GSAS @NIDA : Deep Learning

CNN 1

Assoc.Prof.Thitirat Siriborvornratanakul, Ph.D.

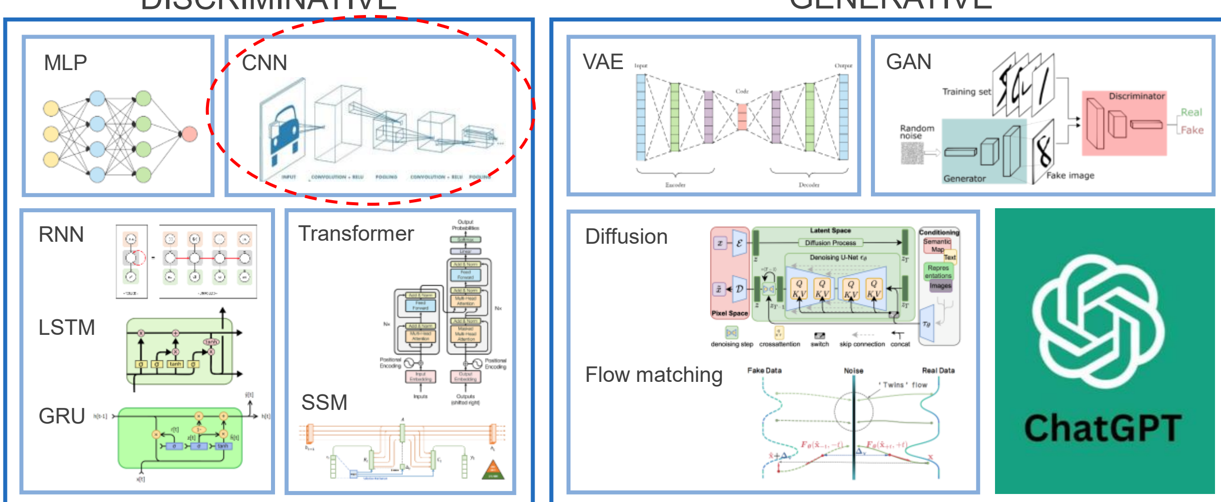
Email: thitirat@as.nida.ac.th
Website: http://as.nida.ac.th/~thitirat/

1

DL models in this course

FUNDAMENTAL
~~DISCRIMINATIVE~~

GENERATIVE



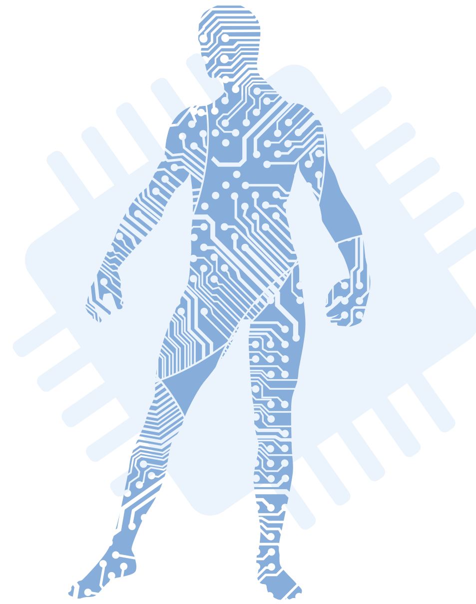
2

OUTLINE

01 Preliminary
The brief history of CNN and the ImageNet moment in computer vision

02 Intro to CNNs
Learn components that make up a basic Convolutional Neural Network for image classification

03 Coding
Combine everything and implement a program with PyTorch



3

Preliminary

The brief history of CNN and the ImageNet moment in computer vision

4

THE OLD HISTORY

- **1943:** The first neural network was proposed by McCulloch and Pitts.
- **1955:** John McCarthy first coined the term Artificial Intelligence.
- **1959:** Samuel coined and popularized the term Machine Learning.

- **1982:** Birth of Hopfield Network
- **1986:** Birth of backpropagation
- **1986:** Birth of RNN and AE

- **1997:** Birth of LSTM
- **1998: Birth of CNN**

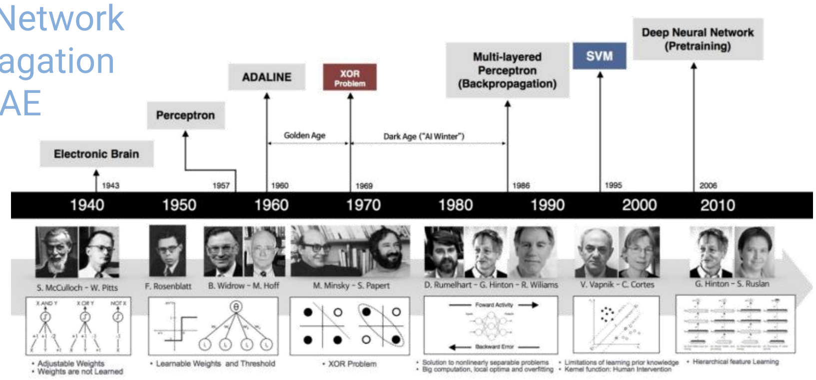


Image credit:
https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

5

THE MODERN HISTORY

- **2006:** The pretraining technique by Hinton et al.

- **2012: ImageNet evolution—AlexNet**

- **2014:** Birth of GRU, VAE, and GAN

- **2015:** Birth of diffusion model

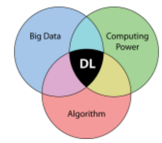
- **2017:** The DeepFake viral
- **2017:** Birth of the groundbreaking Transformer

- **2018:** NLP's ImageNet moment—BERT
- **2018:** ACM Turing Award to Deep NN

6

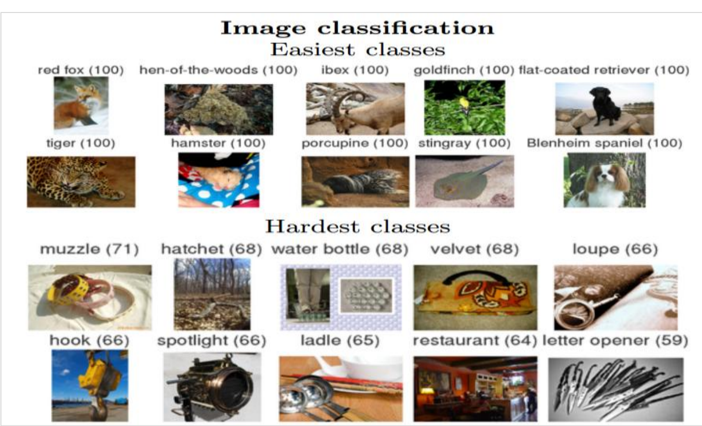
ImageNet (2010-2017)

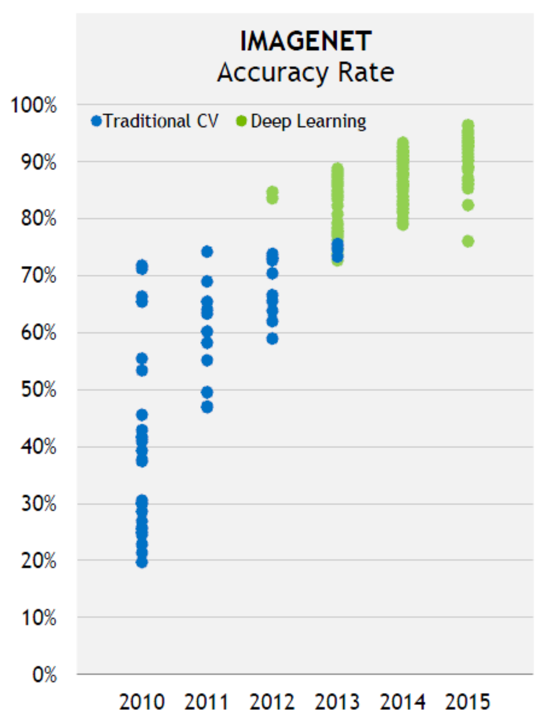
- One of the most prestigious benchmarks in computer vision. Sometimes even referred to as the Olympics of Computer Vision.
- The **ImageNet** dataset was first presented in CVPR 2009.
- The annual **I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (**ILSVRC**):
  - Started in 2010, it is designed to follow on from a similar project called PASCAL VOC which ran from 2005 until 2012.
  - 1,000 different categories of objects where contestants have to scour a database of over 1 million images to find every instance of each object.
  - Announced on Jul 26, 2017: "We are passing the baton to Kaggle (owned by Google). From now on, all three challenges (LOC-CLS, DET, VID) will be hosted on Kaggle!"



J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," In Procs. of IEEE CVPR, 2009 https://ieeexplore.ieee.org/document/5206848
O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," In IJCV, 115, 2015 https://arxiv.org/abs/1409.0575

7



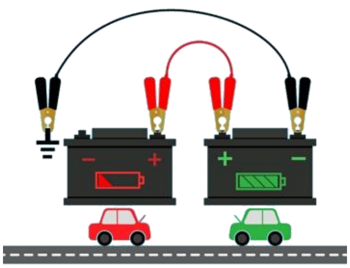


8

ImageNet (2010-2017)

WORLD'S STANDARD IMAGE BENCHMARK | DATASET USED BY MOST FOUNDATION MODELS

- In 2011 and 2012, **ImageNet** dataset became a benchmark for how well image classification algorithms fared against the most complex visual dataset assembled at the time.
- Researchers began to notice something more going on than just a competition—their algorithms worked better when they trained using the **ImageNet** dataset.



> "The nice surprise was that people who trained their models on ImageNet could use them to jumpstart models for other recognition tasks. You'd start with the ImageNet model and then you'd fine-tune it for another task," said Berg. "That was a breakthrough both for neural nets and just for recognition in general."

https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/

9

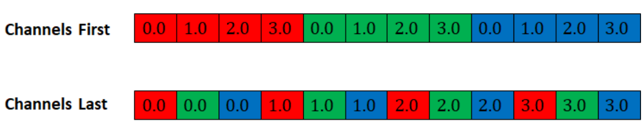
Intro to CNNs

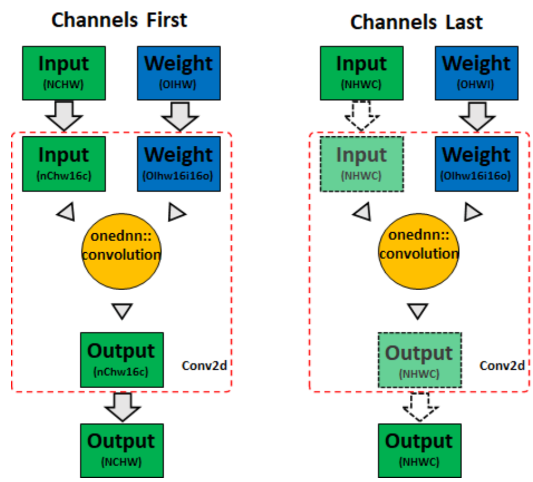
Learn components that make up a basic Convolutional Neural Network for image classification

10

Channel first vs. Channel last

- A batch of five 600x400 RGB images
  - Shape: (5, 400, 600, **3**)
  - Shape: (5, **3**, 400, 600)



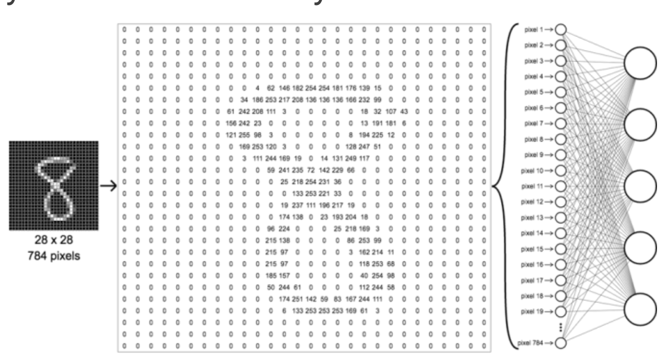


24AUG2022: https://pytorch.org/blog/accelerating-pytorch-vision-models-with-channels-last-on-cpu/

11

MLP for image data

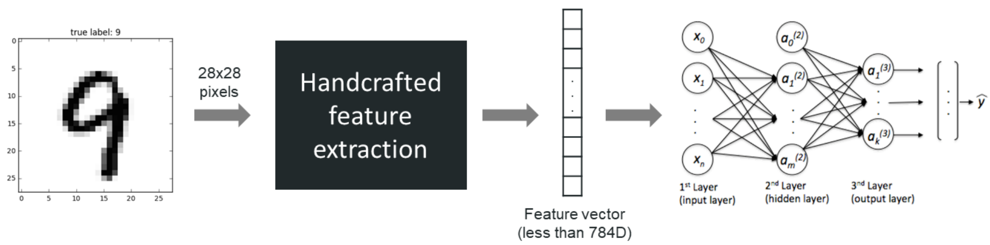
- If one input node (neuron) corresponds to one input pixel:
  - The amount of weight parameters rapidly becomes unmanageable for large images. Also, no weights are shared among pixels.
  - MLP uses fully-connected layers that form a very dense web—resulting in redundancy, inefficiency, and overfitting.
  - MLP is not translation invariant as it reacts differently to an input image and its shifted versions.



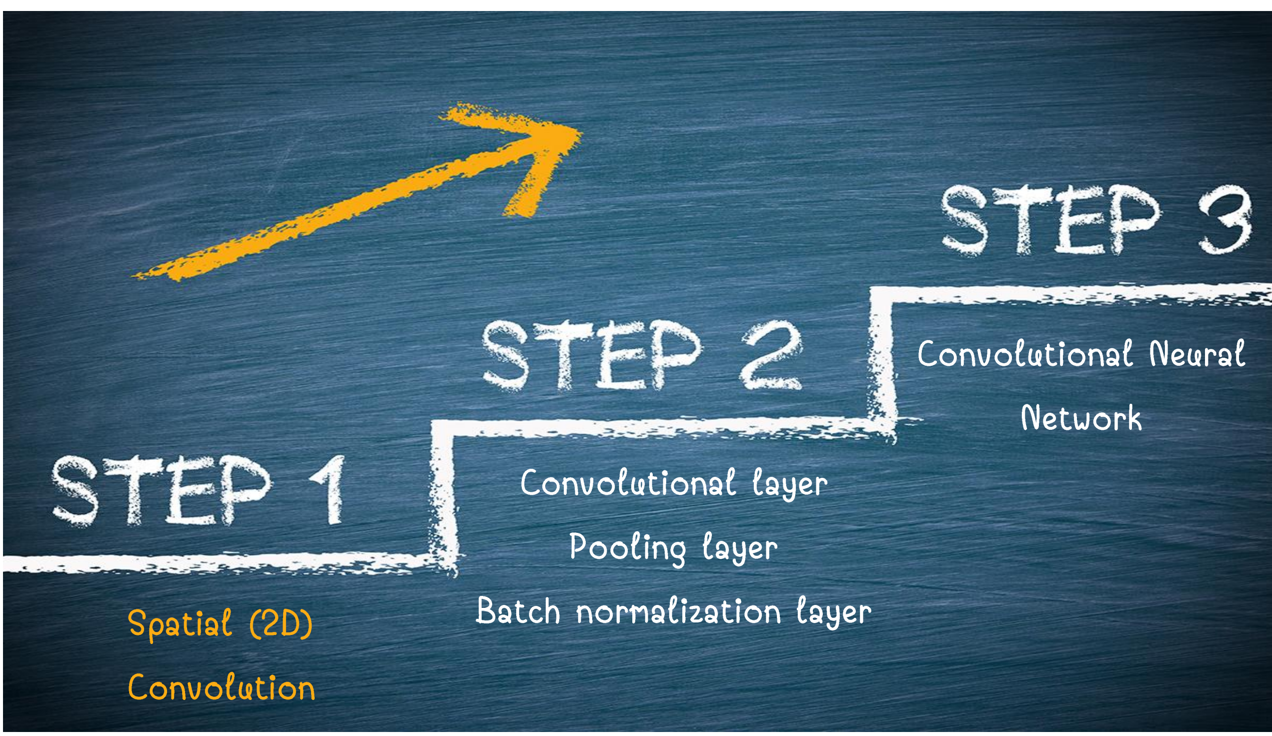
12

MLP for image data

- If an input image is first compressed using any handcrafted feature engineering technique:
  - It becomes a two-step network that is more difficult to debug and cannot be trained in an end-to-end manner using backpropagation.
  - Handcrafted feature engineering was a long-standing bottleneck in computer vision for almost half a century.



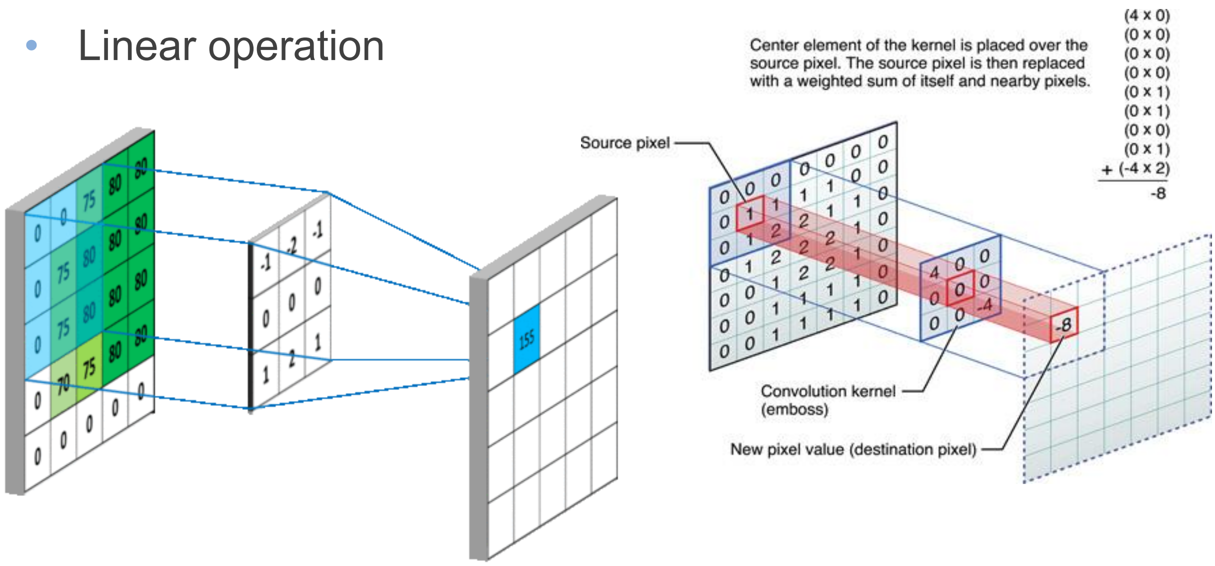
13



14

Spatial (2D) Convolution

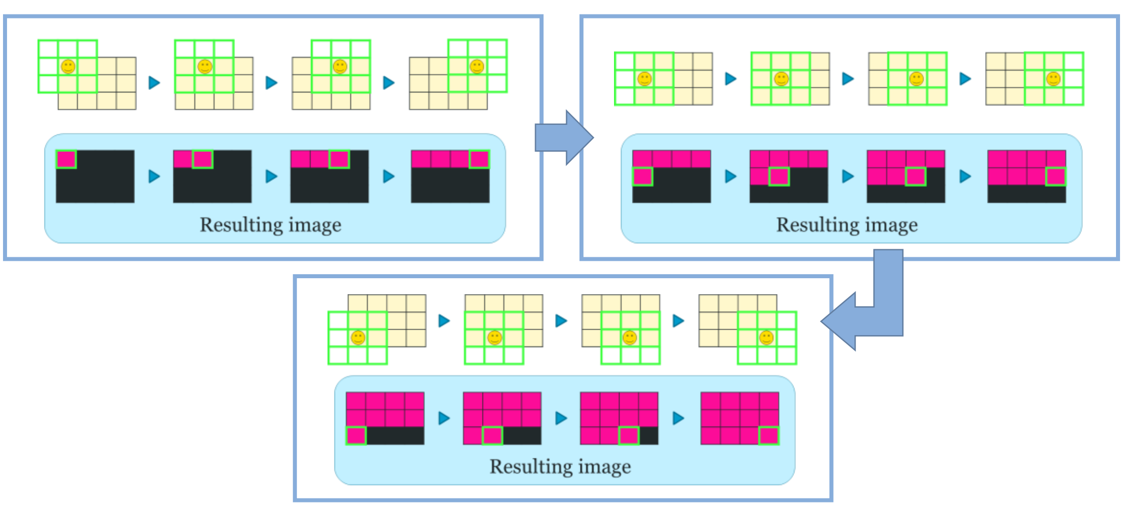
- Linear operation



15

Spatial (2D) Convolution

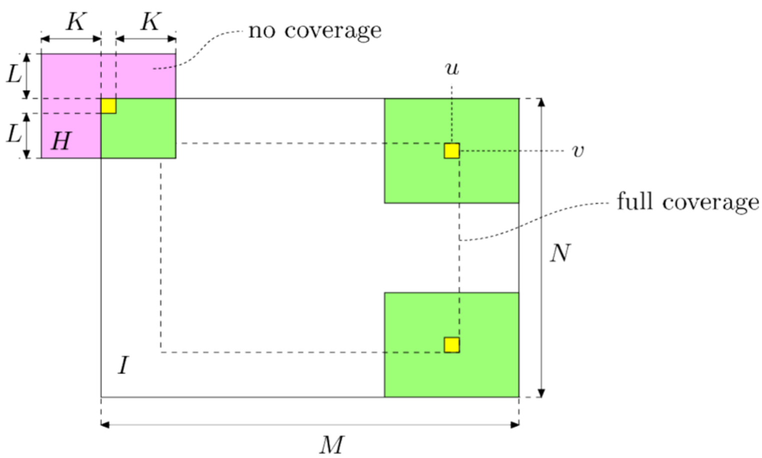
- Kernel sliding (kernel's size, stride)

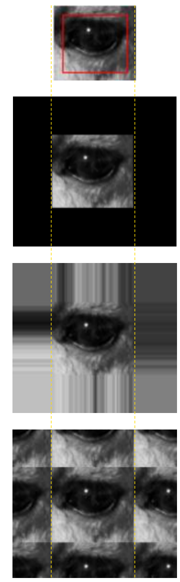


16

Spatial (2D) Convolution

- Padding strategy: pad it or crop it





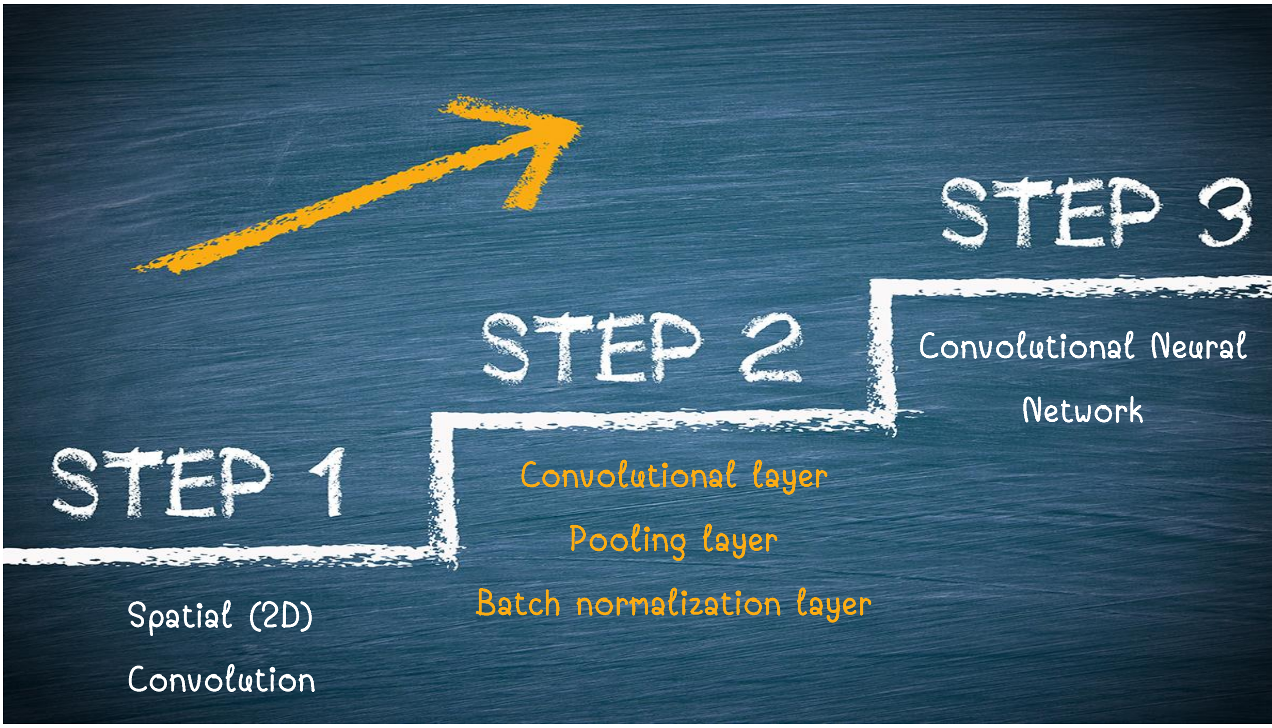
17

How many filters?
What filters?
The sequence of filters?
Kernel's size for each?
Padding or not?
Stride values for each?

Paradox of Choices

Infinite combinations of filters

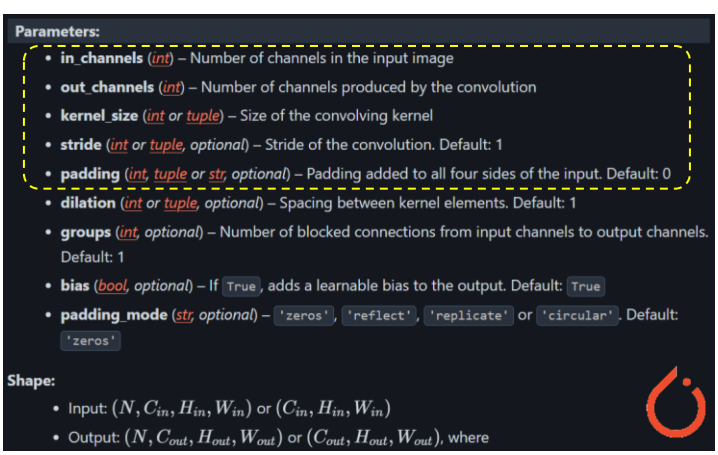
18



19

CONV2D LAYER

- For a single layer of `torch.nn.Conv2d()` :
  - Trainable parameters:
    - Weights (numeric values in each kernel) and biases

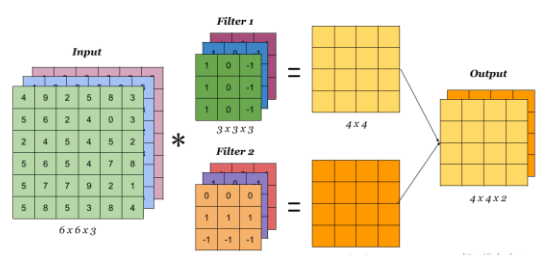


Parameters:

- **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
- **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`
- **padding_mode** (*str, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`

Shape:

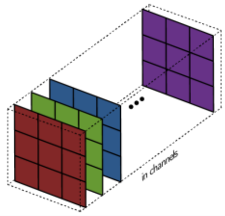
- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where



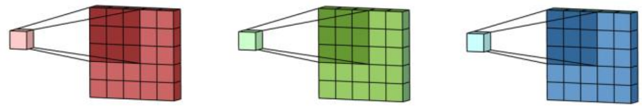

```
torch.nn.Conv2d(
     in_channels=3,
     out_channels=2,
     kernel_size=3,
     padding=0,
     stride=1
)
```

20

1. For each filter, prepare the 2D kernel(s). The number of 2D kernels is automatically assigned by deep learning frameworks.

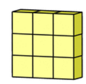


2. Separately apply each kernel in the filter to its corresponding feature map.



3. Aggregate (Sum) results from all kernels in the filter.

4. Finally, add the bias.



CONV2D LAYER

The multi-channel version

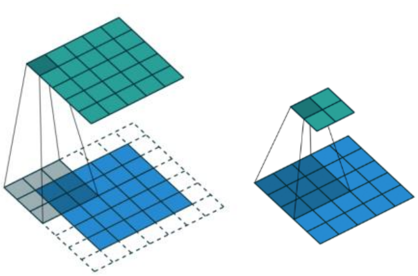
Animation credit: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

21

• **dilation** (*int* or *tuple*, *optional*) – Spacing between kernel elements. Default: 1

Dilated Convolution

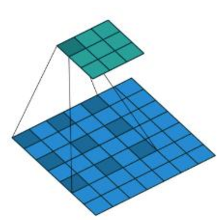
- Expands the receptive field w/o increasing the number of parameters by inserting gaps between filter pixels.
- Captures long-range dependencies efficiently while preserving spatial resolution.
- Useful in segmentation and sequence modeling, as it helps recognize patterns over a wider context w/o losing details.



kernel_size=(3,3)
stride=1
padding='same'

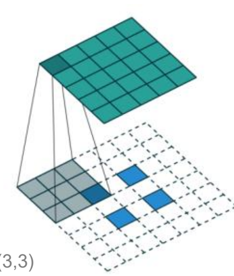
kernel_size=(3,3)
stride=2
padding='valid'

Basic Convolution



kernel_size=(3,3)
dilation_rate=2
stride=1
padding='valid'

Dilated Convolution



kernel_size=(3,3)
stride=2
padding='valid'

(Deconvolution, Fractionally Strided Convolution)

Transposed Convolution

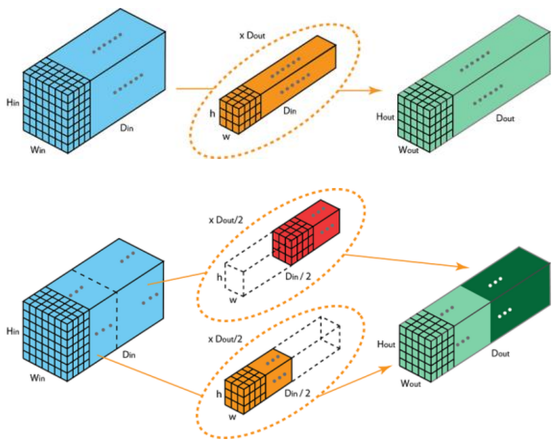
Image(s) credit: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

22

• groups (*int*, *optional*) – Number of blocked connections from input channels to output channels. Default: 1

Grouped Convolution

- A standard convolution connects all input channels to all output channels. **Grouped Convolution** breaks these wires, splitting the layer into `groups` independent parallel paths.




```
torch.nn.Conv2d(
    in_channels=Din,
    out_channels=Dout,
    kernel_size=3,
    padding=0,
    stride=1,
    groups=1
)
```

In PyTorch, `in_channels` and `out_channels` must both be divisible by `groups`.

```
torch.nn.Conv2d(
    in_channels=Din,
    out_channels=Dout,
    kernel_size=3,
    padding=0,
    stride=1,
    groups=2
)
```

16FEB2020: https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

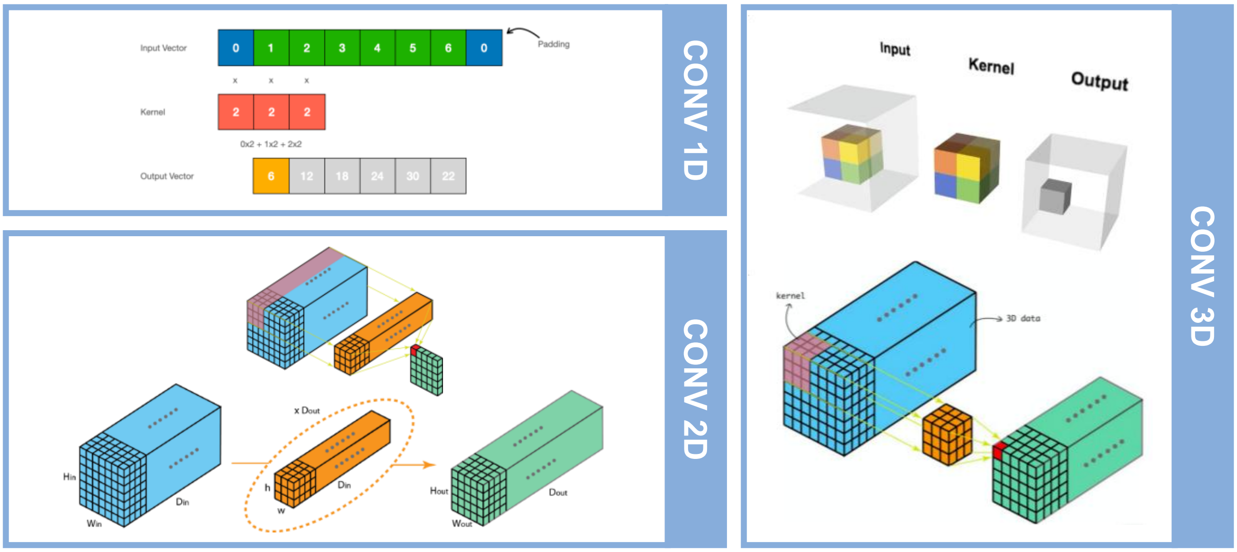
23

Grouped Convolution

- **Parameter reduction:** the number of parameters in kernels is decreased by a linear factor of `groups` because each kernel only sees a fraction (1/`groups`) of the input volume.
  - AlexNet (2012) popularized **Grouped Convolution** by using it to fit model training on two GTX 580 (3GB VRAM) GPUs.
- **Computational efficiency:** FLOPs drop by a factor of `groups`. While the kernel still slides the same distance, the mathematical depth of the dot product at each pixel is `groups` times smaller.
  - Although it does not matter in Big-oh analysis, it has a high impact on hardware computation reduction.
- **Allow each group to specialize differently.**
  - In AlexNet (2012), one group focused on color and orientation, while the other focused on black-and-white textures and edges.
- **Regularization:** By restricting channel cross-talk, the model is forced to learn more robust, distinct features, which helps prevent overfitting.
- **Foundation to many models:** For ultra-efficient models like MobileNet and EfficientNet, **Depthwise Separable Convolutions** (`in_channels == groups`) is a core concept.

24

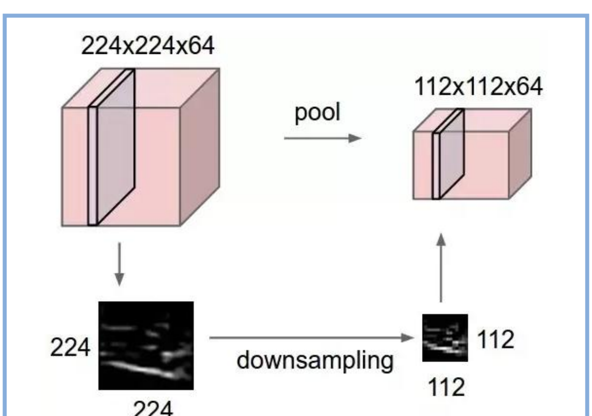
CONV1D, CONV2D, CONV3D

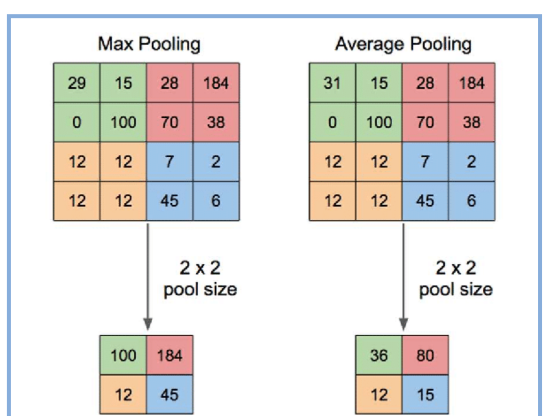


25

POOLING LAYER (2D)

- Goal of the pooling layer is to subsample (shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting).





26

POOLING LAYER (2D)

- In CNN, max pooling is more common than average pooling.
- Max pooling also helps introduce some level of invariance to small translations.
- However, max pooling is very destructive as many input values are dropped.
- In some applications, invariance is not desirable. For example, semantic segmentation requires equivariance, not invariance.

27

CODE REVIEW

Pooling layers

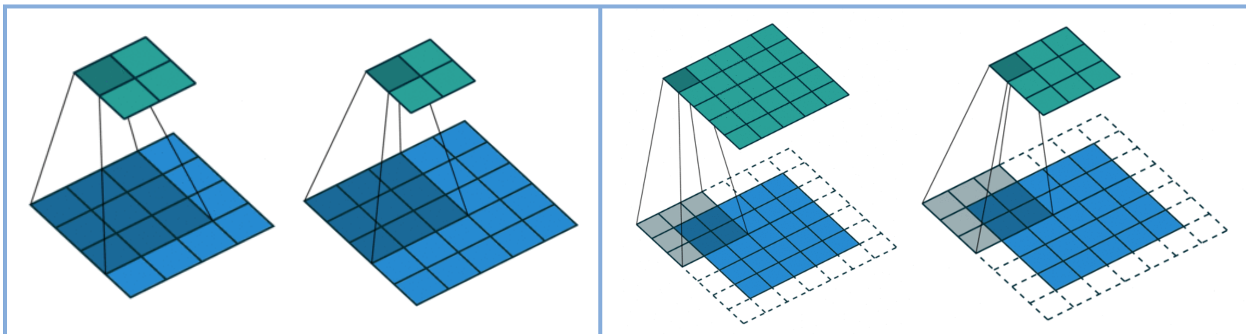
nn.MaxPool1d	Applies a 1D max pooling over an input signal composed of several input planes.
nn.MaxPool2d	Applies a 2D max pooling over an input signal composed of several input planes.
nn.MaxPool3d	Applies a 3D max pooling over an input signal composed of several input planes.
nn.MaxUnpool1d	Computes a partial inverse of MaxPool1d.
nn.MaxUnpool2d	Computes a partial inverse of MaxPool2d.
nn.MaxUnpool3d	Computes a partial inverse of MaxPool3d.
nn.AvgPool1d	Applies a 1D average pooling over an input signal composed of several input planes.
nn.AvgPool2d	Applies a 2D average pooling over an input signal composed of several input planes.
nn.AvgPool3d	Applies a 3D average pooling over an input signal composed of several input planes.
nn.FractionalMaxPool2d	Applies a 2D fractional max pooling over an input signal composed of several input planes.
nn.FractionalMaxPool3d	Applies a 3D fractional max pooling over an input signal composed of several input planes.
nn.LPPool1d	Applies a 1D power-average pooling over an input signal composed of several input planes.
nn.LPPool2d	Applies a 2D power-average pooling over an input signal composed of several input planes.
nn.LPPool3d	Applies a 3D power-average pooling over an input signal composed of several input planes.
nn.AdaptiveMaxPool1d	Applies a 1D adaptive max pooling over an input signal composed of several input planes.
nn.AdaptiveMaxPool2d	Applies a 2D adaptive max pooling over an input signal composed of several input planes.
nn.AdaptiveMaxPool3d	Applies a 3D adaptive max pooling over an input signal composed of several input planes.
nn.AdaptiveAvgPool1d	Applies a 1D adaptive average pooling over an input signal composed of several input planes.
nn.AdaptiveAvgPool2d	Applies a 2D adaptive average pooling over an input signal composed of several input planes.
nn.AdaptiveAvgPool3d	Applies a 3D adaptive average pooling over an input signal composed of several input planes.

https://docs.pytorch.org/docs/stable/nn.html#pooling-layers

28

STRIDED CONVOLUTION

- Both pooling layer and strided convolution (`strides>1`) can be used to encode or reduce the dimensionality of the data.



(No padding) left: strides=1 , right: strides=2 (Padding=1) left:strides=1 , right: strides=2

Animation credit: https://github.com/vdumoulin/conv_arithmetic

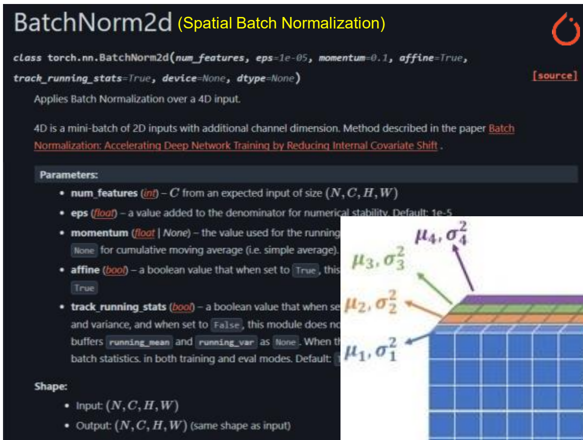
29

STRIDED CONVOLUTION

- Pooling layer vs. Strided convolutional layer:
  - Pooling is a fixed operation whereas (strided) convolution can be learned.
  - Pooling has no trainable parameter whereas (strided) convolution has weights (kernels) and biases to be trained.
  - Pooling is cheaper (faster) in terms of the amount of computation.
  - In many cases, max pooling can be replaced with strided convolution without significant change in the accuracy. "Striving for Simplicity: The All Convolutional Net," ICLR2015, https://arxiv.org/abs/1412.6806
  - Using max pooling in CNN is historical and appears in most state-of-the-art CNNs. Strided convolution is a newer concept.
  - Some researchers said that using strided convolution instead of pooling allows gradient signals to flow better during backpropagation. Nevertheless, like most practices in deep learning communities, no one can guarantee that one technique will always outperform the others.

30

BATCH NORMALIZATION



BatchNorm2d (Spatial Batch Normalization)

```
class torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None)
```
[source]

Applies Batch Normalization over a 4D input.

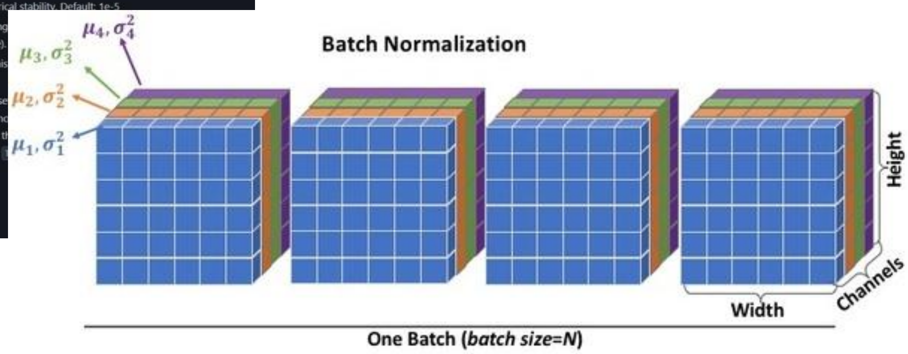
4D is a mini-batch of 2D inputs with additional channel dimension. Method described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

Parameters:

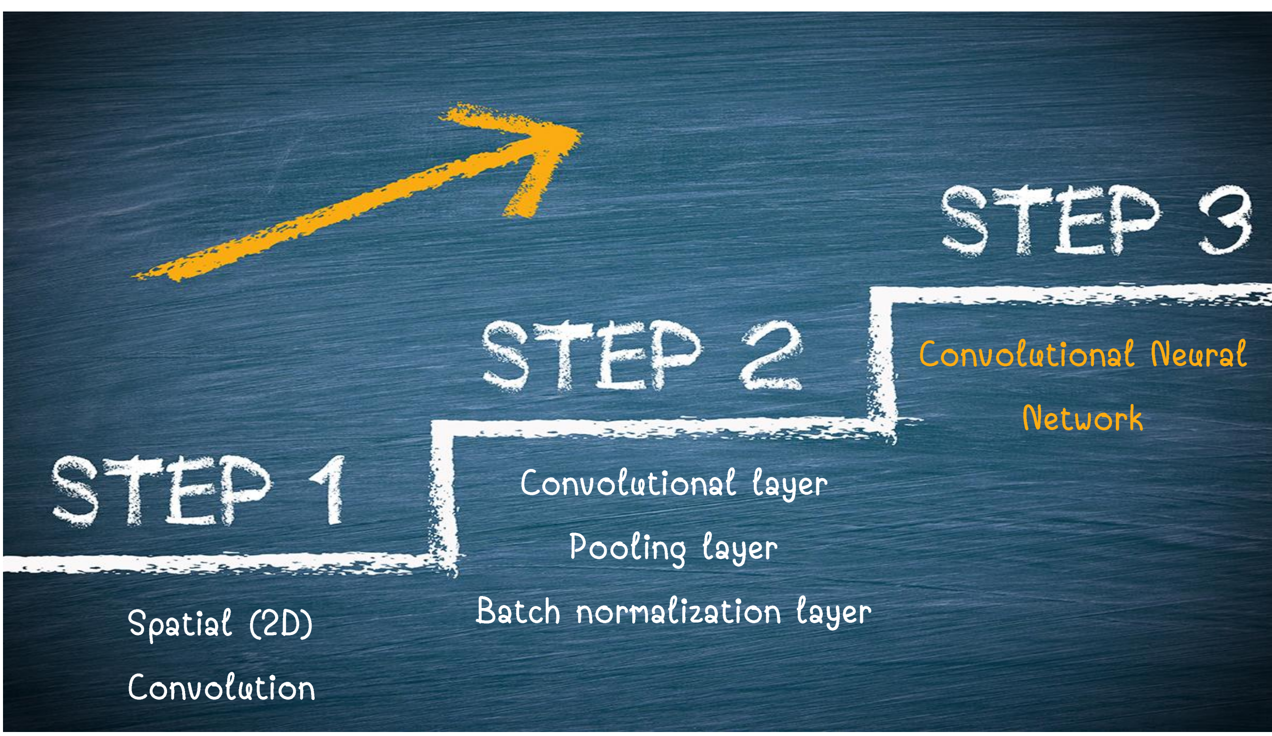
- **num_features** (int) – C from an expected input of size (N, C, H, W)
- **eps** (float) – a value added to the denominator for numerical stability. Default: 1e-5
- **momentum** (float | None) – the value used for the running
  None for cumulative moving average (i.e. simple average).
- **affine** (bool) – a boolean value that when set to True, this
  True
- **track_running_stats** (bool) – a boolean value that when se
  and variance, and when set to False, this module does no
  buffers running_mean and running_var as None. When th
  batch statistics. in both training and eval modes. Default:

Shape:

- Input: (N, C, H, W)
- Output: (N, C, H, W) (same shape as input)

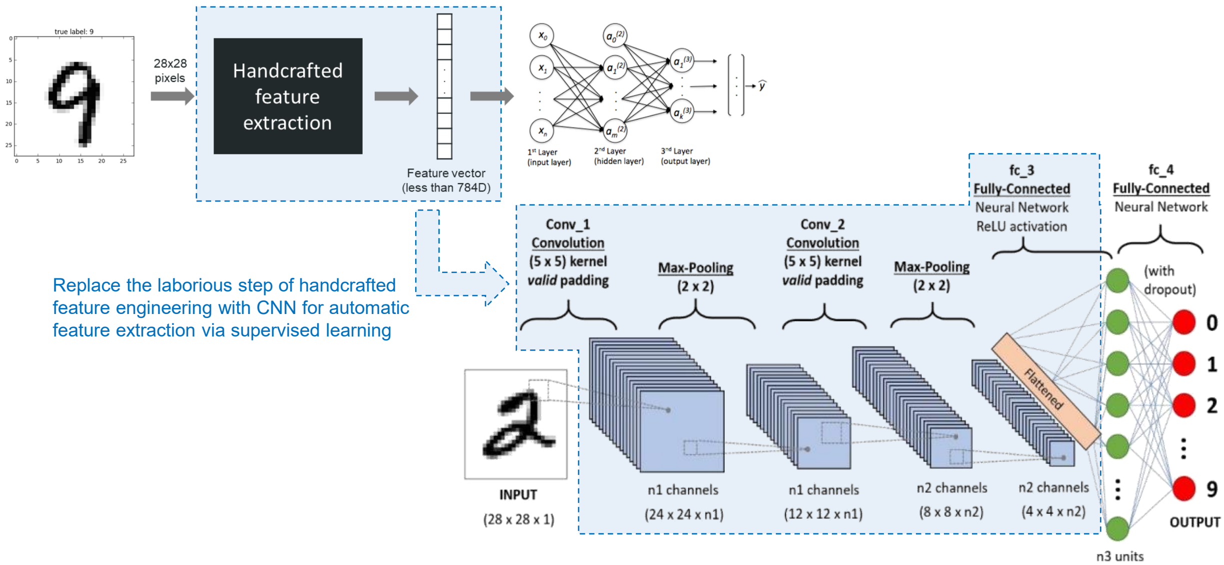


31



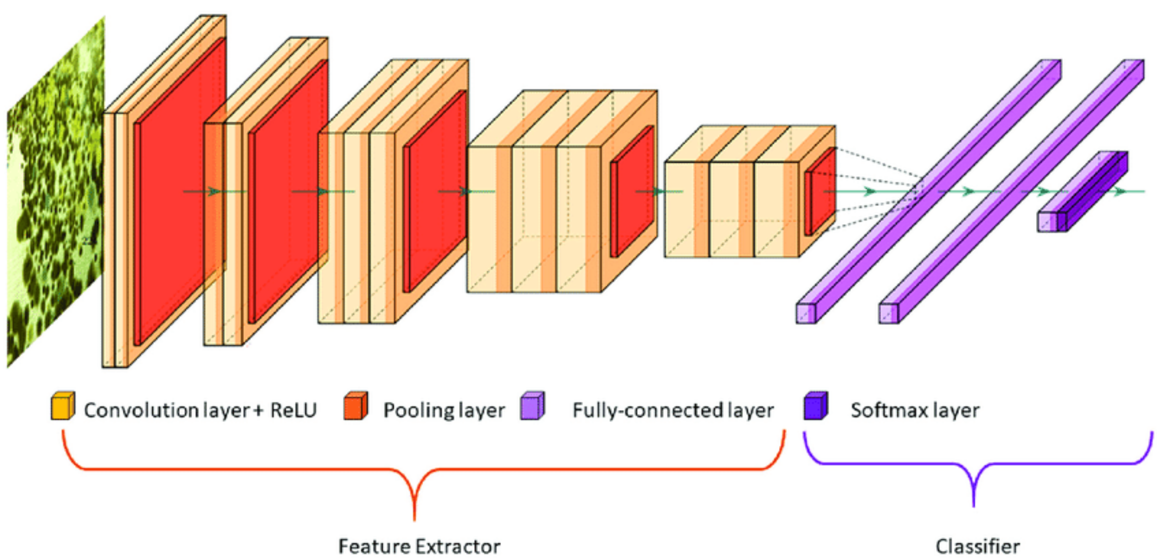
32

CNN for Image Classification



33

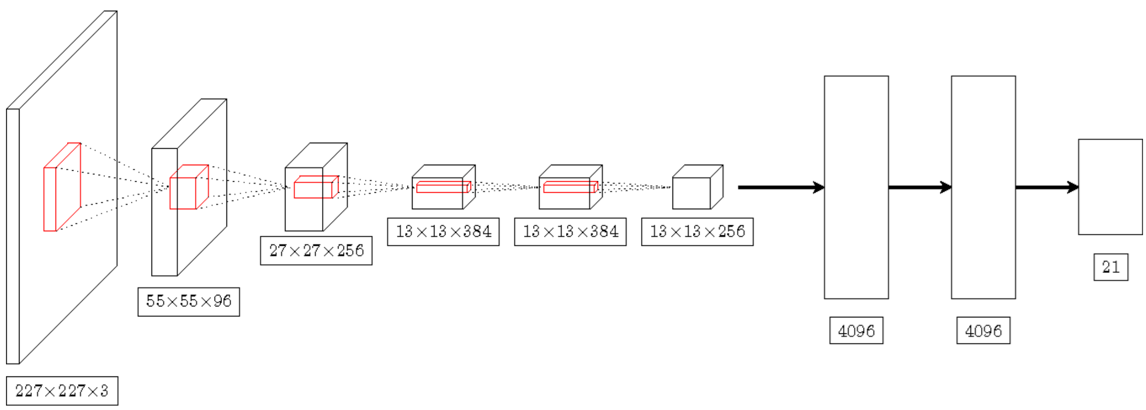
CNN for Image Classification



34

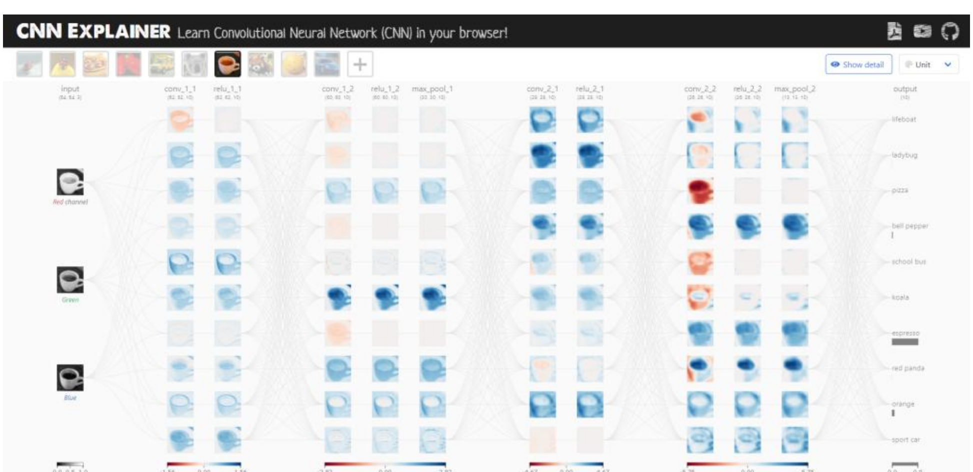
EX1: Convolutional Neural Network (CNN)

- Guess operation(s) being done at each step
- Compute the number of trainable parameters



35

CNN EXPLAINER



"CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," In IEEE Trans. TVCG, 2020
https://arxiv.org/abs/2004.15004 | https://poloclub.github.io/cnn-explainer/

36

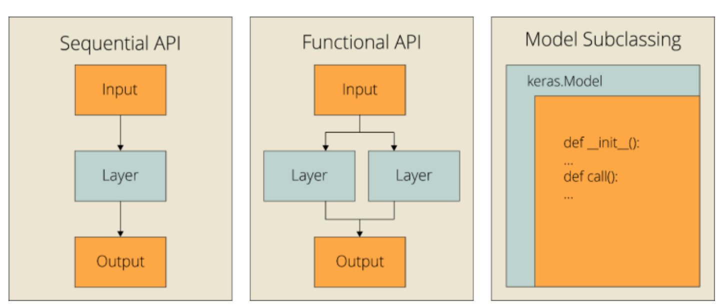
Coding

Combine everything and implement a program with PyTorch

37

CREATING A MODEL

1. Sequential API `keras.models.Sequential()`
   `torch.nn.Sequential()`
2. Functional API `keras.models.Model()`
3. Model Subclassing



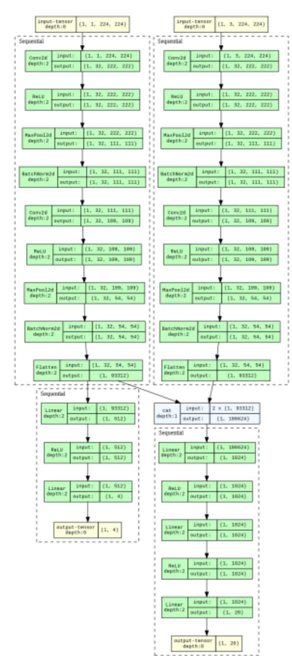
38

EX2: Sequential API vs. Subclassing

- Sequential model:
  - Input: 3-channel RGB image of 502x502 dimension
  - Conv2D_1 (ReLu): 128 kernels of size 7x7, no padding, stride=1
  - Conv2D_2 (ReLu): 64 kernels of size 5x5, no padding, stride=1
  - MaxPool2D_1: pool size = 2x2, stride=2
  - BatchNormalization
  - Conv2D_3 (ReLu): 32 kernels of size 3x3, padding=same, stride=1
  - Conv2D_4 (ReLu): 16 kernels of size 3x3, padding=same, stride=2
  - Conv2D_5 (ReLu): 16 kernels of size 3x3, no padding, stride=2
  - BatchNormalization
  - Flatten
  - Dense1 (ReLu): 1024 nodes
  - Dropout (drop 50%)
  - Dense2 (ReLu): 1024 nodes
  - Dropout (drop 50%)
  - Output (~~softmax~~): 1000 nodes
- Write a program to create this model
- Observe output dimensions and the number of parameters regarding each step

39

EX3: Non-sequential CNN

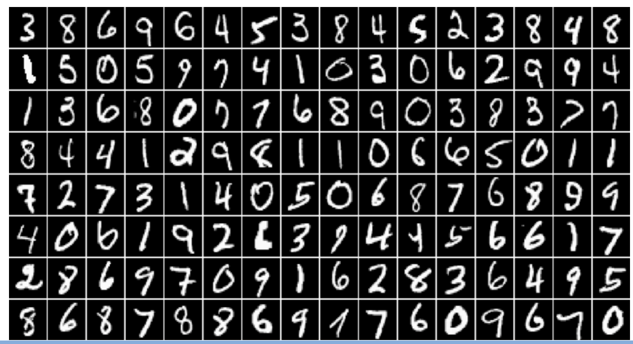


- Non-sequential model refers to:
  - Multiple inputs and/or outputs
  - Skip connection
  - Branching or splitting
- Remind that PyTorch uses a dynamic computational graph by default.

40

EX4: MNIST Digit Classification

- MNIST dataset for handwritten digit classification
- Design, train, and evaluate one MLP (all layers are `Linear` layers)
- Design, train, and evaluate one CNN
- Compare the results from both models



41

42