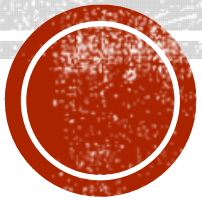
IE4424 Machine Learning Design and Application

Week 3: Convolutional Neural Network (CNN)



Dr Yap Kim Hui

Email: ekhyap@ntu.edu.sg

References

- Stanford Lecture Notes, CS231n: Deep Learning for Computer Vision.
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, http://www.deeplearningbook.org
- Pytorch Tutorial. https://pytorch.org/tutorials/
- University of Wisconsin Madison Lecture Notes, CS638. University of Michigan, EECS 498-007 / 598-005: Deep Learning for Computer Vision
- Shusen Wang, Transformer Model, Youtube Online Videos

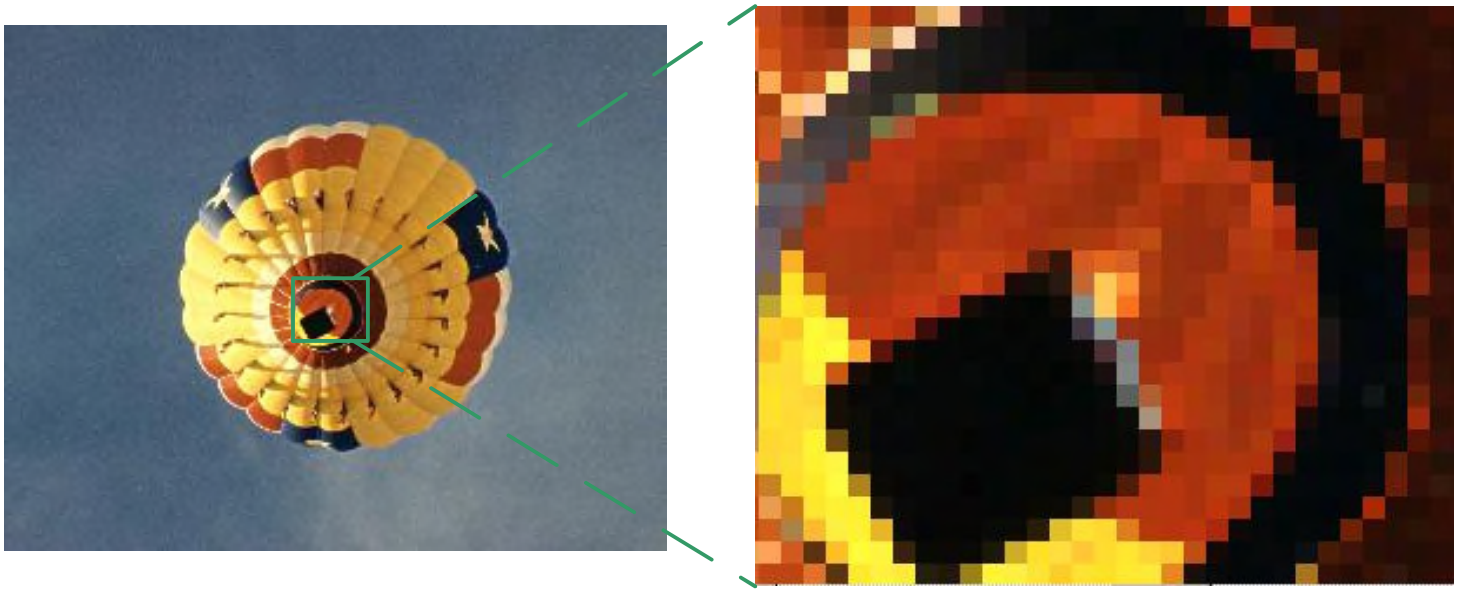
Outline

- Introduction
- Linear Classifier
- Convolutional Neural Networks (CNNs)
- Well-Known CNN Architectures
- New/Emerging Directions

Introduction

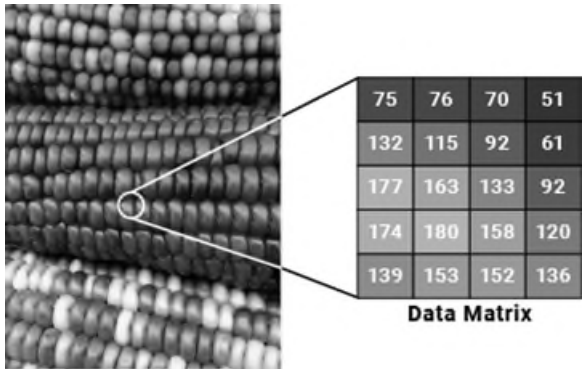
What is an Image?

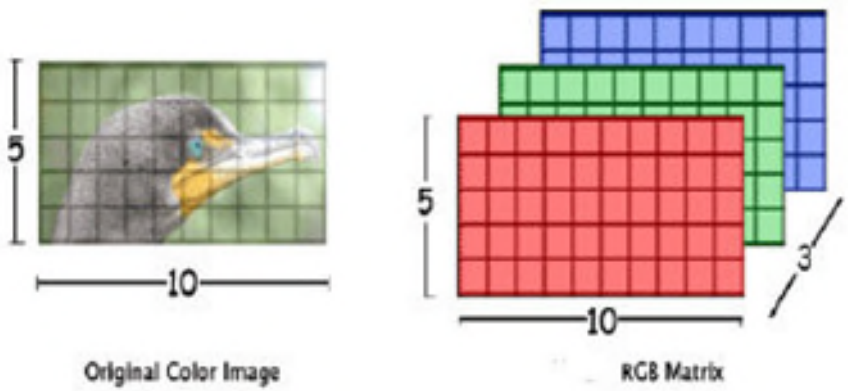
- Digital image is an important media type in information storing, processing, presentation, and interaction.
- Image is represented by a matrix of pixels.
- Resolution: number of pixels in an image
  - Expressed as width x height (e.g., 640 x 480 pixels).
  - Measure the ability to distinguish fine details of images.



Bit Depth/Color Depth

- Bit depth/ color depth: number of bits for each pixel.
- Binary images (1 bit), grayscale images (8 bits) and color images (24 bits).



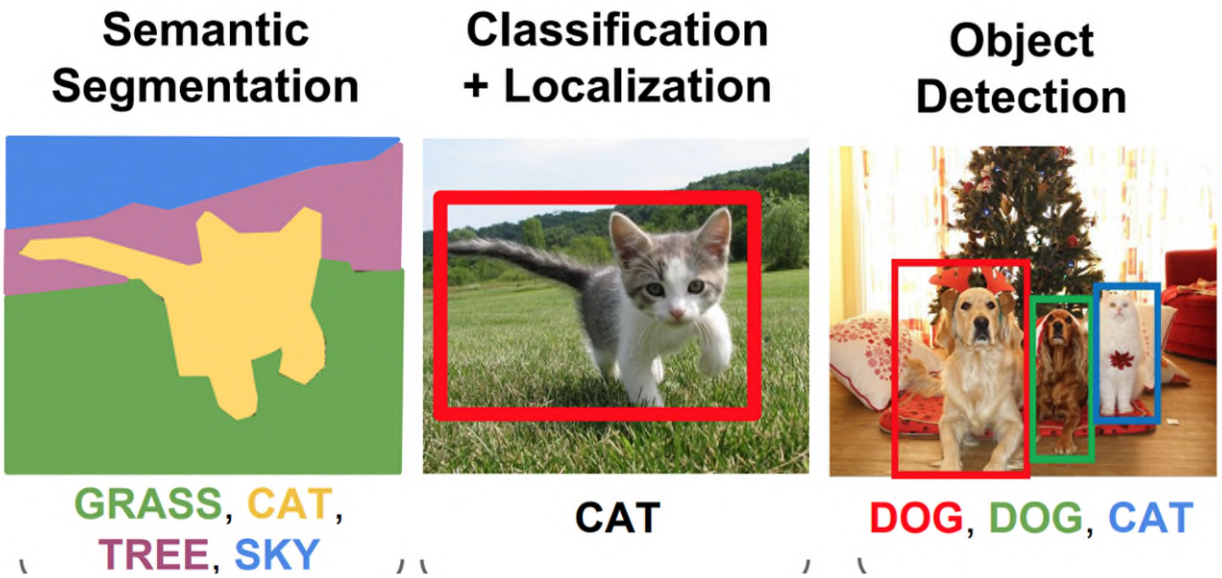


Source: https://www.researchgate.net/figure/Color-image-and-RGB-matrices_fig1_339435251
https://www.mathworks.com/help/matlab/creating_plots/image-types.html

Key CV Tasks

- Image Classification
  - Classify an image into a label/category.
  - Binary vs multi-class.
- Object Detection
  - Detect object instances.
  - E.g., detection of faces, cars, pedestrians, etc.
- Image Segmentation/Semantic Segmentation
  - Label each pixel in an image with a label/category.

Key CV Tasks



Source: Stanford Lecture Notes, CS231n

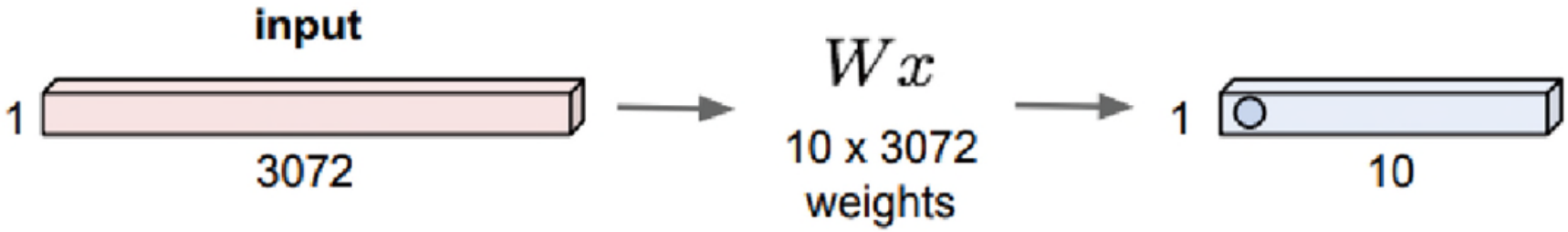
Linear Classifier

Linear Classifier

- Linear classifier
  - Make a classification decision based on a linear combination of input data
- E.g., linear classifier for 10 class labels (e.g., animal classes)

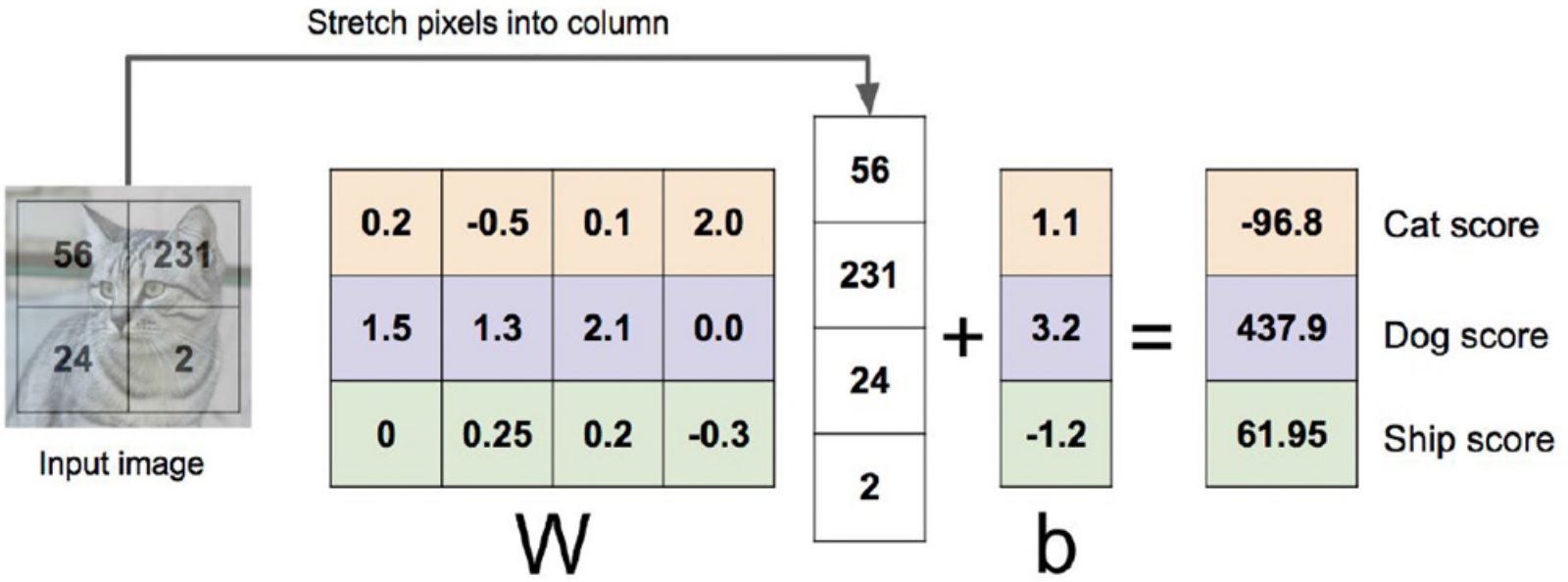
32*32*3 image -> stretch to 3072*1

$$f(x) = Wx + b$$



Source: Stanford Lecture Notes, CS231n

Algebraic Viewpoint



Source: Stanford Lecture Notes, CS231n

Loss Function / Error Function

- How do we determine **W** and **b**?
- We need a loss function (error measurement) which is a metric/distance between predicted values and output target values (teachers) during training.

Common Classification Loss Function

- Softmax loss:
  - Cross-entropy loss with softmax normalization.

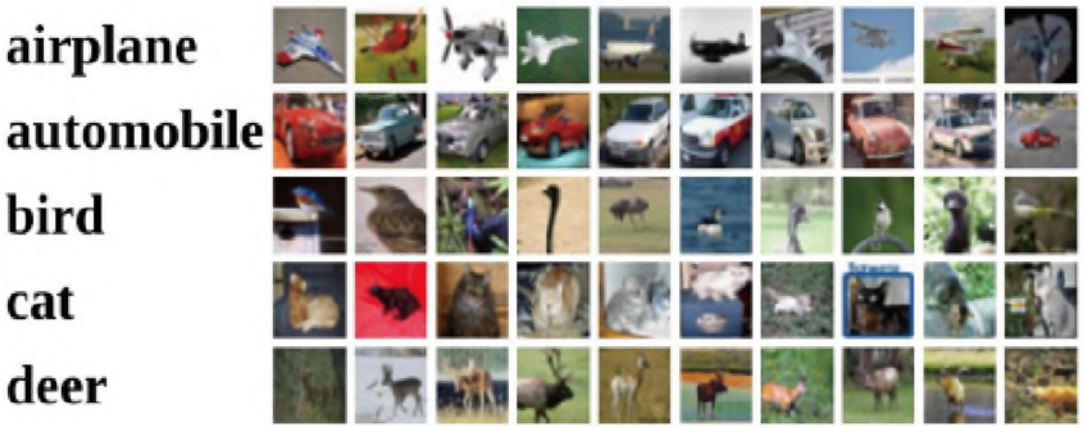
$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}, \text{ where } z_j = f(x_j)$$

$$L = -\sum_j y_j \log_e p_j$$

Softmax Loss Example

- The output scores of a linear classifier for an input cat image are given as follows. What is the softmax loss for this training data?

$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



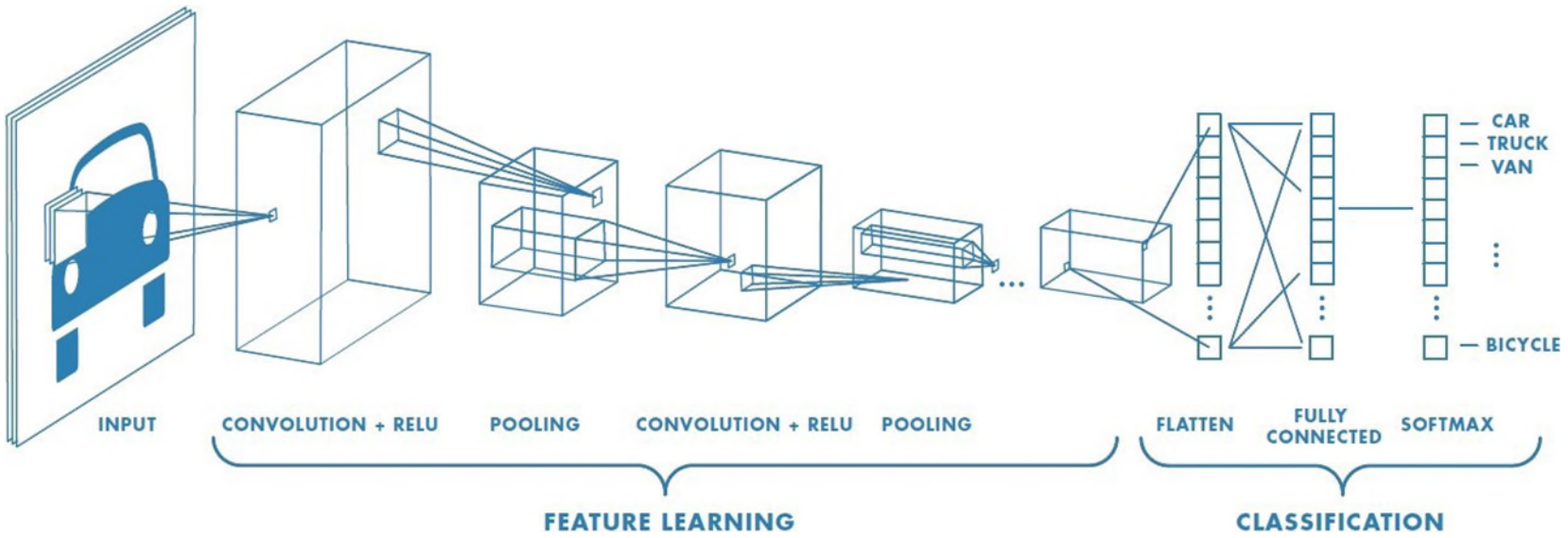
	Predicted output Scores (z)	Ground truth Output Labels (y)	Normalized probabilities (p)
airplane	0.7	0	0.109
automobile	0.1	0	0.060
bird	1.6	0	0.269
cat	2.2	1	0.489
deer	0.3	0	0.073

$$L = -\sum_j y_j \log p_j = 0.715$$

(Note: The log is natural log namely ln)

CNN Architecture

CNN Architecture



Source: “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”, towardsdatascience.com

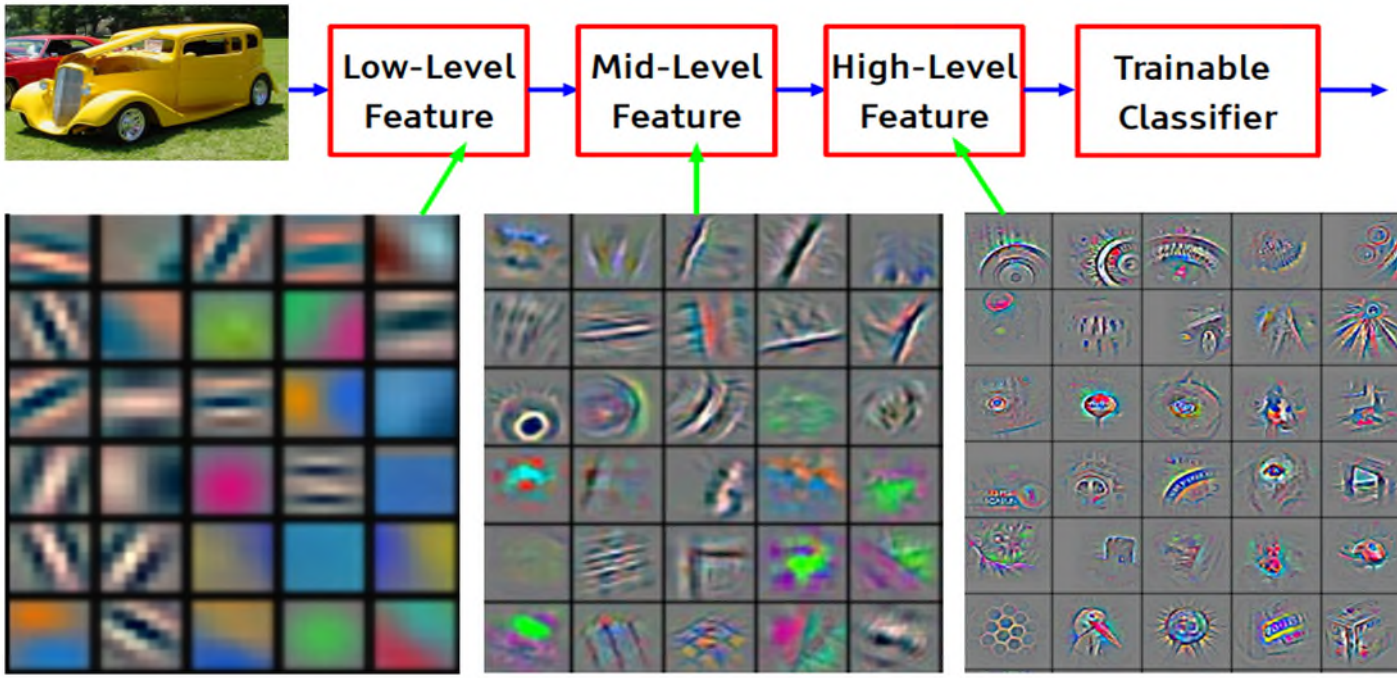
CNN Architecture

- Convolutional Layer
- Activation Function Layer
- Pooling Layer
- Fully-Connected (FC) Layer / Linear Layer
- Softmax Layer

Convolutional Layer

- Extract features in a hierarchical manner.
- Early layers extract low-level features whereas later layers extract high-level features.
- Reduce parameters to be trained by sharing weights through convolution kernel.

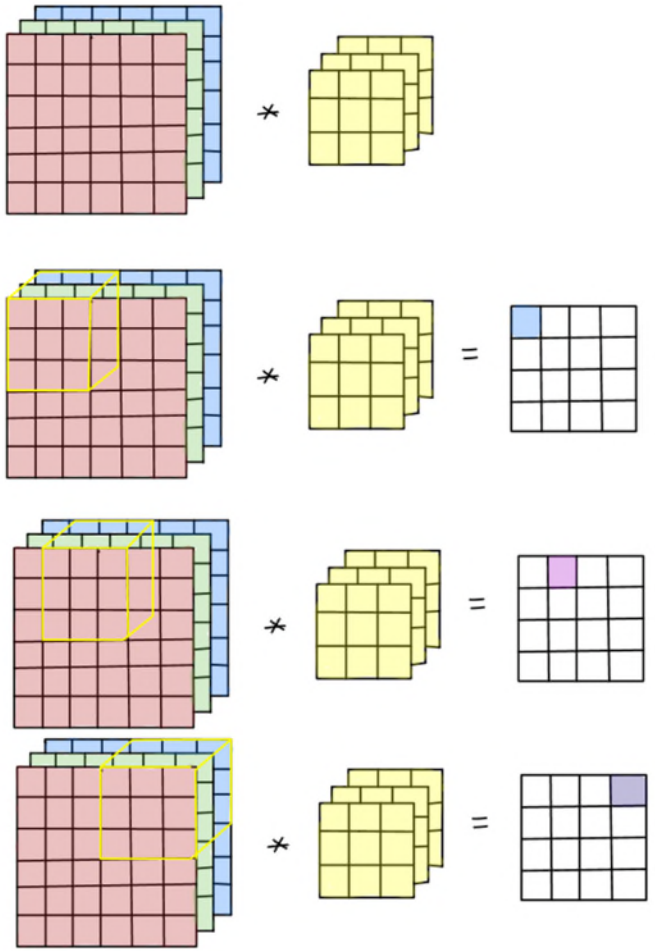
Convolutional Layer Feature Visualization

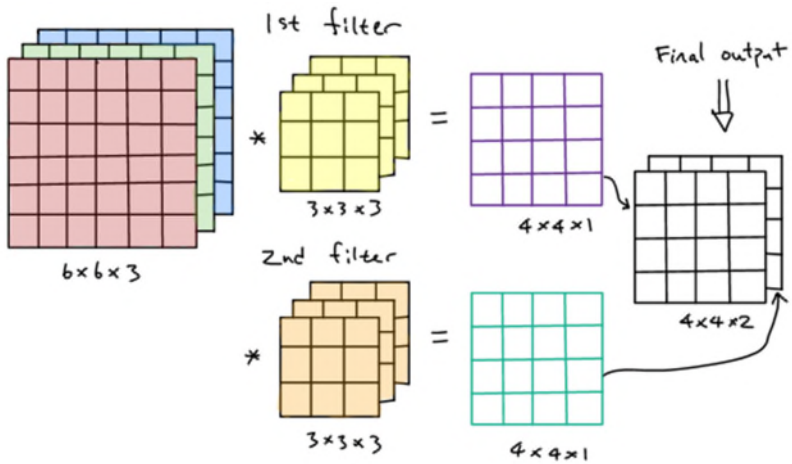


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolution

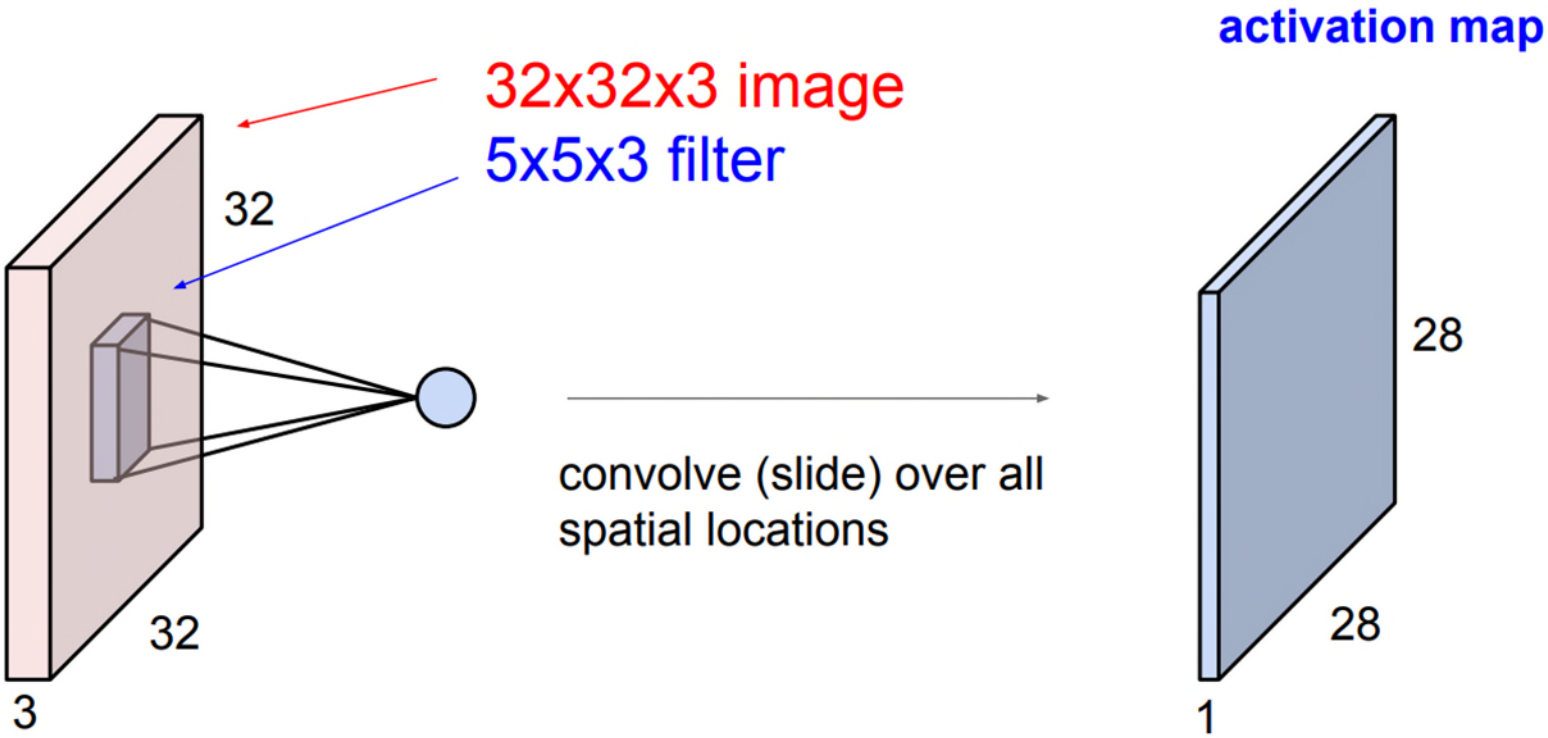
- Convolution / filtering = dot product / sum of product operation.





Source: https://guandi1995.github.io/One-Layer-of-a-Convolutional-Networks/

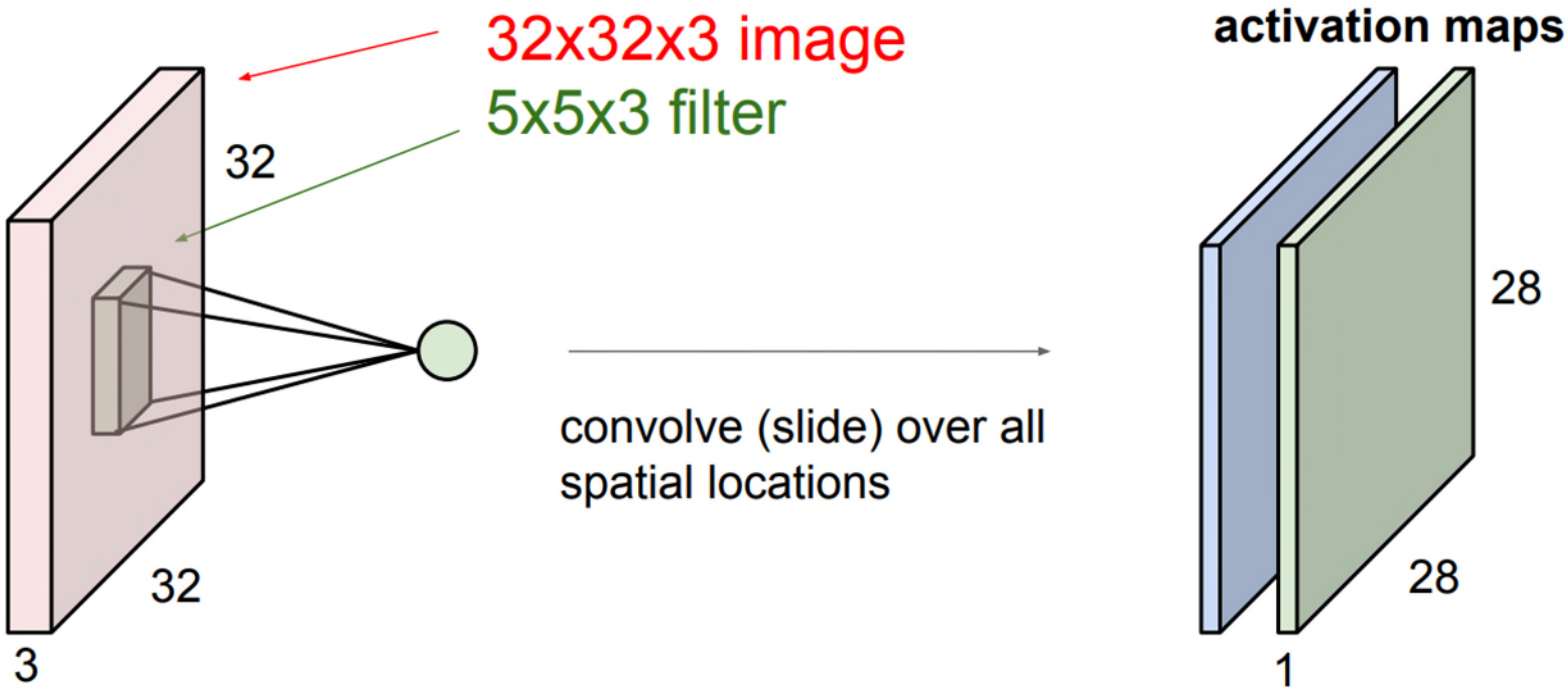
Convolutional Layer



Source: Stanford Lecture Notes, CS231n

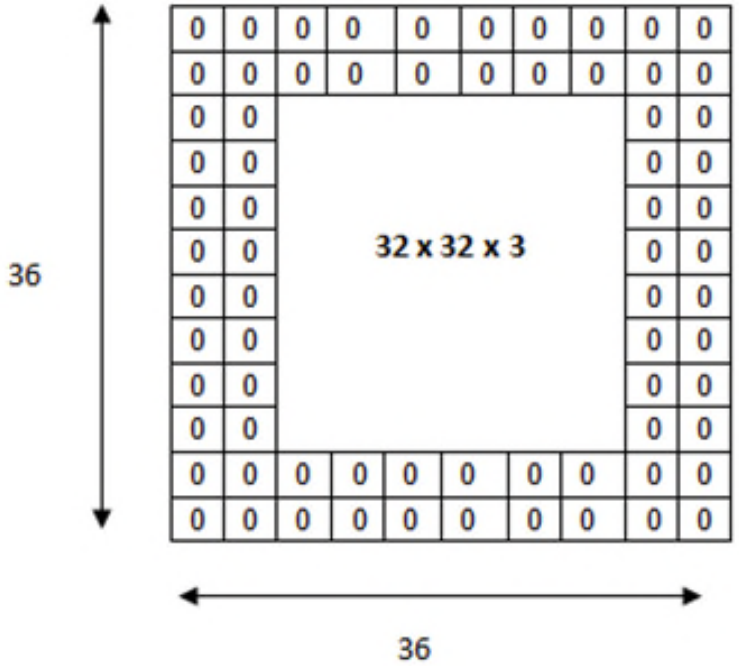
Convolutional Layer

consider a second, green filter



Source: Stanford Lecture Notes, CS231n

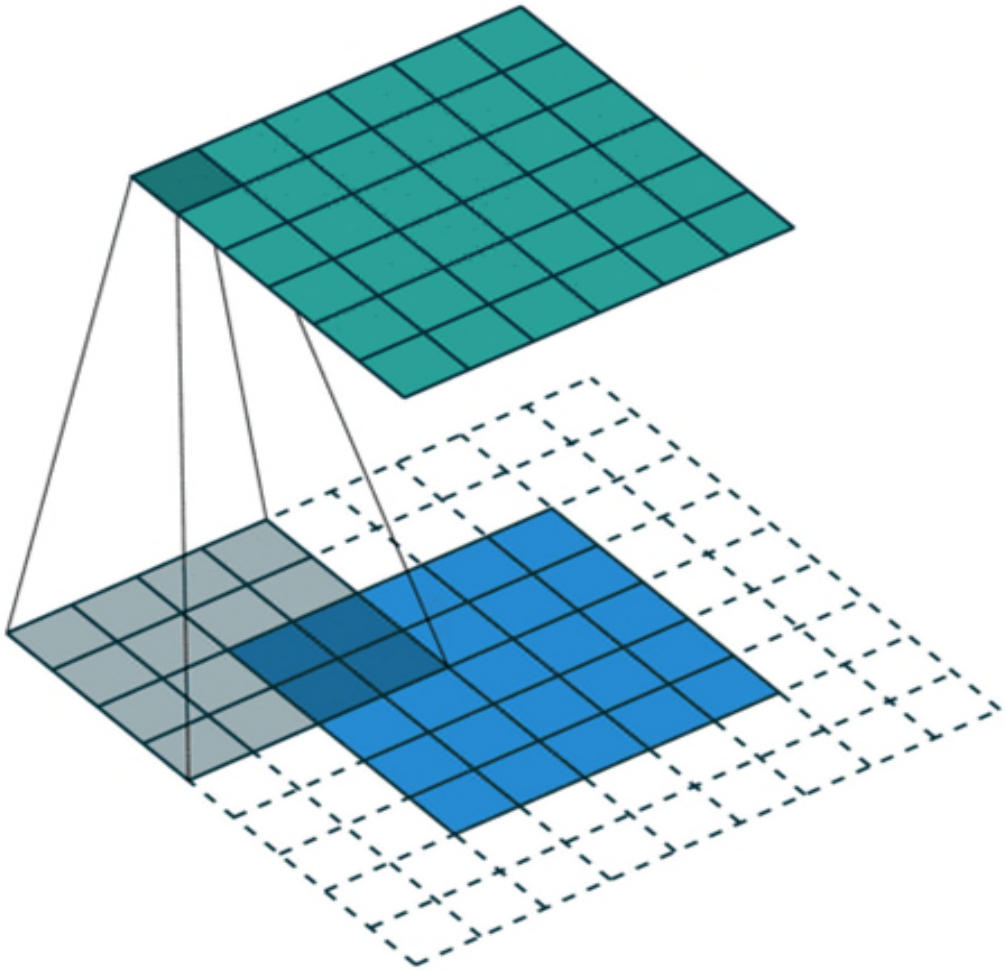
Padding



The input volume is $32 \times 32 \times 3$. If we pad two borders of zeros around the volume, this gives us a $36 \times 36 \times 3$ volume. Then, when we apply our conv layer with three $5 \times 5 \times 3$ filters and a stride of 1, we will get a $32 \times 32 \times 3$ output volume.

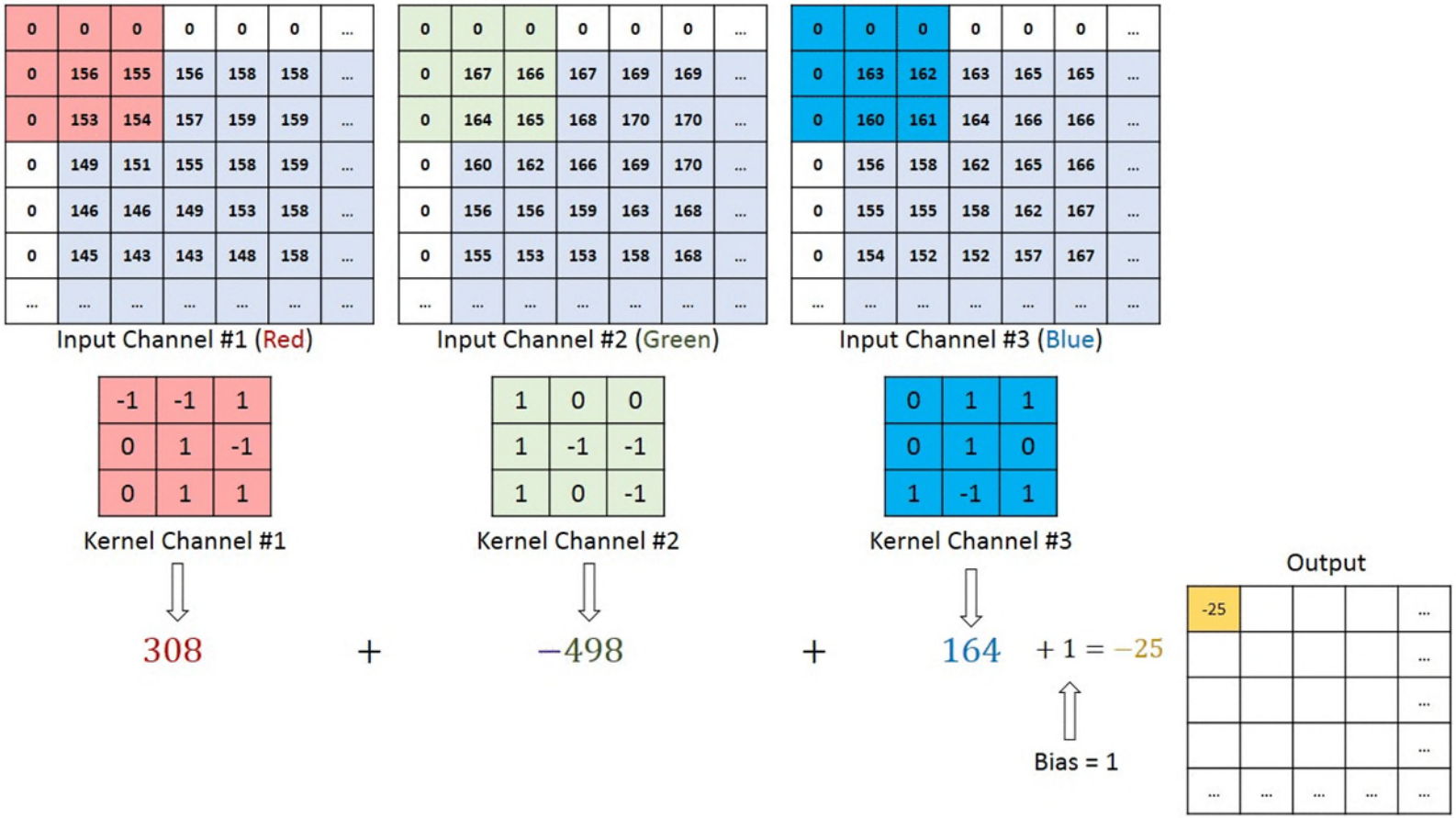
Source: “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2”, Adit Deshpande, github.io

Padding



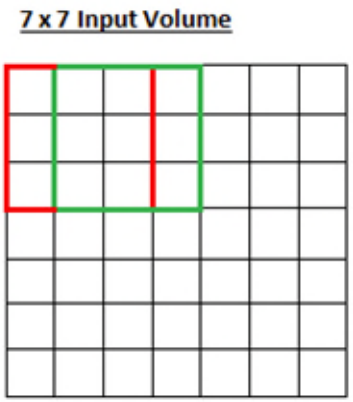
Source: cloud.tencent.com/developer/article/1031131

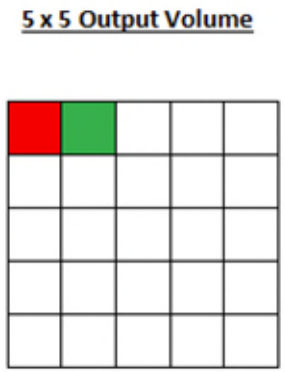
Padding + Convolution



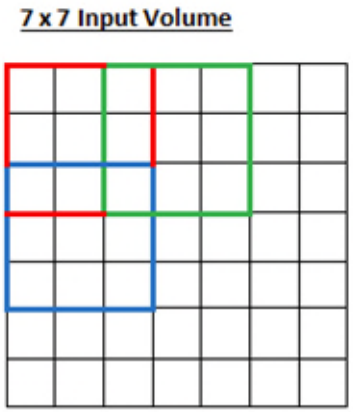
Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks

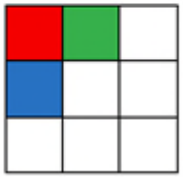
Stride





3 × 3 filter with stride 1



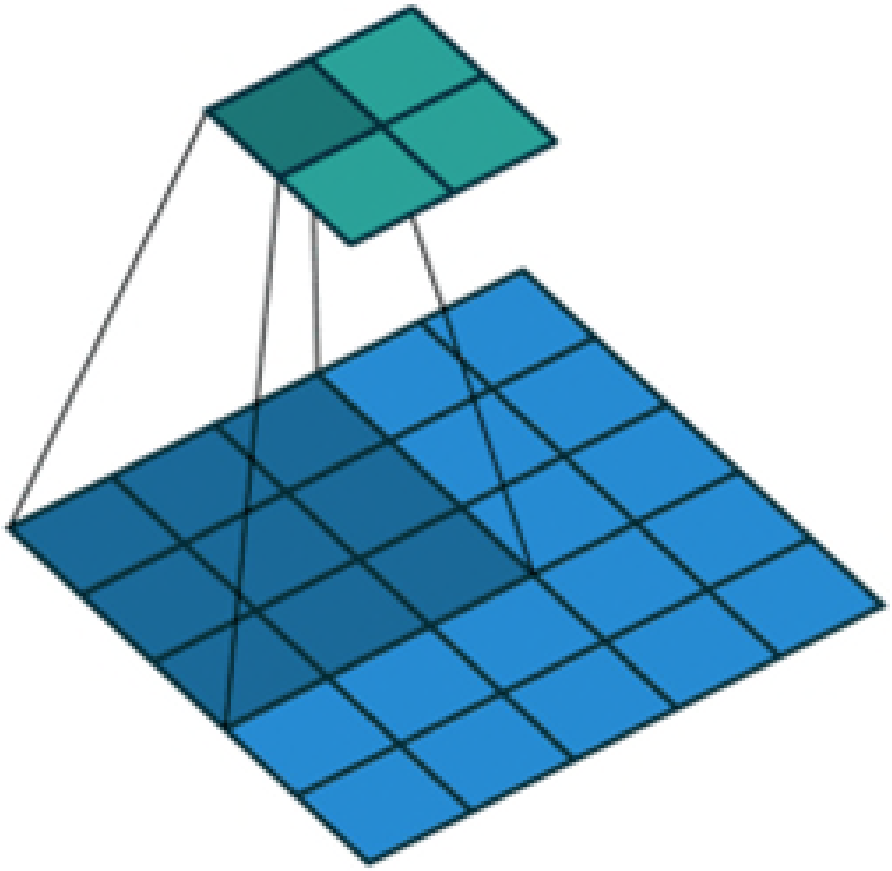


3 × 3 filter with stride 2

Source: “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2”, Adit Deshpande, github.io

Stride

- Stride = 2:



Source: cloud.tencent.com/developer/article/1031131

Convolutional Layer Dimension

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Source: Stanford Lecture Notes, CS231n

Convolution Layer

- Conv layer in Pytorch

```
CLASS torch.nn.Conv2d(in_channels: int, out_channels: int,
      kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T,
      T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T,    [SOURCE]
      Tuple[T, T]] = 1, groups: int = 1, bias: bool = True,
      padding_mode: str = 'zeros')
```

- How to define

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```


Source: Pytorch official tutorial

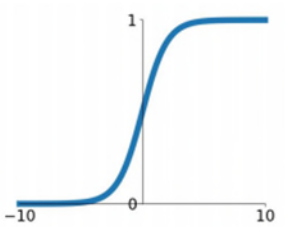
Activation Layer

- Perform element-by-element nonlinear activation function mapping.
- RELU is one of most popular activation functions.
- Often combined with convolution layer.

Activation Layer

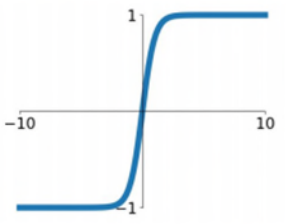
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



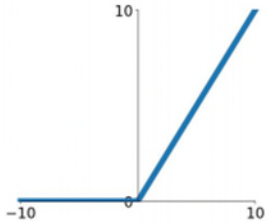
tanh

$$\tanh(x)$$



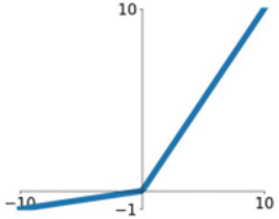
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

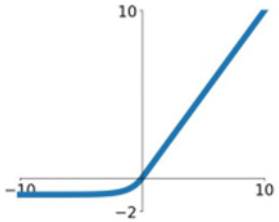


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

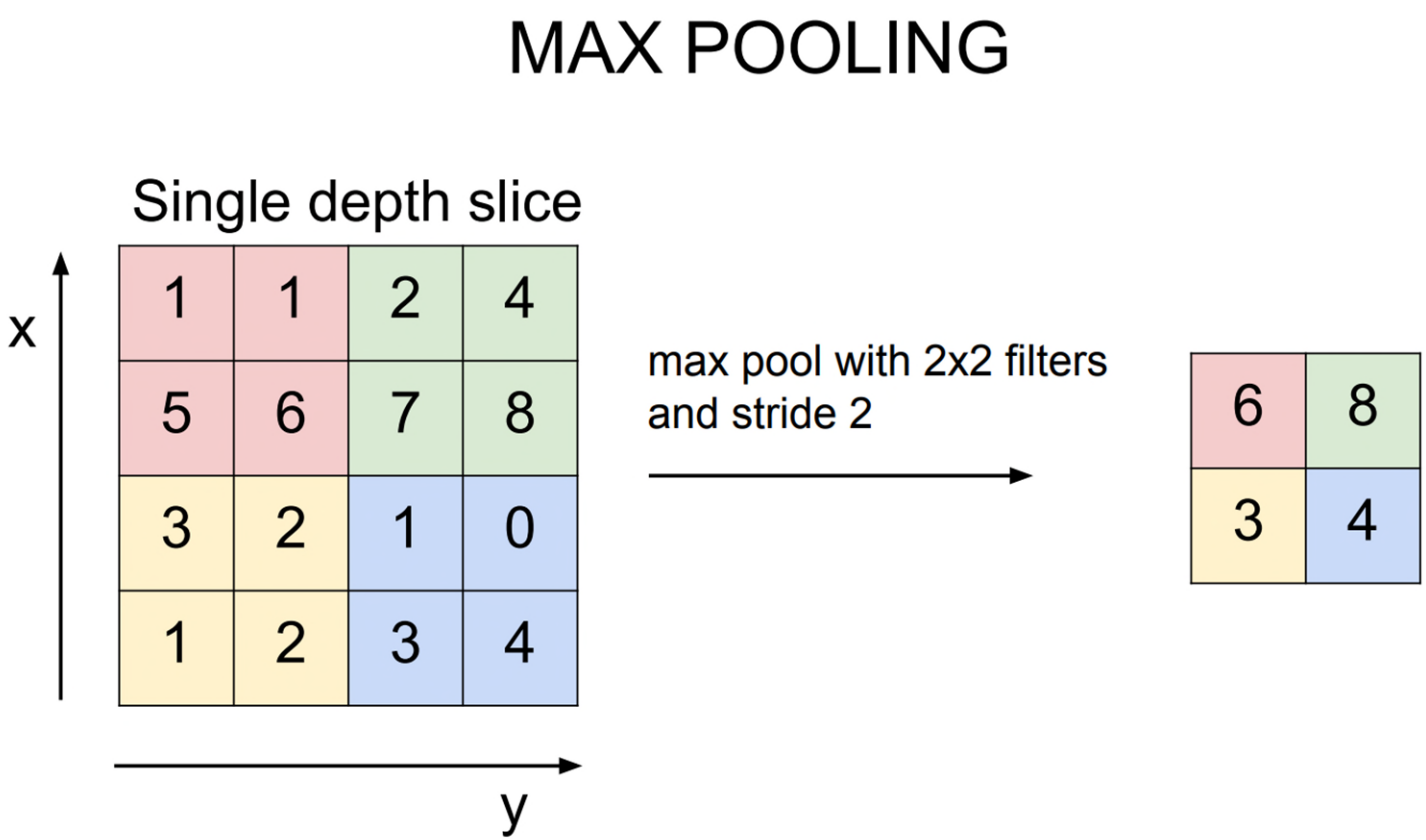


Source: Stanford Lecture Notes, CS231n

Pooling Layer

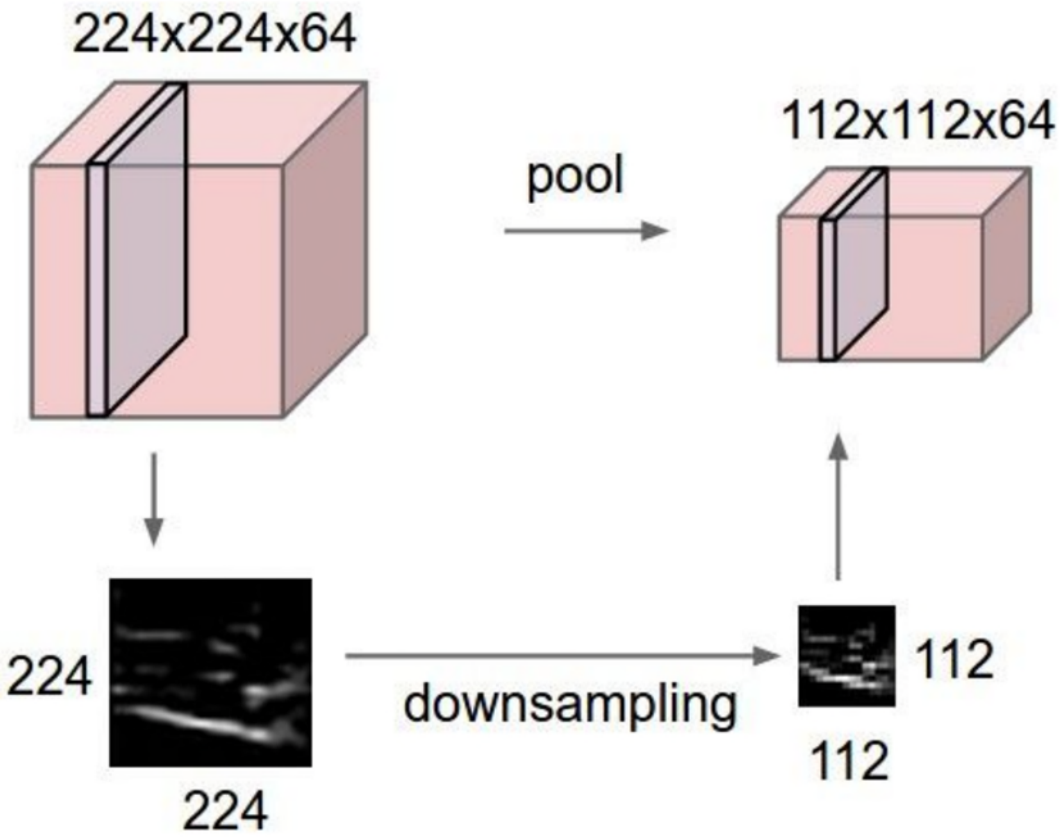
- Reduce activation map dimension, hence reducing computation and storage requirement.
- Provide some degree of location/position invariance.
- Common pooling operation: max pooling, average pooling.
- Each channel of activation/feature maps operate independently.

Pooling Layer



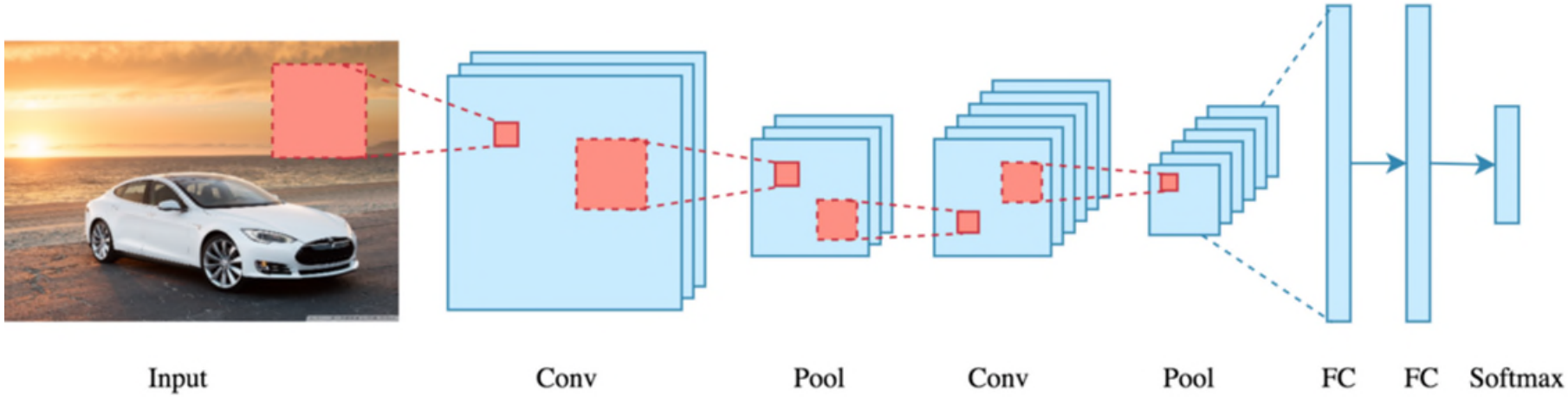
Source: Stanford Lecture Notes, CS231n

Pooling Layer



Source: Stanford Lecture Notes, CS231n

FC Layer



Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

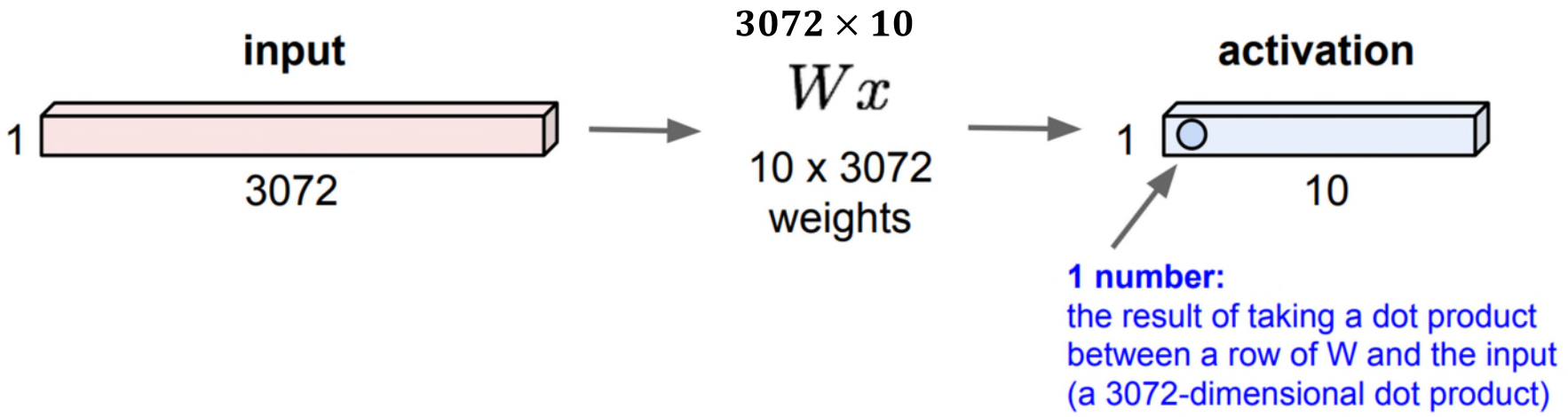
Fully Connected (FC) Layer

- All nodes in one layer are connected to all nodes in the next layer.
- FC layer feature vector can be used as feature (also known as embedding/descriptor) to represent input image.
- FC layer behaves like a linear layer for classification.

FC Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



Source: Stanford Lecture Notes, CS231n

FC Layer

- Define a FC layer

LINEAR

CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

- How to use it

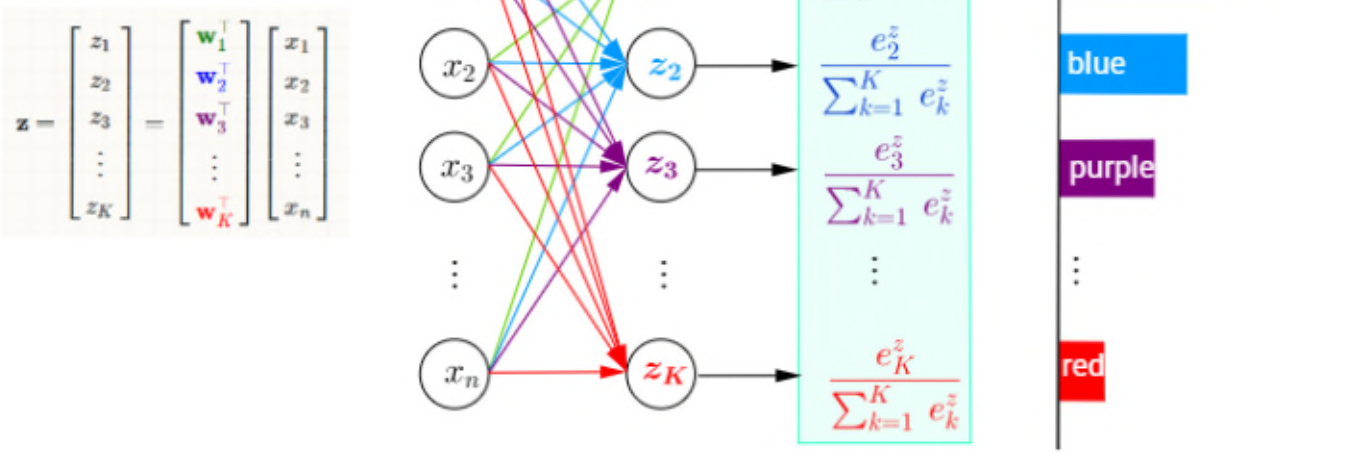
```
>>> m = nn.Linear(20, 30)
>>> input = torch.randn(128, 20)
>>> output = m(input)
>>> print(output.size())
torch.Size([128, 30])
```


Source: Pytorch official tutorial

Softmax Layer

- Map the output scores (logits) from last FC layer into probabilities for classification problem.
- Softmax loss is computed based on the probabilities.

$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



Source: https://stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights

Sample Pytorch CNN Implementation

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Conv Layer

Fully-connected Layer

Pooling Layer

Activation Layer

Layer Definition

Feed Forward Logic

Exercise: CNN Convolution Layer

An input feature matrix I passes through a convolution layer of CNN with the following settings:

Input: $I = \begin{bmatrix} -7 & 1 & 1 \\ 3 & 2 & -8 \\ 0 & -7 & 6 \end{bmatrix}$;

Kernel: $K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$;

Amount of zero padding (at each end of the width and height dimension): 1;

Stride (both horizontally and vertically): 1.

Find the output of this Convolution layer.

Solution

Exercise: CNN FC and Softmax Layers

(a) A particular Convolutional Neural Network (CNN) consists of several convolutional+RELU layers, pooling layers, Fully Connected (FC) layers, and followed by a Softmax layer. The last FC layer of the CNN has the following values for a training sample:

Input to the last FC layer: $\mathbf{x} = \begin{bmatrix} 2 & 2 & 3 & 1 & 0 \end{bmatrix}^T$,

Last FC layer weight matrix: $\mathbf{W} = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.2 & 0.2 \\ 0.4 & 0.3 & 0.2 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$,

Last FC layer bias: $\mathbf{b} = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}^T$.

(i) Find the output $\mathbf{y}$ of the last FC layer.

(ii) Find the output $\mathbf{p}$ after the Softmax layer.

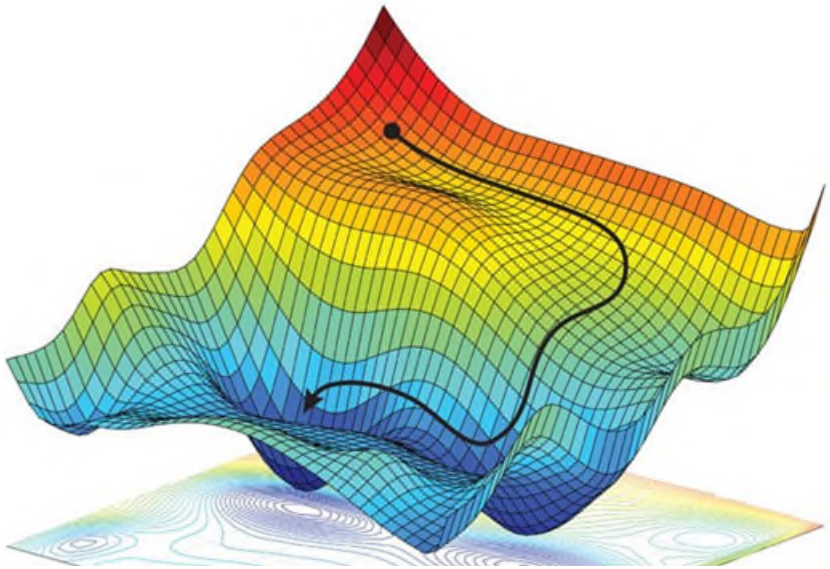
(iii) Assume that the ground truth output label of this training sample is given by $\mathbf{t} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$, calculate the Softmax loss for this training sample.

Solution

CNN Training & Optimization

CNN Training

- The objective is to minimize/optimize the loss function.
- Common strategy is centered to stochastic gradient descent (SGD) and its variants.
- Use computational graph to compute the gradient descent and parameter update.



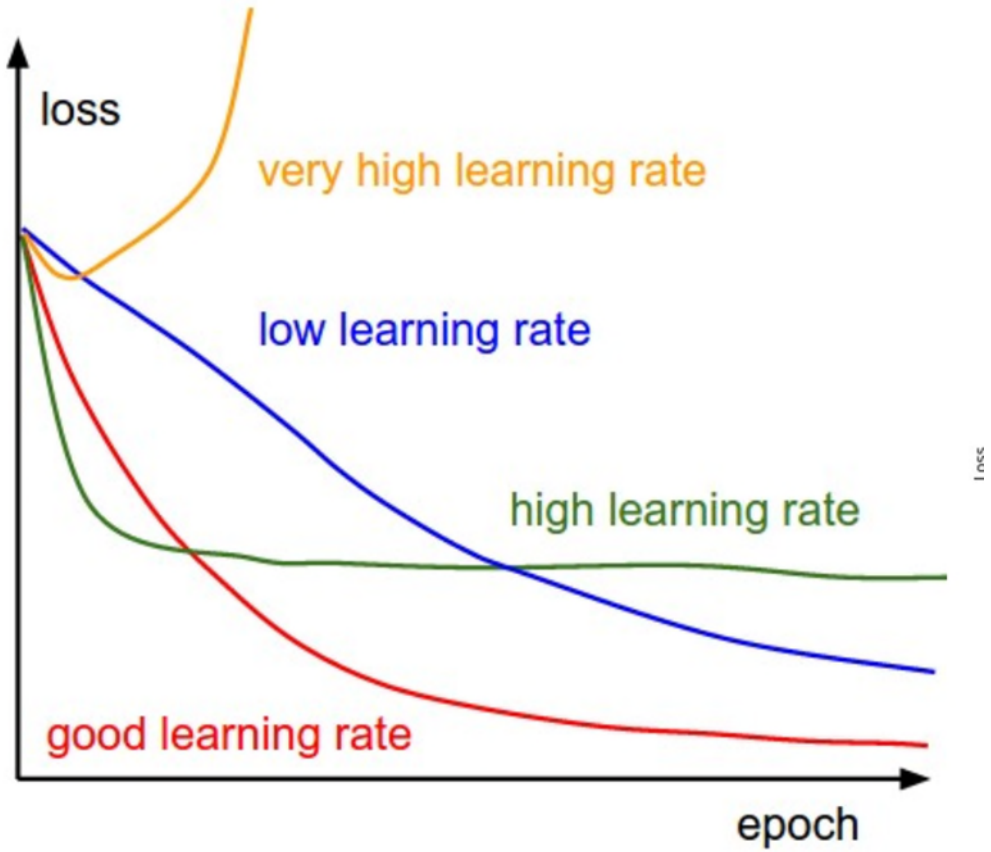
Source: “Coding Deep Learning for Beginners — Linear Regression Part 3”, Kamil Krzyk, towardsdatascience.com

CNN Training

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

$$w_{t+1} = w_t - \alpha \nabla L(w_t)$$



Source: Stanford Lecture Notes, CS231n

Optimizer

- Objective: to optimize the parameters in the network through learning.
- Common optimizers: Stochastic Gradient Descent (SGD), Adam, etc.
- CNN Training in Pytorch

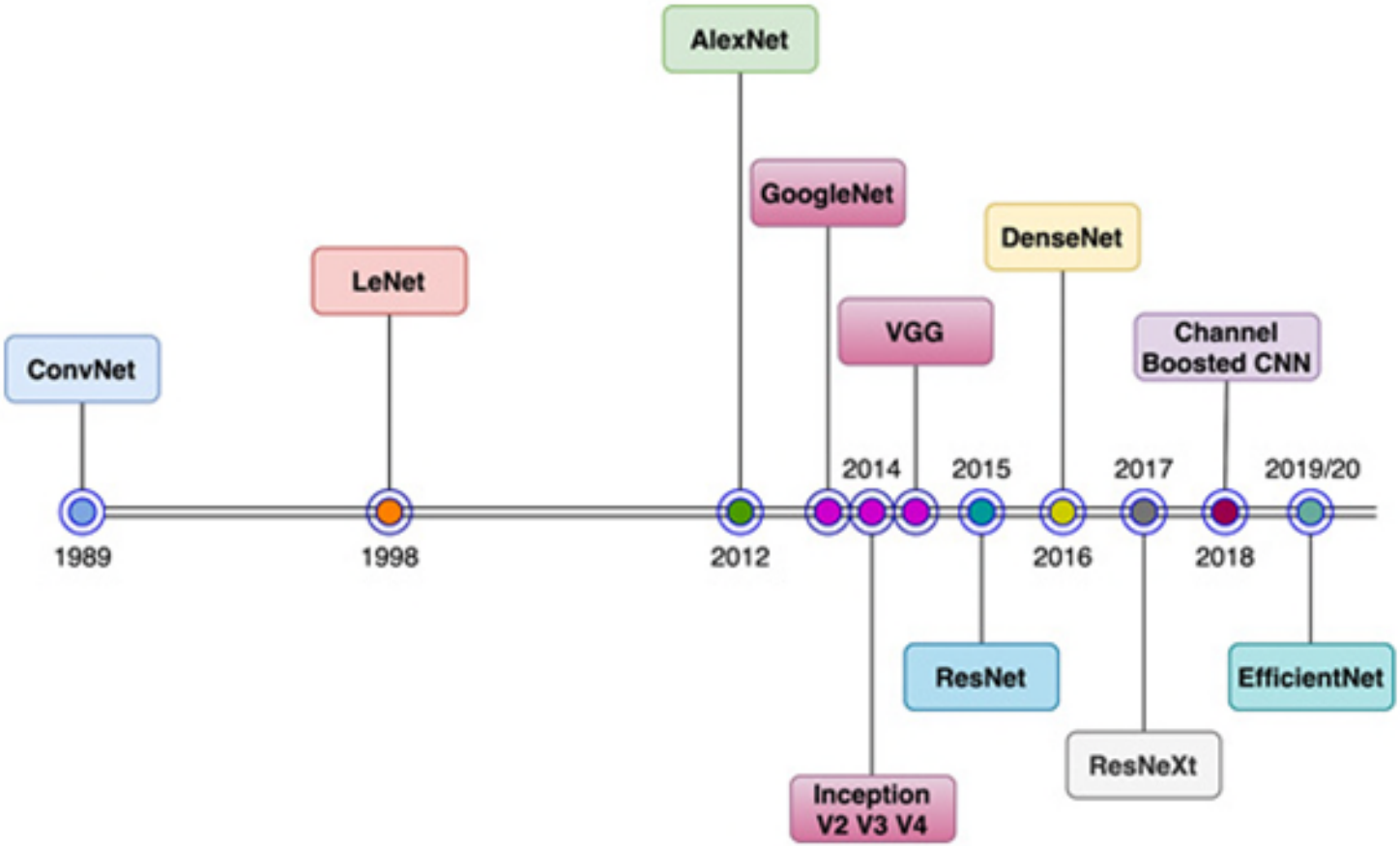
```
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```


Source: Pytorch official tutorial

Well-Known CNN Architectures

CNN Timeline / Milestone

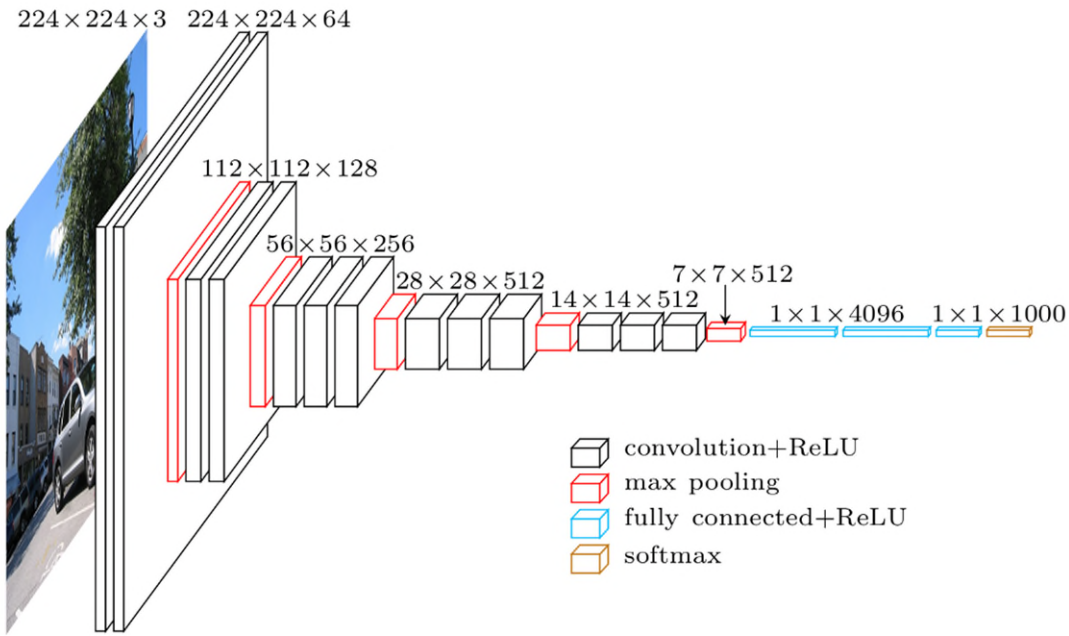


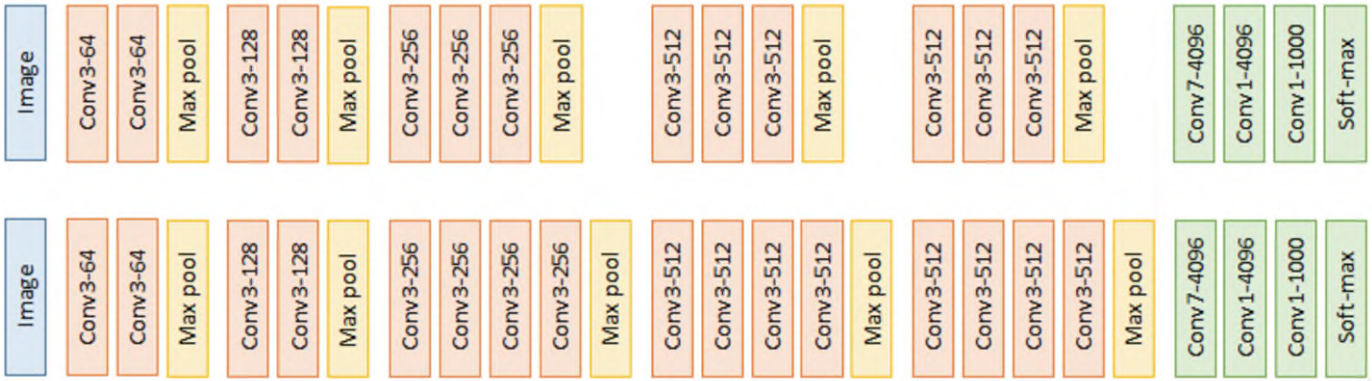
Source: packtpub.com/../evolution-of-cnn-architectures

VGG Network

- Runner up of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014.
- Elegant network architecture.
- Use deep network with small 3×3 kernels.
- Require large number of parameters.
- Two common variants: VGG-16 and VGG-19

VGG Network



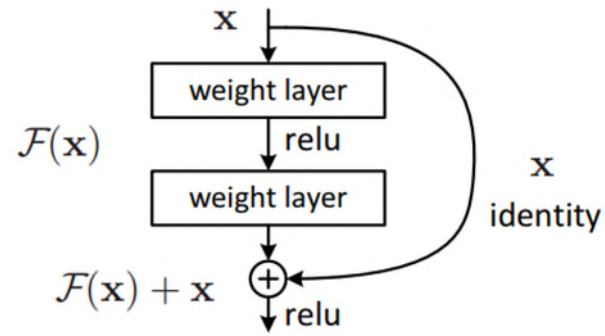


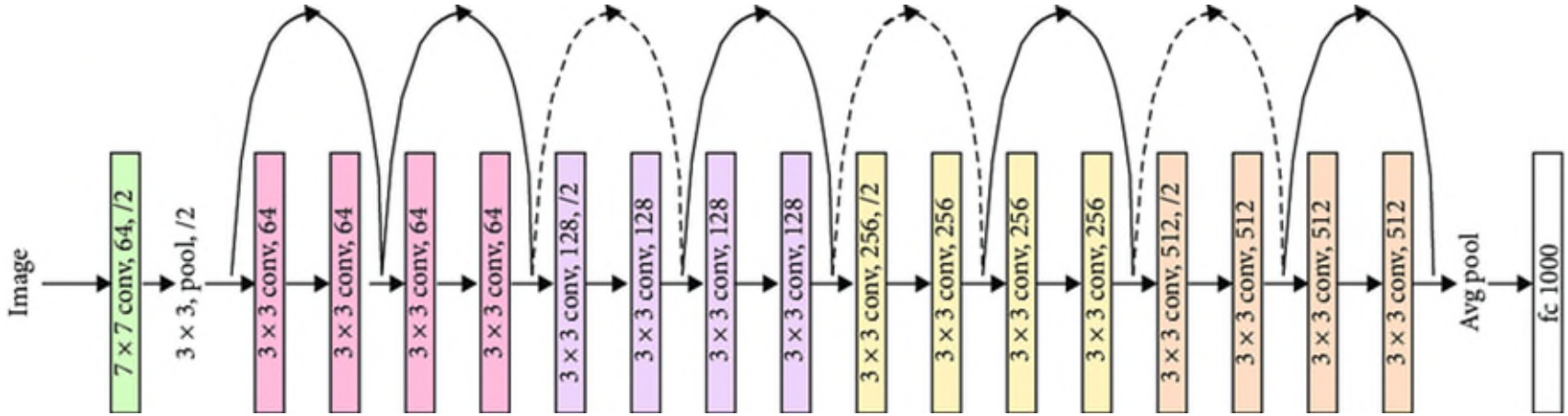
Source: “Very deep convolutional networks for large-scale image recognition”, K. Simonyan et al., 2014

ResNet

- Winner of ILSVRC 2015
- Very deep structure
- Use residual block and highway/skip connection for better gradient backpropagation.

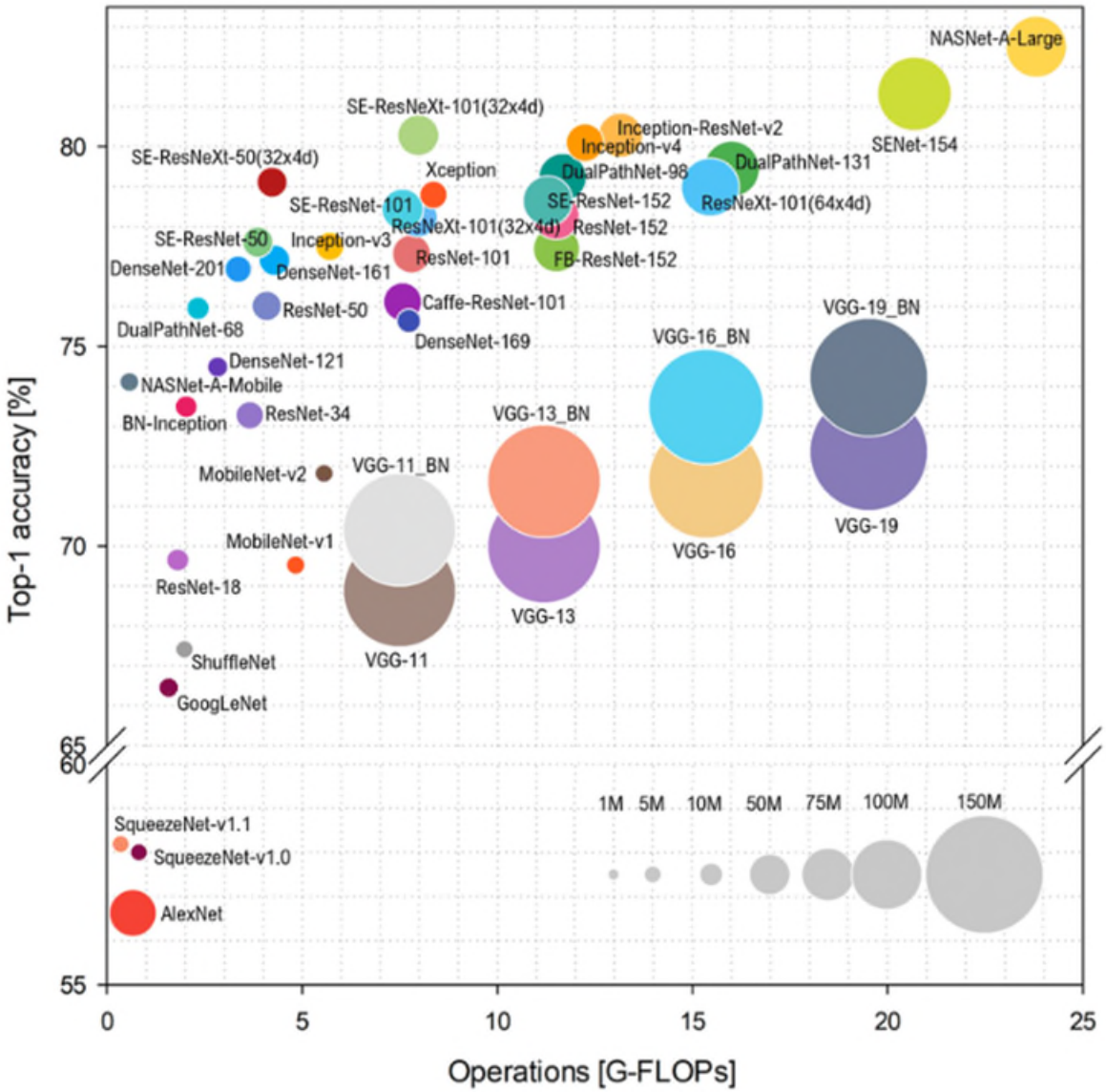
ResNet





Source: “Deep residual learning for image recognition”, Kaiming He et al., 2016

Performance Comparison on ImageNet



Key Performance Metrics

- Accuracy
- Memory footprint (parameters/weights + activation maps)
- Speed/computational complexity (FLOPS)
- Often, what is the important metric will depend on the applications/problems.

New / Emerging Directions

What is a Transformer?

- Initially designed for neural machine translation.
- Later extended to visual tasks such as recognition, detection, etc, with great success.
- Leverage on attention mechanism to analyze the importance of a token (word/image patch) with respect to other tokens/image patches.
- Consist of transformer encoder and transformer decoder.

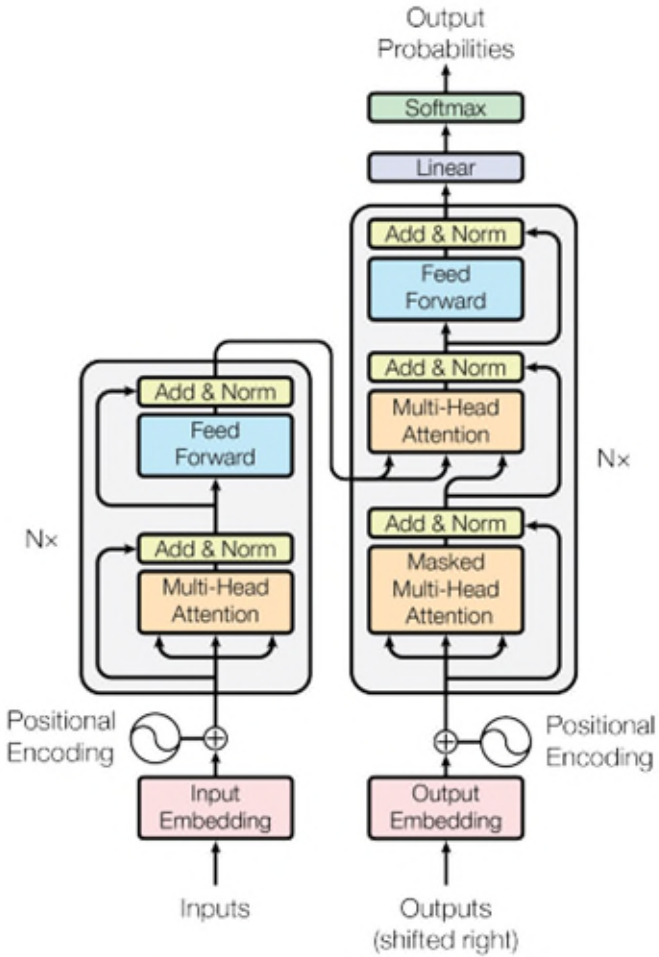


Figure 1: The Transformer - model architecture.

Attention Is All You Need (2017) https://arxiv.org/abs/1706.03762
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Vision Transformer

- A state-of-the-art image classification model based on attention.

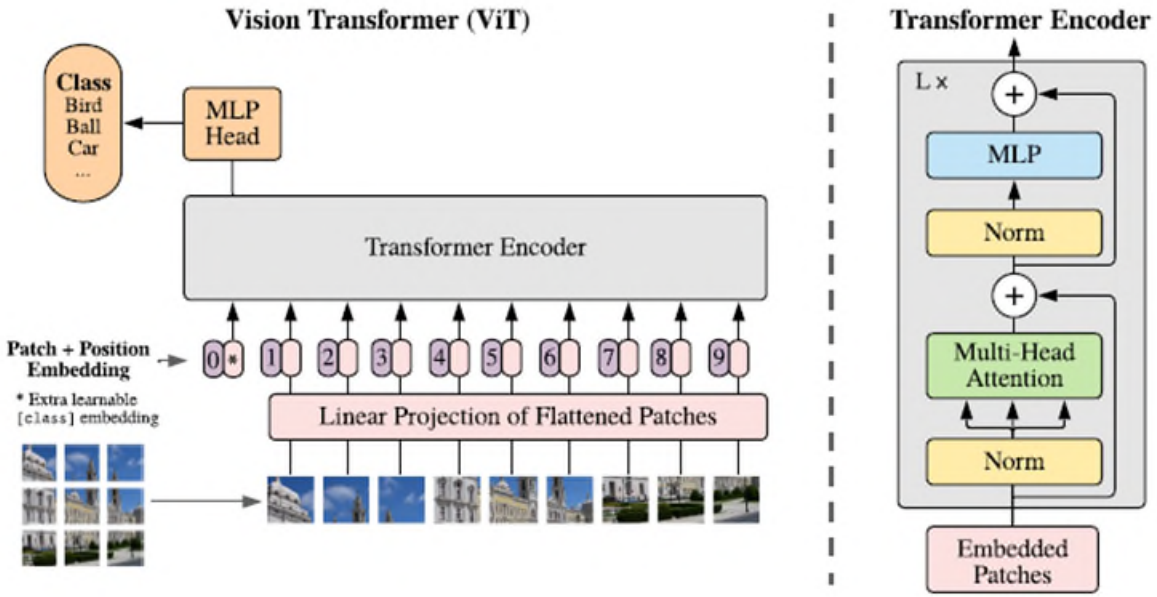


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (2020)
Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby

Summary

- The part covers the following topics:
  - Introduction
  - Linear Classifier
  - Convolutional Neural Networks (CNNs)
  - Well-Known CNN Architectures
  - New/Emerging Directions