THE UNIVERSITY OF NEW SOUTH WALES

Lecture 4b. Image Processing

Never Stand Still

Faculty of Engineering

COMP9444 Week 4b

Sonit Singh

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales, Sydney, Australia

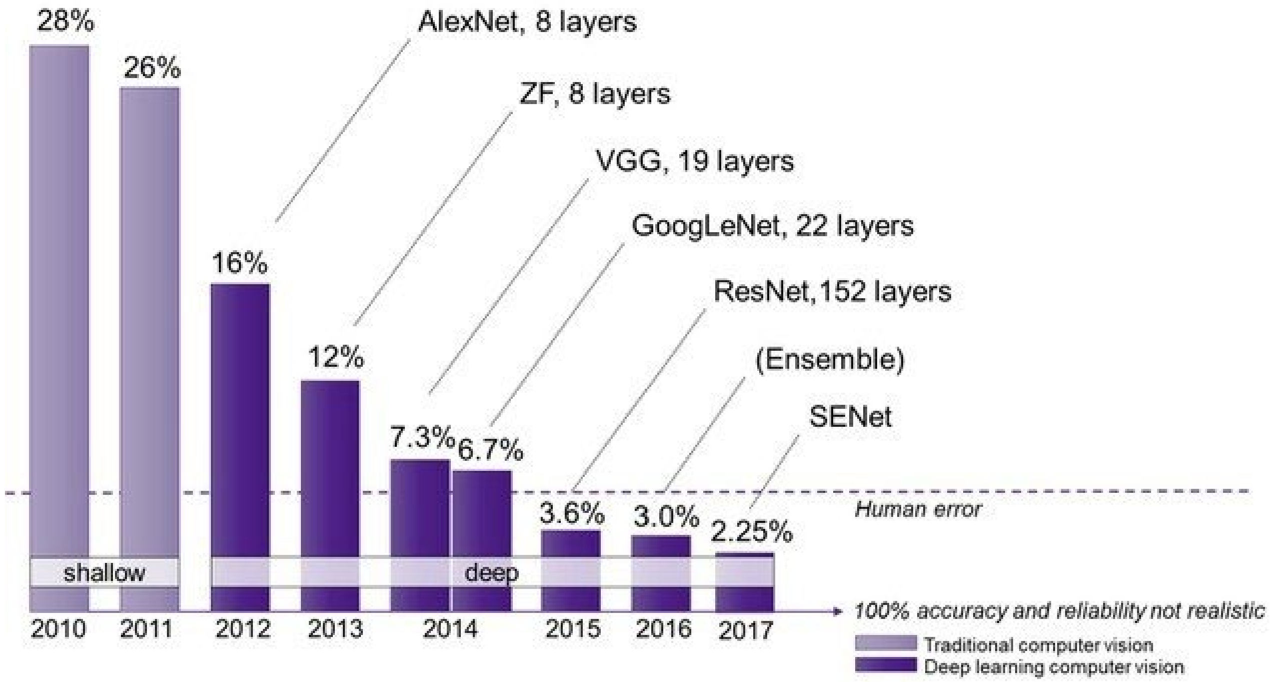
sonit.singh@unsw.edu.au

Agenda

- Convolutional Neural Networks
- Why training Deep Neural Networks is hard?
- DNN training strategy
- Transfer Learning
- Overfitting and Underfitting
- Methods to avoid overfitting
  - Data Augmentation
  - Regularization
- Data Preprocessing
- Batch Normalization
- Choice of optimizers
- Tuning DNNs hyperparameters
- Neural Style Transfer

Convolutional Neural Networks (CNNs)

- A class of deep neural networks suitable for processing 2D/3D data. For e.g., Images and Videos
- CNNs can capture high-level representation of images/videos which can be used for end-tasks such as classification, object detection, segmentation, etc.
- A range of CNNs improving over the years



Source: Convolutional Neural Networks. https://medium.com/@rajat.k.91/convolutional-neural-networks-why-what-and-how-f8f6dbebb2f9

History

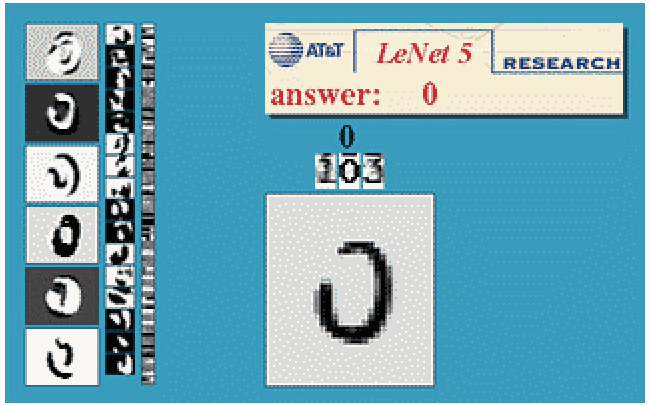
- ImageNet (2009)
  - Consists of 14 million images, more than 21,000 classes, and about 1 million images have bounding box annotations
  - Annotated by humans using crowdsourcing platform “Amazon Mechanical Turk”

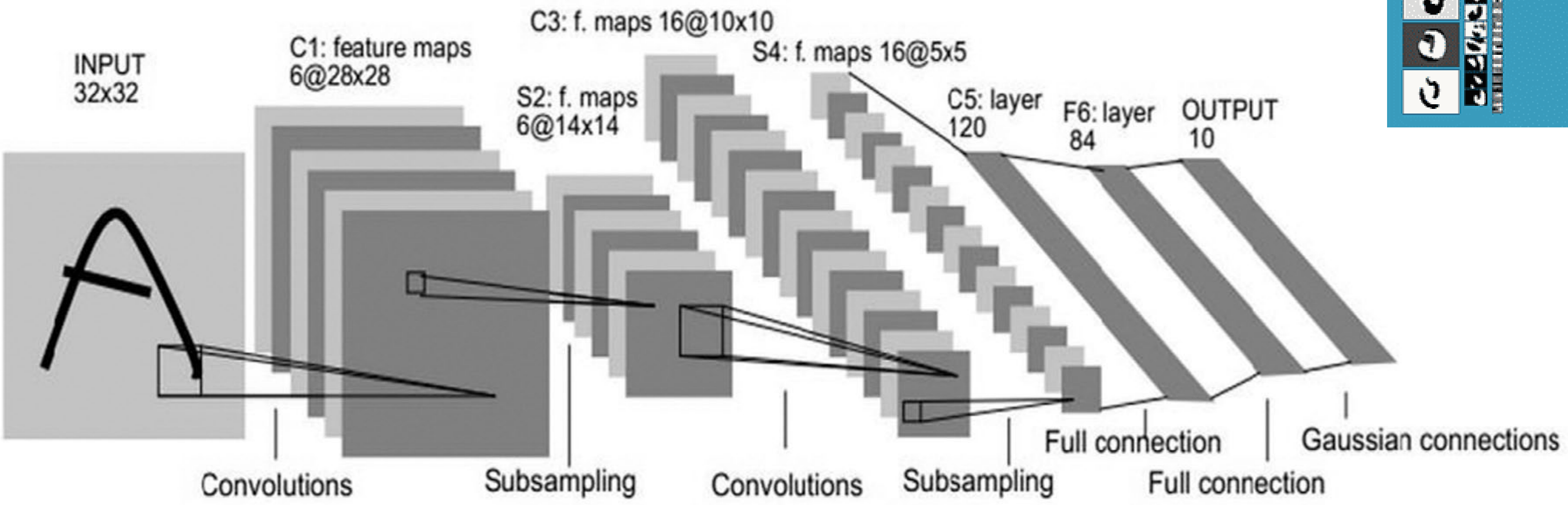
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)
  - annual competition to foster the development and benchmarking of state-of-the-art algorithms in Computer Vision
  - Led to improvement in architectures and techniques at the intersection of CV and DL

Image Credit: Synced. https://syncedreview.com/2020/06/23/google-deepmind-researchers-revamp-imagenet/

LeNet

- First developed by Yann Lecun in 1989 for digit recognition
  - First time backprop is used to automatically learn visual features
  - Two convolutional layers, three fully connected layers (32 x 32 input, 6 and 12 FMs, 5 x 5 filters)
  - Stride = 2 is used to reduce image dimensions
  - Scaled tanh activation function
  - Uniform random weight initialization





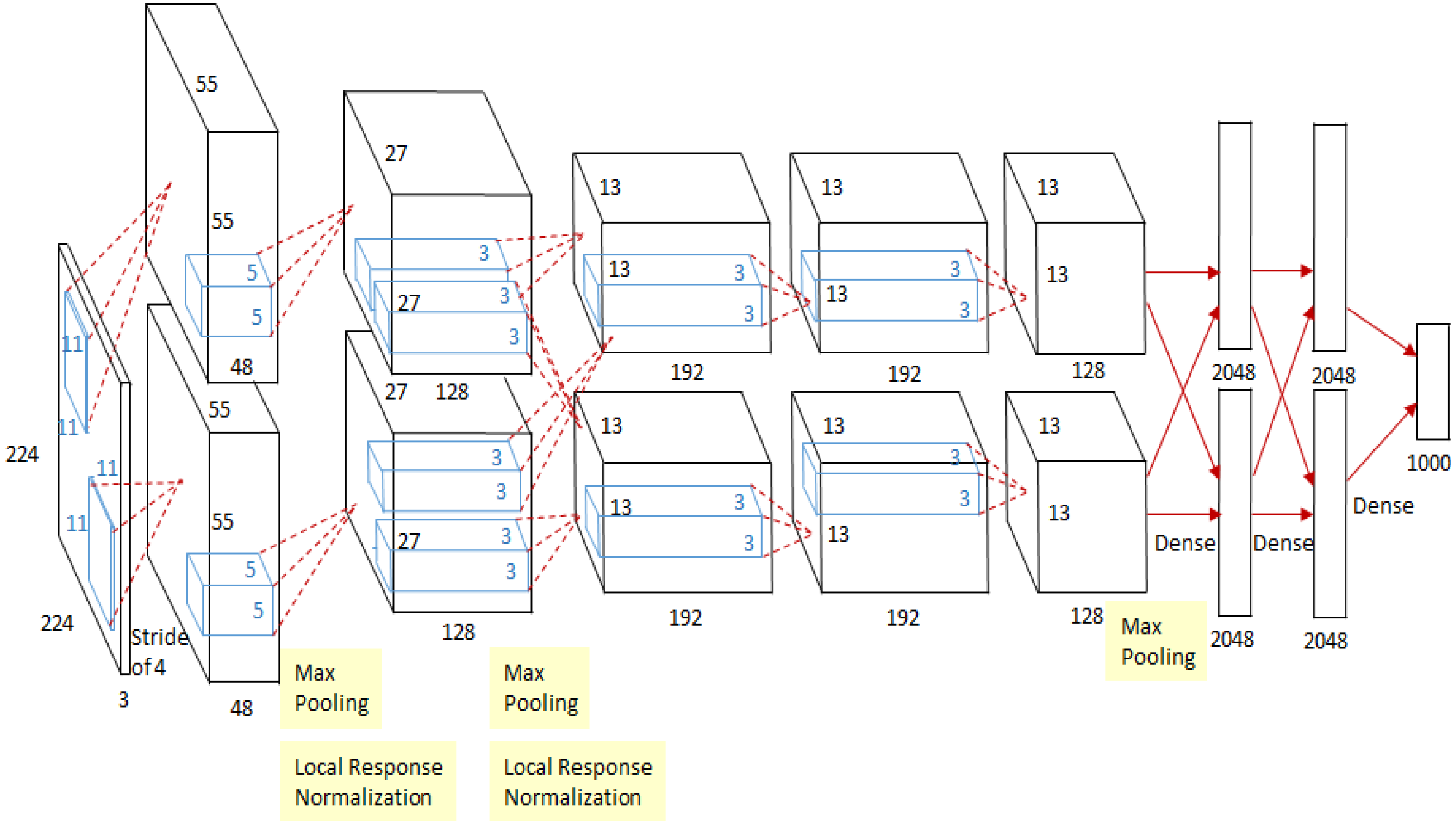
Source: Lecun et al. (1989). Gradient-based learning applied to document recognition.

CNN Architectures

- AlexNet, 8 layers (2012)
- VGG, 19 layers (2014)
- GoogleNet, 22 layers (2014)
- ResNets, 152 layers (2015)
- DenseNet, 201 layers (2017)
- EfficientNet (2019)
- EfficientNetV2 (2021)

AlexNet

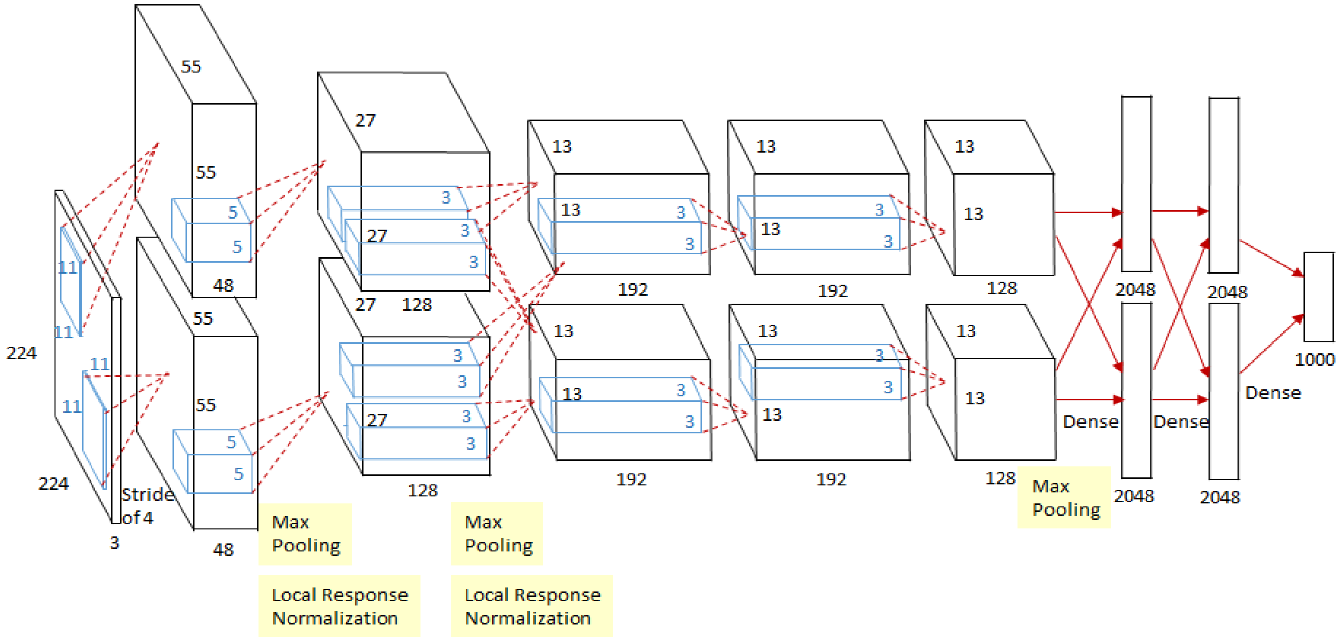
- ➢ 650K neurons
- ➢ 630M connections
- ➢ 60M parameters



- ➢ more parameters than images -------> danger of overfitting

Enhancements

- Rectified Linear Units (ReLUs)
- Overlapping pooling (Width = 3, stride = 2)
- Stochastic gradient descent with momentum and weight decay
- Data augmentation to reduce overfitting
- 50% dropout in the fully connected layers



Dealing with Deep Networks

- > 10 layers
  - weight initialization
  - batch normalization

- > 30 layers
  - skip connections

- > 100 layers
  - identity skip connections

Statistics Example: Coin Tossing

Example: Toss a coin once, and count the number of Heads

$$\begin{aligned}
\text{Mean} \quad \mu \quad &= \tfrac{1}{2}(0+1) = 0.5 \\
\text{Variance} \quad &= \tfrac{1}{2}\left((0-0.5)^2 + (1-0.5)^2\right) = 0.25 \\
\text{Standard Deviation} \quad \sigma \quad &= \sqrt{\text{Variance}} = 0.5
\end{aligned}$$

Example: Toss a coin 100 times, and count the number of Heads

$$\begin{aligned}
\text{Mean} \quad \mu \quad &= 100 * 0.5 = 50 \\
\text{Variance} \quad &= 100 * 0.25 = 25 \\
\text{Standard Deviation} \quad \sigma \quad &= \sqrt{\text{Variance}} = 5
\end{aligned}$$

Example: Toss a coin 10000 times, and count the number of Heads

$$\mu = 5000, \qquad \sigma = \sqrt{2500} = 50$$

Statistics

The mean and variance of a set of n samples $x_1, \ldots, x_n$ are given by

$$\text{Mean}[x] = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$\text{Var}[x] = \frac{1}{n}\sum_{k=1}^{n} (x_k - \text{Mean}[x])^2 = \left(\frac{1}{n}\sum_{k=1}^{n} x_k^2\right) - \text{Mean}[x]^2$$

If w_k, x_k are independent and $y = \sum_{k=1}^{n} w_k x_k$ then

$$\text{Var}[y] = n\,\text{Var}[w]\text{Var}[x]$$

Weight Initialization

Consider one layer (i) of a deep neural network with weights $w_{jk}^{(i)}$ connecting the activations $\{x_k^{(i)}\}_{1\leq k\leq n_i}$ at the previous layer to $\{x_j^{(i+1)}\}_{1\leq j\leq n_{i+1}}$ at the next layer, where $g()$ is the transfer function and

$$x_j^{(i+1)} = g(\text{sum}_j^{(i)}) = g\left(\sum_{k=1}^{n_i} w_{jk}^{(i)} x_k^{(i)}\right)$$

Then

$$\text{Var}[\,\text{sum}^{(i)}\,] = \; n_i \text{Var}[w^{(i)}]\text{Var}[x^{(i)}]$$

$$\text{Var}[x^{(i+1)}] \simeq G_0\, n_i \text{Var}[w^{(i)}]\text{Var}[x^{(i)}]$$

Where G_0 is a constant whose value is estimated to take account of the transfer function.

If some layers are not fully connected, we replace n_i with the average number n_i^{in} of incoming connections to each node at layer $i+1$.

Weight Initialization

If the nework has D layers, with input $x = x^{(1)}$ and output $z = x^{(D+1)}$, then

$$\mathrm{Var}[z] \simeq \Big(\prod_{i=1}^{D} G_0\, n_i^{\mathrm{in}}\, \mathrm{Var}[w^{(i)}]\Big)\mathrm{Var}[x]$$

When we apply gradient descent through backpropagation, the differentials will follow a similar pattern:

$$\mathrm{Var}[\frac{\partial}{\partial x}] \simeq \Big(\prod_{i=1}^{D} G_1\, n_i^{\mathrm{out}}\, \mathrm{Var}[w^{(i)}]\Big)\mathrm{Var}[\frac{\partial}{\partial z}]$$

In this equation, n_i^{out} is the average number of outgoing connections for each node at layer i, and G_1 is meant to estimate the average value of the derivative of the transfer function.

For Rectified Linear Units, we can assume $G_0 = G_1 = \frac{1}{2}$

Weight Initialization

In order to have healthy forward and backward propagation, each term in the product must be approximately equal to 1. Any deviation from this could cause the activations to either vanish or saturate, and the differentials to either decay or explode exponentially.

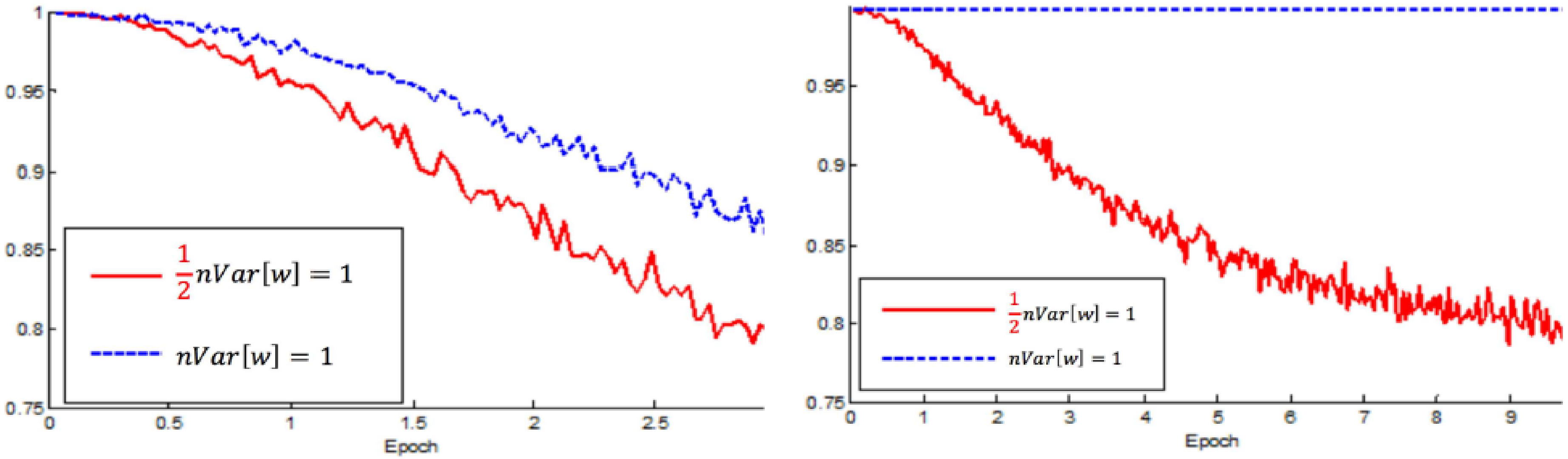
$$\mathrm{Var}[z] \simeq \left(\prod_{i=1}^{D} G_0\, n_i^{\mathrm{in}} \mathrm{Var}[w^{(i)}] \right) \mathrm{Var}[x]$$

$$\mathrm{Var}[\frac{\partial}{\partial x}] \simeq \left(\prod_{i=1}^{D} G_1\, n_i^{\mathrm{out}} \mathrm{Var}[w^{(i)}] \right) \mathrm{Var}[\frac{\partial}{\partial z}]$$

We therefore choose the initial weights $\{w_{jk}^{(i)}\}$ in each layer (i) such that

$$G_1 n_i^{\mathrm{out}} \mathrm{Var}[w^{(i)}] = 1$$

Weight Initialization



- 22-layer ReLU network (left),
  $\mathrm{Var}[w] = \frac{2}{n}$ converges faster than $\mathrm{Var}[w] = \frac{1}{n}$
- 30-layer ReLU network (right),
  $\mathrm{Var}[w] = \frac{2}{n}$ is successful while $\mathrm{Var}[w] = \frac{1}{n}$ fails to learn at all

Batch Normalization

We can normalize the activations $x_k^{(i)}$ of node k in layer (i) relative to the mean and variance of those activations, calculated over a mini-batch of training items:

$$\hat{x}_k^{(i)} = \frac{x_k^{(i)} - \mathrm{Mean}[x_k^{(i)}]}{\sqrt{\mathrm{Var}[x_k^{(i)}]}}$$

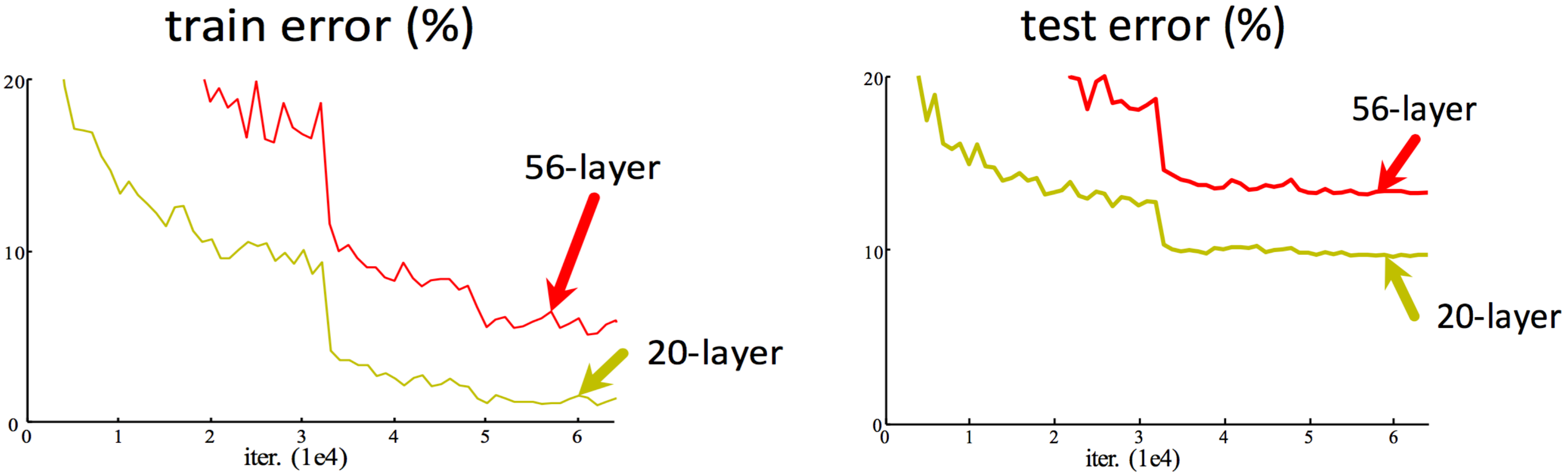
These activations can then be shifted and re-scaled to

$$y_k^{(i)} = \beta_k^{(i)} + \gamma_k^{(i)} \hat{x}_k^{(i)}$$

$\beta_k^{(i)}, \gamma_k^{(i)}$ are additional parameters, for each node, which are trained by backpropagation along with the other parameters (weights) in the network.

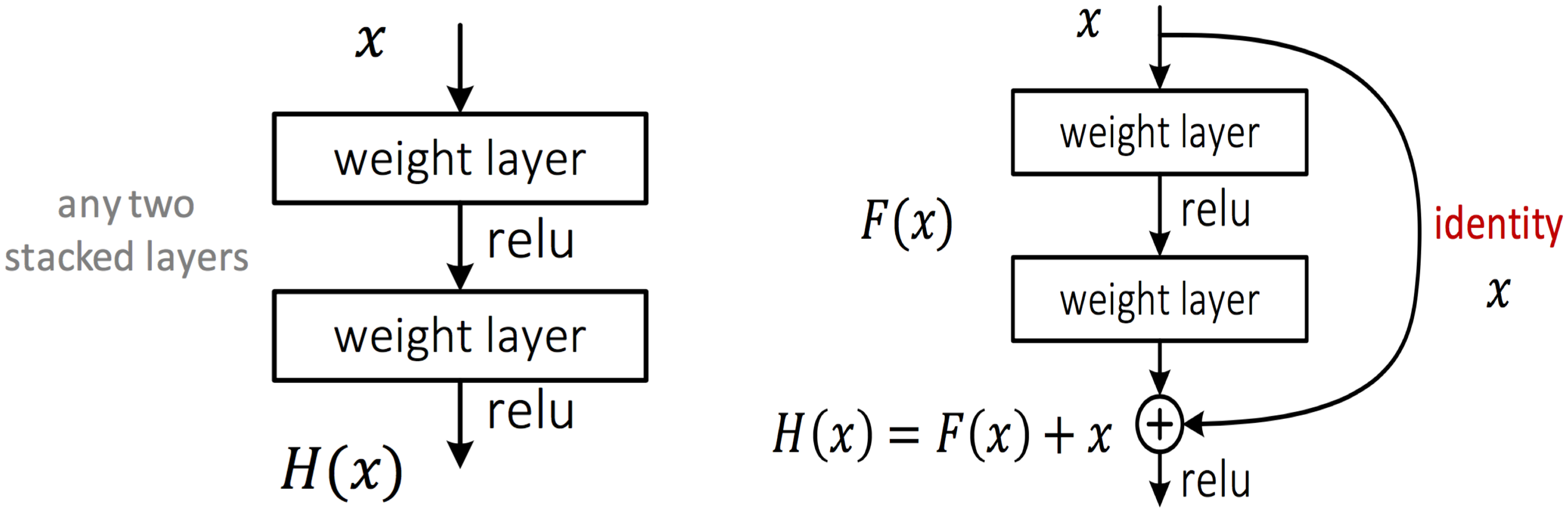
After training is complete, $\mathrm{Mean}[x_k^{(i)}]$ and $\mathrm{Var}[x_k^{(i)}]$ are either pre-computed on the entire training set, or updated using running averages.

Going Deeper



- If we simply stack additional layers, it can lead to higher training error as well as higher test error

Residual Networks

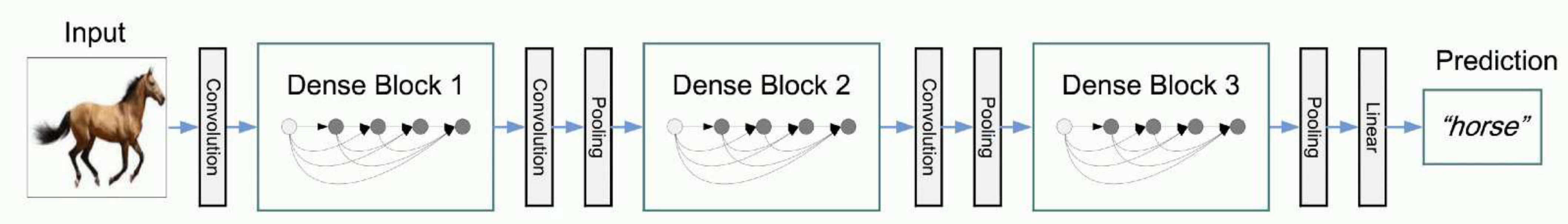


➢ **Idea**: Take any two consecutive stacked layers in a deep network and add a **"skip"** connection which bypasses these layers and is added to their output.

Residual Networks

- the preceding layers attempt to do the “whole” job, making *x* as close as possible to the target output of the entire network
- *F(x)* is a residual component which corrects the errors from previous layers, or provides additional details which the previous layers were not powerful enough to compute
- With skip connections, both training and test error drop as you add more layers
- With more than 100 layers, need to apply ReLU before adding the residual instead of afterwards. This is called an **identity skip connection**.

Dense Networks



- Good results have been achieved using networks with densely connected blocks, within which each layer is connected by shortcut connections to all the preceding layers.

VGG

- Developed at Visual Geometry Group (Oxford) by Simonyan and Zisserman
- 1st runner up (Classification) and Winner (localization) of ILSVRC 2014 competition
- VGG-16 comprises of 138 million parameters
- VGG-19 comprises of 144 million parameters

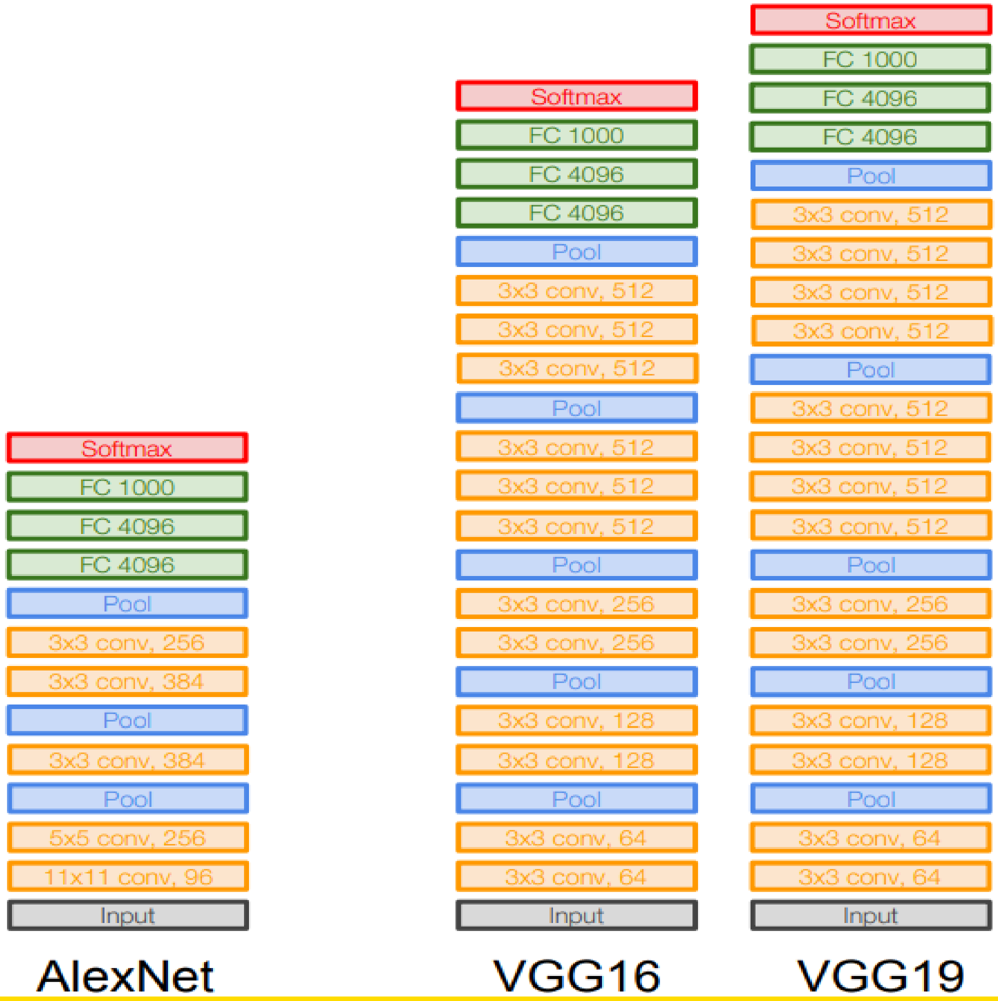
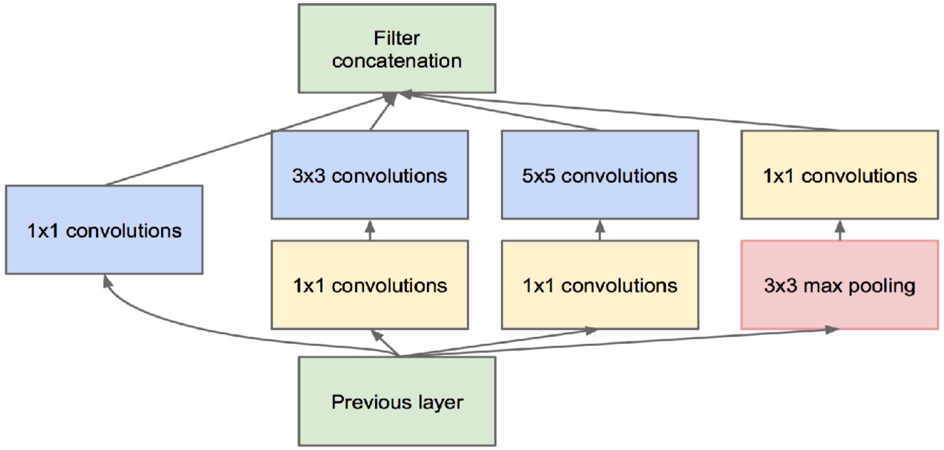
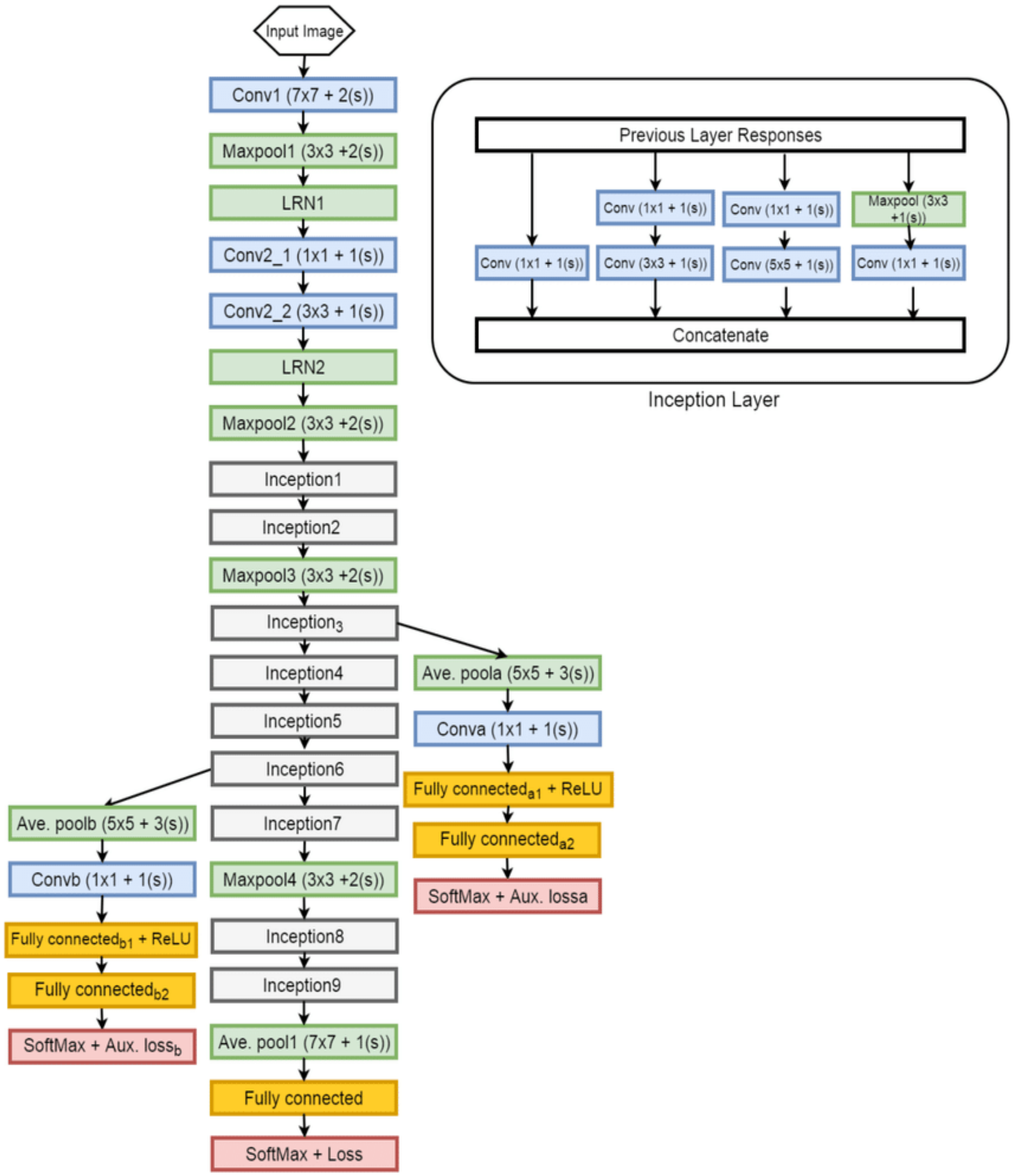


Image Credit: Medium. https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11

GoogLeNet

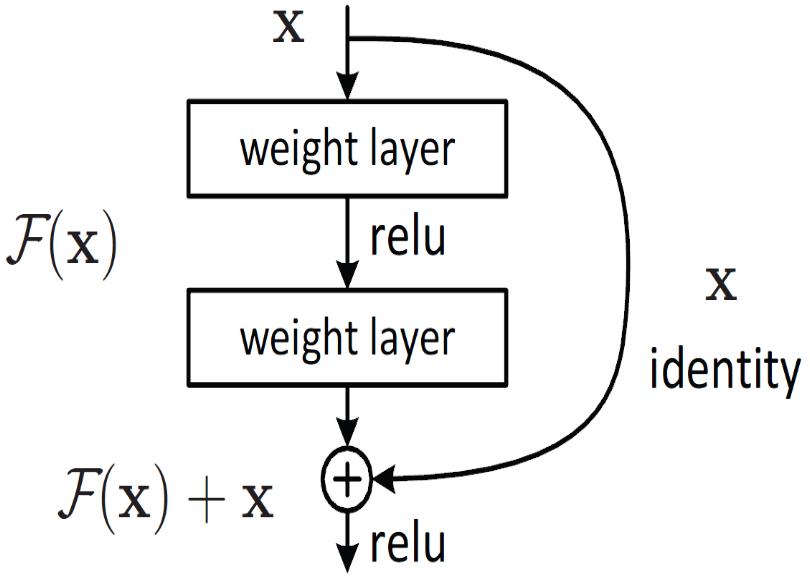
- ➢ A 22-layer CNN developed by researchers at Google
- ➢ Deeper networks prone to overfitting and suffer from exploding or vanishing gradient problem
- ➢ Core idea "**Inception module**"
- ➢ Adding **Auxiliary loss as an extra supervision**
- ➢ Winner of 2014 ILSVRC Challenge

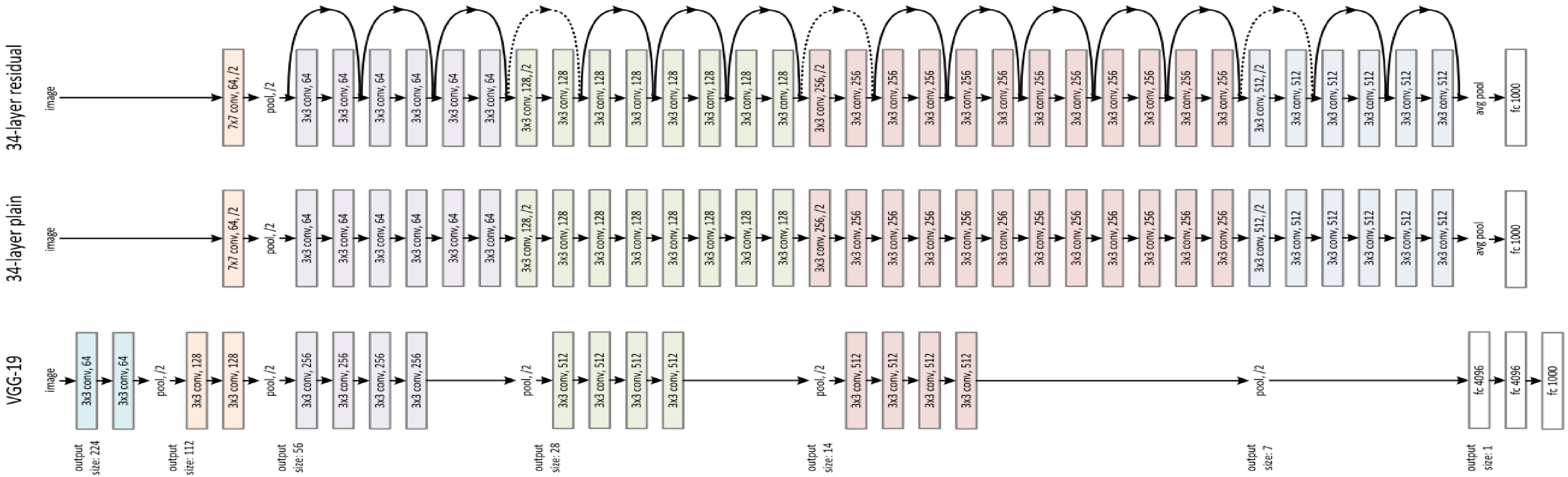




ResNet

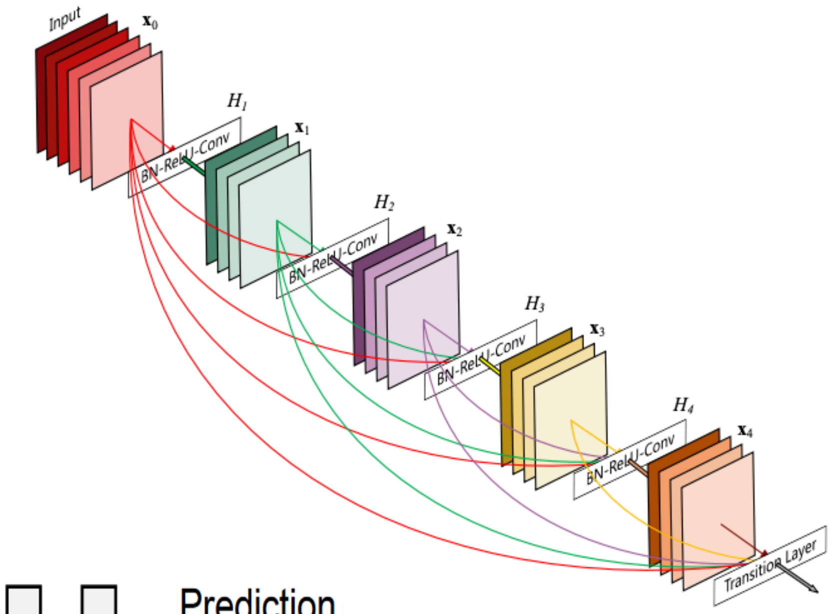
- Developed by researchers at Microsoft
- Core idea **“residual connections”** to preserve the gradient
- The identity matrix transmits forward the input data that avoids the loose of information (the data vanishing problem)

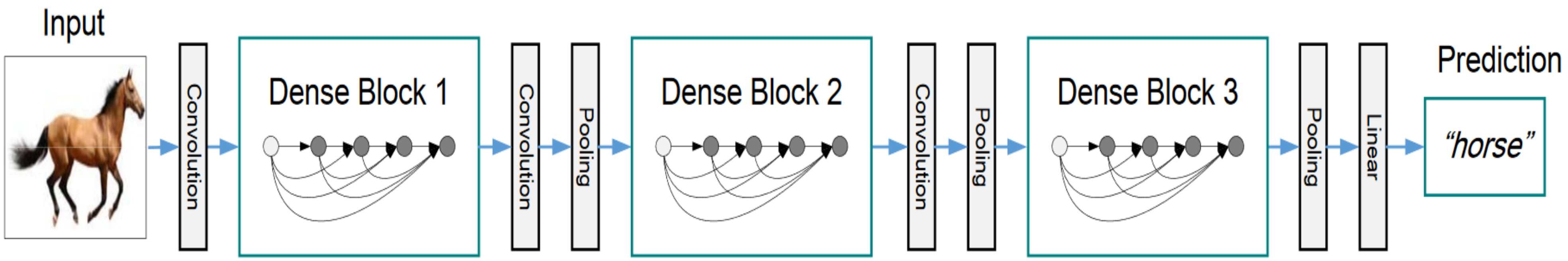




DenseNet

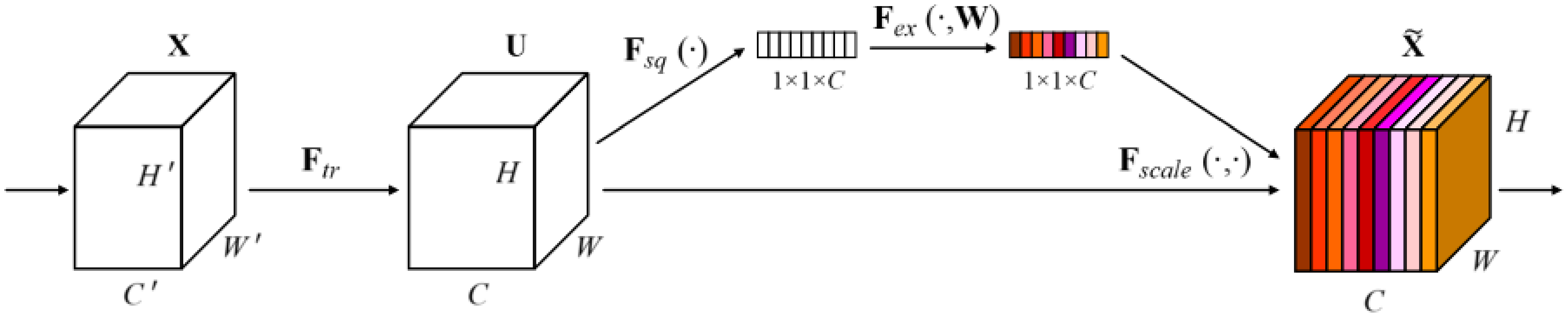
- In a **DenseNet** architecture, each layer is connected to every other layer, hence the name **Densely Connected Convolutional Network**
- For each layer, the feature maps of all the preceding layers are used as inputs, and its own feature maps are used as input for each subsequent layers
- DenseNets have several compelling advantages:
  - alleviate the vanishing-gradient problem
  - strengthen feature propagation
  - encourage feature reuse, and
  - substantially reduce the number of parameters.





SENet (Squeeze-and-Excitation Network)

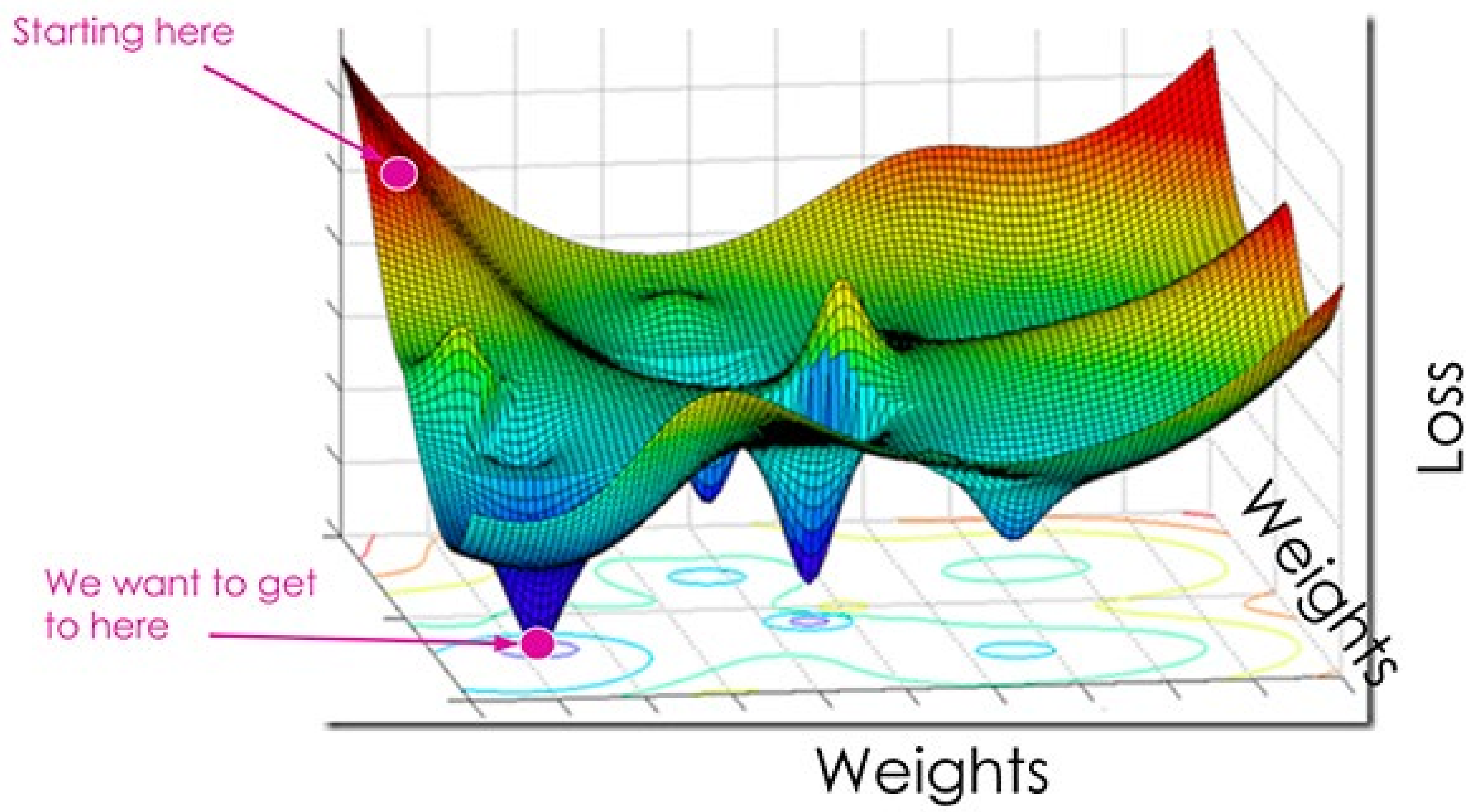
- CNNs fuse the spatial and channel information to extract features to solve the task
- Before this, networks weights each of its channels equally when creating the output feature maps
- SENets added a content aware mechanism to weight each channel adaptively
- SE block helps to improve representation power of the network, able to better map the channel dependency along with access to global information



Why training Deep Neural Networks is hard?

Credit: https://itechindia.co/blog/machine-learning-are-companies-in-india-ready-for-it/

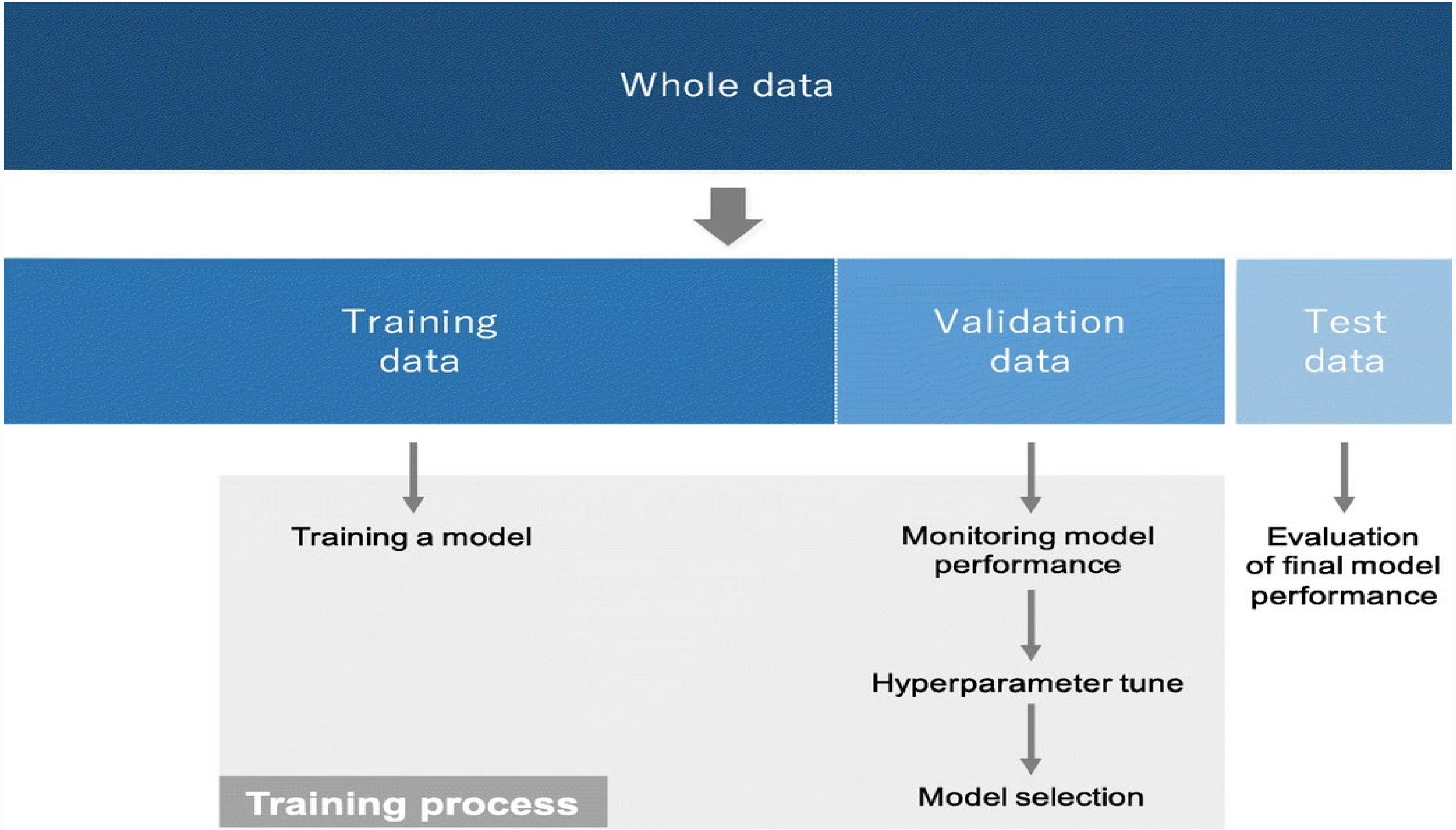
Why training Deep Neural Networks is hard?



Credit: Adrian Rosebrock, PyImageSearch, https://www.pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/

Training Methodology

- steps



Source: Yamashita R. et al. (2018) Convolutional neural networks: an overview and applications in radiology

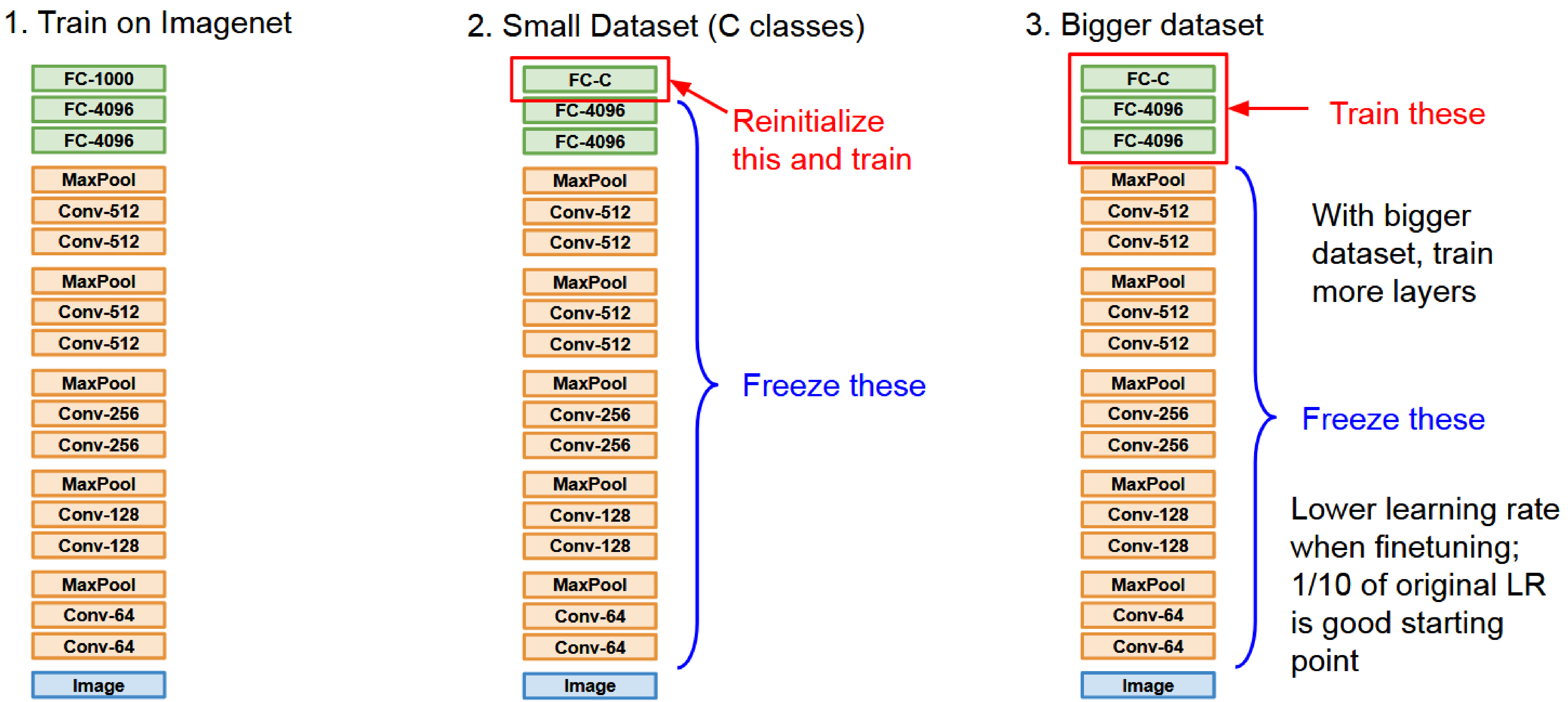
Transfer Learning

- Transfer learning aims to leverage the learned knowledge from a resource-rich domain/task to help learning a task with not sufficient training data.
  - Sometimes referred as **domain adaptation**

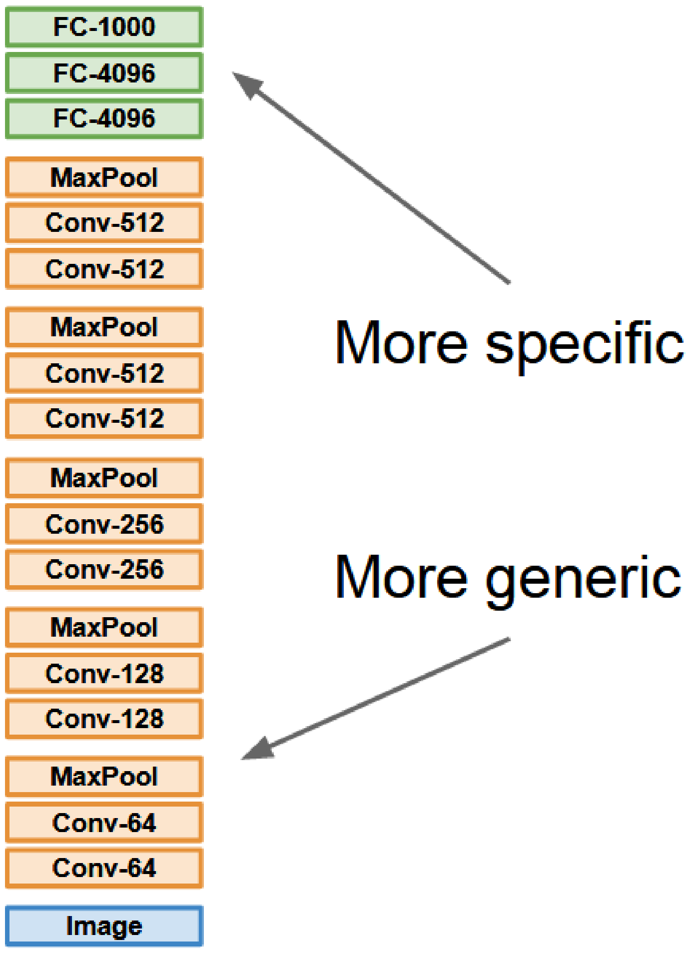
- The resource-rich domain is known as the **source** and the low-resource task is known as the **target**.

- Transfer learning works the best if the model features learned from the source task are **general** (i.e., domain-independent)

Transfer Learning with CNNs

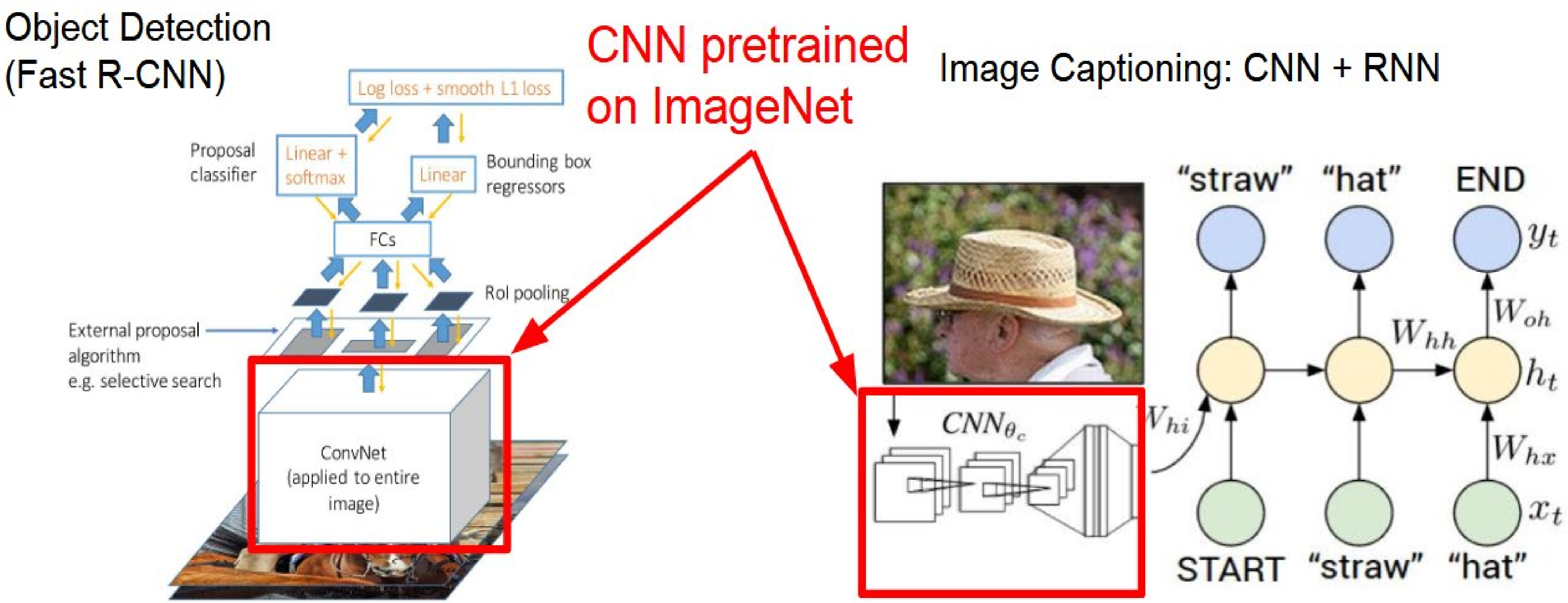


Transfer Learning with CNNs



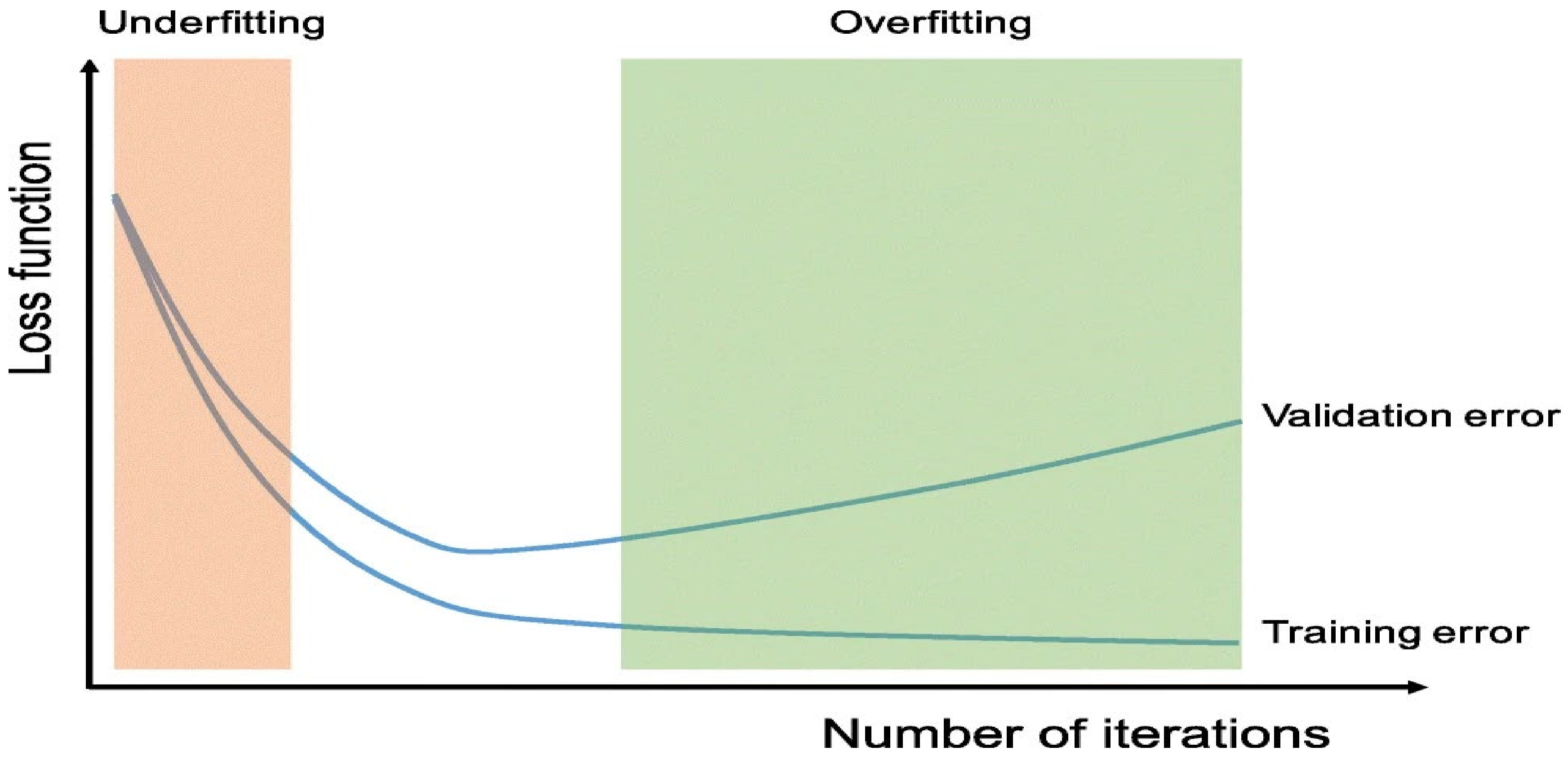
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer Learning is common in all applications



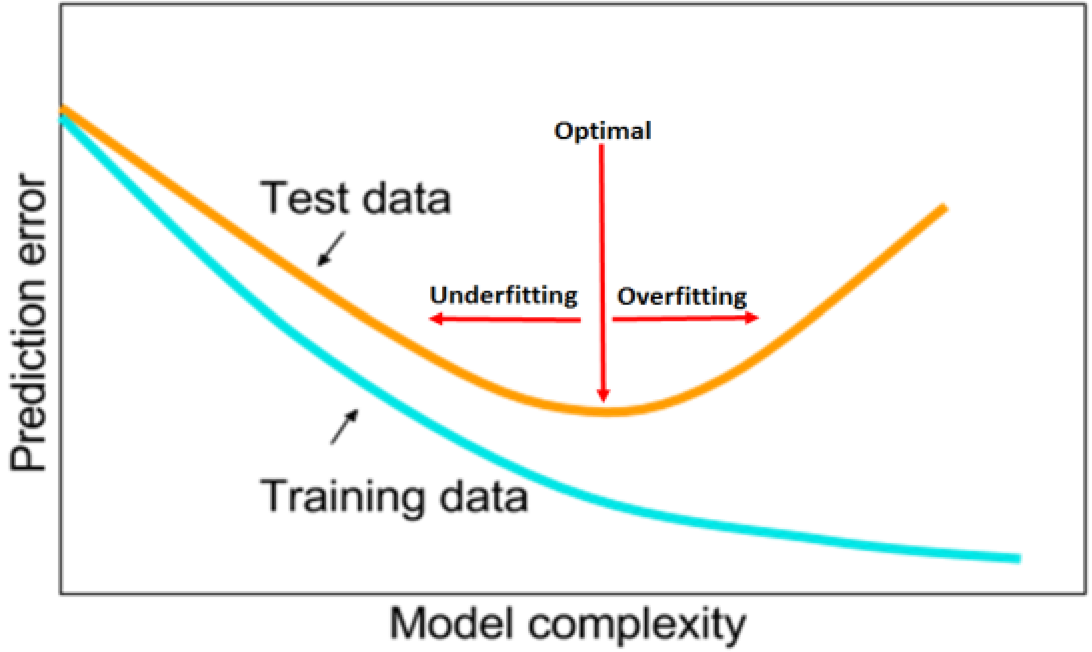
Overfitting and Underfitting

- Monitor the loss on training and validation sets during the training iteration.
- If the model performs poorly on both training and validation sets: Underfitting
- If the model performs well on the training set compared to the validation set: Overfitting



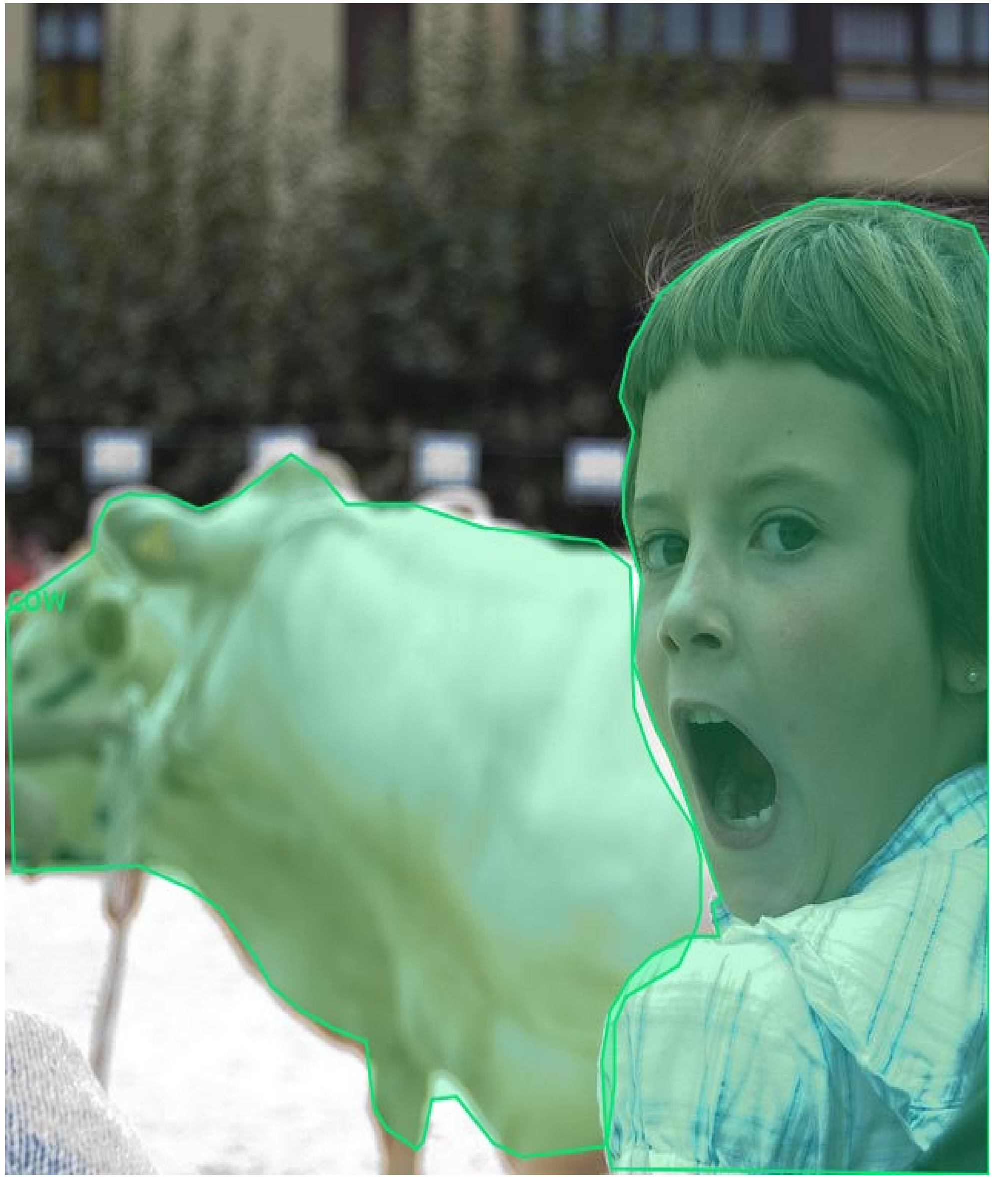
Common methods to mitigate overfitting

- More training data
- Early Stopping
- Data Augmentation
- Regularization (weight decay, dropout)
- Batch normalization



More training data

- Costly
- Time consuming
- Need experts for specialized domains



Source: Fast Annotation Net: A framework for active learning in 2018. https://medium.com/diffgram/fast-annotation-net-a-framework-for-active-learning-in-2018-1c75d6b4af92
Image Datasets — ImageNet, PASCAL, TinyImage, ESP and LabelMe — what do they offer ? Medium Blog

Early Stopping

- Training too little mean model will underfit on the training and testing sets
- Training too much mean model will overfit the training dataset and hence poor performance on test set
- Early Stopping:
  - To stop training at the point when performance on a validation set starts to degrade.
  - Idea is to stop training when generalization error increases
- How to use Early Stopping
  - Monitoring model performance: Using metric to evaluate to monitor performance of the model during training
  - Trigger to stop training:
    - No change in metric over a given number of epochs
    - A decrease in performance observed over a number of epochs
  - Some delay or "patience" is good for early stopping

Source: Machine Learning Mastery: A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks
URL: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

Data Augmentation

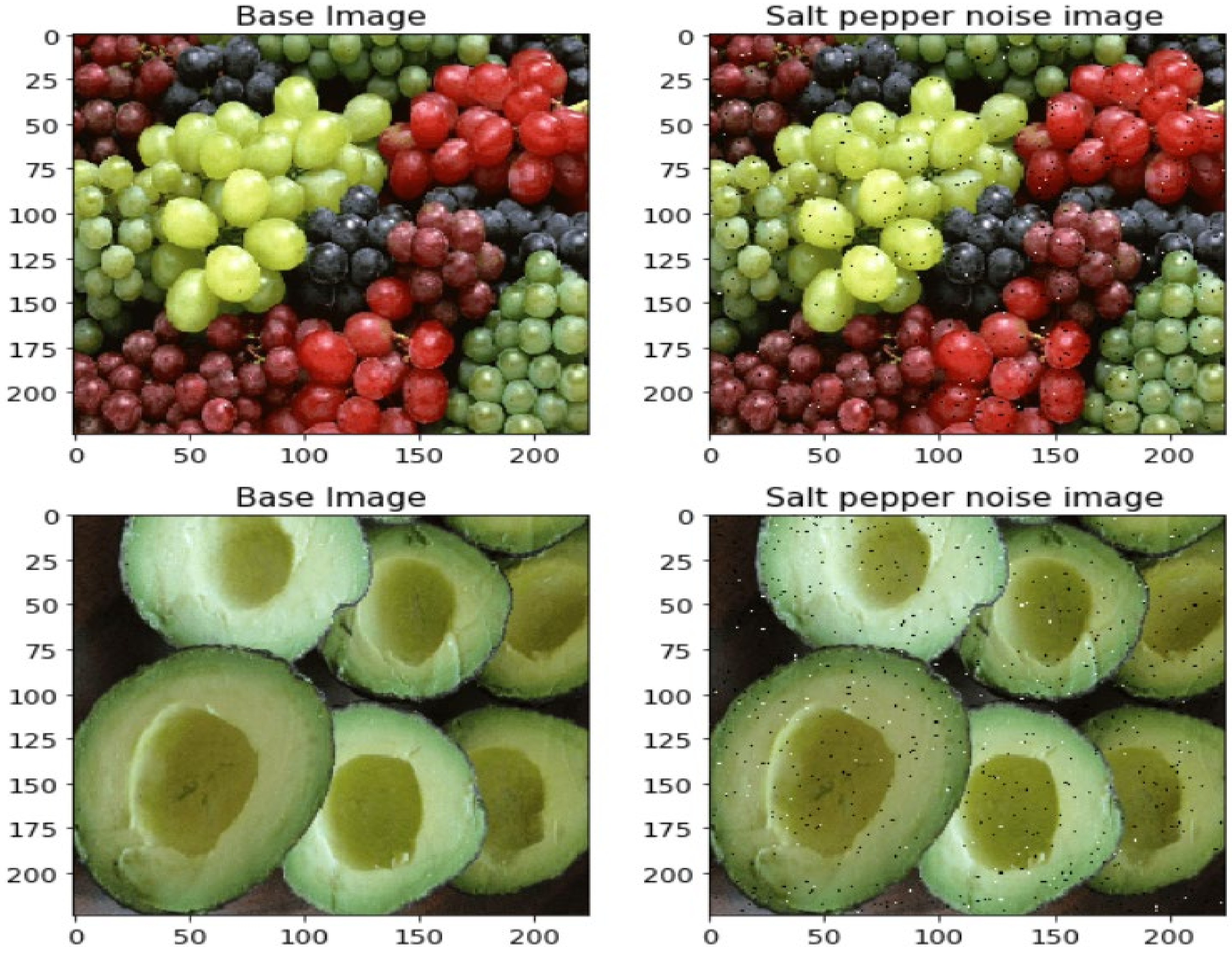
- Data augmentation generate different versions of a real dataset artificially to increase its size
- We use data augmentation to handle data scarcity and insufficient data diversity
- Data augmentation helps to increase performance of deep neural networks

- Common augmentation techniques:
  - Adding noise
  - Cropping
  - Flipping
  - Rotation
  - Scaling
  - Translation
  - Brightness
  - Contrast
  - Saturation
  - Generative Adversarial Networks (GANs)

Source: 13 Data Augmentation Techniques. https://research.aimultiple.com/data-augmentation-techniques/

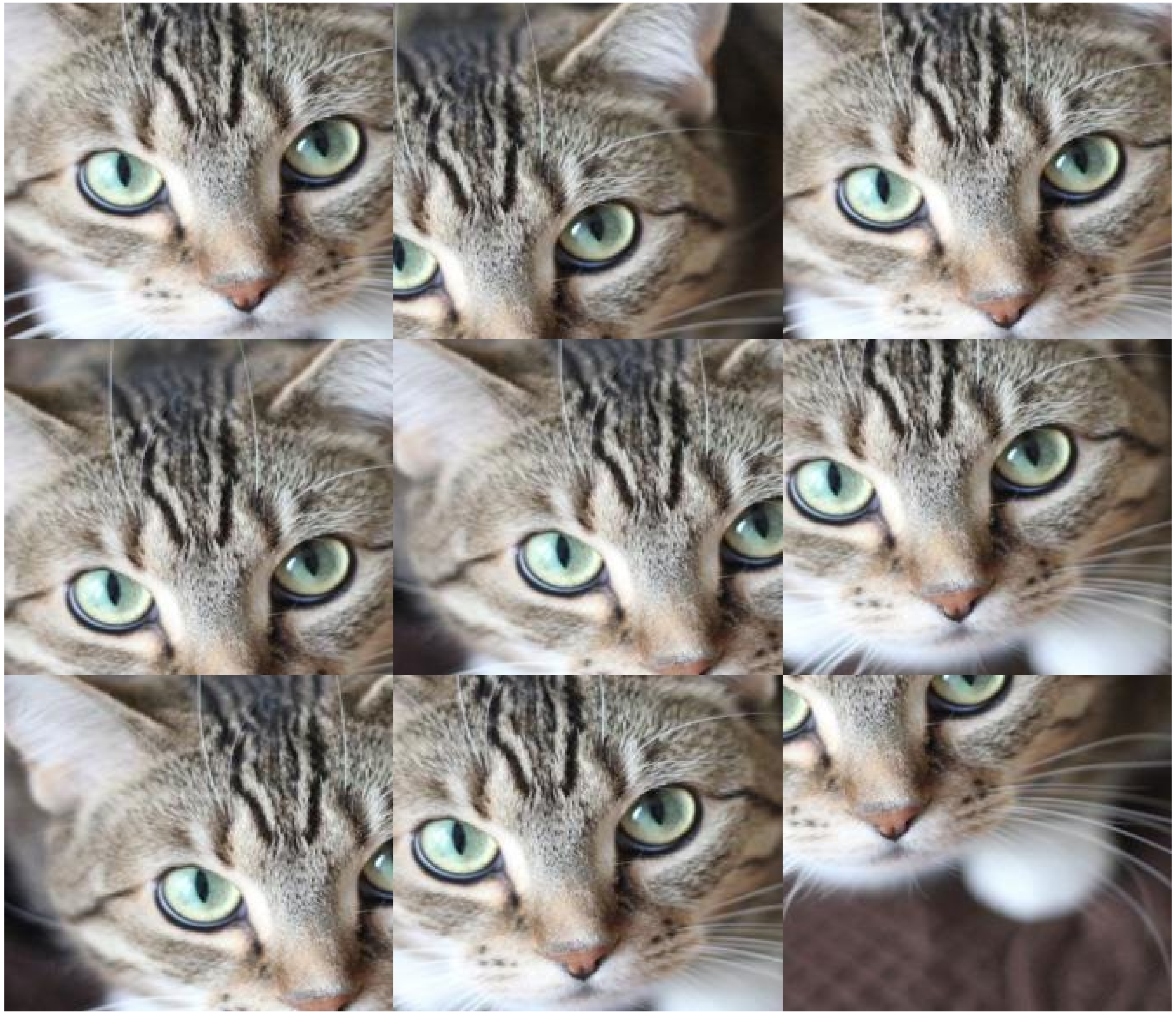
Data Augmentation

- Adding noise



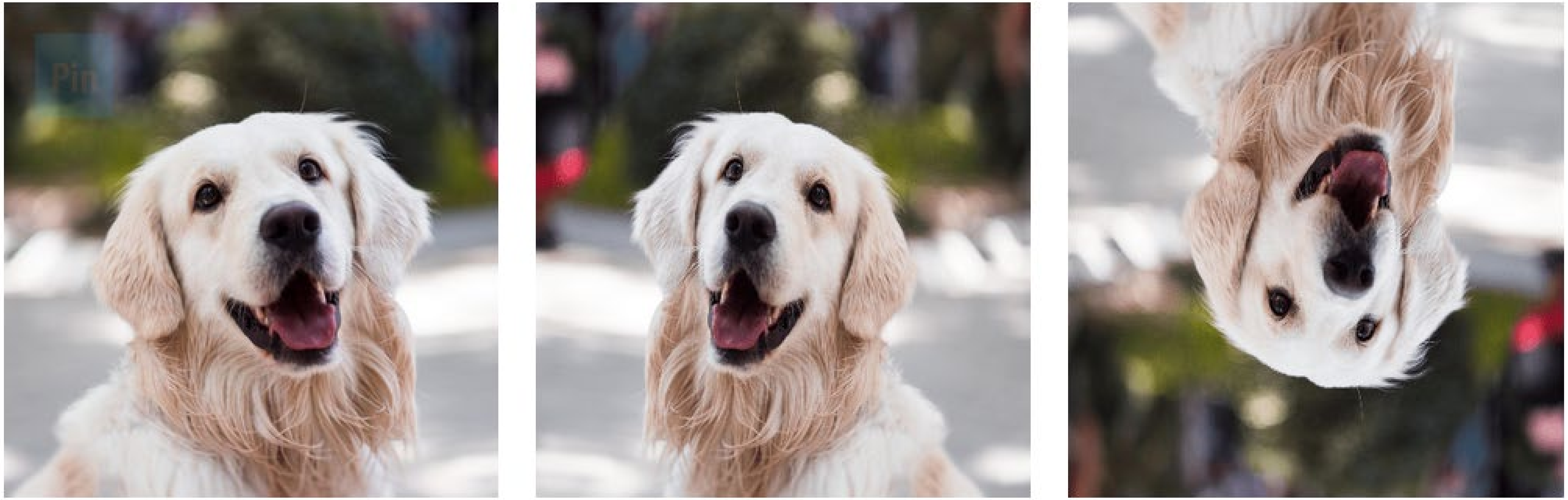
Data Augmentation

- Cropping



Data Augmentation

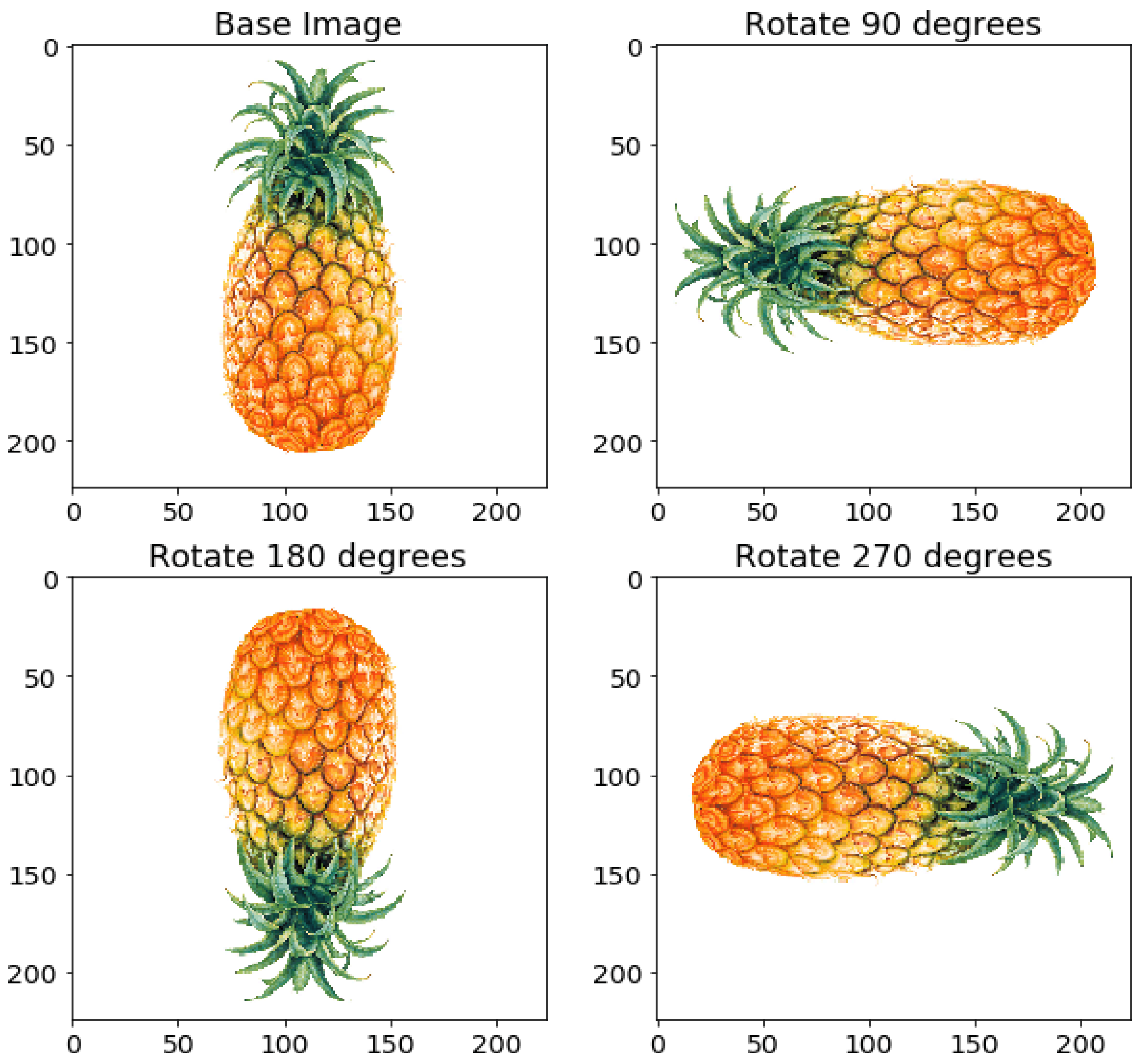
- Flipping



Source: 13 Data Augmentation Techniques. https://research.aimultiple.com/data-augmentation-techniques/

Data Augmentation

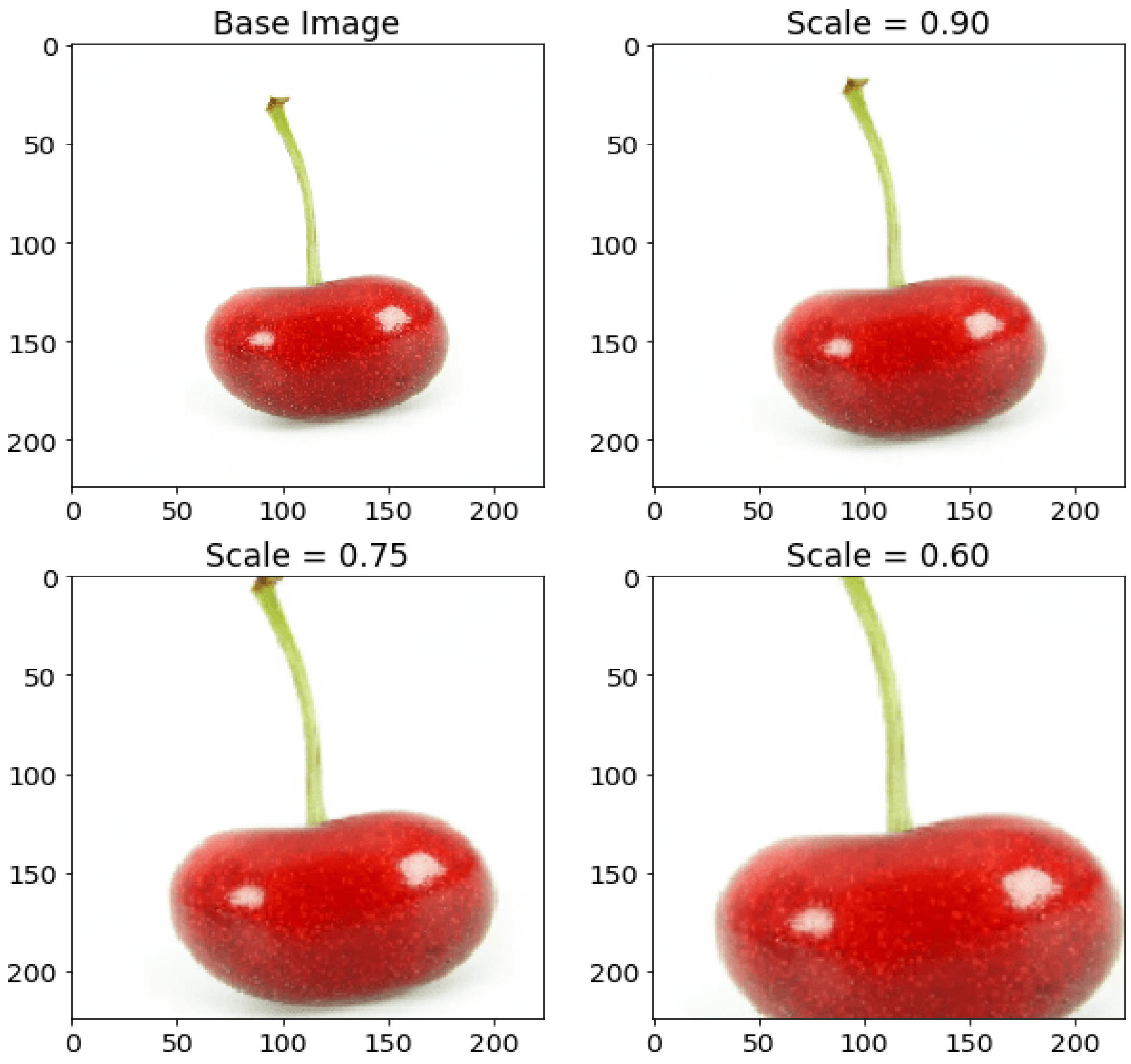
- Rotation



Source: 13 Data Augmentation Techniques. https://research.aimultiple.com/data-augmentation-techniques/

Data Augmentation

- Scaling

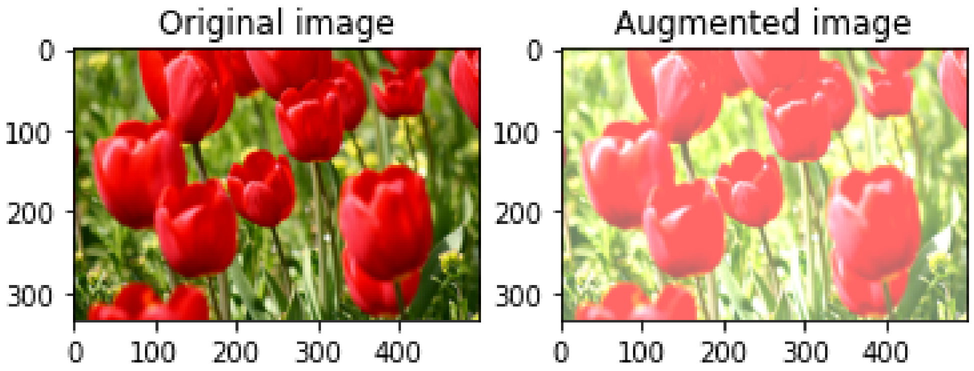


Data Augmentation

- Translation

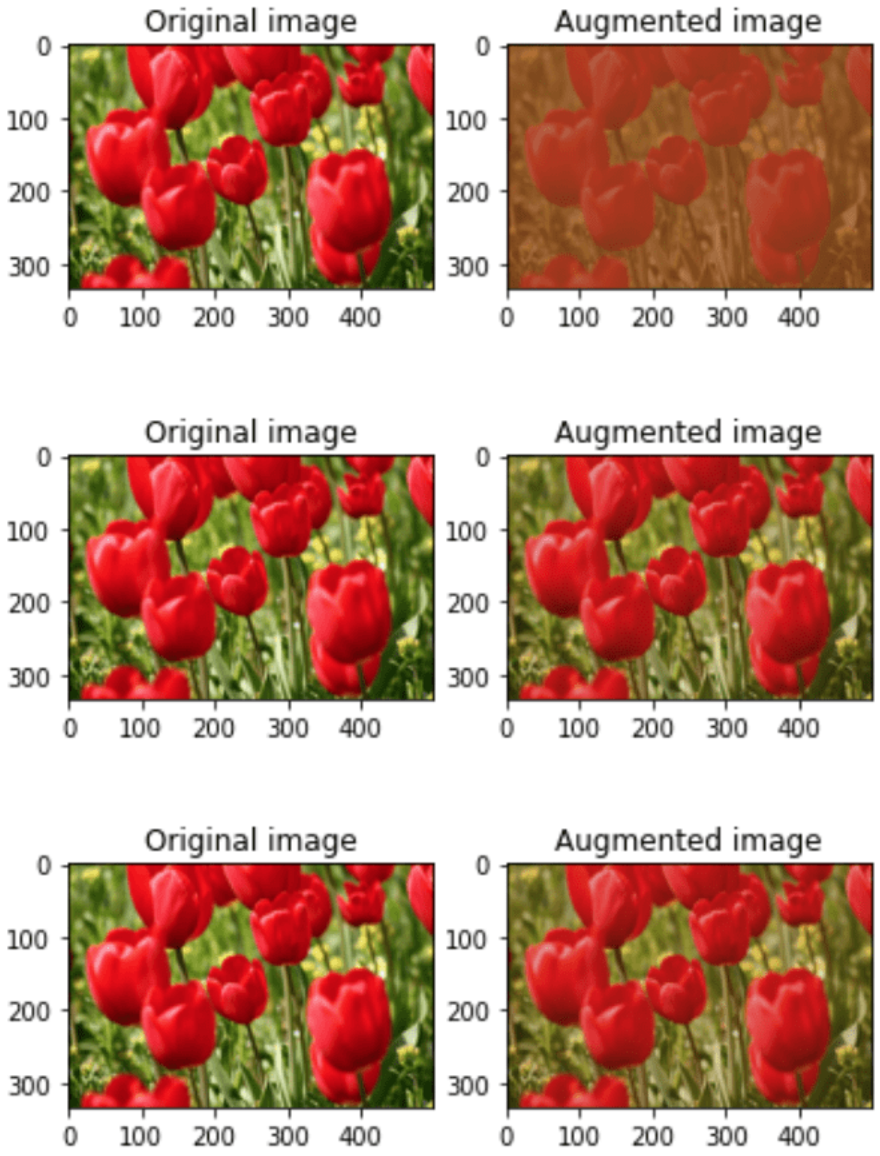
Data Augmentation

- Brightness



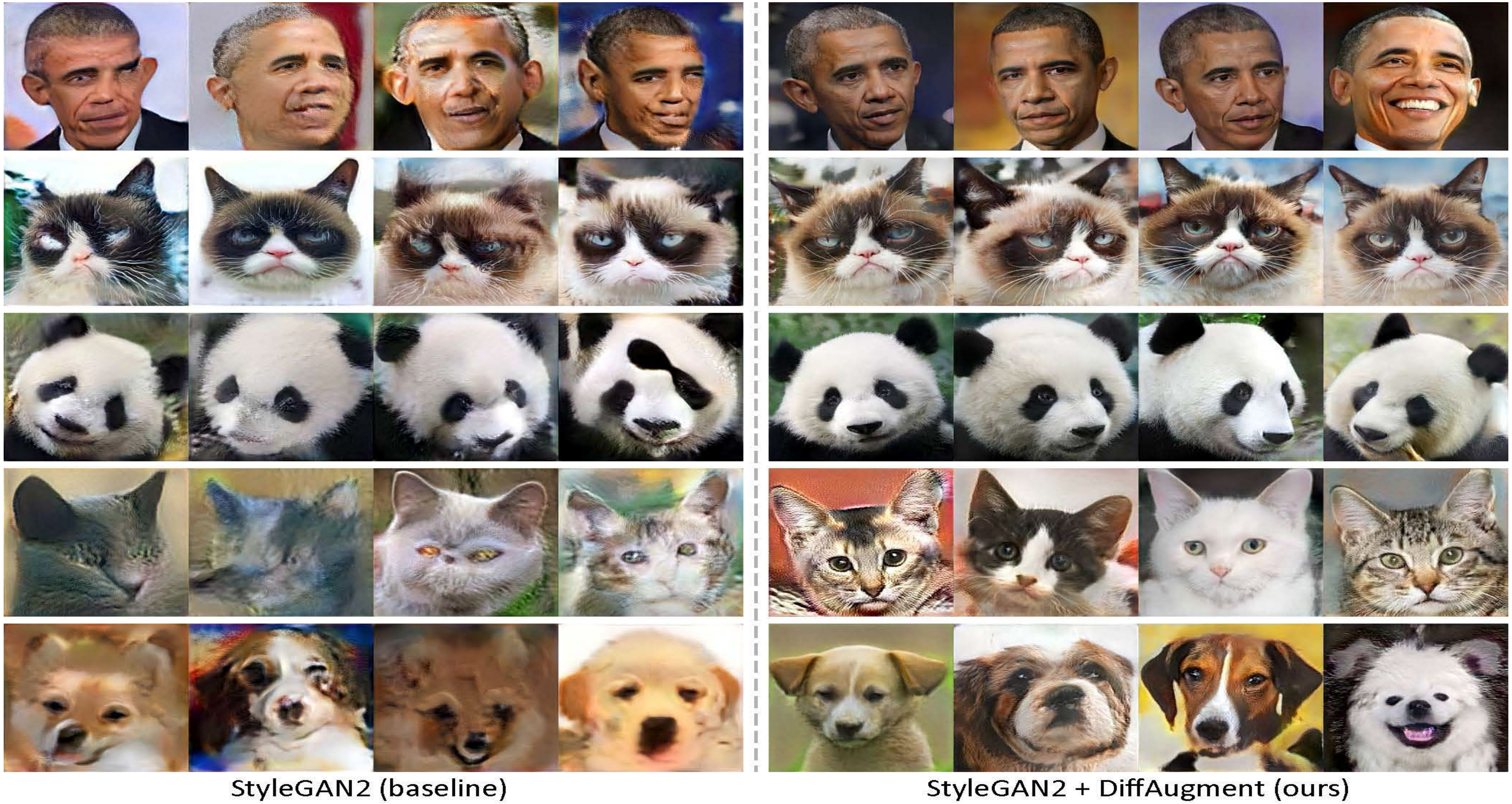
Data Augmentation

- Contrast



Data Augmentation

- Generative Adversarial Networks (GANs) for data augmentation



Source: Zhao et al., Differential Augmentation for Data-Efficient GAN Training, NeurIPS, 2020

Regularization: Weight Decay

- It adds a penalty term to the loss function on the training set to reduce the complexity of the learned model
- Popular choice for weight decay:
  - L1: The L1 penalty aims to minimize the absolute value of the weights

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} |\theta_i|}$$

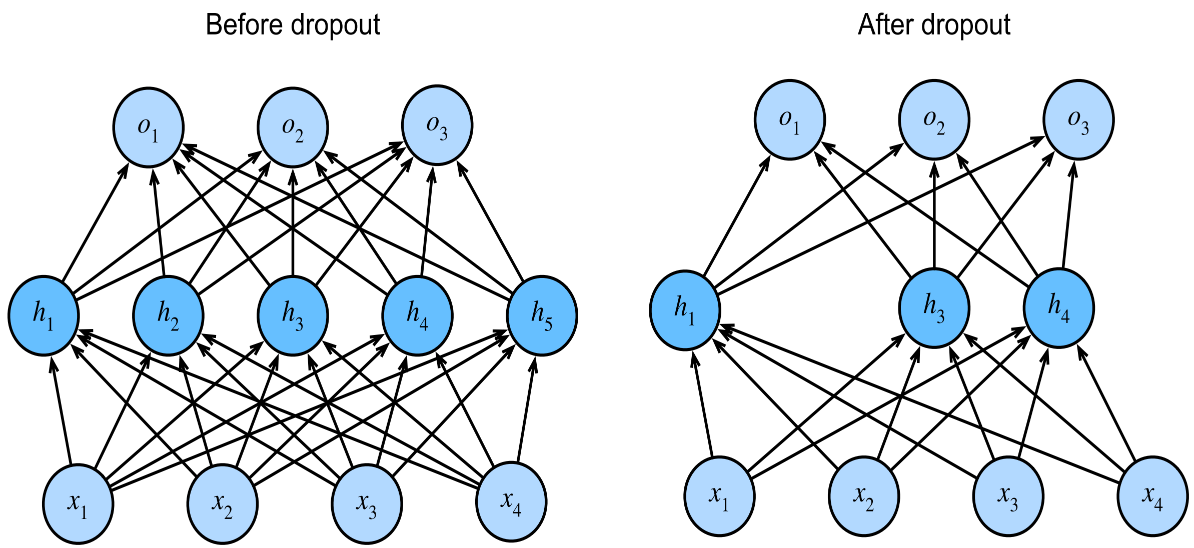
  - L2: The L2 penalty aims to minimize the squared magnitude of the weights

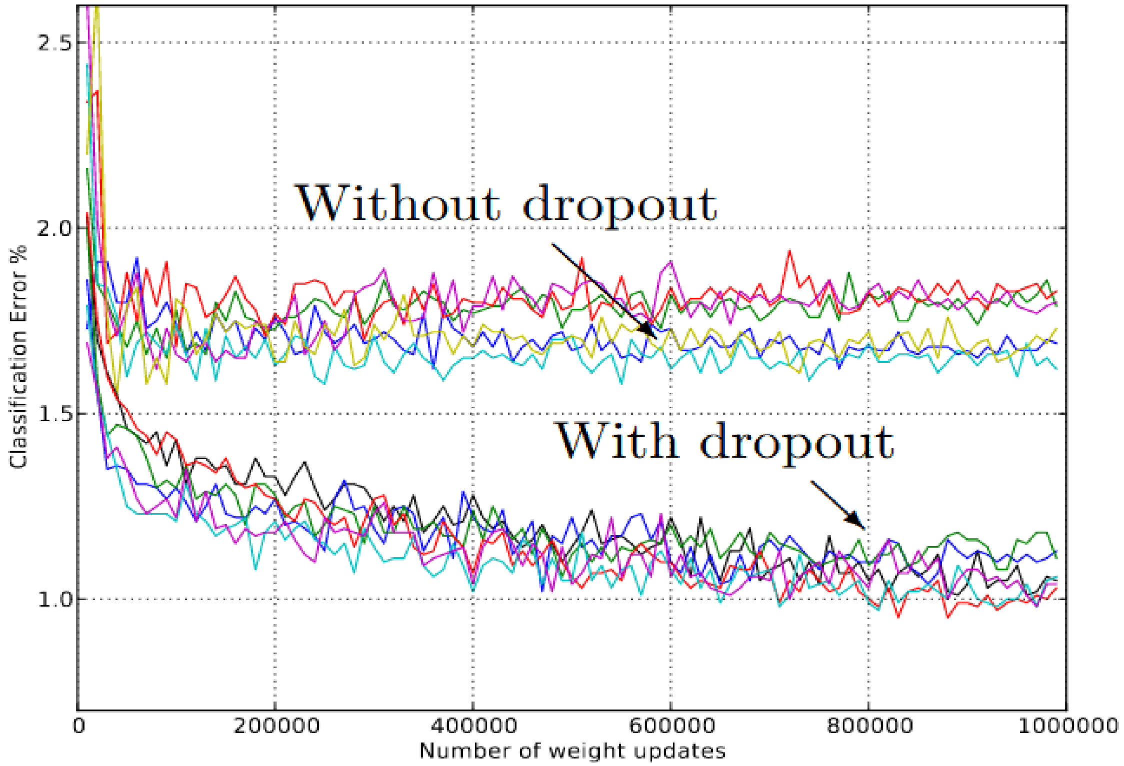
$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

Credit: 5 Techniques to Prevent Overfitting in Neural Networks. https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html

Regularization: Dropout

- L1 and L2 reduce overfitting by modifying the cost function
- Dropout modify the network by randomly dropping neurons from the neural network during training
- Dropout is an efficient way to average many large neural networks

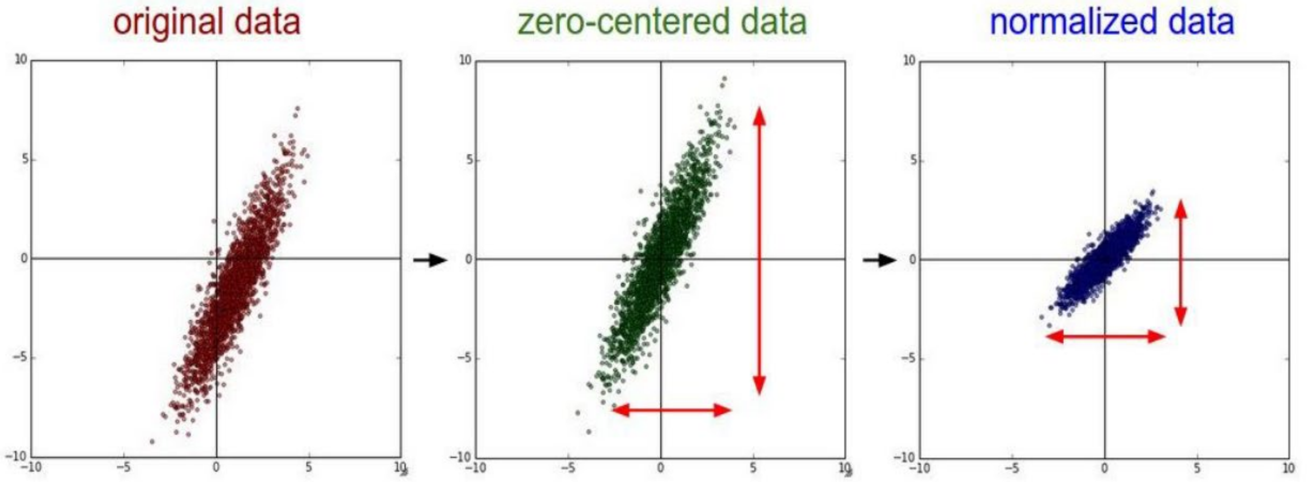




Credit: Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014
https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_multilayer-perceptrons/dropout.ipynb

Data Preprocessing

- The pixel values in images must be scaled prior to given as input to deep neural networks for training or evaluation
- Three main types of pixel scaling:
  - **Pixel Normalization**: scale pixel values to the range 0-1
  - **Pixel Centering**: scale pixel values to have a zero mean
  - **Pixel Standardization**: scale pixel values to have a zero mean and unit variance

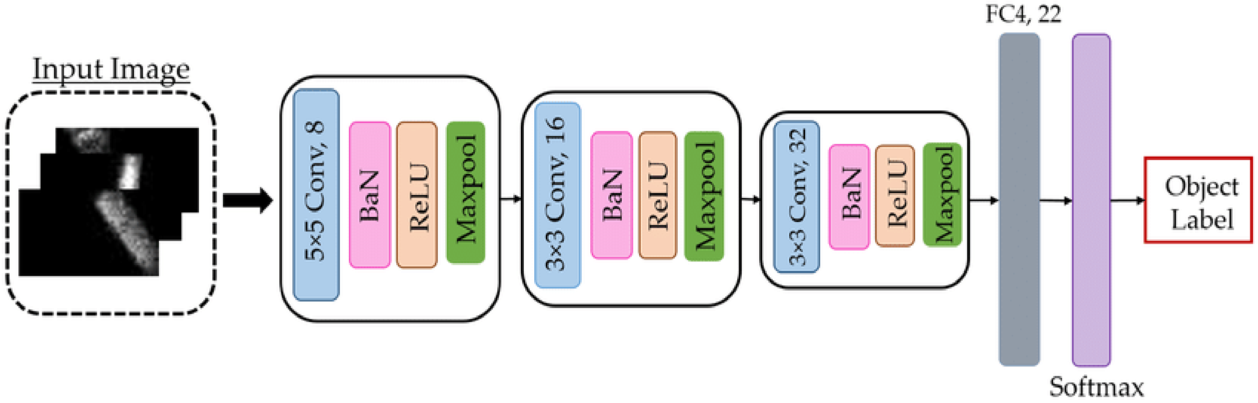


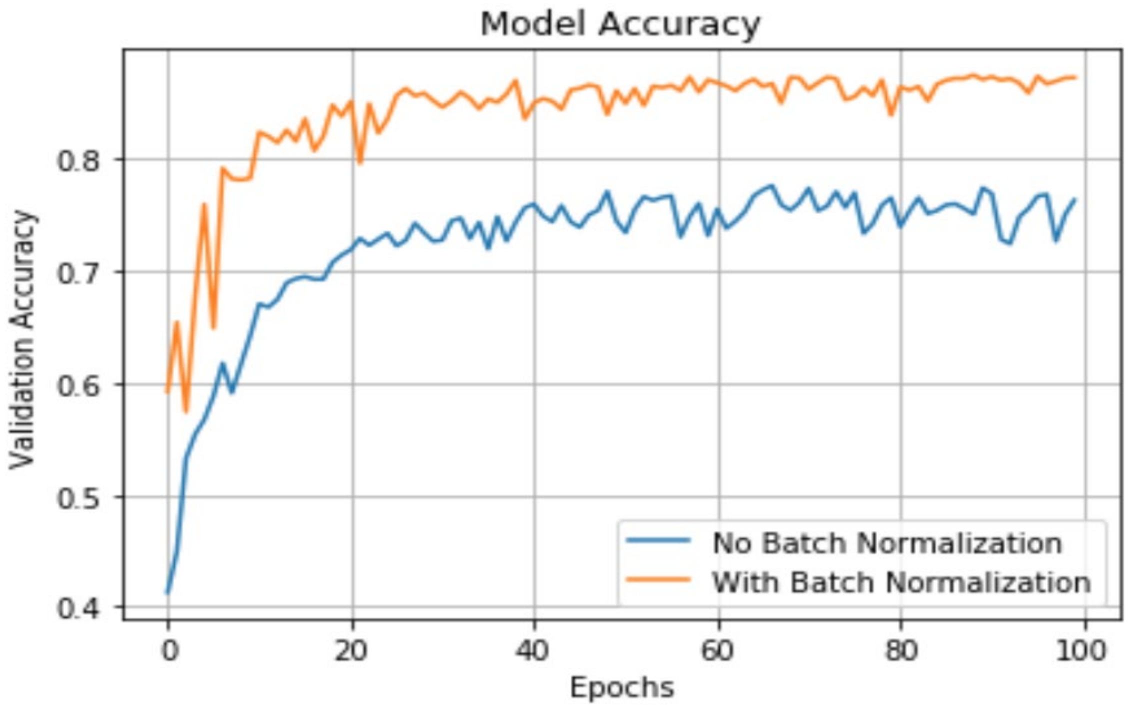

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Batch Normalization

- Enables stable training
- Reduces the **internal covariate shift** (ICS)
- Accelerates the training process
- Reduces the dependence of gradients on the scale of the parameters

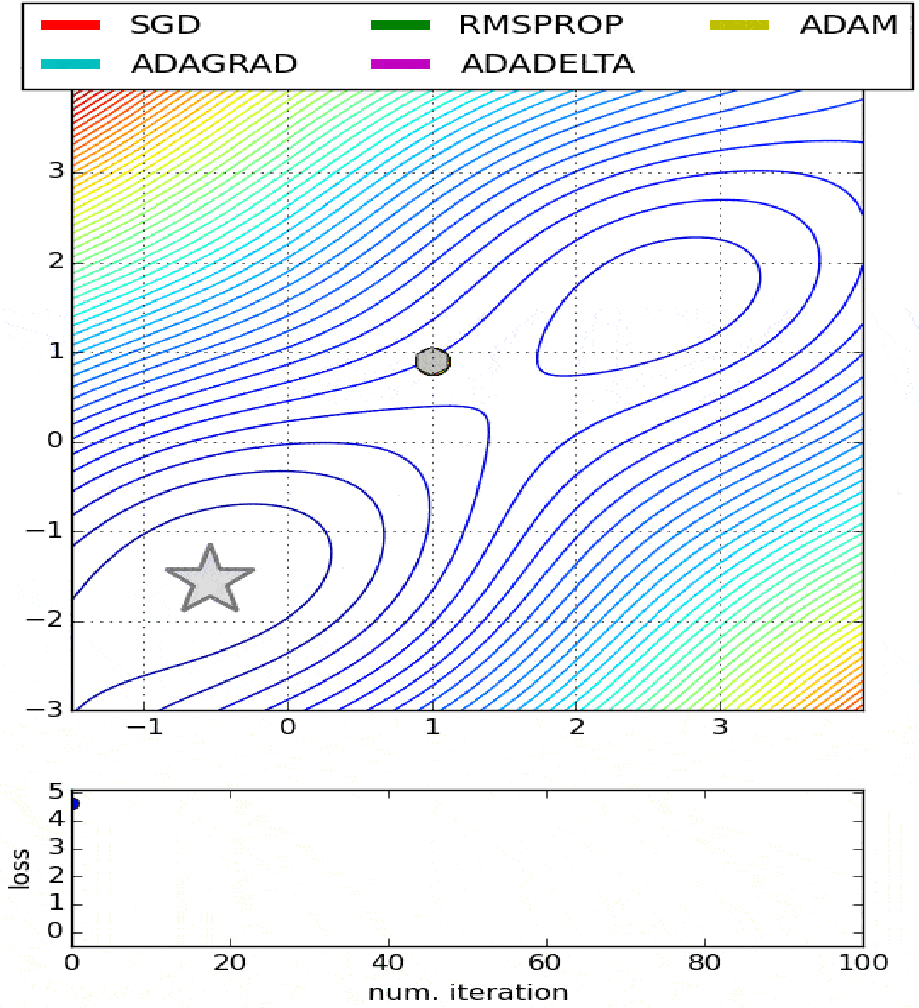




Source: LearnOpenCV: Batch Normalization in Deep Networks. https://learnopencv.com/batch-normalization-in-deep-networks/

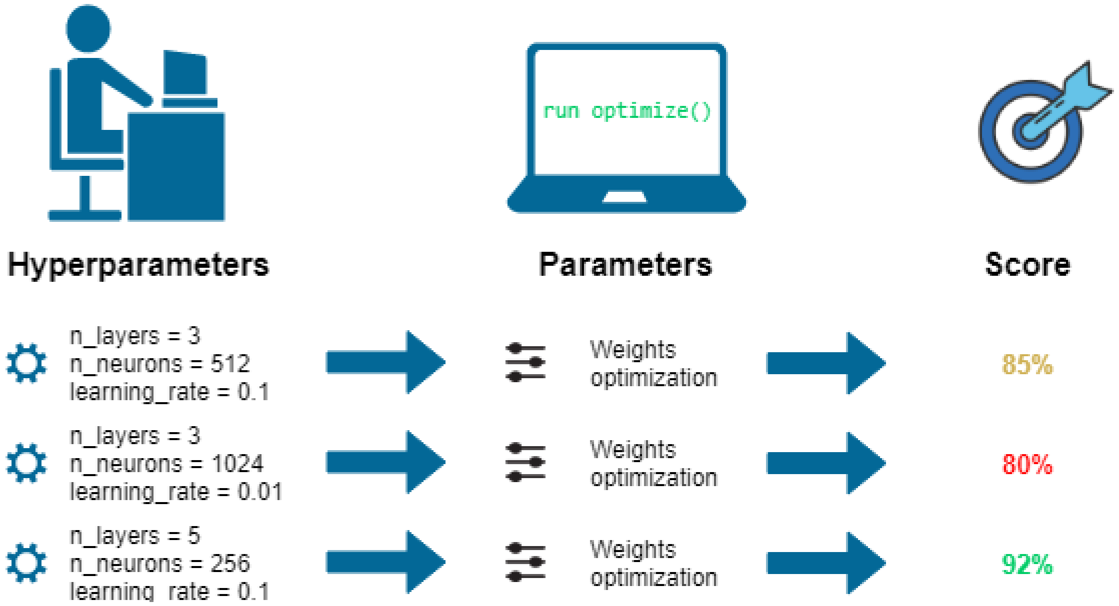
Choice of Optimizers

- Choosing right optimizer helps to update the model parameters and reducing the loss in much less effort
- Most DL frameworks supports various optimizers:
  - Stochastic Gradient Descent (SGD)
  - Momentum
  - Nesterov Accelerated Gradient
  - AdaGrad
  - AdaDelta
  - **Adam**
  - RMSProp

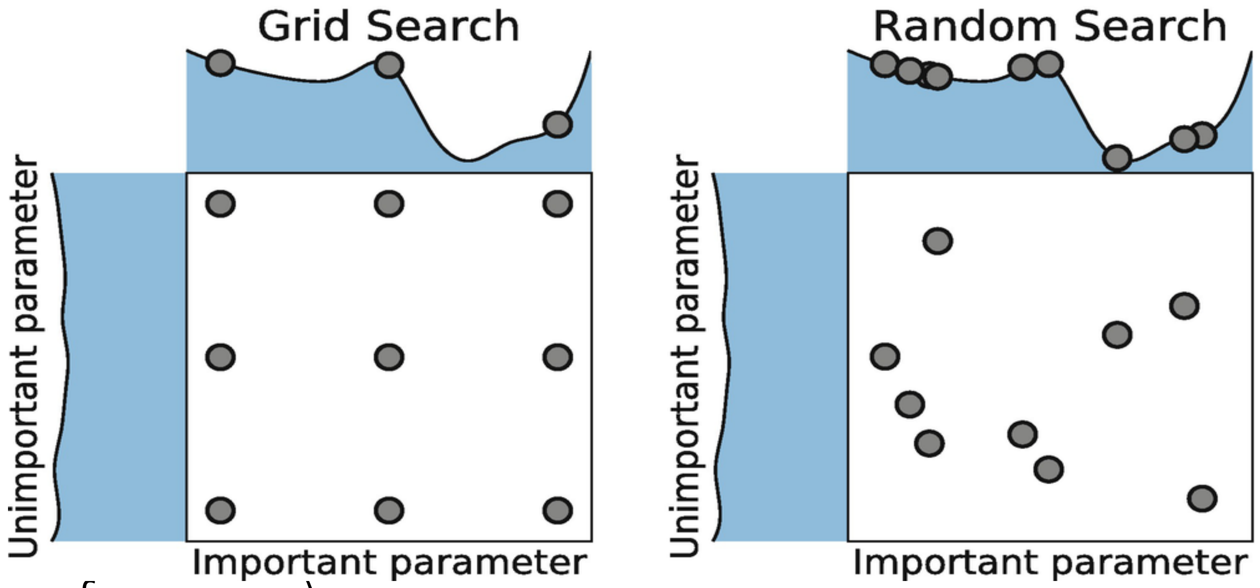


Tuning Hyperparameters

- Hyperparameters are all parameters which can be arbitrarily set by the user before starting training
- Hyperparameters are like knobs or dials of the network (model)
- An optimization problem: We aim to find the right combinations of their values which can help us to find either the minimum (e.g., loss) or the maximum (e.g., accuracy) of a function
- Many hyperparameters to tune:
  - Learning rate
  - No. of epochs
  - Dropout rate
  - Batch size
  - No. of hidden layers and units
  - Activation function
  - Weight initialization
  - …



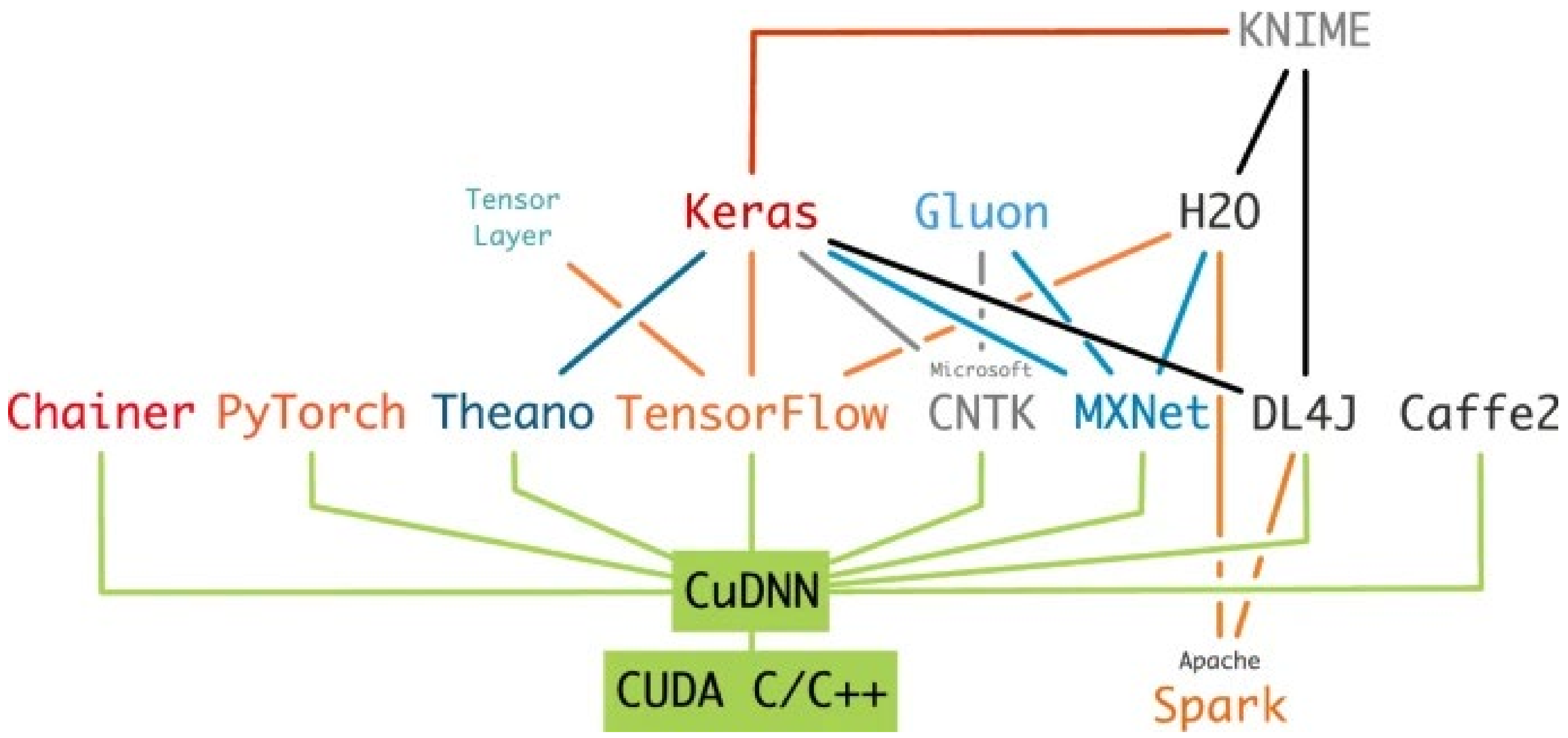
Tuning Hyperparameters strategies



- Random Guess
  - Simply use values from similar work
- Rely on your experience
  - Training DNNs is part art, part science
  - With experience sense of what works and what doesn’t
  - Still chances of being incorrect (suboptimal performance)
- Grid Search
  - Set up a grid of hyperparameters and train/test model on each of the possible combinations
- Automated hyperparameter tuning
  - Use of Bayesian optimization and Evolutionary Algorithms
  - Hyperopt: Distributed Asynchronous Hyperparameter Optimization

Source: KDnuggets: Practical Hyperparameter Optimization. https://www.kdnuggets.com/2020/02/practical-hyperparameter-optimization.html

Deep Learning Frameworks

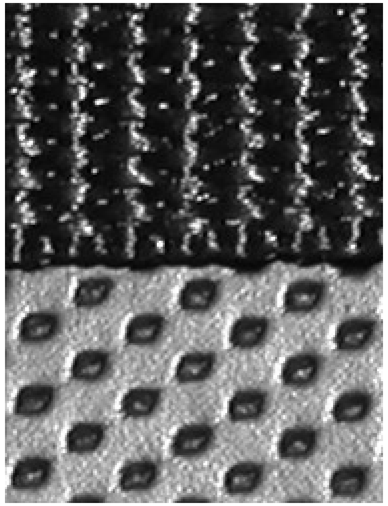


Source: Nguyen et al., (2019). ML and DL frameworks and libraries for large-scale data mining: a survey.

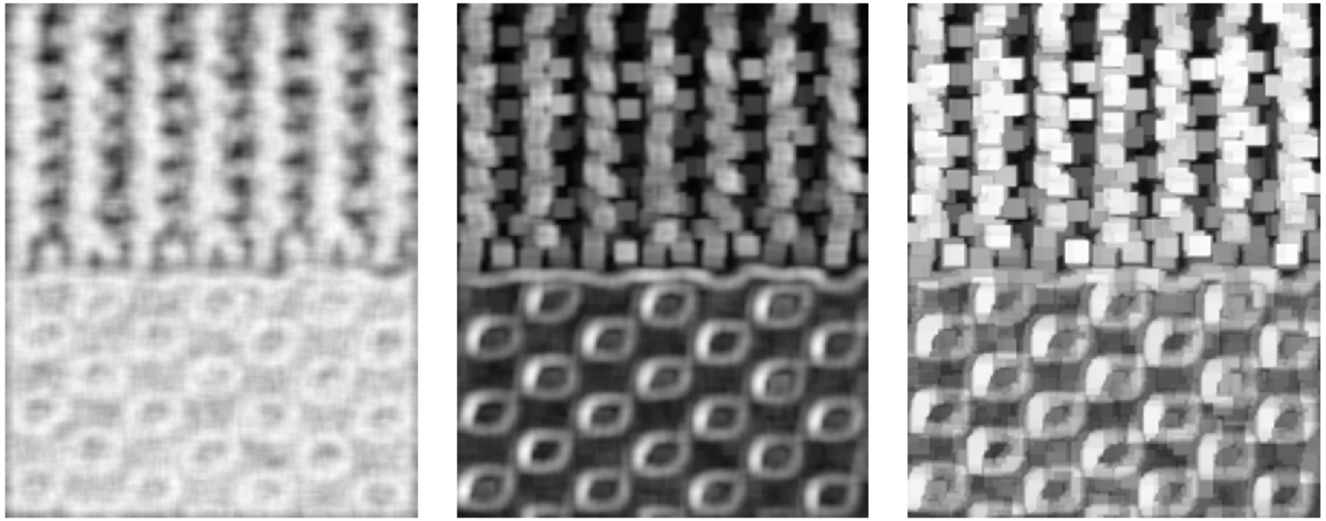
Texture

- Texture is a repeating pattern of local variations in image intensity
- Texture provides information in the spatial arrangement of colors or intensities in an image.
- Texture is characterized by the spatial distribution of intensity levels in a neighborhood.

Original Image



Texture Images Showing Local Entropy, Local Standard Deviation, and Local Range



Source: https://www.mathworks.com/help/images/texture-segmentation-using-texture-filters.html

Texture Synthesis

Neural Texture Synthesis

1. pretrain CNN on ImageNet (VGG-19)
2. pass input texture through CNN; compute feature map F_{ik}^{l} for i^{th} filter at spatial location k in layer (depth) l
3. compute the Gram matrix for each pair of features

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

4. feed (initially random) image into CNN
5. compute L2 distance between Gram matrices of original and new image
6. backprop to get gradient on image pixels
7. update image and go to step 5.

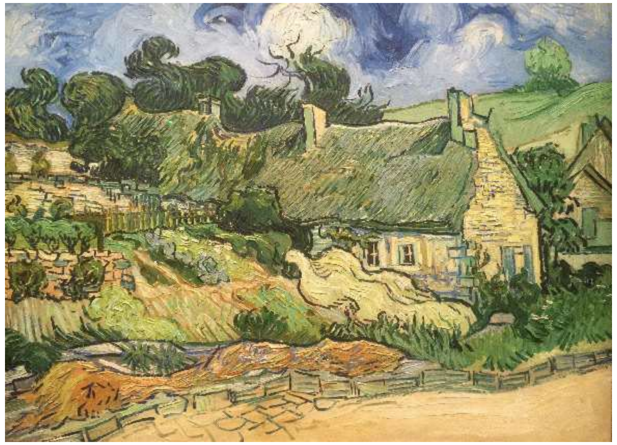
Neural Texture Synthesis

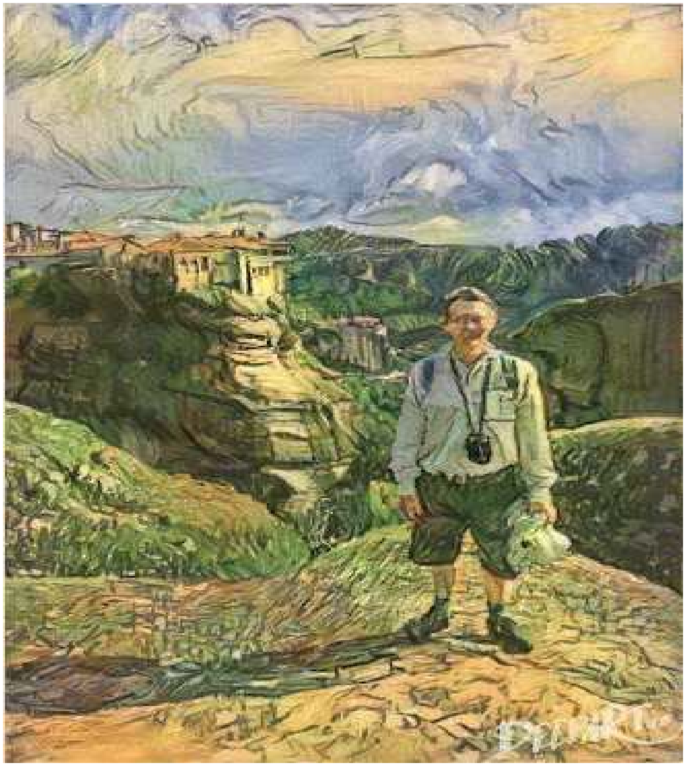
We can introduce a scaling factor w_l for each layer l in the network, and define the Cost function as

$$E_{\text{style}} = \frac{1}{4}\sum_{l=0}^{L}\frac{w_l}{N_l^2 M_l^2}\sum_{i,j}(G_{ij}^l - A_{ij}^l)^2$$

where N_l, M_l are the number of filters, and size of feature maps, in layer l, and G_{ij}^l, A_{ij}^l are the Gram matrices for the original and synthetic image.

Neural Style Transfer





Content + Style → New image

Neural Style Transfer

Neural Style Transfer

For Neural Style Transfer, we minimize a cost function which is

$$E_{\text{total}} = \alpha \; E_{\text{content}} \quad + \quad \beta \; E_{\text{style}}$$

$$= \frac{\alpha}{2} \sum_{i,k} ||F^l_{ik}(x) - F^l_{ik}(x_c)||^2 + \frac{\beta}{4} \sum_{l=0}^{L} \frac{w_l}{N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

where

x_c, x = content image, synthetic image

F^l_{ik} = i^{th} filter at position k in layer l

N_l, M_l = number of filters, and size of feature maps, in layer l

w_l = weighting factor for layer l

G^l_{ij}, A^l_{ij} = Gram matrices for style image, and synthetic image

Key takeaways

- Continuous improvement in CNN architectures and heuristics (tips and tricks)
  - always check literature to find state-of-the-art methods
- Training methodology
  - Split data into training (70 %), validation (10 %), and testing (20 %)
  - Take care of data leakage (e.g., multiple samples of same patients should be in same set)
  - Check distribution of classes, work on balanced datasets (ideally)
  - Tune hyperparameters on validation set. Save best model and do inference on test set (once)
  - Don’t use off-the-shelf model blindly. Do ablation studies to know its working
- Data augmentation techniques are not standardized
  - Get input from experts to know what data augmentations make sense in the domain
  - For e.g., in chest X-rays we don’t want vertical flipping
- Results
  - Use multiple metrics rather a single metric to report results (often they are complementary)
  - Show both qualitative and quantitative results (e.g., image segmentation)

Questions?