Hands-on Introduction to Deep Learning with PyTorch

Introduction to Convolutional Neural Networks

Rocco Meli and Rafael Sarmiento
ETHZürich / CSCS

Lugano, February 28th - March 1st 2024

ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Computer Vision

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

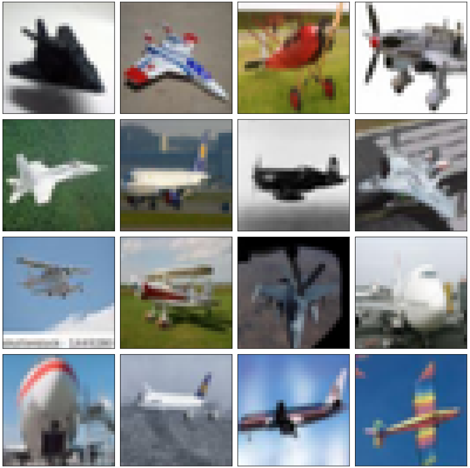
Wikipedia - Computer Vision

Computer Vision Applications

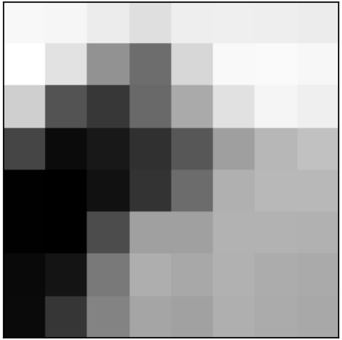
- Image classification
- Self-driving cars/autonomous vehicles
- Medicine and healthcare
- Generative art
- …

Challenges in Computer Vision

- Different scales
- Different orientations
- Different colors
- Different forms
- Different views
- Different background
- ...

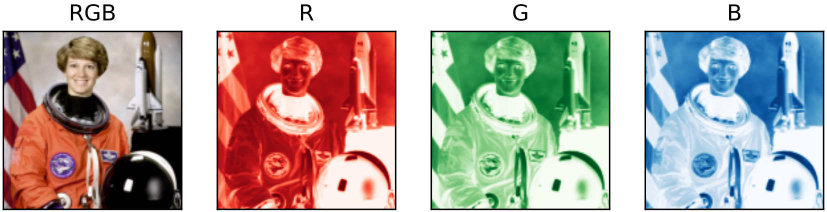


(Grayscale) Images are Matrices



203	202	195	185	196	197	196	195
209	187	128	101	179	204	205	203
173	81	60	98	146	187	201	197
71	28	37	55	84	137	155	163
20	19	32	58	100	150	156	156
20	19	76	139	138	152	152	151
26	35	110	149	144	151	147	145
27	60	118	142	139	149	146	144

(Color) Images are Tensors



PyTorch: torchvision's transforms (V1)

```python
from torchvision import transforms

transform = transforms.Compose([
    # Convert PIL image to tensor (and scale values)
    transforms.ToTensor(), # Deprecated in V2
    # Apply other transforms
    # Apply normalization
    transforms.Normalize(
      mean=[0.485, 0.456, 0.406],
      std=[0.229, 0.224, 0.225]
      ),
])
```

PyTorch: torchvision's transforms (V2)

New set of transforms, fully backward compatible with V1.

```
import torchvision.transforms.v2 as transforms

transform = transforms.Compose([
    # Convert PIL image to tensor
    transforms.ToImage(),
    # Apply other transforms
    # Convert to float32 tensor (scale to range [0,1])
    transforms.ToDtype(torch.float32, scale=True),
    # Apply normalization
    transforms.Normalize(
      mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
      ),
])
```

Fully Connected Neural Networks: How Many Parameters?

- Input: 100×100 pixels
- Linearized input: $10\,000$ neurons
- First hidden layer: 1000 neurons

How many parameters (connections) are there for the input and first hidden layer?

Fully Connected Neural Networks: How Many Parameters?

- Input: 100×100 pixels
- Linearized input: $10\,000$ neurons
- First hidden layer: 1000 neurons

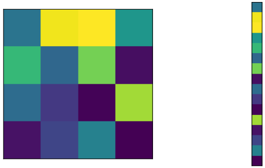
How many parameters (connections) are there for the input and first hidden layer?

$10\,000\,000$ parameters!

Fully Connected Neural Networks

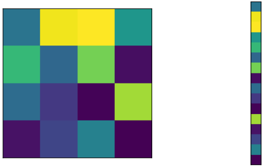
- Too many parameters
- Loss of spatial information

From 2D to 1D:



Fully Connected Neural Networks

From 2D to 1D:



- Too many parameters
- Loss of spatial information

How can the *spatial information* of 2D images be *preserved* and *exploited*?

2D Convolutions

Linear operation: multiplication of a set of weights with the input (just like fully connected NNs)

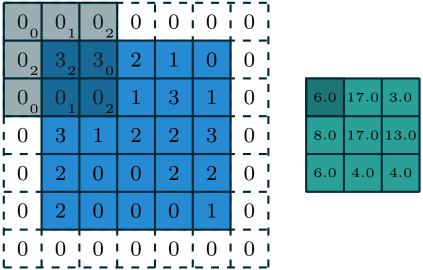
- Weights are shared across pixels
  - Reduces the number of parameters
- A pixel in the output depends only on neighboring pixels in the input
  - Retains spatial information

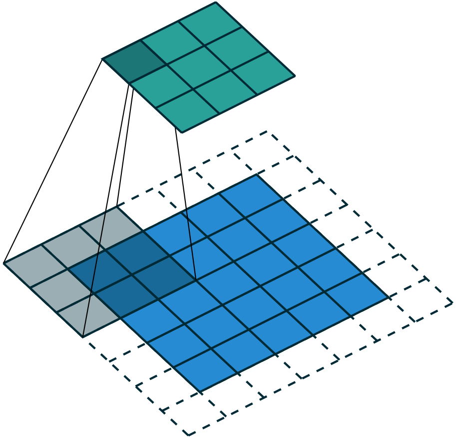
Convolution Operation

- Element-wise multiplication
- Summation

$$\begin{pmatrix} i_{11} & i_{12} \\ i_{21} & i_{22} \end{pmatrix} \star \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = i_{11}\,w_{11} + i_{12}\,w_{12} + i_{21}\,w_{21} + i_{22}\,w_{22}$$

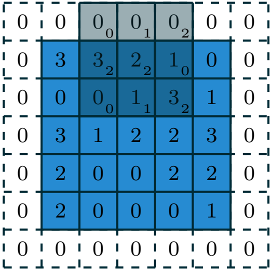
Convolutions Explained



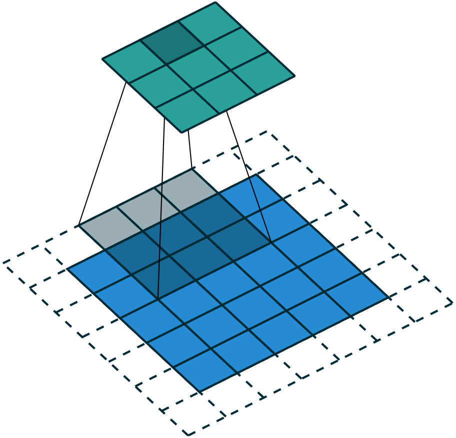


Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016)

Convolutions Explained

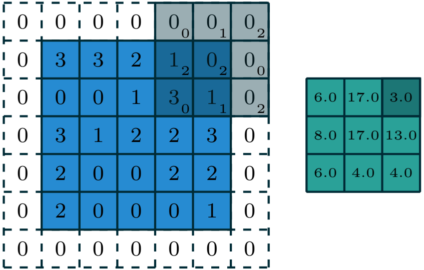


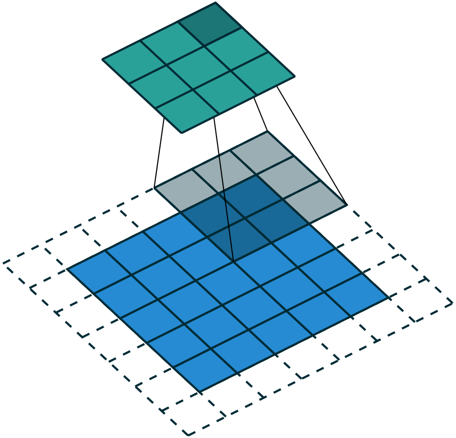
6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016)

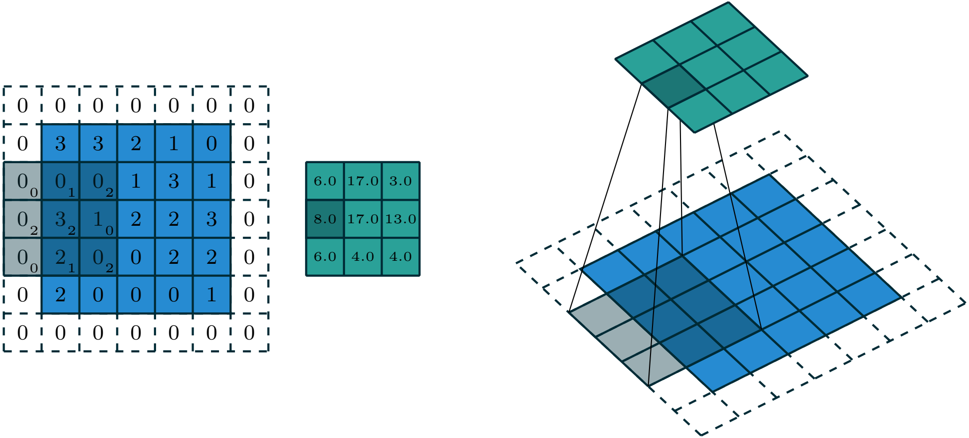
Convolutions Explained





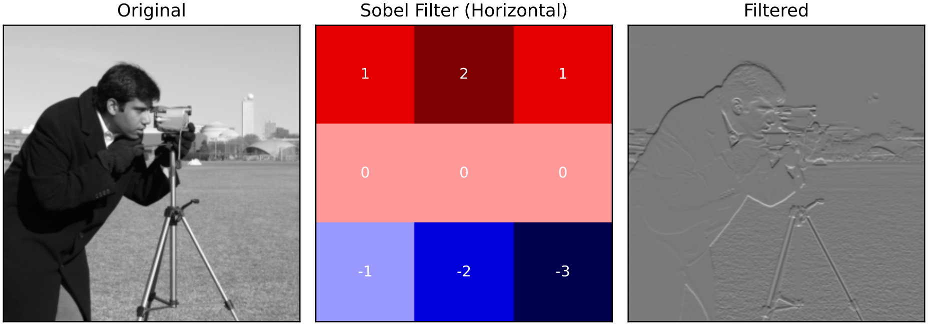
Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016)

Convolutions Explained



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016)

Convolutional Filters: Edge Detection



Convolutional Neural Networks: How Many Parameters?

- Input: 100×100 pixels, 3 color channel
- Convolutional kernel: 5×5
- Bias: true
- Output: 200 feature maps (of 100×100 pixels)

How many parameters are there for the first layer?

Convolutional Neural Networks: How Many Parameters?

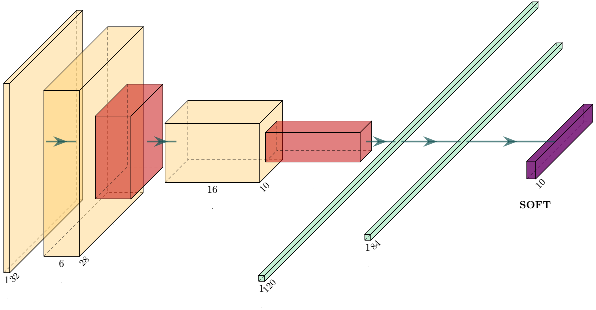
- Input: 100×100 pixels, 3 color channel
- Convolutional kernel: 5×5
- Bias: true
- Output: 200 feature maps (of 100×100 pixels)

How many parameters are there for the first layer?

$$(5 \times 5 \times 3 + 1) \times 200 = 15\,200$$

CNN Ingredients

- Convolutional layers
- Pooling layers
- Activation functions
- Normalization layers
  - Batch normalization
  - Local response normalization
- Fully connected layers



https://github.com/HarisIqbal88/PlotNeuralNet

PyTorch: 2D Convolution

$$O(N_i, C_{O_j}) = b(C_{O_j}) + \sum_{k=0}^{C_I - 1} W(C_{O_j}, k) \star I(N_i, k)$$

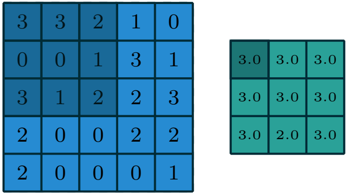
- Input size: (N, C_I, H, W)
- Output size: (N, C_O, H, W)
- b: bias
- N: batch size

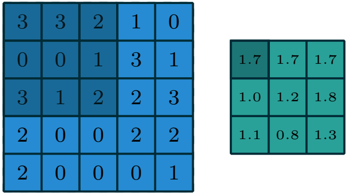
```
torch.nn.Conv2d(
  in_channels, out_channels,
  kernel_size,
  stride=1, padding=0, dilation=1,
  bias=True, padding_mode='zeros'
)

# padding="same"
```

PyTorch: Pooling

```
torch.nn.MaxPool2d(
  kernel_size, stride=None,
  padding=0, dilation=1
)
```

```
torch.nn.AvgPool2d(
  kernel_size, stride=None,
  padding=0
)
```



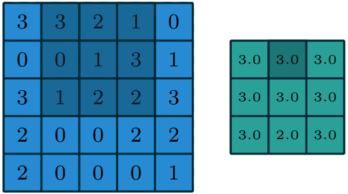


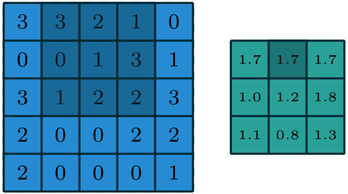
Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

PyTorch: Pooling

```
torch.nn.MaxPool2d(
  kernel_size, stride=None,
  padding=0, dilation=1
)
```

```
torch.nn.AvgPool2d(
  kernel_size, stride=None,
  padding=0
)
```



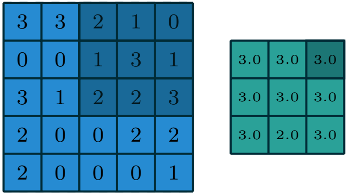


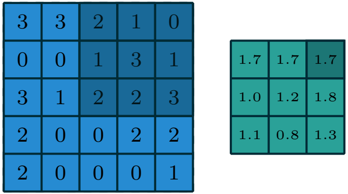
Dumoulin, Vincent, and Francesco Visin. ”A guide to convolution arithmetic for deep learning.” arXiv preprint arXiv:1603.07285 (2016).

PyTorch: Pooling

```
torch.nn.MaxPool2d(
  kernel_size, stride=None,
  padding=0, dilation=1
)
```

```
torch.nn.AvgPool2d(
  kernel_size, stride=None,
  padding=0
)
```



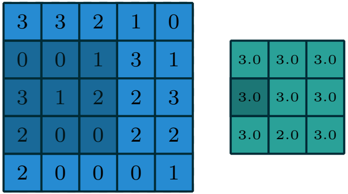


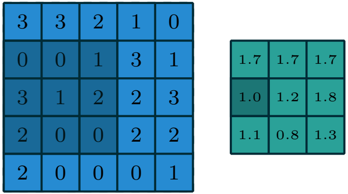
Dumoulin, Vincent, and Francesco Visin. ”A guide to convolution arithmetic for deep learning.” arXiv preprint arXiv:1603.07285 (2016).

PyTorch: Pooling

```
torch.nn.MaxPool2d(
  kernel_size, stride=None,
  padding=0, dilation=1
)
```

```
torch.nn.AvgPool2d(
  kernel_size, stride=None,
  padding=0
)
```

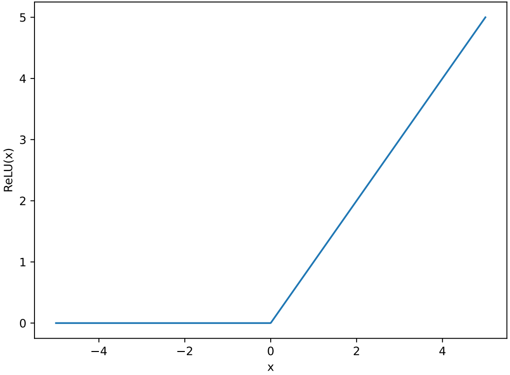




Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

PyTorch: ReLU activation function

$$\text{ReLU}(x) = \max(0, x)$$




```python
torch.nn.ReLU(inplace=False)

# Functional API
torch.nn.functional.relu(
    input,
    inplace=False
)
```

Simple CNN in PyTorch

```
nn.Sequential([
    nn.Conv2D(3, 16, 3),
    nn.MaxPool2d(2),
    nn.ReLU(),
    nn.Conv2D(16, 32, 3),
    nn.MaxPool2d(2),
    nn.ReLU(),
    nn.Flatten(),
    nn.Linear(256, 10)
])
```

PyTorch: `nn.Flatten` and `view`

`nn.Flatten()` is useful in conjunction with `nn.Sequential`, while `torch.Tensor.view()` is more general (and often used directly in `forward()`)

```
input = torch.randn(32, 1, 5, 5)
m = nn.Flatten(
  start_dim=1,
  end_dim=-1
)
output = m(input)
output.size()
# torch.Size([32, 25])
```

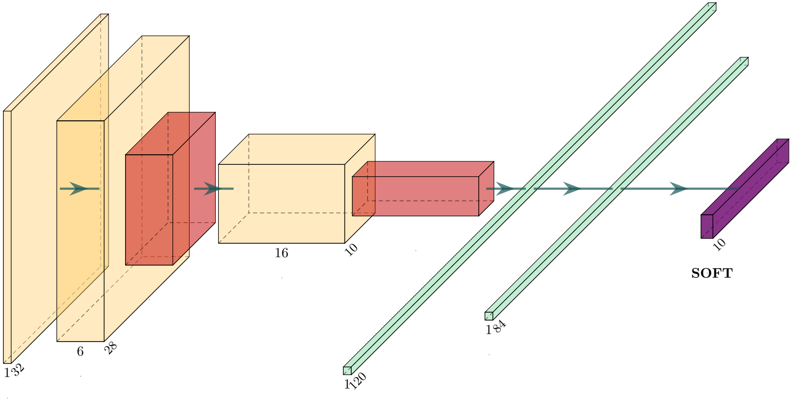
```
input = torch.randn(32, 1, 5, 5)
output = input.view(-1, 25)
output.size()
# torch.Size([32, 25])
```

CNN Architectures: LeNet 5

- Classification of handwritten digits (MNIST)
- 32×32 input layer (grayscale images)
- Two convolutional layers, two sub-sampling layers, two fully connected layers
- 10 outputs (`softmax`)

LeCun, Yann, et al. "Comparison of learning algorithms for handwritten digit recognition." International conference on artificial neural networks. Vol. 60. No. 1. 1995.

CNN Architectures: LeNet 5



https://github.com/HarisIqbal88/PlotNeuralNet

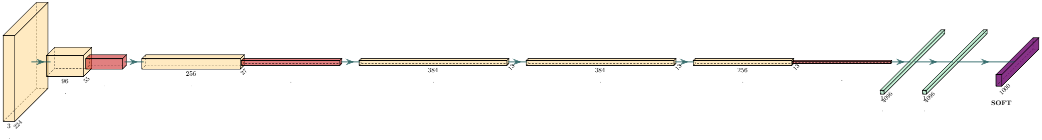
CNN Architectures: AlexNet

- Classification of color images (1000 classes)
- $256 \times 256 \times 3$ input layer
- Five convolutional layers, three fully-connected layers
- ReLU activation functions
- Local response normalization
  - Not needed (ReLU do not saturate), but helps generalization
- 1000 outputs (`softmax`)

Winner of the ImageNet competition in 2012

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. ”Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems 25 (2012).

CNN Architectures: AlexNet



https://github.com/HarisIqbal88/PlotNeuralNet

PyTorch: Local Response Normalisation

$$b_c = a_c \left(k + \frac{\alpha}{n} \sum_{c'=\max(0, c-n/2)}^{\min(N-1, c+n/2)} a_{c'}^2 \right)^{-\beta}$$

- n: number of "adjacent" feature maps
- N: total number of feature maps

```
torch.nn.LocalResponseNorm(
  size,
  alpha=0.0001,
  beta=0.75,
  k=1.0
)
```

PyTorch: Local Response Normalisation

- Competitive activation
  - Strongly activated neurons inhibit other neurons
  - Neurons are located at the same position in the feature map

Encourages feature maps to learn a wide and different range of features, improving generalization.

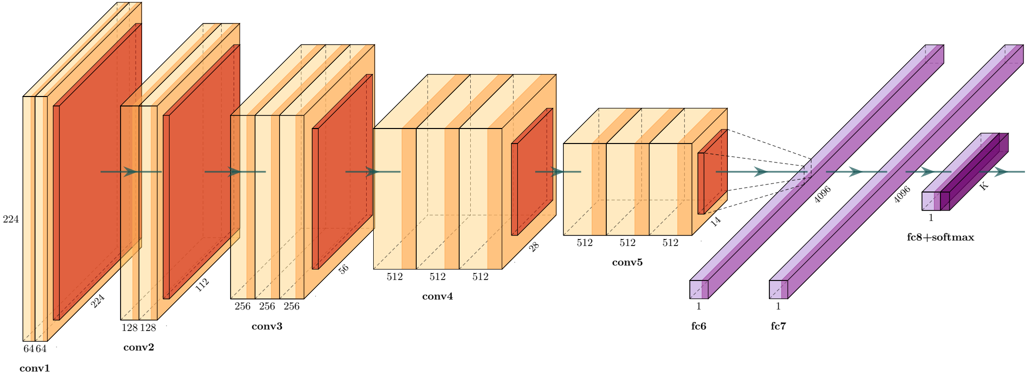
CNN Architectures: VGGNet

- Variants: VGG-13, VGG-16, VGG-19
  - Different number of layers
- $224 \times 224 \times 3$ input layer
- Stacks of convolutional layers, with small kernels (3×3)

Runner-up of the ImageNet competition in 2014

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

CNN Architectures: VGG16



https://github.com/HarisIqbal88/PlotNeuralNet

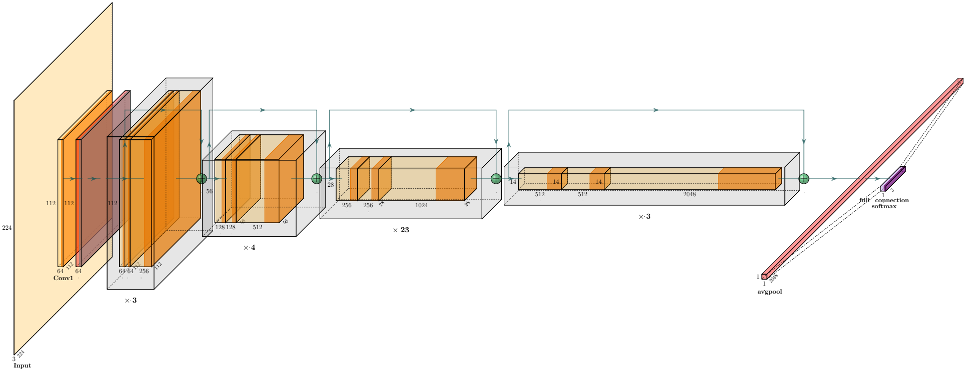
CNN Architectures: ResNet

- Variants: ResNet-34, ResNet-50, ResNet-101, ResNet-152 (number of layers)
  - Different number of layers
- Deeper than VGG nets
- $224 \times 224 \times 3$ input layer
- Batch normalization
  - Higher learning rate
  - Less sensitive to initialization
  - Acts as regularizer

Winner of the ImageNet competition in 2015

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

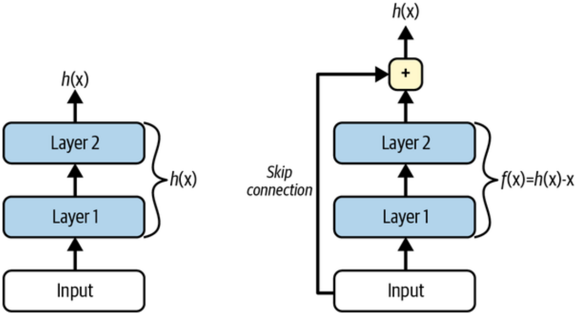
CNN Architectures: ResNet 101



https://github.com/HarisIqbal88/PlotNeuralNet

ETH zürich
CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

Skip Conncetions and Residual Learning



- Speed-up training (identity)
- Facilitate signal propagation (forward/backward)

```python
# torchvision/models/resnet.py
def forward(self, x):
  identity = x

  out = self.conv1(x)
  out = self.bn1(out)
  out = self.relu(out)

  out = self.conv2(out)
  out = self.bn2(out)

  out += identity
  out = self.relu(out)

  return out
```

Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

Batch Normalisation

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

Training:

- Compute $\mathrm{E}[x]$ (mean) and $\mathrm{Var}[x]$ (variance) of the input over batch
- Normalize the input
- Estimate μ and σ^2 using exponential moving averages

Inference:

- Normalize input using μ and σ^2

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

PyTorch: Batch Normalisation

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

- $\mathrm{E}[x]$: mean over mini-batch
- $\mathrm{Var}[x]$: variance over mini batch
- Normalization over C
- Statistics on (N, H, W)
- β and γ are learned

```
torch.nn.BatchNorm2d
  num_features, eps=1e-05,
  momentum=0.1, affine=True,
  track_running_stats=true
)
```

Advantages of Batch Normalisation

- Greatly reduces the vanishing gradient problem
  - Possible to use saturating activation functions ($\sigma(x)$, $\tanh(x)$, ...)
- Reduces sensitivity to weight initialization
- Allows the use of larger learning rates
  - Speed-up training
- Act as regularizer

CNN Architectures' Zoo

Many more CNN architectures:

- GoogLeNet
- DenseNet
- ...

Test different architectures/variants and choose the best for your task!

Protein-Ligand Docking

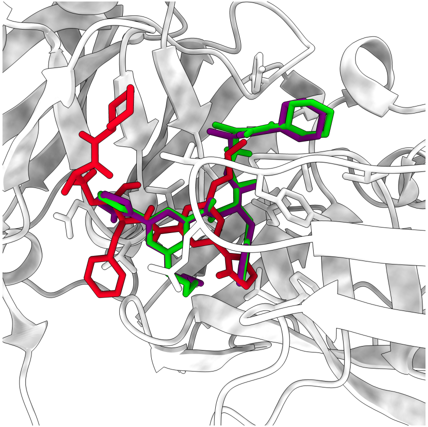


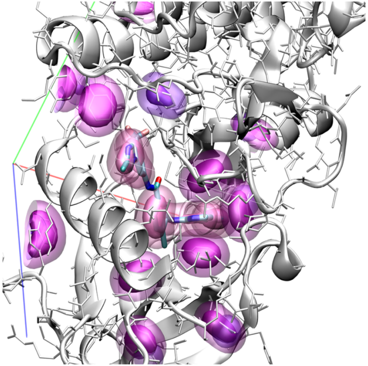
Image: R. Meli, "Deep Learning Applications in Structure-Based Drug Discovery", University of Oxford, 2022.

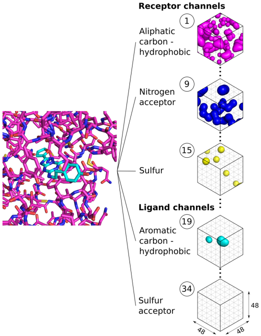
Given a small molecule (ligand) and a protein target:

- Generate plausible conformations
- Rank plausible conformations

CNNs for Volumetric Data: Protein-Ligand Docking

CNNs can be easily generalized to volumetric (3D) data, and can be applied to other domains.





Ragoza, Matthew, et al. "Protein–ligand scoring with convolutional neural networks." Journal of chemical information and modeling 57.4 (2017): 942-957 | Imrie, Fergus, et al. "Protein family-specific models using deep neural networks and transfer learning improve virtual screening and highlight the need for more data." Journal of chemical information and modeling 58.11 (2018): 2319-2330

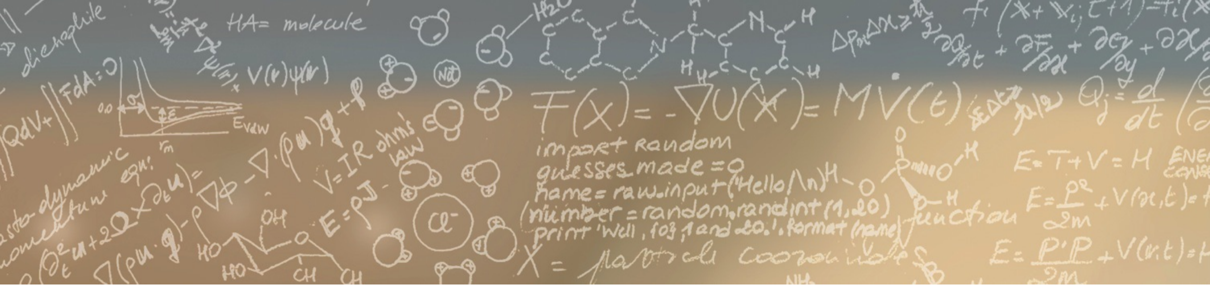
What have we missed?

- Classification and localization
  - Regression problem for bounding box (center, height, width)
- Object detection
  - Classification and localization of multiple objects
- Semantic segmentation
  - Classification of every pixel (building, bridge, car, …)
- Vision transformer (ViT)
  - Pure transformer architecture on sequence of image patches, no convolutions
- …

[lab] CNN for Image Classification on CIFAR-10

Implement and train a CNN from scratch for image classification:

- Load and inspect data
- Implement a simple CNN
- Implement training loop
- Train CNN
- Evaluate CNN performance

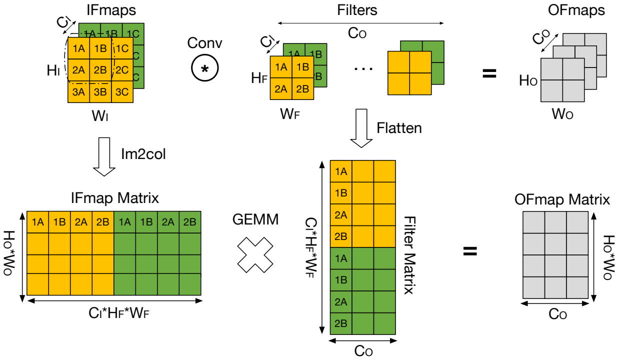


Thank you for your attention!

ETH zürich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

From Convolutions to GEMM: im2col



Zhou, Yangjie, et al. "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators." 2021 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2021.