Deep Learning and Convolutional Neural Network (42028)

Convolutional Neural Network (CNN) - 3

Optimizers

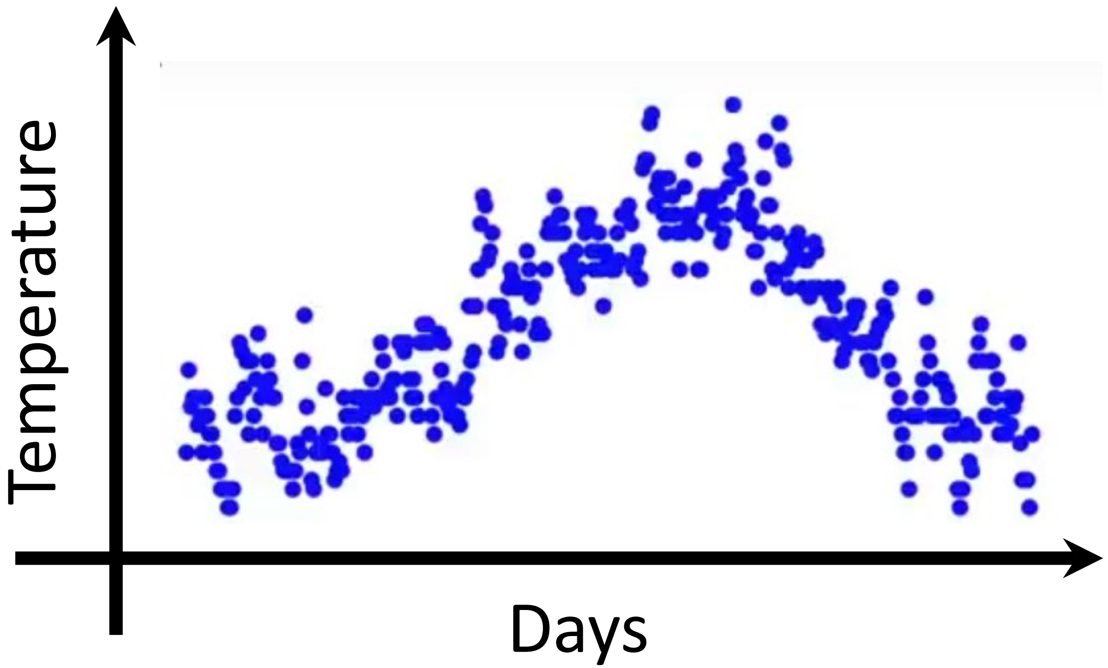
- SGD with momentum
- RMSProp
- Adam

Exponentially Weighted Averages

- One of the popular algorithm for smoothing sequential data
- Also called Moving Average
- Weight the number of observations and using their average

- Example:

  Temperature over θ days



Source and reference https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://medium.com/datadriveninvestor/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9

Exponentially Weighted Averages

V_t : Moving average on day 't'

So, let

$V_0 = 0$

$V_1 = 0.9\ V_0 + 0.1\ \theta_1$

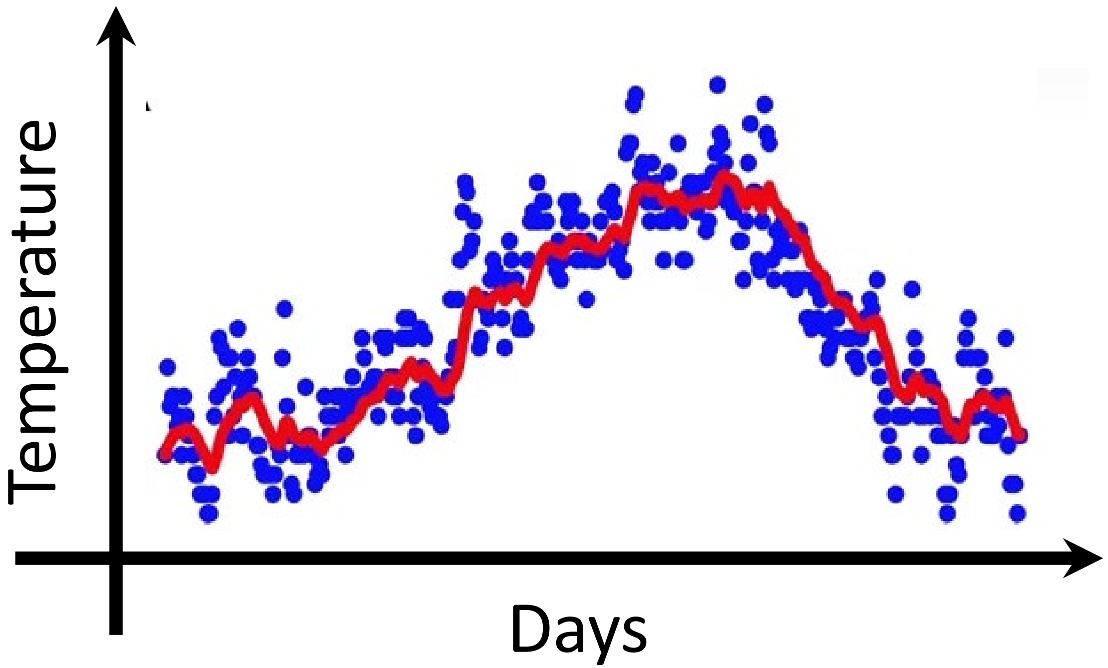
$V_2 = 0.9\ V_1 + 0.1\ \theta_2$

$V_3 = 0.9\ V_2 + 0.1\ \theta_3$

:

:

$V_t = 0.9\ V_{t-1} + 0.1\ \theta_t$



Source and reference https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://medium.com/datadriveninvestor/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9

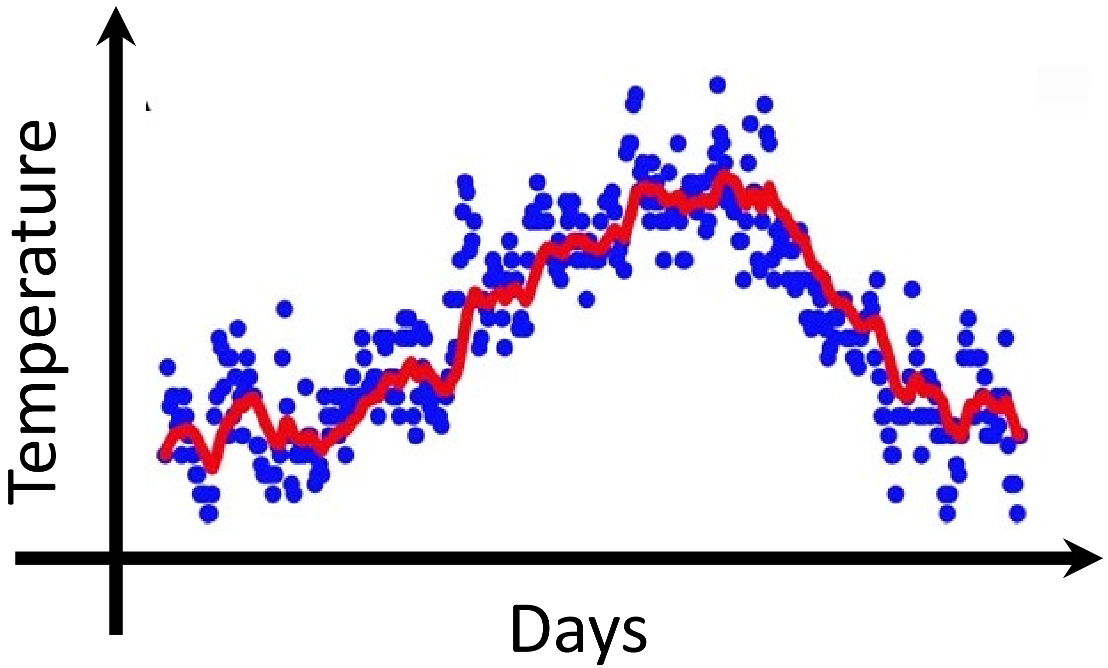
Exponentially Weighted Averages

$V_t = 0.9\ V_{t-1} + 0.1\ \theta_t$

If $\beta = 0.9$,

$$\boxed{V_t = \beta\, V_{t-1} + (1-\beta)\ \theta_t}$$

This equation gives the moving average shown by the red line.



Source and reference https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://medium.com/datadriveninvestor/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9

Exponentially Weighted Averages

$$V_t = \beta V_{t-1} + (1-\beta)\ \theta_t$$

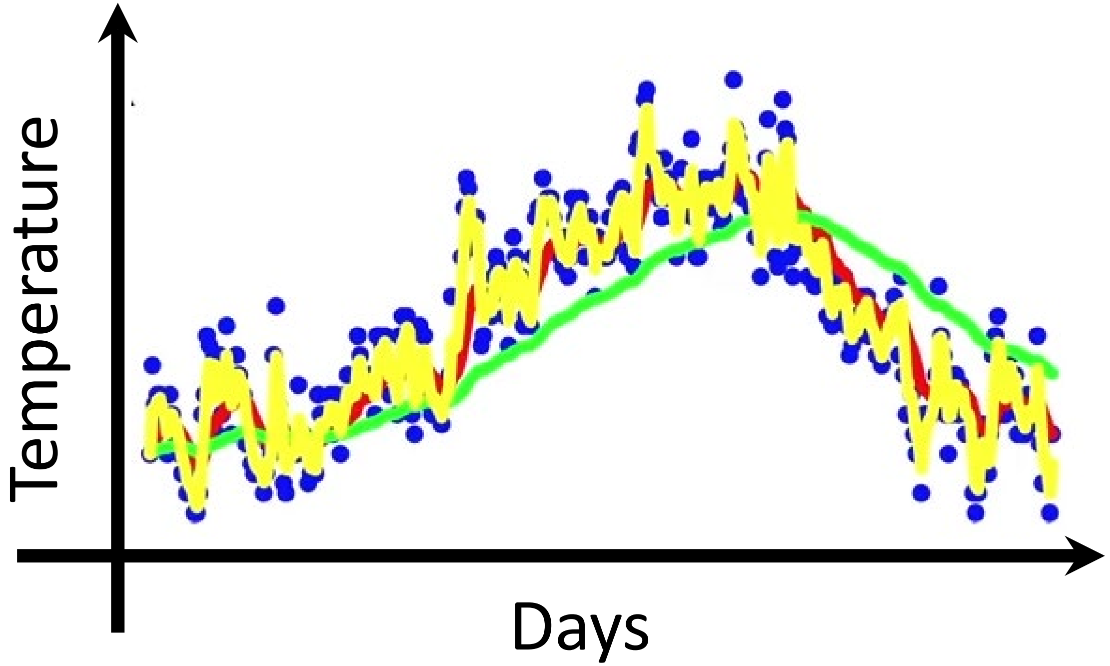
V_t is approximate average over $\approx \frac{1}{1-\beta}$ days

So,

β = 0.9 is closer to 10 days temperature

β = 0.98 is closer to 50 days temperature

β = 0.5 is closer to 2 days temperature



Source and reference https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://medium.com/datadriveninvestor/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9

Exponentially Weighted Averages

What is Exponentially Weighted Averages doing?

$V_t = \beta V_{t-1} + (1-\beta)\ \theta_t$

For,

$V_{100} = 0.9\ V_{99} + 0.1\ \theta_{100}$

$V_{99} = 0.9\ V_{98} + 0.1\ \theta_{99}$

Substituting, V_{99}

$V_{100} = 0.1\ \theta_{100} + 0.9\ (0.9\ V_{98} + 0.1\ \theta_{99})$

$V_{100} = 0.1\ \theta_{100} + 0.9\ (\ 0.1\ \theta_{99} + 0.9\ (0.9\ V_{97} + 0.1\ V_{98}))$ ..

Source and reference https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://medium.com/datadriveninvestor/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9

Optimizers – SGD with Momentum

- *"Compute the Exponentially weighted average of the gradients and use that gradient to update weights"* - ***Andrew NG***
- One of the most popular algorithms
- Helps to accelerate the gradient vectors in right direction and reduces oscillation
- Always faster than the SGD

Source and reference https://www.youtube.com/watch?v=k8fTYJPd3_I

Optimizers – SGD with Momentum

Algorithm:

At iteration t:

Calculate dw and db on the current mini-batch

$$V_{dw} = \beta V_{dw} + (1 - \beta)\, dw \Rightarrow V_t = \beta V_{t-1} + (1-\beta)\ \theta_t$$

$$V_{db} = \beta V_{db} + (1 - \beta)\, db$$

Update w and b:

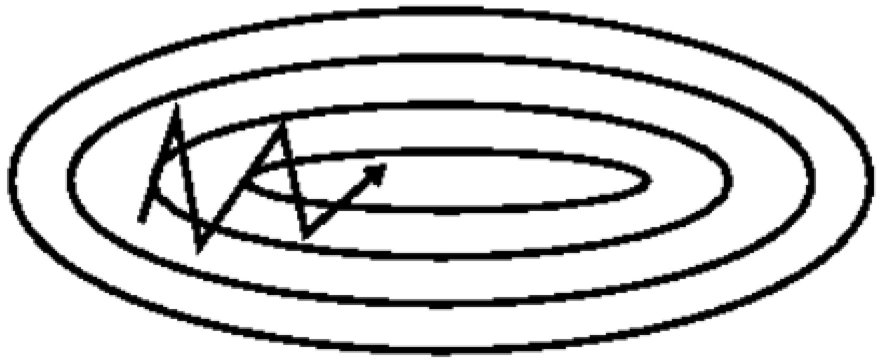
$$w = w - \alpha V_{dw},\ b = b - \alpha V_{db}$$

Hyper-parameters: α, β

Source and reference https://www.youtube.com/watch?v=k8fTYJPd3_I

Optimizers – SGD with Momentum

SGD Without Momentum



SGD With Momentum

Faster convergence and reduced oscillation

Image Source and reference http://ruder.io/optimizing-gradient-descent/index.html#momentum

Optimizers – RMSProp

- Root Mean Square Propagation
- Unpublished adaptive learning method by Geoffery Hinton
- RMSProp also reduces oscillation but in a different way than Momentum
- RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients.

Source and reference https://www.youtube.com/watch?v=k8fTYJPd3_I

Optimizers – RMSProp

Algorithm:

At iteration t:

Calculate dw and db on the current mini-batch

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2)\, dw^2$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2)\, db^2$$

Squaring the derivatives

Update w and b:

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}}}, \quad b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Square root of derivatives

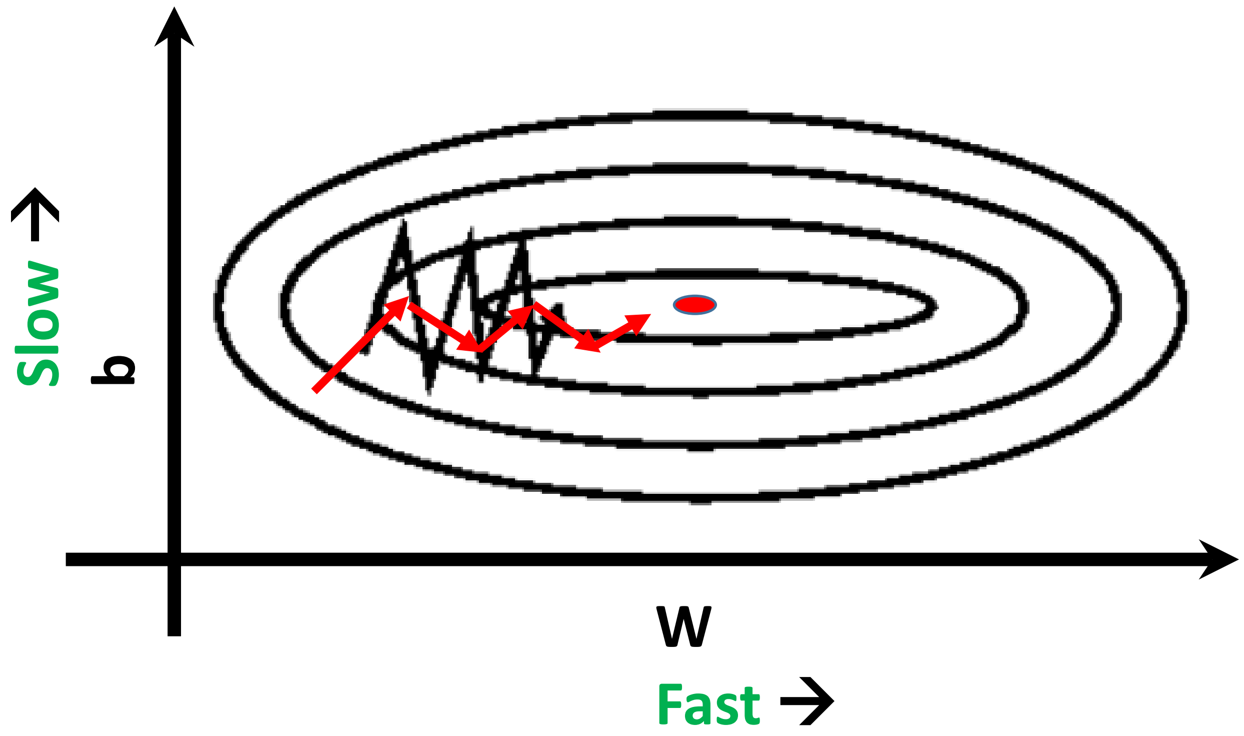
Source and reference https://www.youtube.com/watch?v=k8fTYJPd3_I

Optimizers – RMSProp

Intuition:

$S_{dw} \rightarrow$ Smaller number expected

$S_{db} \rightarrow$ Larger number expected



So,

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}}}, \quad b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Smaller number
So, w is larger

Larger number
So, b is small

In Practice add ε :

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}, \quad b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$

$\varepsilon \rightarrow$ small number, 10^{-8}

Source and reference https://www.youtube.com/watch?v=k8fTYJPd3_I

Optimizers – Adam

- Adam → Adaptive Moment Estimation
- Combination of RMSProp and Momentum
- Work well for a wide range of deep learning architecture

Source and reference: https://www.youtube.com/watch?v=JXQT_vxqwIs

Optimizers – Adam

Algorithm:

Initialize $V_{dw} = 0$, $V_{db} = 0$, $S_{dw} = 0$, $S_{db} = 0$

At iteration t:

Calculate $dw\ and\ db$ on the current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1)\, dw$, $V_{db} = \beta_1 V_{db} + (1 - \beta_1)\, db$ ← From Momentum, β_1

$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2)\, dw^2$, $S_{db} = \beta_2 S_{db} + (1 - \beta_2)\, db^2$ ← From RMSProp, β_2

Update w and b:

$$w = w - \alpha \frac{V_{dw}}{\sqrt{S_{dw}} + \varepsilon}, \quad b = b - \alpha \frac{V_{db}}{\sqrt{S_{db}} + \varepsilon}$$

Source and reference: https://www.youtube.com/watch?v=JXQT_vxqwIs

Optimizers – Adam

In practice: Bias correction is required as $V_{dw}, V_{db}, S_{dw}, S_{db}$ are initialized to 0 and are biased towards zero. Hence, a bias correction is required as follows:

$$V'_{dw} = \frac{V_{dw}}{(1 - \beta_1)}, \quad V'_{db} = \frac{V_{db}}{(1 - \beta_1)}$$

$$S'_{dw} = \frac{S_{dw}}{(1 - \beta_2)}, \quad S'_{db} = \frac{S_{db}}{(1 - \beta_2)}$$

Update w and b:

$$w = w - \alpha \frac{V'_{dw}}{\sqrt{S'_{dw}} + \varepsilon}, \quad b = b - \alpha \frac{V'_{db}}{\sqrt{S'_{db}} + \varepsilon}$$

Source and reference: https://www.youtube.com/watch?v=JXQT_vxqwIs

Optimizers – Adam

Hyper parameter guide:

α (Learning rate)→ should be tunned, start with 0.001

β_1 (Momentum term) → 0.9 (dw)

β_2 (moving weighted average) → 0.999 (dw^2)

ε → 10^{-8}

Optimization Demo:

https://vis.ensmallen.org/

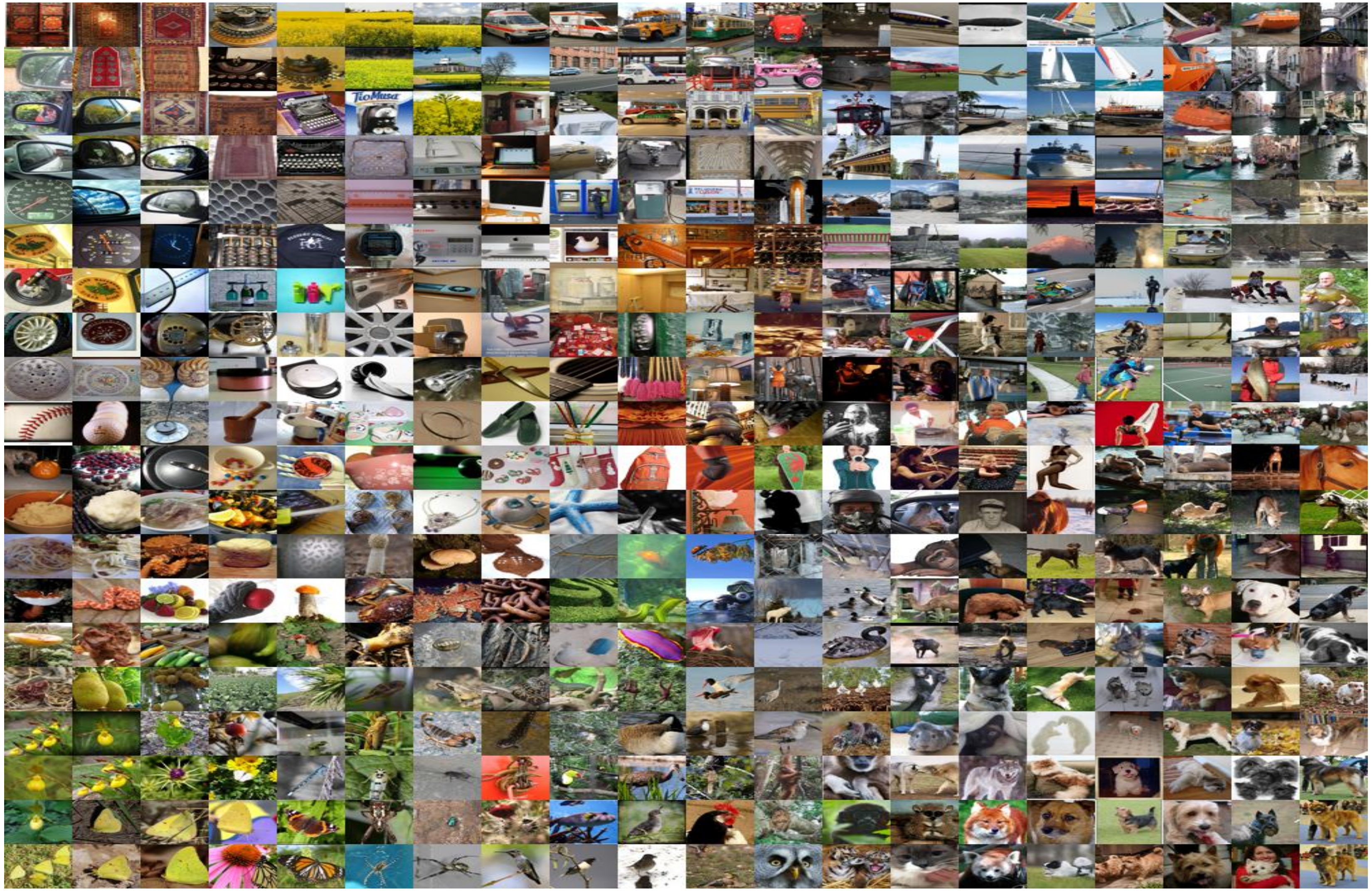
Source and reference: https://www.youtube.com/watch?v=JXQT_vxqwIs

ImageNet Dataset:

- 15+ million labelled high-resolution images
- 22000 categories
- ILSVRC (**Large Scale Visual Recognition Challenge**) used a subset of ImageNet:
  - ~1000 images per category
  - 1000 categories
  - Train: 1.2 million images
  - Validation: 50k images
  - Test : 150k images

Source and reference: http://www.image-net.org/
http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

ImageNet Dataset:



Source and reference https://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_1k.jpg

ImageNet Dataset Results:

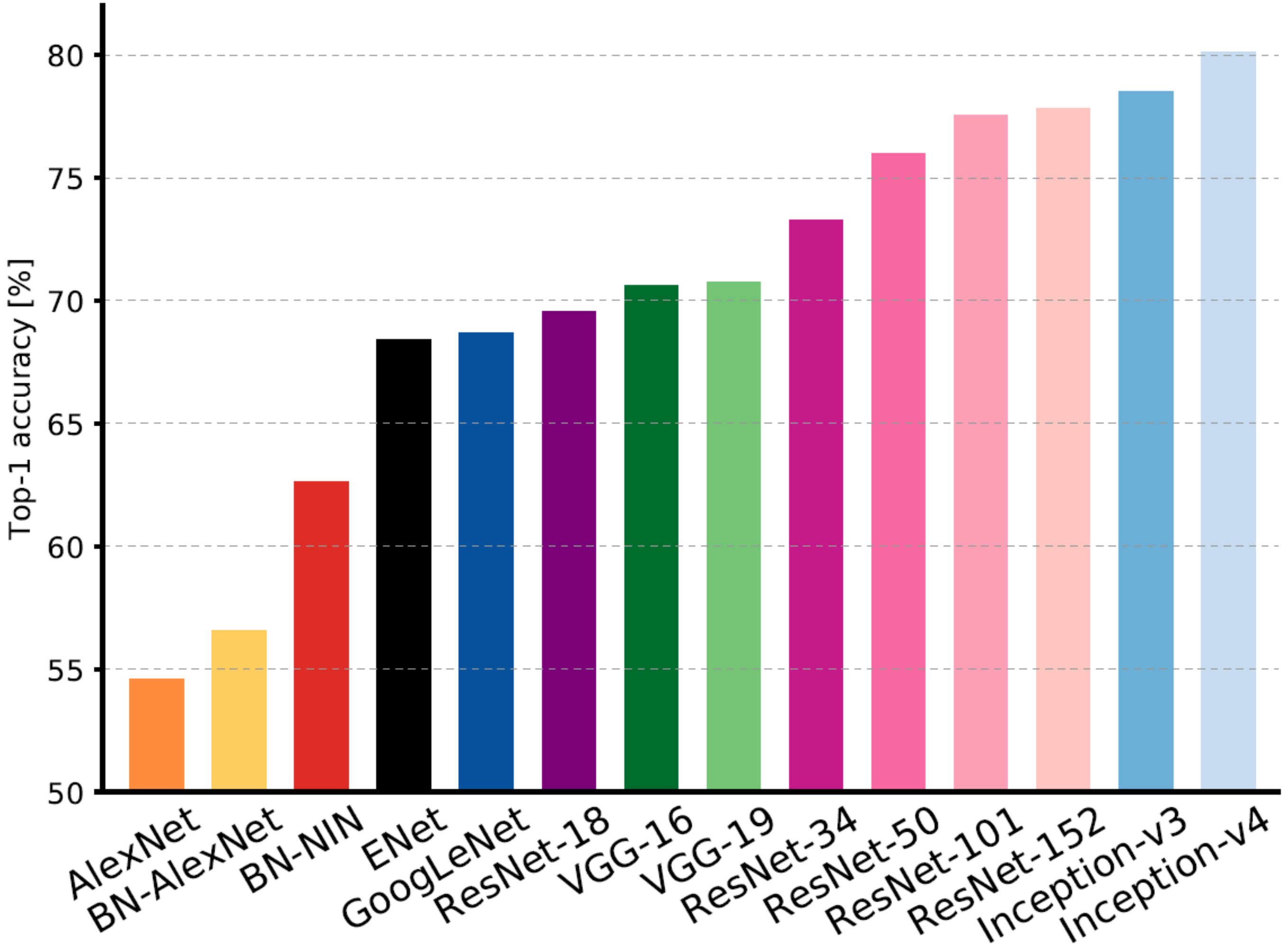


Image Source and reference: An Analysis of Deep Neural Network Models for Practical Applications, 2017, https://arxiv.org/abs/1605.07678

Transfer Learning

- Knowledge acquired while solving one task, can be used to solve related tasks.
- Example:
  - You know how to ride a Bi-cycle → You can learn how to ride a Motorbike
  - You know how to use a Tablet → You can easily learn how to use a Laptop/desktop
- Similar to the way humans apply knowledge acquired from one task to solve a new but similar/related task.
- We learned how to read in Year-1 in literacy class. Reading skills acquired in the literacy classes made it easy to understand Physics in Year-9.

Source and reference: https://cs231n.github.io/transfer-learning/
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning Benefits

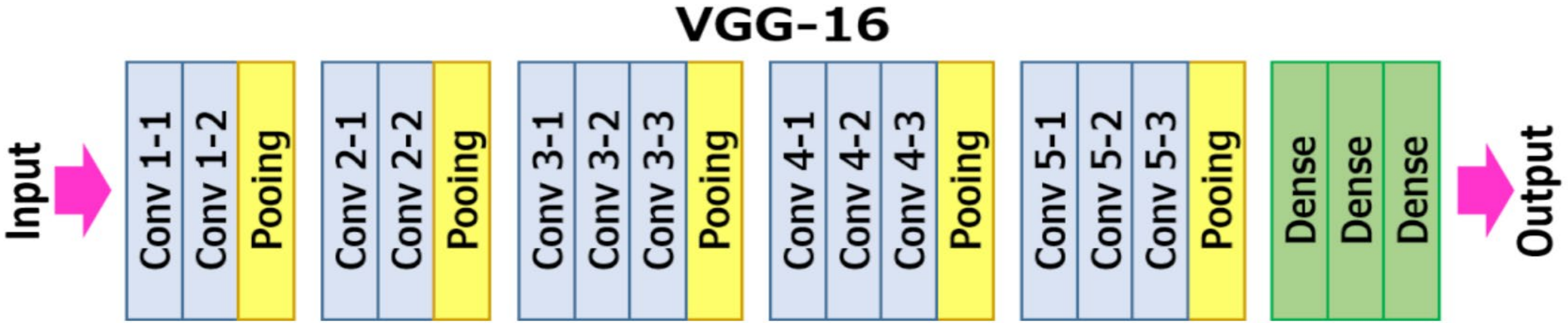
1. **Less training data required:** Don’t have enough data to train a Deep Learning model from scratch. Model trained using a large (similar) dataset can be used.

2. **Faster training** : Training can converge faster, due the use to existing knowledge (weights) to start with rather than from scratch.

3. **Better model generalization**: Model is trained to identify features which can be applied to new contexts.

Source and reference: https://cs231n.github.io/transfer-learning/
https://missinglink.ai/guides/neural-network-concepts/transfer-learning-overview/

Transfer Learning Strategies



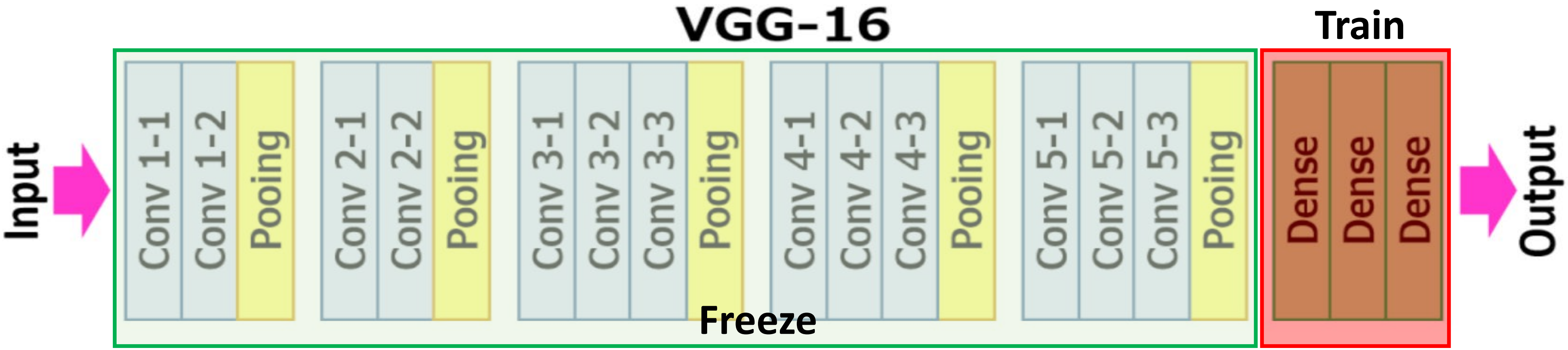
Option-1: (VGG-16 considered as an example)

Use pre-trained (ImageNet) model for prediction, without any training.

→Useful when your dataset distribution is similar to ImageNet, with small number of samples.

Source and reference: https://cs231n.github.io/transfer-learning/
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning Strategies



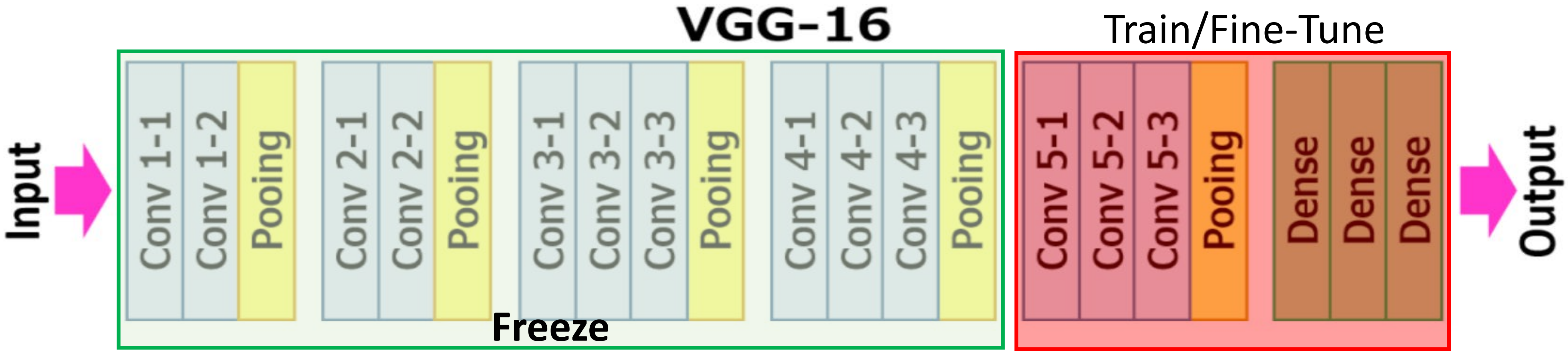
Option-2: **(VGG-16 considered as an example)**

Train Full-Connected layer, Use CONV layers for feature extraction

→Useful when your dataset distribution is similar to ImageNet (or original dataset), but number of classes are different and your dataset is small.

Source and reference: https://cs231n.github.io/transfer-learning/
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning Strategies



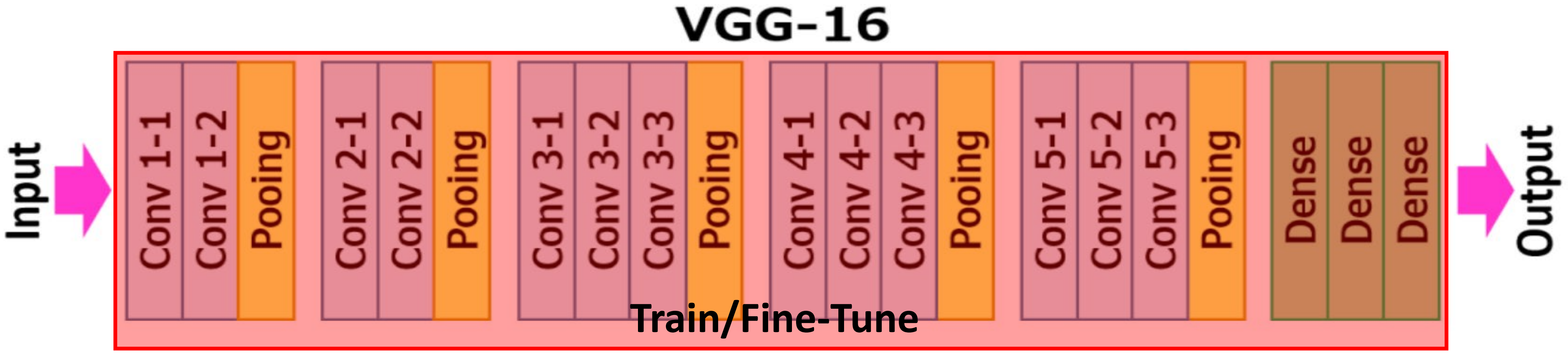
Option-3: **(VGG-16 considered as an example)**

Partially Train CONV layers (usually last layer(s) which have specialised features) + Full Connection (FC) layer (with modifications)

→Useful when your dataset distribution is not similar to ImageNet (or original dataset), number of classes are different and your dataset is small.

Source and reference: https://cs231n.github.io/transfer-learning/
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning Strategies



<u>Option-4:</u> (VGG-16 considered as an example)

Train all the CONV layers + Full Connection (FC) layer (with modifications)

→Useful when your dataset distribution is not similar to ImageNet, number of classes are different, your is dataset large and the task is complex.

Source and reference: https://cs231n.github.io/transfer-learning/
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Classic CNN Architectures

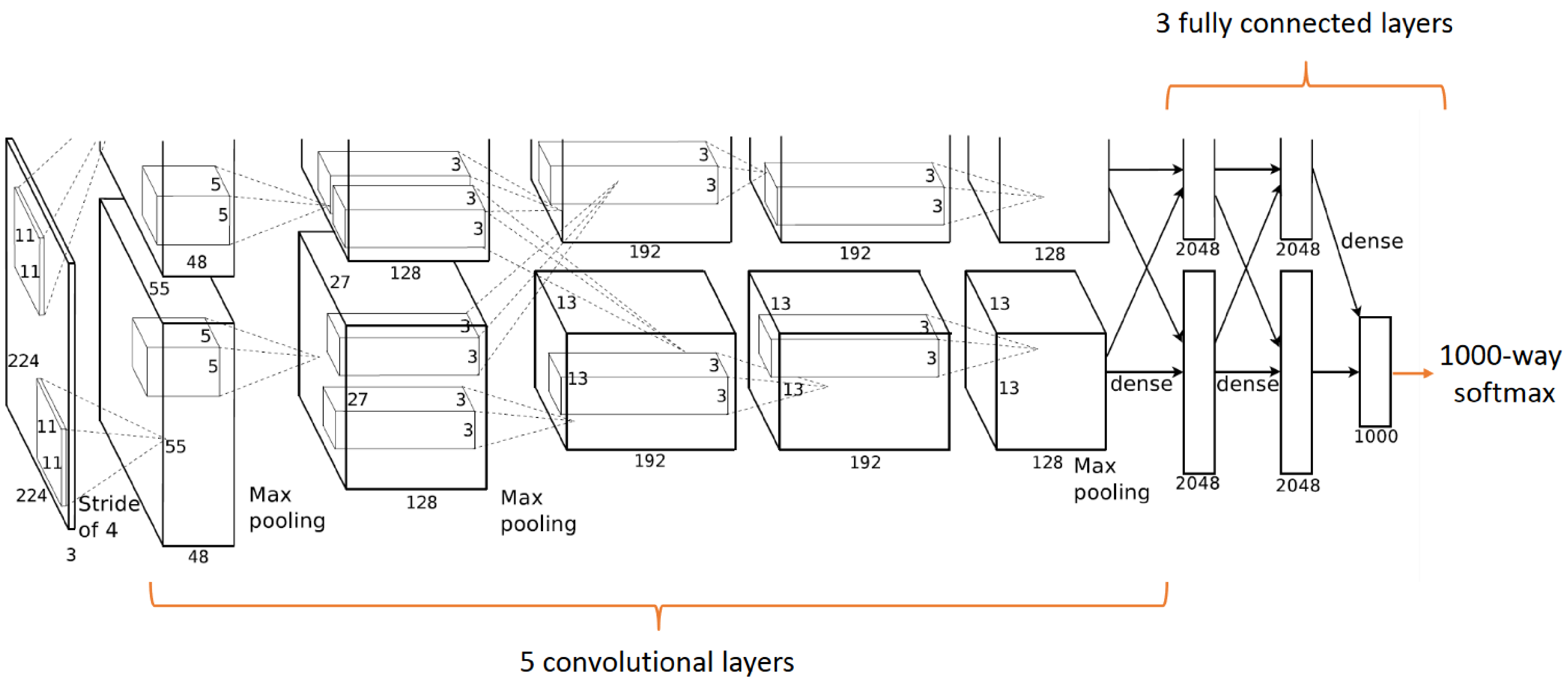
Case Study: AlexNet

- Similar architecture as LeNet by Yann LeCunn et al. but deeper with more layers
- Simple architecture:
  - CONV : 5 layers
  - FC: 3 layer
  - Max pooling
  - Dropout

- Accuracy: top-5 test error rate of 15.3%
- Winner of ILSVRC 2012!
- First CNN to be successful on a very big dataset!

Source and reference: http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

Case Study: AlexNet



Source and reference: http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

Case Study: AlexNet

Input: 224x224x3 image

CONV1 → CONV2 → CONV3 → CONV4 → CONV5 → FC1 → FC2 → FC3

CONV1	CONV2	CONV3	CONV4	CONV5	FC1	FC2	FC3
Filters: 96 **Dim**: 11x11 **Stride**: **4** **Pad**: 0	**Filters**: 256 **Dim**: 5x5 **Stride**: **1** **Pad**: 2	**Filters**: 384 **Dim**: 3x3 **Stride**: **1** **Pad: 1**	**Filters**: 384 **Dim**: 3x3 **Stride**: **1** **Pad: 1**	**Filters**: 256 **Dim**: 3x3 **Stride**: **1** **Pad: 1**	**4096 Neuron**	**4096 Neuron**	**1000 Neuron**

Activations: Relu after each CONV and FC layer

Optimizer: SGD with Momentum

Regularization: Dropout in FC1 and FC2

Total Trainable parameter: ~60Million

Training settings: 2 X Nvidia GTX 580 3GB GPUs for 5-6days!

Source and reference: http://cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

Case Study: GoogleNet/Inception(2014)

- Accuracy: top-5 test error rate of 6.7%
- Close to human level performance
- Winner of ILSVRC 2014!
- 22 layer Deep CNN
- Number of trainable parameters: 4 Million (Alexnet ~ 60M), Significantly reduced
- A novel inception module was introduced.
- Optimizer: RMSProp

Source and reference: https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

Case Study: GoogleNet/Inception(2014)

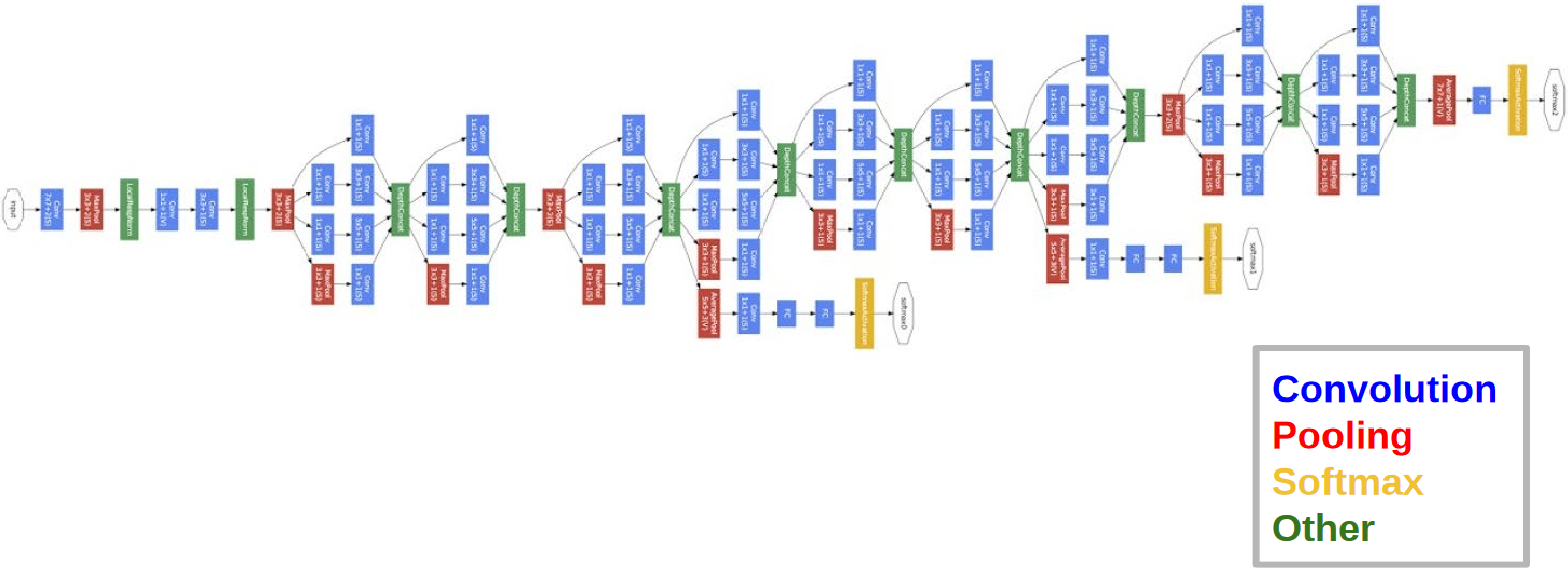
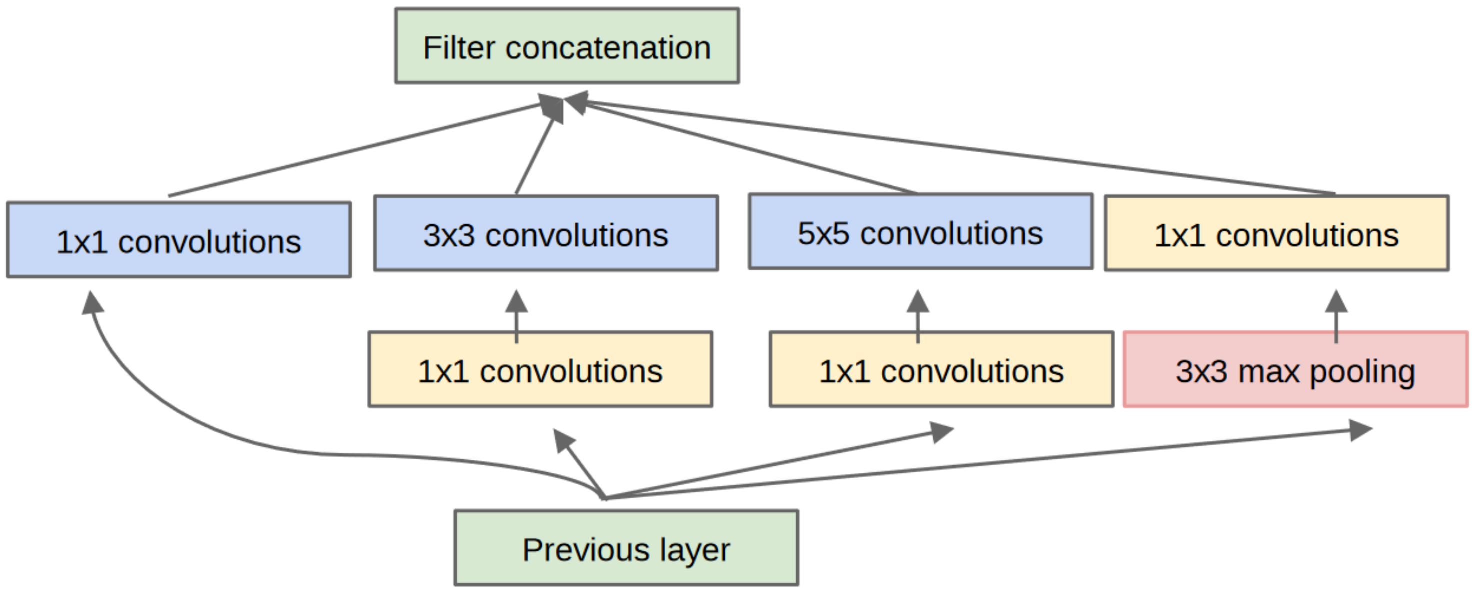


Image Source and reference: https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

Case Study: GoogleNet/Inception(2014)

Inception Module



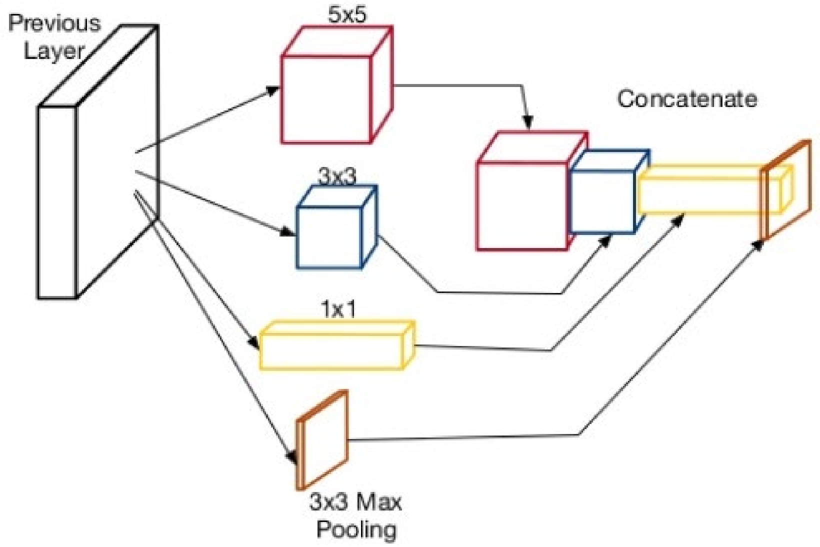


Image Source and reference: https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf