Spring 2024

CMPE 258-01 Deep Learning

Dr. Kaikai Liu, Ph.D. Associate Professor

Department of Computer Engineering

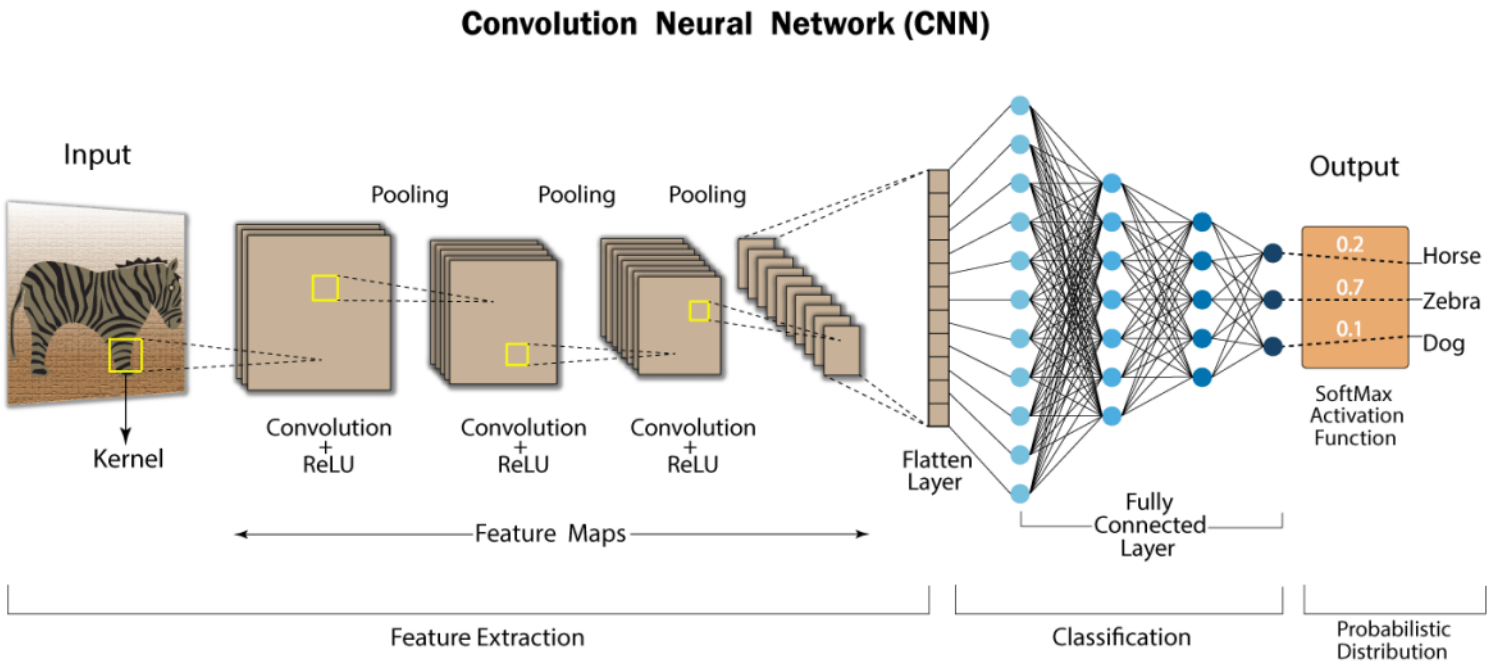
San Jose State University

Email: kaikai.liu@sjsu.edu

Website: https://www.sjsu.edu/cmpe/faculty/tenure-line/kaikai-liu.php

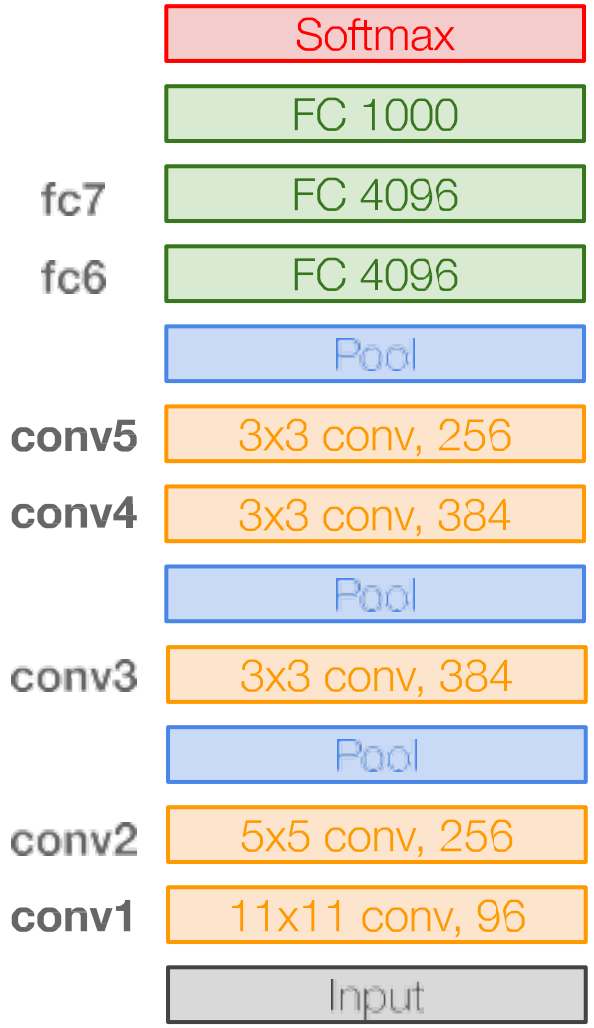
Transfer Learning

- Transfer learning
  - Transfer learning allows you to take a trained model and re-train it to perform another task.
    - For example, an image classification model could be retrained to recognize new categories of image. Re-training takes less time and requires less data than training a model from scratch.



AlexNet (2012)

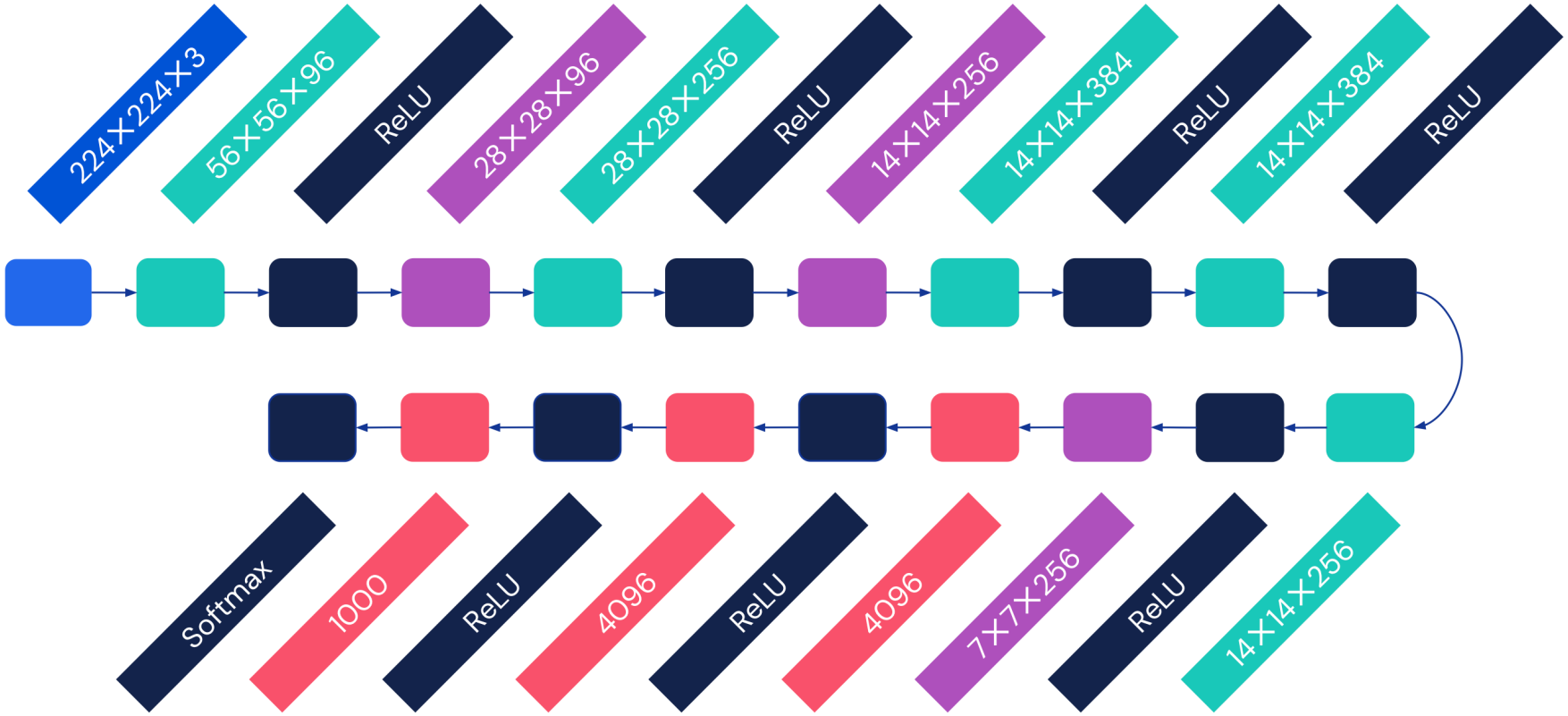
- Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ***ImageNet classification with deep convolutional neural networks***, Neural Information Processing Systems (2012)
  - Architecture: 8 layers, ReLU, dropout, weight decay
  - Input image: 224x224x3
  - Made up of 5 conv layers. It was the first architecture that employed max-pooling layers, ReLu activation functions, and dropout for the 3 enormous linear layers.
    - Layer1 convolution: 11x11, 96 channels, stride=4,
      - ->56x56x96
    - ReLU, Max-Pooling (2x2 -> 28x28x96)
    - Layer 8 fully connected

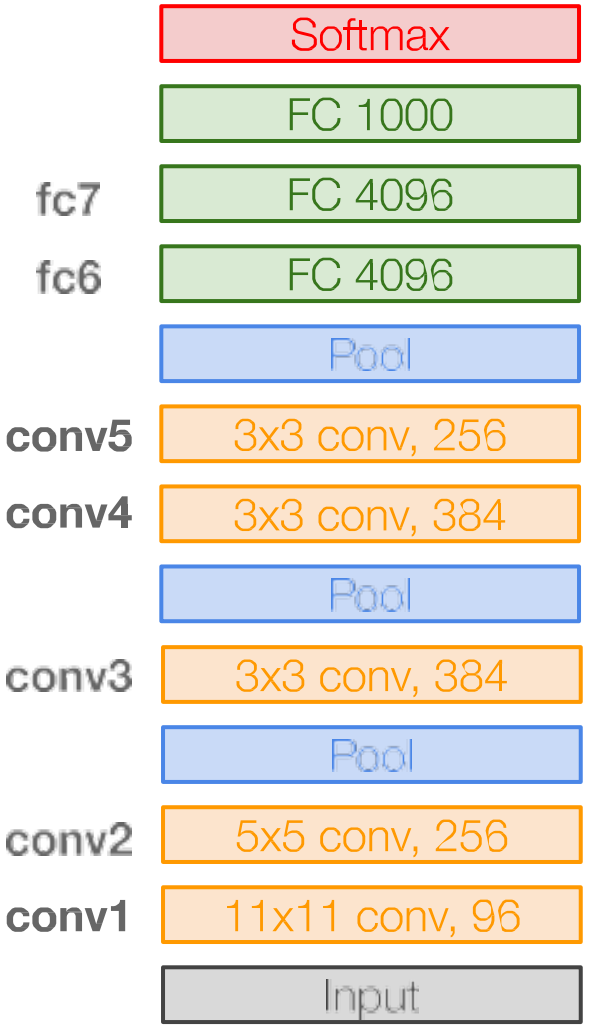


AlexNet

AlexNet (2012)

- AlexNet (2012)
  - Each layer is a linear classifier by itself
  - More layers – more nonlinearities

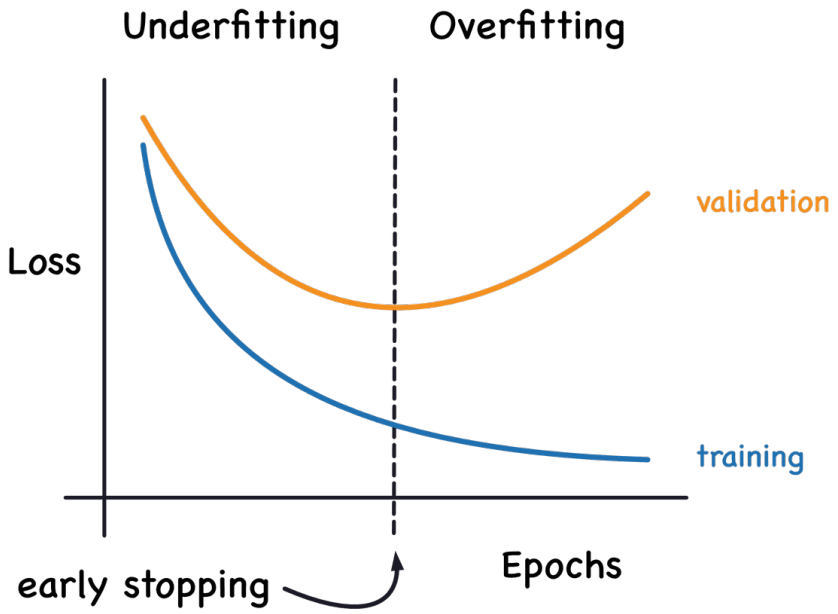




AlexNet

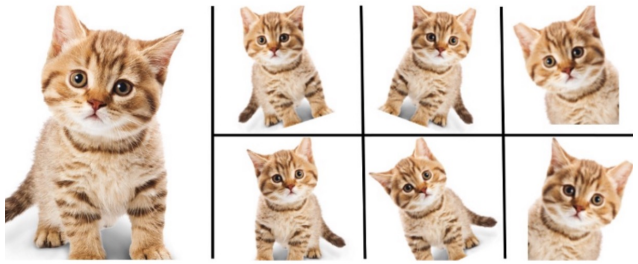
Regularization Solution

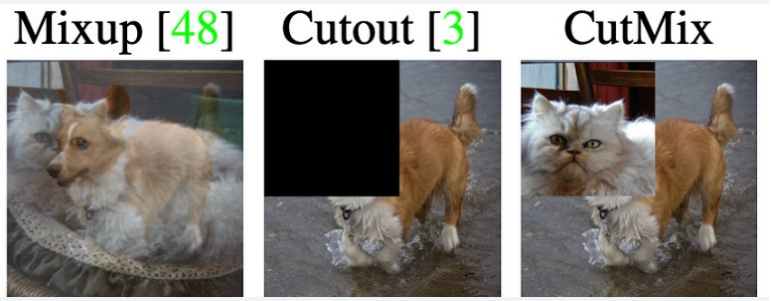
- Stochastic Depth: It drops entire network blocks while keeping the model intact during testing. The most popular application is in large ResNets where we bypass certain blocks through their skip connections.
- Early stopping: It refers to the process of stopping the training when the training error is no longer decreasing but the validation error is starting to rise.
  - This implies that we store the trainable parameters periodically and track the validation error. After the training stopped, we return the trainable parameters to the exact point where the validation error started to rise, instead of the last ones.
  - It can also be proven that in the case of a simple linear model with a quadratic error function and simple gradient descent, early stopping is equivalent to L2 regularization.



Data augmentation

- Data augmentation
  - Data augmentation refers to the process of generating new training examples to our dataset. More training data means lower model's variance, a.k.a lower generalization error. Simple as that. It can also be seen as a form of noise injection in the training dataset.
  - Data augmentation can be achieved in many different ways.
    - Basic Data Manipulations: Image flipping, cropping, rotations, translations, image color modification, image mixing, cutout, Mixup, cutmix
    - Feature Space Augmentation: we can apply transformations on the feature space. For example, an autoencoder might be used to extract the latent representation. Noise can then be added in the latent representation which results in a transformation of the original data point.





Data augmentation

- Data augmentation
  - Mixup: (ICLR 2018) Zhang et al. in mixup: Beyond Empirical Risk Minimization
  - Mixup is a data augmentation technique that generates a weighted combination of random image pairs from the training data. Given two images and their ground truth labels: a synthetic training example is generated as:

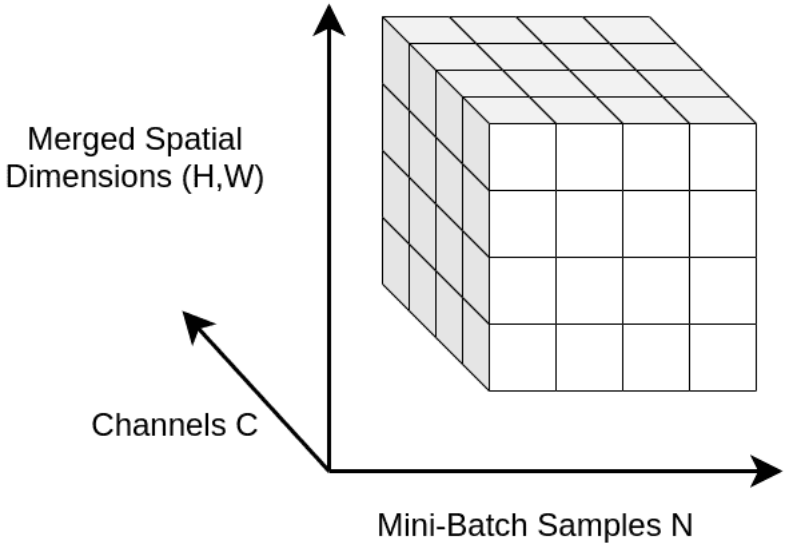
$$\hat{x} = \lambda x_i + (1 - \lambda) x_j$$

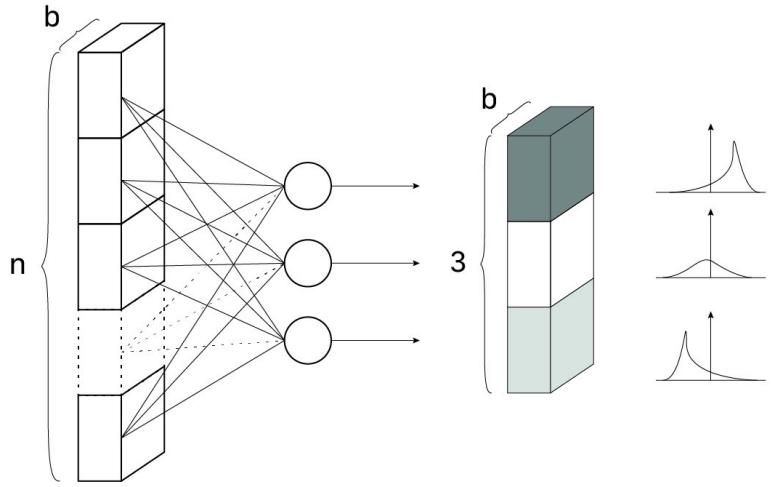
$$\hat{y} = \lambda y_i + (1 - \lambda) y_j$$

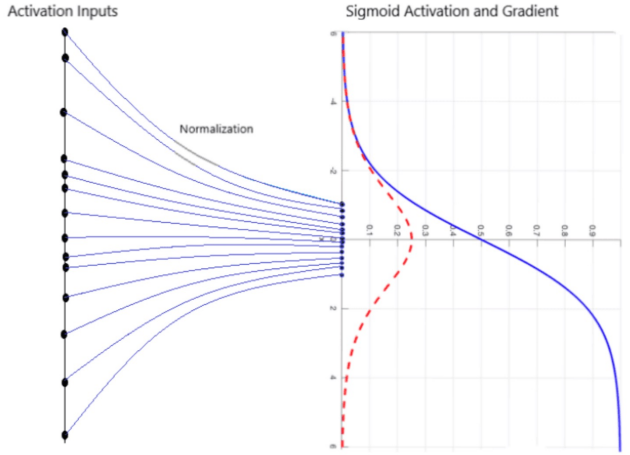
  - Cutout: DeVries et al. in Improved Regularization of Convolutional Neural Networks with Cutout
    - Cutout is an image augmentation and regularization technique that randomly masks out square regions of input during training. and can be used to improve the robustness and overall performance of convolutional neural networks.
    - simulate occluded examples
  - CutMix
    - Instead of simply removing pixels as in Cutout, we replace the removed regions with a patch from another image. The ground truth labels are also mixed proportionally to the number of pixels of combined images. The added patches further enhance localization ability by requiring the model to identify the object from a partial view.

In-layer Normalization

- In-layer normalization techniques for training very deep neural networks
  - The batch features are x with a shape of [N, C, H, W].
  - N will be the batch size, while H refers to the height, W to the width, and C to the feature channels
  - Visualize the 4D activation maps x by merging the spatial dimensions => 3D

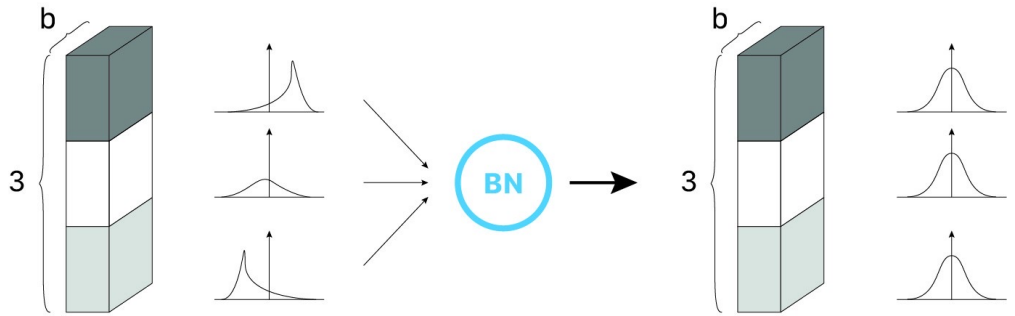






Batch-Normalization (BN)

- Ioffe, S.; Szegedy, C. ***Batch normalization: Accelerating deep network training by reducing internal covariate shift***, International conference on machine learning (2015)
  - Makes the training of Deep Neural Networks (DNN) **faster** and **more stable**.
  - It consists of **normalizing activation vectors from hidden layers** using the first and the second statistical moments (mean and variance) of the current batch. This normalization step is applied right before (or right after) the nonlinear function.
  - The BN layer first determines the **mean μ** and **the standard deviation σ** of the activation values across the batch. It then **normalizes the activation vector**



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Batch-Normalization (BN)

- **Batch-Normalization (BN)** is an algorithmic method which makes the training of Deep Neural Networks (DNN) **faster** and **more stable.**
  - It finally calculates the **layer's output Ẑ(i)** by applying a linear transformation with γ and β, two trainable parameters
  - Such step allows the model to choose the optimum distribution for each hidden layers, by adjusting those two parameters :
    - γ allows to adjust the standard deviation
    - β allows to adjust the bias, shifting the curve on the right or on the left side.

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

  - At each iteration, the network computes the mean μ and the standard deviation σ corresponding to the current batch. Then it trains γ and β *through* gradient descent, using an Exponential Moving Average (EMA) to give more importance to the latest iterations
  - **Pytorch** : torch.nn.BatchNorm1d, torch.nn.BatchNorm2d, torch.nn.BatchNorm3d
  - **Tensorflow / Keras** : tf.nn.batch_normalization, tf.keras.layers.BatchNormalization

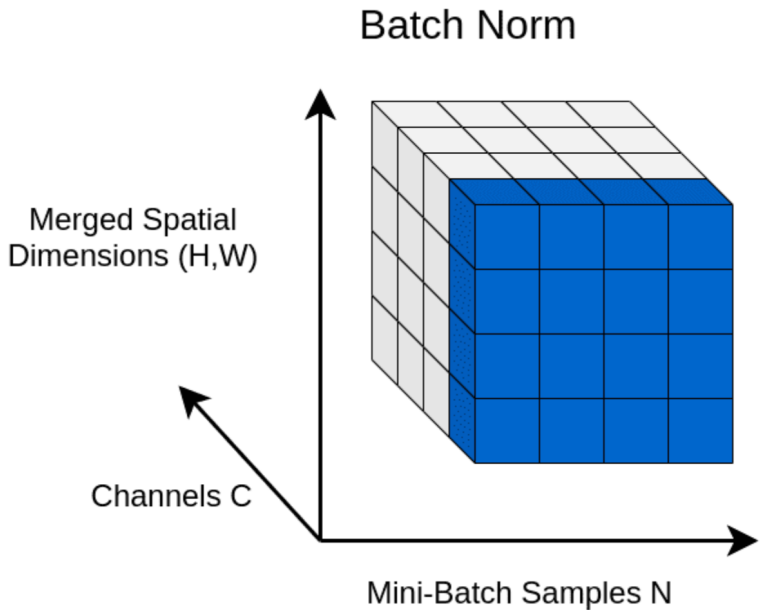
Batch-Normalization (BN)

- Batch-Normalization (BN)
  - The spatial dimensions, as well as the image batch, are averaged. This way, we concentrate our features in a compact Gaussian-like space
  - Some advantages of BN:
    - BN accelerates the training of deep neural networks.
    - For every input mini-batch we calculate different statistics. This introduces some sort of regularization (restricts the complexity)
  - Disadvantages:
    - Inaccurate estimation of batch statistics with small batch size
    - Problems when batch size is varying.

$$BN(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

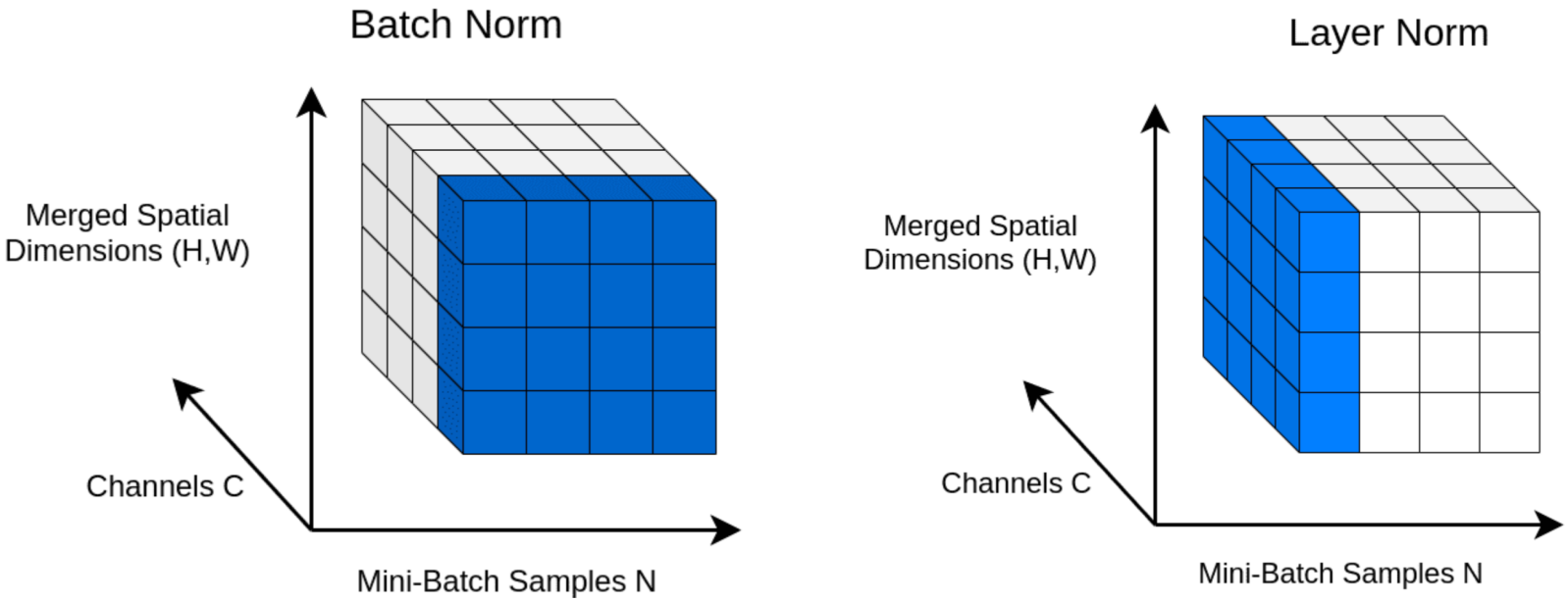
$$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2}$$



Layer normalization

- Layer normalization (2016)
  - In BN, the statistics are computed across the batch and the spatial dims. In contrast, in Layer Normalization (LN), the statistics (mean and variance) are computed across all channels and spatial dims. Thus, the statistics are independent of the batch. This layer was initially introduced to handle vectors (mostly the RNN outputs).



Layer normalization

- Layer normalization (2016)
  - Become popular when the transformer paper came out
  - Vectors with batch size of N: $R^{N \times K}$
  - We don't want to be dependent on the choice of batch and it's statistics, we normalize with the mean and variance of each vector.

$$\mu_n = \frac{1}{K}\sum_{k=1}^{K} x_{nk}$$

$$\sigma_n^2 = \frac{1}{K}\sum_{k=1}^{K}(x_{nk} - \mu_n)^2$$

$$\hat{x}_{nk} = \frac{x_{nk} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}, \hat{x}_{nk} \in R$$

$$\mathrm{LN}_{\gamma,\beta}(x_n) = \gamma\hat{x}_n + \beta, x_n \in R^K$$

Generalizing into 4D feature map tensors

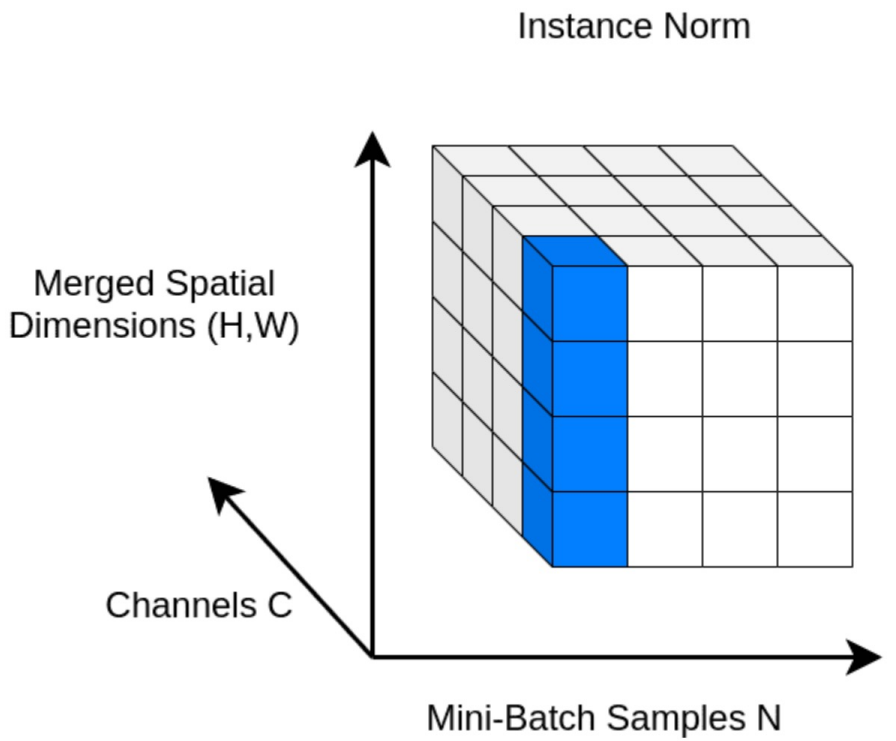
$$LN(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

$$\mu_n(x) = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_n(x) = \sqrt{\frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_n(x))^2}$$

Instance Normalization

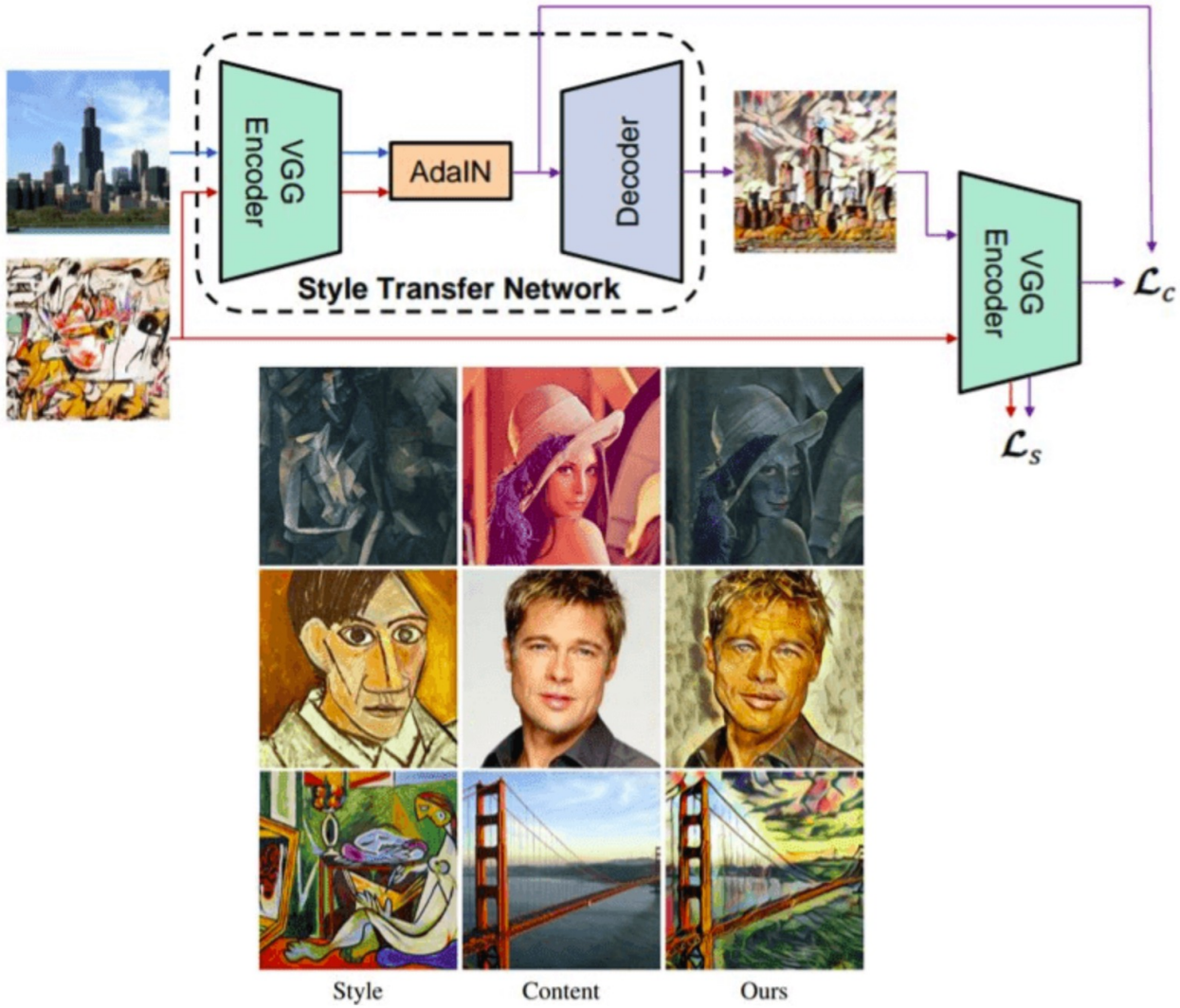
- Instance Normalization: The Missing Ingredient for Fast Stylization (2016)
  - Instance Normalization (IN) is computed only across the features' spatial dimensions. So it is independent for each channel and sample.
  - Surprisingly, the affine parameters in IN can completely change the style of the output image. IN can normalize the style of each individual sample to a target style (modeled by γ and β): global ones (i.e. style information).
  - One can design a network to model a plethora of finite styles, which is exactly the case of the so-called conditional IN



Adaptive Instance Normalization

- Adaptive Instance Normalization (2017)
  - Normalization and style transfer are closely related.
  - Adaptive Instance Normalization (AdaIN) receives an input image x (content) and a style input y, and simply aligns the channel-wise mean and variance of x to match those of y. Mathematically:

$$AdaIN(x, y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y)$$



https://github.com/xunhuang1995/AdaIN-style

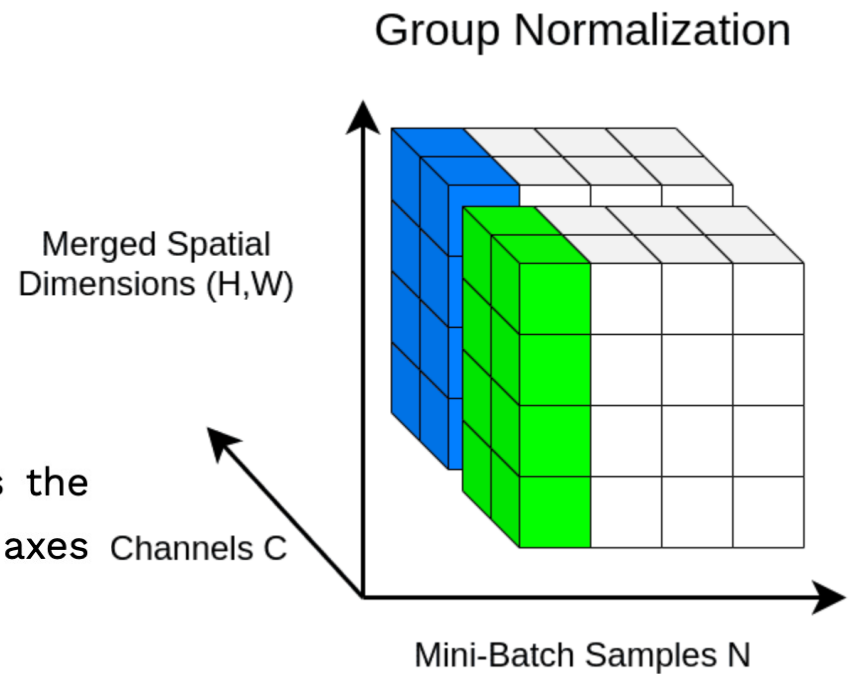
Group normalization

- Group normalization (2018)
  - Group normalization (GN) divides the channels into groups and computes the first-order statistics within each group.
  - GN's computation is independent of batch sizes, and its accuracy is more stable than BN in a wide range of batch sizes.

$$\mu_i = \frac{1}{m}\sum_{k\in\mathcal{S}_i} x_k, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{k\in\mathcal{S}_i}(x_k-\mu_i)^2+\epsilon}$$

$$\mathcal{S}_i = \left\{k \mid k_N = i_N, \left\lfloor \frac{k_C}{C/G} \right\rfloor = \left\lfloor \frac{i_C}{C/G} \right\rfloor\right\}$$

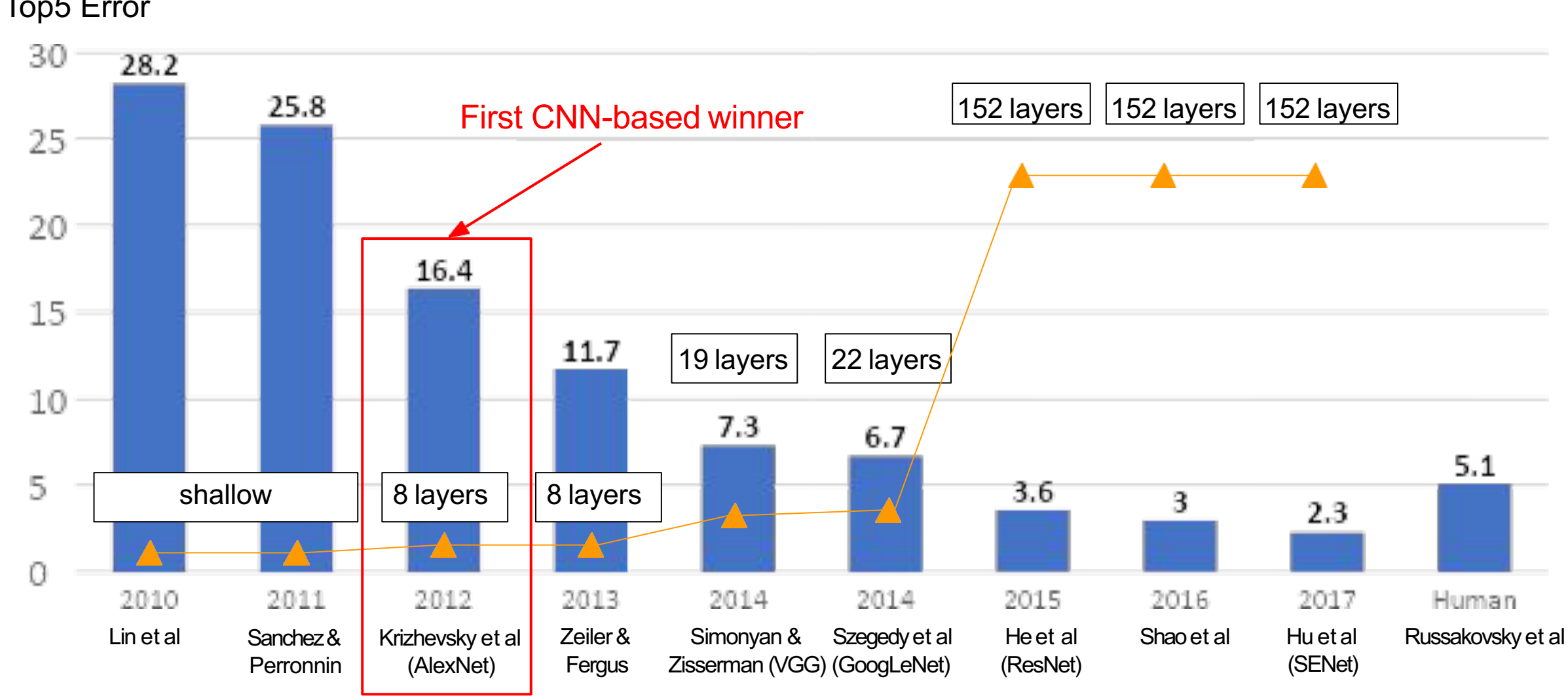
Note that G is the number of groups, which is a hyper-parameter. C/G is the number of channels per group. Thus, GN computes μ and σ along the (H, W) axes and along a group of C/G channels.



Synchronized Batch Normalization

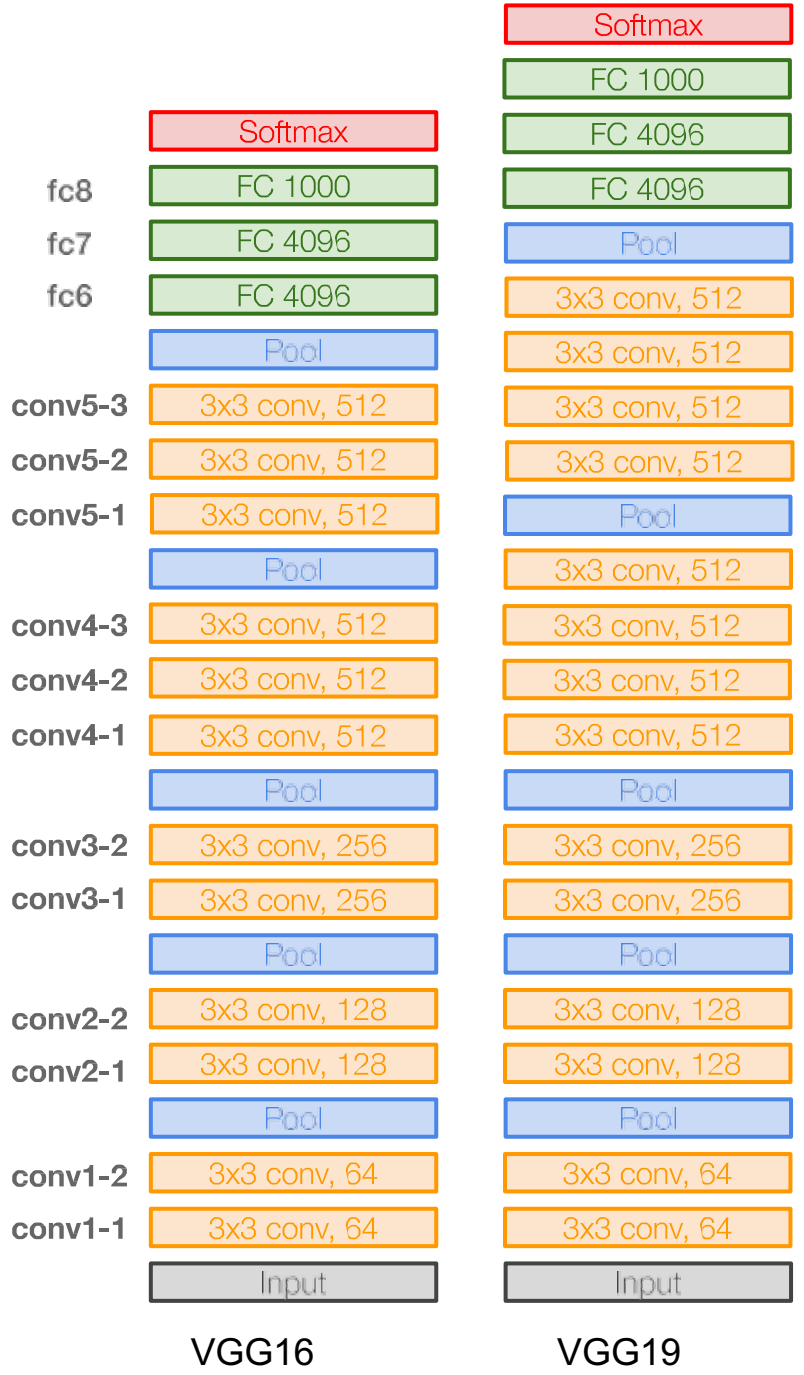
- Synchronized Batch Normalization (2018)
  - Paper: https://arxiv.org/abs/1803.08904v1
  - As the training scale went big, some adjustments to BN were necessary. The natural evolution of BN is Synchronized BN(Synch BN). Synchronized means that the mean and variance is not updated in each GPU separately.
  - In multi-worker setups, Synch BN indicates that the mean and standard-deviation are communicated across workers
  - They first calculate $\sum_{i=1}^{N} x_i^2$, and $(\sum_{i=1}^{N} x_i)^2$ individually on each device. Then the global sums are calculated by applying the reduce parallel programing technique (https://www.youtube-nocookie.com/embed/prLb1MbAm8M)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



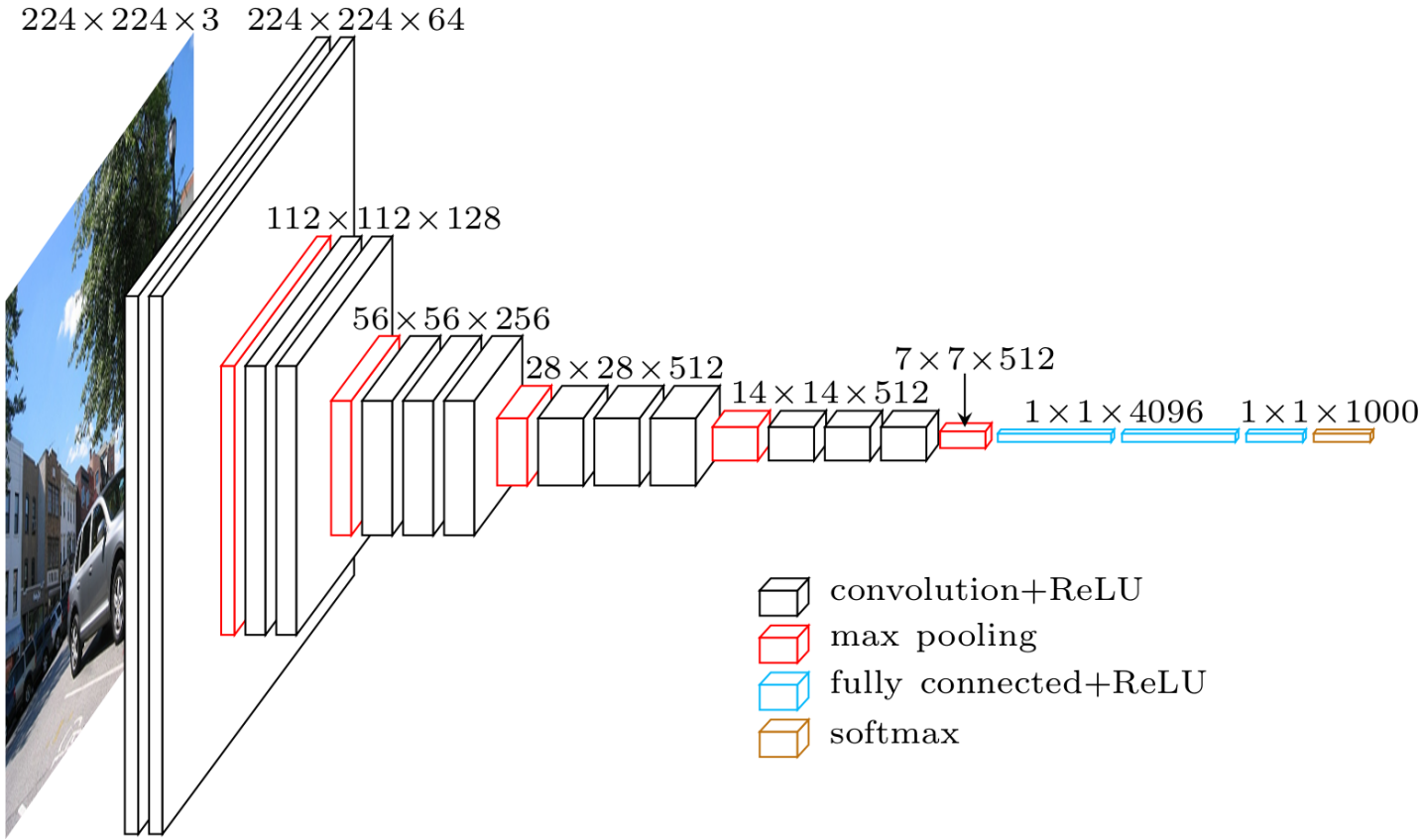
VGGNet (2014)

- Simonyan, K.; Zisserman, A., ***Very deep convolutional networks for large-scale image recognition***, International Conference on Learning Representations (2015)
  - Stack many convolutional layers before pooling
  - Use “same” convolutions (3x3) to avoid resolution reduction
  - Up to 19 layers
  - The main principle is that a stack of three 3×3 conv. layers are similar to a single 7×7 layer. And maybe even better! Because they use three non-linear activations in between (instead of one), which makes the function more discriminative.
    - $3*(3^2)C^2=27\times C^2$ weights, compared to a 7×7 conv. layer that would require $1*(7^2)\,C^2 = 49C^2$ parameters (81% more)



VGG16

- VGG16
  - the input is a 224x224x3 tensor (that means a 224x224 pixel RGB image)



VGG16

Thank You

Address:
ENG257, SJSU

Email Address:
Kaikai.liu@sjsu.edu