Deep Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2022 - 2023

Convolutional Neural Networks (CNN)
5a parte - Batch Normalization e ResNet

Sommario

- Internal covariate shift
- Batch normalization
- Architettura Residual Network (ResNet)

Motivazioni

- L'addestramento delle architetture Deep mostra alcune problematiche aggiuntive oltre quelle già discusse per le MLP. Una significativa è il tempo necessario per addestrarle.

- Spesso si operano *standardizzazioni* nei valori delle features in ingresso con forme di pre-processamento, es:
  - imporre μ=0 (zero mean) o la unit-variance (cioè dividere per la stddev)
  - *zero mean* sul valore delle features, considerando la singola istanza; spesso utile per dati con informazioni spaziali.
- Ci garantisce che durante l'addestramento i valori dei parametri rimangano in intervalli ottimali, sia considerando i layer per l'intera profondità della rete, sia tra i nodi di un singolo layer, sia tra i valori di ogni parametro per la durata dell'addestramento.

Motivazioni

- Inoltre un layer che produce valori di attivazione molto elevati rispetto agli altri (es. x100) richiede aggiustamenti (es. modificando il learning rate in modo adattivo per produrre variazioni più efficaci durante il training).

- Infine, per affrontare l'overfitting è spesso utile introdurre *regolarizzazioni* sul valore dei parametri (es. aggiungendo del rumore).

Internal covariate shift

- Con **Internal covariate shift** si indica la circostanza in cui **la distribuzione dei valori di attivazione nella rete cambia a causa della variazione dei parametri durante il training**.
  - Fenomeno fondamentale nelle architetture deep (con molti layers).
    - E' chiaro che i parametri influenzano le attivazioni, ma **la distribuzione dei valori** non dovrebbe alterarsi a causa dei parametri.
  - Il vanishing/exploding gradient ricade in questa circostanza.
  - ReLU, e le sue varianti, riducono il fenomeno ma non lo escludono.
- L'obiettivo è ridurre il **covariance shift** all'interno della reti.

Batch Normalization

- La **batch normalization** (**BN**) è una tecnica per affrontare tale problema. Prima della funzione di attivazione di ogni layer:
  - **Normalizza gli input**, centrandoli in 0 e dividendoli per la deviazione standard σ.
  - Introduce **2 parametri**, uno per determinare la **scalatura** e uno per lo **shift**. Tali parametri saranno soggetti ad addestramento.
- Dopo la normalizzazione **la rete apprende il valore medio e la scala più giusta degli input per ogni layer**.
  - La normalizzazione è frequente nei dati in ingresso degli approcci basati su ML. La tecnica proposta estende tale tecnica ad ogni layer della rete.
- Per normalizzare bisogna prima conoscere valor medio e varianza dei dati. Si stimano entrambi **impiegando mini-batch,** cioè un piccolo sottoinsieme del training set. Da questo il termine **batch normalization**.

Batch Normalization

- Consiste in un **algoritmo** applicato ad ogni singola istanza in input x_i, considerando un mini-batch B di m istanze con **media** μ_B e **varianza** σ_B^2
- I parametri da apprendere durante il training sono γ (**scale**) e β (**offset**).

ε è una costante aggiunta alla varianza per evitare divisione per 0 (es. 10^{-3})

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

Trasformazione lineare

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

da Ioffe e Szegedy "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" 2015

Batch Normalization - Considerazioni

- La tecnica BN può essere impiegata sui singoli layer, soprattutto sugli hidden, oppure all'intera rete.
- La stima di media e deviazione standard sono ricavate sul mini batch corrente.
- Possiamo interpretare i parametri *scale* e *offset* stimati durante l'apprendimento come un mezzo per "recuperare" i gradi di libertà persi a causa della normalizzazione e limitarsi a considerare mini-batch invece dell'intero dataset.
- Con mini-batch di dimensione adeguata (un iperparametro da definire) si raggiungono buoni incrementi di prestazioni e una *stabilità* nell'andamento. Ma richiede un tuning che dipende dai dati impiegati.
- La tecnica non permette al valore dei parametri di divergere. Inoltre permette di incrementare il *learning rate*.

Batch Normalization - Considerazioni (2)

- Può sembrare illogico introdurre approssimazioni nei valori di media e deviazione standard, ma nella pratica rappresentano una sorta di rumore introdotto artificialmente che garantisce tempi più rapidi e minor effetto overfitting.
- Valori spesso ottimali della dimensione del mini batch sono 50-100 istanze, che garantiscono la giusta *quantità di rumore* introdotto durante l'apprendimento.

Batch Normalization - Considerazioni (3)

- In casi particolari ***γ*** e **β** possono assumere valori tali da "invertire" il processo di normalizzazione degli input del processo di *batch normalization*, se questo garantisce l'ottimalità durante il training.
- La BN può essere vista come una **trasformazione lineare**, perciò facilmente **differenziabile** durante il calcolo dei gradienti.
- La normalizzazione basata su mini-batch è essenziale per garantire l'efficienza di tale tecnica, ma è inutile nel test e in produzione.
  - **In produzione vogliamo una rete che renda l'output dipendente unicamente e deterministicamente dall'input**, perciò non influenzata dallo specifico mini-batch.
- Per tale motivo la normalizzazione sarà calcolata sull'intera popolazione **{x}** con valori di media e varianza costanti durante l'elaborazione:

$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Batch Normalization layer

- Poiché la BN dipende dalla dimensione del mini batch, e perciò dai dati di training, non possiamo ignorarla quando definiamo la nostra architettura.
- Per reti FC si può applicare la BN tra la trasformazione lineare e il calcolo della funzione di attivazione.
- Per conv layers l'approccio è simile, ma consideriamo la BN per ogni singolo canale, valutandola sui i dati sparsi spazialmente. Perciò ogni canale avrà una stima diversa di media e deviazione standard.
- Questo è in linea col principio di *invarianza spaziale*, cioè nel calcolo possiamo ignorare l'informazione relativa alla posizione.

Batch Normalization - Altri vantaggi

- Si dimostra empiricamente che la BN, oltre a ridurre il vanishing gradients, garantisce ulteriori benefici:

Batch Normalization - Altri vantaggi

- Si dimostra empiricamente che la BN, oltre a ridurre il vanishing gradients, garantisce ulteriori benefici:
  - Permette di impiegare **funzioni di attivazione che saturano** per input molto grandi o piccoli (es. logistic e tanh).

Batch Normalization - Altri vantaggi

- Si dimostra empiricamente che la BN, oltre a ridurre il vanishing gradients, garantisce ulteriori benefici:
  - Permette di impiegare **funzioni di attivazione che saturano** per input molto grandi o piccoli (es. logistic e tanh).
  - **Riduce la dipendenza** sugli effetti di una certa **scelta dei parametri iniziali**.

Batch Normalization - Altri vantaggi

- Si dimostra empiricamente che la BN, oltre a ridurre il vanishing gradients, garantisce ulteriori benefici:
  - Permette di impiegare **funzioni di attivazione che saturano** per input molto grandi o piccoli (es. logistic e tanh).
  - **Riduce la dipendenza** sugli effetti di una certa **scelta dei parametri iniziali**.
  - Richiamo: Introduce una certa **regolarizzazione** dei parametri, sebbene non sostituisce le tecniche più efficaci (es. dropout)
  - Richiamo: Permette l'uso di **learning rate più elevati**, riducendo i tempi di apprendimento.
    - Es. Per un tipico task di image classification, si ottengono incrementi x14.

Batch Normalization

Perché non ci limitiamo a normalizzare i dati in input ad ogni layer
e lasciare alla rete determinare i parametri W per l'ottimalità input-output?

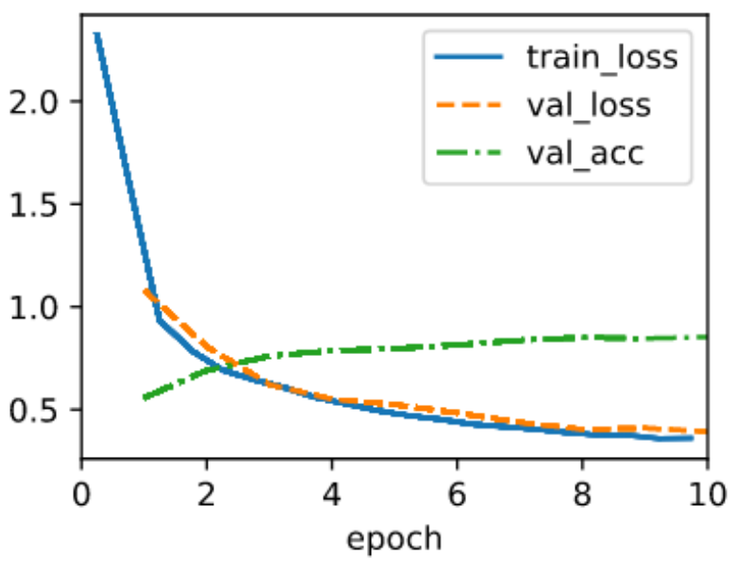
Batch Normalization

Perché non ci limitiamo a normalizzare i dati in input ad ogni layer e lasciare alla rete determinare i parametri W per l'ottimalità input-output?

- Se impieghiamo funzioni di attivazioni logistiche, **forziamo al rete a lavorare in regime di quasi-linearità**, riducendo la capacità di costruire relazioni input-output non lineari.

Batch Normalization e LeNet - Keras

```
import tensorflow as tf
!pip install d2l==1.0.0a1.post0
from d2l import tensorflow as d2l

class BNLeNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6, kernel_size=5,
                                   input_shape=(28, 28, 1)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
            tf.keras.layers.Conv2D(filters=16, kernel_size=5),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
            tf.keras.layers.Flatten(), tf.keras.layers.Dense(120),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.Dense(84),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.Dense(num_classes)])

trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128)
with d2l.try_gpu():
    model = BNLeNet(lr=0.5)
    trainer.fit(model, data)
```

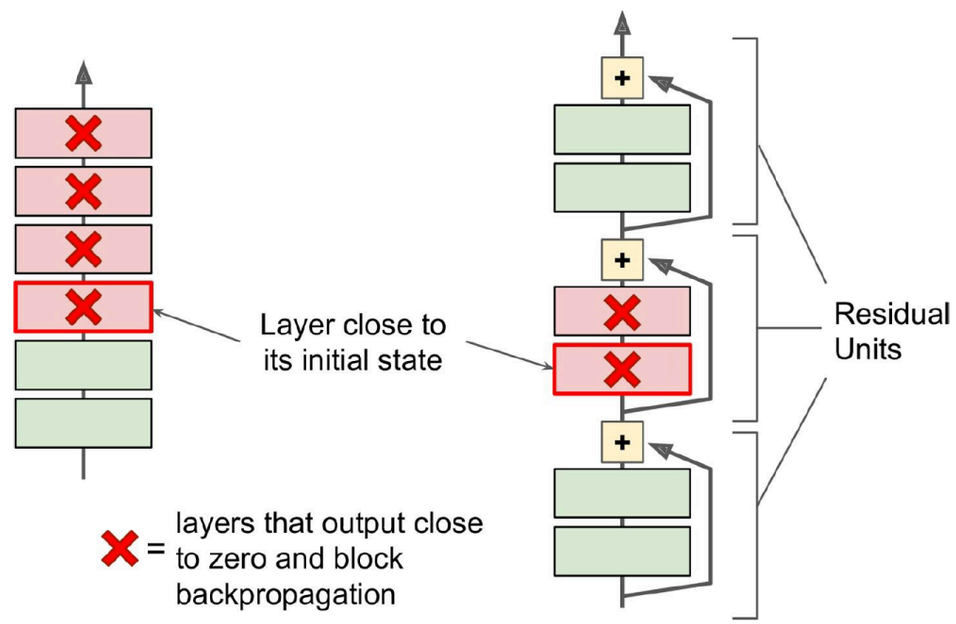


Architettura Residual Network - Motivazioni

- In generale, architetture di reti complesse (es. più profonde) possono stimare una classe più ampia di funzioni. Ma se aggiungiamo layer, nessuno ci garantisce che l'apprendimento ci permette di trovarle, anzi in taluni casi possiamo allontanarci dall'ottimo.
- Inoltre reti profonde potrebbero soffrire del vanishing gradient problem.
- Ma se aggiungiamo layer che mirano a stimare una funzione identità, i.e., f(x)=x, sicuramente manteniamo la stessa efficacia della rete iniziale.
- L'ipotesi è che, i layer che aggiungiamo alla rete dovrebbero avere più probabilità nel rappresentare funzioni identità per garantire prestazioni ottimali.

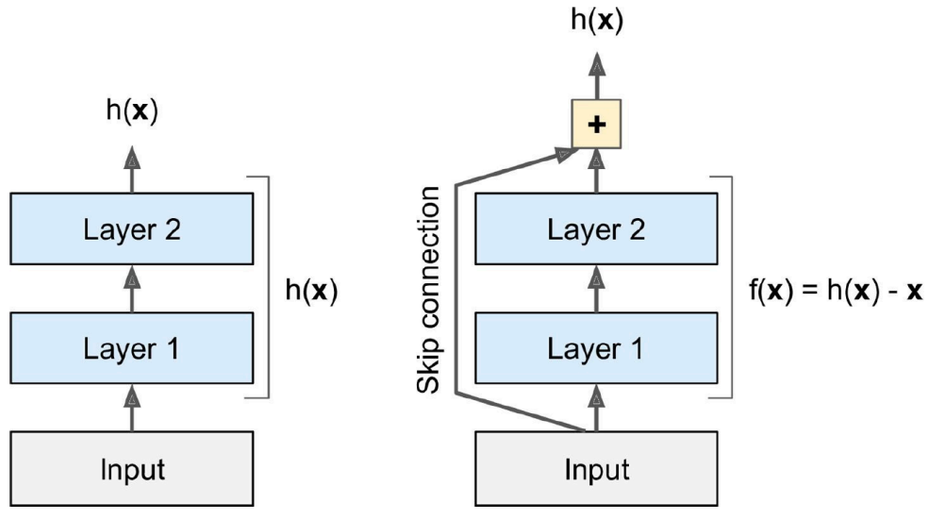
Architettura Residual Network (ResNet)

- **ResNet** è stata presentata a ILSVRC 2015 (top-5: 3.6%) e consiste in 152 layers.
- Si introducono le **skip connections**, che propagano l'output di un certo layer nell'input di un layer che è posizionato più a valle.
  - L'ipotesi è di rendere **più semplice e veloce propagare segnali** su varie parti della rete.
  - Nelle fasi iniziali (comportamento random) si obbliga parti della rete ad comportarsi in modo da riproporre i valori in input, rendendo **più veloce l'apprendimento**.

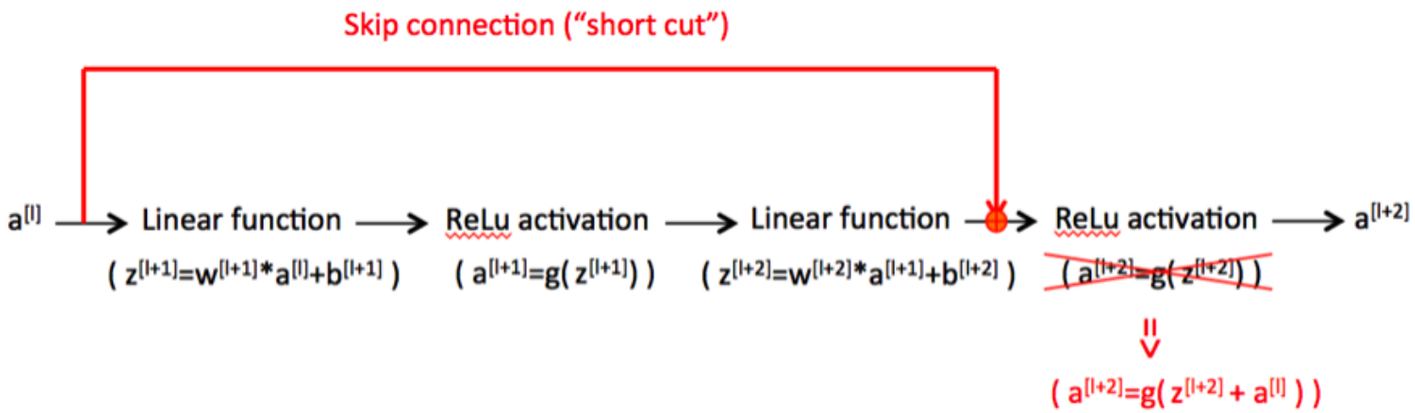


ResNet: Residual learning e residual block

- Addestrare una rete neurale può essere interpretato come approssimare una funzione h(**x**). Se aggiungi un valore x all'output della rete, allora la rete è obbligata a modellare la funzione f(**x**) = h(**x**) - **x**. Tale approccio è chiamato **residual learning o mapping**.

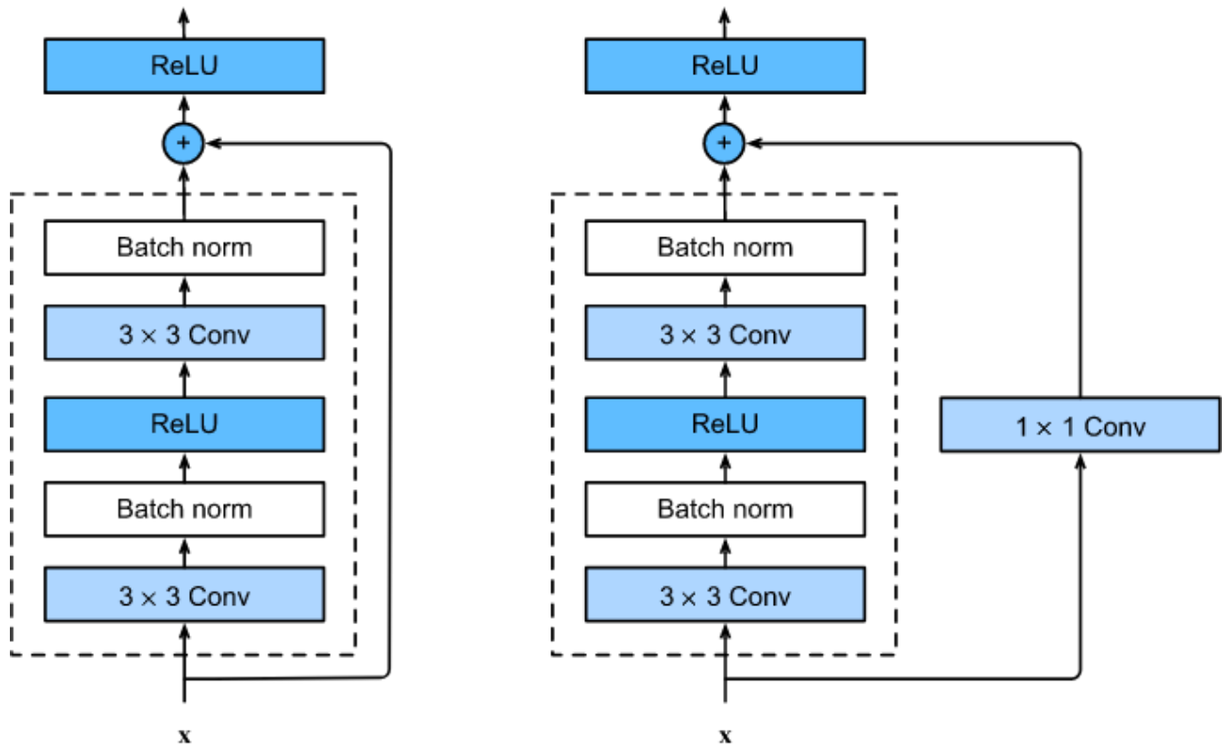


- Dal punto di vista operativo, è sufficiente combinare l'output di un layer con l'output di un layer posizionato più a monte prima di valutare la funzione di attivazione (ReLU).



Architettura ResNet

- L'architettura ResNet impiega conv layer 3x3 (simili a VGG).
- Ogni blocco ResNet ha due 3x3 conv layer seguite dalla batch normalization e attivazione ReLU. Prima dell'ultima ReLU sommiamo l'input dalla skip connection.
- La 1x1 conv layer è necessaria per adattare i canali dell'input con quelli ottenuti a valle del blocco.



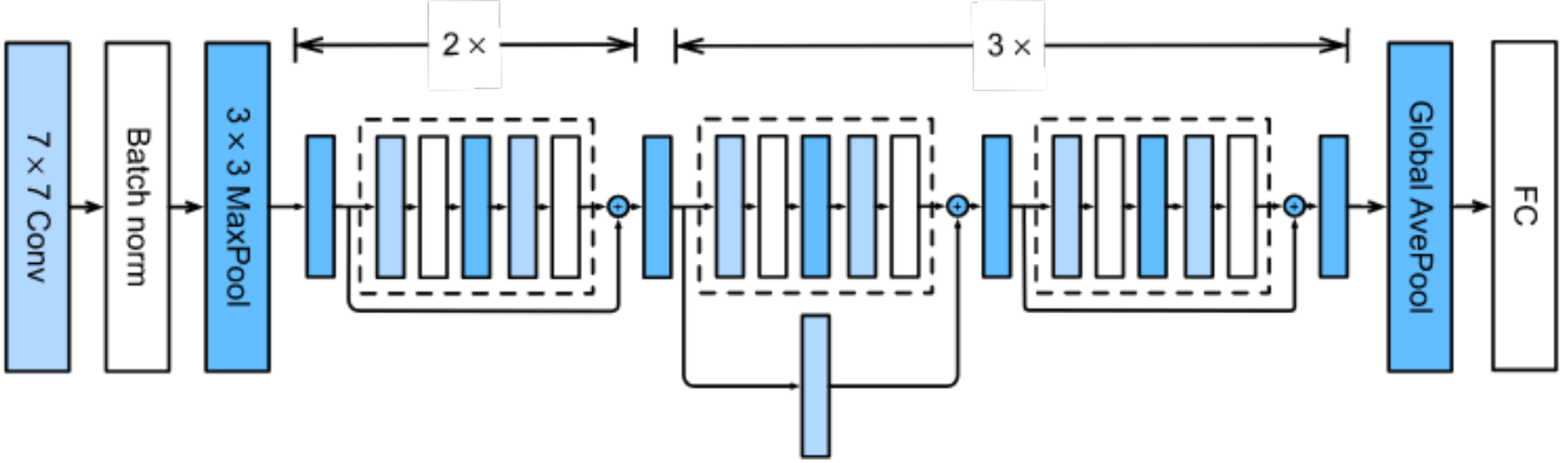
Blocco ResNet e Keras

```
class Residual(tf.keras.Model):

    def __init__(self, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(num_channels, padding='same',
                                            kernel_size=3, strides=strides)
        self.conv2 = tf.keras.layers.Conv2D(num_channels, kernel_size=3,
                                            padding='same')
        self.conv3 = None

        # dipende se vogliamo usare o meno il 1x1 conv layer
        if use_1x1conv:
            self.conv3 = tf.keras.layers.Conv2D(num_channels, kernel_size=1,
                                                strides=strides)
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.bn2 = tf.keras.layers.BatchNormalization()

    def call(self, X):
        Y = tf.keras.activations.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3 is not None:
            X = self.conv3(X)
        Y += X
        return tf.keras.activations.relu(Y)

blk = Residual(3)
X = tf.random.normal((4, 6, 6, 3))
Y = blk(X)
Y.shape
```

```
TensorShape([4, 6, 6, 3])
```

Architettura ResNet-18

- I primi layer di ResNet sono simili a GoogleNet, ma in ResNet si usa la Batch normalization.
- Seguono vari moduli ripetuti ResNet. La ResNet-18 include 18 layer totali, ma si hanno modelli addestrati con molti più layer, es. ResNet-152.



Architettura ResNet e Keras

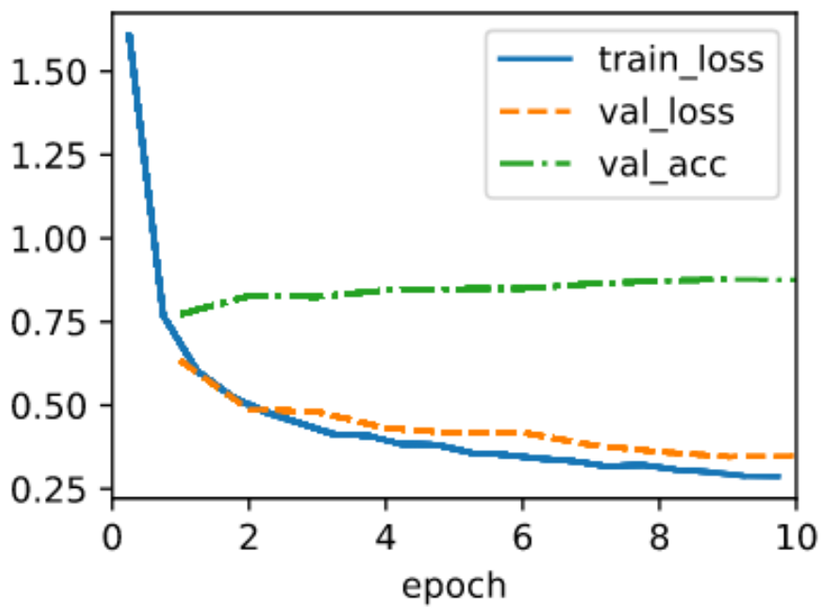
```
class ResNet(d2l.Classifier):
    def b1(self):
        return tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(64, kernel_size=7, strides=2,
                                   padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('relu'),
            tf.keras.layers.MaxPool2D(pool_size=3, strides=2,
                                      padding='same')])
```

```
@d2l.add_to_class(ResNet)
def block(self, num_residuals, num_channels, first_block=False):
    blk = tf.keras.models.Sequential()
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.add(Residual(num_channels, use_1x1conv=True, strides=2))
        else:
            blk.add(Residual(num_channels))
    return blk
```

```
@d2l.add_to_class(ResNet)
def __init__(self, arch, lr=0.1, num_classes=10):
    super(ResNet, self).__init__()
    self.save_hyperparameters()
    self.net = tf.keras.models.Sequential(self.b1())
    for i, b in enumerate(arch):
        self.net.add(self.block(*b, first_block=(i==0)))
    self.net.add(tf.keras.models.Sequential([
        tf.keras.layers.GlobalAvgPool2D(),
        tf.keras.layers.Dense(units=num_classes)]))
```

Architettura ResNet e Keras

```
class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((2, 64), (2, 128), (2, 256), (2, 512)),
                         lr, num_classes)

ResNet18().layer_summary((1, 96, 96, 1))

Sequential output shape:     (1, 24, 24, 64)
Sequential output shape:     (1, 24, 24, 64)
Sequential output shape:     (1, 12, 12, 128)
Sequential output shape:     (1, 6, 6, 256)
Sequential output shape:     (1, 3, 3, 512)
Sequential output shape:     (1, 10)

trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
with d2l.try_gpu():
    model = ResNet18(lr=0.01)
    trainer.fit(model, data)
```



CNN - Esercizio

Quali sono le innovazioni di AlexNet rispetto a LeNet-5? E per quanto riguarda GoogleLeNet e ResNet?

CNN - Esercizio

Quali sono le innovazioni di AlexNet rispetto a LeNet-5? E per quanto riguarda GoogleLeNet e ResNet?

- **AlexNet** è più profonda e ampia rispetto a LeNet-5, e crea stack di convolutional layer uno sull'altro, invece di alternarli con pooling layer.

CNN - Esercizio

Quali sono le innovazioni di AlexNet rispetto a LeNet-5? E per quanto riguarda GoogleLeNet e ResNet?

- **AlexNet** è più profonda e ampia rispetto a LeNet-5, e crea stack di convolutional layer uno sull'altro, invece di alternarli con pooling layer.
- **GoogleNet** impiega inception modules, che permettono di avere reti ancora più profonde ma con meno parametri rispetto alle precedenti.

CNN - Esercizio

Quali sono le innovazioni di AlexNet rispetto a LeNet-5? E per quanto riguarda GoogleLeNet e ResNet?

- **AlexNet** è più profonda e ampia rispetto a LeNet-5, e crea stack di convolutional layer uno sull'altro, invece di alternarli con pooling layer.
- **GoogleNet** impiega inception modules, che permettono di avere reti ancora più profonde ma con meno parametri rispetto alle precedenti.
- **ResNet** introduce le skip connections, che permettono un numero di layer oltre i 100. Anche la relativa semplicità la contraddistingue.

Esercizio su CNN e MNIST

- Prova costruire una tua architettura CNN (cioè con uno o più convolution layers, pooling layers, etc) per raggiungere la migliore accuratezza per i dataset MNIST.
  - MNIST dataset: http://yann.lecun.com/exdb/mnist/
  - MNIST e Tensorflow: https://www.tensorflow.org/quantum/tutorials/mnist