Università degli Studi di Ferrara

Corso di Laurea Magistrale in Intelligenza Artificiale, Data Science e Big Data

Project Work Deep Learning

CNN Traffic Sign Recognition: Progettazione, Evoluzione e Valutazione di modelli su GTSRB

A.A. 2024 - 2025

Gabrieli Andrea

Indice

Il project work aveva come fine il riconoscimento dei segnali stradali sul dataset GTSRB mediante l'uso di varie reti neurali convoluzionali.

1 Dataset e Preprocessing

Il dataset utilizzato è il German Traffic Sign Recognition Benchmark (GTSRB)[1]. È un dataset di classificazione multiclasse, 43 in particolare, ognuna corrispondente a un determinato cartello (vedi Appendice A).

Una volta studiato e osservato, ho iniziato ad elaborare e formattare le varie immagini. Le ho **ritagliate secondo le coordinate ROI** (Region of Interest) presenti nel file .csv incluso all'interno di ogni sottodirectory; fatto il resize della dimensione a **32x32 pixel**; e poi li ho **normalizzati** nell'insieme **[0, 1]**.

Di seguito viene illustrata la distribuzione del dataset una volta fatte queste prime operazioni.

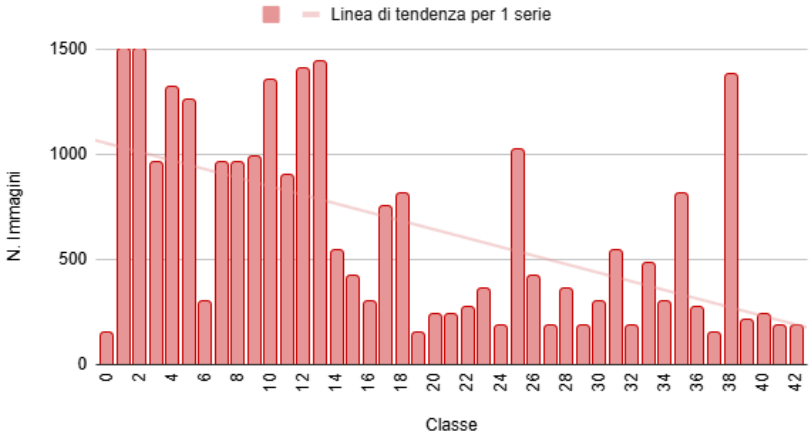


Figura 1: Distribuzione delle classi nel dataset originale

Ho successivamente suddiviso in **train-test-validation** set il dataset. Da qui, è emerso il **forte sbilanciamento** tra classi. La **data augmentation** è stata necessaria. Le nuove immagini sono state generate eseguendo le seguenti operazioni: leggere **rotazioni**, aumento **luminosità/contrasto**, **flip orizzontale**. Con queste operazioni, le classi sotto rappresentate sono state portate alla **soglia minima di 420 img**. La data aug. produce nuovi dati falsi. È una tecnica utile, ma deve comunque essere mantenuto un buon rapporto tra immagini vere/reali e nuove. Se ad esempio avessi scelto una **soglia maggiore**, come 520/550, le **immagini sintetiche sarebbero state molte** di più, e così facendo i modelli che poi sarei andato ad addestrare avrebbero **appreso pattern falsati**, perdendo o **compromettendo la capacità di generalizzazione**.

La data aug. l'ho poi ri-eseguita, **eliminando il flip** or, ma aggiungendo le operazioni: leggero **zoom** e aumento della **saturazione**. Questo **perché**, una volta addestrati i primi due modelli, e fatti i test su cartelli esterni al dataset, ho scoperto che entrambe le reti sbagliavano la direzione del cartello (sx con dx e viceversa). Ripercorrendo tutte le

[1] https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

operazioni effettuate a ritroso, ho capito che il problema era qua, e in particolare nell'operazione di flip. Questa ruotava le immagini, ma così facendo **diventavano di un'altra classe**, **ma** rimanevano **assegnate all'etichetta originale**. Così ho tolto quella operazione e inserito solo operazioni che non alteravano la direzione. (Il flip l'avevo inserito perché pensavo potesse riflettere una maggiore varietà delle immagini, e ad esempio il fatto che un cartello fosse fotografato da uno specchietto dell'auto, ma senza pensare alla presenza dei cartelli direzionali).

La **suddivisione** nei sotto-dataset è stata effettuata secondo le probabilità: **70%** per il train e **15%** sia per il validation che per il test. Qui, ho usato la **stratificazione** per mantenere la distribuzione tra classi.

Il grafico sottostante mostra la distribuzione del training set sia prima che una volta effettuata la data augmentation.

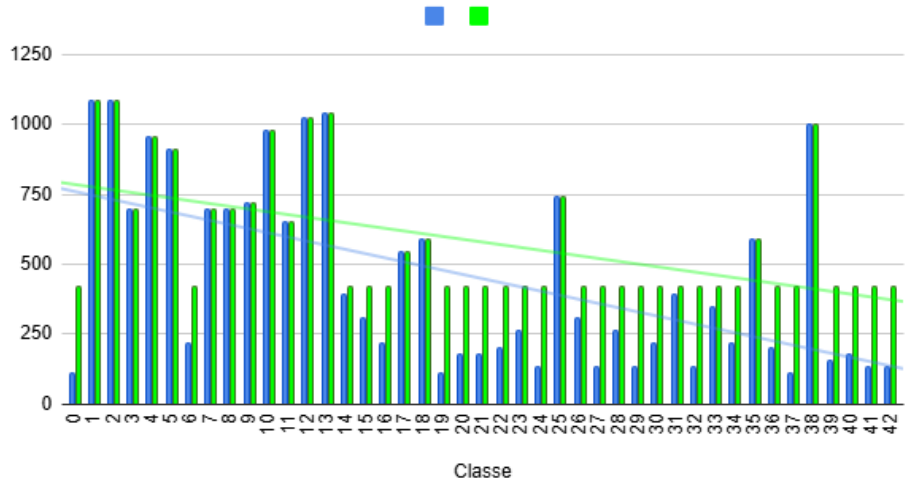


Figura 2: Distribuzione training set prima - dopo la augmentation

Risulta evidente che la data augmentation abbia portato a un **miglioramento** del forte **sbilanciamento**, portando la retta di tendenza a una pendenza minore. Le immagini di seguito rappresentano come la data augmentation abbia avuto effetto sul dataset.

Figura 3: Immagini di alcune classi prima e dopo la data augmentation

2 Struttura file system

- ➔ data/ Segnali originali di GTSRB
- ➔ dataset/ Dataset: Training (prima e dopo), Validation e Test
- ➔ images/
  - ◆ dataset/ Segnali prima e dopo la data augmentation
  - ◆ summary/ Struttura delle reti (*model.summary()*)
  - ◆ test/
    - internet/ Cartelli ritagliati da internet
    - original/ Immagini originali
    - particular/ Cartelli ritagliati con font/forme particolari
    - personal/ Cartelli ritagliati dalle foto personali
  - ◆ training/ Andamento dei training di tutte le reti e le matrici di confusione
- ➔ list/ Segnali ed etichette una volta effettuato il preprocessing
- ➔ models/ CNN salvate
- ➔ networks/ Costruzione delle CNN
- ➔ scripts/ Tutte le altre operazioni

3 Implementazione delle CNN

Questo capitolo analizza le reti convoluzionali pensate per risolvere il problema. Mostra quindi il procedimento ed i passaggi per arrivare al modello finale, includendo anche i dettagli e i risultati dell'apprendimento delle stesse.
Tutti gli addestramenti sono stati effettuati con le seguenti scelte:

- ➔ Loss: Categorical cross entropy
- ➔ Ottimizzatore: Adam
- ➔ Epoche: 25
- ➔ Batch size: 128
- ➔ Metriche: Accuracy (Recall)

L'accuracy è la metrica principale che è stata usata. Per la prima rete ho fatto una prova ulteriore usando la recall, in quanto, in presenza di dataset molto sbilanciati, la accuracy da sola può non essere molto indicativa.
I training di tutte le reti sono stati effettuati due volte, la prima con il training set a cui era stata applicata la data augmentation con il flip orizzontale, mentre la seconda volta con la data augmentation corretta ma su un numero di epoche ridotto (20).

3.1 LeNet 5

La prima rete implementata è stata **LeNet 5**. Ho scelto di iniziare con questa rete, perché è una CNN **semplice** e **la prima sviluppata** storicamente. Poi siccome ero ancora all'inizio, ho pensato di non iniziare con un modello troppo complicato, ma bensì con una rete **veloce da addestrare** e con **pochi parametri**, anche per capire e valutare meglio se le operazioni fatte precedentemente erano state eseguite nel modo corretto.

Architettura della rete

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 6)	456
average_pooling2d (AveragePooling2D)	(None, 16, 16, 6)	0
conv2d_1 (Conv2D)	(None, 12, 12, 16)	2,416
average_pooling2d_1 (AveragePooling2D)	(None, 6, 6, 16)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 120)	69,240
dense_1 (Dense)	(None, 84)	10,164
dense_2 (Dense)	(None, 43)	3,655

Total params: 85,933 (335.68 KB)
Trainable params: 85,931 (335.67 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)

Figura 4: LeNet 5 summary

L'architettura è quella classica di una LeNet 5. **Due blocchi convoluzionali** formati da due convoluzioni con filtri crescenti, 6 per la prima e 16 per la seconda e **kernel** di dimensione **5x5**. Con l'**Average Pooling 2x2** e stride a 2, in modo da dimezzare la dimensione della feature map. Dopo questi due blocchi, lo strato **Flatten** e gli strati **fully connected** che terminano con lo strato **Dense con softmax** per la classificazione. Per tutti gli altri strati, invece, relu come funzione di attivazione.

Descrizione training

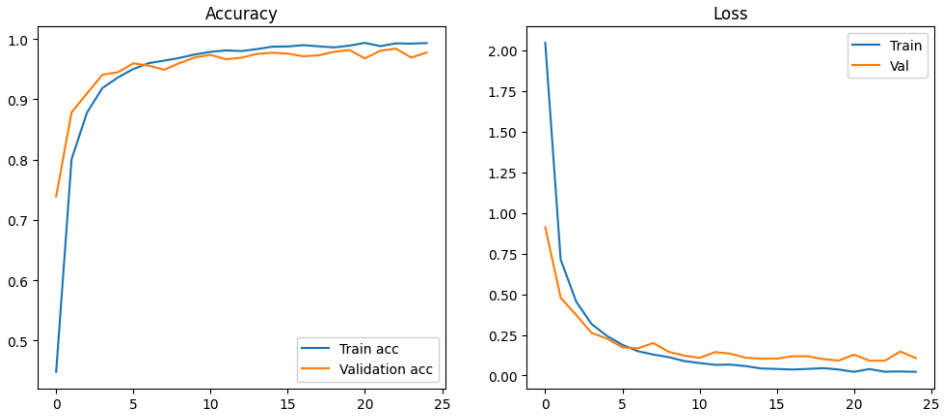


Figura 5: Training di LeNet 5 (1)

I valori di accuracy/recall/loss sul test set sono stati:

- ➔ Accuracy = 0.9875
- ➔ Recall = 0.98123
- ➔ Loss = 0.08727

I risultati ottenuti sono comunque promettenti. La rete è molto semplice, poco profonda, ma l'accuracy ottenuta è del 98.75%. Le due curve, train e validation sia di accuracy che della loss, si stabilizzano superate circa le 10 epoche, anche se le curve di validation mostrano un andamento più "ballerino" e la loss mostri una piccola tendenza all'aumento, quindi un piccolo principio di overfitting.

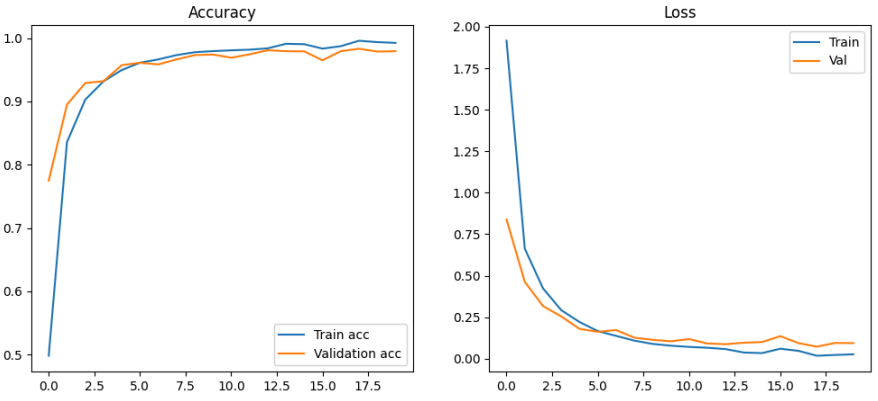


Figura 6: Training di LeNet 5 (2)

Il secondo addestramento ha restituito:

- ➔ Accuracy: 0.9807
- ➔ Loss: 0.0915

Anche in questo caso l'accuracy si attesta sul 98.07%, con una differenza molto leggera di 0.68. Qui le due curve si stabilizzano tra la quinta e la sesta epoca, ma anche in questo caso, analizzando la loss, si nota una leggera tendenza alla risalita.

Risultato finale

I risultati ottenuti sono, tutto sommato, buoni. Già nelle prime 5 epoche, accuracy e loss aumentano/calano in modo ripido. Si arriva già ad una val-accuracy alta fin da subito, e nelle successive epoche continua a migliorare. Intorno alle ultime epoche, si iniziano a notare piccoli segnali di aumento della loss di validazione. Considerando la rete, la sua semplicità, ottenere una accuracy di **98.07**, significa che la rete ha ottenuto risultati molto buoni. La base di partenza è dunque molto valida.

3.2 LeNet 5 Plus

Questa seconda architettura l'ho pensata per cercare di migliorare le performance del precedente modello. È una rete a cui ho applicato importanti modifiche rispetto a LeNet 5, soprattutto nella profondità.

Architettura della rete

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 43)	5,547

Total params: 657,133 (2.51 MB)
Trainable params: 656,683 (2.51 MB)
Non-trainable params: 448 (1.75 KB)
Optimizer params: 2 (12.00 B)

Figura 7: LeNet 5 Plus summary

Qui i **blocchi convoluzionali passano a tre**, in questo modo la rete può apprendere feature più complesse/astratte. La **Batch Normalization** dopo ogni convoluzione permette di ridurre l'internal covariate shift. L'average pooling è stato sostituito con il **max pooling**, in modo da mantenere le feature più importanti. Il **numero di filtri** è stato **aumentato** in modo progressivo, **raddoppiandoli ogni volta**. 32 per la prima convoluzione, passando a 64, e arrivando a 128.
La dimensione del **kernel** è stata ridotta, ora sono **3x3**, così da avere una precisione migliore. Nella parte fully connected, gli strati densi sono rimasti in numero uguale, ma con

quantità diverse: 256, 128 e 43. L'altra differenza è l'aggiunta di **dropout** tra uno strato denso e l'altro, in modo da migliorare la generalizzazione.

Descrizione training

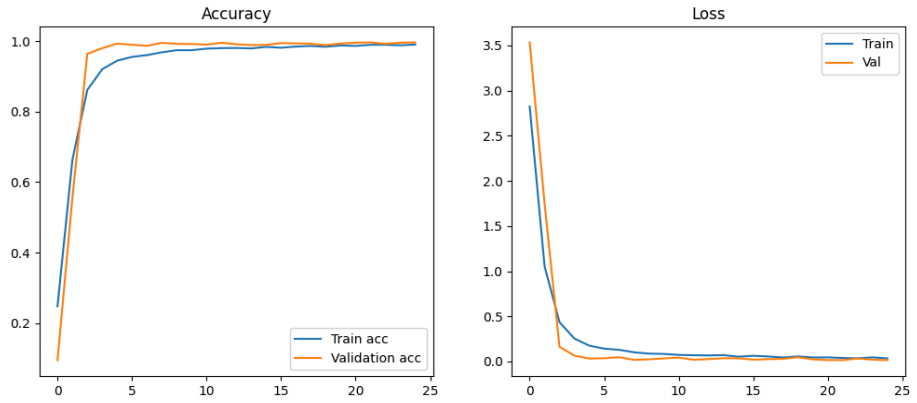


Figura 8: Training di LeNet 5 Plus (1)

Il primo addestramento ha restituito:

- ➔ Accuracy: 0.9947
- ➔ Loss: 0.0211

Training molto buono, stabilizzazione già dopo la quinta epoca, gap tra le curve ridotto al minimo e nessuna tendenza alla risalita nella loss, quindi nessun segno di overfitting.

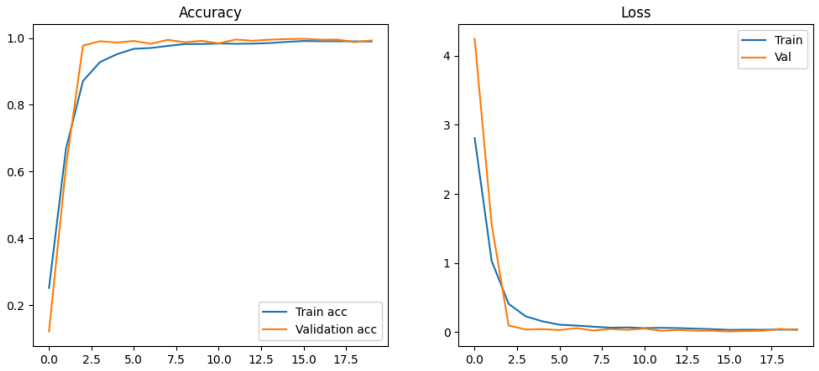


Figura 9: Training di LeNet 5 Plus (2)

Il secondo addestramento, invece, ha restituito:

- ➔ Accuracy: 0.9937
- ➔ Loss: 0.0244

È chiaramente visibile una rapida variazione dei valori già nelle prime tre epoche, per poi stabilizzarsi e mantenere un gap tra le curve estremamente ridotto.

Risultato finale

L’andamento dell’addestramento è molto buono anche per questa rete. Si nota immediatamente un aumento della accuracy rispetto alla rete precedente. Questa rete nella sua versione migliore arriva a 99.3, **quasi 99.4** con una loss molto bassa. Le due curve, train e validation, sia loss che accuracy sono sempre molto vicine tra loro. La loss di validazione ha un calo estremamente rapido per poi stabilizzarsi intorno alla terza epoca. Segnali di una tendenza alla risalita della loss non sono visibili immediatamente, sintomo che la rete riesca a generalizzare molto bene.

3.3 LeNet 5 Pro

LeNet 5 Pro è la terza rete che ho sviluppato. È la rete in cui ho cercato di portare le modifiche aggiunte nella precedente al loro massimo possibile, considerando anche le risorse a disposizione.

Architettura della rete

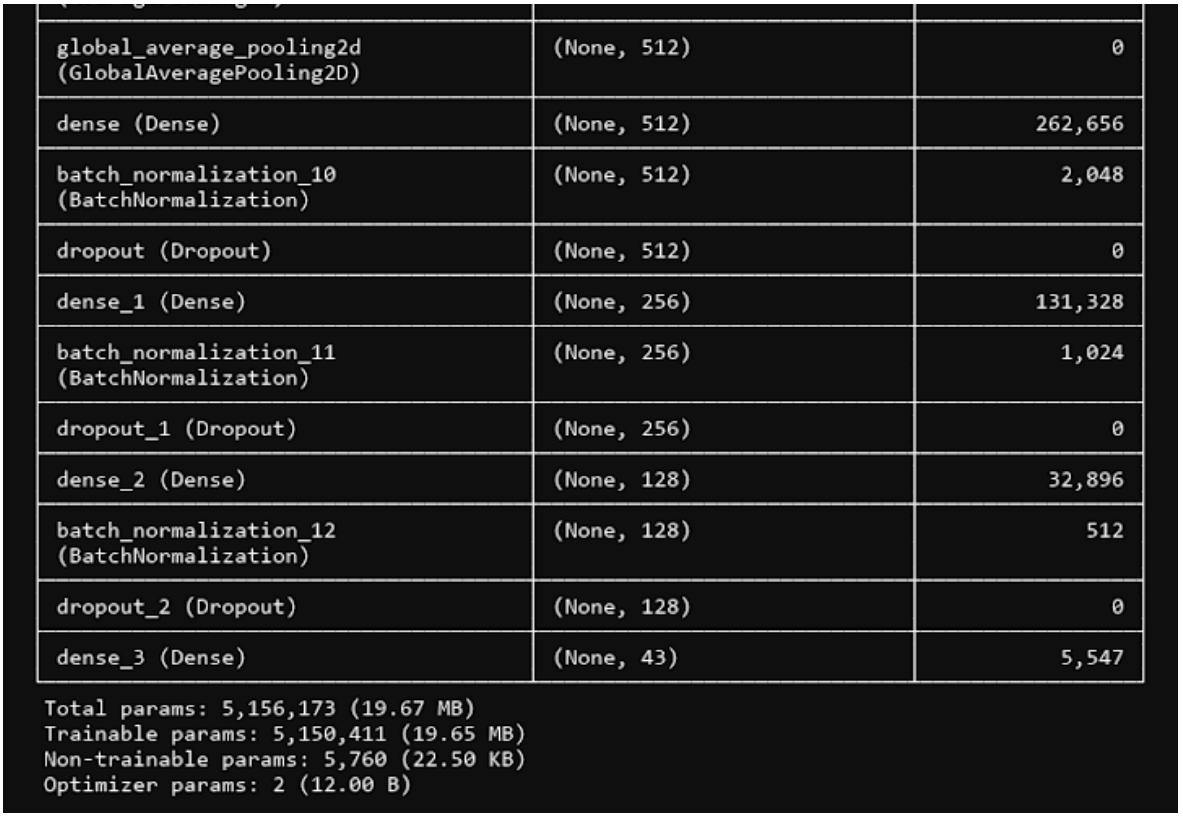
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
average_pooling2d (AveragePooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
average_pooling2d_1 (AveragePooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
average_pooling2d_2 (AveragePooling2D)	(None, 4, 4, 128)	0
conv2d_6 (Conv2D)	(None, 4, 4, 256)	295,168
batch_normalization_6 (BatchNormalization)	(None, 4, 4, 256)	1,024
conv2d_7 (Conv2D)	(None, 4, 4, 256)	590,080
batch_normalization_7 (BatchNormalization)	(None, 4, 4, 256)	1,024
average_pooling2d_3 (AveragePooling2D)	(None, 2, 2, 256)	0

Figura 10: LeNet 5 Pro summary (prima parte)

Il numero di blocchi convoluzioni passa a cinque. Tutti i layers di convoluzione usano kernel 3x3 con funzione di attivazione relu. Ognuno di questi è costituito da una doppia convoluzione dove il numero di filtri raddoppia ogni volta. Si parte dal primo blocco con 32 arrivando al quinto blocco da 512 filtri. Anche in questa rete, dopo ogni convoluzione è stato aggiunto un layer di Batch Normalization e un ritorno all'average pooling, come nella prima rete.

La caratteristica principale di questa rete è avere una doppia convoluzione a blocco. Questa **idea** è quella **su cui si basa VGG net**, dove due convoluzioni 3x3 hanno lo stesso campo recettivo di una singola convoluzione 5x5, ma con un numero di parametri decisamente minore.



global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 512)	262,656
batch_normalization_10 (BatchNormalization)	(None, 512)	2,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_11 (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
batch_normalization_12 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 43)	5,547

Total params: 5,156,173 (19.67 MB)
Trainable params: 5,150,411 (19.65 MB)
Non-trainable params: 5,760 (22.50 KB)
Optimizer params: 2 (12.00 B)

Figura 11: LeNet 5 Pro summary (seconda parte)

Nella parte fully connected vi sono ulteriori modifiche. Il posto del layer **Flatten** è stato occupato da **GlobalAveragePooling2D**. GAP è sempre un modo per ottenere una rappresentazione vettoriale, la differenza principale sta nell'output. Infatti il Flatten produce un output molto grande, mentre il GAP permette di ridurlo, e quando si ha un modello molto grande, se esiste un modo più leggero, è spesso una buona idea da tenere in considerazione. La differenza tra i due si può facilmente illustrare con un esempio. Se l'output del layer precedente fosse 4x4x128, lo strato Flatten produrrebbe un vettore 4x4x128 = 2048 neuroni, mentre il GAP ha solamente 128 neuroni.

Nella rete gli strati **densi passano** da tre **a quattro**, in modo da poter aumentare il numero di units, in questo modo il primo è da 512, per poi decrescere fino a quello da 43 in softmax

per la classificazione finale. Tra un denso e l'altro vi continua ad essere un **layer di dropout**, ma viene anche aggiunta la **Batch Normalization**.

Descrizione training

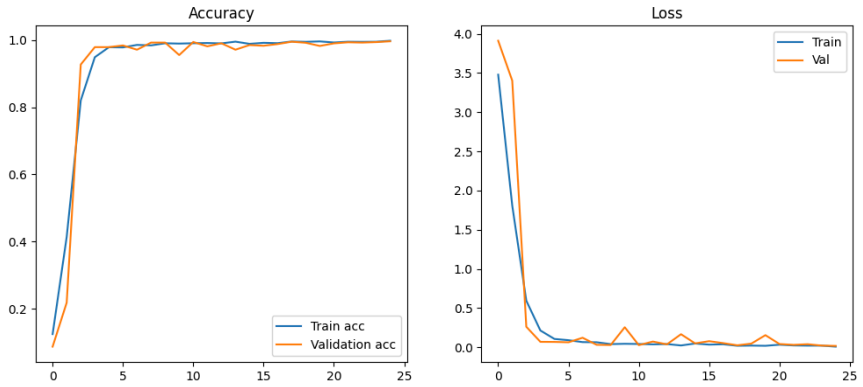


Figura 12: Training di LeNet 5 Pro (1)

I valori ottenuti da questo addestramento sono:

- ➔ Accuracy: 0.99499
- ➔ Loss: 0.0218

L'andamento parte da valori molto bassi, ma già alla 3'/4' epoca il training inizia già a stabilizzarsi. La loss ha un andamento calante molto ripido fino alla stabilizzazione sempre intorno alla 4' epoca. La distanza tra le curve di train e validation è molto ridotta ma sono visibili alcune oscillazioni, ma comunque non risultano grandi segnali di overfitting.

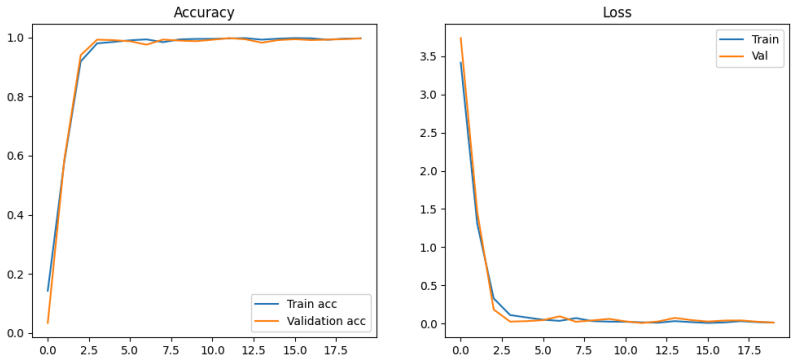


Figura 13: Training di LeNet 5 Pro (1)

I valori ottenuti sono:

- ➔ Accuracy: 0.99699
- ➔ Loss: 0.0194

Rispetto al precedente, le due curve, train e validation, sono sempre molto vicine tra loro. Scendono/aumentano, a seconda del fatto che si consideri loss/accuracy molto velocemente per poi stabilizzarsi e rimanere stabili fino alla fine del training.

Risultato finale

L’addestramento del modello è stato molto buono. Questa rete si attesta a un livello di accuracy di **99.7** e una loss di circa $2x10^{-2}$. Le due curve sono sin dall’inizio completamente sovrapposte per stabilizzarsi già dopo la seconda epoca. La loss mostra qualche leggera oscillazione, sintomo che la rete sta ancora migliorando i parametri. Le aggiunte/modifiche apportate hanno prodotto l’effetto desiderato.

4 Test su altre immagini

In questo capitolo analizzo i risultati ottenuti dai modelli su immagini prese da altre fonti. Queste altre immagini provengono in parte da foto scattate personalmente, e da foto trovate da ricerche su Internet, per entrambi i casi, le foto testate sono state ritagliate dalle immagini originali più grandi e sottoposte alle stesse operazioni delle immagini del dataset.

4.1 Test su immagini personali

Le immagini testate sono state i seguenti cartelli: Limite 60 km/h (3), Divieto di sorpasso (9), Divieto di accesso (17), Pericolo animali selvatici (31) e Passaggio a sinistra (39).

I test sono stati molto buoni perché tutti e tre i modelli classificano correttamente tutti i cartelli. Viene commesso un solo errore dal modello intermedio, la CNN Plus. Questo errore è sul cartello 31, attraversamento animali selvatici. Questo viene confuso con il pericolo di doppia curva (21). I motivi possono essere diversi, quello più plausibile potrebbe essere il fatto che il cervo raffigurato ha in effetti una forma che potrebbe in qualche modo richiamare l'andamento della doppia curva. Analizzando i cartelli, infatti, entrambi hanno quell'andamento che parte dalla punta destra del triangolo e termina nel lato opposto. Aggiungendo il fatto che la rete non abbia una elevata capacità, ecco che l'errore può essere spiegato.

4.2 Test su immagini da Internet

Le immagini appartenenti a questa categoria sono le seguenti: Dare precedenza (13), Curva pericolosa a sinistra (19), Due Lavori in corso da cantiere (25) e Passaggio a sinistra (39).

I primi due modelli classificano correttamente tutte e cinque le foto senza commettere errori. Il terzo modello invece classifica male uno dei due cartelli 25, quello leggermente più ruotato. Una possibile spiegazione di questo problema potrebbe essere la seguente.
Nell'immagine che sbaglia vi è in primo piano il cartello dei lavori in corso, ma sullo sfondo (angolo destro) vi è un secondo cartello, un divieto di sosta. La rete potrebbe aver dato maggior peso al cartello di sfondo che a quello in primo piano. In più, quel cartello sullo sfondo, non compare tra le 43 classi del dataset, quindi il modello potrebbe avergli assegnato la classe più vicina, la num. 5 (limite 80). Nella seconda foto, la situazione cambia, perché vi è il cartello e niente sullo sfondo, la rete non può che concentrarsi su quello, e classificarlo bene. Riassumendo, la rete, nella prima foto, potrebbe essere stata distratta dallo sfondo.
Questo risultato mostra come una rete più profonda-complessa abbia appreso pattern più difficili, e commetta errori, dove reti più semplici non li commettevano.

Test su immagini particolari

Le immagini di questa categoria sono le seguenti: Limite 30 e 70 km/h (1 e 4), Divieto di sorpasso (9), Pericolo generico (18), Lavori in corso (25), Pericolo di attraversamento ciclabile (29), Pericolo ghiaccio/neve su strada (30), Obbligo di curva a destra (33), Rotatoria (40).

Questi cartelli presentano uno stile diverso rispetto ai cartelli contenuti nel dataset. Presentano uno stile molto simile a cartelli francesi/austriaci. Presentano differenze nel font di scrittura per i limiti di velocità, immagini più stilizzate per le auto, e rappresentazioni più minimali per il “lavori in corso” e per “l’attraversamento ciclabile”. Provando le tre CNN su queste immagini, ho potuto testare la loro robustezza e simulare anche contesti diversi. I risultati sono stati in generale buoni. Gli unici cartelli più ostici sono stati l’attraversamento ciclabile (29) per tutte e tre le reti e il limite dei 30 (1) per LeNet 5. L’errore sul cartello 29 viene commesso da tutte le reti, ma questo errore è abbastanza comprensibile. Considerando che il soggetto raffigurato ha uno stile molto particolare, il cartello presenta condizioni di usura evidenti ed anche il fatto che non è perfettamente allineato (l’immagine è vista angolata), mettendo tutto insieme può essere giustificabile. Sul cartello 1, LeNet 5 lo associa alla classe 5 (limite 80). Questo può essere interpretato in due modi. Da un punto di vista visivo può sembrare che la rete abbia solo confuso un 3 per un 8, magari per la qualità dell’immagine. Ma da un punto di vista reale, immaginando che una rete come questa sia inclusa in un sistema di guida autonoma, associare a un limite 30 il limite degli 80 porta a prendere decisioni completamente opposte. Da un lato a una guida in cui si può accelerare, dall’altro a dover prestare una maggior attenzione al circondario. È dunque un errore critico, perché mostra come sbagliare un numero possa portare a prendere decisioni completamente diverse.

Conclusioni e prospettive future

L'obiettivo del project work era la classificazione dei segnali stradali. Per raggiungere questo obiettivo ho implementato tre diverse architetture, tre diverse reti neurali convoluzionali. Nella tabella sottostante sono riassunti i migliori risultati in termini di accuracy di tutti i modelli.

LeNet 5	LeNet 5 Plus	LeNet 5 Pro
98.07 %	99.37 %	99.7 %

La prima rete, nonostante le semplicità si rivelata una buona architettura, con un livello di accuracy che si attesta sul 98.07%. Buona rete, robusta nella classificazione di segnali con piccole modifiche.

La seconda rete, LeNet 5 Plus, è una CNN sviluppata a partire dalla precedente. Ha una potenza complessiva maggiore e il risultato si è potuto osservare dall'accuracy ottenuta, 99.37%. Rete più complessa, ma con ottimi risultati, soprattutto nel test.

La terza rete, LeNet 5 Pro, è invece una CNN in cui ho cercato di portare al limite le modifiche fatte nella precedente, e, oltre a questo, ho cercato di sfruttare l'idea alla base di VGG. Sebbene le premesse promettessero bene e con un livello di accuracy pari a 99.7%, durante i test su dati mai visti, ha commesso un errore sistematico su un cartello. Questo è sintomo che la rete si è specializzata troppo sul dataset, perdendo la potenza di generalizzazione.

Dalle tre architetture sviluppate e soprattutto dai test effettuati, ho potuto quindi notare che l'aumento della complessità non porti in automatico alla costruzione del modello perfetto. Modelli più semplici, con meno parametri, hanno appreso pattern più superficiali, e questo gli ha permesso in certi casi di generalizzare meglio di modelli più profondi e con più parametri. Questi ultimi infatti, si sono forse specializzati troppo sulle caratteristiche del dataset, andando a perdere un po' della capacità di generalizzazione sui dati mai visti. In generale ho potuto riflettere su:

1. Quanto la qualità/coerenza dei dati sia molto importante. Avere un dataset che sia ben formato è forse più importante di avere una rete estremamente complessa.
2. I numeri che si ottengono terminati il training devono essere ben analizzati, non basta fidarsi del numero ottenuto, spesso modelli con risultati minori, hanno performance migliori.
3. Gli errori di classificazione sono una grande opportunità per capire cosa è funzionato o meno nell'architettura e sono un importante metodo per capire come migliorare.

Posso concludere dicendo "La semplicità vince contro la complessità" perché è meglio avere un modello chiaro e leggero che funzioni nei casi reali, più che uno complesso e profondo che sbagli al variare di piccole condizioni.

Eventuali ulteriori sviluppi possono seguire due direzioni.

1. Incrementare ulteriormente la profondità della rete. Significherebbe costruire una rete con più blocchi convoluzionali, più pooling, inserendo magari strutture simili alle Inception di GoogleNet con le convoluzioni 1x1.

2. Utilizzo di modelli pre-addestrati. Applicare quindi tecniche di fine tuning a reti come ResNet o GoogleNet, andando a congelare i primi layer convoluzionali e creano/allenando solo il nuovo classificatore finale, adattato al nostro problema.

Entrambe le strategie presentano alcuni problemi. Considerando che la rete LeNet 5 Pro ha raggiunto una accuracy di 99.7%, costruire una rete ancora più profonda/complessa porterebbe a una maggiore probabilità di overfitting. La rete si concentrerebbe troppo sul dataset di training, perdendo la propria capacità di generalizzazione. L'altra tecnica, invece, introdurrebbe complessità ulteriore e tempi di training decisamente maggiori, richiedendo risorse che possono essere elevate.

Quindi, in entrambe le direzioni, il guadagno che si potrebbe ottenere è molto limitato (siamo nell'ordine di 0.1-0.2%) rispetto a una complessità decisamente maggiore e un maggiore rischio di overfitting. Dato che le reti create hanno dimostrato performance molto buone, il rapporto costi/benefici sarebbe dunque sfavorevole.

Appendice A

Elenco classi - relativo segnale stradale

0	Limite massimo di velocità 20 km/h	22	Strada deformata
1	Limite massimo di velocità 30 km/h	23	Strada sdrucciolevole
2	Limite massimo di velocità 50 km/h	24	Strettoia asimmetrica a destra
3	Limite massimo di velocità 60 km/h	25	Lavori in corso
4	Limite massimo di velocità 70 km/h	26	Segnale di traffico con semaforo
5	Limite massimo di velocità 80 km/h	27	Attraversamento pedonale
6	Fine limite massimo di velocità 80 km/h	28	Passaggio pedonale (uscita pedoni)
7	Limite massimo di velocità 100 km/h	29	Attraversamento ciclabile
8	Limite massimo di velocità 120 km/h	30	Possibile presenza di neve o ghiaccio sulla carreggiata
9	Divieto di sorpasso	31	Attraversamento di animali selvatici
10	Divieto di sorpasso per autocarri	32	Via libera
11	Intersezione con strada senza diritto di precedenza da sinistra/destra	33	Direzione obbligatoria a destra
12	Strada con diritto di precedenza	34	Direzione obbligatoria a sinistra
13	Dare precedenza	35	Direzione obbligatoria diritto
14	Stop	36	Direzioni consentite: diritto e destra
15	Divieto di transito	37	Direzioni consentite: diritto e sinistra
16	Divieto di transito per autocarri	38	Passaggio obbligatorio verso destra
17	Divieto di accesso	39	Passaggio obbligatorio verso sinistra
18	Pericolo generico	40	Rotatoria
19	Curva pericolosa a sinistra	41	Fine del divieto di sorpasso
20	Curva pericolosa a destra	42	Fine del divieto di sorpasso per autocarri
21	Doppia curva, la prima a sinistra		