Convolutional neural networks (CNNs)

Operatia de convolutie in cazul 2D discret:

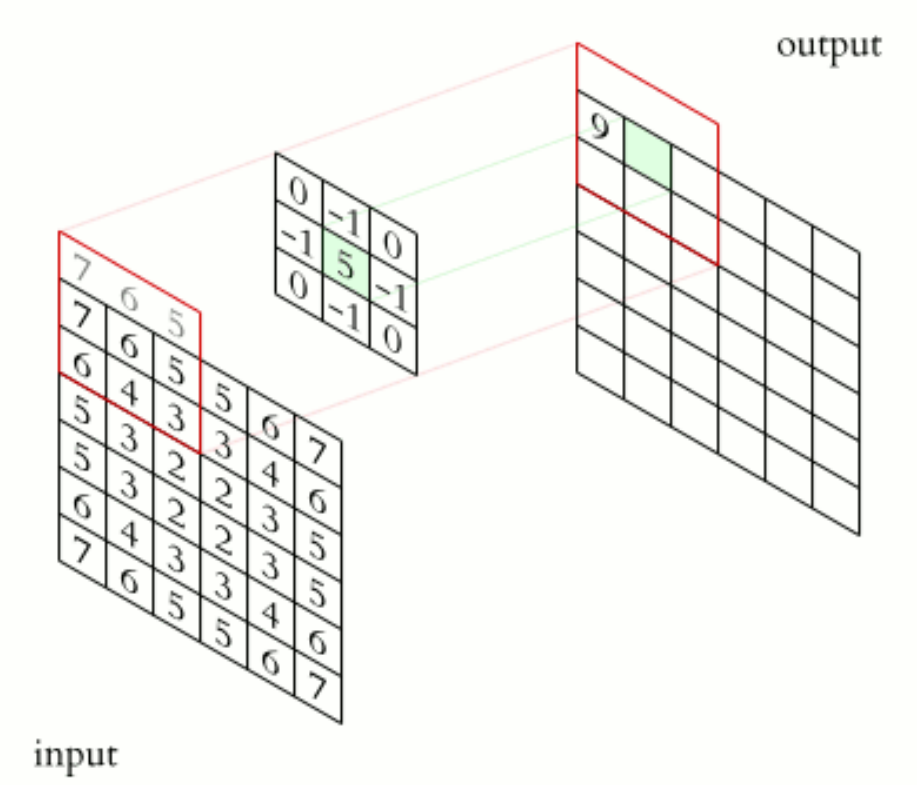
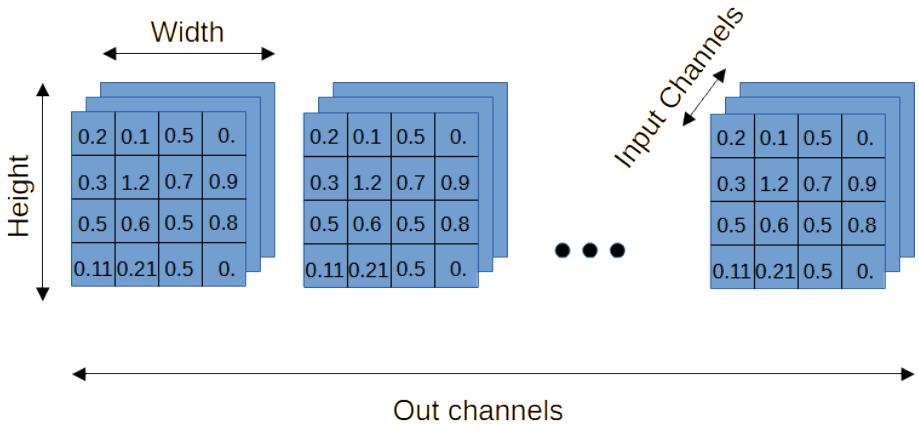


Image source: https://en.wikipedia.org/wiki/Convolution#/media/File:2D_Convolution_Animation.gif

Definirea stratului convolutional:

`nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)`

Tensorul asociat stratului convolutional anterior:



Definirea stratului de pooling:

`nn.MaxPool2d(kernel_size, stride)`

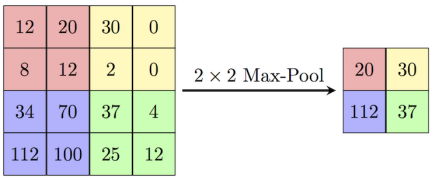


Image source: https://paperswithcode.com/method/max-pooling

Proprietati stratul convolutional

1. **parameter sharing**

Intr-o retea fully connected (MLP) un parametrul al acesteia este folosit o singura data in procesarea unui exemplu, pe cand in CNN un parametru (o pozitie dintr-un filtru) este folosit pentru fiecare pozitie a tensorului de intrare (daca stride-ul este 1).

1. **local connectivity**

In CNNs o valoare din mapa de activare rezulta din aplicarea filtrului pe o zona locala, pe cand in MLP fiecare neuron este conectat cu intreg semnalul de intrare.

1. **equivariance to translation**

Un filtru va produce acelasi nivel de activare pentru un pattern in semnal, indiferent de locul unde pattern-ul respectiv se afla.

Proprietati stratul de pooling

1. **Invarianta la mici translatii** datorata sumarizarii cu o statistica a zonelor din semnal.

O retea convolutionala:

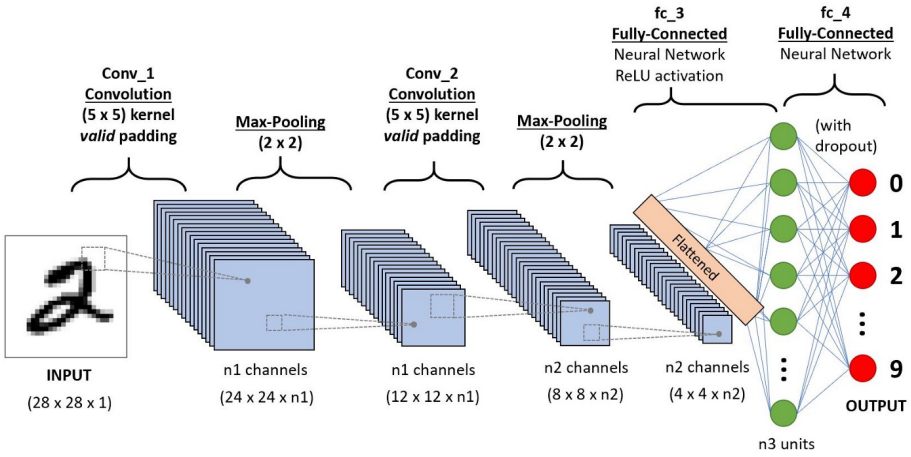


Image source: https://paperswithcode.com/methods/category/convolutional-neural-networks

In imaginea de mai sus se foloseste **Flatten** pentru a transforma tensorul din forma 3D intr-unul 1 dimensional. O alta varianta este sa aplicam pooling + squeeze pe dimensiunile spatiale. In Pytorch putem implementa asta cu **nn.AdaptiveMaxPool2d**: https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveMaxPool2d.html#torch.nn.AdaptiveMaxPool2d

Exemplu:

```
import torch.nn as nn
pooling = nn.AdaptiveMaxPool2d((1, 1))
result = pooling(torch.rand(5, 64, 4, 4))
print(f"Shape dupa operatia de pooling:{result.shape}")
print(f"Shape dupa squeeze:{torch.squeeze(result).shape}")
```

```
Shape dupa operatia de pooling:torch.Size([5, 64, 1, 1])
Shape dupa squeeze:torch.Size([5, 64])
```

3. Exercitii

1. Creati reteaua din imaginea de mai sus, dar cu dimensiunea filtrelor (kernel size) setata la 3. Antrenati si testati performanta retelei pe setul de date **CIFAR10**.

a) folositi $n1 = 32$, $n2 = 64$ si $n3 = 128$ si adaugati activarea ReLU dupa straturile convolutionale.

b) folositi $n1 = 64$, $n2 = 128$ si $n3 = 256$ si adaugati activarea ReLU dupa straturile convolutionale.

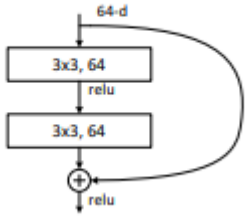
c) aceeasi configuratie ca la punctul b), dar setati kernel size la 5 pentru primul strat.

```
trainset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=ToTensor()) testset = datasets.CIFAR10(root='./data',
train=False, download=True, transform=ToTensor())
```

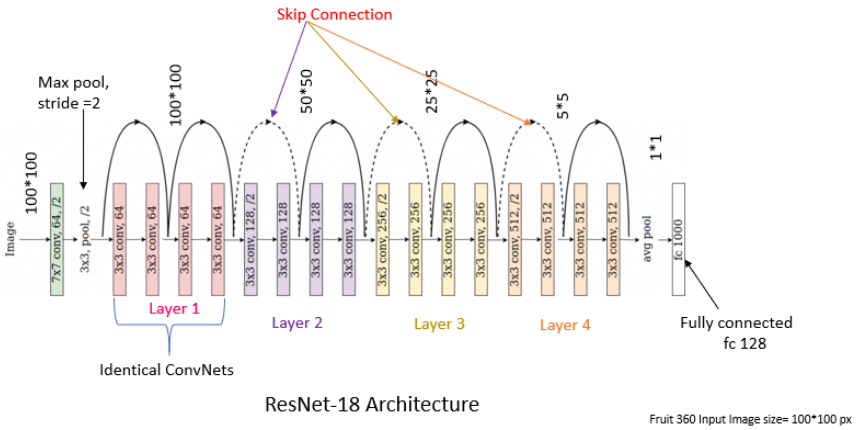
1. Introduceti straturi BatchNorm2D in reteaua de la exercitiul precedent. Testati din nou performanta pe setul de date CIFAR10. Ce observati? Documentatie: https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html
2. Inlocuiti **Flatten** in reteaua de la exercitiul anterior cu max pooling ca in exemplul de mai sus.

4.In acest exercitiu vom implementa **ResNet-18**.

a) Creati o clasa care sa implementeze stack-ul de straturi descris in imaginea urmatoare. Numarul de canale in imagine este 64, in implementarea voastra faceti-l configurabil. Observati operatia de adunare efectuata intre tensorul de intrare si iesire din bloc (poarta numele de skip/residual connection).



b) Folosindu-va de bloc-ul de straturi implementat la punctul anterior, implementati arhitectura din imaginea urmatoare. In cazul nostru imaginile nu sunt de rezolutie (100, 100), de aceea setam kernel size in primul strat la 3, in loc de 7 cum este in imagine.



1. Implementati propria functie echivalenta cu cross entropia.

```
def cross_entropy(prediction, target)
```

1. Modificati reteaua de la exercitiul 2 astfel incat sa renuntati la straturile fully connected de la final si sa le inlocuiti cu straturi de pooling si straturi convolutionale.
2. Creati o vizualizare pentru filtrele din primul si ultimul strat convolutional (pentru primele 8 canale). Folositi metoda parameters() a modelului pentru a accesa straturile.