



# Deep Learning & Application's Project

Zedda Luca Fadda Massimiliano



**01** Analisi componenti base

**02** Residual networks

**03** Analisi Dataset

**04** Iperparametri e scelte progettuali

**05** Iperparametri e scelte progettuali

**06** Iperparametri e scelte progettuali



# **[01]Analisi componenti base**

# Componenti di base: Layer di convoluzione

I layer di convoluzione sono delle componenti fondamentali delle CNN, tramite essi vengono estratte delle features dai dati forniti in input applicando un set di filtri, generalmente l'insieme delle features estratte da un layer di convoluzione vengono chiamate feature maps.

I principali parametri sono

- Kernel size: Dimensione dei kernel applicati
- Stride: Il passo secondo il quale i filtri si spostano
- Padding: Per definire un possibile padding ai dati in input

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

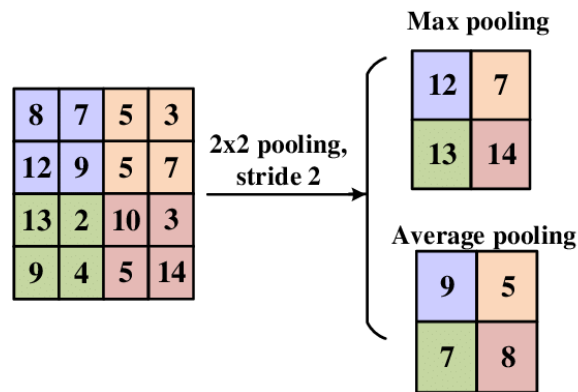
Convolved  
Feature

# Componenti di base: Pooling layer

Il pooling layer è una componente che serve a ridurre la dimensione dei dati in ingresso, mantenendo le informazioni più importanti. Ciò è utile per ridurre il numero di parametri della rete e ridurre l'overfitting. Il pooling può essere effettuato utilizzando diverse strategie, come la max o media.

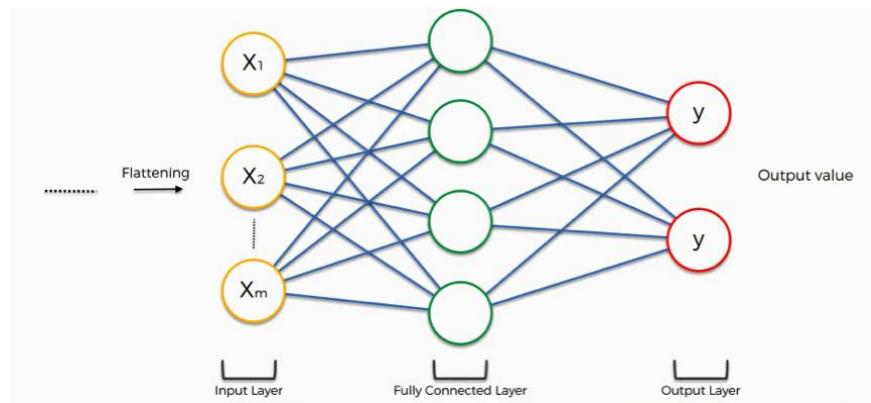
I principali parametri sono

- Kernel size: Dimensione del kernel applicato
- Stride: Il passo secondo il quale i filtri si spostano



# Componenti di base: Fully connected layer

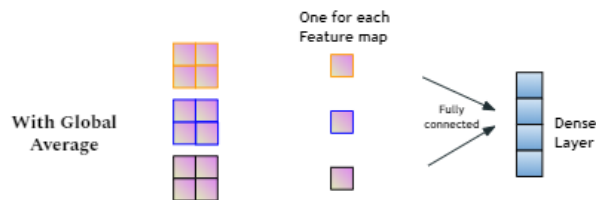
Il fully connected layer è l'ultimo strato di una CNN, che utilizza un gran numero di neuroni connessi a tutti gli elementi dell'ultimo feature map generato dal precedente strato per effettuare la classificazione finale e decidere la classe delle immagini in ingresso.



# Componenti di base: Global pooling layer

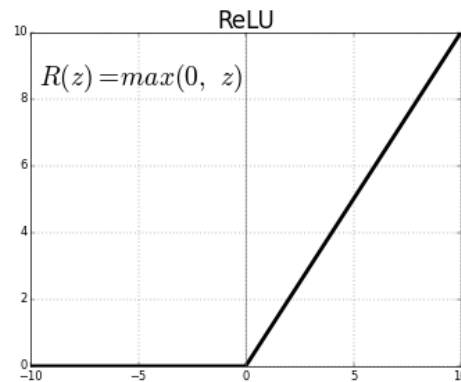
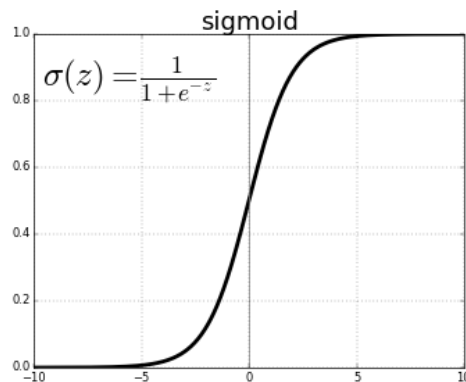
Il Global Pooling layer è un tipo di pooling utilizzato in alcune reti neurali, soprattutto in quelle utilizzate per il riconoscimento di immagini. Il suo scopo è quello di ridurre la dimensione delle feature maps generati dai precedenti strati di convolution, mantenendo le informazioni più importanti. Il Global Pooling è diverso dal pooling tradizionale, poiché esso riduce tutte le dimensioni delle feature maps in un unico valore, invece di ridurre solo alcune dimensioni

. Questo è fatto mediante l'utilizzo di una funzione di aggregazione, come la media o la massima, su tutti gli elementi delle feature maps. Il Global Pooling Layer viene solitamente inserito alla fine della rete, prima del Fully Connected layer, al fine di fornire una rappresentazione più compatta e densa dei dati in ingresso alla rete.



# Componenti di base: Activation layer

L'activation layer è una componente che applica una funzione di attivazione alle uscite del layer precedente. La funzione di attivazione è utilizzata per introdurre una non linearità nella rete, permettendo al modello di apprendere relazioni più complesse tra i dati di input e di output. Le funzioni di attivazione più comuni sono la ReLU, sigmoid, ...

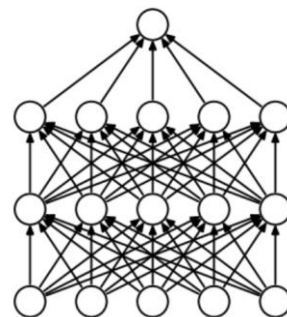




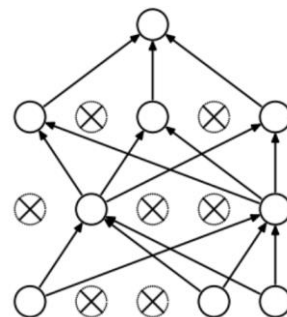
# Componenti di base: Dropout layer

Un dropout layer può essere applicato alle feature maps generati dai precedenti layer. Il processo consiste nel "scartare" casualmente alcuni elementi delle feature maps durante l'addestramento, evitando che la rete si "appoggi" troppo su una singola feature map o gruppo di feature maps.

Ciò significa che alcuni elementi dei feature maps non partecipano alla propagazione del gradiente durante la fase di addestramento, aumentando la robustezza della rete e favorendo una maggiore generalizzazione del modello. Il Dropout Layer può essere inserito tra i vari strati di una CNN, stabilendo una percentuale di elementi dei feature maps da ignorare durante una passata.



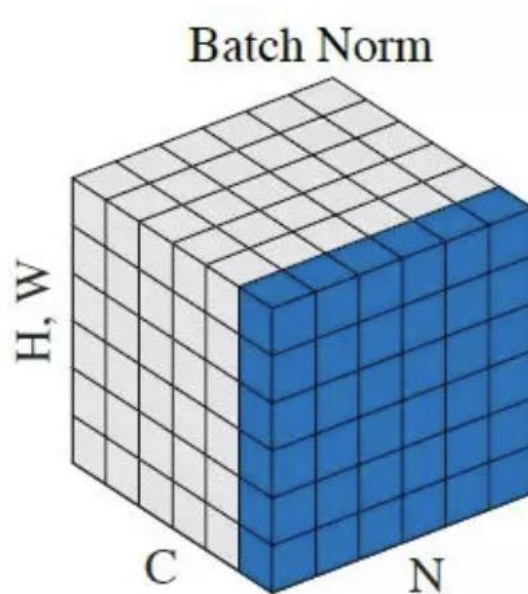
(a) Standard Neural Net



(b) After applying dropout.

# Componenti di base: Batch Norm layer

Il batch normalization è una tecnica utilizzata per migliorare l'addestramento di una rete neurale. Il suo scopo è quello di normalizzare i valori di attivazione dei neuroni in una determinata camada, al fine di rendere più stabile e veloce l'addestramento della rete.



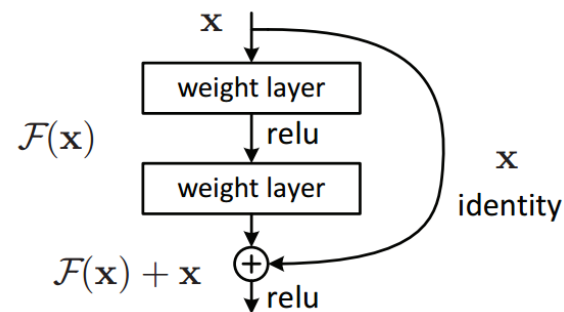


## **[02]Residual networks**

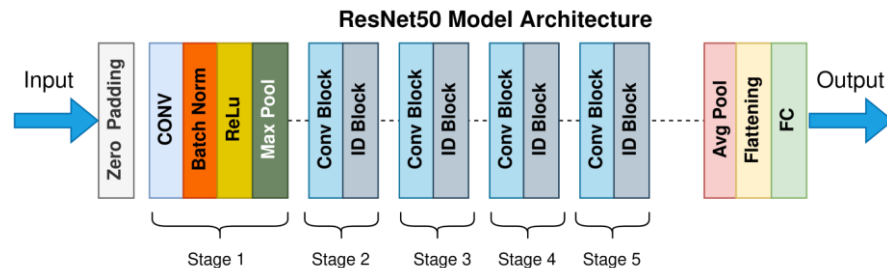
# Residual Networks

Una delle sfide principali nell'addestramento di una rete neurale profonda è il fenomeno dell' exploding o del vanishing gradient. Il gradiente è il valore che indica la pendenza della loss rispetto ai pesi della rete.

Durante l'addestramento, i pesi vengono aggiornati in base al gradiente per ridurre la funzione di loss. Tuttavia, se i pesi diventano troppo grandi, il gradiente può diventare troppo piccolo, causando un rallentamento dell'apprendimento o addirittura un arresto.



# Residual Networks



Le ResNet hanno introdotto una soluzione a questo problema, utilizzando una costruzione chiamata "skip connection". Invece di far passare tutto il segnale attraverso una pila di layer, una porzione del segnale originale viene aggiunta direttamente all'output di un layer, bypassando così gli strati intermedi.

Ciò consente di mantenere il segnale originale intatto e di evitare che si perda troppa informazione nei livelli intermedi, riducendo il rischio di exploding o vanishing gradient.

# Quale ResNet e perchè

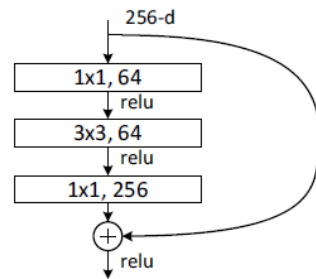
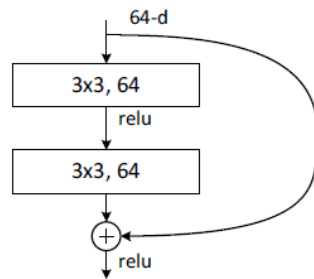
Le ResNet vengono introdotte nel 2015 con le versioni a 18 e 34 layer, successivamente nel 2016 vengono introdotte le versioni a 50, 101 e 151 layer. La differenza principale che distingue i diversi modelli è il numero di layer. Ma non solo.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$

Come possibile osservare dalla figura esempio ciò che varia dalle architetture con 50 layers o più rispetto alle altre è la composizione degli stessi layer ovvero l'utilizzo dei bottleneck layers

# Quale ResNet e perchè: Bottleneck layers

Lo scopo dei bottleneck layers è quello di ridurre la complessità di un modello riducendo il numero di parametri richiesti e dunque diminuendo il rischio di overfitting. Garantendo comunque la possibilità di apprendere features complesse



Nello specifico in figura viene mostrata la differenza tra layer classico appartenente alle ResNet 18/32 e i bottleneck layer

# Schema diverse architetture di ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	$112 \times 112$	$7 \times 7, 64, \text{stride } 2$				
conv2_x	$56 \times 56$	$3 \times 3 \text{ max pool, stride } 2$				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$





## **[03]Analisi Dataset**

# Dataset

Il dataset 'Best Artworks of All Time' reperibile al link  
<https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time>

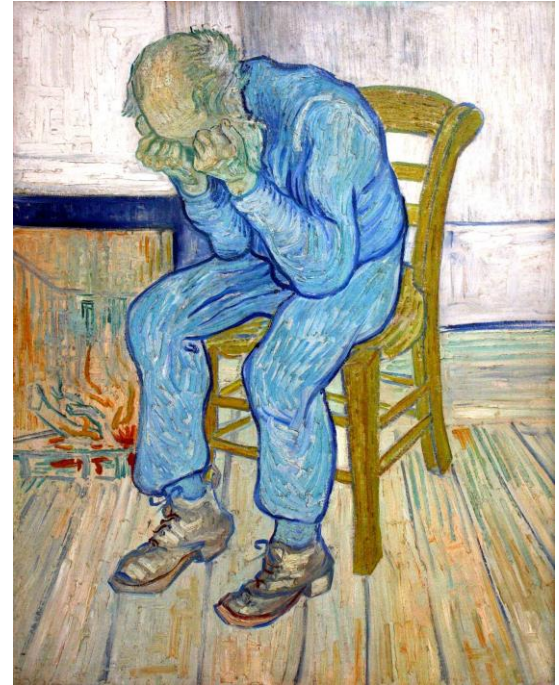
E' un dataset che comprende immagini relative alle opere di 50 artisti tra i più influenti della storia

Essendo l'obiettivo del progetto un processo di classificazione dell'autore dell'opera mediante un approccio di Deep learning è necessario predisporre l'intero processo in due macro fasi:

Addestramento e Analisi dei risultati

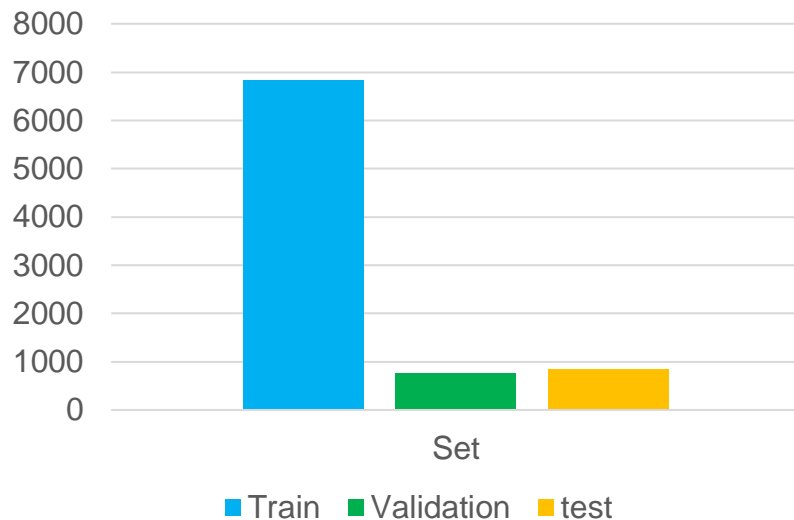
L'approccio proposto dunque è quello di dividere le immagini presenti nel dataset in 3 split seguendo le seguenti distribuzioni

80% Train set, 10% Validation set, 10% Test set



**Opera: Studenti magistrali**  
**Autore: Vincent Van Gogh**

# Dataset distribuzione record nei set



**Opera: Studenti magistrali**  
**Autore: Vincent Van Gogh**

# Data preprocessing e augmentation

Il preprocessing delle immagini consiste in una serie di passaggi che vengono eseguiti sull'input prima che venga fornito al modello di deep learning. Ad esempio, può includere la ridimensionamento delle immagini per adattarsi all'input richiesto dalla rete neurale convoluzionale (CNN), la normalizzazione dei pixel per garantire che i dati di input siano nello stesso intervallo di valori,

In particolare abbiamo applicato i seguenti step

- sottratto la media dai valori di ogni pixel e dividere il risultato per la deviazione standard. Questo porta a una distribuzione dei valori dei pixel con media 0 e deviazione standard 1. Questo per ciascun canale(RGB).
- Ridimensionamento delle immagini alla dimensione 224x224x3



**Opera: Studenti magistrali**  
**Autore: Vincent Van Gogh**

# Data preprocessing e augmentation

L'augmentation delle immagini, d'altra parte, consiste nell'applicare delle trasformazioni casuali alle immagini di input al fine di generare nuovi dati di addestramento. Questo può includere la rotazione, la traslazione, lo zoom, la riflessione e la distorsione. L'augmentation delle immagini è utile per aumentare la quantità di dati disponibili e per aiutare a prevenire l'overfitting del modello.

In particolare abbiamo applicato soltanto la seguente tecnica

- Rotazione casuale di 90/180/270 gradi



**Opera: Studenti magistrali**  
**Autore: Vincent Van Gogh**



## **[04] Iperparametri e scelte progettuali**



# Cross entropy loss

La funzione di loss cross-entropy, è una funzione di loss comunemente utilizzata per problemi di tipo supervisionato, come i problemi di classificazione. La funzione misura la dissomiglianza tra la distribuzione di probabilità prevista e la distribuzione vera, calcolando la log-likelihood negativa delle labels vere date le probabilità previste.

$$L_i = -\log \frac{e^{s_{y_i}}}{\sum_{k=1}^C e^{s_k}}$$

L'output della funzione di loss è un valore scalare che rappresenta la dissomiglianza tra la distribuzione prevista e quella vera e l'obiettivo è quello di minimizzare questo valore durante il processo di addestramento.



# Cross entropy loss

La funzione di loss cross-entropy, è una funzione di loss comunemente utilizzata per problemi di tipo supervisionato, come i problemi di classificazione. La funzione misura la dissomiglianza tra la distribuzione di probabilità prevista e la distribuzione vera, calcolando la log-likelihood negativa delle labels vere date le probabilità previste.

$$L_i = -\log \frac{e^{s_{y_i}}}{\sum_{k=1}^C e^{s_k}}$$

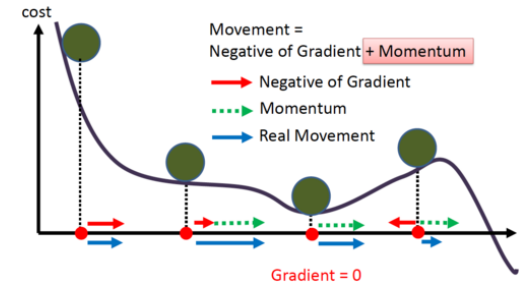
L'output della funzione di loss è un valore scalare che rappresenta la dissomiglianza tra la distribuzione prevista e quella vera e l'obiettivo è quello di minimizzare questo valore durante il processo di addestramento.



# Adam

Adam è un tecnica di ottimizzazione che fa uso di due altre soluzioni quali RMSprop e Momentum

Per **Momentum** si intende un algoritmo di ottimizzazione che incorpora i precedenti gradienti dell'update corrente dei pesi di un modello. E' possibile tenendo conto della media mobile dei gradienti, viene detto anche spesso definita come vettore velocità  $v$



Questo approccio permette al modello di convergere prima e soprattutto riduce le problematiche legate ai minimi locali

$$v^{k+1} = \beta \cdot v^k - \alpha \cdot \nabla_{\theta} L(\theta^k)$$

accumulation rate ('friction', momentum)    velocity    learning rate    Gradient of current minibatch

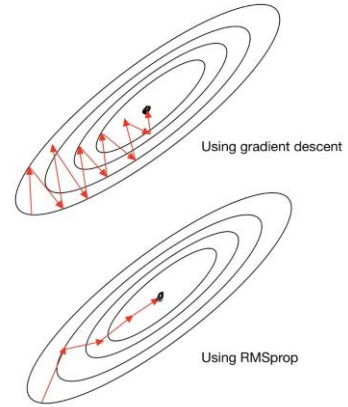
$$\theta^{k+1} = \theta^k + v^{k+1}$$

weights of model    velocity

# Adam

In **RMSprop**, per ogni peso della rete neurale, teniamo traccia della media mobile del quadrato del gradiente rispetto a quel peso. Questo ci dà una misura della varianza del gradiente rispetto a quel peso. Quindi, quando si aggiorna il peso, si divide il gradiente attuale per la radice quadrata della varianza gradiente. Questo ci dà una misura della "direzione" in cui dovremmo aggiornare il peso, ma ci permette anche di adattare la velocità di apprendimento in base alla varianza del gradiente.

Questo approccio permette al modello di convergere prima e soprattutto riduce le problematiche direzioni le quali presentano un'alta varianza



$$\mathbf{s}^{k+1} = \beta \cdot \mathbf{s}^k + (1 - \beta) [\nabla_{\theta} L \circ \nabla_{\theta} L]$$

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\nabla_{\theta} L}{\sqrt{\mathbf{s}^{k+1}} + \epsilon}$$

Element-wise multiplication

# Adam

Come accennato in precedenza Adam fa frutto di queste due soluzioni e le combina al suo interno caratteristiche presenti nelle due soluzioni

$$\mathbf{m}^{k+1} = \beta_1 \cdot \mathbf{m}^k + (1 - \beta_1) \nabla_{\theta} L(\theta^k) \quad \mathbf{v}^{k+1} = \beta_2 \cdot \mathbf{v}^k + (1 - \beta_2) [\nabla_{\theta} L(\theta^k) \circ \nabla_{\theta} L(\theta^k)]$$

Dove  $\mathbf{m}$  e  $\mathbf{v}$  sono rispettivamente media e varianza del gradiente. Questi ultimi vengono inizializzati a zero durante il primo update, dunque hanno un bias verso lo 0, il modo in cui si può correggere questa problematica è il seguente

$$\hat{\mathbf{m}}^{k+1} = \frac{\mathbf{m}^{k+1}}{1 - \beta_1^{k+1}} \quad \hat{\mathbf{v}}^{k+1} = \frac{\mathbf{v}^{k+1}}{1 - \beta_2^{k+1}}$$

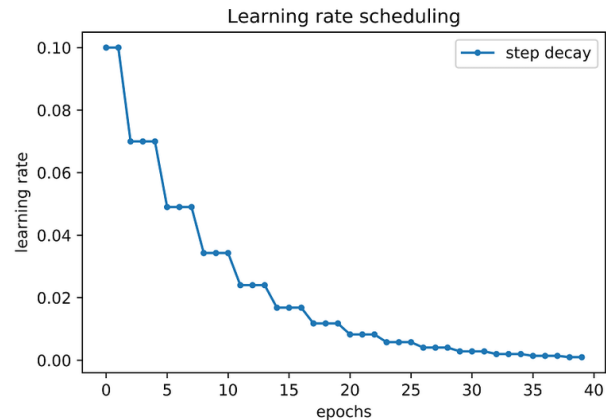
Si ottiene dunque che

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\hat{\mathbf{m}}^{k+1}}{\sqrt{\hat{\mathbf{v}}^{k+1} + \epsilon}}$$

# Learning Rate scheduler

In generale il **learning rate scheduler** permette di adattare il learning rate in modo che sia più alta all'inizio del processo di addestramento, per permettere al modello di esplorare le diverse regioni dello spazio di parametri, e poi abbassarlo gradualmente, per permettere al modello di convergere verso una soluzione ottimale.


Nel nostro caso la learning rate viene moltiplicata per un **gamma=0.7** ad ogni epoca






# Metriche utilizzate

Trovandoci di fronte a dati moderatamente sbilanciati abbiamo optato per metriche di valutazione che tengono conto dello sbilanciamento dei dati. Terremo conto della MacroAVG accuracy

La macro avg accuracy è una metrica utilizzata per valutare l'accuratezza di un modello di classificazione. La macro avg accuracy calcola la media aritmetica delle accuracy di ciascuna classe, senza tenere conto del numero di esempi per ciascuna classe.



	precision	recall	f1-score	support
Aeroplane 	0.67	0.67	0.67	3
Boat 	0.25	1.00	0.40	1
Car 	1.00	0.50	0.67	6
accuracy			0.60	10
macro avg	0.64	0.72	0.58	10
weighted avg	0.82	0.60	0.64	10



## **[05]Esperimenti svolti**



# Esperimenti svolti

**01** Addestrare ResNet50 from scratch

**02** FineTuning ResNet50

**03** FineTuning ResNet50 freeze layer 1-4

**04** FineTuning ResNet50 freeze layer 1-3

**05** Addestrare ResNet50 modificata



# Addestramento VS Fine-tuning

In sintesi, il fine-tuning è utile per adattare un modello pre-addestrato a un nuovo compito o dataset, mentre l'addestramento di un modello da zero è utile per compiti che sono molto diversi dal compito su cui è stato addestrato il modello pre-addestrato.

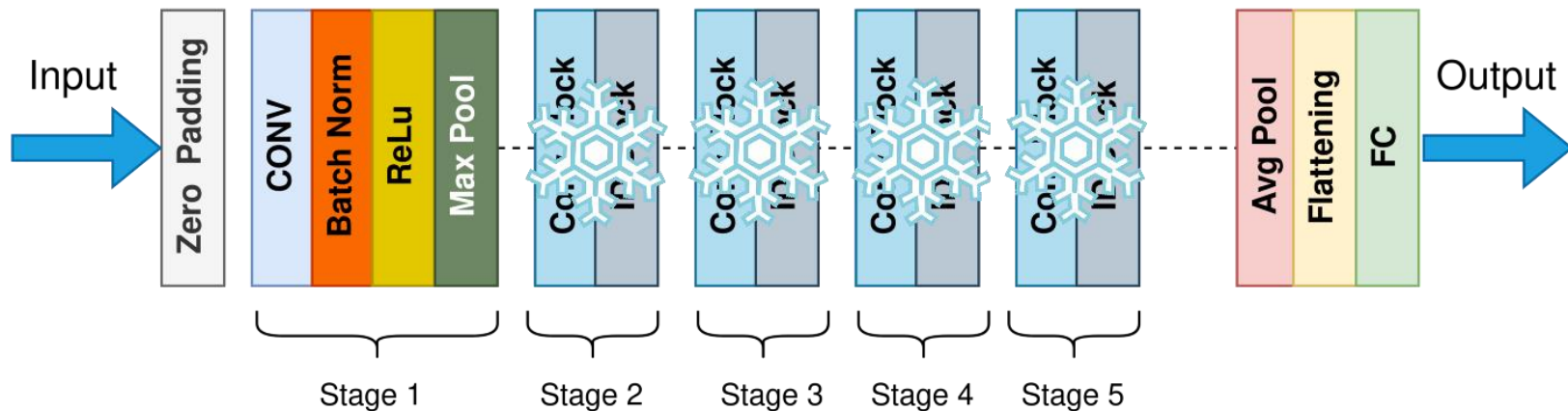
In particolare abbiamo effettuato il freezing di particolari sezioni del del modello pretrainato, permettendo solo ad alcuni layer di apprendere.

Nel nostro caso i pesi utilizzati per il finetuning sono relativi al dataset ImageNet1K  
Mentre le learning rate utilizzate per l'addestramento e il fine-tuning sono rispettivamente  $3e-3$  e  $3e-4$



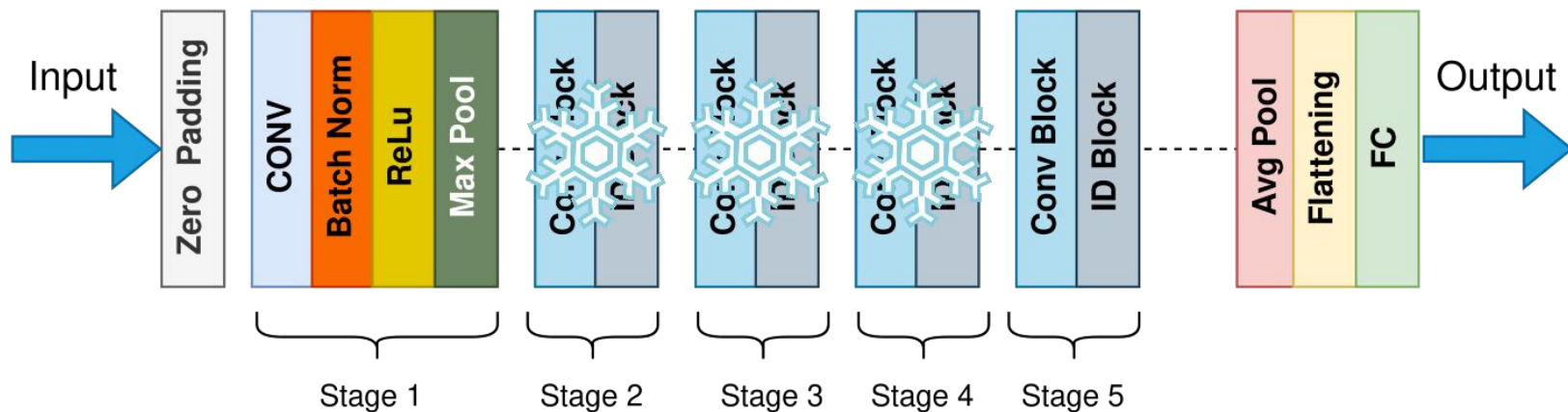
# Esperimenti freeze 1-4

ResNet50 Model Architecture



# Esperimenti freeze 1-3

ResNet50 Model Architecture





## **[06]Miglioramenti proposti**

# Cosa possiamo migliorare?

La ResNet50 risulta performare bene per l'obiettivo che ci siamo posti, però la domanda che sorge spontanea è «Possiamo fare di meglio?»

L'idee che abbiamo seguito sono le seguenti:

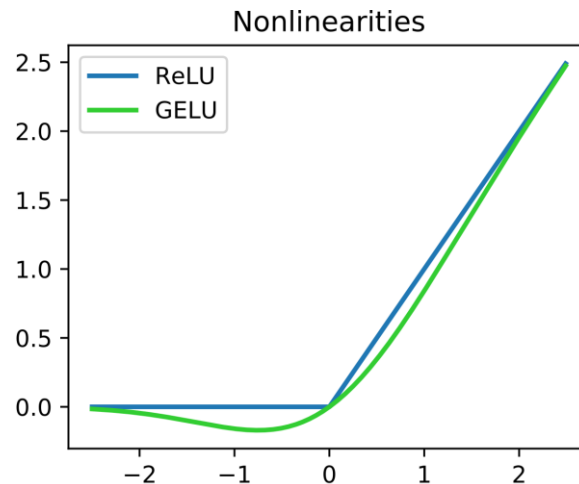
- Sostituzione funzione di attivazione RELU -> GELU
- Depth-wise convolution per i layer di convoluzione con kernel-size pari a 3
- Aggiunta di moduli di attenzione: CBAM



# Problemi ReLU

La "dying ReLU" è un problema comune nelle reti neurali che utilizzano l'attivazione ReLU. La ReLU (Rectified Linear Unit) è una funzione di attivazione popolare perché introduce non linearità nella rete e consente di risolvere i problemi di saturazione delle unità sigmoid e tanh. Tuttavia, un problema con la ReLU è che può morire, ovvero diventare costantemente zero per alcuni dei suoi neuroni.

Una possibile soluzione è l'utilizzo di funzioni di attivazione quali leaky ReLU, noi per questo caso abbiamo optato per la GELU che al giorno d'oggi è ampiamente usata all'interno di Transformer e Vision Transformer

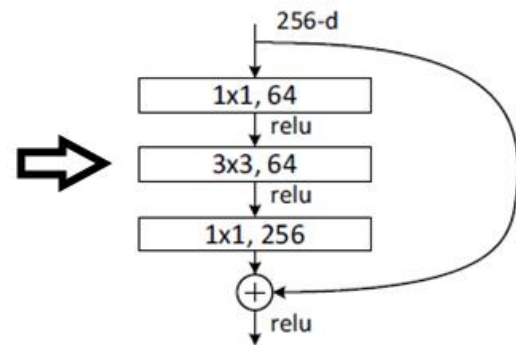


# Depthwise convolution

La depthwise convolution è un tipo di convoluzione usato in alcune reti neurali convolutionali. La convoluzione tradizionale applica un filtro a ciascun canale dell'input, mentre la depthwise convolution applica un filtro a ciascun canale indipendentemente.

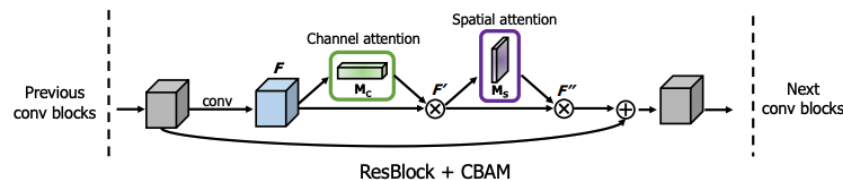
Questo tipo di convoluzione presenta alcuni vantaggi, quali:

- Permette di ridurre il numero di parametri, riducendo il rischio di overfitting
- Per ogni canale il modello è libero di apprendere pesi separati



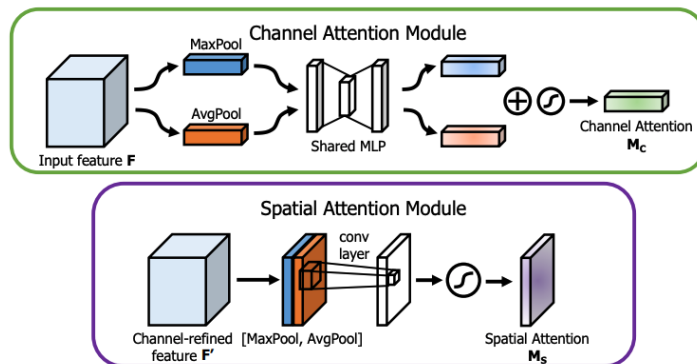
# CBAM (Convolutional Block Attention Module)

Il CBAM è un modulo di attenzione utilizzato in alcune reti neurali convoluzionali per migliorare la capacità di attenzione della rete. Il modulo CBAM combina due tipi di attenzione: attenzione alla posizione (spatial attention) e attenzione al canale (channel attention).



In generale CBAM consente di effettuare una selezione degli elementi più importanti in una feature map, consentendo una maggiore efficienza nell'elaborazione e migliorando le performance della rete.

# CBAM funzionamento



$$F' = M_c(F) \otimes F,$$

$$F'' = M_s(F') \otimes F',$$

$$\begin{aligned} M_c(F) &= \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F))) \\ &= \sigma(W_1(W_0(F_{avg}^c)) + W_1(W_0(F_{max}^c))), \end{aligned}$$

$$\begin{aligned} M_s(F) &= \sigma(f^{7 \times 7}([AvgPool(F); MaxPool(F)])) \\ &= \sigma(f^{7 \times 7}([F_{avg}^s; F_{max}^s])), \end{aligned}$$