

# **DLA Parte 1**

Classificazione binaria - Riconoscimento di deep fake

# Il problema

Il problema affrontato in questo progetto è stato quello della recognition di **deep fake**, il tutto è stato ricondotto a un problema di **classificazione binaria**.

Per affrontare il problema è stato scelto di utilizzare il linguaggio python, con l'aggiunta delle librerie pytorch, pandas, pillow, scikit-learn.

Per raggiungere l'obiettivo è stato usato il dataset "**Open Forensics**", composto in totale da circa 115000 immagini, suddivise in training set, validation set e test set.

In ogni immagine erano contenuti uno o più volti, e per ognuno di questi veniva fornito il bounding box e il contorno, con etichetta di classe corrispondente.

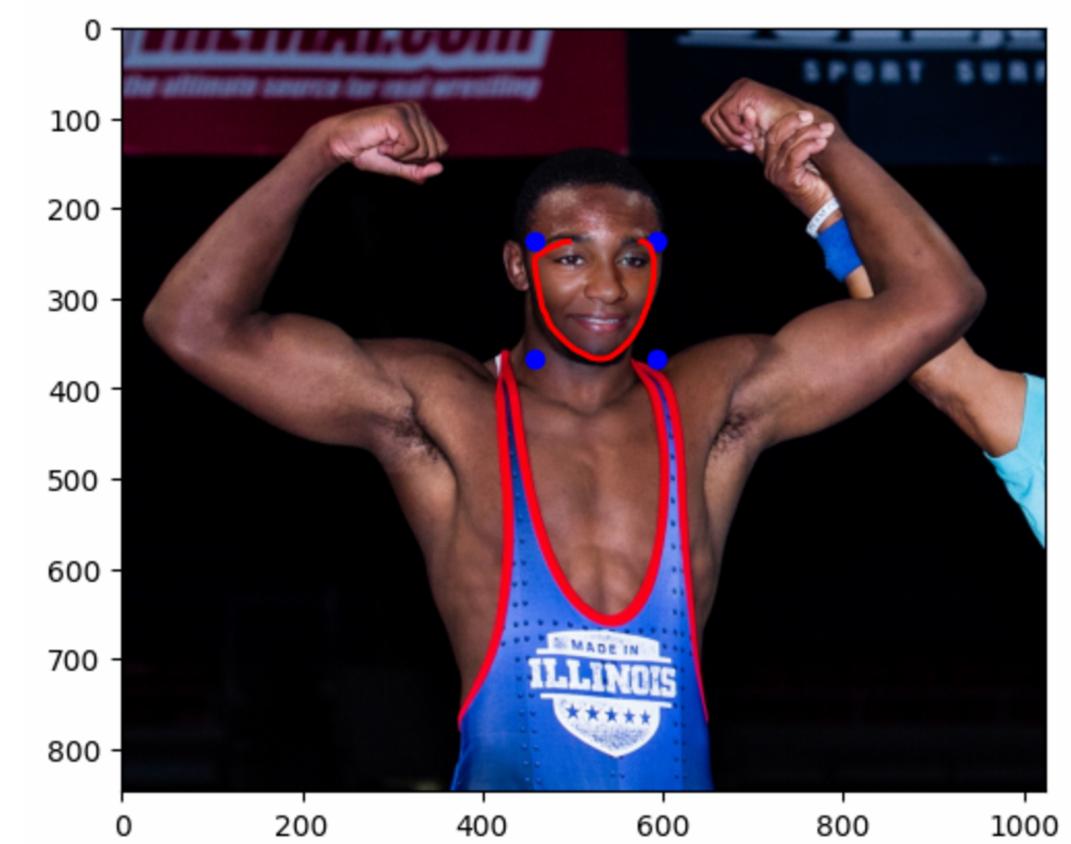


Figura 1: campione del dataset, i punti blu indicano gli estremi del BBox, in rosso il contorno del viso

# Dataset

Tutto il dataset, per ogni insieme suddiviso era rappresentato dalle immagini + rispettivi json contenenti tutte le informazioni dell'immagine.

## 3 json (train, test val) contenenti dizionari

**su:**

- Informazioni sulle etichette
- Informazioni generali sulle singole immagini
- Informazioni sui volti per le singole immagini

## Informazioni generali sull'immagine

```
{  
    id: 0,  
    file_name: 9...8f.jpg,  
    width: 1024,  
    height: 847  
}
```

## Informazioni su ogni volto presente nelle immagini

```
{  
    id: 0,  
    image_id: 0,  
    is_crowd: 0,  
    area: 13661,  
    category_id: 1,  
    bbox: [x1,x2,y1,y2],  
    segmentation: [496,...,237]  
}
```

# Elaborazione Dataset

Per questioni di risorse di calcolo è stato scelto di utilizzare solo il Training Set originale, scartando insiemi di valutazione e testing forniti dagli autori. A partire dall'insieme di training è stato sviluppato l'intero progetto, applicando specifiche elaborazioni per rendere i passaggi successivi più immediati.



1

## Estrazione informazioni dai json

Lo scopo di questa fase è stato quello di combinare i dati delle immagini e dei volti, in modo tale da tenere in un unico punto solo le informazioni utili al problema di classificazione.

Le informazioni successivamente sono state riorganizzate in un CSV.



2

## Ritaglio dei visi dalle immagini e salvataggio

A partire dal csv sono stati ritagliati tutti i visi dalle immagini e salvati in una cartella a parte, questo procedimento ha permesso di risparmiare tempo nelle fasi successive in quanto le immagini dei singoli volti sono più leggere rispetto alle intere immagini.



## Informazioni nel CSV finale

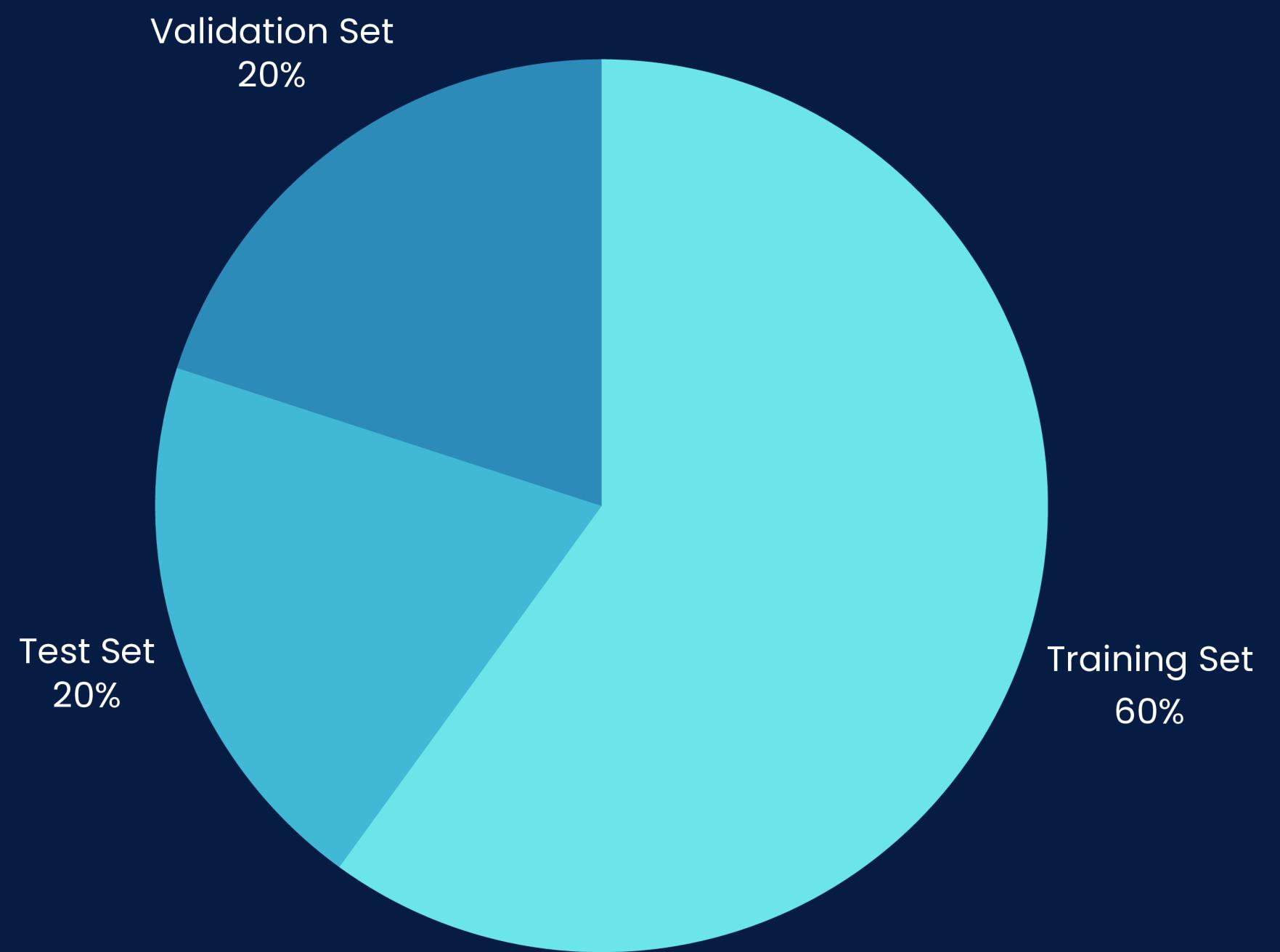
id\_x  
image\_id  
category\_id  
bbox  
file\_name  
cutted\_name

**In totale  
sono stati estratti  
circa 150.000 volti**

# Suddivisione Dataset

Dopo le elaborazioni necessarie il dataset è stato suddiviso in Training Set, Validation Set, Test Set.

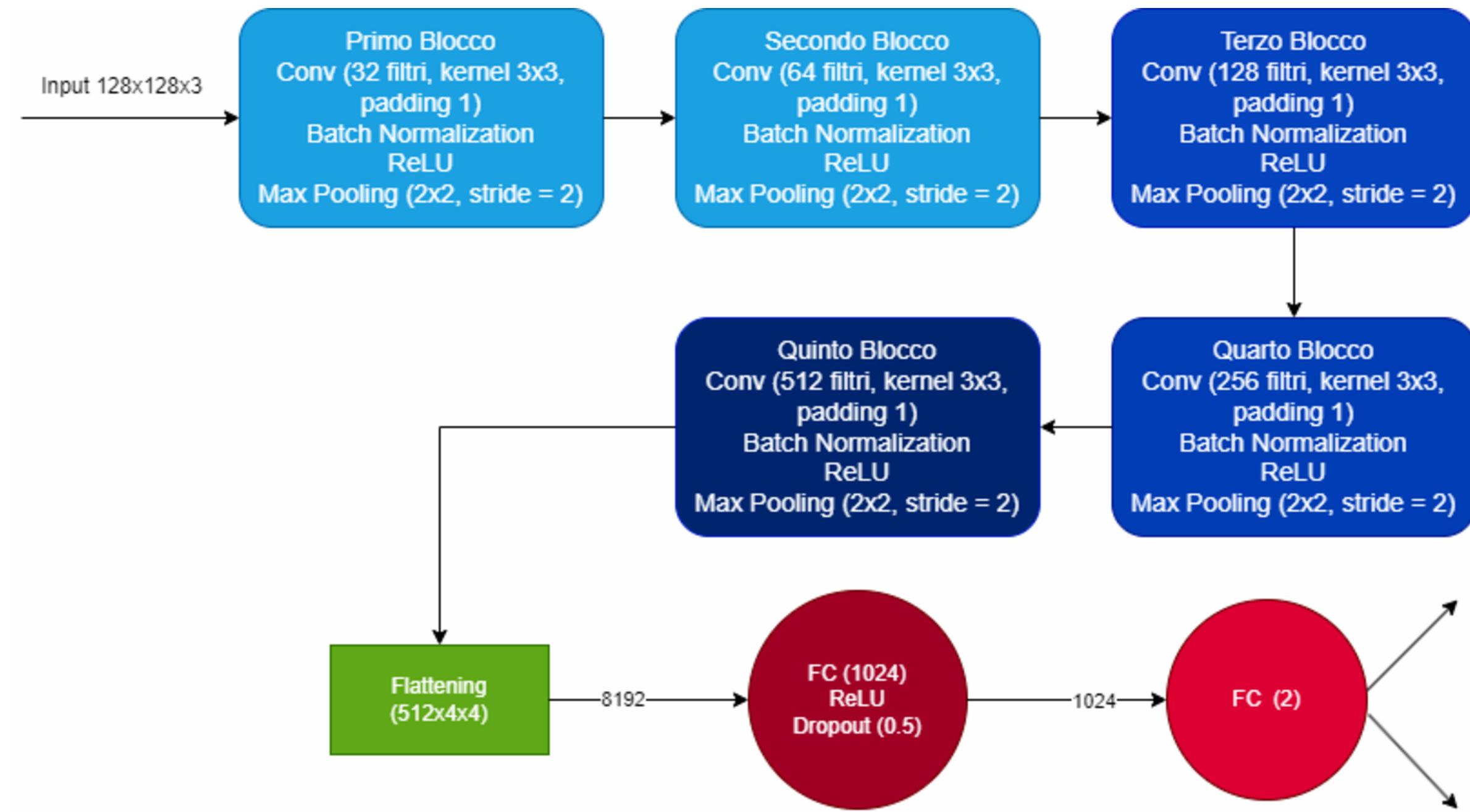
I dati della suddivisione sono stati salvati, così da utilizzare gli stessi dati per i diversi addestramenti.



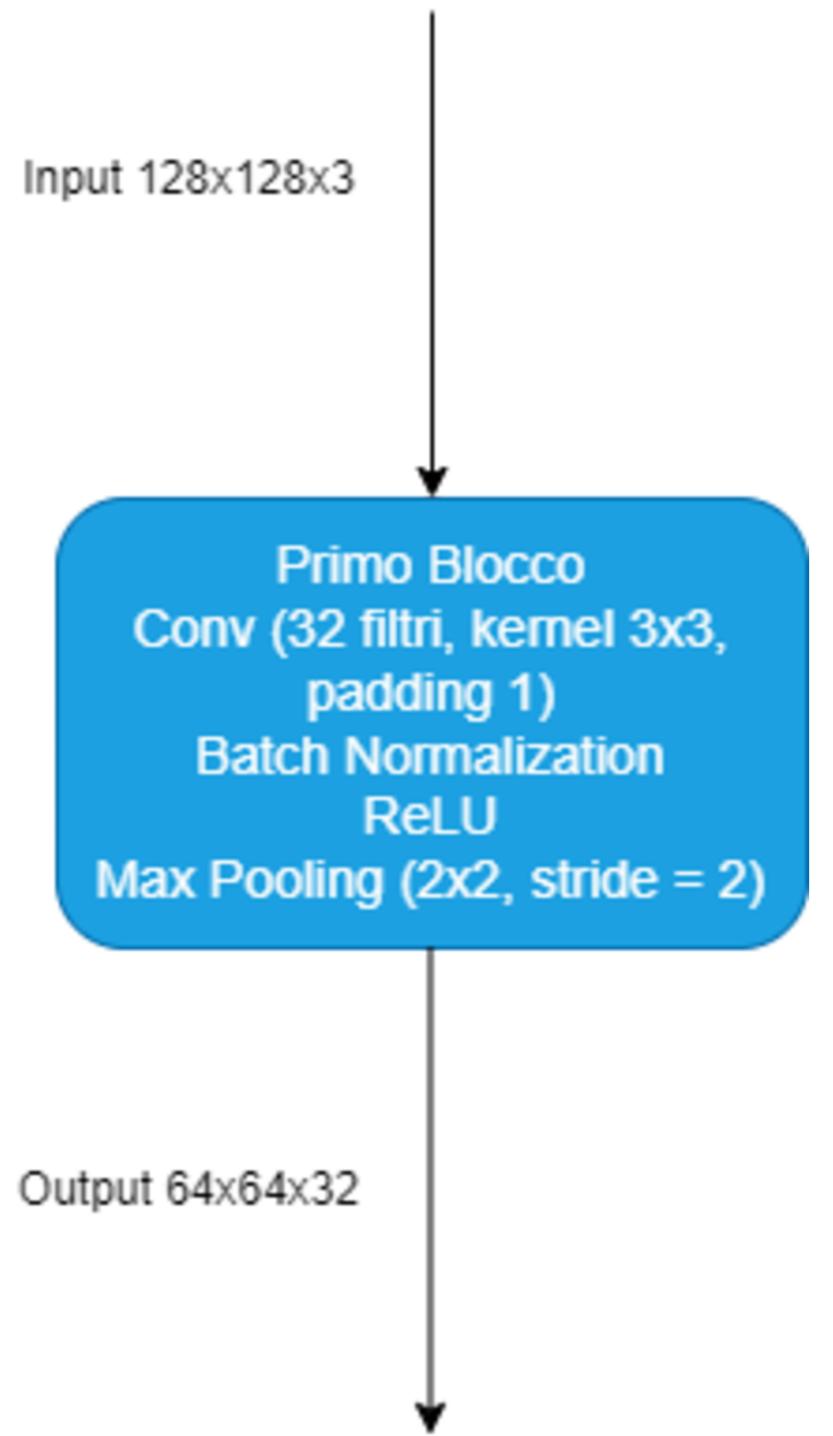
# Scelta dei modelli

Una volta stabilito il dataset si è passato alla scelta dei modelli, le specifiche richiedevano di creare una rete custom, e, di eseguire un addestramento anche su una rete pre-addestrata.

- **CNN custom:**
  - Addestrata da 0
- **MobileNet V2:**
  - Pre-addestrata su Image Net
  - Transfer Learning
  - <https://arxiv.org/abs/1801.04381v4>

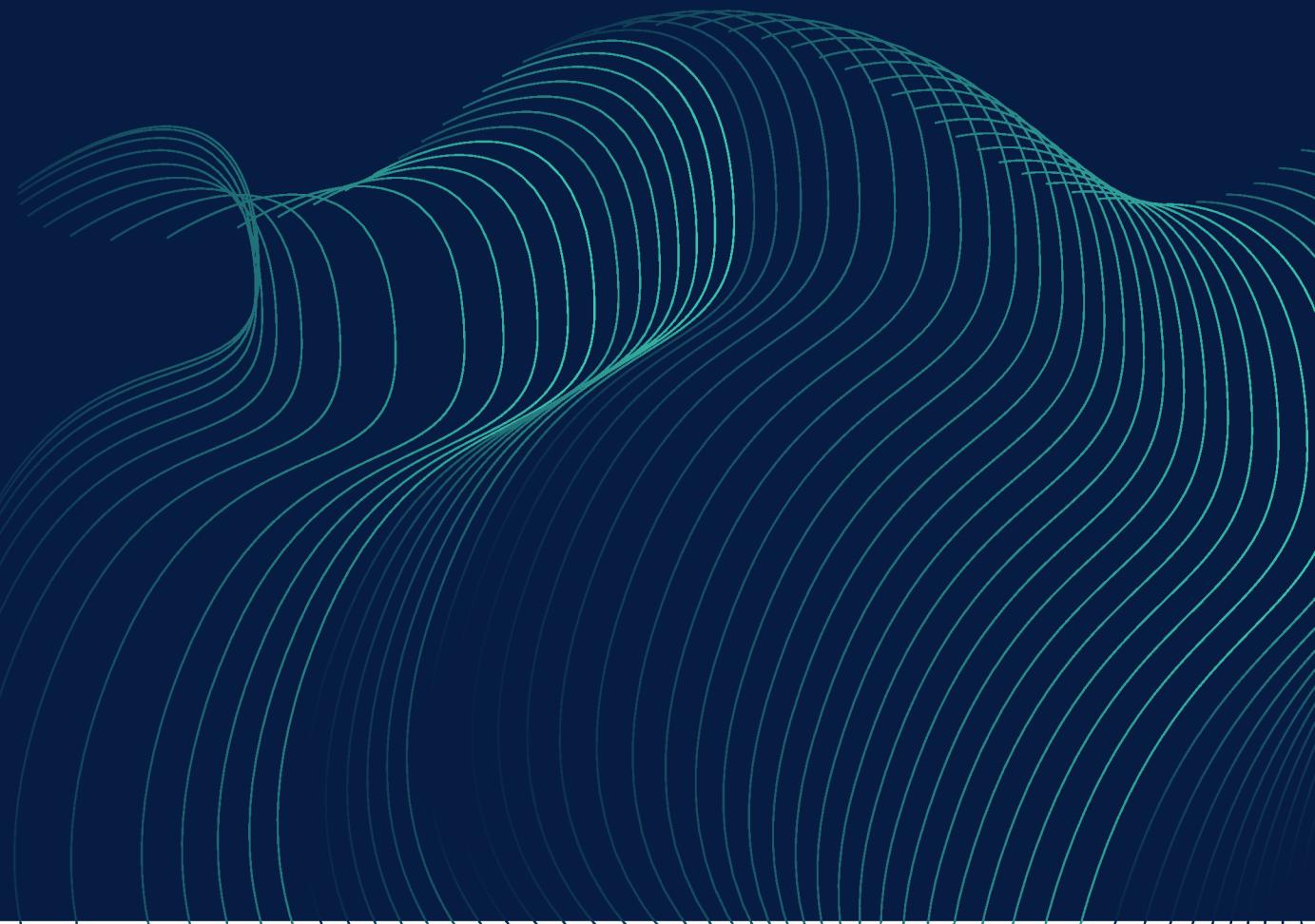


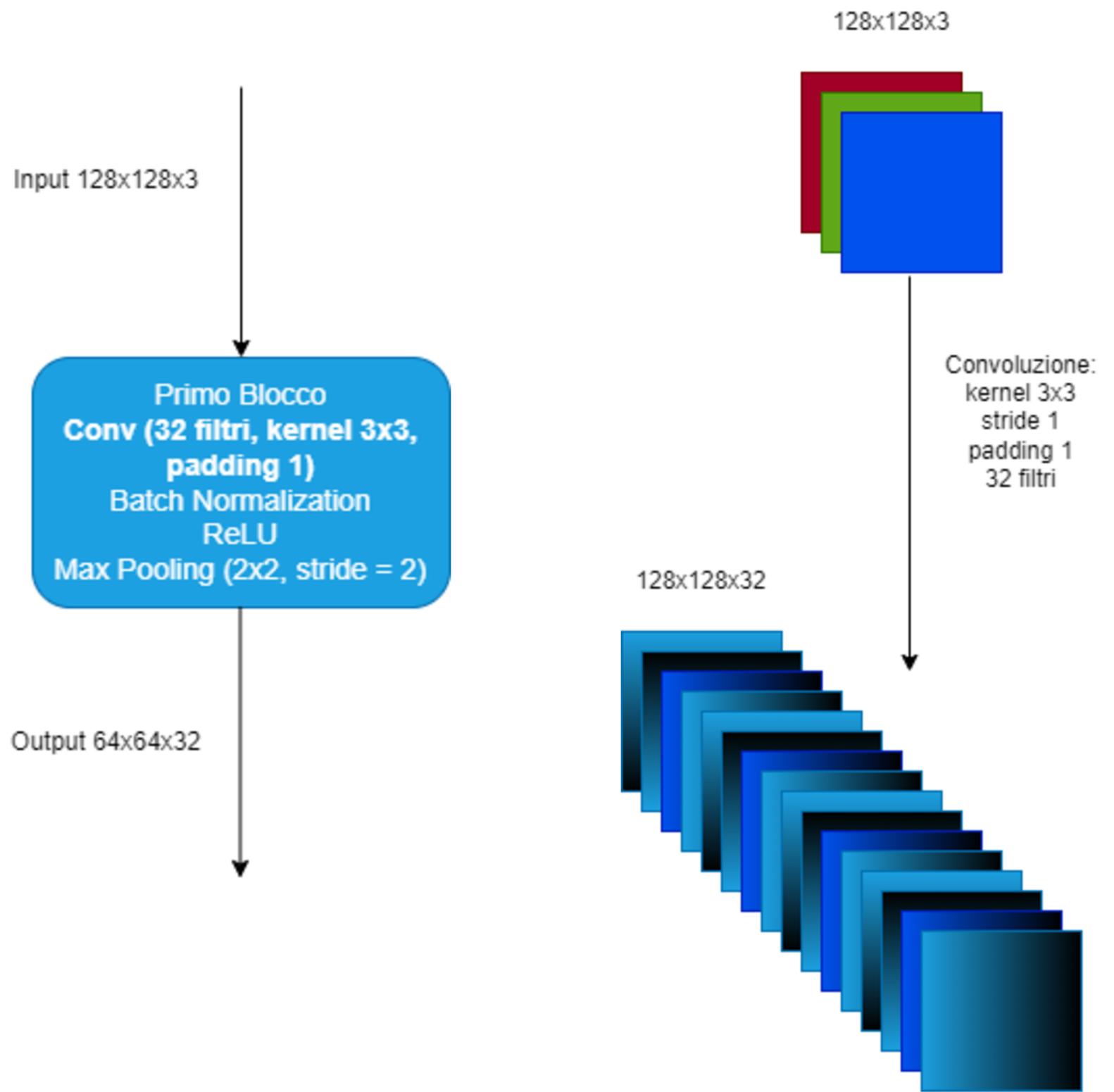
# Rete Custom



# Primo blocco

- Conv: aumenta la profondità dei canali a 32, mantenendo la risoluzione spaziale.
- Batch Normalization 1: stabilizza l'output della convoluzione.
- ReLU: introduce non-linearietà.
- Max Pooling 1: riduce la risoluzione spaziale a 64x64

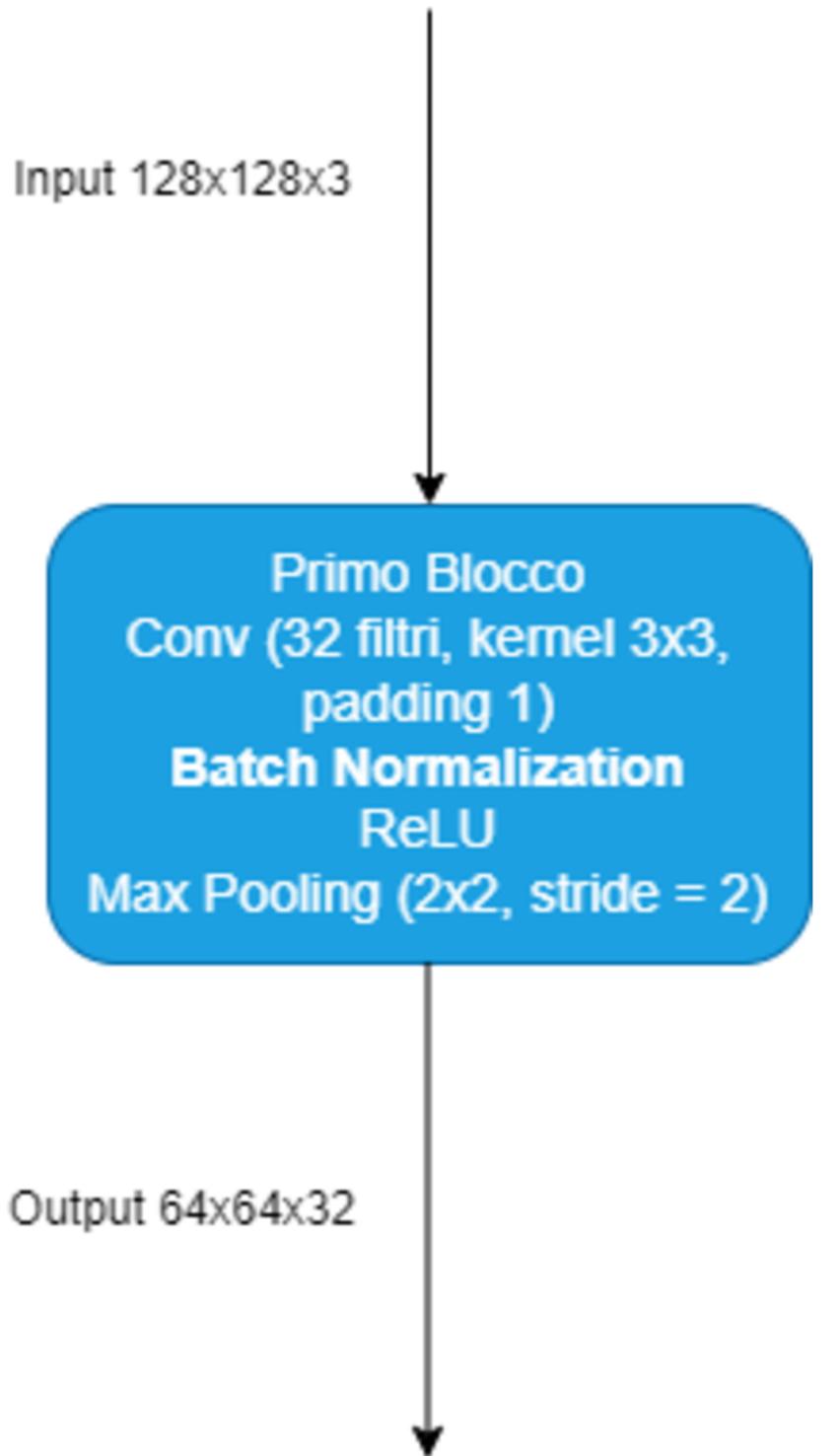




# Convoluzione

- Input: Immagini con dimensioni  $128 \times 128 \times 3$
- Output:  $128 \times 128 \times 32$
- Il padding permette di non cambiare le dimensioni spaziali
- La convoluzione è classica quindi, viene applicato lo stesso filtro  $3 \times 3$  su i 3 canali contemporaneamente, tutto questo per 32 filtri
- Il risultato sono 32 mappature dell'immagine

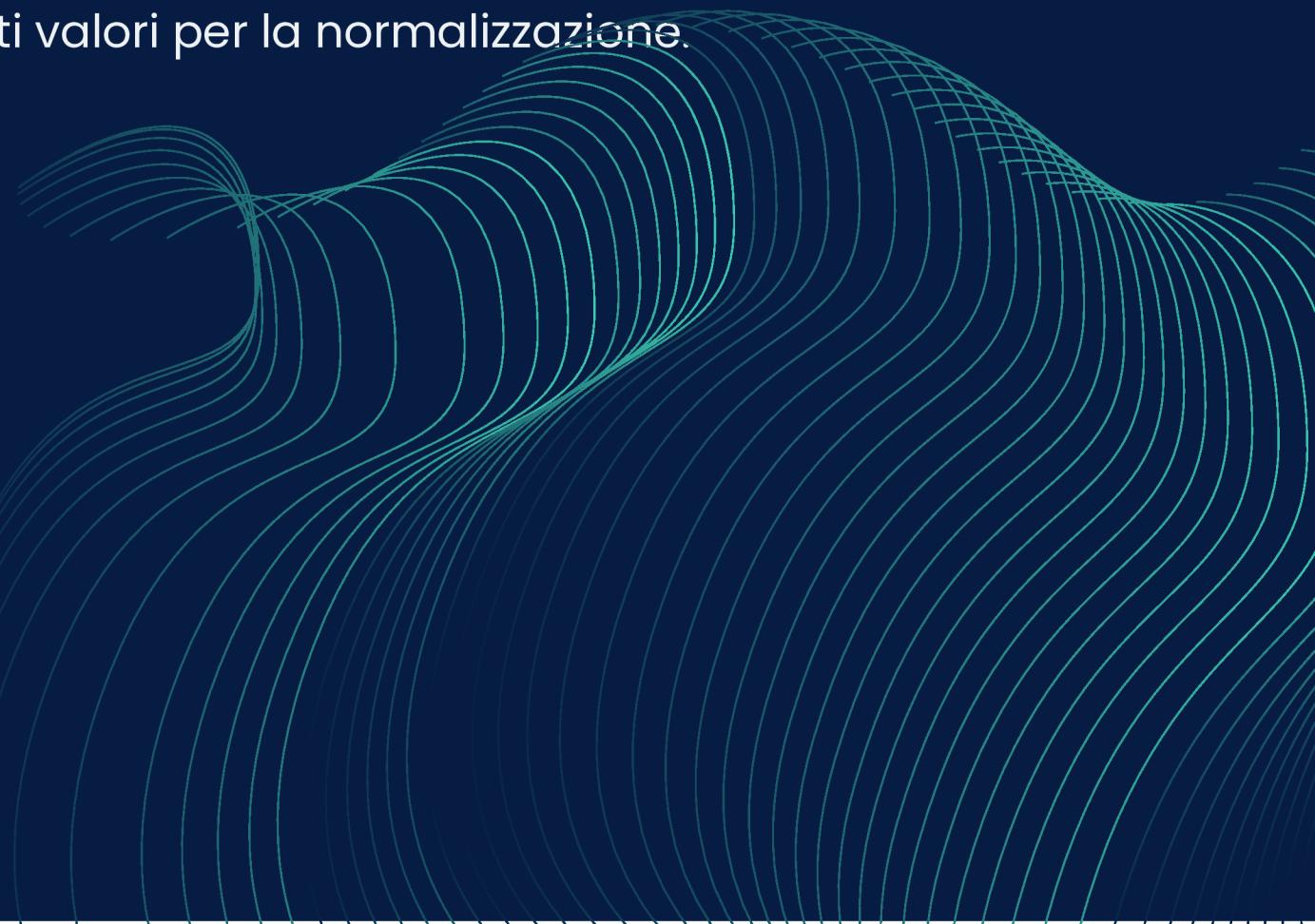




# Batch Normalization

Normalizza i valori in input per minimizzare i cambiamenti nella distribuzione.

Calcola la media e la deviazione standard basandosi sui valori di quella feature in tutto il batch corrente di dati, successivamente utilizza questi valori per la normalizzazione.



# Batch Normalization

Formula normalizzazione

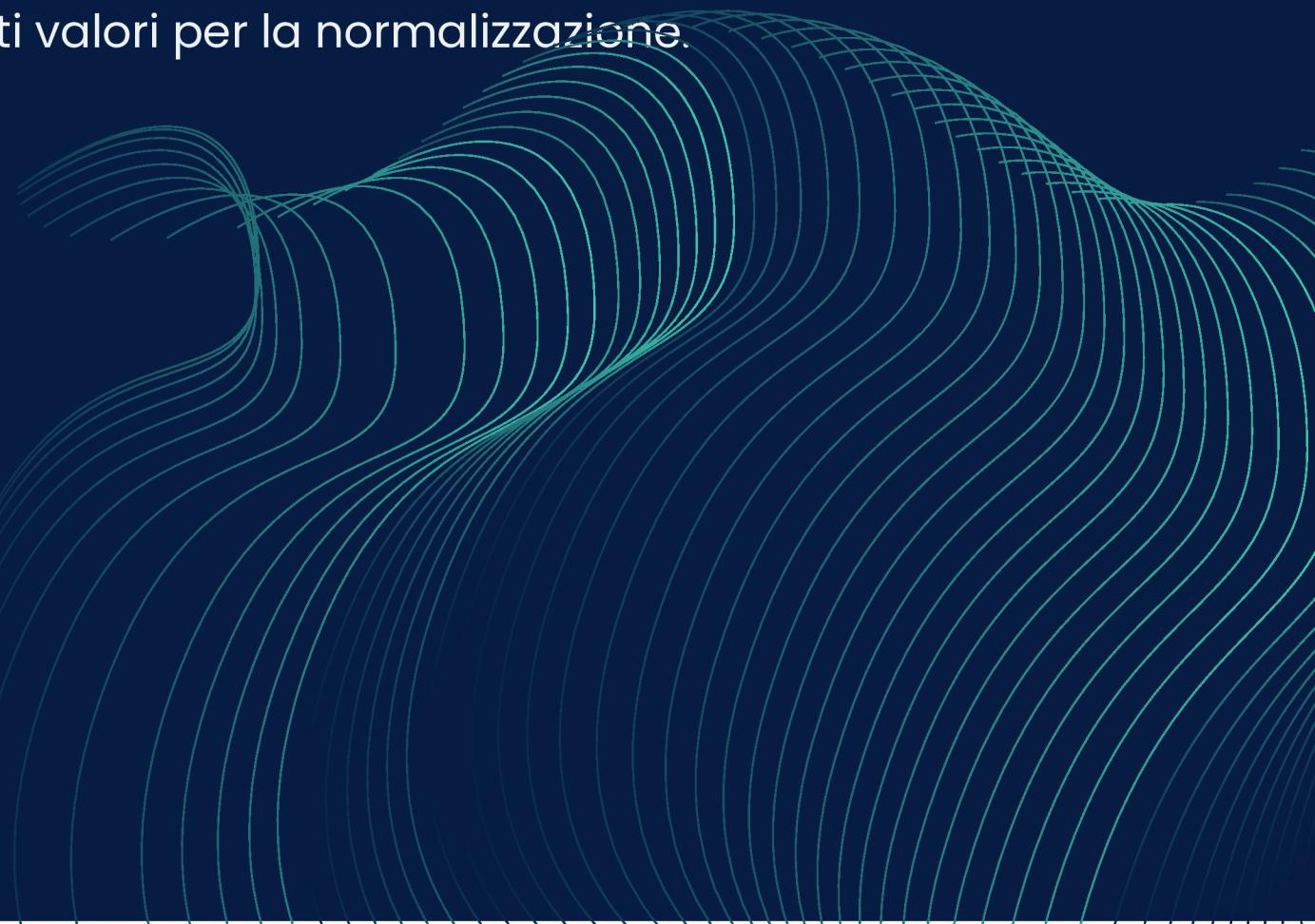
$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2} + \epsilon}$$

Dove:

- $x_i$  : input feature i
- $\mu_B$  : media del batch
- $\sigma_B^2$  : varianza del batch
- $\epsilon$  : piccola costante per evitare divisioni per zero

Normalizza i valori in input per minimizzare i cambiamenti nella distribuzione.

Calcola la media e la deviazione standard basandosi sui valori di quella feature in tutto il batch corrente di dati, successivamente utilizza questi valori per la normalizzazione.



# Batch Normalization

Scaling e shift

$$y_i = \gamma \hat{x}_i + \beta$$

Dove:

$-\gamma$  : il fattore di scala

$-\beta$  : termine di traslazione

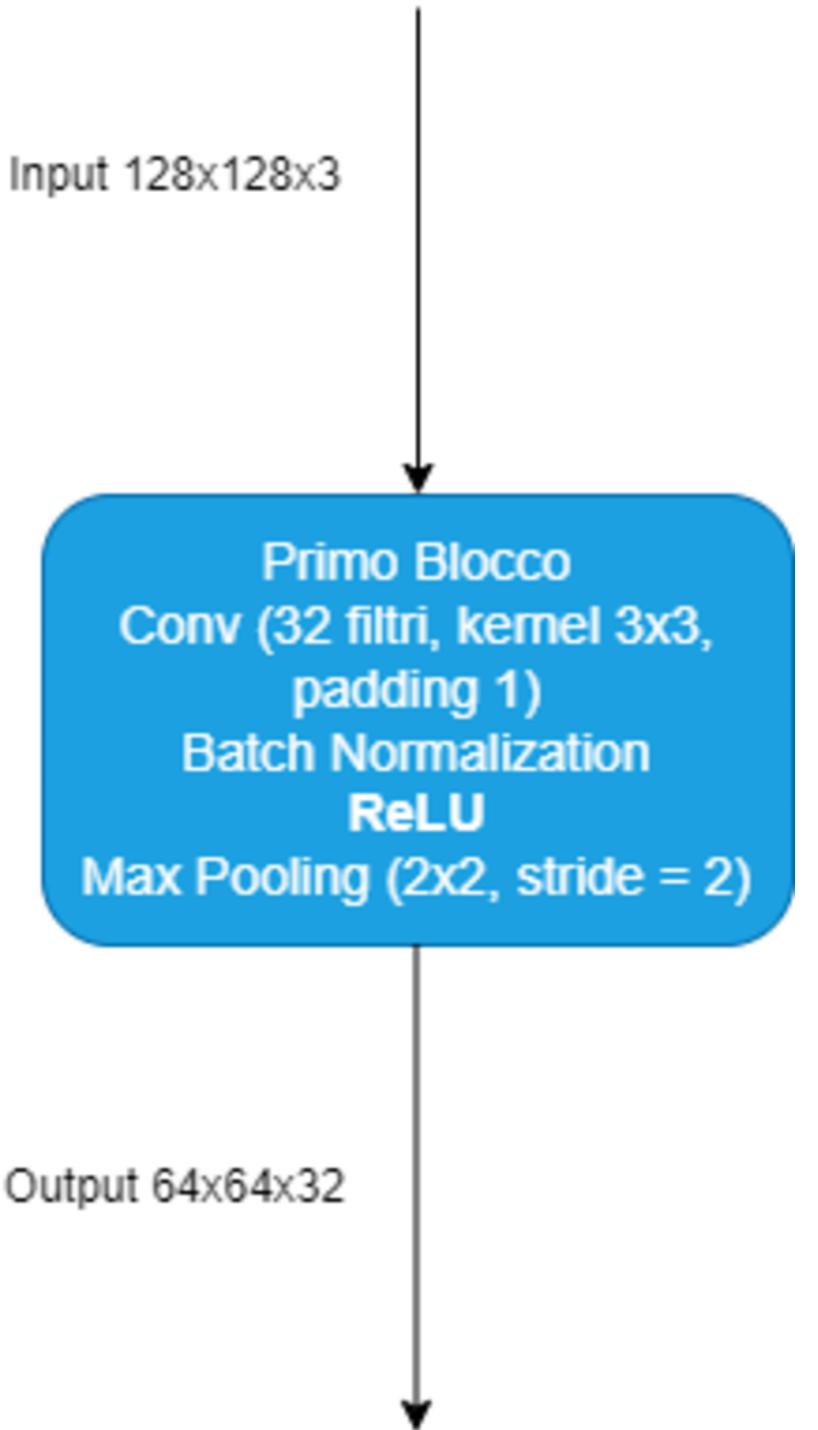
i due parametri vengono aggiornati con l'addestramento

Normalizza i valori in input per minimizzare i cambiamenti nella distribuzione.

Calcola la media e la deviazione standard basandosi sui valori di quella feature in tutto il batch corrente di dati, successivamente utilizza questi valori per la normalizzazione.

Il fattore di scala modula l'ampiezza dei dati normalizzati, il fattore di traslazione sposta la distribuzione.



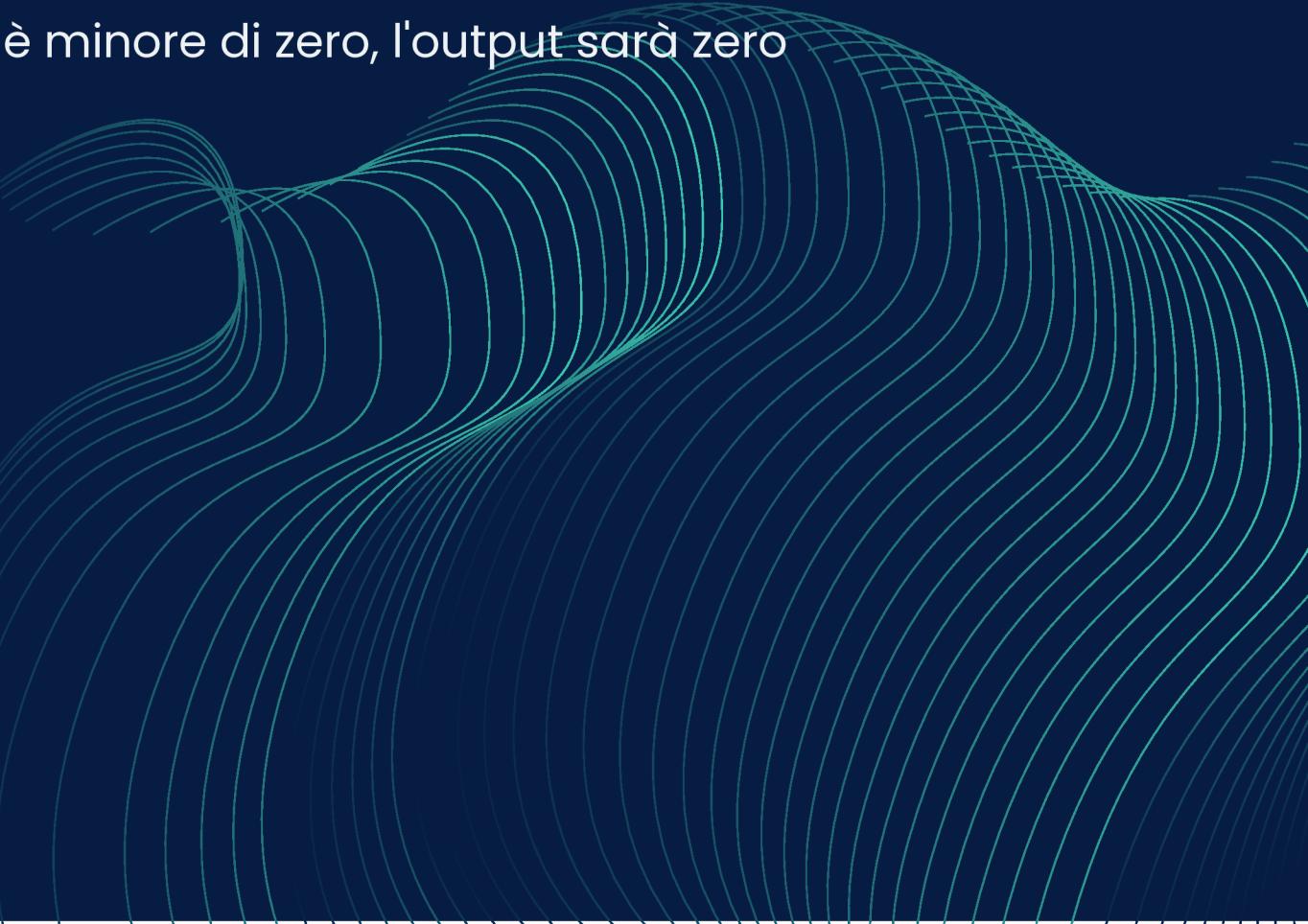


# ReLU

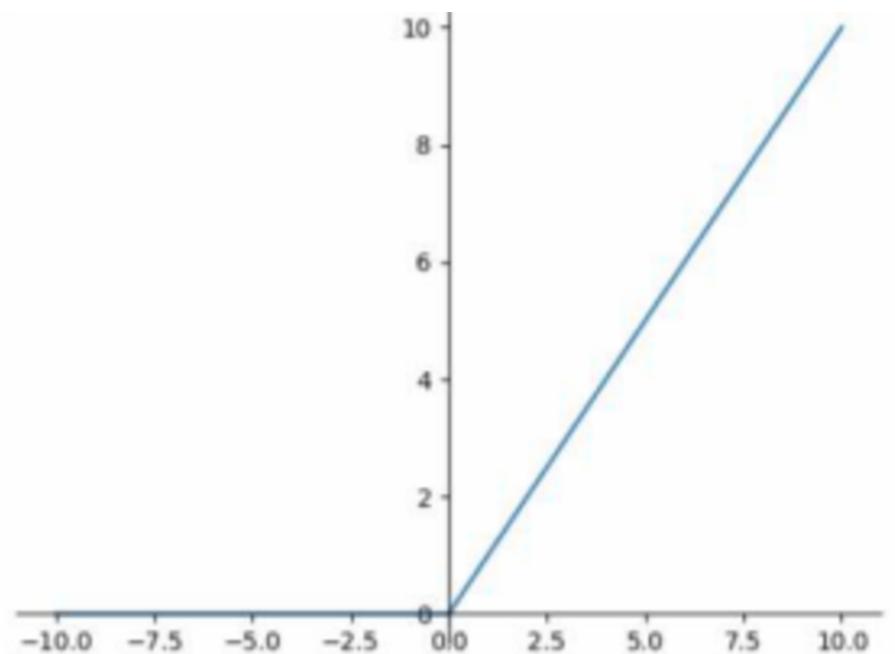
La rectified linear unit è una funzione di attivazione.

La ReLU opera su ogni input singolo, producendo zero per ogni valore negativo e lasciando invariato ogni valore non negativo.

In termini matematici, se l'input è maggiore o uguale a zero, l'output sarà lo stesso dell'input; se l'input è minore di zero, l'output sarà zero



$$f(x) = \max(0, x)$$



Ex:

$$x = [1, -0.5, 3, -2, 0]$$

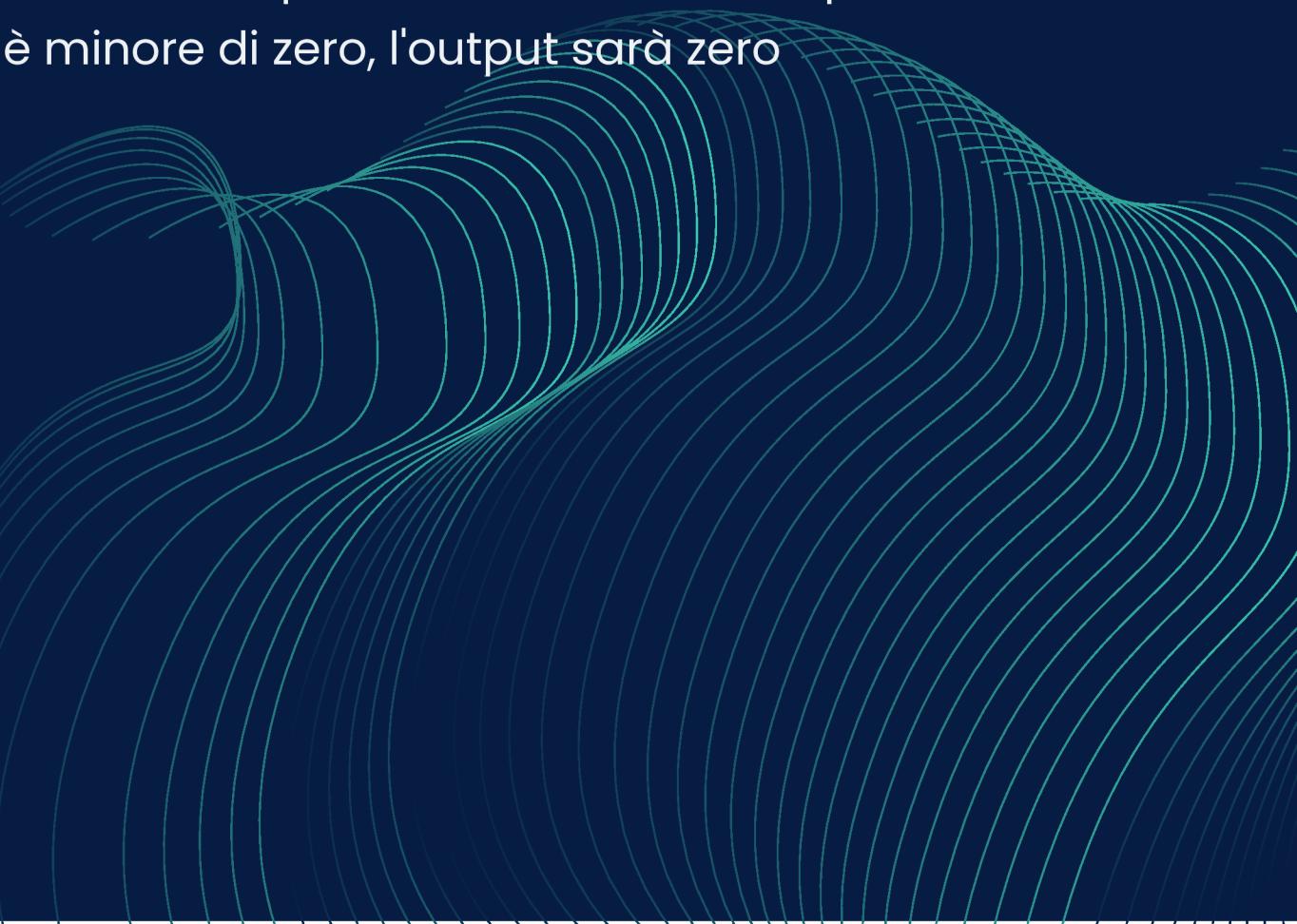
$$\text{ReLU}(x) = [1, 0, 3, 0, 0]$$

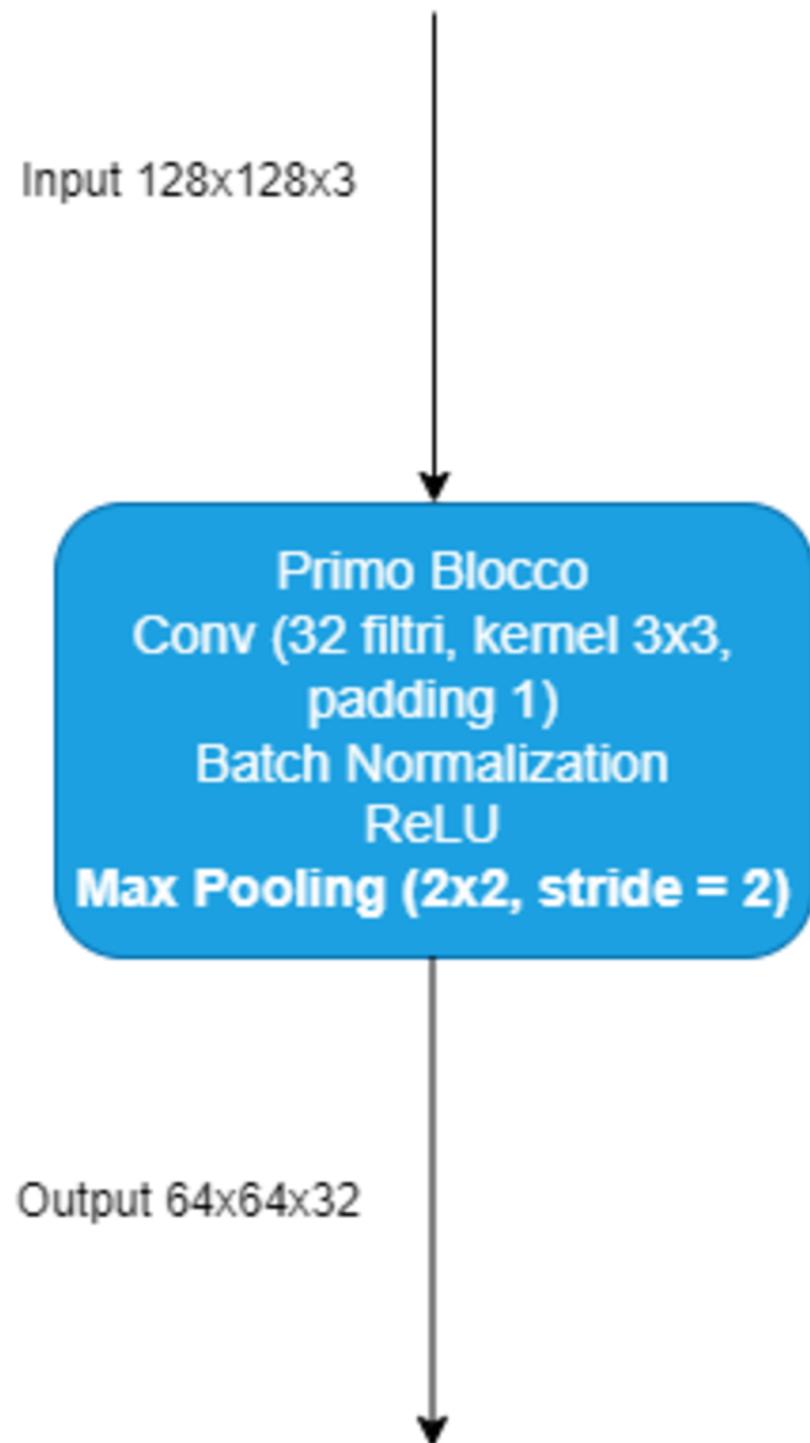
# ReLU

La rectified linear unit è una funzione di attivazione.

La ReLU opera su ogni input singolo, producendo zero per ogni valore negativo e lasciando invariato ogni valore non negativo.

In termini matematici, se l'input è maggiore o uguale a zero, l'output sarà lo stesso dell'input; se l'input è minore di zero, l'output sarà zero





# Max Pooling

Nel nostro caso usata per ridurre la dimensione spaziale delle mappe di caratteristiche, mantenendo le informazioni più “importanti”, a livello numerico, il numero più grande nella finestra.

Nella rete viene utilizzata con:

- Kernel 2x2 (stabilisce la dimensione di una finestra)
- Stride 2

Queste dimensioni permettono di applicare il pooling su finestre 2x2 e muovendosi con uno stride di 2 si evita la sovrapposizione delle finestre.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



6	8
14	16

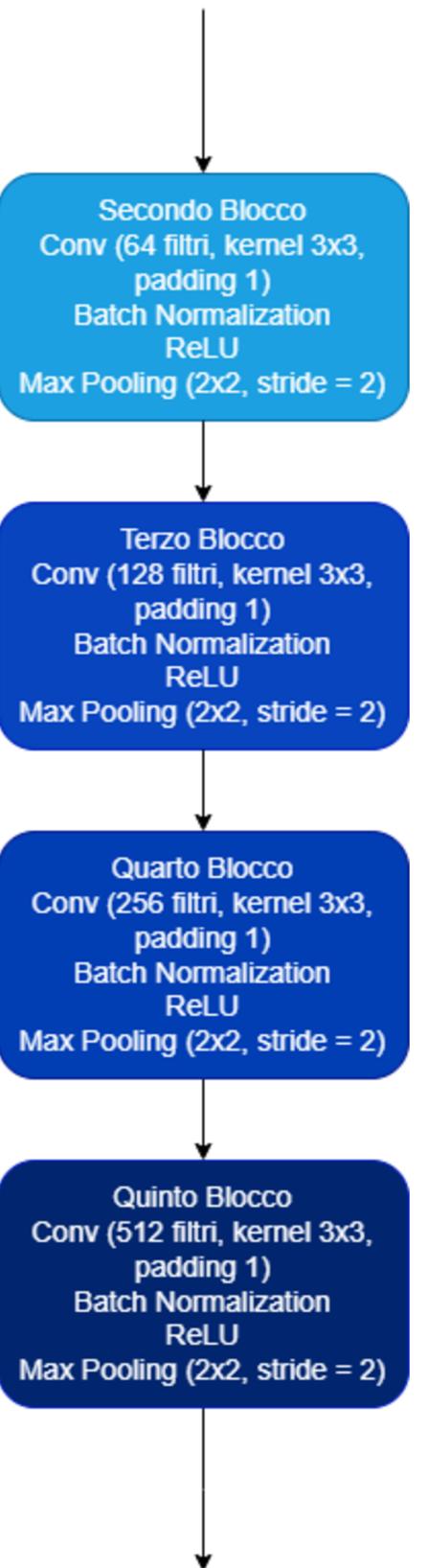
# Max Pooling

Nel nostro caso usata per ridurre la dimensione spaziale delle mappe di caratteristiche, mantenendo le informazioni più “importanti”, a livello numerico, il numero più grande nella finestra.

Nella rete viene utilizzata con:

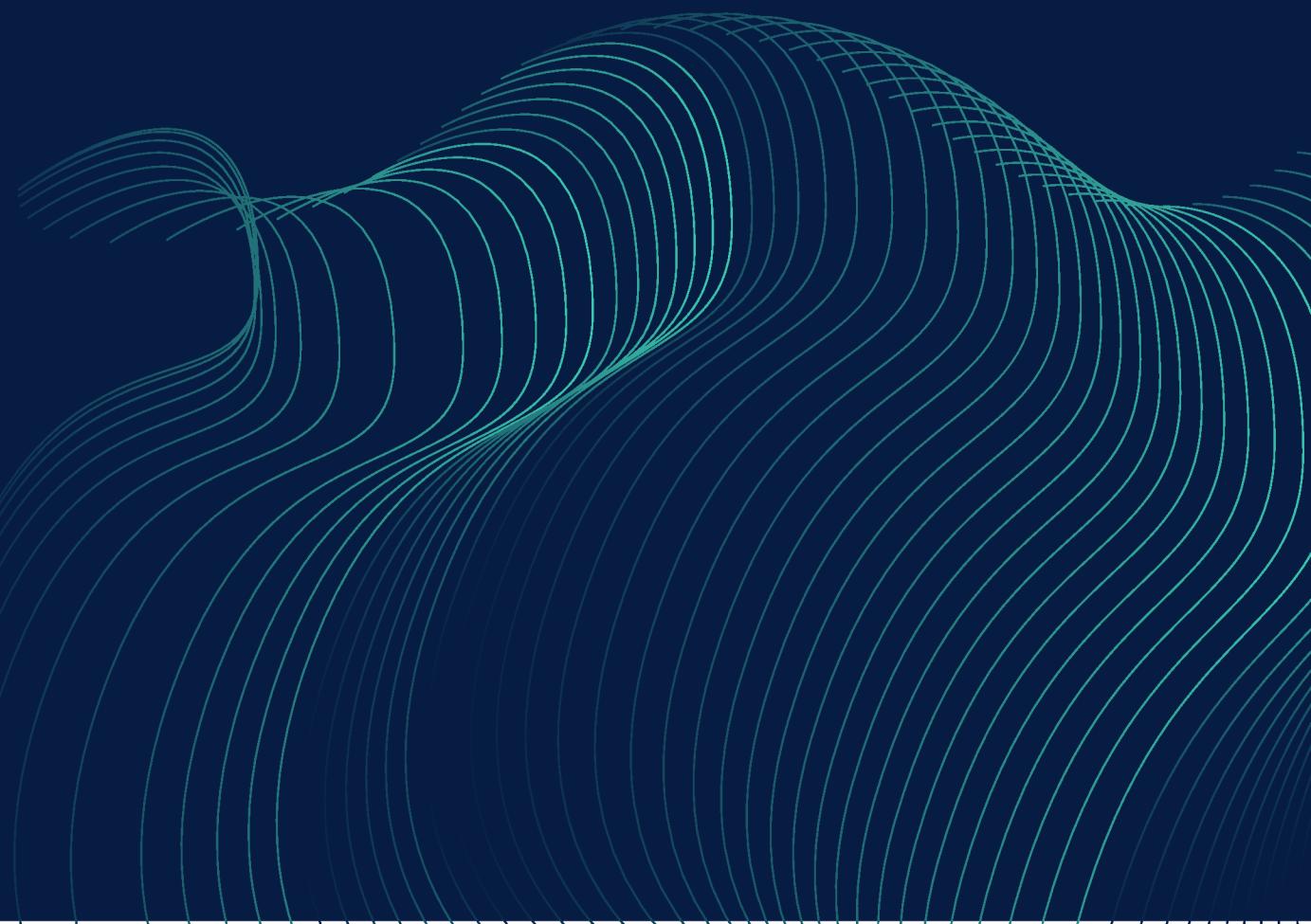
- Kernel 2x2 (stabilisce la dimensione di una finestra)
- Stride 2

Queste dimensioni permettono di applicare il pooling su finestre 2x2 e muovendosi con uno stride di 2 si evita la sovrapposizione delle finestre.



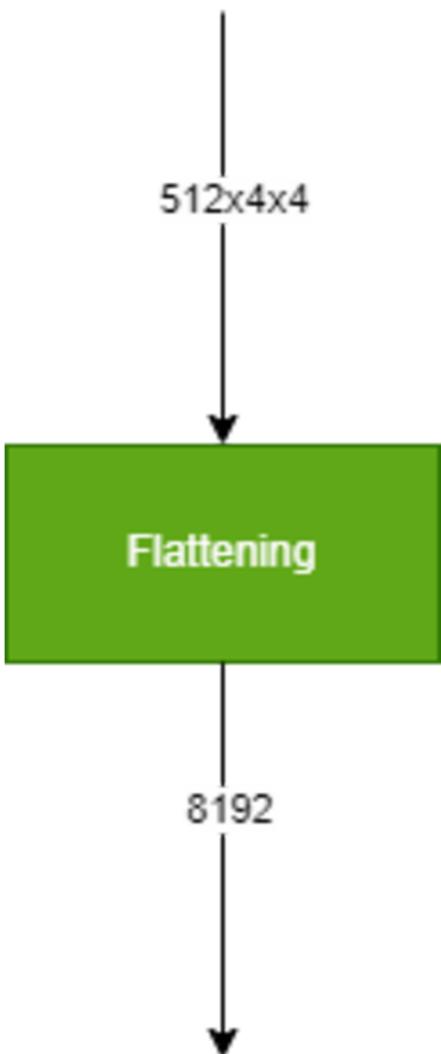
# Altri blocchi convoluzionali

Al primo blocco convoluzionale, seguono altri quattro blocchi identici, che differiscono solamente sui canali di output delle convoluzioni.



# Flattening

Viene utilizzata l'operazione di flattening per appiattire l'output dei layer convoluzionali. In sostanza scorre gli elementi di un tensore e li concatena in una lista. Questo è necessario per poter dare in pasto l'output delle convoluzioni ai layer fully connected dei passaggi successivi.



1	2
3	4

5	6
7	8

9	10
11	12



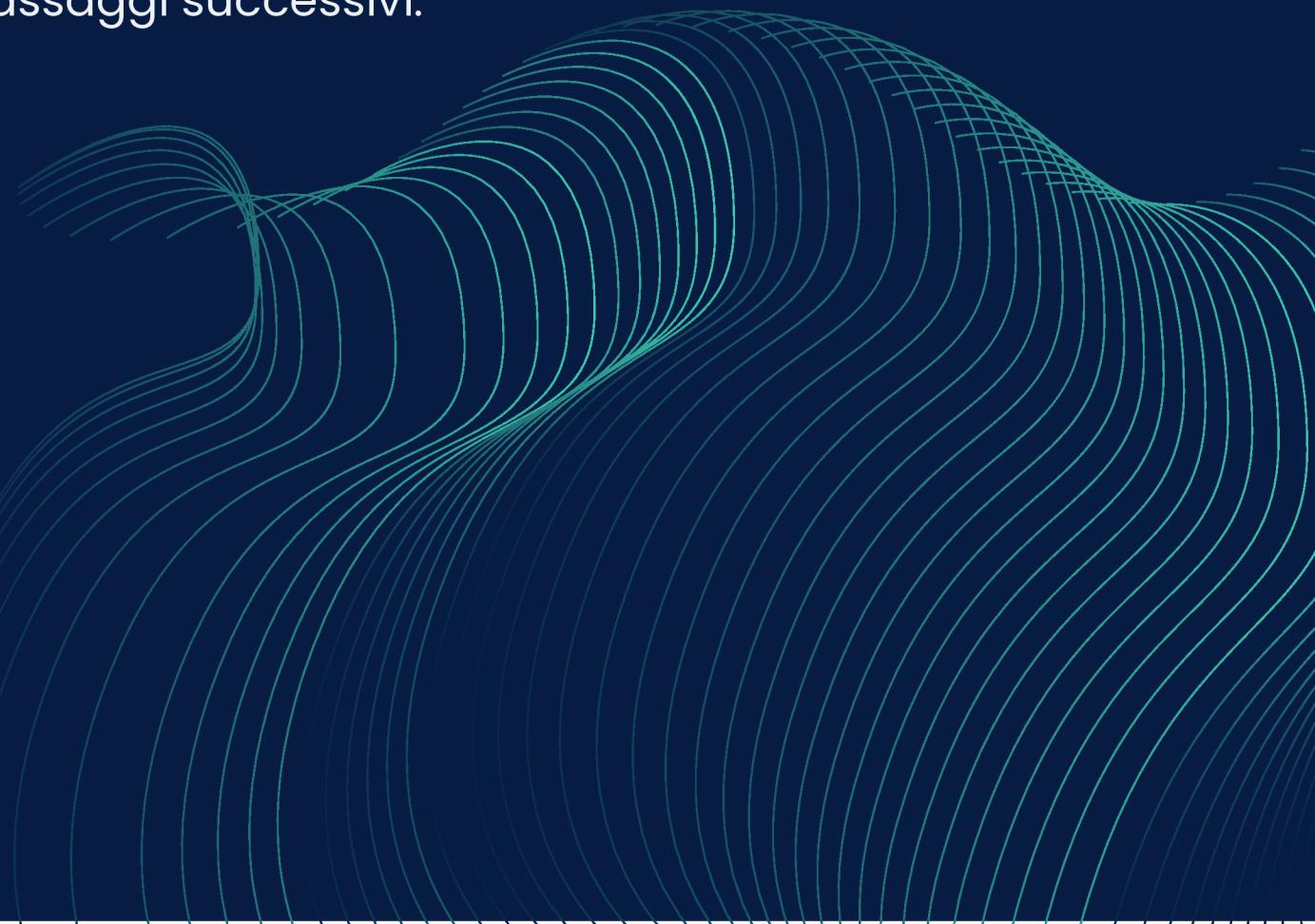
1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

# Flattening

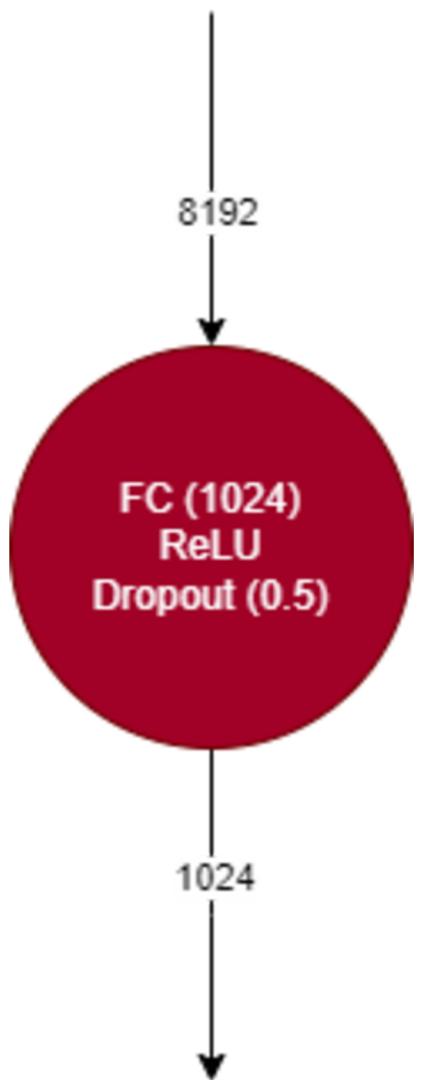
Viene utilizzata l'operazione di flattening per appiattire l'output dei layer convoluzionali.

In sostanza scorre gli elementi di un tensore e li concatena in una lista.

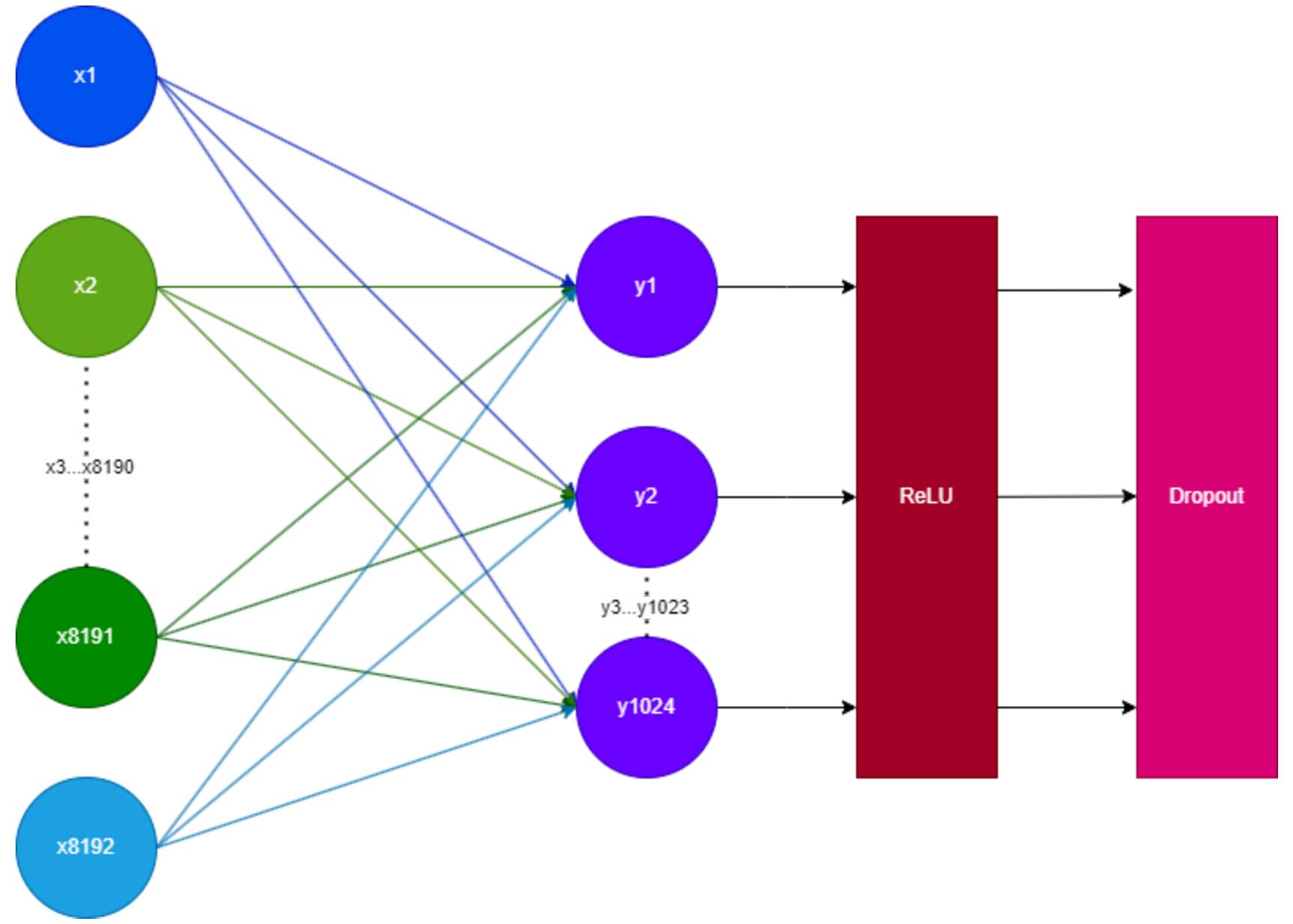
Questo è necessario per poter dare in pasto l'output delle convoluzioni ai layer fully connected dei passaggi successivi.



# Primo strato FC



Attraverso un primo strato fully connected si passa da un input di 8192 neuroni a 1024, dopodichè viene applicata una ReLU e un Dropout di 0.5.



$$y = xA^T + b$$

Dove:

- x: vettore di 8192 elementi
- A: matrice di pesi associati alle connessioni, dimensioni (1024,8192)
- b: vettore di bias associato a y, di 1024 elementi

## Primo strato FC

Attraverso un primo strato fully connected si passa da un input di 8192 neuroni a 1024, dopodichè viene applicata una ReLU e un Dropout di 0.5.

$$y = xA^T + b$$

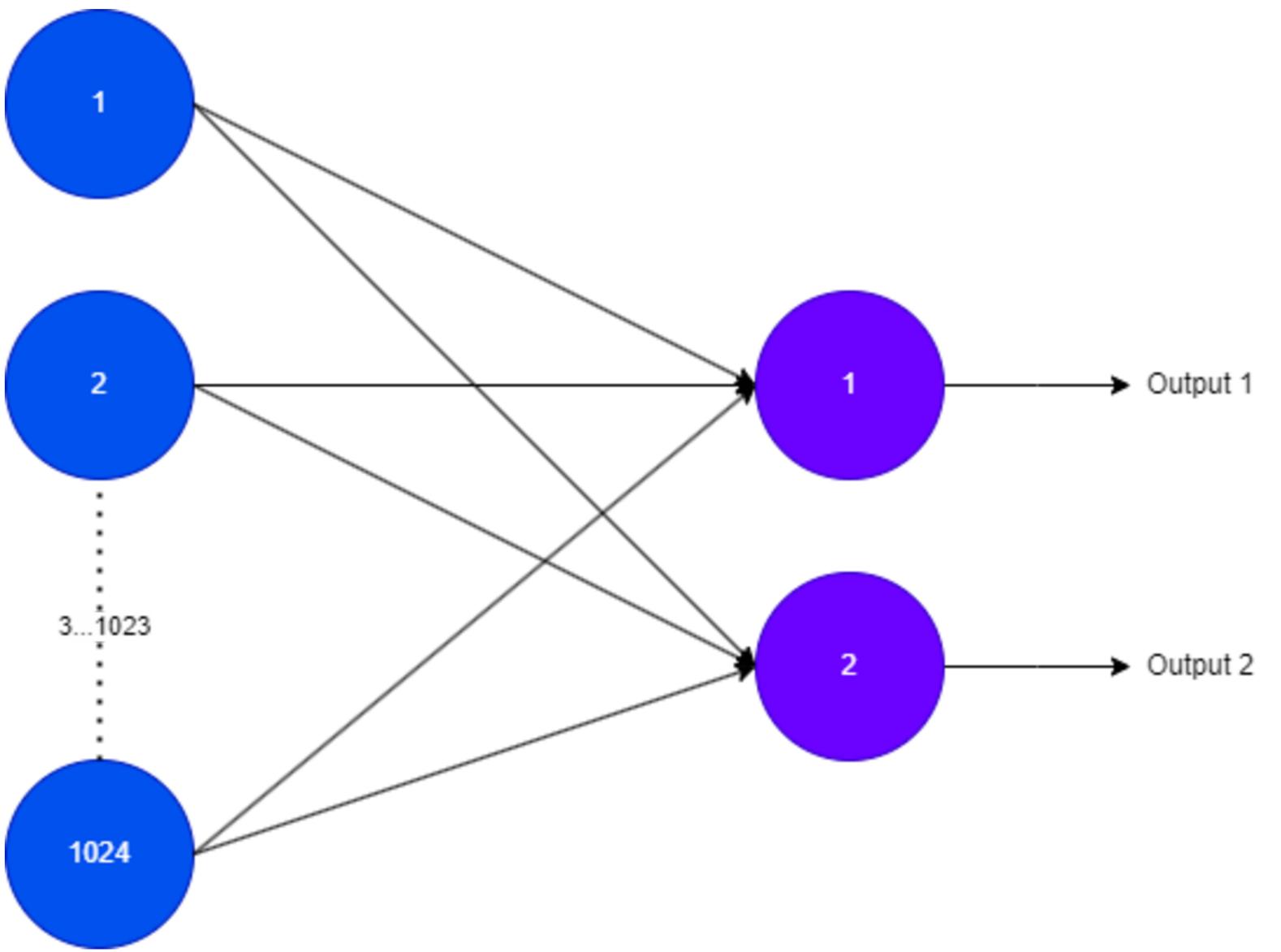
Dove:

- x: vettore di 8192 elementi
- A: matrice di pesi associati alle connessioni, dimensioni (1024,8192)
- b: vettore di bias associato a y, di 1024 elementi



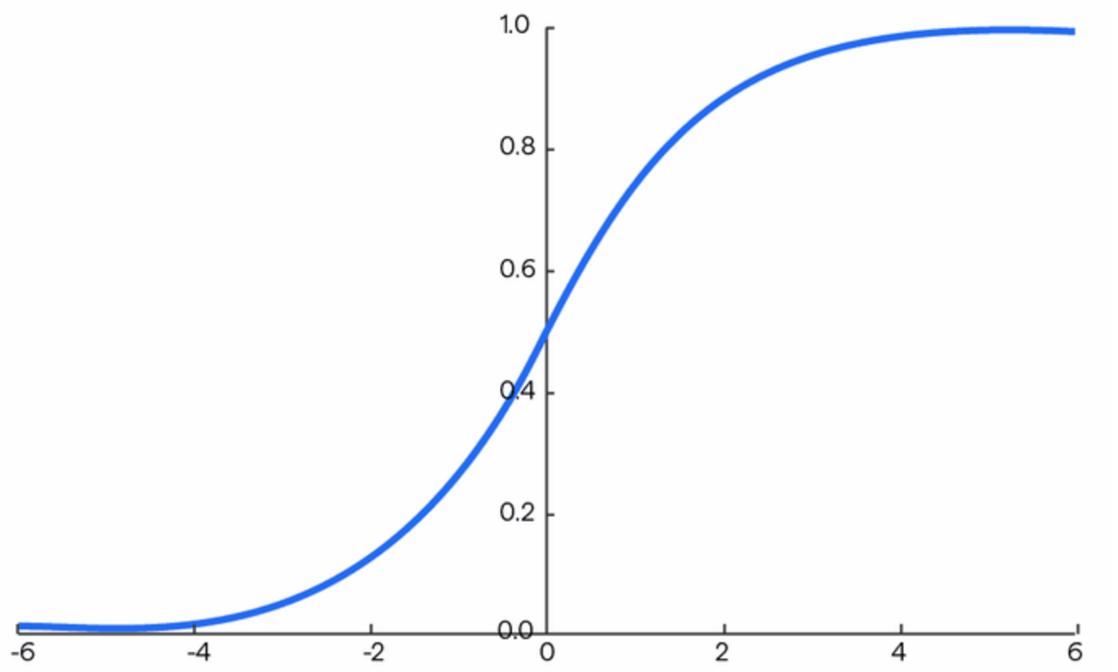
## Primo strato FC

Attraverso un primo strato fully connected si passa da un input di 8192 neuroni a 1024, dopodichè viene applicata una ReLU e un Dropout di 0.5.



## Secondo strato FC

Attraverso un secondo strato fully connected si passa da un input di 1024 neuroni a 2. Questi saranno gli output finali.



$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

$z_i$ :  $i$ -esimo elemento del vettore di input

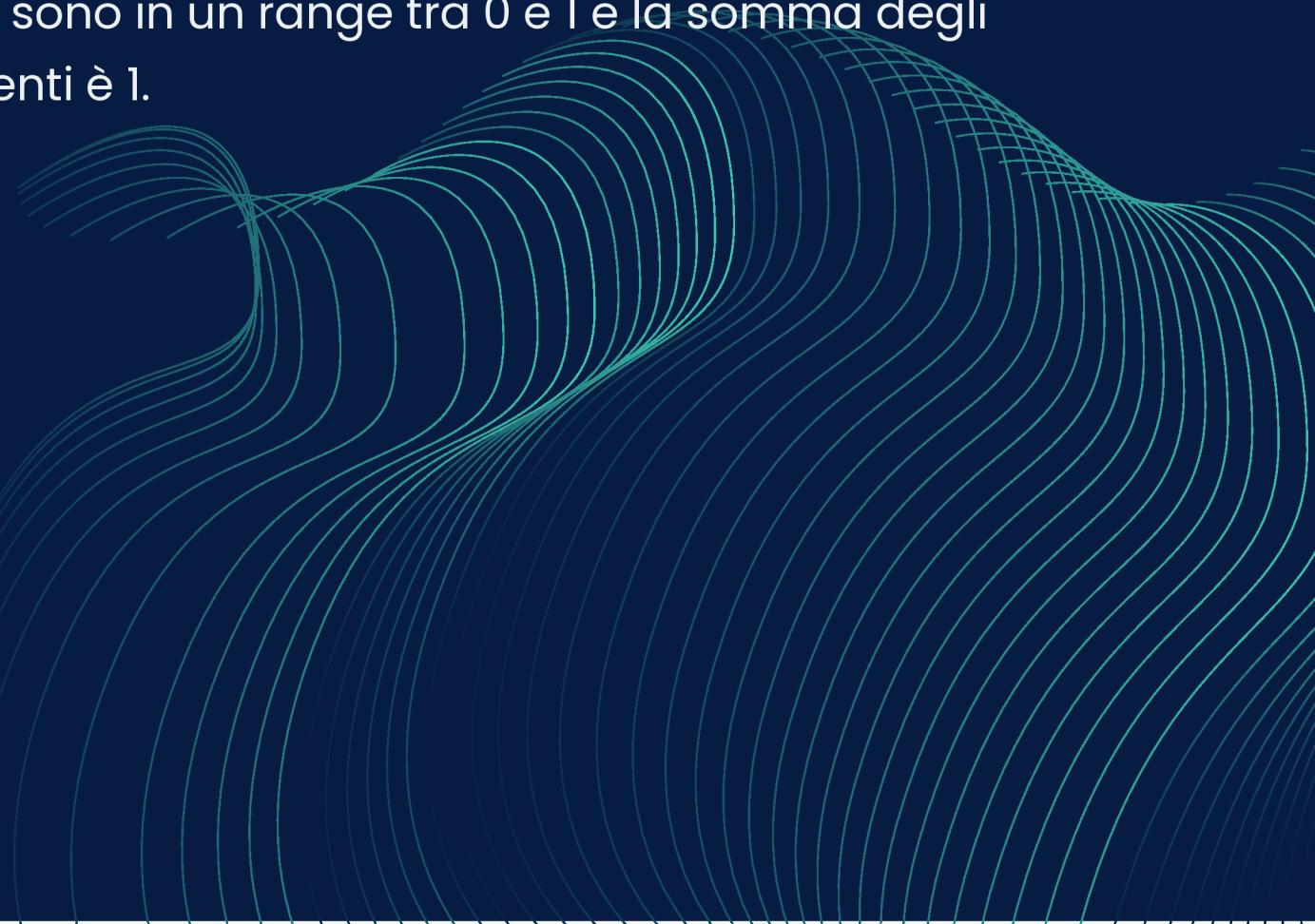
$K$ : il numero totale di elementi nel vettore di input

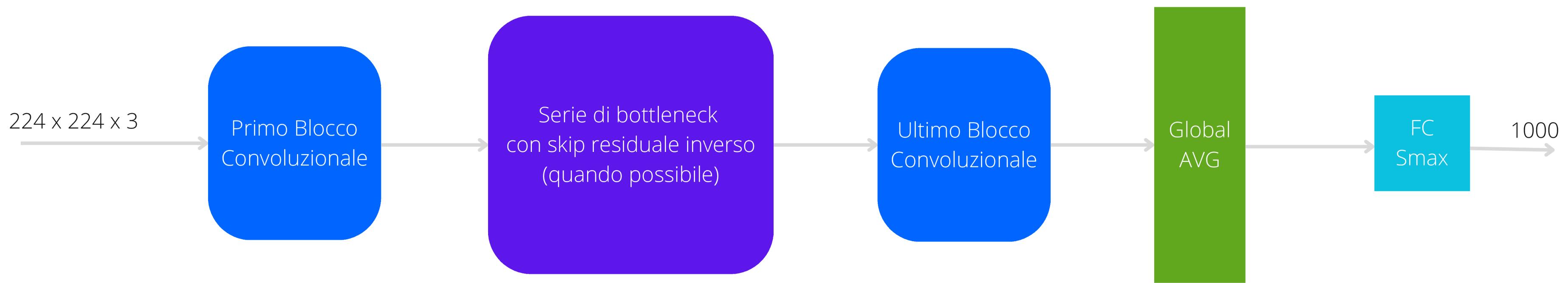
$e$ : base del logaritmo naturale

# Softmax

L'ultima operazione da fare sarebbe una softmax, tuttavia questa non è stata esplicitata in questa fase perchè per il calcolo della perdita viene utilizzata una CrossEntropyLoss che contiene all'interno una softmax.

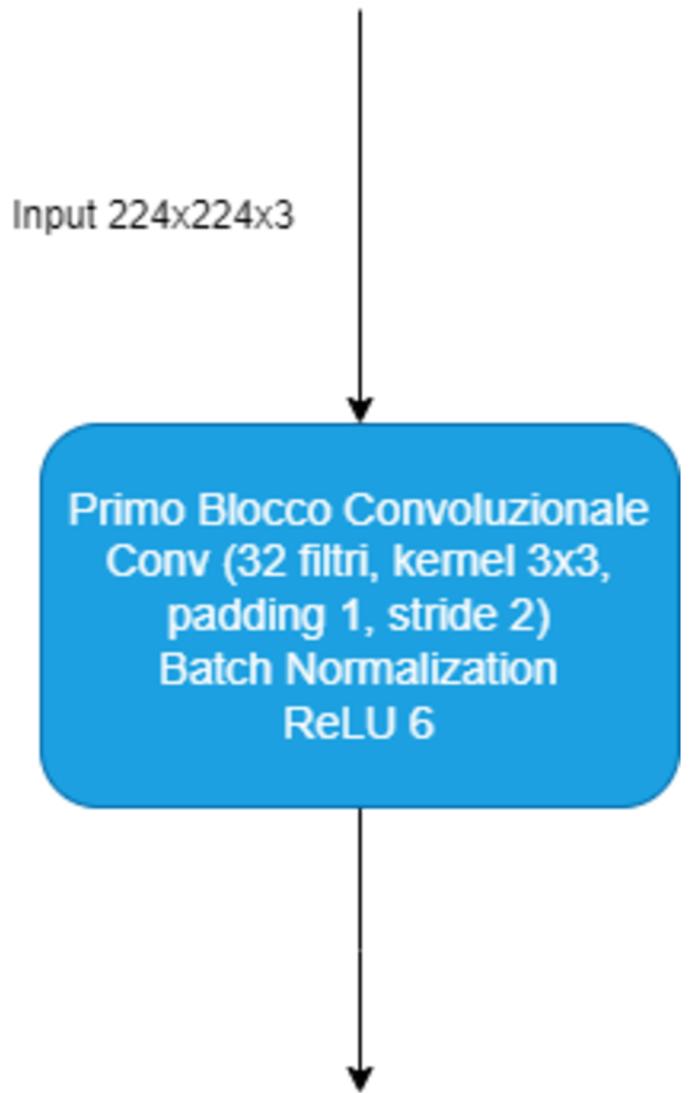
In sostanza la softmax trasforma un vettore di numeri reali in un vettore di probabilità, in cui i valori sono in un range tra 0 e 1 e la somma degli elementi è 1.





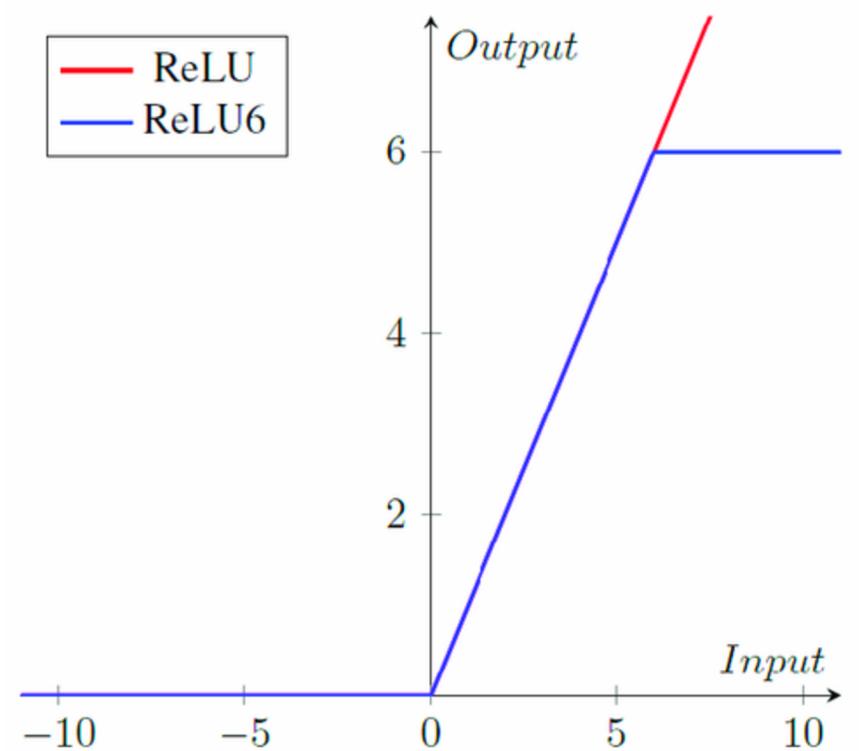
# MobileNet V2

# Primo blocco convoluzionale



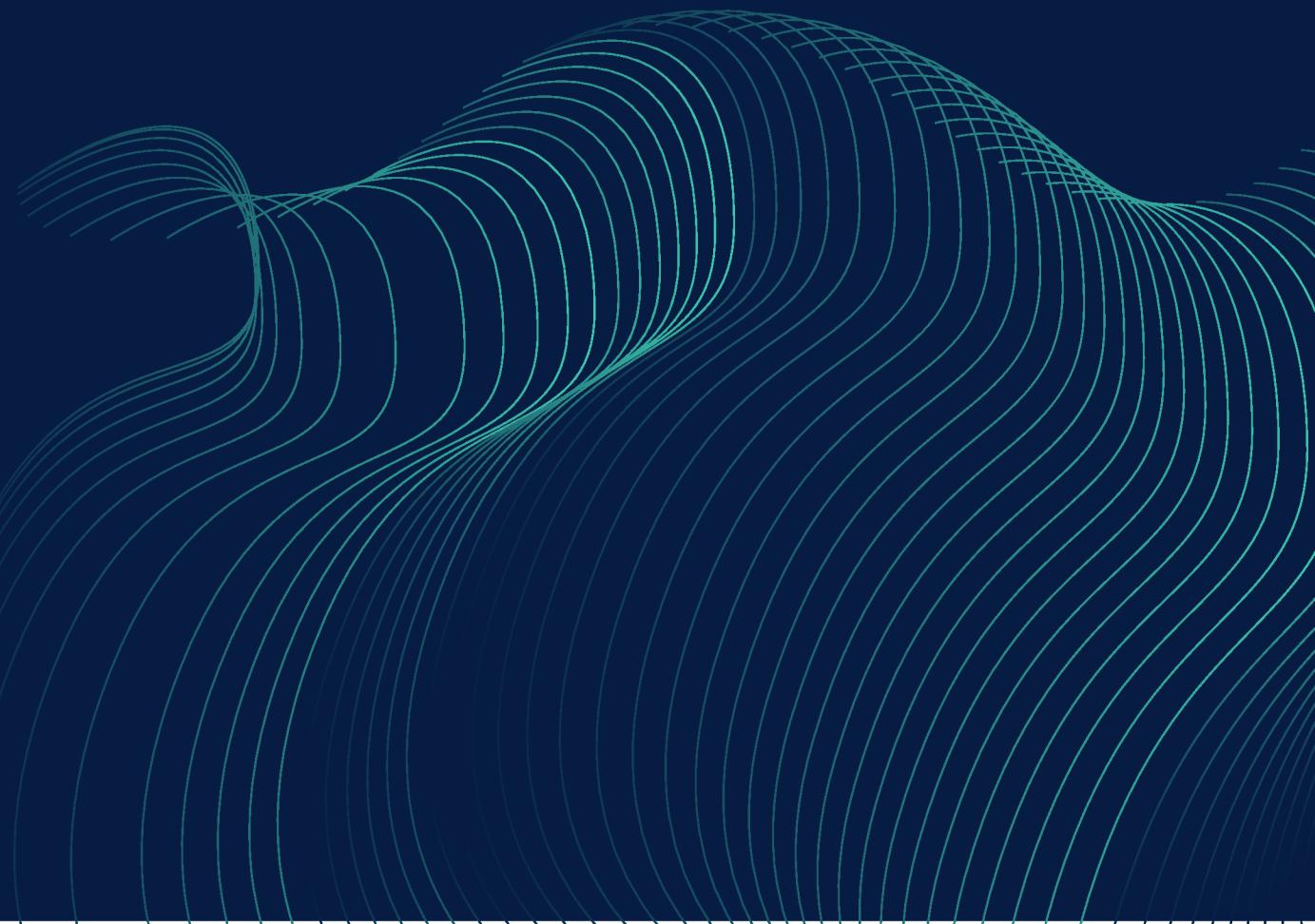
- Convoluzione 3x3 con 32 filtri, stride 2, padding 1
- Batch Normalization
- ReLU6
- Output: 32 canali, dimensioni spaziali 112x112

$$\text{ReLU6}(x) = \min(\max(0, x), 6)$$



# ReLU6

In sostanza una ReLU6 è come una ReLU classica, solo che pone un limite superiore a 6 per l'output.



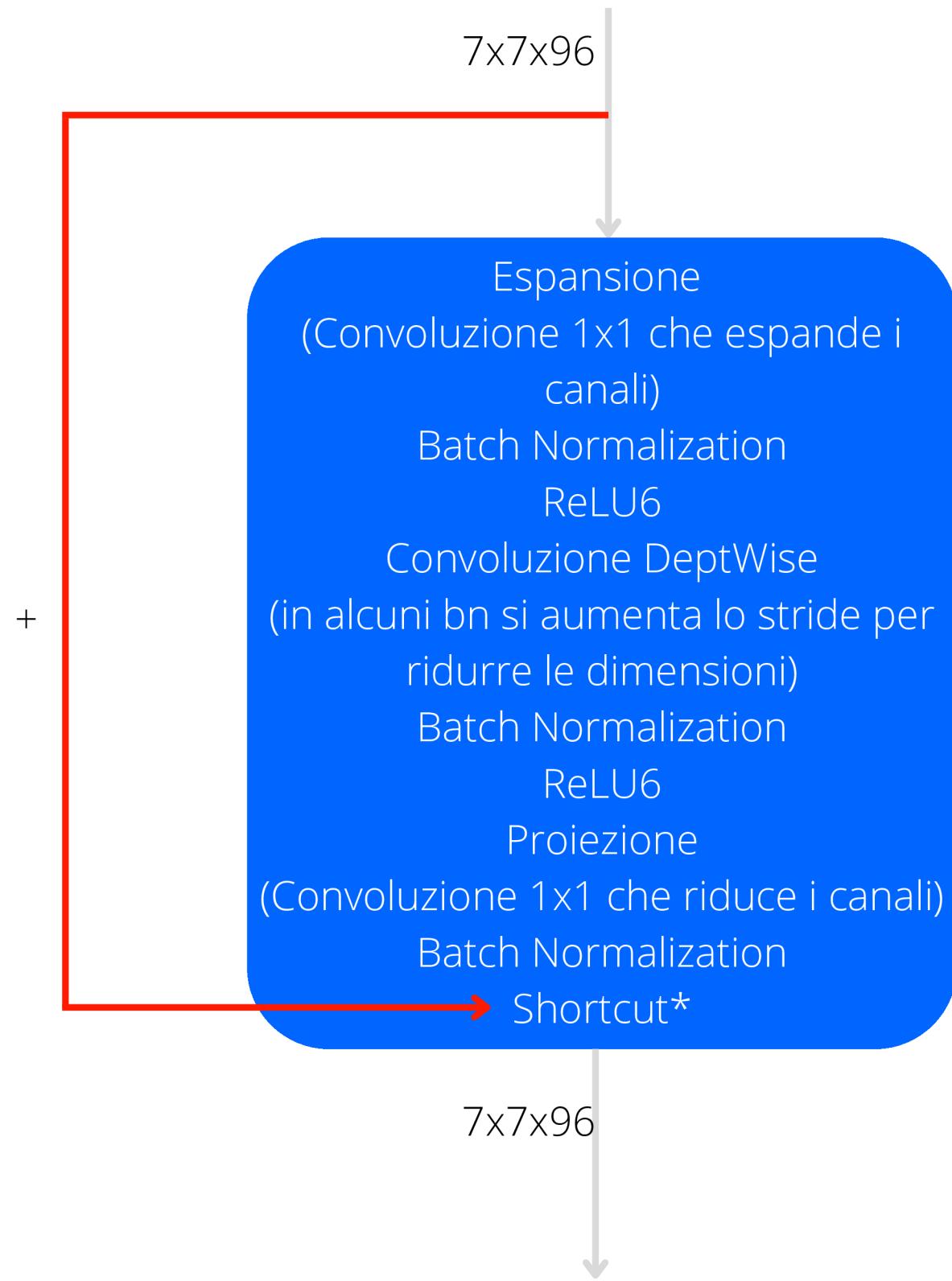
# Blocco di BottleNeck

Il blocco di bottleneck è uno stratagemma per migliorare l'efficienza di una rete, di solito consiste in 3 fasi principali:

- Aumento dei canali con una convoluzione 1x1 e stride 1
- Convoluzione deptwise (di solito 3x3 con stride variabile)
- Riduzione dei canali con una convoluzione 1x1 e stride 1

↓  
Espansione  
(Convoluzione 1x1 che espande i canali)  
Batch Normalization  
ReLU6  
Convoluzione DeptWise  
(in alcuni bn si aumenta lo stride per ridurre le dimensioni)  
Batch Normalization  
ReLU6  
Proiezione  
(Convoluzione 1x1 che riduce i canali)  
Batch Normalization

↓

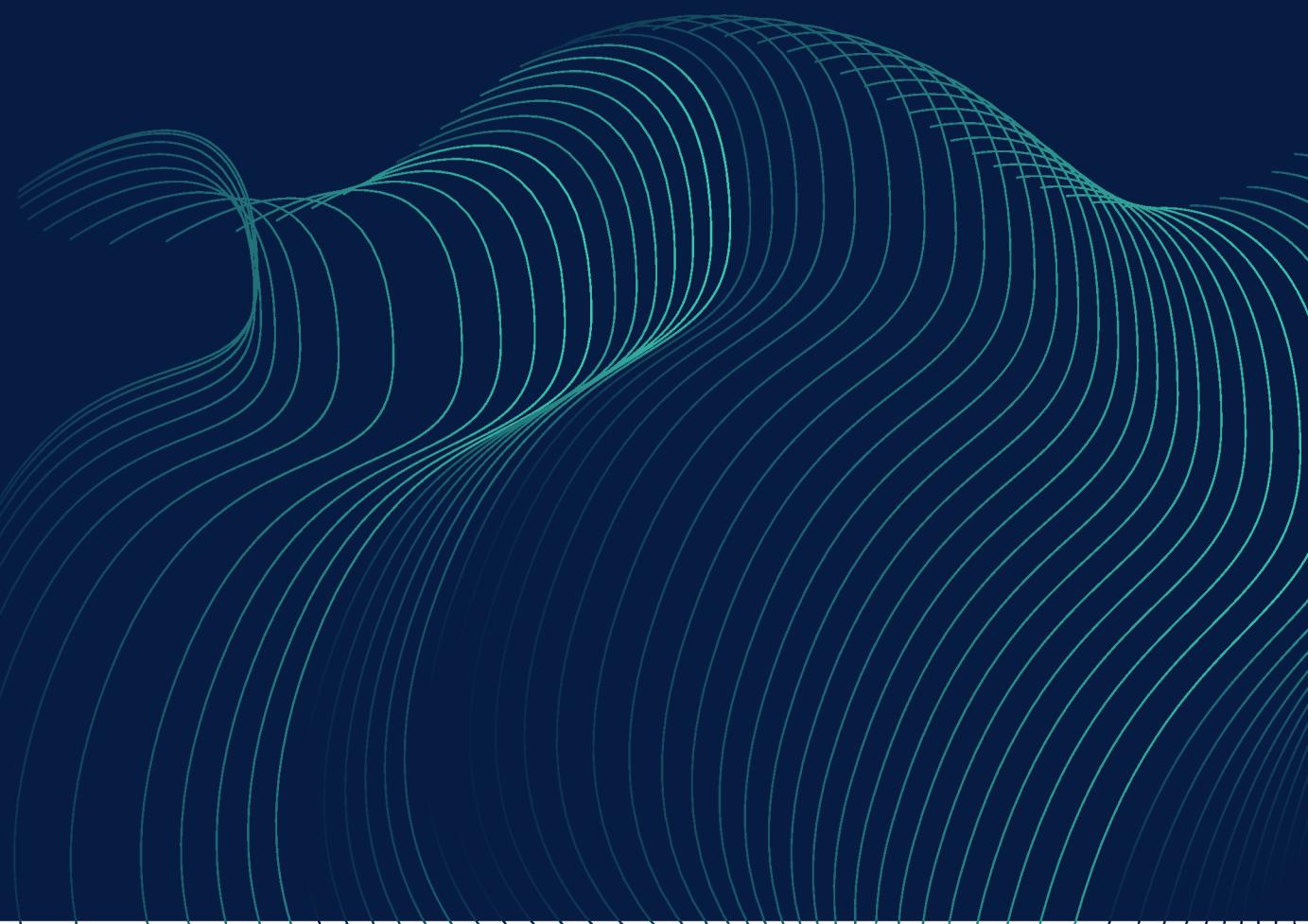


# Blocco di BottleNeck Residuale

Il blocco di bottleneck è uno stratagemma per migliorare l'efficienza di una rete, di solito consiste in 3 fasi principali:

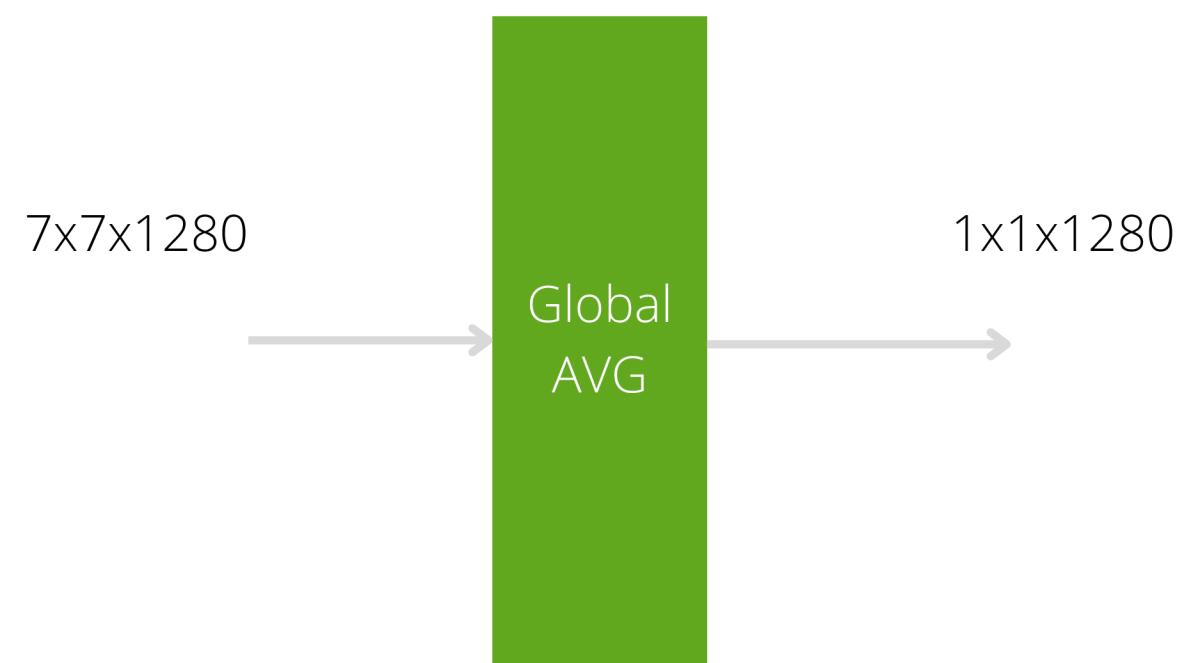
- Aumento dei canali con una convoluzione 1x1 e stride 1
- Convoluzione deptwise (di solito 3x3 con stride variabile)
- Riduzione dei canali con una convoluzione 1x1 e stride 1

# Ultimo blocco convoluzionale



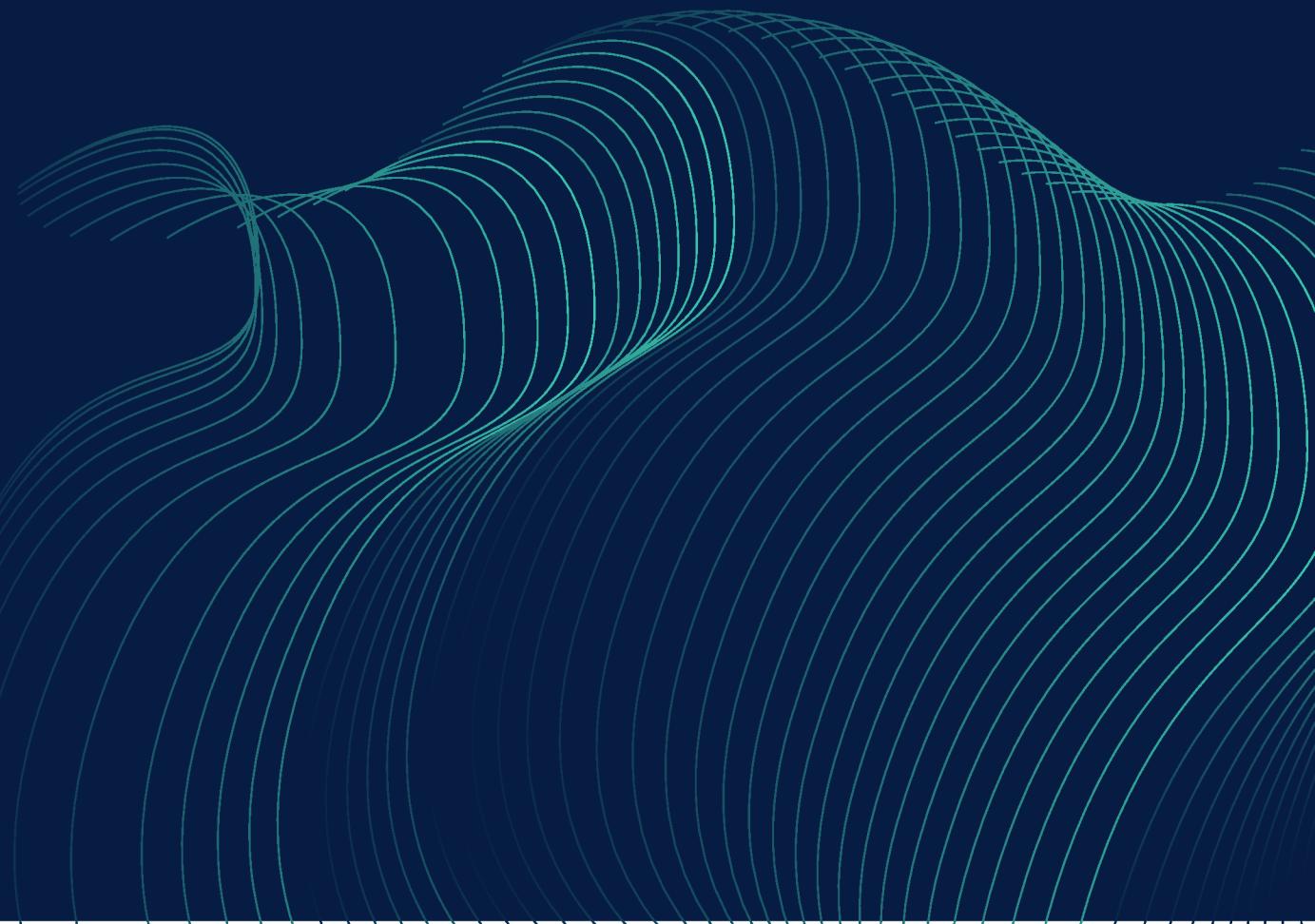
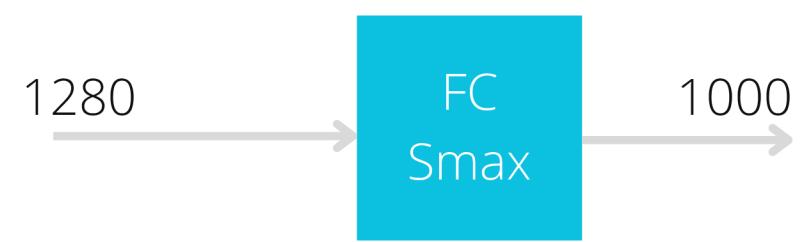
# Global AVG Pooling

Calcola la media di ogni mappa  $7 \times 7$  e la inserisce in un vettore di 1280 elementi (numero canali)



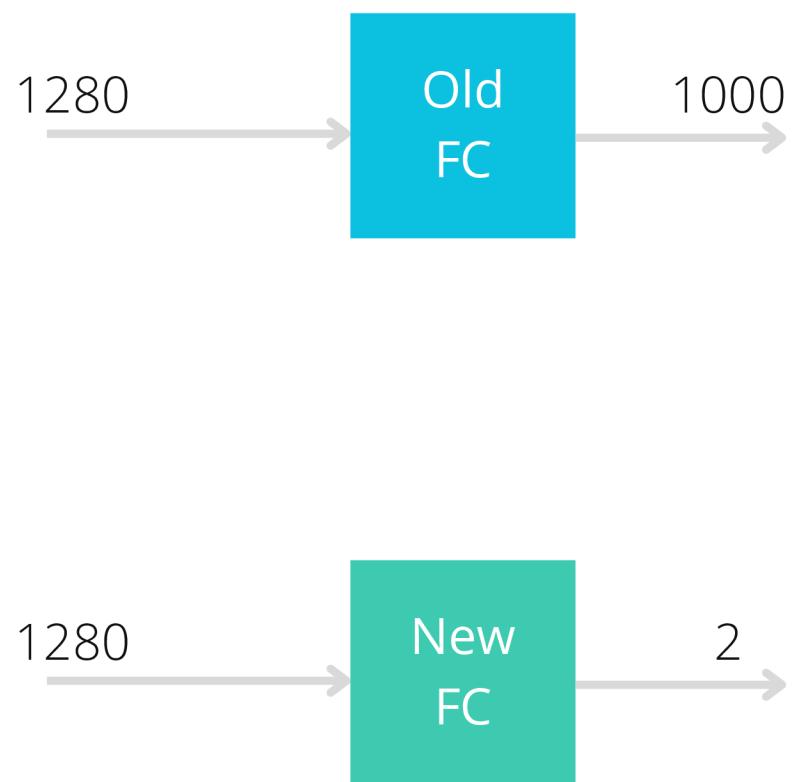
# Fully Connected

Attraverso un secondo strato fully connected si passa da un input di 1024 neuroni a 1000. Questi saranno gli output finali.



# Transfer Learning

Per applicare il metodo del transfer learning è stato sostituito lo strato FC, sostituendo con un nuovo strato FC con dimensioni di output a 2.



---

Rete Custom

MobileNet V2  
**con** congelamento pesi

MobileNet V2  
**senza** congelamento pesi

Train: 2 modelli



# Inizializzazione Pesi

## Layer di Convoluzione:

- Pesi inizializzati con la distribuzione di Kaiming (He)
- Bias inizializzati a zero

## Layer di Batch Normalization:

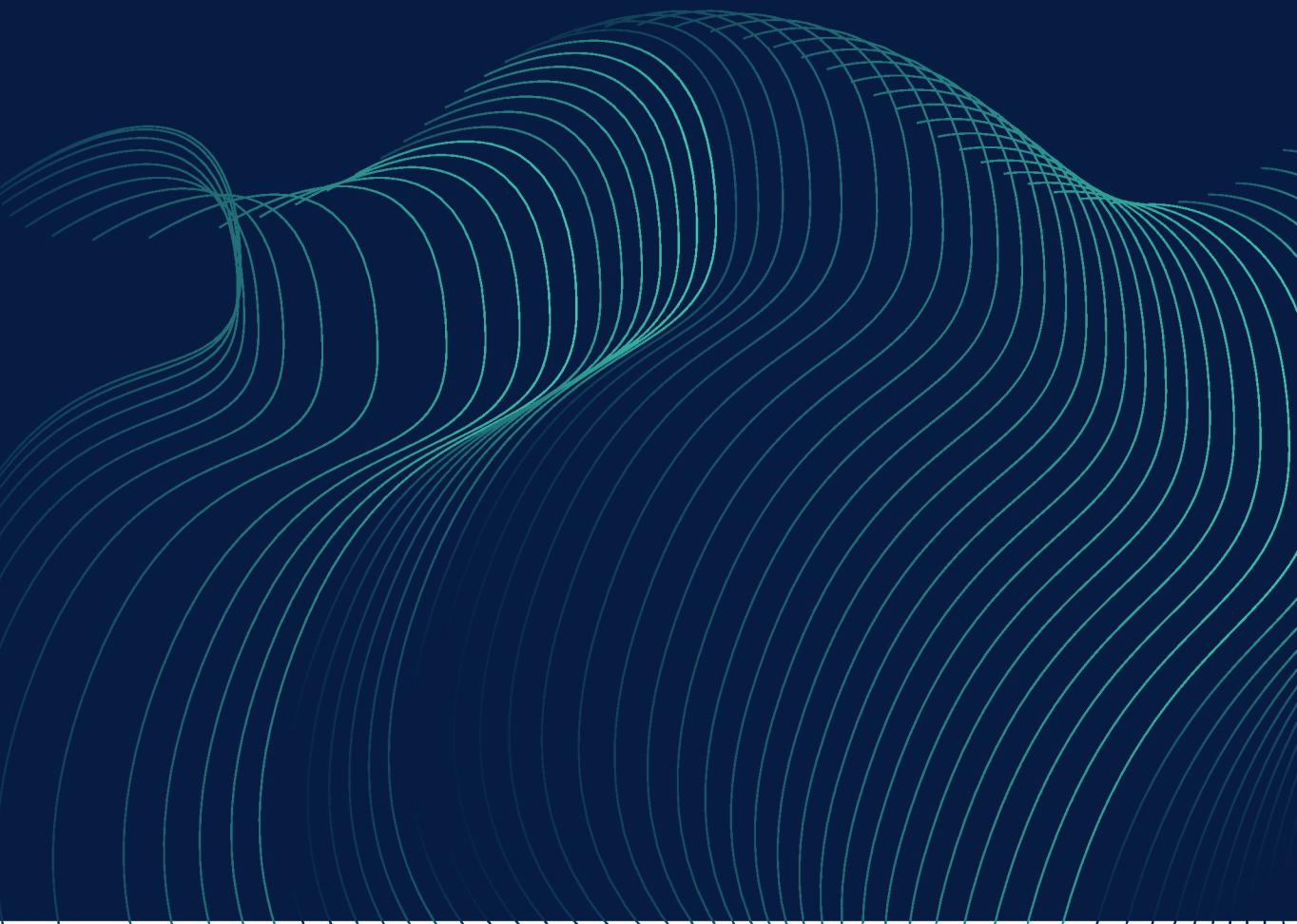
- La scala è inizializzata a 1 (parametro gamma)
- Il bias è inizializzato a 0 (parametro beta)

## Layer FC:

- Pesi inizializzati con la distribuzione di Kaiming (He)
- Bias inizializzati a zero

Pytorch di default inizializza i pesi e i bias nelle reti che vengono costruite, in base a "cosa conviene nel caso specifico", anche se offre la possibilità di definire inizializzazioni custom.

Viene riportata una lista di come vengono inizializzati di default.



# Inizializzazione Pesi He

In sostanza, si calcola la varianza come:

$$Var(w) = \frac{2}{n_{input}}$$

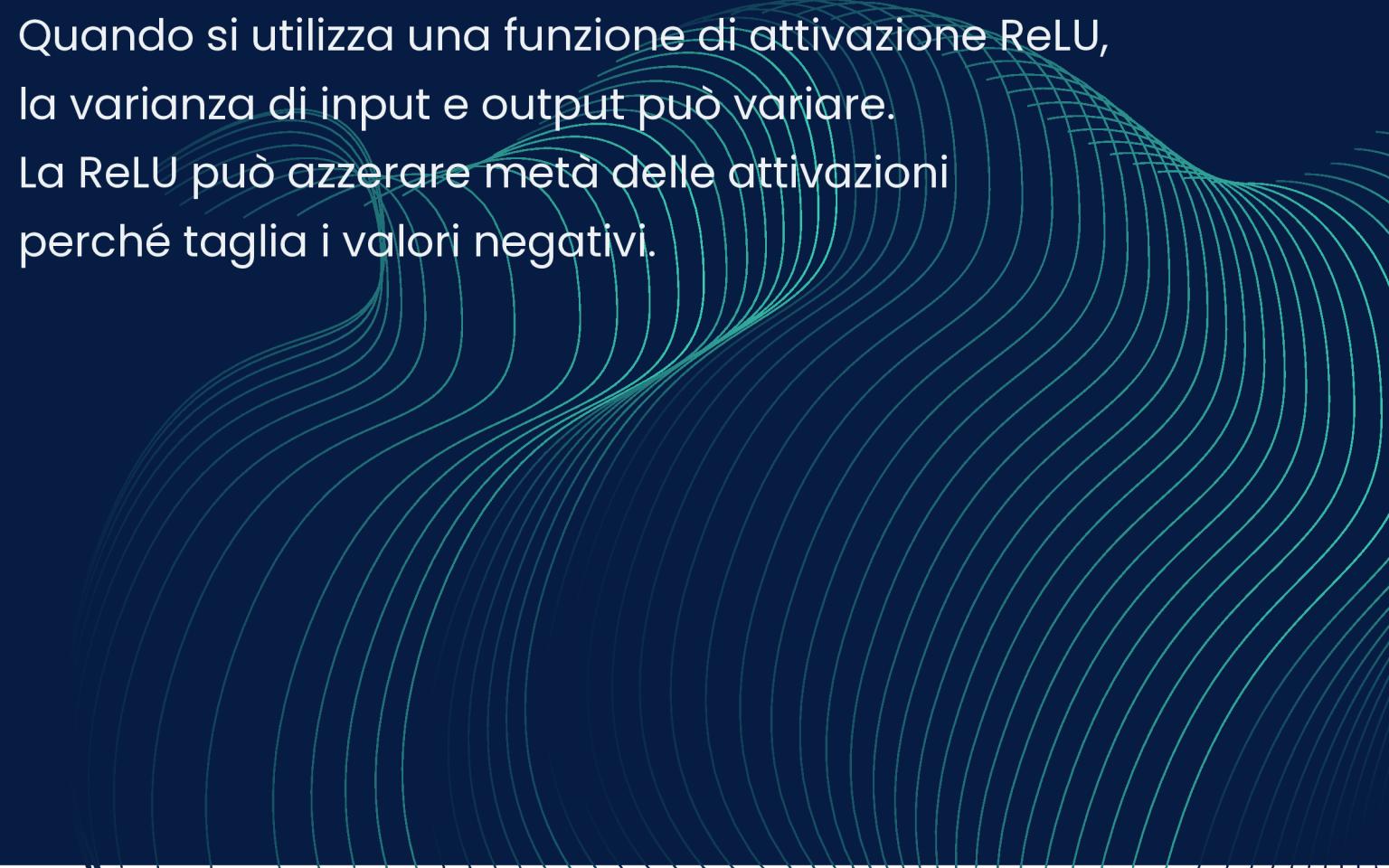
Dove  $n_{input}$  sta per il numero di neuroni nel layer di input

Una volta calcolata la varianza si prelevano i valori (pesi) secondo una distribuzione normale:

$$w \sim N(0, \frac{2}{n_{input}})$$

L'inizializzazione di Kaiming è progettata per mantenere la varianza dei dati attraverso gli strati della rete, specialmente quando si utilizzano le funzioni di attivazione ReLU. Per fare ciò, i pesi vengono inizializzati seguendo una distribuzione normale con media zero e varianza calcolata in base all'input.

Quando si utilizza una funzione di attivazione ReLU, la varianza di input e output può variare. La ReLU può azzerare metà delle attivazioni perché taglia i valori negativi.



# Inizializzazione Pesi He

Es: 100 neuroni in input

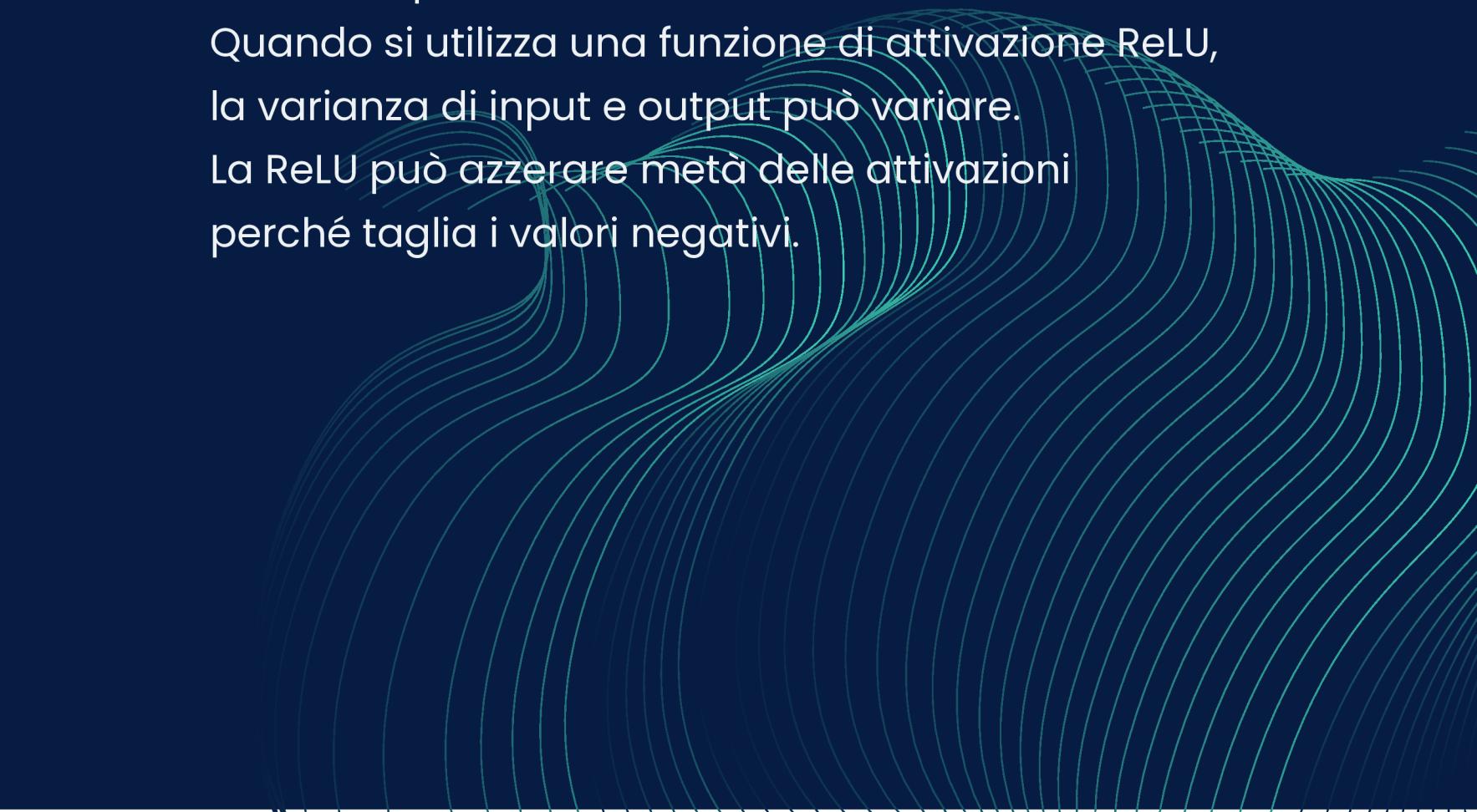
$$Var(w) = \frac{2}{100}$$

Una volta calcolata la varianza si prelevano i valori (pesi) secondo una distribuzione normale:

$$w \sim N(0, 0.02)$$

L'inizializzazione di Kaiming è progettata per mantenere la varianza dei dati attraverso gli strati della rete, specialmente quando si utilizzano le funzioni di attivazione ReLU. Per fare ciò, i pesi vengono inizializzati seguendo una distribuzione normale con media zero e varianza calcolata in base all'input.

Quando si utilizza una funzione di attivazione ReLU, la varianza di input e output può variare. La ReLU può azzerare metà delle attivazioni perché taglia i valori negativi.



# Loss Function

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

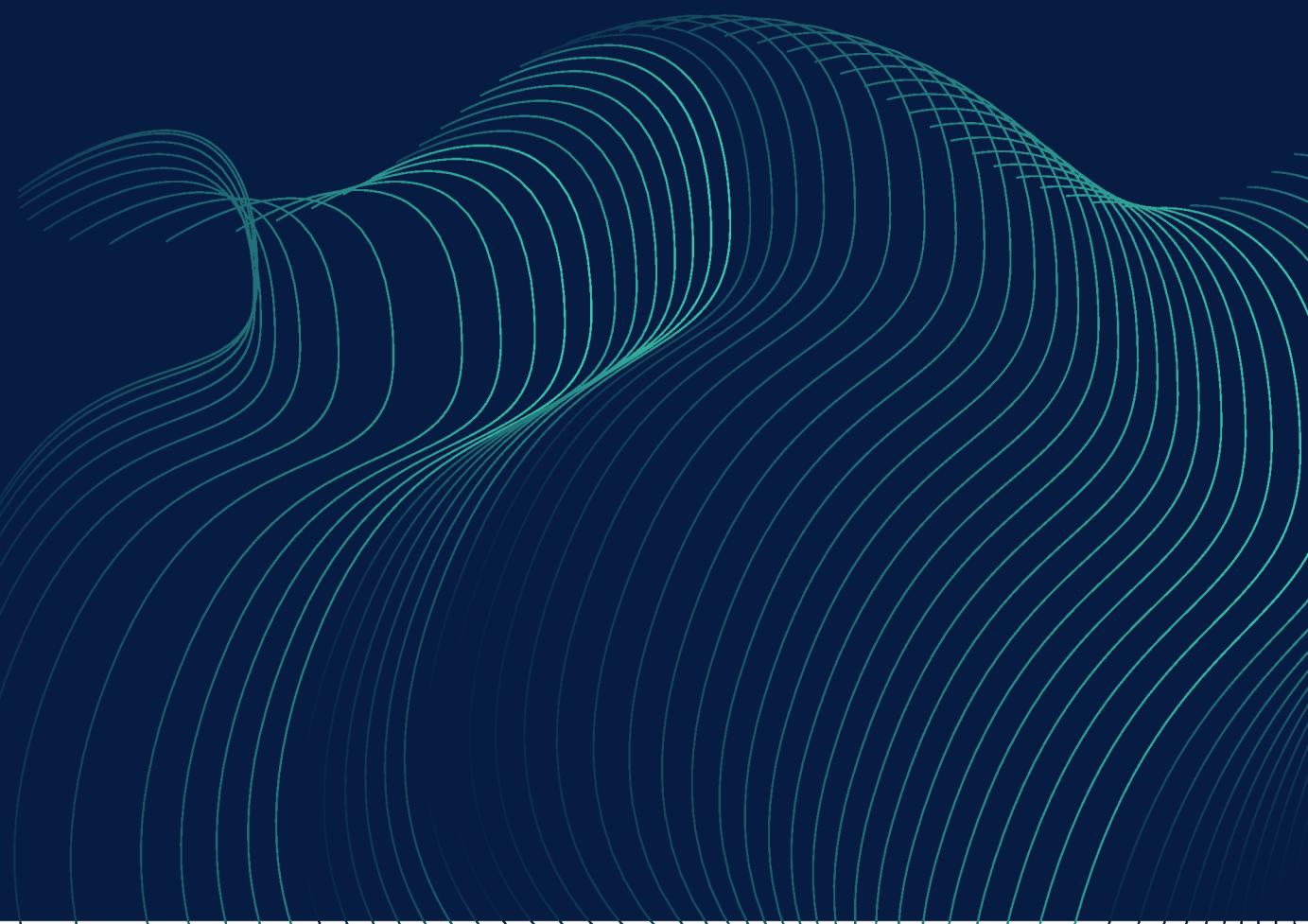
Simile alla SoftMax con l'aggiunta del -log

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(w)$$

N: numero classi

$\lambda$ : fattore di regolarizzazione

Per quanto riguarda il calcolo della loss function è stata utilizzata la Cross Entropy Loss, con un fattore di regolarizzazione L2.



# Regularization

$$L2 : R(w) = \sum_{i=1}^n w_i^2$$

$$L = \hat{L} + \lambda R(w)$$

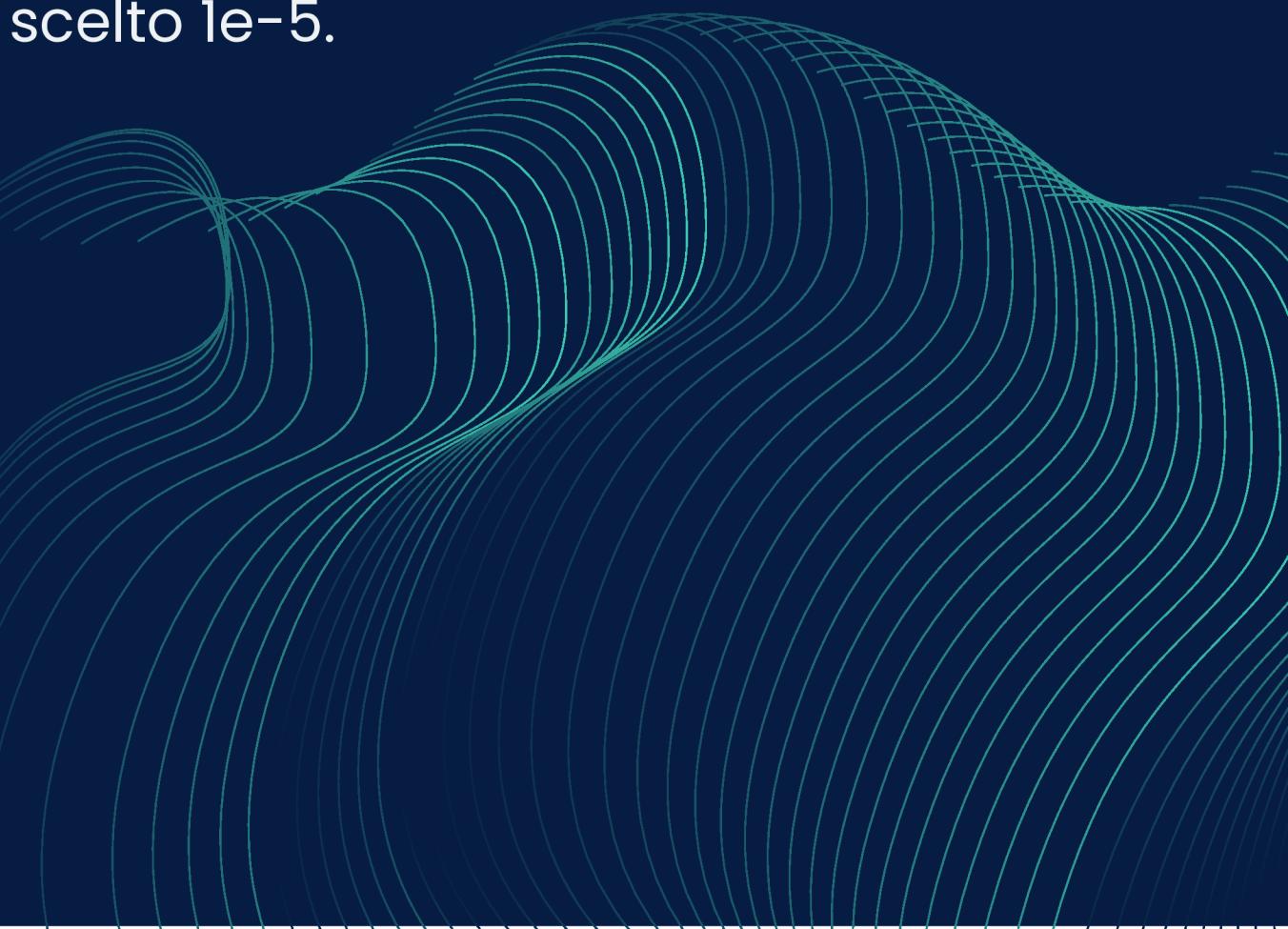
Dove:

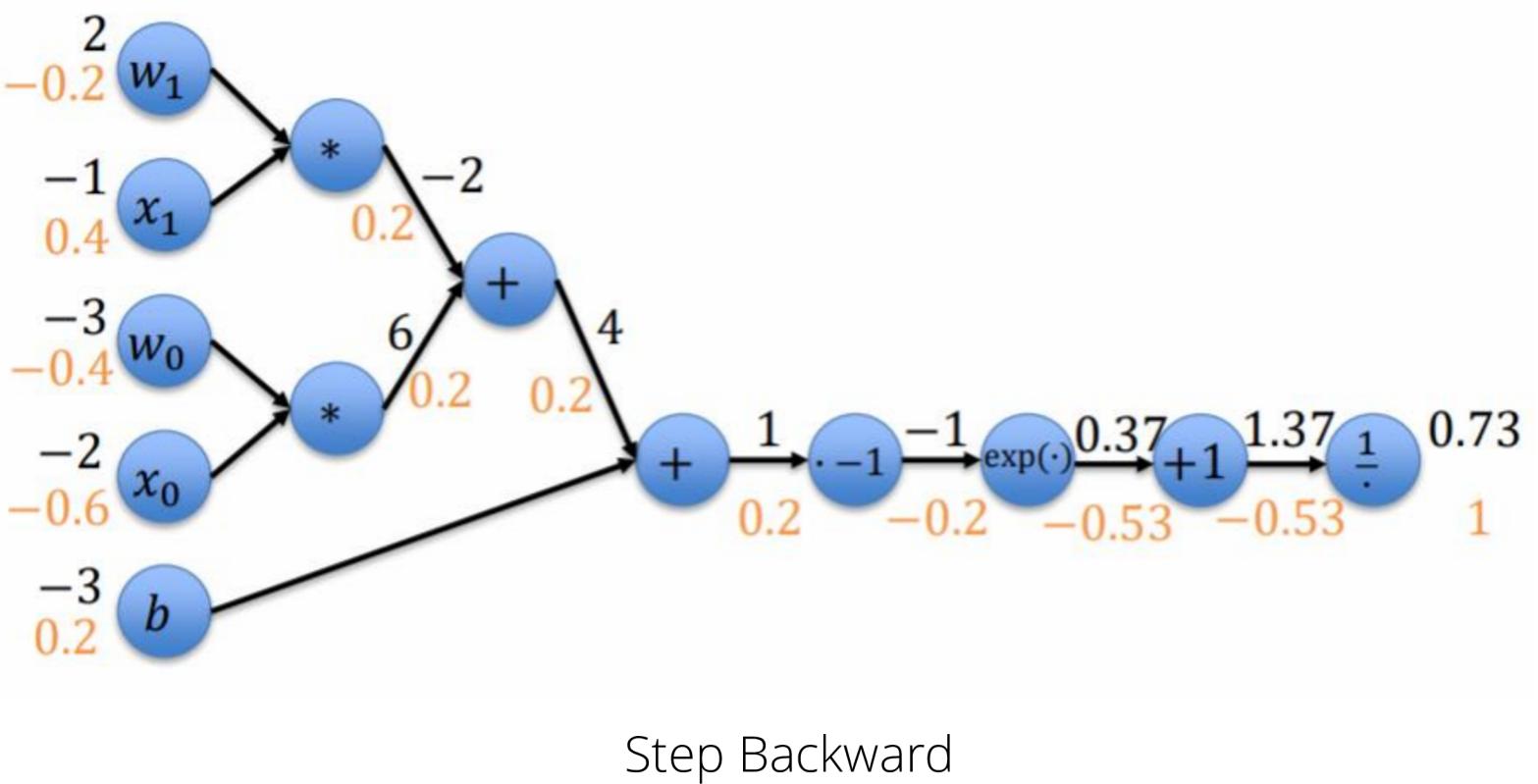
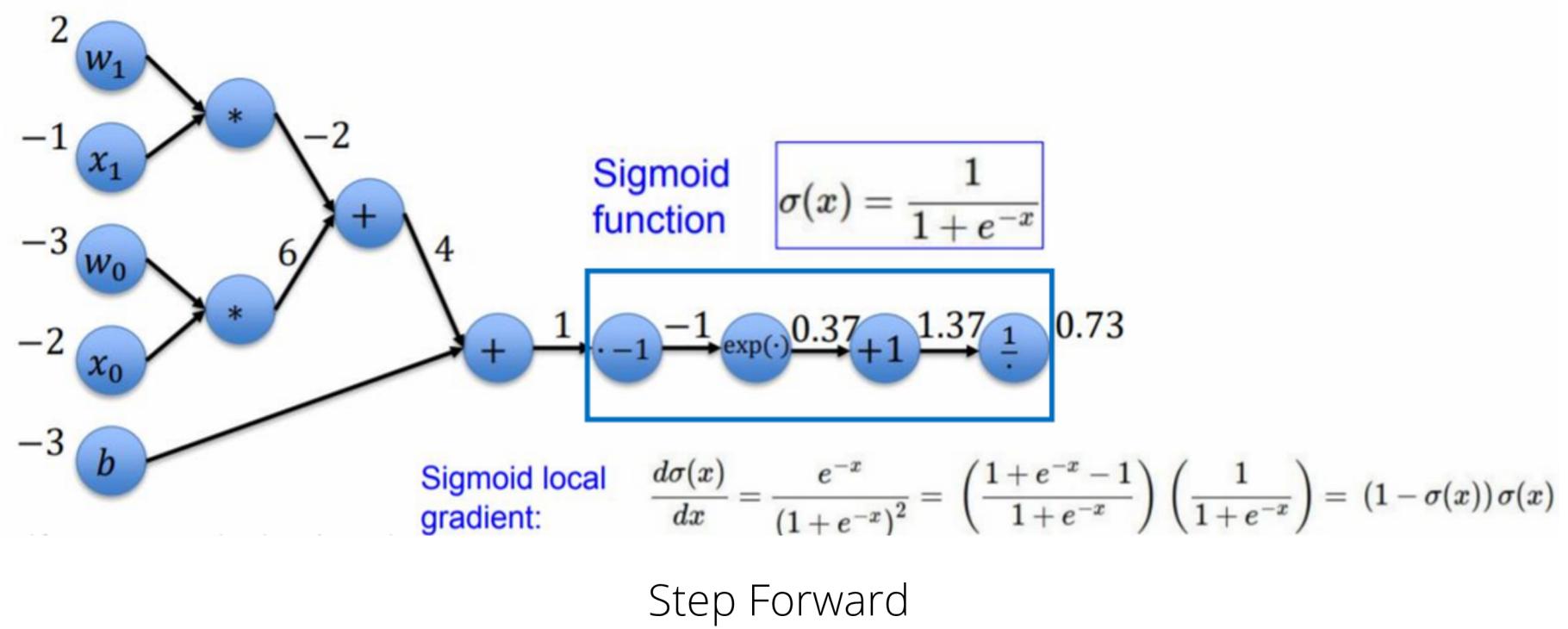
$\hat{L}$  : Cross Entropy Loss

$\lambda$  : Fattore regolarizzazione = 1e-5

Di seguito viene mostrata la regolarizzazione.

La regolarizzazione permette di ridurre l'overfitting penalizzando i pesi grandi. Come fattore di regolarizzazione è stato scelto 1e-5.





# BackWard Propagation

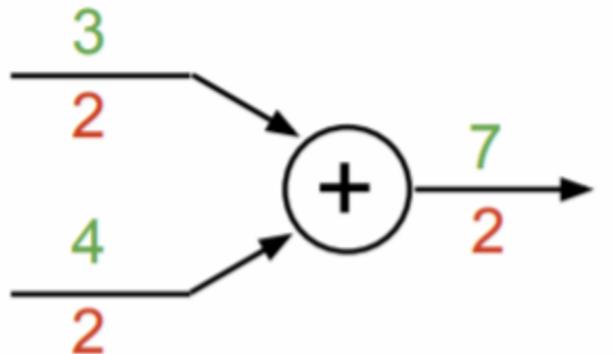
A partire dalla funzione di loss si calcolano i gradienti della funzione di perdita rispetto ai pesi e ai bias del modello.

Per velocizzare il calcolo si utilizzano i grafi computazionali in combinazione con la regola della catena.

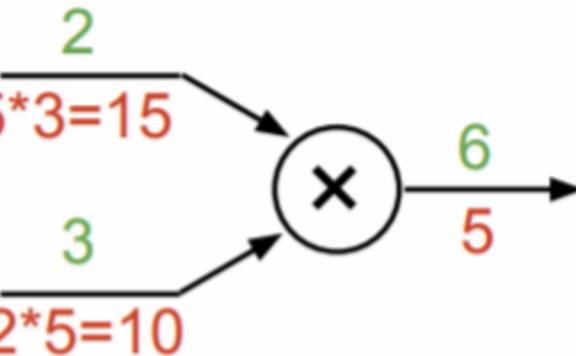
Questi grafi sono una rappresentazione della rete in forma di grafo, i nodi rappresentano le operazioni "a basso livello" e gli archi rappresentano i flussi di dati.

# Semplificazione usando le operazioni a basso livello

**add gate:** gradient distributor



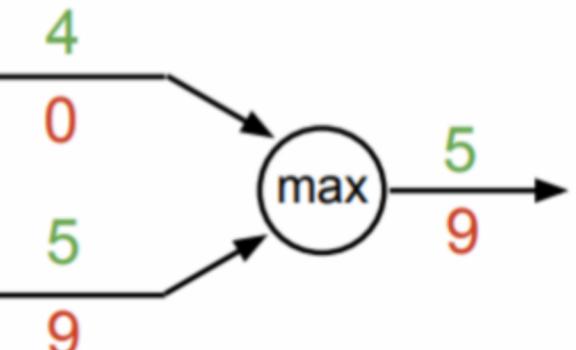
**mul gate:** “swap multiplier”



**copy gate:** gradient adder



**max gate:** gradient router



## **Quattro fasi principali:**

- Inizializzazione
- Aggiornamento dei momenti
- Correzione
- Aggiornamento dei parametri

# **Adam**

L'ottimizzatore Adam (Adaptive Moment Estimation) è utile per l'aggiornamento dei pesi e dei bias nel modello, con conseguente minimizzazione della perdita.

Adam tiene traccia:

- del gradiente (come fa AdaGrad)
- del gradiente quadrato (come fa RMSProp)

Infatti utilizza queste informazioni per aggiornare pesi e bias del modello.



Momento di primo ordine a 0

$$m_0 = 0$$

Momento di secondo ordine a 0

$$v_0 = 0$$

Parametri:

Tasso di apprendimento:  $\eta = 0.001$

Fattore di decadimento

esponenziale per il momento di primo ordine:

$$\beta_1 = 0.9$$

Fattore di decadimento

esponenziale per il momento di secondo ordine:

$$\beta_2 = 0.999$$

Termine di stabilizzazione:

$$\epsilon = 10^{-8}$$

# Adam: Inizializzazione

L'ottimizzatore Adam (Adaptive Moment Estimation) è utile per l'aggiornamento dei pesi e dei bias nel modello, con conseguente minimizzazione della perdita.

Adam tiene traccia:

- del gradiente (come fa AdaGrad)
- del gradiente quadrato (come fa RMSProp)

Infatti utilizza queste informazioni per aggiornare pesi e bias del modello.



A ogni iterazione t:

Gradiente corrente della funzione di costo rispetto a pesi o bias :  $g_t$

Aggiornamento del momento di primo ordine  $m_t$  :

$$m_t = \beta_1 + m_{t-1} + (1 - \beta_1) * g_t$$

In sostanza si combina il gradiente corrente col momento precedente  
alleggerendo le variazioni

Aggiornamento del momento di secondo ordine  $m_t$  :

$$v_t = \beta_2 + v_{t-1} + (1 - \beta_2) * g_t^2$$

In sostanza si combina il gradiente corrente  
col momento di secondo ordine precedente,

normalizzando l'aggiornamento dei pesi in base alla "magnitude" dei gradienti

Questo comporta una riduzione dell'aggiornamento  
quando i gradienti sono grandi

e un aumento dell'aggiornamento quando sono piccoli

# Adam: Aggiornamento momenti

L'ottimizzatore Adam (Adaptive Moment Estimation) è utile per l'aggiornamento dei pesi e dei bias nel modello, con conseguente minimizzazione della perdita.

Adam tiene traccia:

- del gradiente (come fa AdaGrad)
- del gradiente quadrato (come fa RMSProp)

Infatti utilizza queste informazioni per aggiornare pesi e bias del modello.

# Adam: Correzione

Essendo che i momenti sono inizializzati a zero  
nelle prime iterazioni si ha una tendenza verso valori molto piccoli

Per mitigare questo fenomeno si fa una "correzione"

Tenendo conto che i momenti siano sottostimati

Correzione primo ordine:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

Correzione del secondo ordine:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

L'ottimizzatore Adam (Adaptive Moment Estimation) è utile per l'aggiornamento dei pesi e dei bias nel modello, con conseguente minimizzazione della perdita.

Adam tiene traccia:

- del gradiente (come fa AdaGrad)
- del gradiente quadrato (come fa RMSProp)

Infatti utilizza queste informazioni per aggiornare pesi e bias del modello.

# Adam: Aggiornamento parametri

Calcolati i momenti si procede con l'aggiornamento dei parametri

Per i pesi:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} * \hat{m}_t$$

Dove:

$\theta_t$  : pesi all'iterazione t

$\eta$  : tasso di apprendimento o learning rate

$\hat{m}_t$  : momento di primo ordine con correzione

$\sqrt{\hat{v}_t} + \epsilon$  : momento di secondo ordine con correzione

sommato al valore molto piccolo per evitare divisioni per zero

Analogamente si procede con l'aggiornamento di bias

# Scheduler

All'ottimizzatore è stato combinato uno scheduler di tipo **ReduceLROnPlateau**, questo tipo di operazione serve a monitorare una metrica (come la Loss in fase di valutazione) e ridurre il learning rate se questa metrica smette di migliorare. Si è scelto di ridurre il learning rate di 1/10 del suo valore se non c'è miglioramento per 3 epoch consecutive sulla perdita in fase di valutazione.

# Early stopping

Il processo di training è stato raffinato con dei criteri di early stopping, se non migliora la perdita nella valutazione per 5 epochhe il processo si ferma. Questo permette di tagliare i tempi di training nel caso in cui si presentino dei peggioramenti evidenti.

# Trasformazione dei dati

A partire dalle immagini di partenza sono state effettuate alcune trasformazioni:

- Resize: 128x128 per la rete custom, 224x224 per la MobileNet V2
- Conversione in tensori
- Normalizzazione (sottrazione della media e divisione per la stdev):
  - Media: [0.485, 0.456, 0.406]
  - Deviazione standard: [0.229, 0.224, 0.225]

# Epoche

Per ogni epoca:

- I dati sono stati caricati a batch di 32
- Opzione shuffle per train set e validation set a True
- Monitorate perdita e accuratezza sia per il training che per la validazione

Possiamo vedere un'epoca come un passaggio di tutti i dati nella rete.

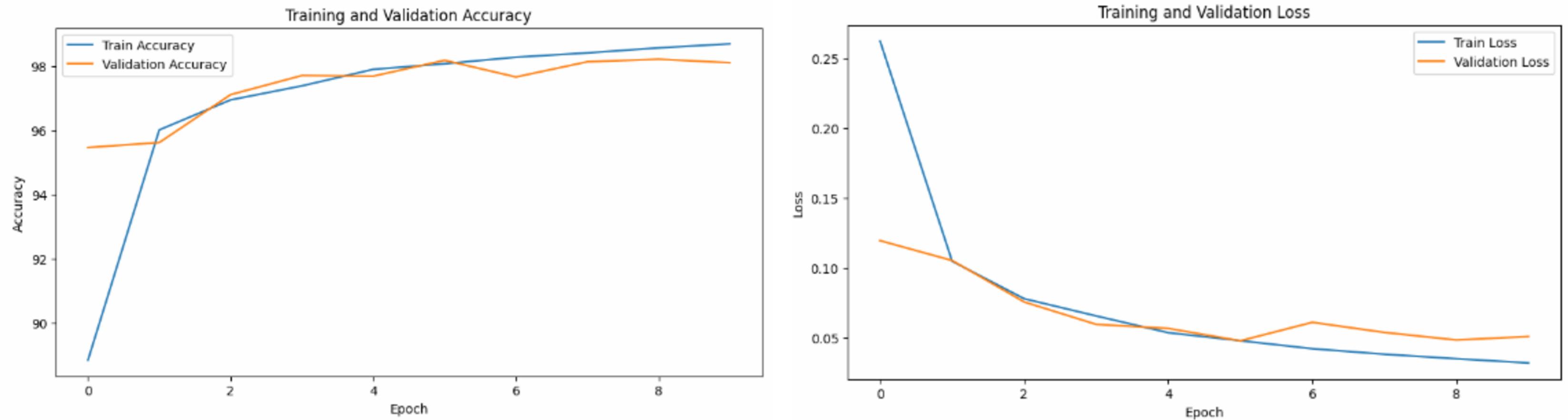
Nel train dei modelli utilizzati nel progetto sono state utilizzate 10 epoche.

Ogni epoca è composta da:

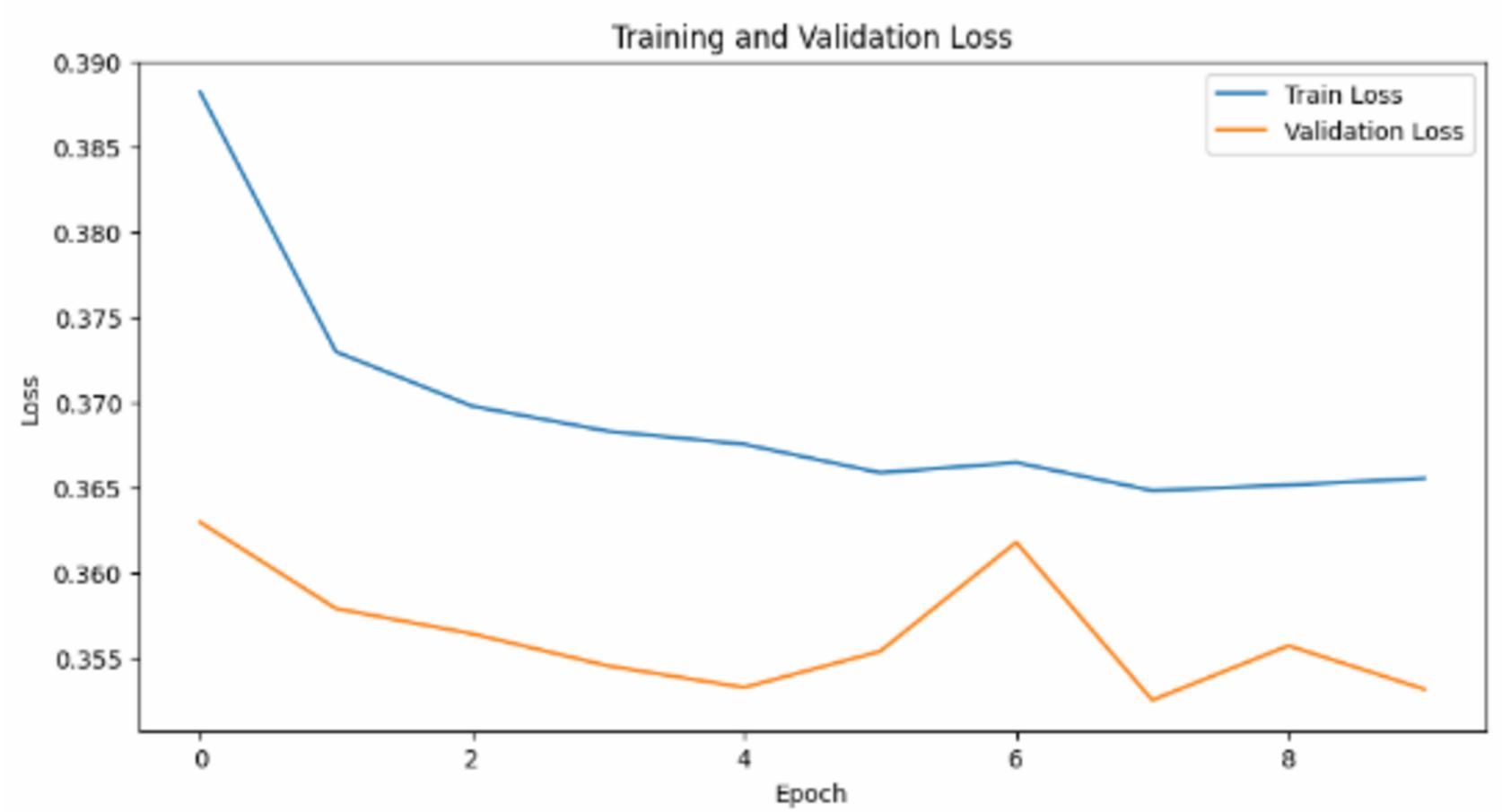
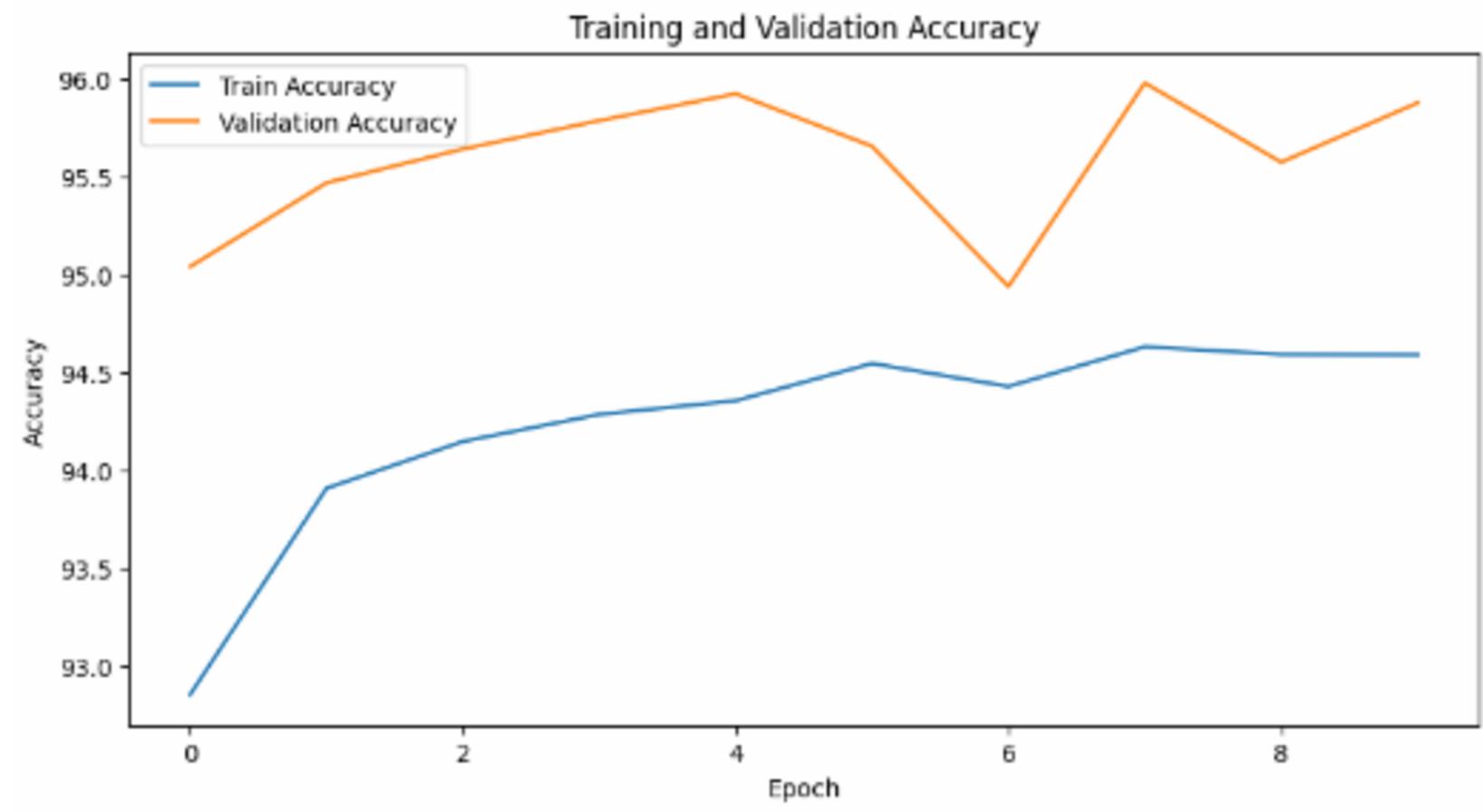
- steps di train (forward e backward)
- steps di valutazione

Il numero di steps dipende da quanti sono i dati e da come vengono suddivisi in batch.

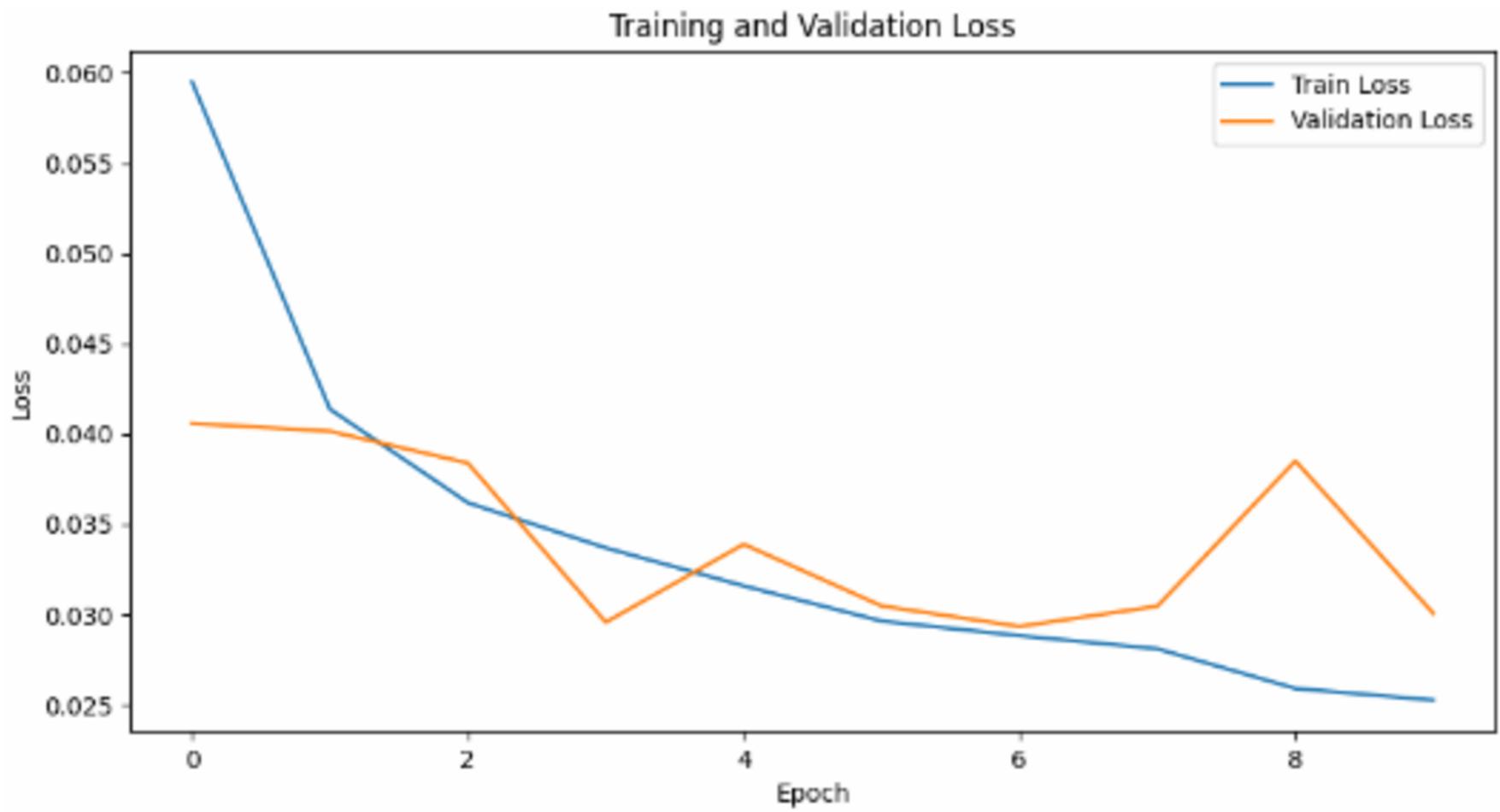
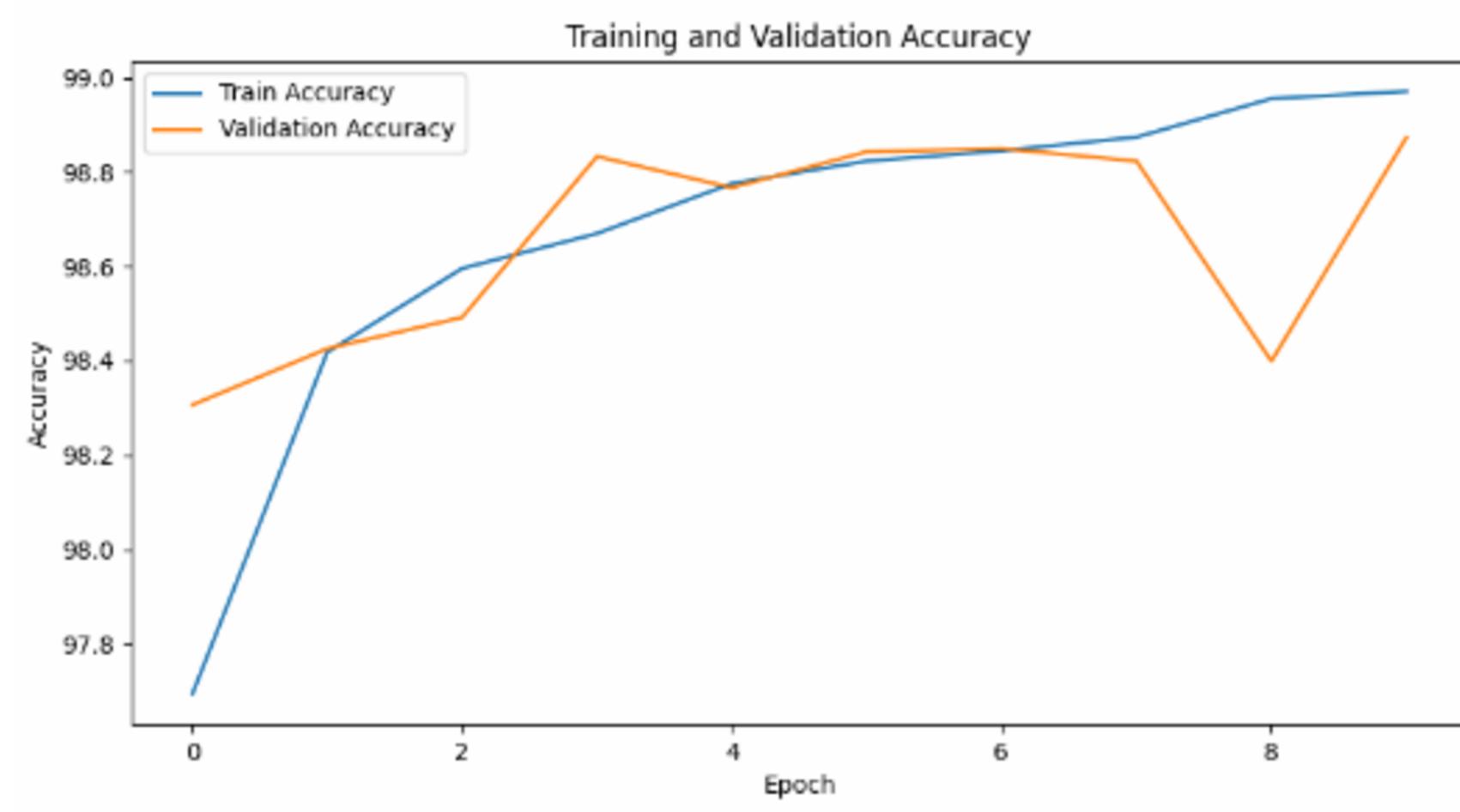




Misurazioni effettuate:  
Rete custom



Misurazioni effettuate:  
MobileNet V2 I



Misurazioni effettuate:  
MobileNet V2 II

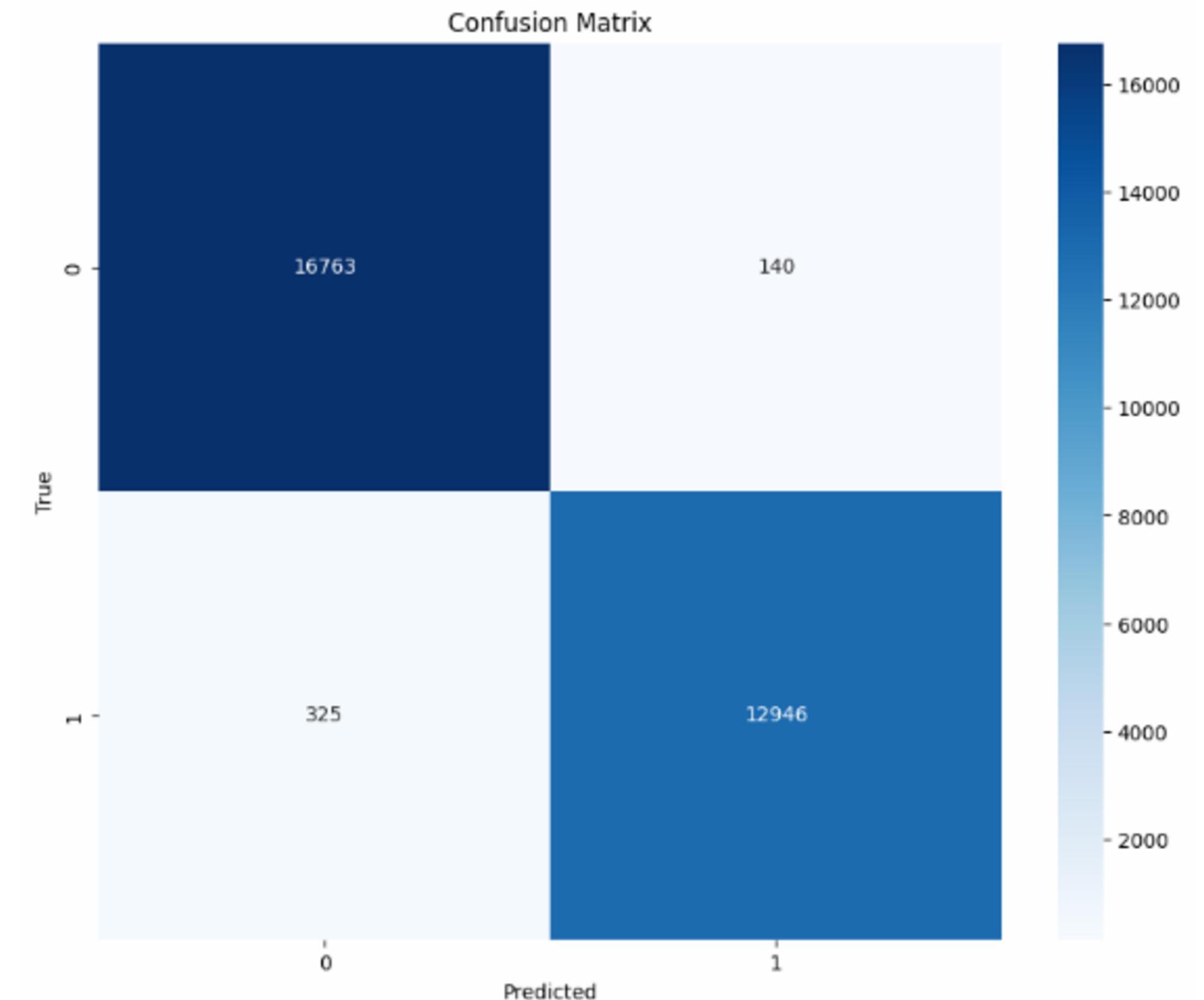
# Valutazione Modelli

Valutazione per metriche

Valutazione sugli errori

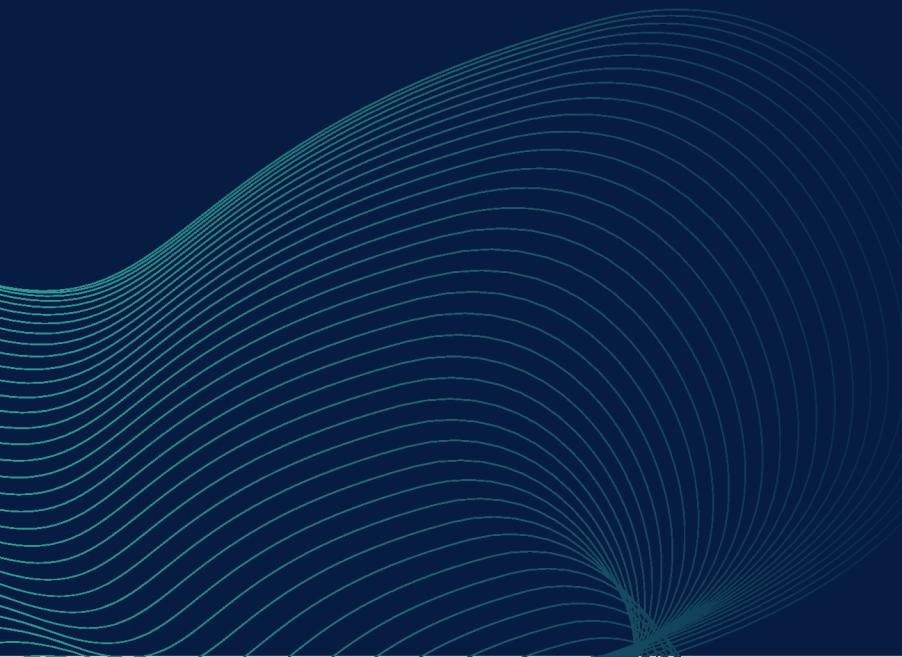
# Rete custom

- **Matrice di confusione**
- Altre metriche



# Rete custom

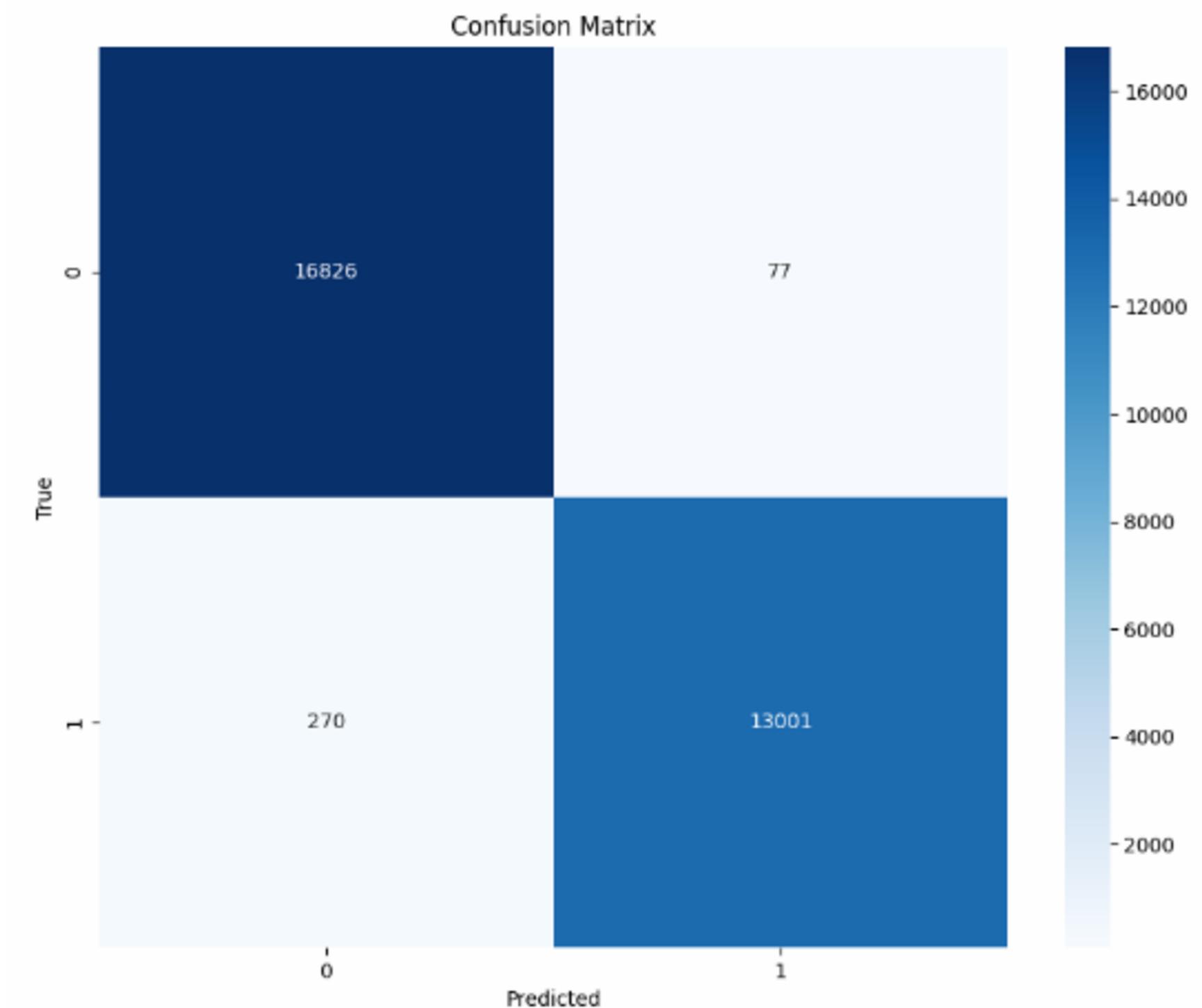
- Matrice di confusione
- **Altre metriche**



1. **Precision:** rapporto tra il numero di veri positivi (campioni correttamente classificati come appartenenti a una classe) e il numero totale di campioni classificati come appartenenti a quella classe (veri positivi + falsi positivi).
  - Precision (classe 0): 0.98
  - Precision (classe 1): 0.99
2. **Recall:** rapporto tra il numero di veri positivi e il numero totale di campioni che appartengono effettivamente a quella classe (veri positivi + falsi negativi).
  - Recall (classe 0): 0.99
  - Recall (classe 1): 0.98
3. **F1 Score:** media armonica tra precisione e richiamo.
  - F1-Score (classe 0): 0.99
  - F1-Score (classe 1): 0.98
4. **Accuracy:** proporzione di campioni correttamente classificati rispetto al numero totale di campioni.
  - Accuracy: 0.98

# Mobile Net V2 II

- **Matrice di confusione**
- Altre metriche



# Mobile Net V2 II

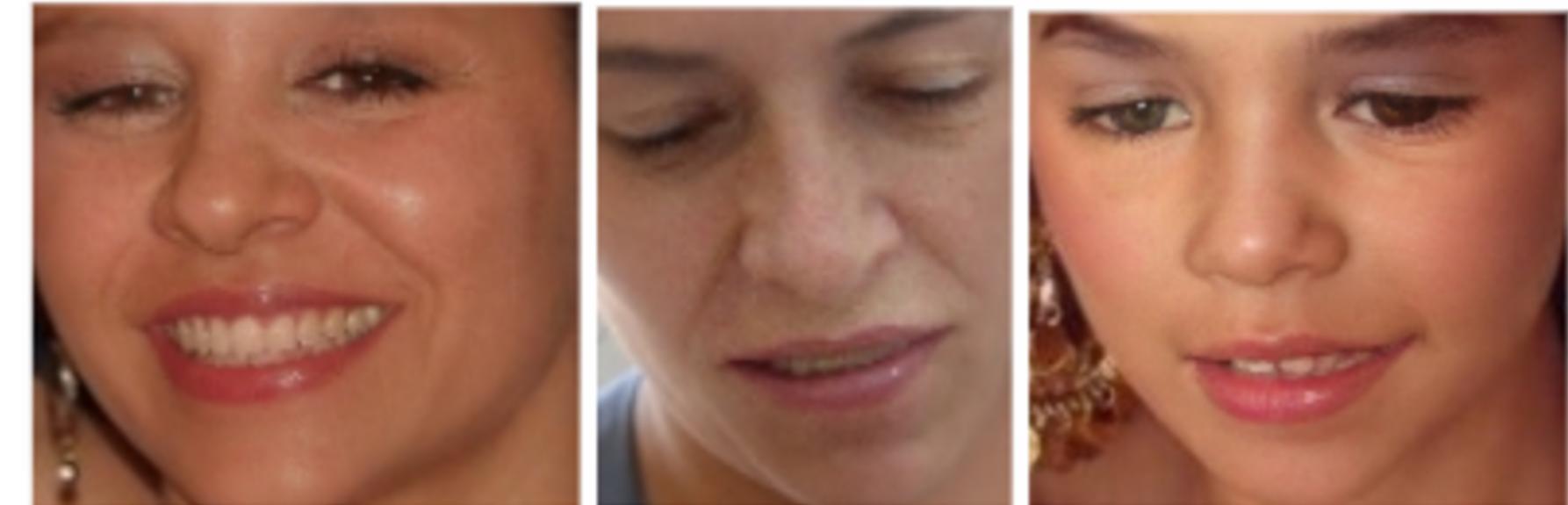
- Matrice di confusione
- **Altre metriche**



1. **Precision:** rapporto tra il numero di veri positivi (campioni correttamente classificati come appartenenti a una classe) e il numero totale di campioni classificati come appartenenti a quella classe (veri positivi + falsi positivi).
  - Precision (classe 0): 0.98
  - Precision (classe 1): 0.99
2. **Recall:** rapporto tra il numero di veri positivi e il numero totale di campioni che appartengono effettivamente a quella classe (veri positivi + falsi negativi).
  - Recall (classe 0): 1.00
  - Recall (classe 1): 0.98
3. **F1 Score:** media armonica tra precisione e richiamo.
  - F1-Score (classe 0): 0.99
  - F1-Score (classe 1): 0.99
4. **Accuracy:** proporzione di campioni correttamente classificati rispetto al numero totale di campioni.
  - Accuracy: 0.99

# Valutazione errori Rete Custom

- Immagini reali ma classificate come deep fake



# Valutazione errori Rete Custom

- Deep fake classificati come visi reali



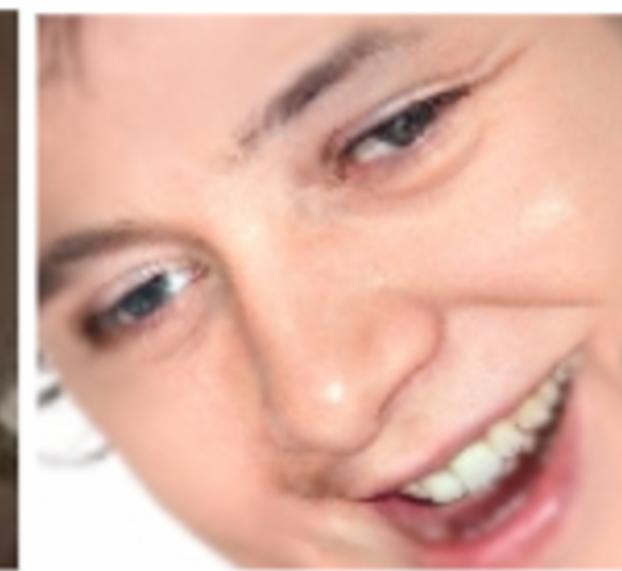
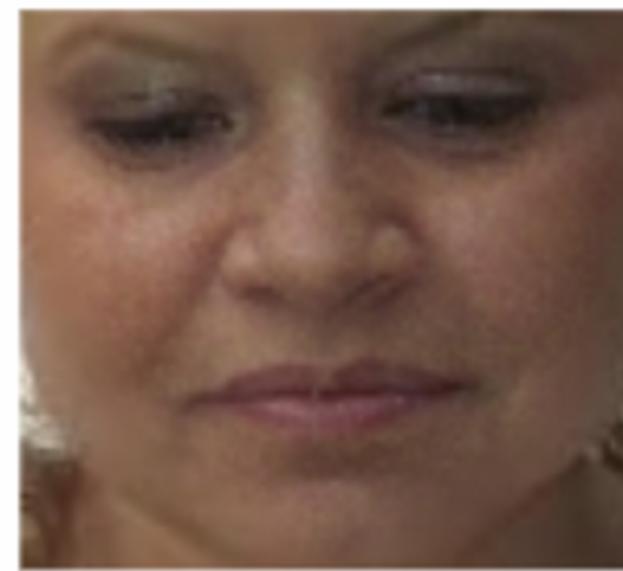
# Valutazione errori MobileNet V2

- Immagini reali ma classificate come deep fake

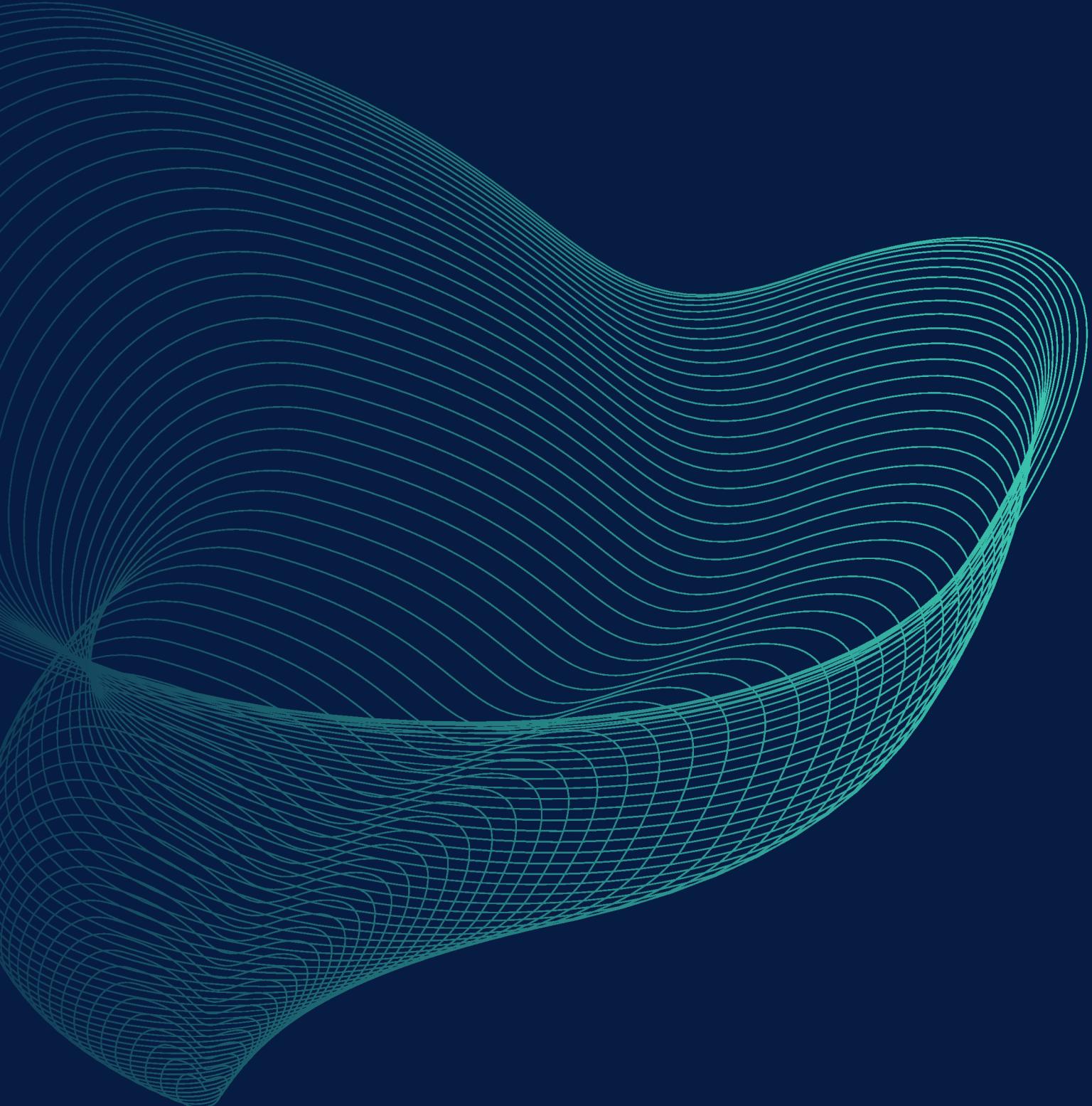


# Valutazione errori Mobile Net V2

- Deep fake classificati come visi reali



---



Fine

