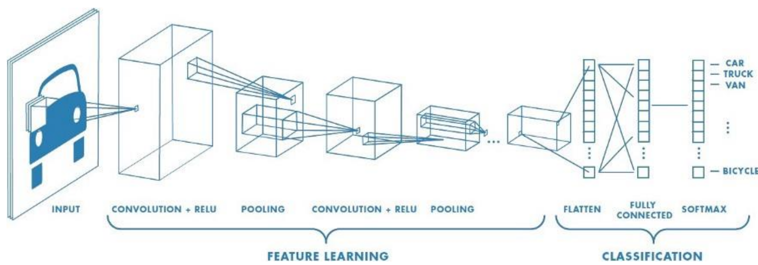
Convolutional Neural Network (CNN)

Reference:

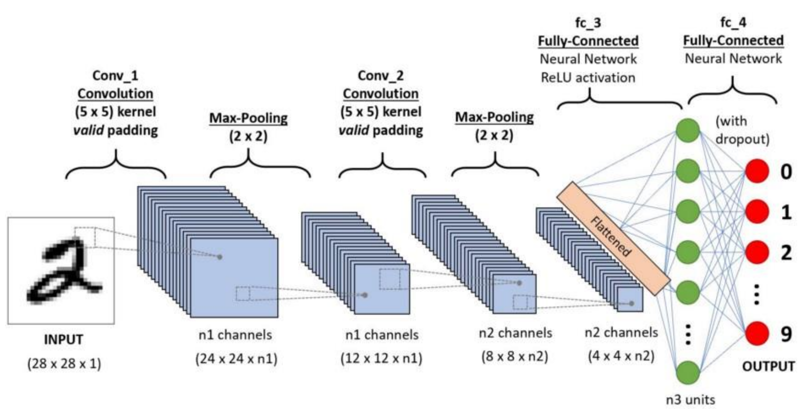
A comprehensive guide to CNN, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Yangfeng Ji, Machine learning, lecture notes

1 Overview



- CNN works well for image classification (benefits from feature learning algorithms)
- This architecture repeats the two components twice to learn features in images before connecting with a fully-connected classification task.
    1) convolutional layer
    2) subsampling (pooling) layer
- The main steps of CNN
    1) Rescale data into suitable range $[0,1]$
    2) Feature learning (twice): convolution+ReLu+pool
    3) Flat the features into 1-dimension after step 2
    4) Fully-connected Neural Network
    5) **Notice**: pay attention to the dimension of data in the process



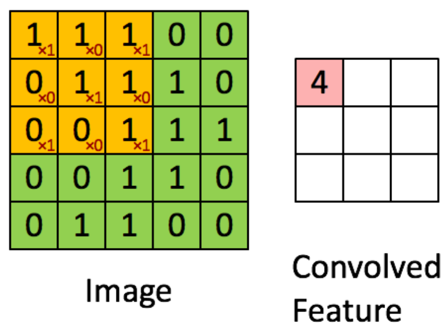
2 Why ConvNets over Feed-Forward Neural Nets?

- **Feed-Forward Neural Nets**: flat the image and feed it into multi-level perception (MLP). However, it will lose relative information.
- **ConvNet**: For complex images with pixel dependencies throughout, it can successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters (Convolution+ReLu+Pooling). The architecture performs a better fitting to the image dataset by reusing weights and reducing the number of parameters.

3 Input Image

- **Images**: can be grayscale (1 channel), RGB (3 channel). Furthermore, the images can be complex.
- **Function of ConvNet**: reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

4 Convolution Layer — The Kernel



- The **input image**: has dimension $n_1 \times m \times m$, where n_1 is the number of channels, $m \times m$ is the size of the image for a single channel.
- **The kernel**: should also have the same number of channels, i.e., $n_1 \times k \times k$.
- **The output**: the dimension is affected by the input image, kernel, and Stride_length.

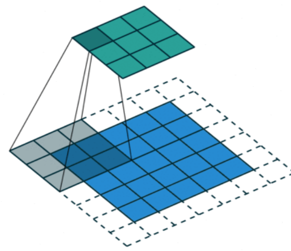
In the above demonstration, the green section resembles our **5x5x1 input image, I.** The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K,** represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

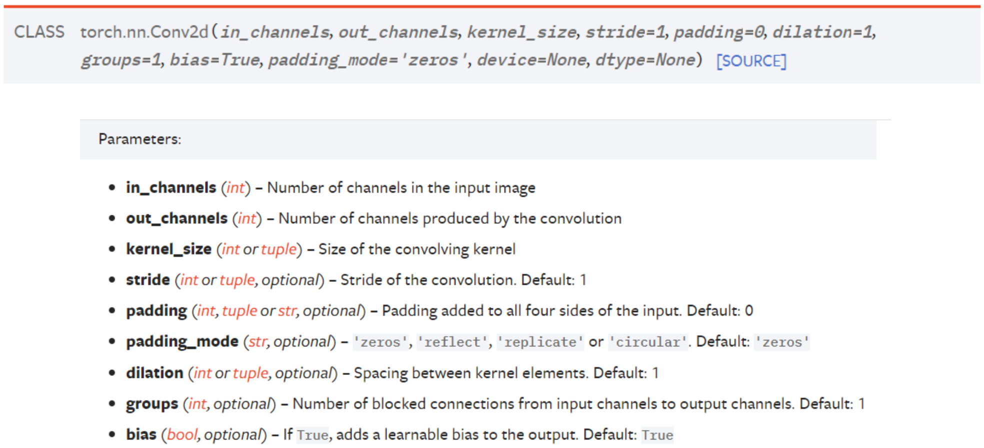
```
Kernel/Filter, K =

1  0  1
0  1  0
1  0  1
```

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.

- **Kernel function in Pytorch**: (the following kernel with padding=1)





CLASS torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*) [SOURCE]

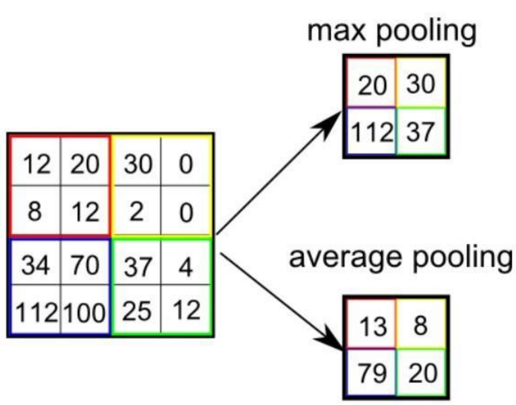
Parameters:

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int, tuple or str, optional*) – Padding added to all four sides of the input. Default: 0
- **padding_mode** (*str, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

5 Pooling Layer

- **Functions of Pooling layer**: i) Reduce the spatial size of the Convolved Feature. ii) extracting dominant features.
- **Pool types**: 1) Average pooling; 2) Max pooling. In particular, Max Pooling also performs as a Noise Suppressant. we can say that Max Pooling performs a lot better than Average Pooling.

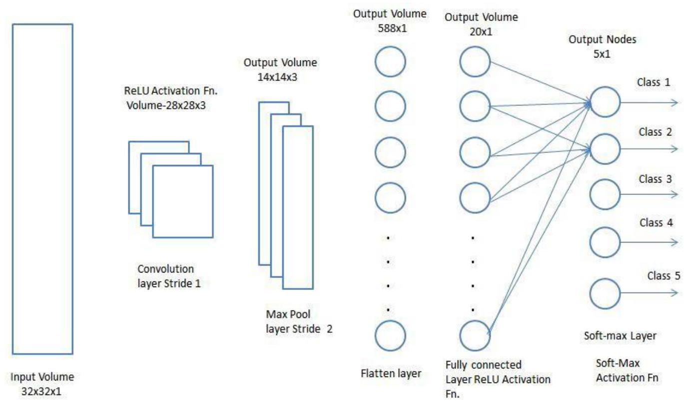


6 A Complete ConvNet layer

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network.

After all complete convolution Nets, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

7 Classification — Fully Connected Layer (FC Layer)



Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer.

Then the fully-connected MLP is trained by feeding the flattened output to a feed-forward neural network and backpropagation is applied to every iteration of training.

Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

8 Example-CNN for the Fashion Minist dataset

- Input data: $1 \times 28 \times 28$. The batch size $S = 20$
- The CNN network

```
class Net(nn.Module):   # Set convolutional network
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(20, 32, 5)
        self.fc1 = nn.Linear(32 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)


    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

input channel output channel

Conv2d (a, b, c) kernel size

$1\times28\times28 \rightarrow (1, 20, 5\times5)$

$\rightarrow 20\times24\times24 \xrightarrow{\text{maxpool }(2,2)} 20\times12\times12$

$\rightarrow 20\times12\times12 \xrightarrow{\text{kernel}} (20, 32, 5\times5)$

$\rightarrow 32\times8\times8 \xrightarrow{\text{maxpool}(2,2)} 32\times4\times4$

$\rightarrow$ flatten

- **Cost function and optimization method**: CrossEntropyLoss and SGD
  https://pytorch.org/docs/stable/optim.html

The torch.optim function: To use `torch.optim` you have to construct an optimizer object, that will hold the current state and will update the parameters based on the computed gradients.

```
'''
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)


# ######################################################################
```

```
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

- The main optimization steps of one round includes:
  a) Clean the gradient,
  b) generate the output of a model by a given input,
  c) Compute the loss
  d) Backpropagation the gradient
  e) update the model by one optimization step

- The classification accuracy of CNN: 90%

```
GroundTruth:  Ankle boot Pullover Trouser Trouser
Predicted:  Ankle boot Pullover Trouser Trouser
Accuracy of the network on the 10000 test images: 90 %
Accuracy for class: T-shirt/top is 85.3 %
Accuracy for class: Trouser is 97.9 %
Accuracy for class: Pullover is 88.6 %
Accuracy for class: Dress is 90.9 %
Accuracy for class: Coat  is 90.6 %
Accuracy for class: Sandal is 97.7 %
Accuracy for class: Shirt is 68.2 %
Accuracy for class: Sneaker is 97.7 %
Accuracy for class: Bag   is 98.3 %
Accuracy for class: Ankle boot is 94.2 %
```