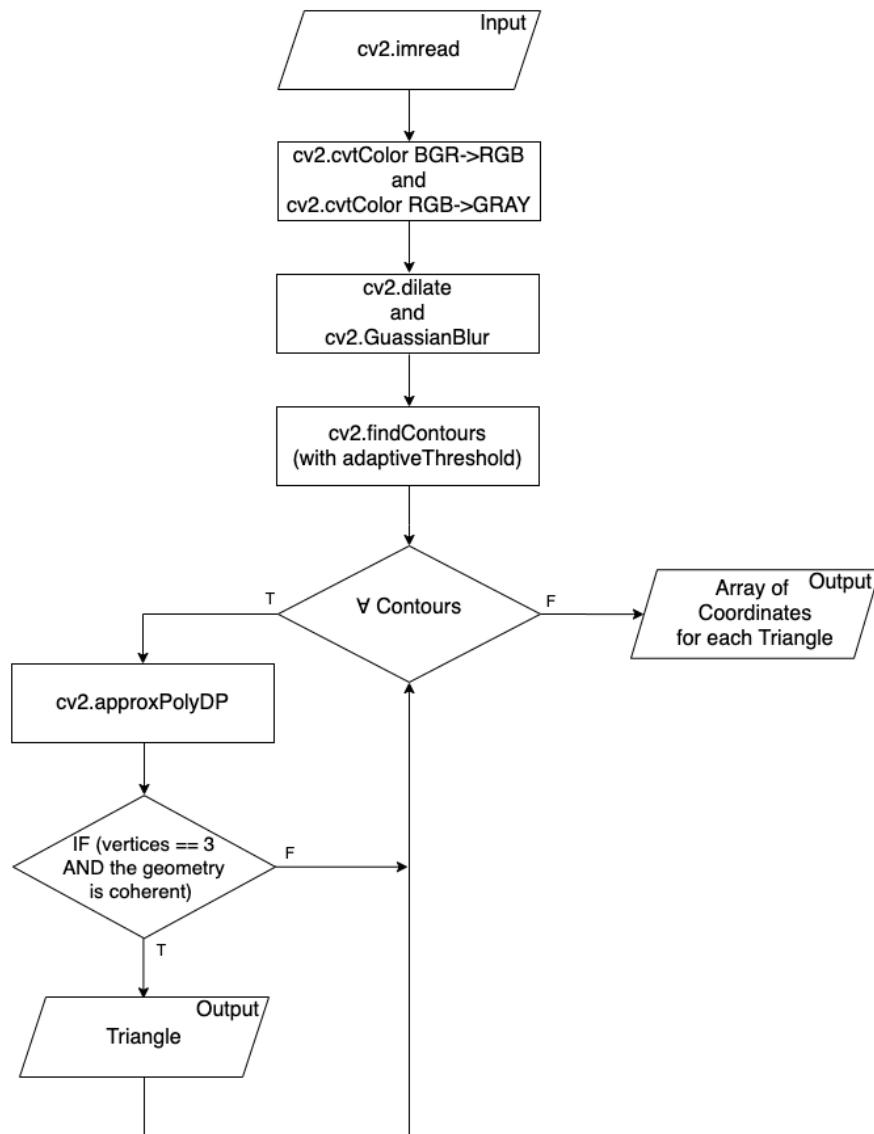
- `header_size`: questo parametro viene impostato come il risultato riportato dalla funzione "adaptiveThreshold";
- `mode`: esprime la modalità di recupero del contorno. È stato impostato a "CV\_RETR\_LIST" cosicché vengano recuperati tutti i contorni senza stabilire relazioni gerarchiche;
- `method`: corrisponde al metodo di approssimazione del contorno. È stato impostato a "CV\_CHAIN\_APPROX\_SIMPLE" cosicché vengano compressi i segmenti orizzontali, verticali e diagonali, lasciando solo i loro punti finali. Ad esempio, un contorno triangolare è codificato con 3 punti.

Ottenuto quindi un array di contorni, per ognuno di essi viene applicata la funzione "approxPolyDP" di OpenCV, la quale approssima una curva o un poligono con un'altra curva o poligono con meno vertici, in modo che la distanza tra essi sia minore o uguale alla precisione specificata, e restituisce quindi tale approssimazione. Per svolgere tale compito, la funzione utilizza l'algoritmo Douglas-Peucker. I parametri di tali funzioni sono stati impostati come segue:

- `epsilon`: corrisponde all'accuratezza dell'approssimazione. Questa è la distanza massima tra la curva originale e la sua approssimazione. È stata impostata a "0,07\*opencv.arcLength(contorno, True)", dove la funzione "arcLength" calcola una lunghezza della curva o un perimetro di contorno chiuso;

- closed: valore boolean che se viene impostato a "True" corrisponde a una curva approssimata chiusa, cioè il suo primo e l'ultimo vertice sono collegati. Se invece è impostato a falso, la curva non è chiusa. Nell'attuale caso noi stiamo cercando forme geometriche chiuse, quindi è stata impostata a True.

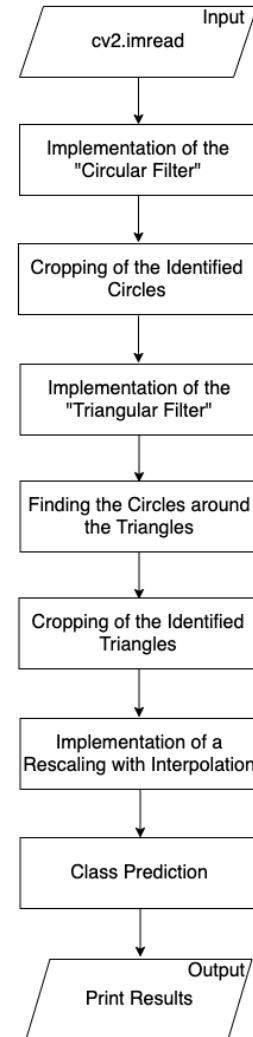
Analizzando, quindi, l'approssimazione restituita, se quest'ultima presenta solo 3 vertici, allora può essere considerato un triangolo, altrimenti viene scartata. In realtà sono stati eseguiti ulteriori controlli per stabilire con maggior certezza se l'approssimazione restituita sia un triangolo o meno, ma questi ultimi verranno illustrati successivamente. Infine è possibile visionare una rappresentazione grafica della pipeline del filtro triangolare appena illustrato:



### 3.3 Pipeline dell'Object Detection and Recognition

Allo scopo di rilevare e riconoscere i cartelli stradali partendo da un'immagine non pre-processata, è stata sviluppata una pipeline apposita di "Object Detection e Recognition". Quest'ultima espande le potenzialità dell'algoritmo di Classification coniugandolo con i filtri geometrici precedentemente esposti.

Dopo aver ottenuto l'immagine tramite la funzione "imread" di OpenCV, la suddetta pipeline ne crea una copia e applica a quest'ultima il filtro circolare precedentemente esposto. Se sono state rilevate forme circolari, tale filtro restituisce un array di coordinate dei cerchi, ognuno dei quali è caratterizzato dal centro  $(x, y)$  e dal raggio  $r$ . Tramite tali coordinate vengono eseguiti e salvati tanti crop quanti sono i cerchi individuati. Questi ultimi circoscrivono le rilevazioni e si presentano di forma quadrata, i cui vertici sono computati come segue:  $(x - r, y - r, x + r, y + r)$ . Successivamente viene applicato il filtro triangolare ad un'ulteriore copia dell'immagine inizialmente importata. In questo secondo caso il filtro restituisce un array di coordinate dei triangoli, ognuno dei quali è caratterizzato dalle tre coppie di valori che individuano i vertici di ogni triangolo:  $[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$ . Per svolgere il crop di tali triangoli in maniera omologa al caso precedente l'array delle coordinate viene prima passato ad una funzione che ritorna le coordinate dei cerchi che circoscrivono ogni triangolo e poi, solo successivamente, si esegue il medesimo crop del caso precedente. Terminata quindi questa prima fase, a tutti i crop ottenuti viene applicato un ridimensionamento (cv2.resize) con interpolazione cubica (cv2.INTER\_CUBIC), che per l'appunto ridimensiona tutte le immagini portandole alla stessa estensione grafica richiesta in input dalla rete già pre-addestrata, cioè 32x32 pixels. Infine l'ultimo step è la classificazione dei possibili cartelli stradali con la relativa stampa della predizione ottenute.



# Capitolo 4

## Risultati

In questo capitolo verranno illustrati gli step svolti e i risultati ottenuti per raggiungere l’obiettivo preposto. Le analisi svolte si possono distinguere in:

- Addestramento e ottimizzazione della PersonalCNN, in maniera tale da avere il più alto livello di accuracy possibile per quanto riguarda la classification dei cartelli stradali;
- Analisi e miglioramento del rilevamento (Detection) dei cartelli stradali, con particolare attenzione ai filtri geometrici e all’ottimizzazione della fase di rilevamento;
- Valutazione dei risultati finali ottenuti dalla pipeline completa.

### 4.1 Addestramento e Ottimizzazione della PersonalCNN

La PersonalCNN è una rete convoluzionale molto leggera, pertanto è in grado di essere addestrata abbastanza velocemente sia in GPU che in CPU. Ovviamente le due modalità presentano tempistiche di esecuzione differenti: con il numero di nodi degli strati convoluzionali impostati a 32 nodi, la GPU ha richiesto circa un secondo ad epoca, mentre in CPU all’incirca 20-30 secondi. Alzando i nodi a 64, la GPU è rimasta sugli 1-2 secondi, mentre la CPU a circa 40-50 secondi. Per quanto le tempistiche ottenute con la CPU non siano eccessivamente elevate, i risultati hanno indotto ad addestrare la rete principalmente con la GPU (NVIDIA).

Invece, per quanto riguarda la dimensione del batch di addestramento, quest’ultima è stata impostata a 100. Tale scelta ha comportato le migliori prestazioni computazionali e temporali. Inoltre è presente una seconda batch size che regola il numero di samples letti in input e raggruppati dalla funzione ImageDataGenerator di Keras. Per una più semplice acquisizione e manipolazione del dataset nella forma  $\langle X, y \rangle$ , essa è stata impostata pari al numero di samples per ogni subset di immagini (train set, validation set e test set). In aggiunta, per quanto riguarda il processo di ottimizzazione, esso ha seguito i seguenti step:

1. Ricerca del numero ottimo di nodi per i layers;
2. Analisi dell’early stopping;

3. Confronto tra le funzioni di attivazione "ReLU" e "LeakyReLU";
4. Valutazione dell'addestramento al variare della presenza o meno dell'augmentation sulle immagini;
5. Valutazione generale delle prestazioni raggiunte dalla rete.

Occorre precisare, infine, che il primo test è stato svolto con la seguente configurazione iniziale della rete:

1. Epoche Massime = 10 ;
2. EarlyStopping = NO;
3. Numero di nodi dei layers convoluzionale = 128;
4. Numero di nodi del penultimo layer Dense = 1024.

#### 4.1.1 Ricerca del Numero Ottimale di Nodi per i Layers

Nel primo step di ottimizzazione sono stati valutati diversi valori per i layers convoluzionali della rete. I risultati ottenuti sono i seguenti:

NUM. NODI	EPOCHE	ACCURACY	LOSS	TEMPO
32	10	0.9837	0.0653	02:10
64	10	0.9842	0.0619	03:27
128	10	0.9737	0.110	06:15

Osservando i report sopra riportati, è possibile notare che con l'aumento del numero di nodi provoca sia un deterioramento prestazionale che un peggioramento dei livelli di accuracy e di loss. Inoltre, confrontando i casi in cui nodi sono settati a 32 e a 64, possiamo notare come la loss e l'accuracy siano praticamente identiche. Le variazioni di millesimi tra i due casi non hanno rilevanza, in quanto ogni addestramento porterà sicuramente a oscillazioni che non permettono di determinare una casistica migliore tra le due. Osservando però le tempistiche d'addestramento, possiamo notare come il caso con 32 nodi sia di circa il 37% più veloce rispetto al caso con 64 nodi. Risulta evidente quindi come la rete con 32 nodi sia la più efficiente, coniugando soddisfacentemente le prestazioni computazionali e temporali desiderate.

In aggiunta a questo primo test, poi, è stata svolta una seconda analisi che si è concentrata nell'individuare il numero ottimo di nodi per il penultimo layer fully-connected. Sono stati testati dei valori molto più alti rispetto al test precedente, essendo consigliabile eseguire una discesa graduale dai molteplici risultati dei layers convoluzionali verso la fine della rete. I risultati sono i seguenti:

NUM. NODI	EPOCHE	ACCURACY	LOSS	TEMPO
256	10	0.9698	0.1140	02:10
512	10	0.9799	0.1009	03:27
1024	10	0.9783	0.0914	05:15

Da tali responsi si può osservare come non ci sono variazioni evidenti tra le casistiche in esame. L'ultima osservazione degna di nota riguarda il penultimo layer Dense costituito da 512 nodi in cui si sono raggiunte le stesse performance del livello composto da 1024 nodi, ma con un risparmio temporale e computazionale che, anche se lieve, risulta significativo e vantaggioso per accelerare l'esecuzione dei futuri test da effettuare. Pertanto il penultimo layer sarà impostato a 512 nodi.

#### 4.1.2 Analisi dell'Early Stopping

L'Early Stopping è una tecnica di regolarizzazione utilizzata per evitare l'overfitting dell'addestramento tramite l'analisi della "val\_loss". L'overfitting si verifica quando l'addestramento dura troppo a lungo e il modello, per via della scarsità di samples, si adatta a delle caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei test. La val\_loss, per l'appunto, calcola la loss al termine di ogni epoca in base all'applicazione dell'addestramento ottenuto sul validation set. Quest'ultimo non viene usato durante il training e quando vi sono eccessivi e consecutivi aumenti della val\_loss, è spesso indice di overfitting, e infatti applicare tale addestramento al validation set porta a un calo prestazionale. L'analisi dell'Early Stopping, quindi, si rende necessaria per comprendere se quest'ultimo porta dei benefici o meno alla rete. Per testare questa funzionalità la rete è stata impostata con 32 nodi nei layers convoluzionali e 512 nodi nel penultimo layer fully-connected. I risultati ottenuti hanno riportato i seguenti valori:

EPOCHE	EARLY STOPPING	ACCURACY	LOSS	TEMPO
10	NO (10 svolte realmente)	0.9620	0.1665	03:24
10	SI	0.9799	0.1009	03:27
15	NO	0.9757	0.0982	03:14
15	SI (12 svolte realmente)	0.9800	0.0816	02:51

I test sono stati svolti sia con 10 che con 15 epoche in quanto molto spesso l'addestramento a 10 epoche non veniva interrotto dall'early stopping, pertanto si è deciso di aumentare il numero di epoche massime per permettere all'early stopping di valutare il maggior numero di epoche possibili prima di interrompere l'addestramento quando necessario. Dai risultati riportati è evidente come una rete di così piccole dimensioni non sfrutta a pieno tutti i benefici che normalmente sono introdotti con l'early stopping. La precisione acquisita, infatti, non varia di molto con il suo utilizzo, ma si è ritenuto appropriato adottare tale tecnica sia per ridurre la durata degli esperimenti, diminuendo di conseguenza anche il relativo costo computazionale, e sia come forma di prevenzione contro la degradazione delle prestazioni.

#### 4.1.3 Confronto tra le funzioni di attivazione "ReLU" e "LeakyReLU"

Un terzo test effettuato ha coinvolto le funzioni di attivazione dei tre livelli convoluzionali e del penultimo layer di tipo Dense che compongono la rete. Finora è sempre stata utilizzata la funzione di attivazione ReLU (Rectified Linear Unit). In questo test verrà svolto il confronto tra quest'ultima e la funzione di attivazione

LeakyReLU. Essa, a differenza delle ReLU, non comporta la disattivazione completa dei nodi, e quindi permette loro di addestrarsi nella speranza che successivamente possano portare beneficio alla rete. I risultati ottenuti sono i seguenti:

FUNZ. DI ATTIVAZIONE	EPOCHE	ACCURACY	LOSS	TEMPO
ReLU	15	0.9800	0.0816	02:51
LeakyReLU	15	0.9824	0.0785	02:17

Negli esiti dei test ottenuti non c'è una differenza significativa tra i due per poter stabilire quale funzione di attivazione sia effettivamente la migliore. Di conseguenza si è scelto di adottare entrambe le funzioni nella prossima fase di test che coinvolge l'augmentation, al fine di effettuare una scelta discriminante solo nel seguito dell'esperimento, quando per l'appunto ci saranno più risultati su cui valutare la miglior scelta.

#### 4.1.4 Analisi del Pre-Processing per il Training Dataset

In quest'ultimo test verranno valutate sia le casistiche in cui l'augmentation è presente nel train set e/o nel test set e sia i risultati al variare della funzione di attivazione:

FUNZ. DI ATTIV.	TRAIN AG.	TEST AG.	ACCUR.	LOSS	TEMPO
ReLU	No	No	0.9800	0.0816	02:40
LeakyReLU	No	No	0.9824	0.0685	02:17
ReLU	Si	No	0.9730	0.0997	02:54
LeakyReLU	Si	No	0.9761	0.0904	03:19
ReLU	No	Si	0.6926	2.4078	02:00
LeakyReLU	No	Si	0.6860	2.1190	01:23
ReLU	Si	Si	0.9430	0.1952	01:38
LeakyReLU	Si	Si	0.9579	0.1608	02:51

I valori qui riportati mostrano, come prevedibile, un lieve decremento prestazionale dovuto all'augmentation: cartelli stradali modificati sono più difficili da apprendere e riconoscere. Tuttavia, si ritiene necessario effettuare tale fase di pre-processing in quanto è estremamente utile per migliorare le prestazioni della rete in casi più realistici e prossimi ad un utilizzo pratico. Se così non fosse, ci si ricondurrebbe al caso analizzato nella precedente tabella in cui l'augmentation è effettuata solo sul test set: in questa casistica appare immediato come la rete raggiunga risultati mediocri, in quanto in fase di addestramento non ha mai osservato e appreso i cartelli diversi da quelli ideali. Inoltre il test ha mostrato come la funzione di attivazione LeakyReLU, su cui prima vi era indecisione, sia preferibile ad una normale ReLU, in quanto raggiunge, seppur di poco, risultati più soddisfacenti.

#### 4.1.5 Valutazione Finale dei Risultati Ottenuti

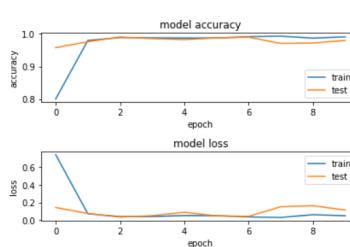
Dopo aver svolto tali ottimizzazioni, sono stati analizzati i risultati ottenuti principalmente in due modalità: la prima si concentra sul valutare i miglioramenti generali

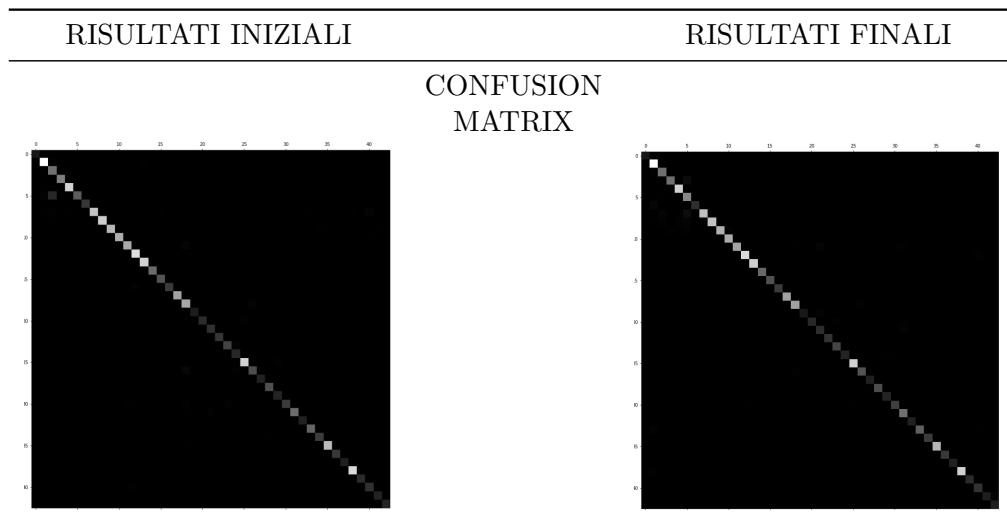
ottenuti dalla rete durante l'addestramento, analizzando gli andamenti e le statistiche per comprendere se si è usufruito dei benefici portati dalle scelte fatte durante i precedenti test; la seconda analisi si concentrerà sull'esaminare specificatamente dei sottoinsiemi di cartelli stradali nel caso di classificazioni corrette ed errate, al fine di comprendere il motivo per il quale la rete compie ancora degli errori.

### 1) Analisi Dei Miglioramenti Generali:

Prima di confrontare i risultati ottenuti in questa prima analisi, occorre aggiungere una precisazione: la configurazione iniziale non presenta l'augmentation del training dataset, mentre nella configurazione finale è presente. Il problema risiede nelle prestazioni della rete, le quali diminuiscono considerevolmente quando vi è augmentation del training dataset, anche se quest'ultima permette una maggior efficienza in casistiche più reali. Pertanto, al fine di svolgere un confronto più equo, verranno analizzati i risultati ottenuti tra la configurazione iniziale e finale prima nel caso in cui il training dataset non abbia subito augmentation e poi nel caso in cui sia stata applicata al training dataset.

Di seguito vi sono i risultati del primo caso:

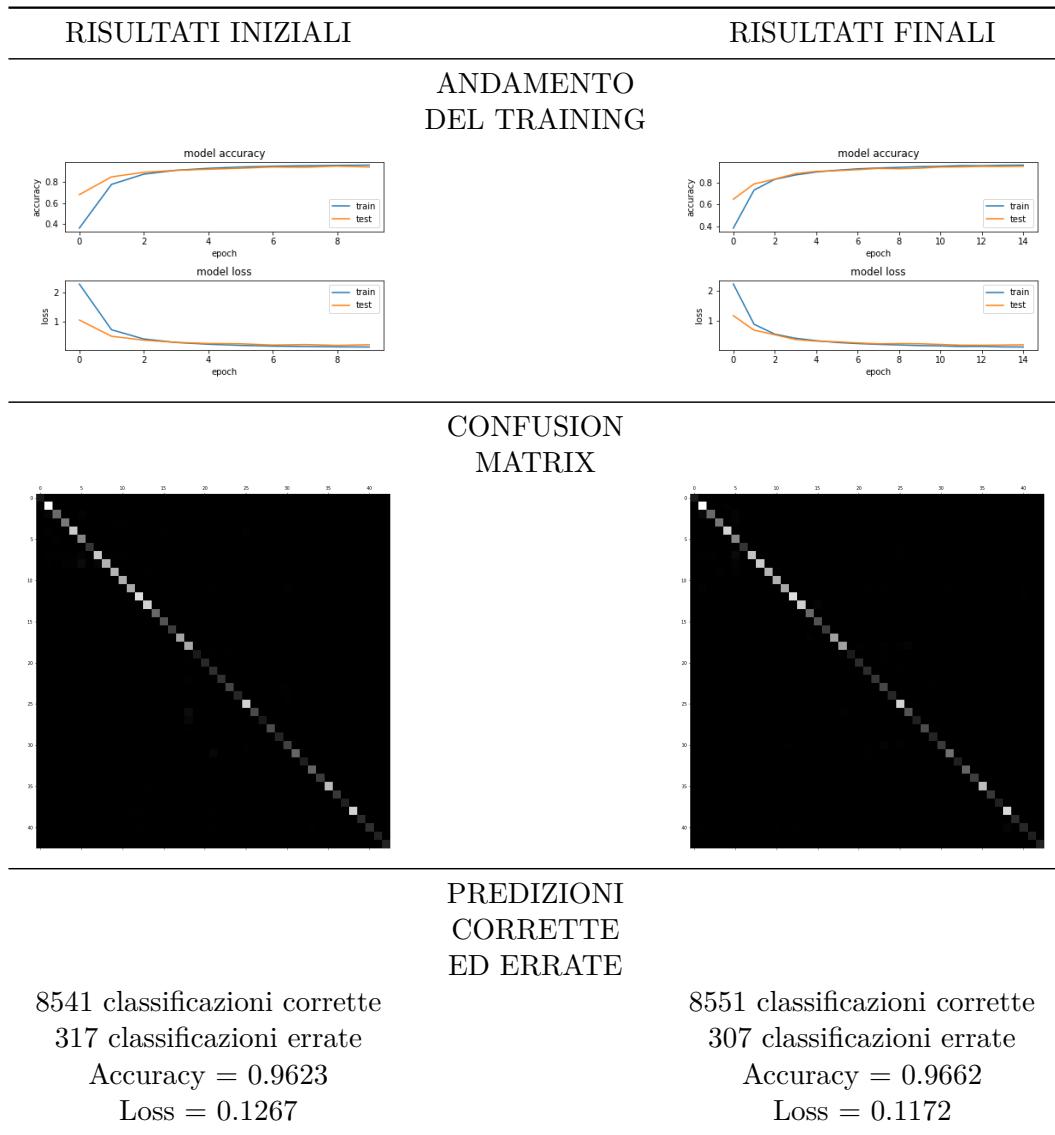
RISULTATI INIZIALI	SETTAGGI	RISULTATI FINALI
10 Epoche Massime; EarlyStopping = NO; nodi dei layers convoluzionali = 128; nodi del layer Dense = 1024; Train/Test Aug. = NO;	SETTAGGI	15 Epoche Massime; EarlyStopping = SI; nodi dei layers convoluzionali = 32; nodi del layer Dense = 512; Train/Test Aug. = NO;
8581 classificazioni corrette 277 classificazioni errate Accuracy = 0.9737 Loss = 0.110	PREDIZIONI CORRETTE ED ERRATE	8585 classificazioni corrette 273 classificazioni errate Accuracy = 0.9824 Loss = 0.0685
ANDAMENTO DEL TRAINING		
		



I report sopra riportati mostrano come la PersonalCNN sia sempre stata prossima all'ottimo, in quanto ha sempre raggiunto un'accuracy tra il 95% e il 99%, riportato al contempo dei bassi livelli di loss. Questa precisione nella classificazione è ulteriormente osservabile anche dalle confusion matrices sopra riportate, dove solo pochissimi samples sono classificati erroneamente. Tuttavia, analizzando i grafici dell'andamento del training, appare chiaro come il processo di ottimizzazione della rete abbia permesso lo sviluppo di un addestramento molto più regolare rispetto alla situazione iniziale. In quest'ultima casistica l'andamento era caratterizzato da lunghi tratti costanti simili a plateau, ad accentuare lo spreco di risorse della rete in quelle zone, e al contempo vi erano dei lievi innalzamenti della val\_loss, ad indicare l'overtiffing e il peggioramento delle prestazioni della rete con il proseguire dell'addestramento. Invece, nel caso che rappresenta i risultati ottenuti al termine dei test, l'andamento è risultato estremamente più regolare, testimoniato dalla monotonicità crescente dell'accuracy e dalla scomparsa dei peggioramenti della val\_loss.

Pertanto, osservando i risultati ottenuti finora, le migliori applicate sembrano aver portato beneficio alla rete, ma per trarre una conclusione definita è necessario analizzare anche il caso più realistico in cui è stata applicata l'augmentation sia al train set che al test set. Di seguito sono riportati i risultati degli addestramenti con la presenza dell'augmentation nel train set e nel test set in entrambe le configurazioni:

RISULTATI INIZIALI	RISULTATI FINALI
SETTAGGI	
10 Epoche Massime; EarlyStopping = NO; nodi dei layers convoluzionali = 128; nodi del layer Dense = 1024; Train/Test Aug. = SI;	15 Epoche Massime; EarlyStopping = SI; nodi dei layers convoluzionali = 32; nodi del layer Dense = 512; Train/Test Aug. = SI;



In quest'ultimo caso non si possono notare miglioramenti tangibili. L'unica osservazione degna di nota riguarda le prestazioni temporali: la configurazione iniziale ha richiesto circa 3 volte il tempo di addestramento rispetto alla configurazione finale. Nel caso di addestramento tramite l'ausilio di una GPU, questa differenza non ha gravato molto sulle prestazioni, ma riproponendo i test su CPU i tempi richiesti sono migliorati del 282% circa, passando da 26 minuti e 34 secondi per la configurazione iniziale a 6 minuti e 56 secondi per la configurazione finale. Osservando quindi tali risultati, anche in questo caso è consigliabile l'utilizzo della configurazione finale per alleggerire il carico computazionale, mantenendo al contempo risultati ottimali. Infine, quindi, è possibile concludere come le scelte svolte portino benefici diversi rispetto alle differenti casistiche prese in esame, ma che in generale queste ultime ottengono sempre un'ottimizzazione delle prestazioni, pertanto tali scelte verranno mantenute per disporre dei risultati migliori.

## 2) Analisi delle Classificazioni Corrette ed Errate:

L'analisi di alcuni esempi di classificazioni corrette ed errate è necessaria per ipotizzare e cercare di comprendere come mai le predizioni non producono sempre una classificazione corretta. Dato l'intento di essere il più possibile legati a casi realistici, verranno analizzate nove predizioni corrette e nove predizioni errate ottenute nel caso in cui l'augmentation è stata applicata sia al train set che al test set:

### CLASSIFICAZIONI:

8551 Corrette

307 Errate

### CONVERSIONE DELLE DIGITS:

'1' <=> 'Speed limit (30km/h)'	'14' <=> 'Stop'
'4' <=> 'Speed limit (70km/h)'	'15' <=> 'No vechiles'
'5' <=> 'Speed limit (80km/h)'	'16' <=> 'Vehicles over 3.5 metric tons prohibited'
'6' <=> 'End of speed limit (80km/h)'	'20' <=> 'Dangerous curve to the right'
'7' <=> 'Speed limit (100km/h)'	'23' <=> 'Slippery road'
'9' <=> 'No passing'	'30' <=> 'Beware of ice/snow'
'10' <=> 'No passing for vehicles over 3.5 metric tons'	'33' <=> 'Turn right ahead'
'11' <=> 'Right-of-way at the next intersection'	'35' <=> 'Ahead only'
'13' <=> 'Yield'	'38' <=> 'Keep right'
	'40' <=> 'Roundabout mandatory'



Come si può notare dalle predizioni errate, la maggior parte di queste ultime possono essere considerate accettabili in quanto gli errori derivano da cartelli eccessivamente distorti e difficilmente distinguibili anche per un essere umano. In particolare, alcune presentano un'eccessiva sovra-illuminazione, che rende rarefatta la forma distintiva del cartello, come nel caso del cartello in [riga=3, colonna=2], ma in realtà nella maggior parte dei casi sembrano presentarsi dei problemi dovuti ad un'eccessiva sotto-illuminazione. Quest'ultimo caso corrisponde a delle situazioni reali in cui le

strade non sono ben illuminate e i fari delle automobili non sono in grado di fornire sufficiente luce per illuminare i cartelli. Di particolare rilevanza poi sono le digits che corrispondono agli speed limits. Nei cartelli in questione in [riga=1, colonna=3] e [riga=3, colonna=3] possiamo osservare come l'errore compiuto ha portato al riconoscimento della digit 5 al posto della 7, la quale altro non è che un cartello simile, ma con limite di velocità diverso. Questo induce a pensare che quantomeno la "categoria" del cartello è stata individuata correttamente, e cioè che vi è stato un buon addestramento di fondo. Infine, tra gli esempi riportati, quello che tra tutti è il più preoccupante è il cartello in posizione [riga=1, colonna=1], dove non vi è eccessiva distorsione, ma la classe non è stata comunque individuata correttamente. Confrontando inoltre le densità tra i subsets di una classe predetta correttamente e i subsets di una classe predetta erroneamente (Classe Predetta Correttamente: Yield — train=2994, valid=528, test=399; Classe Predetta Erroneamente: No vehicles — train=862, valid=151, test=158) possiamo osservare come la classe Yield presenti uno dei più alti numeri di samples per classe, mentre nel caso della classe "No vehicles" i samples non sono altrettanto numerosi. Si può quindi ipotizzare che per risolvere il problema sia necessario fornire più samples alle classi che presentano un minor numero totale di elementi, così da permettere alla rete di addestrarsi con più accuratezza anche con le classi che attualmente sono più povere di samples.

In conclusione, date le ottime prestazioni ottenute dalla PersonalCNN, i risultati ottenuti possono ritenersi soddisfacenti per utilizzare la rete in questione nella pipeline completa di rilevazione e riconoscimento dei cartelli stradali.

## 4.2 Analisi e Miglioramento della Pipeline Completa

Per poter migliorare l'efficienza della pipeline nella sua totalità è stata svolta inoltre un'analisi e un'ottimizzazione della suddetta tramite un approccio top-down. Data l'influenza di ogni fase di ottimizzazione sulle successive, anche l'analisi dei test svolti e dei risultati ottenuti seguirà lo stesso approccio.

### 4.2.1 Ottimizzazione del Rilevamento dei Cartelli Circolari

Il primo step a cui viene sottoposta l'immagine non pre-processata è il filtro circolare. I parametri delle funzioni utilizzate da tale filtro sono stati ottimizzati per individuare cartelli che non siano né eccessivamente vicini e né eccessivamente lontani, come già precedentemente esposto. Il filtro circolare però non riporta sempre rilevazioni ottimali. Sono usuali, infatti, dei casi in cui il suddetto filtro individua forme circolare annidate l'una dentro l'altra, non centrate nel cartello individuato oppure un numero eccessivo di cerchi. Per ovviare a tali problematiche si è deciso di utilizzare dei parametri più restrittivi per le funzioni che individuano forme circolare, ma non è stato possibile eliminare il problema in quanto alcune immagini richiedono valori specifici che non sono ottimi per la maggior parte degli altri casi in esame. Pertanto si è deciso di utilizzare dei valori che sono moderatamente restrittivi, ma che riportino dei risultati ottimi nella maggior parte delle immagini. Un esempio di tali problematiche è riportato nelle immagini successive:



Figura 4.1. Eccessive rilevazioni



Figura 4.2. Rilevazioni non centrate

Data l'impossibilità di centrare con assoluta precisione la maggior parte delle forme circolari individuate si è deciso di applicare un approccio che prevede di ritagliare diverse porzioni dell'immagine per ogni rilevazione. Ad ogni rilevazione vengono eseguiti tre ritagli di forma quadrata i cui vertici presentano le seguenti coordinate:  $(x - r, y - r, x + r, y + r)$ ,  $(x - r/2, y - r/2, x + r/2, y + r/2)$  e  $(x - 2 * r, y - 2 * r, x + 2 * r, y + 2 * r)$ . In questo modo, se la rilevazione iniziale è all'interno del reale cerchio d'interesse, verrà poi analizzata anche una versione del quadrato che sia il doppio più grande e che probabilmente comprenderà tutto il cartello circolare. Analogamente, se la rilevazione iniziale circoscrivesse eccessivamente il cerchio, verrà poi analizzata anche una versione del quadrato che sia la metà dell'originale e che probabilmente comprenderà specificatamente il cartello d'interesse. Inoltre occorre precisare che questi ulteriori crop vengono svolti se e solo se non fuoriescono dai margini dell'immagine originale. Questo controllo è effettuato perché il filtro individua forme circolari chiuse, quindi se il crop fuoriesce dall'immagine sicuramente non si sta focalizzando su di un cerchio. Infine un esempio dell'utilità di tale tecnica è visibile nella Figura 4.3 dove possiamo analizzare i risultati di classificazione della Figura 4.2 precedentemente esposta:

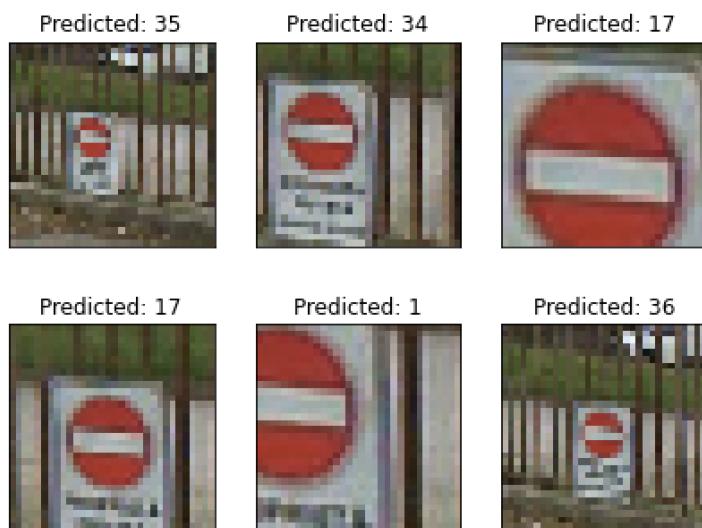


Figura 4.3. Esempio sulla classificazioni di crop multipli

### CONVERSIONE DELLE DIGITS:

'17' <=> 'No entry'  
'22' <=> 'Bumpy road'  
'34' <=> 'Turn left ahead'  
'35' <=> 'Ahead only'  
'36' <=> 'Go straight or right'

In primis occorre specificare che le due rilevazioni iniziali corrispondono alle immagini in [riga=1, colonna=2] e [riga=2, colonna=3]. Una delle due viene correttamente identificata, in quanto presenta il cartello all'incirca al centro della figura, ma l'altra no. Per predire correttamente questo secondo caso può tornare utile il crop di dimensione dimezzata rispetto all'originale, e infatti si può osservare che in quest'ultimo caso una classificazione viene svolta correttamente. Il secondo crop di dimensione dimezzata invece non riporta il valore corretto perché il cartello in sé è eccessivamente ritagliato. Per ovviare a questo secondo tipo di errore, la rete andrebbe addestrata su più casi in cui i cartelli sono ritagliati, ma data la scarsità di samples nel dataset usato si è deciso di non forzare eccessivamente questo aspetto durante l'addestramento. Pertanto, date tali considerazioni, l'errore ottenuto in questo caso è accettabile. Per quel che riguarda invece i crop svolti su un quadrato di dimensione doppia rispetto all'originale, in questo specifico caso non ha portato un beneficio alla classificazione perché ci si allontana ancora di più dal cartello d'interesse, ma possono sempre capitare casistiche in cui ampliare il campo d'interesse è utile tanto quanto in questo caso è stato utile restringerlo.

Infine, per risolvere la problematica delle eccessive rilevazioni si è deciso di imporre un numero massimo di rilevazioni accettabili. Considerando che per ogni rilevazione possono essere svolti fino a 3 crop e che quello di dimensione massima spesso non viene effettuato dato che i cartelli sono posti ai margini delle rilevazioni, il numero limite di ritagli accettati è 70. Se si supera tale valore tutti i crop effettuati non saranno considerati. Si è deciso di applicare questo filtraggio perché è altamente probabile che, se le rilevazioni sono eccessive, allora i parametri del filtro circolare non sono adeguati all'analisi della particolare immagine in esame e, pertanto, è più saggio passare direttamente al filtro triangolare.

#### **4.2.2 Ottimizzazione del Rilevamento dei Cartelli Triangolari**

Il filtro triangolare presenta una struttura ben più complessa rispetto al filtro circolare, pertanto è altrettanto complesso da ottimizzare. Alcuni parametri espressi nei capitoli precedenti sono individuati empiricamente dopo una serie di svariati test, mentre altri sono stati scelti in quanto consigliati dalla documentazione disponibile. Purtroppo, però, anche se sufficientemente ottimi, tali parametri non hanno risolto un problema importante: il filtro non solo rileva nell'immagine non pre-processata forme triangolari che coincidono con dei cartelli stradali, ma individua anche vari insiemi di poligoni a 3 vertici che non corrispondono con questi ultimi. Un esempio è riportato nella figura sottostante:



**Figura 4.4.** Esempio sull'imprecisione del filtro triangolare

Al fine di risolvere questo problema, è necessario eseguire una selezione per individuare con maggior precisione i triangoli. È stato quindi inserito un controllo che calcola la distanza tra i vertici di ogni rilevazione e la considera valida se e solo se: ogni lato è lungo almeno 20 pixel; se il rapporto tra tutti i lati con gli altri disponibili è compreso da 0.8 e 1.2. Questa tecnica elimina esaurientemente le rilevazioni indesiderate. In aggiunta, per eliminare le ridondanze di triangoli concentrici, è stato inserito l'ulteriore controllo che non effettua il salvataggio dei triangoli che hanno ciascuno dei vertici a meno di 20 pixels dai vertici delle rilevazioni già salvate. Il risultato ottenuto è il seguente:



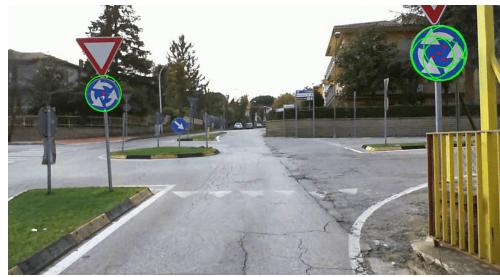
**Figura 4.5.** Esempio del filtro triangolare ottimizzato

Come si evince dalla figura sovrastante, il risultato ottenuto è ottimo già solo con questo primo filtraggio e pertanto gli accorgimenti per eseguire crop multipli su una stessa rilevazione non sono necessari. L'unico espediente utilizzato nel filtro precedente e ripreso anche in quest'ultimo è la possibilità di non salvare le rilevazioni quando ne vengono individuate troppe. Come nel caso precedente, eccessive rilevazioni potrebbero essere il segno che i parametri generici scelti per le funzioni del filtro non sono adeguati all'immagine. Pertanto, se vengono rilevati più di 15

triangoli, non ne verrà valutato nessuno.

#### 4.2.3 Analisi dei Canali di Colore

Al fine di distinguere con maggior precisione le false rilevazioni o i crop che non contengono un cartello stradale, si è tentato un approccio di analisi delle immagini attraverso l'istogramma che rappresenta il numero di pixels per ogni livello d'intensità e per ogni colorazione. L'idea alla base di questo approccio è che la maggior parte dei cartelli presenta 3 colori principali: rosso, blu e bianco. Analizzando quindi il suddetto istogramma, se l'immagine contiene per lo più solo il cartello stradale, è possibile individuare un picco nel canale di colore che maggiormente caratterizza il cartello, mentre negli altri canali è possibile individuare dei valori più tenui e omogenei. In tale modo si potrebbero distinguere le immagini che contengono oggetti terzi e che quindi presentano meno picchi e colori più omogenei. Pertanto le prime sperimentazioni di questa tecnica sono state effettuate sulla prossima immagine dove i filtri hanno svolto un buon lavoro nell'individuazione dei cartelli stradali:

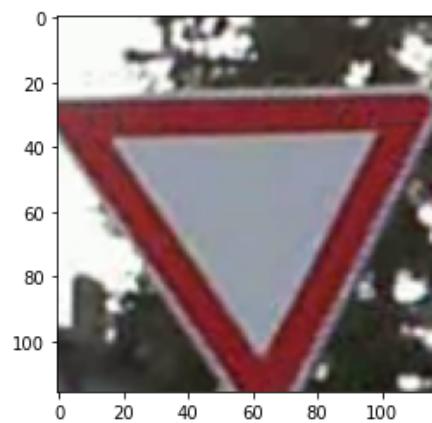
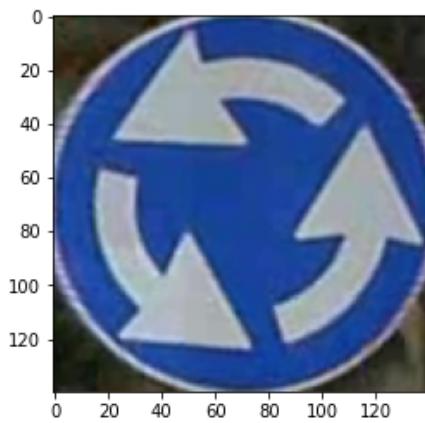


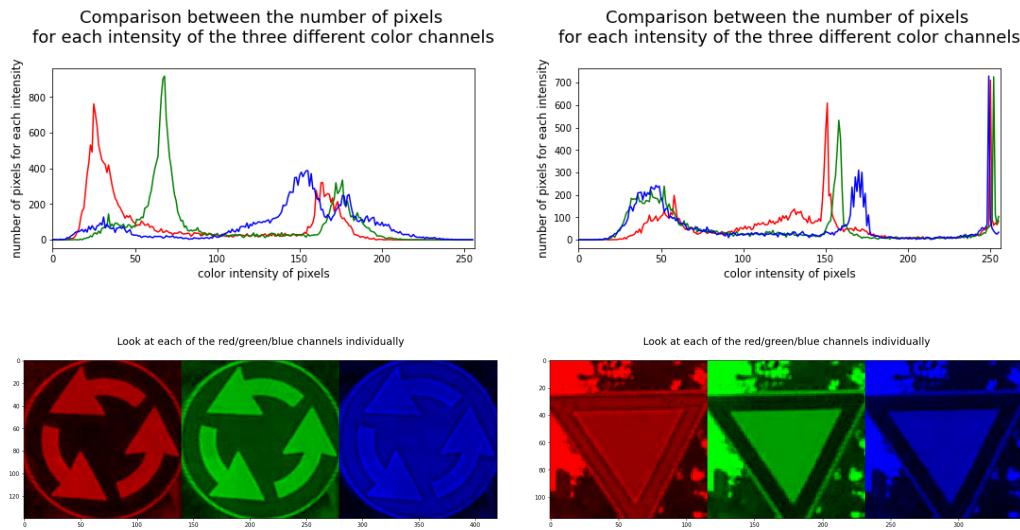
**Figura 4.6.** Rilevamento dei Cerchi



**Figura 4.7.** Rilevamento dei Triangoli

Prendendo quindi in esame il crop di un cerchio e il crop di un triangolo che sono ben visibili e centrati, i rispettivi istogrammi sono i seguenti:





I report sopra riportati non mostrano una prevalenza evidente dei colori più caratteristici. Per assurdo, nel caso del cartello dove vi è il simbolo di una rotonda per lo più blu, sono visibili alti livelli di rosso e verde, mentre il colore blu ha un picco di circa la metà rispetto agli altri canali di colore. Per quanto riguarda invece il cartello triangolare, il picco più alto è il rosso, ma è comunque presente un picco di colore verde che ha quasi la stessa intensità e quasi lo stesso numero di pixels del precedente colore. Questi risultati sono confermati anche dall'analisi delle immagini in cui è presente solo un canale di colore alla volta. Inoltre, per confermare con maggior certezza il risultato ottenuto, sono state analizzate altre immagini a campione e tutte hanno riportato istogrammi simili. I picchi inaspettati per ogni immagine sono indice del fatto che i colori distintivi dei cartelli non sono ottenuti con il solo utilizzo del colore primario d'interesse, ma con l'ausilio di tutte e tre i canali. Questo è un grande problema e non permette l'utilizzo di questa tecnica per identificare con maggior precisione i crop contenenti i cartelli, pertanto non è stato più utilizzato.

### 4.3 Valutazione e Conferma dei Risultati Finali

Infine l'ultima analisi riguarda la valutazione dei risultati ottenuti dalla pipeline di rilevamento e riconoscimento dei cartelli. Riepilogando i parametri ottimizzati, in questa fase l'addestramento della rete è stato eseguito con le seguenti impostazioni:

- 15 Epoche Massime;
- EarlyStopping = SI;
- Nodi dei layers convoluzionali = 32;
- Nodi del layer Dense = 512;
- Train/Test Aug. = SI.

Mentre ai filtri sono stati applicati i seguenti accorgimenti:

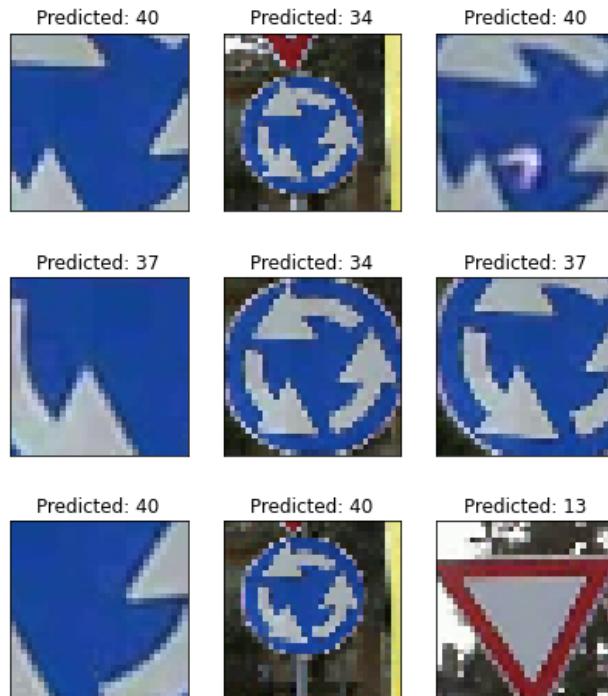
- Crop multipli alle rilevazioni del filtro circolare, quando possibile;
- Analisi dei poligoni nel filtro triangolare per confermare la forma triangolare;
- Analisi della numerosità delle rilevazioni in entrambi i filtri.

L'immagine che invece è stata selezionata per svolgere l'ultimo test sulla pipeline completa è la seguente:



**Figura 4.8.** Esempio sulla classificazioni di crop multipli

Nell'immagine in Figura 4.8 sono stati effettuati 15 crop, e di questi ne sono stati scelti 9 randomicamente per mostrare un esempio delle predizioni ottenute:



CONVERSIONE DELLE DIGITS:

'13' <=> 'Yield'  
'34' <=> 'Turn left ahead'  
'37' <=> 'Go straight or left'  
'40' <=> 'Roundabout mandatory'

In primis occorre notare che l'intera esecuzione della pipeline ha richiesto 4 secondi e 30 millesimi di secondo. Le tempistiche non sono sufficientemente basse reputare l'esecuzione real-time, ma al contempo è sufficientemente veloce per valutare l'utilizzo di tale pipeline in ulteriori ambiti: un tra questi potrebbe essere l'applicazione di tale pipeline in un sistema di rilevazione e riconoscimento dei cartelli che informa l'autista, tramite il quadrante, sulla velocità massima di marcia concessa o di particolari situazioni stradali. Ad ogni modo, oltre all'efficiente temporale, bisogna anche ovviamente valutare l'efficienza della rete. Dai risultati riportati è possibile osservare come il rilevamento dei cartelli stradali è molto preciso e che la maggior parte di essi viene correttamente predetto, sia nel caso in cui il cartello sia ben visibile su tutta l'immagine e sia nel caso il cartello sia lievemente ravvicinato e ritagliato. Le predizioni che danno maggiormente errore sono i crop troppo ravvicinati del cartello. Il caso più rilevante di predizione errata è il cartello in posizione [riga=1, colonna=3]. In quest'ultimo la predizione non è corretta perché probabilmente la rete non è stata addestrata a dover nel distinguere il cartello "Roundabout mandatory" dal cartello "Turn left ahead". Ad ogni modo è un errore di poco conto, in quanto come già precedentemente riportato, una possibile soluzione a tal problema potrebbe essere aumentare il numero di samples delle classi più scarne così da addestrare la rete a distinguerle meglio.

Ad ogni modo i risultati ottenuti al termine di tutti i test possono essere ritenuti ottimi.

# Capitolo 5

## Conclusioni

Osservando i risultati evidenziati dall'ultima valutazione, è possibile denotare come la pipeline creata svolga un ottimo lavoro nel rilevamento e nel riconoscimento di cartelli stradali. I filtri geometrici individuato i cartelli d'interesse nella maggior parte dei casi in esame e con una buona precisione. La rete di classificazione, invece, è stata addestrata in maniera egregia, ottenendo delle accuracy molto elevate anche in casistiche complesse e realistiche. L'aver poi a disposizione una rete così ben addestrata, ha facilitato il connubio tra i componenti dediti al rilevamento dei cartelli stradali con il riconoscimento di quest'ultimi, in quanto, anche se le rilevazioni spesso non si sono dimostrate precise, la rete ha comprovato un'elevata versatilità e affidabilità nel riconoscere le classi corrette.

Ad ogni modo, però, la pipeline presenta alcune difficoltà in situazioni altrettanto rilevanti. Per giunta, trattandosi di una pipeline che elabora informazioni per un'ipotetica automobile, qualsiasi problematica riscontrata è di assoluta rilevanza per la sicurezza di un eventuale essere umano a bordo della vettura, e pertanto è di rilievo esplicitare anche le principali problematiche incontrare: la prima riguarda l'assenza di alcuni tipi di cartelli stradali nel dataset d'addestramento della rete. Questa carenza di cartelli comporta spesso un'iniziale rilevazione corretta, ma una successiva classificazione errata, in quanto ovviamente non esiste la classe corrispondente. Una nota positiva che è possibile trarre da questa mancanza è che comunque la rete pre-addestrata spesso indica come classe corretta quella di un cartello che presenta colori e forme simili, e quindi è possibile denotare che la rete è stata addestrata correttamente. Pertanto, per risolvere tale problema, è sufficiente ampliare o sostituire il training dataset utilizzato. Invece un secondo problema individuato, e di maggior rilievo rispetto al precedente, riguarda le eccessive rilevazioni che spesso non corrispondono nemmeno ad oggetti triangolari o circolare. Teoricamente i parametri delle funzioni che ricercano forme triangolari o circolari andrebbero personalizzati ed adattati per ogni immagine, ma lo scopo di questo progetto era quello di creare una pipeline unica e ottima per il maggior numero di immagini possibile. Pertanto, dato tale obiettivo, è difficile individuare dei valori generici che siano perfetti in tutti i casi. L'unico modo per risolvere tale problema senza cambiare approccio di rilevazione è cercare delle funzioni aggiuntive che aiutino l'algoritmo nella sua selezione dei cartelli stradali o aggiungere ulteriori filtri per aumentarne la precisione. Purtroppo, però, in entrambi i casi si verificherebbe un inevitabilmente aumento del costo

computazionale, e pertanto anche una degradazione delle prestazioni temporali. Quest'ultimo peggioramento sembrerebbe non essere un problema primario, in quanto non avendo mai tempistiche di esecuzioni real-time, la pipeline non avrebbe comunque potuto essere utilizzata come parte di un sistema di guida autonoma. Ad ogni modo, però, esistono altre casistiche di utilizzo che soffrirebbero di tale degradazione: un esempio è un impiego di tale pipeline in un sistema di rilevamento e riconoscimento dei cartelli che, come esperto nel capitolo precedentemente, informi l'autista sulla velocità massima di marcia concessa o di particolari situazioni stradali. In quest'ultimo approccio non occorrono obbligatoriamente delle prestazioni real-time, ma non è nemmeno consigliabile una latenza eccessiva, poiché se il tempo richiesto dalla pipeline è troppo prolungato, è al contempo probabile che l'automobile incontri nuovi cartelli, rendendo di conseguenza obsoleti quelli in elaborazione. Pertanto, date le considerazioni appena esposte, la soluzione migliore è la pipeline attuale, nella quale sono stati coniugati al meglio il bisogno di risultati ottimi con delle prestazioni temporali adeguate.

In conclusione, quindi, osservando tutti i risultati ottenuti e le varie problematiche riscontrate al termine degli esperimenti, si può ritenere raggiunto con successo l'obiettivo di rilevare e riconoscere svariati cartelli stradali attraverso una rete di classificazione pre-addestrata e l'utilizzo di filtri geometrici.

# Bibliografia

- [1] tensorflow.org, “Documentazione di tensorflow,”  
*Sito di Riferimento: <https://www.tensorflow.org/tutorials/images/cnn>.*
- [2] opencv.org, “Documentazione di opencv,”  
*Sito di Riferimento: <https://docs.opencv.org/2.4/index.html>.*
- [3] keras.io, “Documentazione di keras,”  
*Sito di Riferimento: <https://keras.io/api/>.*
- [4] E. Alati and F. Pirri, “Tipi di funzione di attivazione.” Lezione "MQI – lezione 15" del corso di "Metodi Quantitativi per l'Informatica", slide 5-6-22-23, June 2020.
- [5] Gupta and Dishashree, “Fundamentals of deep learning–activation functions and when to use them,”  
*Sito di Riferimento: <https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them>, 2017.*
- [6] H. Sharma, “Activation functions: Sigmoid, relu, leaky relu and softmax basics for neural networks and deep learning,”  
*Sito di Riferimento: <https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e>, 2019.*
- [7] R. Gómez, “Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names,”  
*Sito di Riferimento: [https://gombru.github.io/2018/05/23/cross\\_entropy\\_loss/](https://gombru.github.io/2018/05/23/cross_entropy_loss/), 2018.*
- [8] E. Alati and F. Pirri, “Tipi di optimizer.” Lezione "MQI – lezione 15" del corso di "Metodi Quantitativi per l'Informatica", slide 13-14-15-16, June 2020.
- [9] flo2607, “Dataset = "traffic signs classification",”  
*Sito di Riferimento: <https://www.kaggle.com/flo2607/traffic-signs-classification/metadata>, March 2020.*
- [10] Sapienza, “Dataset = "dits classification" and "dits detection",”  
*Sito di Riferimento: <http://users.diag.uniroma1.it/bloisi/ds/dits.html>.*

- [11] I. F. Neurpinformatik, "Dataset = "gtsdb, the german traffic sign detection benchmark","  
*Sito di Riferimento: <http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset>*, 2013.