UNIVERSITÀ
DEGLI STUDI
FIRENZE

Convolutional Neural Network

Dario Mezzasalma

7 luglio 2025

Contents

1 Introduzione

In questo laboratorio si vuole verificare la relazione tra la profondità di una rete neurale e le performance ottenute dimostrando anche l'importanza delle reti residuali che attraverso le skip-connection rendono l'addestramento più stabile in termini di flusso del gradiente.

Setup:

- **Learning rate**: inizialmente settato a 0.01 con scheduler StepLR
- **Ottimizzatore**: Adam
- **Batch size**: 64
- **Epoche di training**: 20

2 MLP

Inizialmente ho implementato un semplice MLP (Multi Layer Perceptron) con solo layer lineari.

Ho valutato le performance di questo modello al variare della profondità: 4, 8, 16, 32.

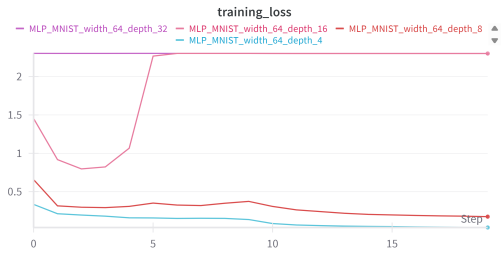

Figure 1: Training loss - MLP

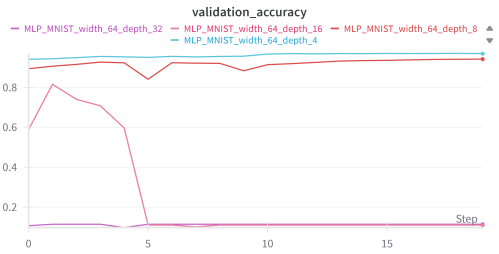

Figure 2: Validation Accuracy - MLP

Dal grafico si nota che all'aumentare della profondità della rete le performance peggiorano, e questo è spiegabile attraverso il fenomeno del *Vanishing Gradient*.

Infatti andando a valutare la norma dei gradienti per ogni layer della rete si può osservare che, nel caso delle reti di $depth = 16$ e $depth = 32$, la norma dei gradienti è nulla (ciò porta al non aggiornamento dei parametri apprendibili, quindi la rete non sta imparando).

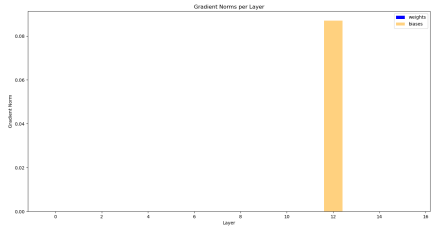

Figure 3: $depth = 16$

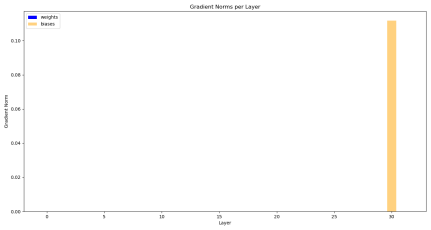

Figure 4: $depth = 32$

Figure 5: Norma dei gradienti per layer

Quando, però, modifico la rete aggiungendo dei layer di *BatchNorm* successivi ad ogni layer lineare, il training diventa più stabile portando a ottime prestazioni in termini di accuracy.

Per quanto riguarda invece la rete con $depth = 32$, l'addestramento risulta molto altalenante con risultati pessimi. Questo potrebbe essere dovuto al fatto che la classificazione su MNIST è un task relativamente semplice per un modello così complesso.

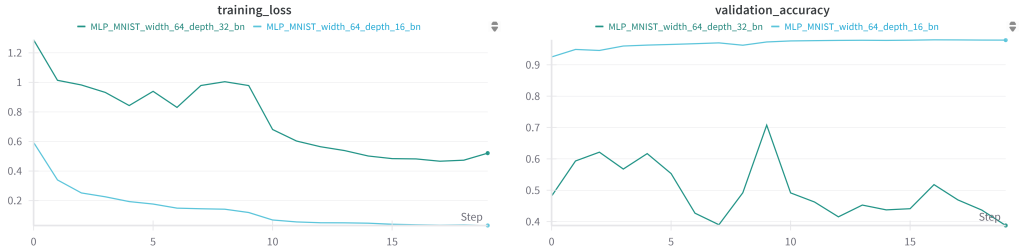

Figure 6: Curve di training loss e validation accuracy - MLP con BatchNorm

Proviamo adesso ad effettuare gli stessi esperimenti utilizzando le *skip-connection* messe in pratica nelle *reti residuali*.

3 Res-MLP

Dai grafici sottostanti si osserva che, l'addestramento della rete profonda a 32 layer con ResidualMLP è stato significativamente più stabile, senza oscillazioni marcate. Inoltre, ha raggiunto un'accuratezza finale del 94% , risultato non ottenuto con l'MLP standard. Questo conferma che l'introduzione delle connessioni residuali facilita l'ottimizzazione anche in architetture più profonde, mitigando problemi legati alla degradazione delle prestazioni.

Figure 7: Curve di training loss e validation - ResMLP

Un'ulteriore conferma di quanto osservato è fornita dal grafico relativo al flusso del gradiente, che evidenzia come le connessioni residue contribuiscano a mitigare il problema del *Vanishing Gradient* nelle reti profonde.

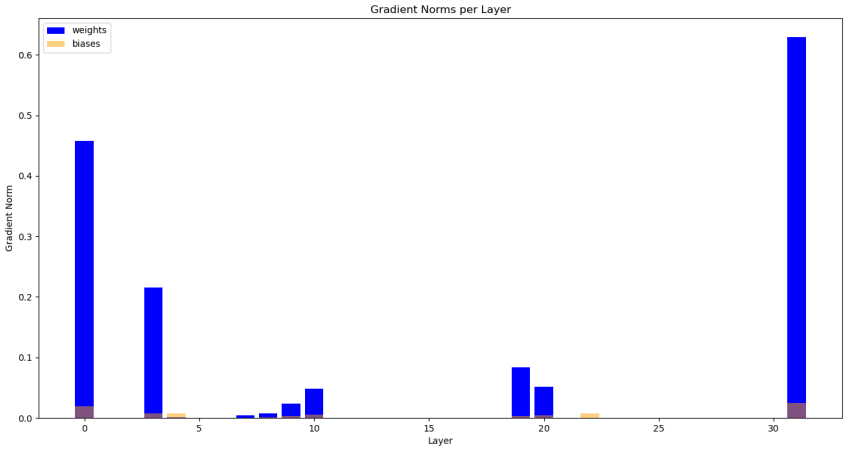

Figure 8: Norma dei gradienti - ResMLP a 32 layer

4 CNN

Poiché il task di classificazione su MNIST risulta piuttosto semplice, ho deciso di sperimentare l'utilizzo delle CNN su un dataset più complesso come CIFAR-10.

In questo caso, settiamo diversamente gli iperparametri:

- **Epoche**: 30

- **Learning rate**: 10^{-3} con scheduler StepLR
- **Batch size**: 128

L'ottimizzatore è uguale a quello del caso precedente. Per quanto riguarda i layer convoluzionali fissiamo:

- **Kernel size**: 3
- **Numero di filtri**: 64
- **Padding**: 1

Come nei precedenti paragrafi, studiamo le curve di training loss e di validation accuracy al variare del numero di layer convoluzionali: 4, 8, 16, 32.

Figure 9: Curve di training loss e validation accuracy - CNN

La rete più profonda (32 layer) è quella che soffre il problema del *Vanishing Gradient*, le altre invece presentano prestazioni piuttosto simili. La rete convoluzionale a 8 layer raggiunge un accuracy del 76.06% che è il maggior risultato ottenuto.

Utilizziamo la versione con *Residual Block* solo per le due reti più profonde.

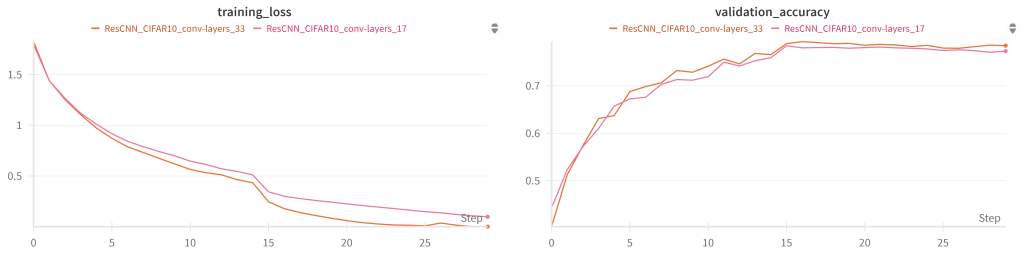

Figure 10: Curve di training loss e validation accuracy - ResCNN

5 CAM (Class Activation Map)

Ho utilizzato la rete *Resnet18* pre-addestrata e implementato CAM (Class Activation Map) per la spiegazione delle singole predizioni del modello.

$$\text{CAM}_c(x, y) = \sum_k w_k^c \cdot f_k(x, y)$$

dove w_k^c rappresenta il contributo della feature-map k sulla classe c e $f_k(x, y)$ rappresenta la feature-map k in posizione spaziale (x, y). Le feature-map di cui si parla sono estratte dall'ultimo layer convoluzionale della rete.

Ho estratto due immagini dal dataset *Imagenette* a bassa risoluzione (160x160): la prima di classe *tench* e la seconda di classe *English springer* e valutato la heat-map calcolata attraverso la suddetta formula.

L'output di CAM è il seguente:

Figure 11: CAM per l'immagine di classe English Springer