Topics:

- Transfer Learning
- ConvNeXt
- Recurrent Neural Networks (RNNs)

CS 4644-DL / 7643-A

ZSOLT KIRA

- **Project Proposal – In grace period!**

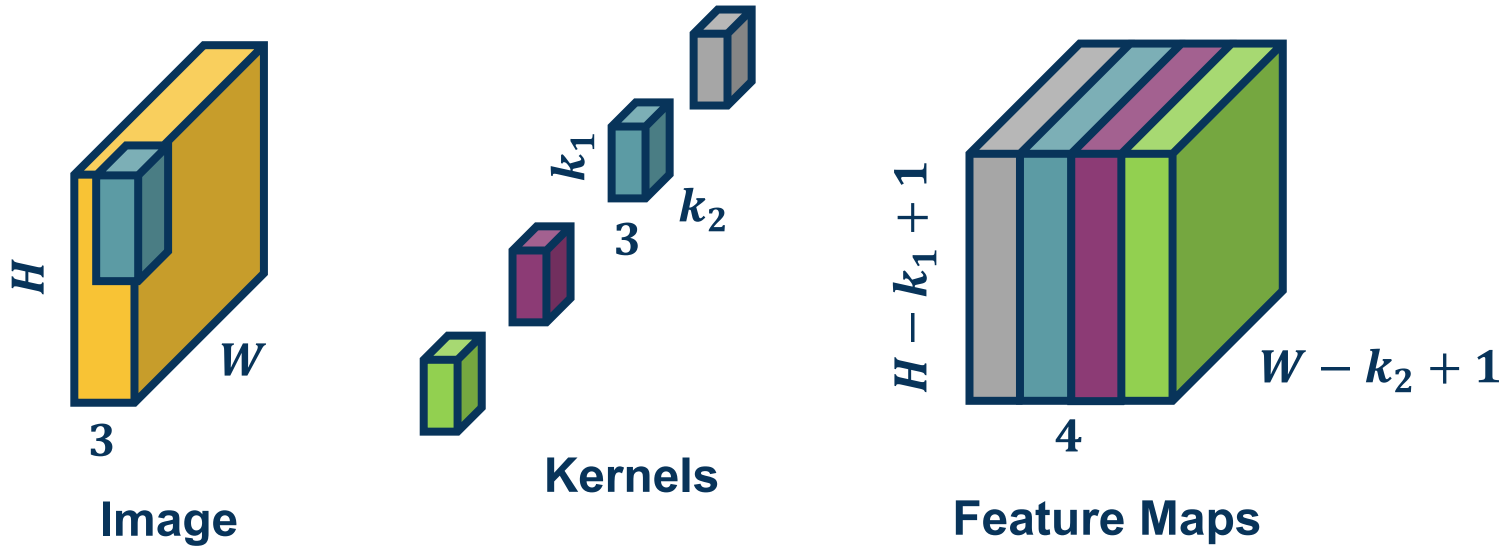
- **Assignment 2 – Due June. 22nd**
  - Implement convolutional neural networks
  - Resources (in addition to lectures):
    - DL book: Convolutional Networks
    - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
    - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
    - **HW2 Tutorial (@176)**
    - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6) (https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0)

- **Meta Office Hours – First one on language modeling Wed. 3pm ET**

- **GPU resources**: Google Colab, PACE-ICE
  - Google Cloud posted soon (for projects)

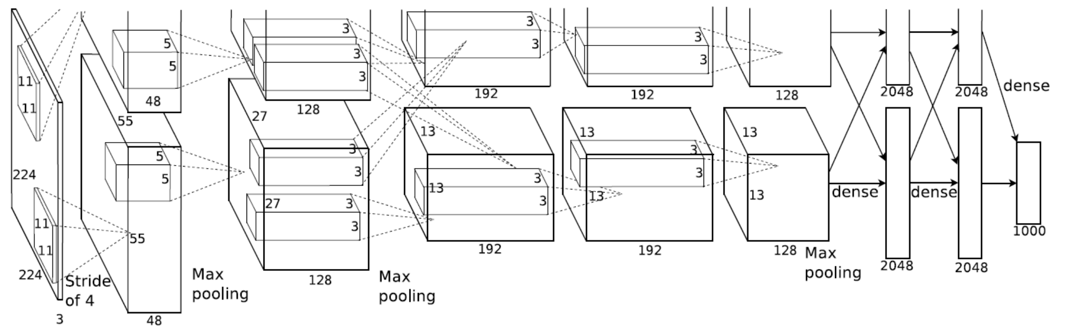
Number of parameters with N filters is: $N * (k_1 * k_2 * 3 + 1)$

- Example:
  $k_1 = 3, k_2 = 3,\ N = 4\ input\ channels = 3$, then $(3 * 3 * 3 + 1) * 4 = 112$



AlexNet – Layers and Key Aspects

Full (simplified) AlexNet architecture:
[224x224x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

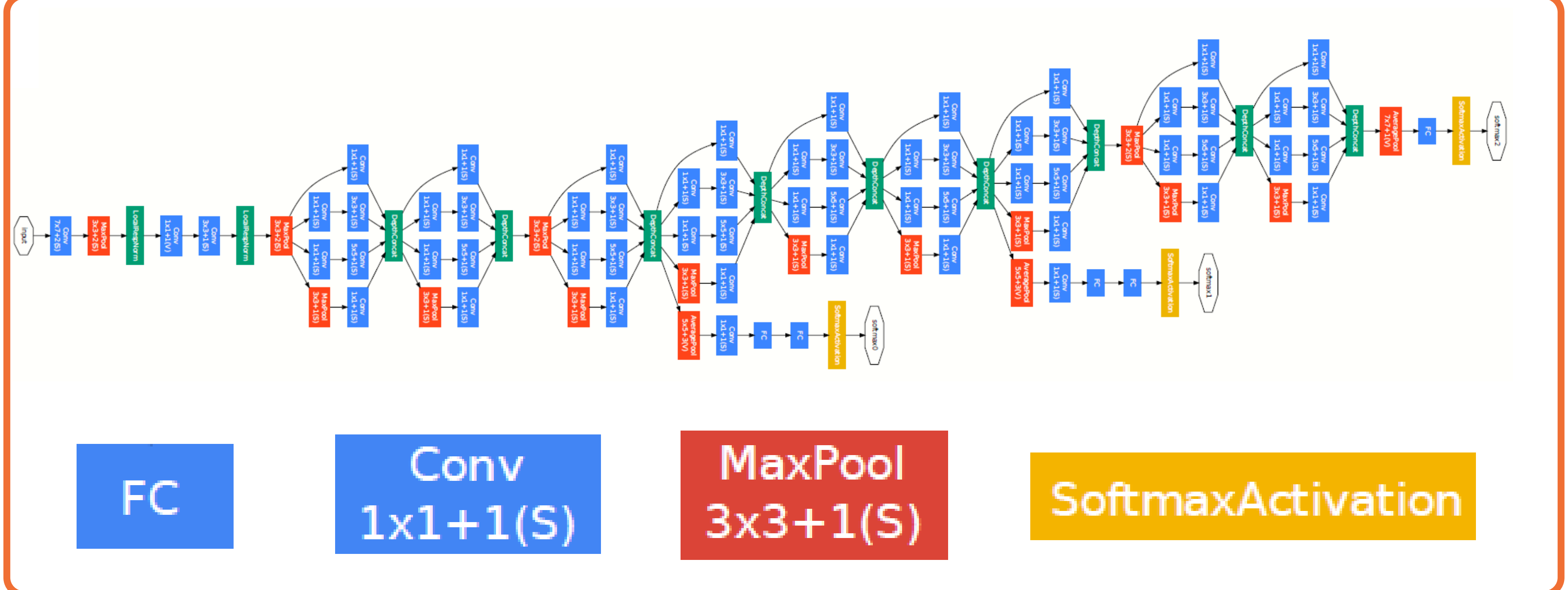


Key aspects:

- ReLU instead of sigmoid or tanh
- Specialized normalization layers
- PCA-based data augmentation
- Dropout
- Ensembling

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

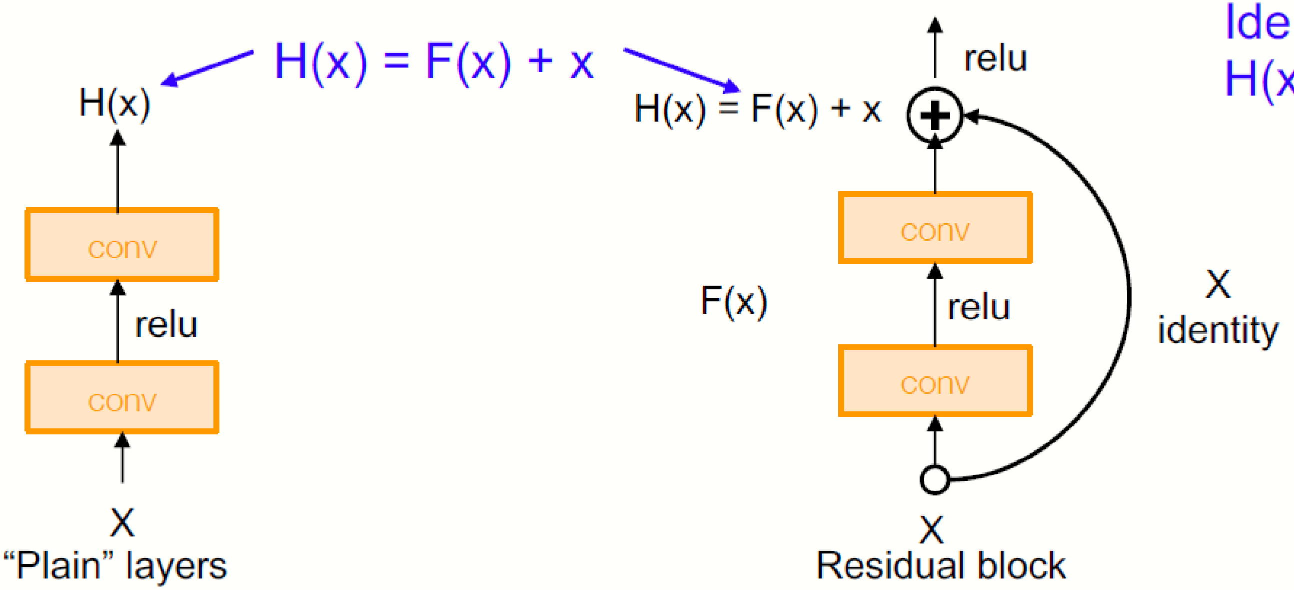
But have become **deeper and more complex**



From: Szegedy et al. Going deeper with convolutions

[He et al., 2015]

Solution: Change the network so learning identity functions as extra layers is easy



Identity mapping:
$H(x) = x$ if $F(x) = 0$

Use layers to fit **residual**
$F(x) = H(x) - x$
instead of
$H(x)$ directly

Skip Connections

layer name	output size	18-layer	34-layer			
conv1	112×112					
conv2_x	56×56					
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$			
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$			
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$			
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$			
	1×1			ave		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

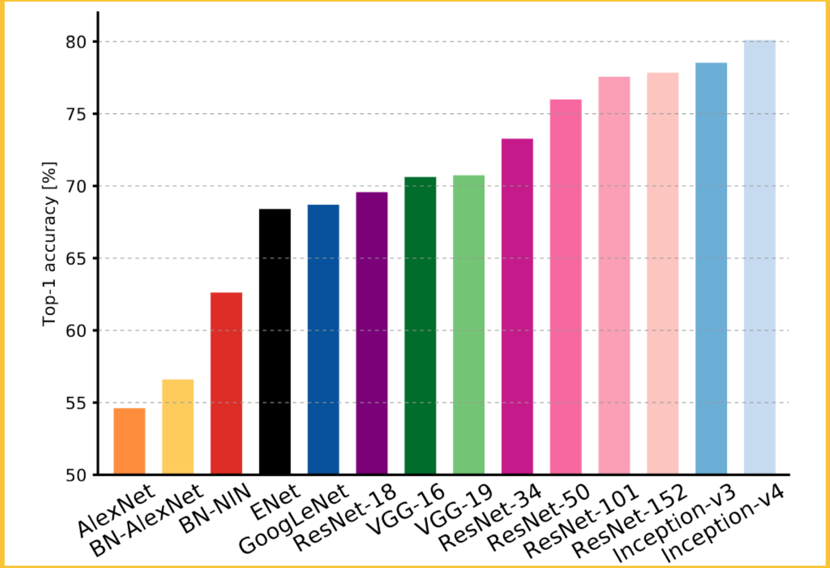
```
>>> import torch
>>> from torchvision.models import resnet18
>>> model = resnet18()
>>> summary(model2, (3, 224, 224), device='cpu')
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,408
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]          36,864
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,864
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
       BasicBlock-11           [-1, 64, 56, 56]               0
           Conv2d-12           [-1, 64, 56, 56]          36,864
      BatchNorm2d-13           [-1, 64, 56, 56]             128
             ReLU-14           [-1, 64, 56, 56]               0
           Conv2d-15           [-1, 64, 56, 56]          36,864
      BatchNorm2d-16           [-1, 64, 56, 56]             128
             ReLU-17           [-1, 64, 56, 56]               0
       BasicBlock-18           [-1, 64, 56, 56]               0
           Conv2d-19          [-1, 128, 28, 28]          73,728
      BatchNorm2d-20          [-1, 128, 28, 28]             256
             ReLU-21          [-1, 128, 28, 28]               0
           Conv2d-22          [-1, 128, 28, 28]         147,456
      BatchNorm2d-23          [-1, 128, 28, 28]             256
```

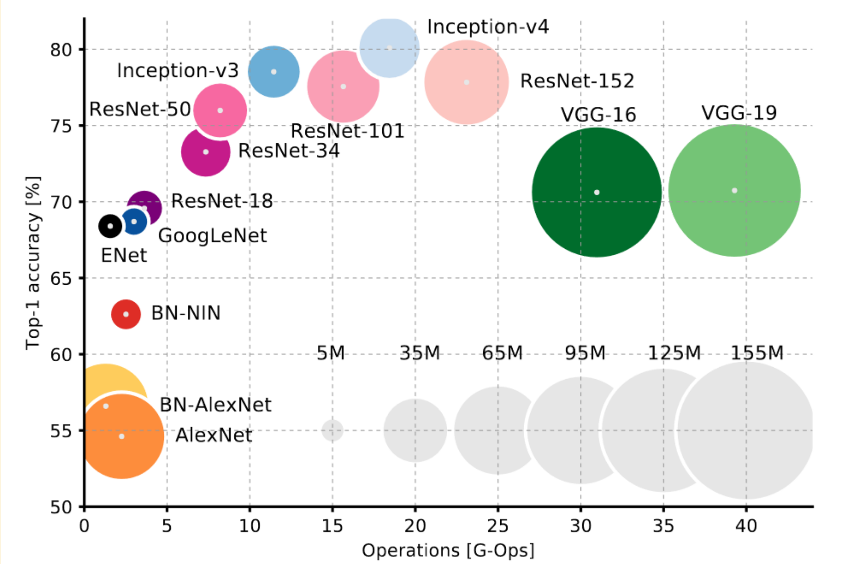
ResNet Details

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Computational Complexity



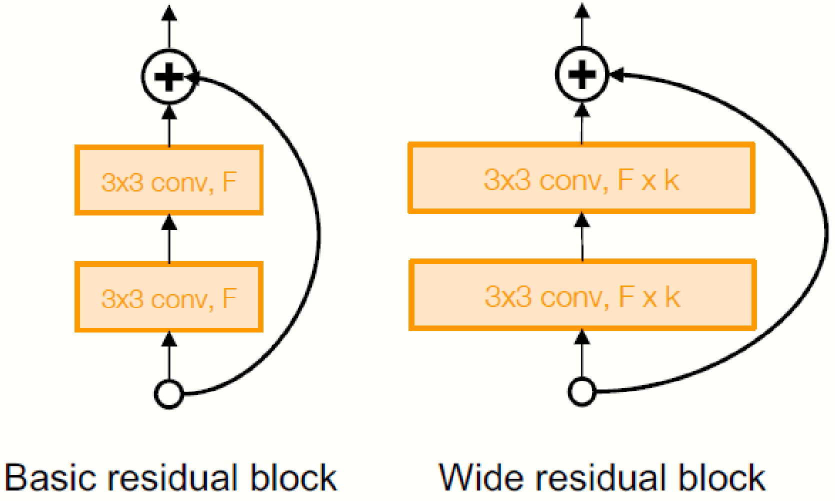


From: An Analysis Of Deep Neural Network Models For Practical Applications

Wide Residual Networks

[Zagoruyko et al. 2016]

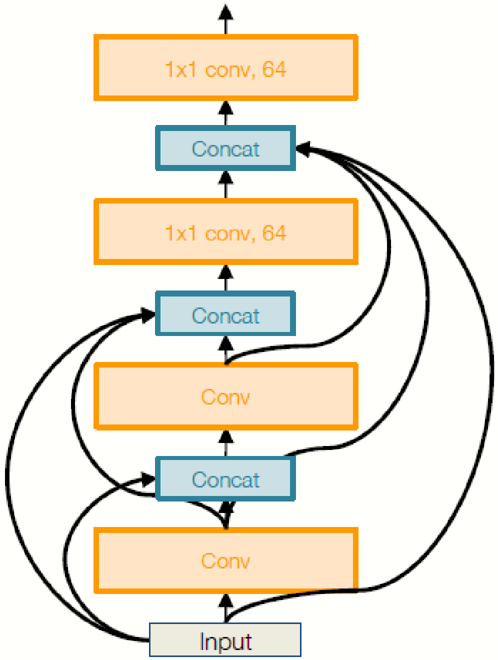
- Argues that residuals are the important factor, not depth
- Use wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



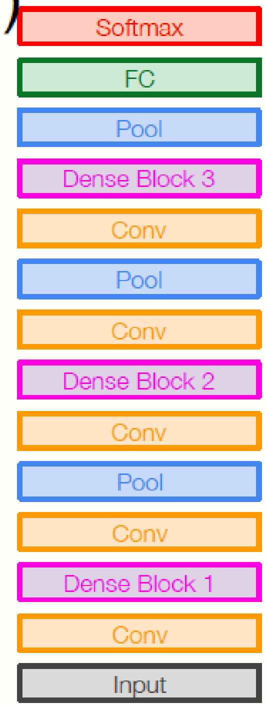
Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



Dense Block



- Convolutional neural networks (CNNs) stack pooling, convolution, non-linearities, and fully connected (FC) layers

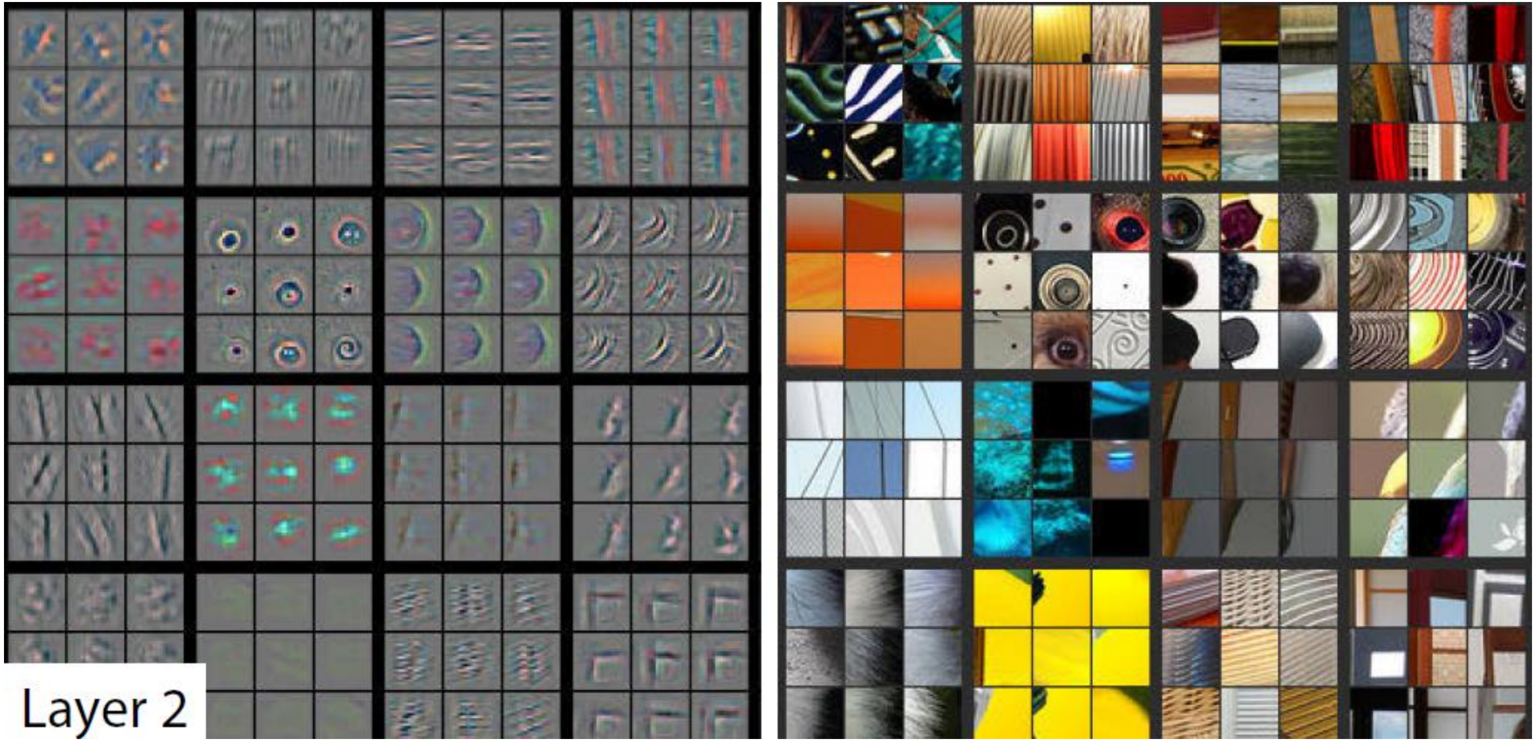
- Feature engineering => architecture engineering!
  - Tons of small details and tips/tricks
  - Considerations: Memory, compute/FLO, dimensionality reduction, diversity of features, number of parameters/capacity, etc.

Summary

What do CNNs Learn?

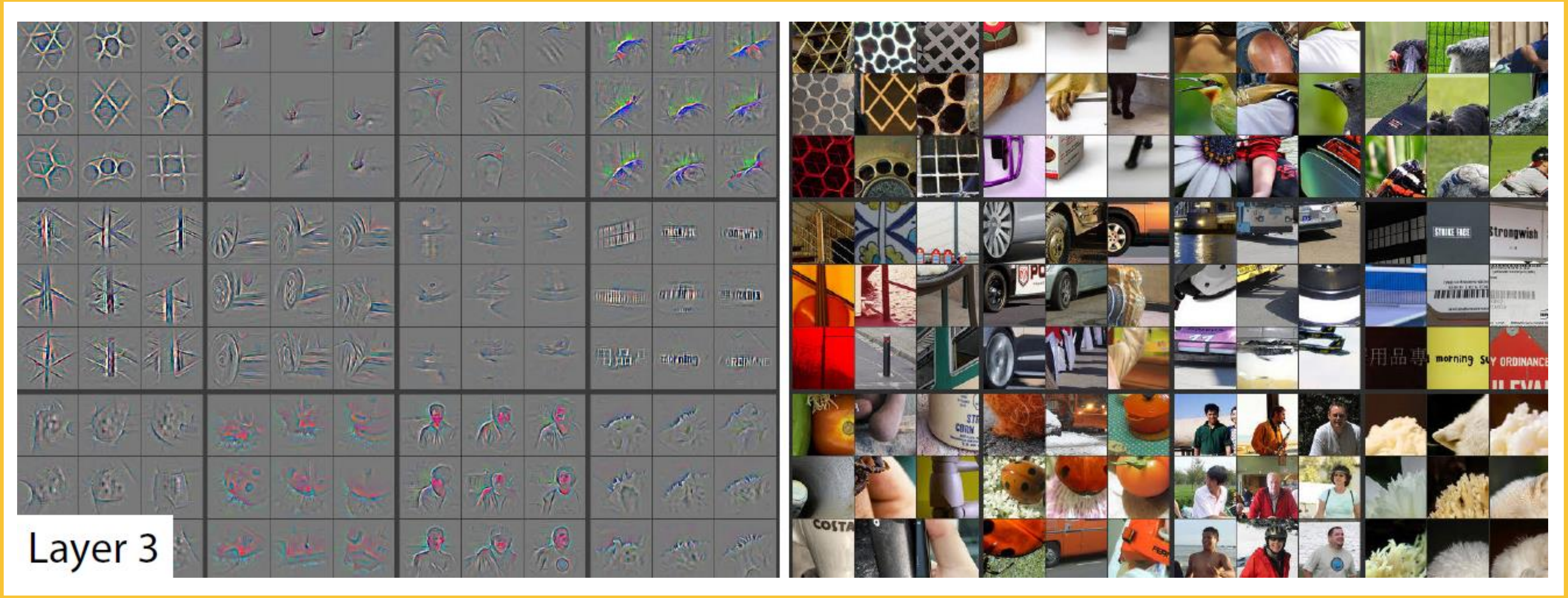
Georgia Tech

VGG Layer-by-Layer Visualization



From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

VGG Layer-by-Layer Visualization

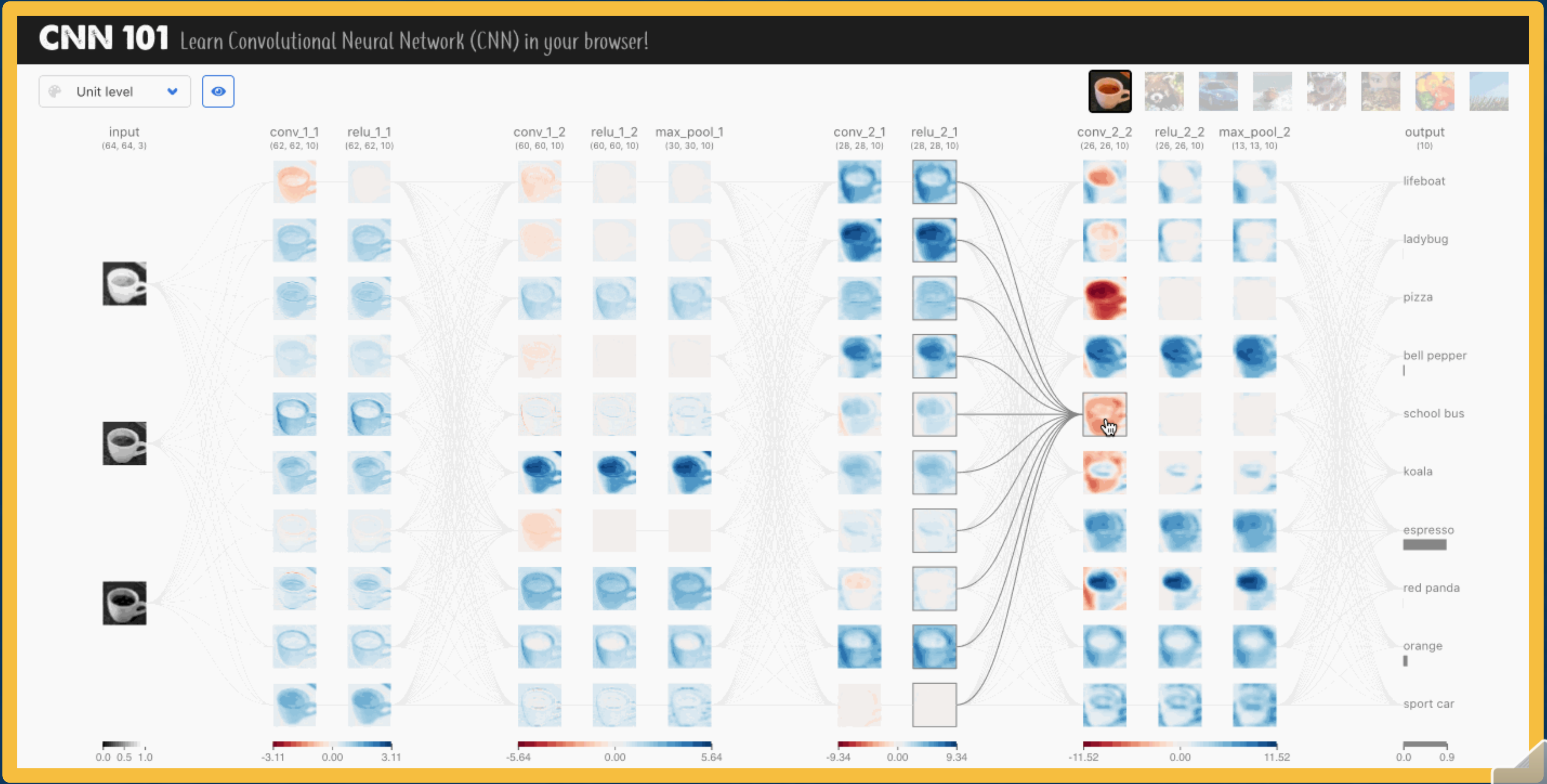


From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

VGG Layer-by-Layer Visualization

From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

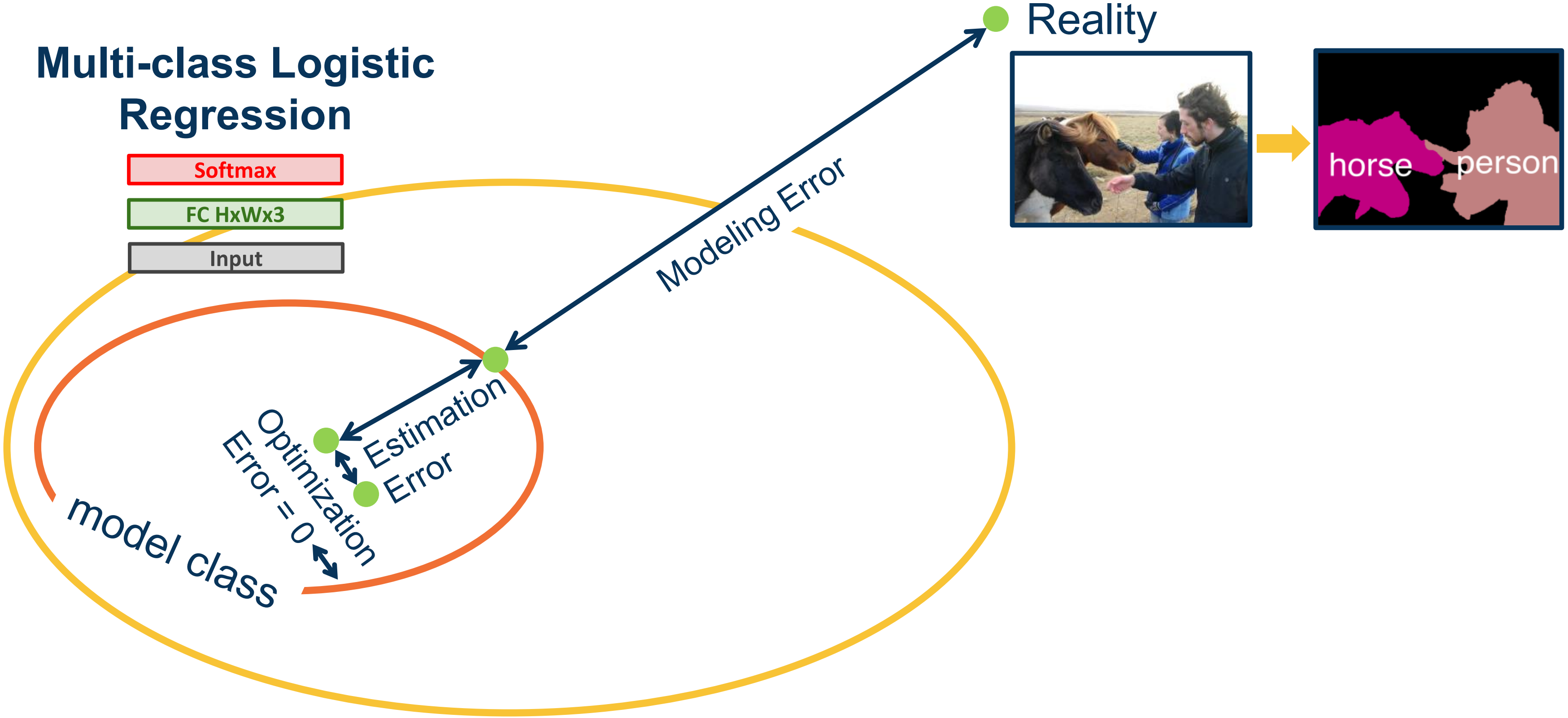
CNN101 and CNN Explainer



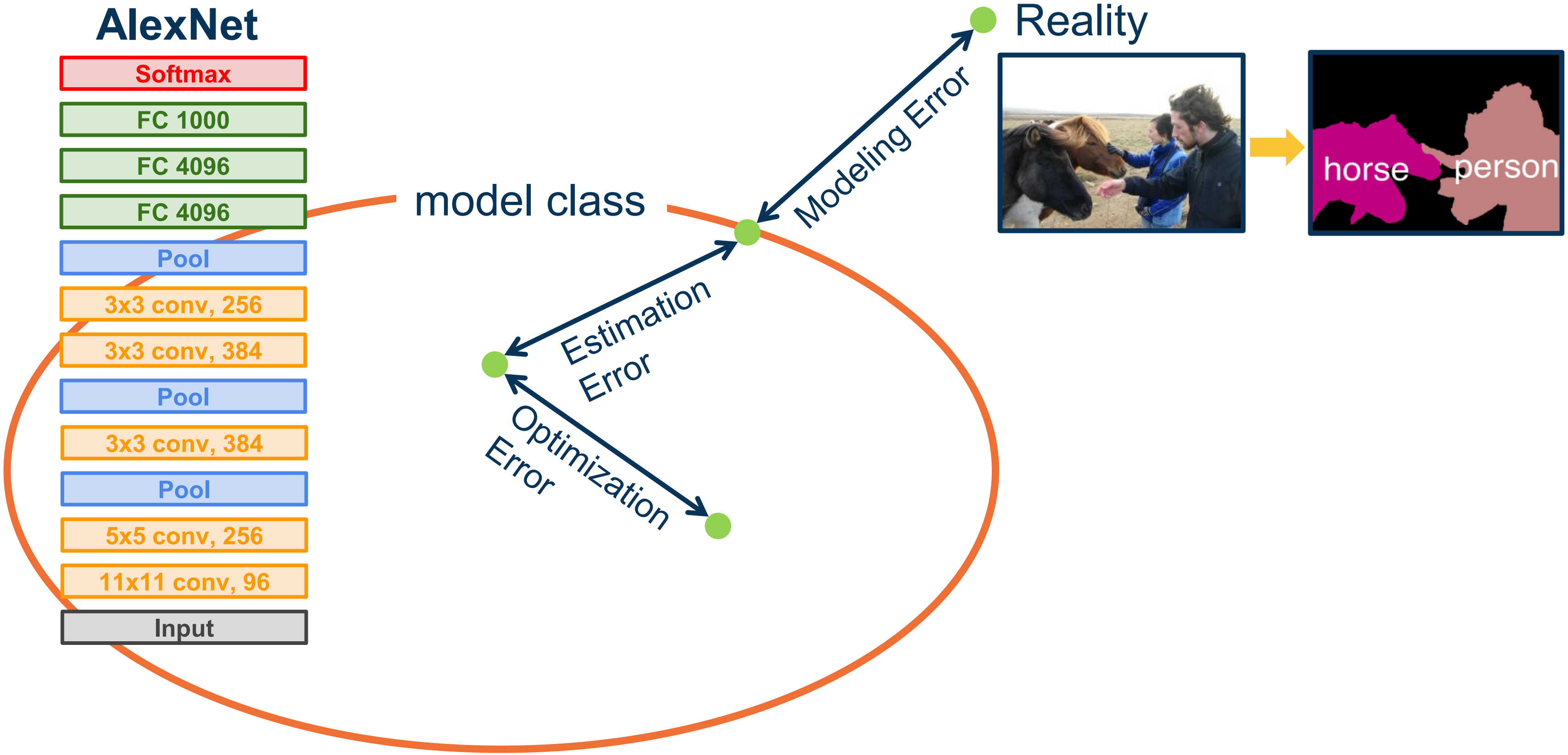
https://poloclub.github.io/cnn-explainer/ *https://fredhohman.com/papers/cnn101*

Transfer Learning & Generalization

Multi-class Logistic Regression

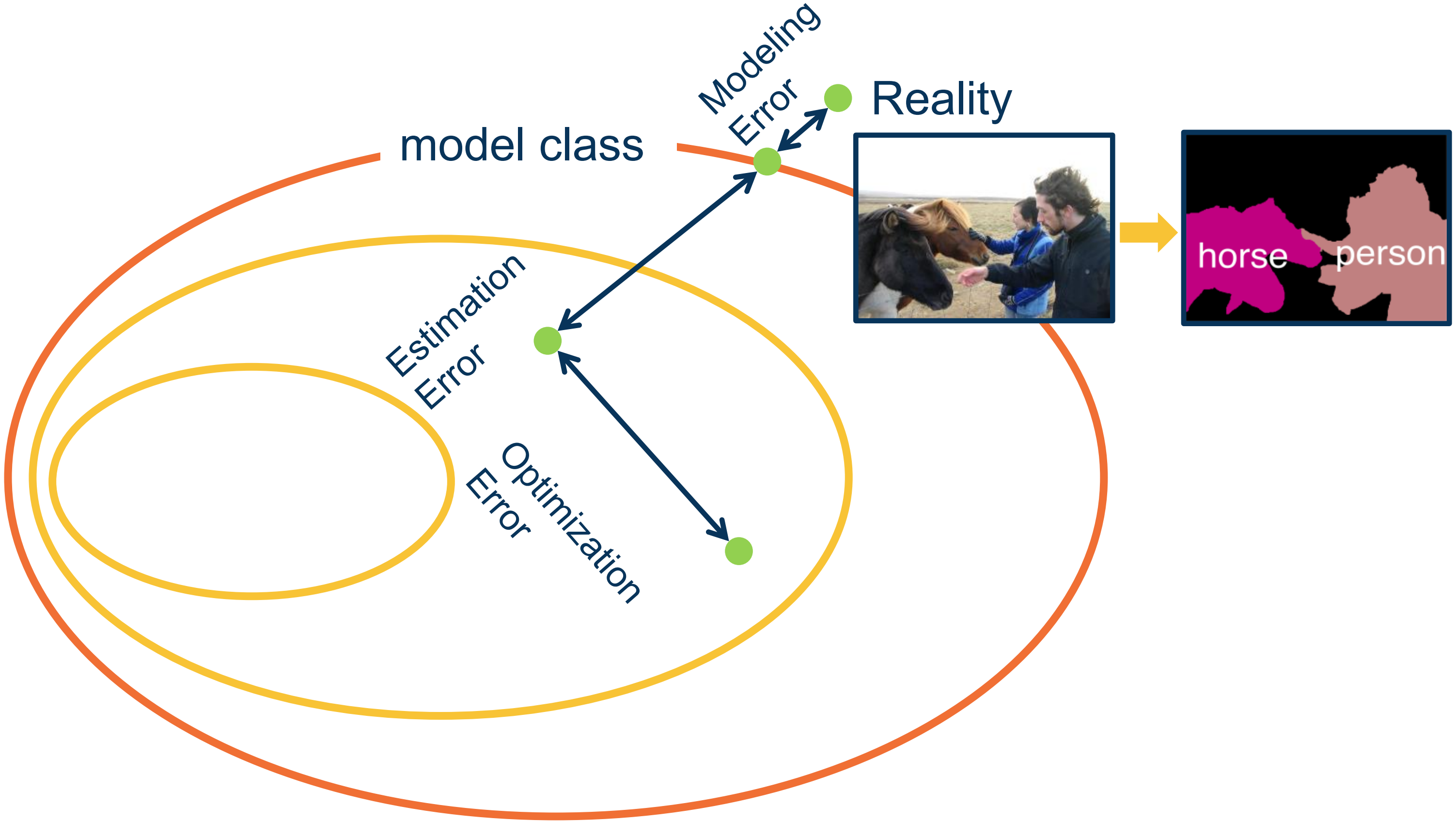


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Generalization

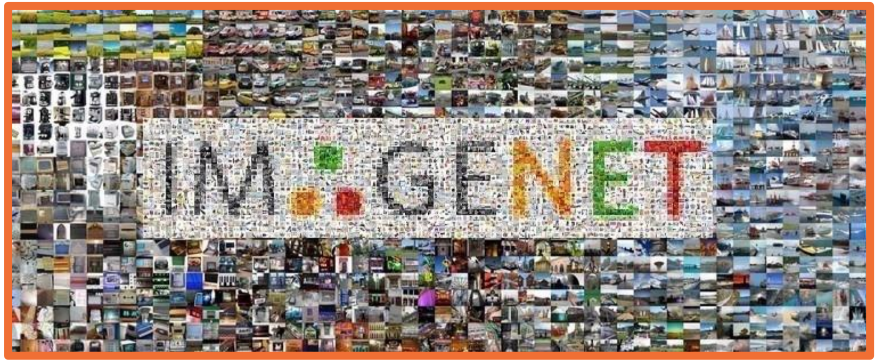


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

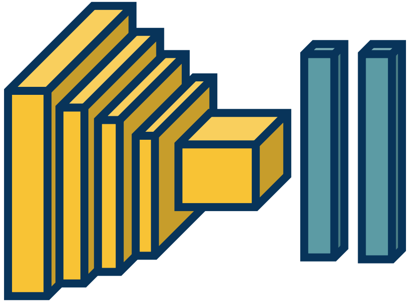
Generalization

What if we don’t have enough data?

Step 1: Train on large-scale dataset



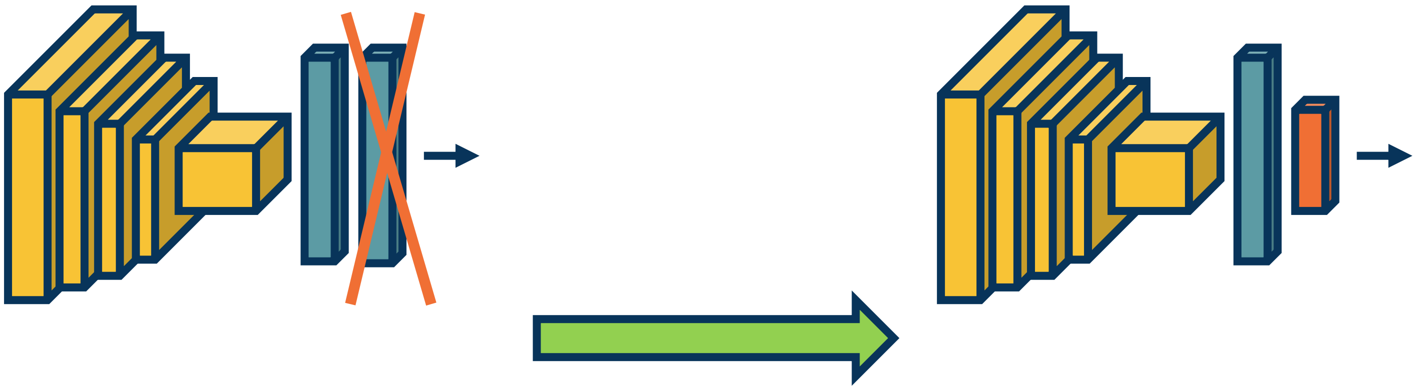
Input Image



→ Predictions

Convolutional Neural Networks

Step 2: Take your custom data and **initialize** the network with weights trained in Step 1



Replace last layer with new fully-connected for output nodes per new category

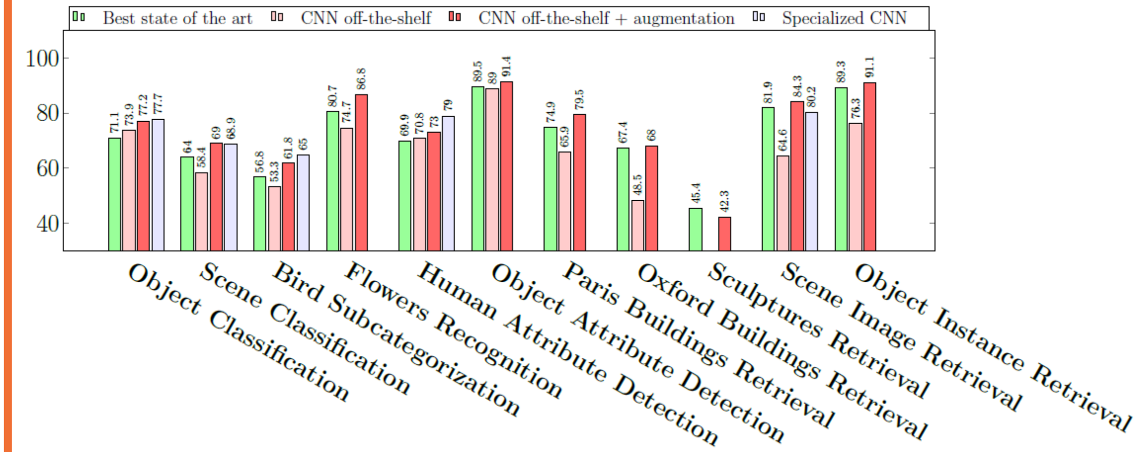
Step 3: (Continue to) train on new dataset

- **Finetune:** Update all parameters
- **Freeze** feature layer: Update only last layer weights (used when not enough data)

Replace last layer with new fully-connected for output nodes per new category

This works extremely well! It was surprising upon discovery.

- Features learned for 1000 object categories will work well for 1001st!
- Generalizes even across tasks (classification to object detection)



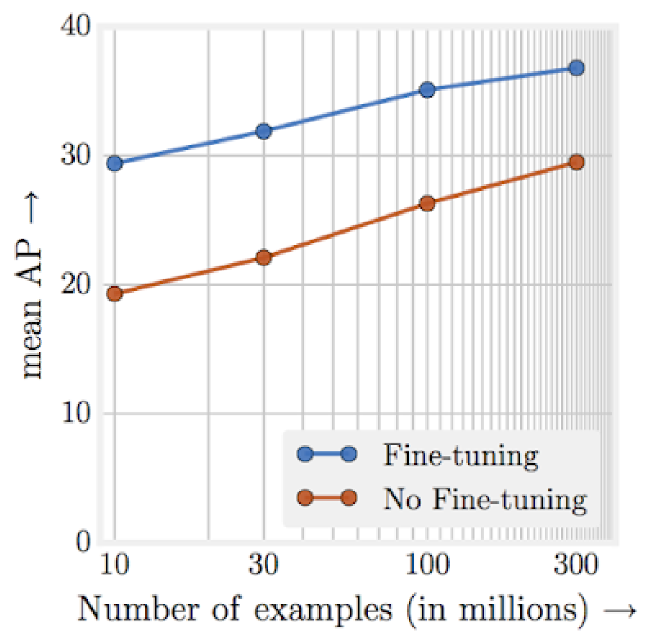
From: Razavian et al., CNN Features off-the-shelf: an Astounding Baseline for Recognition

Learning with Less Labels

But it doesn't always work that well!

- If the **source** dataset you train on is very different from the **target** dataset, transfer learning is not as effective
- If you have enough data for the target domain, it just results in faster convergence
  - See He et al., "Rethinking ImageNet Pre-training"

Effectiveness of More Data



From: Revisiting the Unreasonable Effectiveness of Data
https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

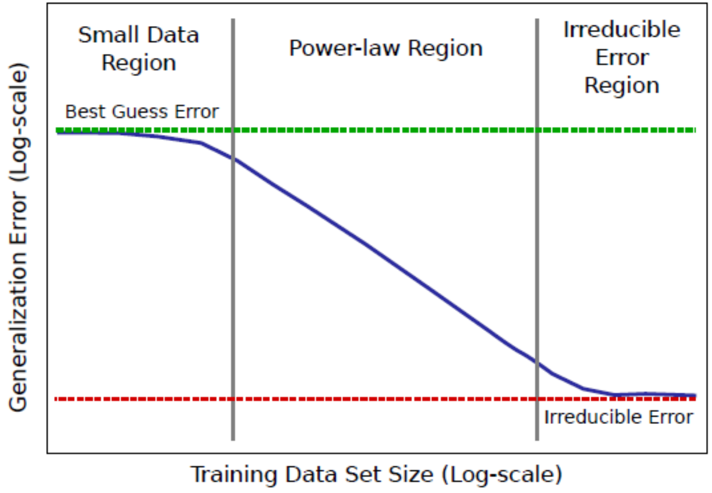


Figure 6: Sketch of power-law learning curves

From: Hestness et al., Deep Learning Scaling Is Predictable

Georgia Tech

There is a large number of different low-labeled settings in DL research

Setting	Source	Target	Shift Type
Semi-supervised	Single labeled	Single unlabeled	None
Domain Adaptation	Single labeled	Single unlabeled	Non-semantic
Domain Generalization	Multiple labeled	Unknown	Non-semantic
Cross-Task Transfer	Single labeled	Single unlabeled	Semantic
Few-Shot Learning	Single labeled	Single few-labeled	Semantic
Un/Self-Supervised	Single unlabeled	Many labeled	Both/Task

Non-Semantic Shift


Semantic Shift


Data Augmentation

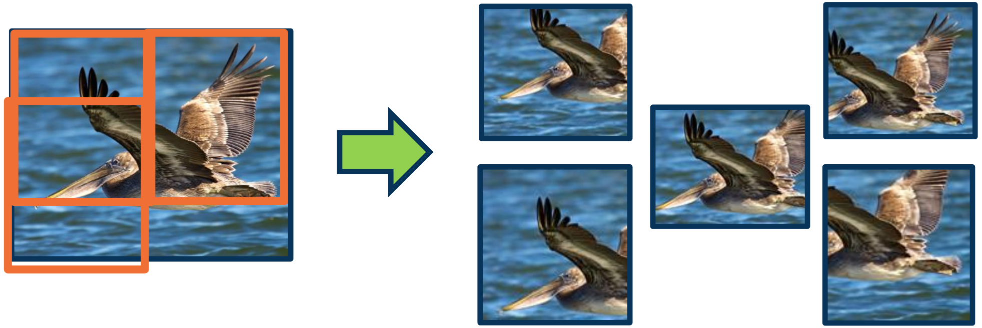
Data augmentation – Performing a range of **transformations** to the data

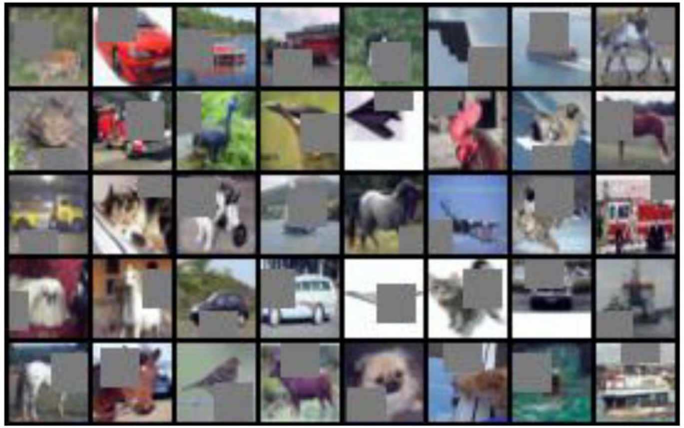
- This essentially "**increases**" your dataset
- Transformations should not change meaning of the data (or label has to be changed as well)

Simple example: Image Flipping

Random crop

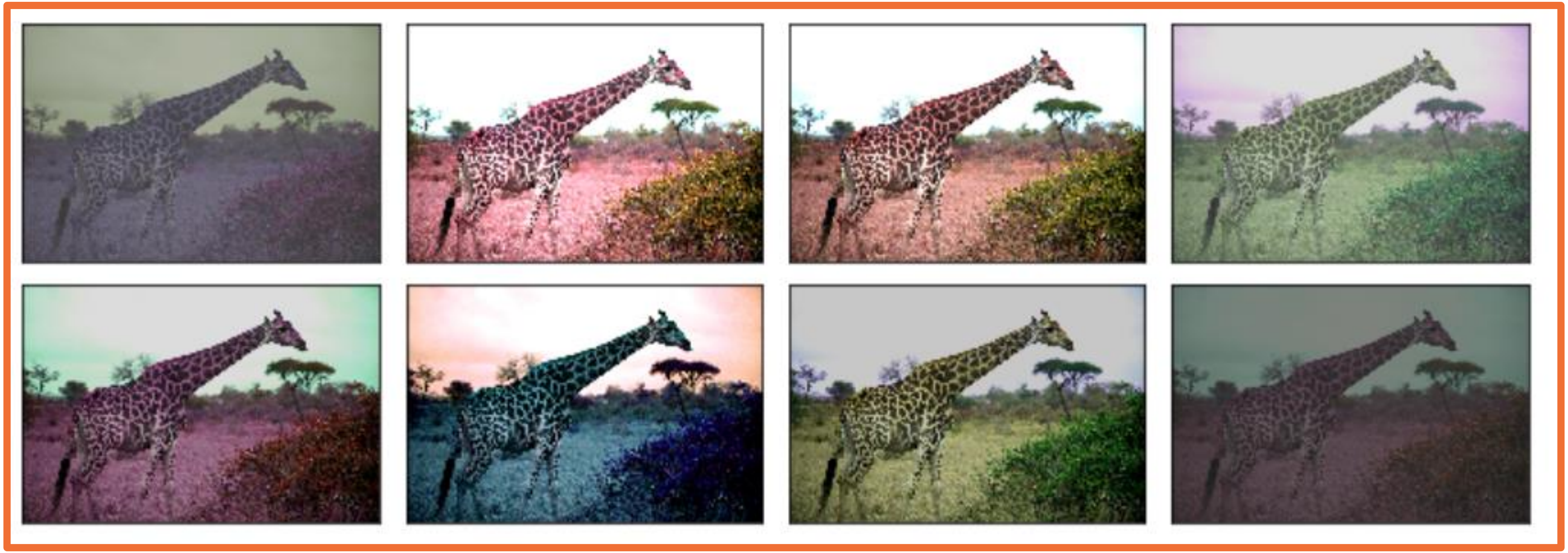
- Take different crops during training
- Can be used during inference too!





CutMix

Color Jitter



From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

We can apply **generic affine transformations:**

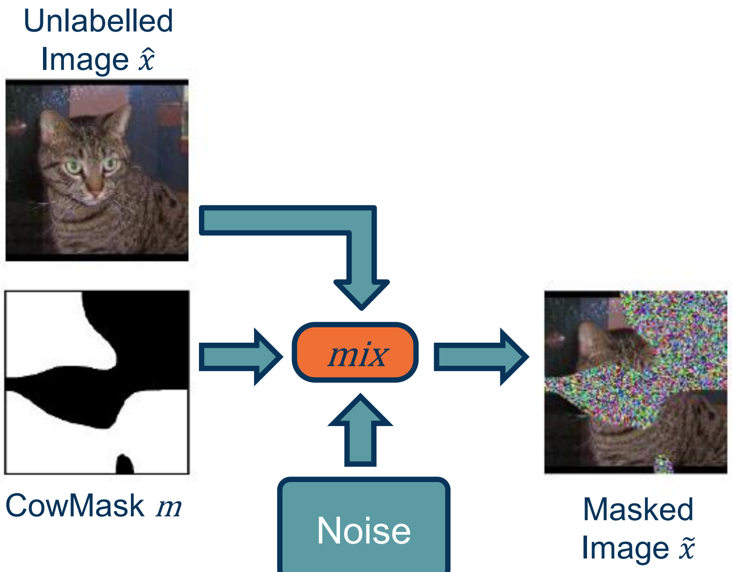
- **Translation**
- **Rotation**
- **Scale**
- **Shear**

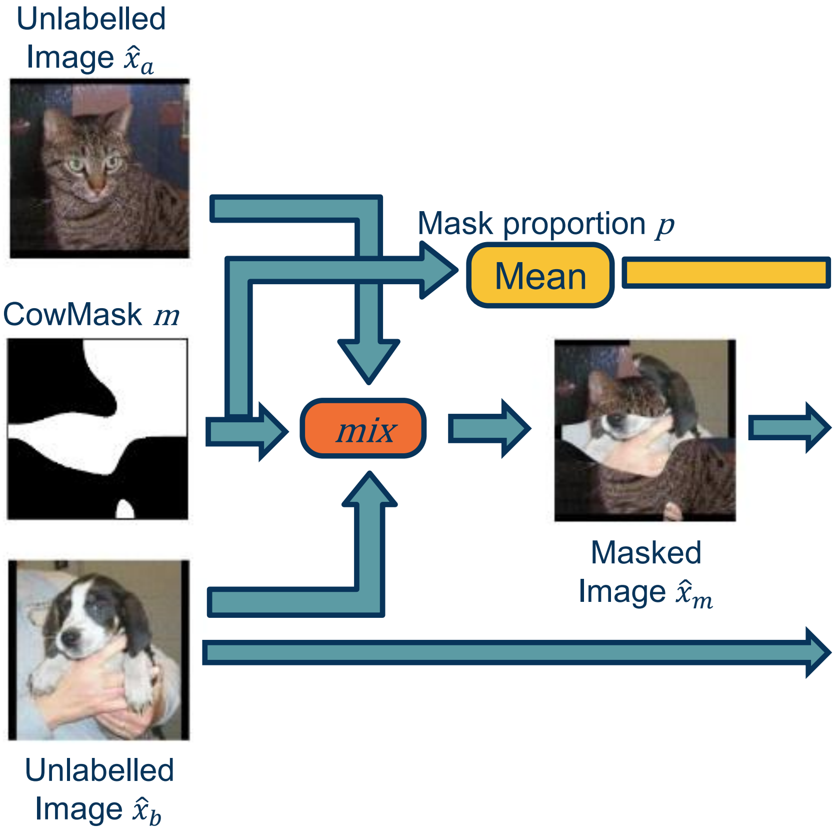
We can **combine these transformations** to add even more variety!

From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

Combining Transformations

Other Variations





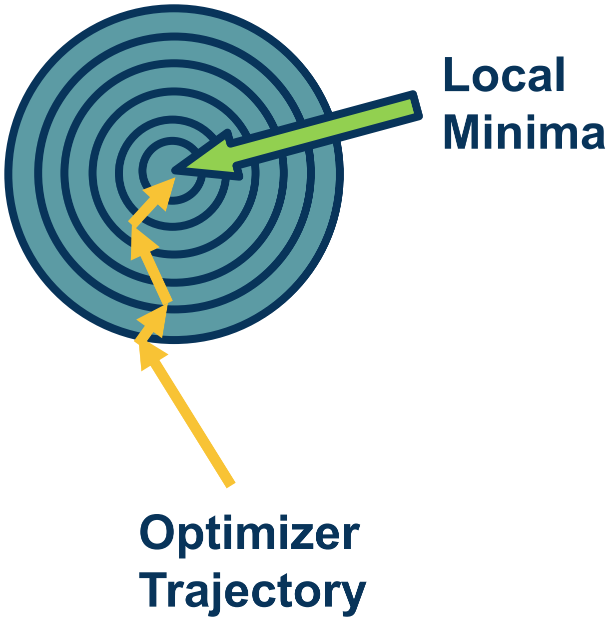
CowMix

From French et al., "Milking CowMask for Semi-Supervised Image Classification"

Georgia Tech

The Process of Training Neural Networks

- Training deep neural networks is an art form!
- Lots of things matter (together) – the key is to find a combination that works
- **Key principle**: Monitoring everything to understand what is going on!
  - Loss and accuracy curves
  - Gradient statistics/characteristics
  - Other aspects of computation graph



Proper Methodology

Always start with **proper methodology**!

- **Not uncommon** even in published papers to get this wrong

Separate data into: **Training, validation, test set**

- **Do not look** at test set performance until you have decided on everything (including hyper-parameters)

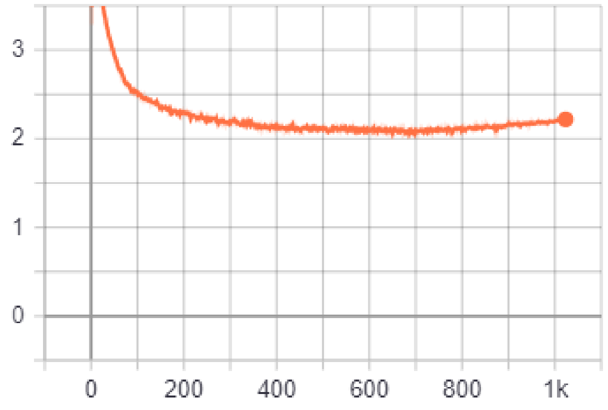
Use **cross-validation** to decide on hyper-parameters if amount of data is an issue

Check the bounds of your loss function

- E.g. cross-entropy ranges from $[\mathbf{0}, \infty]$
- Check initial loss at small random weight values
  - E.g. $-\log(p)$ for cross-entropy, where $p = 0.5$

Another example: Start without regularization and make sure loss goes up when added

Key Principle: Simplify the dataset to make sure your model can properly (over)-fit before applying regularization



Validation Loss

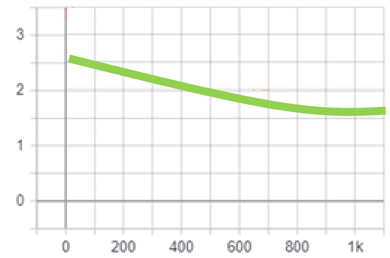
Change in loss indicates speed of learning:

- Tiny loss change -> too small of a learning rate
- Loss (and then weights) turn to NaNs -> too high of a learning rate

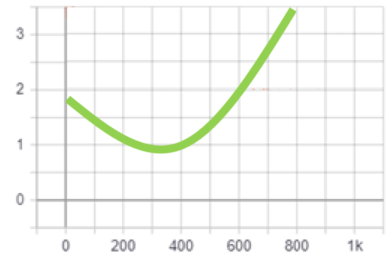
Other bugs can also cause this, e.g.:

- Divide by zero
- Forgetting the log!

In pytorch, use autograd's detect anomaly to debug

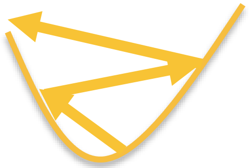


Learning Rate Too Low



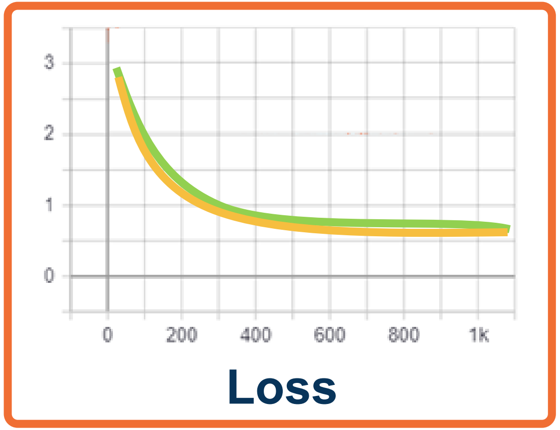
Learning Rate Too High

```
with autograd.detect_anomaly():
  output = model(input)
  loss = criterion(output, labels
  loss.backward()
```



Loss and Not a Number (NaN)

- Classic machine learning signs of under/overfitting still apply!
- **Over-fitting**: Validation loss/accuracy starts to get worse after a while
- **Under-fitting:** Validation loss very close to training loss, or both are high
- **Note:** You can have higher training loss!
  - Validation loss has no regularization
  - Validation loss is typically measured at the end of an epoch



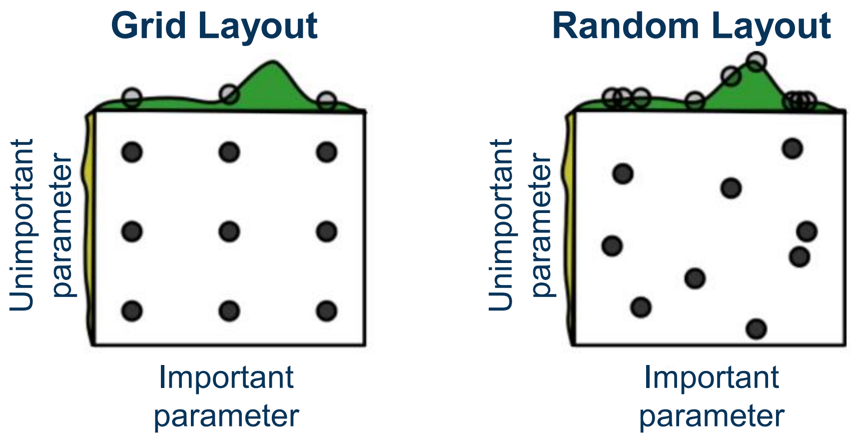
Overfitting

Many hyper-parameters to tune!

- Learning rate, weight decay crucial
- Momentum, others more stable
- **Always tune** hyper-parameters; even a good idea will fail un-tuned!

Start with coarser search:

- E.g. learning rate of {0.1, 0.05, 0.03, 0.01, 0.003, 0.001, 0.0005, 0.0001}
- Perform finer search around good values



From: Bergstra et al., "Random Search for Hyper-Parameter Optimization", JMLR, 2012

Automated methods are OK, but intuition (or random) can do well given enough of a tuning budget

Hyper-Parameter Tuning

Georgia Tech

Inter-dependence of Hyperparameters

Note that hyper-parameters and even module selection are **interdependent**!

Examples:

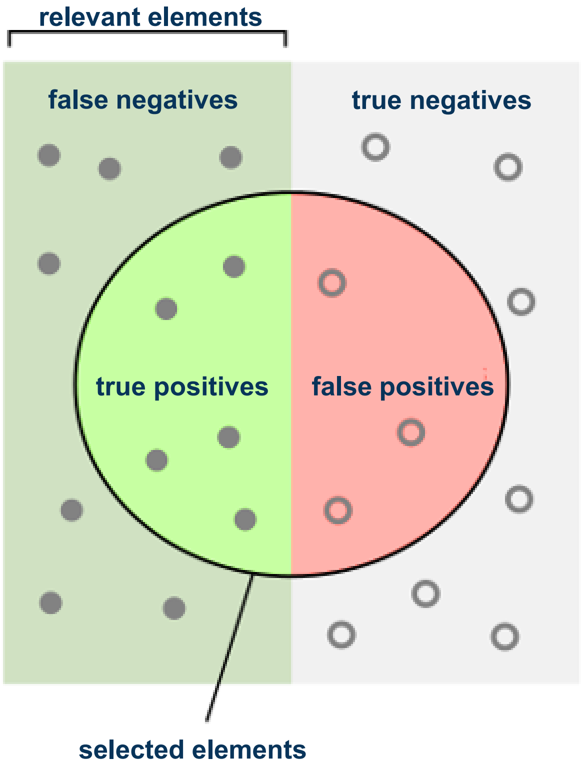
- Batch norm and dropout **maybe not be needed together** (and sometimes the combination is worse)
- The learning rate should be **changed proportionally to batch size** – increase the learning rate for larger batch sizes
    - **One interpretation:** Gradients are more reliable/smoother

Note that we are optimizing a **loss function**

What we actually care about is **typically different metrics that we can't differentiate**:

- Accuracy
- Precision/recall
- Other specialized metrics

The relationship between the two can be complex!



From https://en.wikipedia.org/wiki/Precision_and_recall

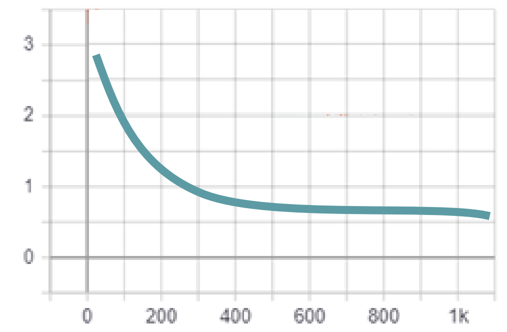
- **Example:** Cross entropy loss

$$L = -log\, P(Y = y_i | X = x_i)$$

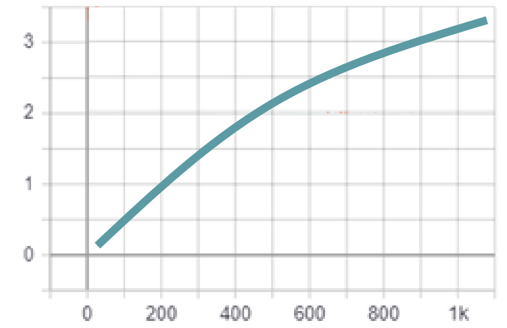
- **Accuracy** is measured based on:

$$argmax_i(P(Y = y_i\, | X = x_i))$$

- Since the correct class score only has to be slightly higher, we can have **flat loss curves but increasing accuracy**!

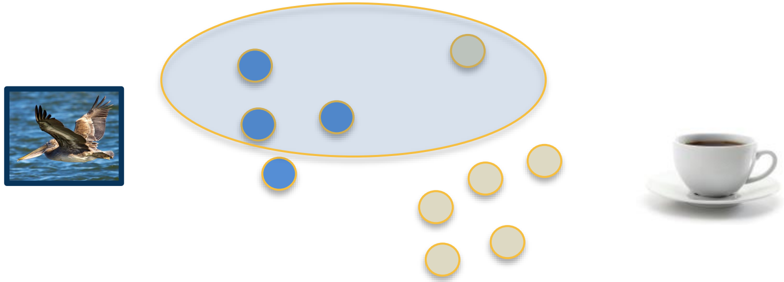


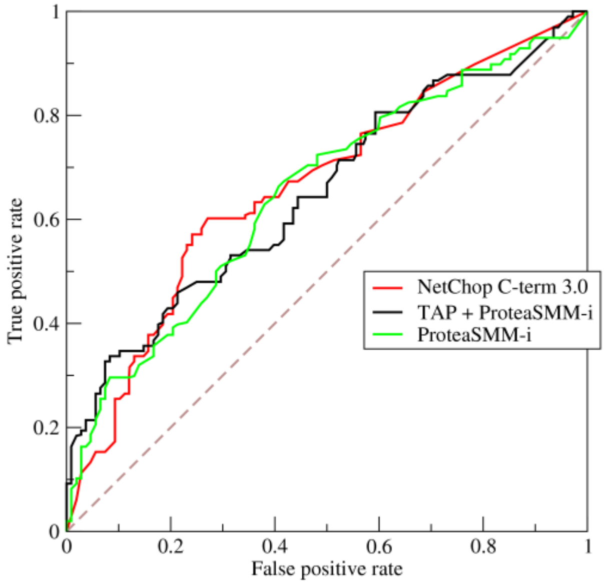
Loss



Accuracy

- **Precision/Recall curves** represent the inherent tradeoff between number of positive predictions and correctness of predictions
- **Definitions**
  - True Positive Rate: $\boldsymbol{TPR = \frac{tp}{tp+fn}}$
  - False Positive Rate: $\boldsymbol{FPR = \frac{fp}{fp+tn}}$
  - $\boldsymbol{Accuracy = \frac{tp+tn}{tp+tn+fp+fn}}$

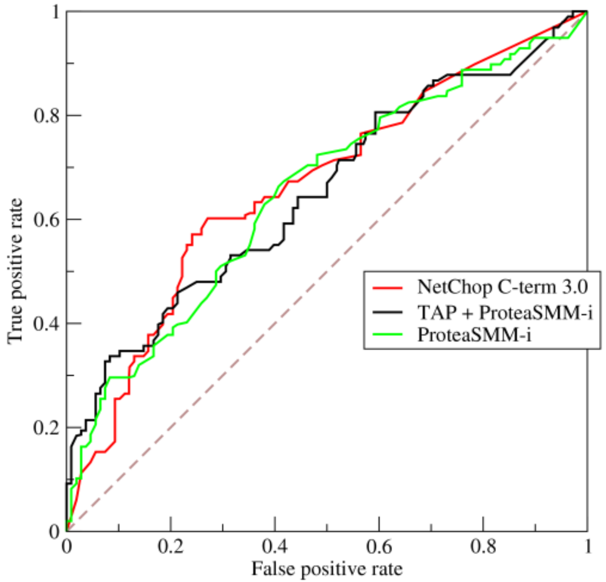




From https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Example: Precision/Recall or ROC Curves

- **Precision/Recall curves** represent the inherent tradeoff between number of positive predictions and correctness of predictions
- **Definitions**
  - True Positive Rate: $\boldsymbol{TPR = \frac{tp}{tp+fn}}$
  - False Positive Rate: $\boldsymbol{FPR = \frac{fp}{fp+tn}}$
  - $\boldsymbol{Accuracy = \frac{tp+tn}{tp+tn+fp+fn}}$
- We can obtain a **curve** by varying the (probability) threshold:
  - **Area under the curve (AUC)** common single-number metric to summarize
- Mapping between this and loss is **not simple**!

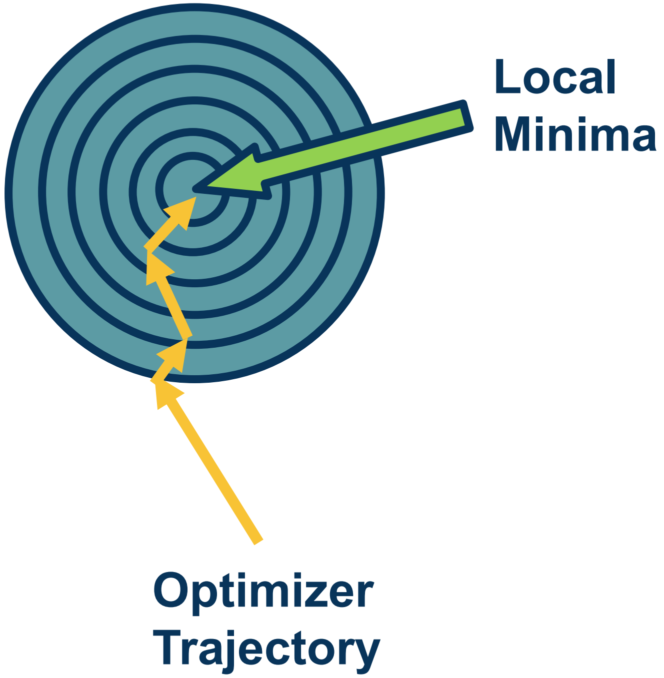


From https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Example: Precision/Recall or ROC Curves

Resource:

- A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay, Leslie N. Smith



ConvNeXt

ConvNeXt (2022)

A ConvNet for the 2020s

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie

The "Roaring 20s" of visual recognition began with the introduction of Vision Transformers (ViTs), which quickly superseded ConvNets as the state-of-the-art image classification model. A vanilla ViT, on the other hand, faces difficulties when applied to general computer vision tasks such as object detection and semantic segmentation. It is the hierarchical Transformers (e.g., Swin Transformers) that reintroduced several ConvNet priors, making Transformers practically viable as a generic vision backbone and demonstrating remarkable performance on a wide variety of vision tasks. However, the effectiveness of such hybrid approaches is still largely credited to the intrinsic superiority of Transformers, rather than the inherent inductive biases of convolutions. In this work, we reexamine the design spaces and test the limits of what a pure ConvNet can achieve. We gradually "modernize" a standard ResNet toward the design of a vision Transformer, and discover several key components that contribute to the performance difference along the way. The outcome of this exploration is a family of pure ConvNet models dubbed ConvNeXt. Constructed entirely from standard ConvNet modules, ConvNeXts compete favorably with Transformers in terms of accuracy and scalability, achieving 87.8% ImageNet top-1 accuracy and outperforming Swin Transformers on COCO detection and ADE20K segmentation, while maintaining the simplicity and efficiency of standard ConvNets.

ConvNeXt (2022)

- ***To bridge the gap between the Conv Nets and Vision Transformers (ViT)***
  - ViT, Swin Transformer has been the SOTA visual model backbone
  - Is convolutional networks really not as good as transformer models?
- Investigation
  - The author start with ResNet-50 and reimplement the CNN networks with modern designs
  - The results showing that ConvNeXt achieves beat the ViT models, again.

ConvNeXt (2022)

- What problem does this paper focus on?
  - Is this new or already explored?
  - Is this important?
  - What key applications this is relevant for?
  - What assumptions does this paper make about

ConvNeXt (2022)

- What is the key “golden nugget” – intuition, idea, etc. that leads to approach

ConvNeXt (2022)

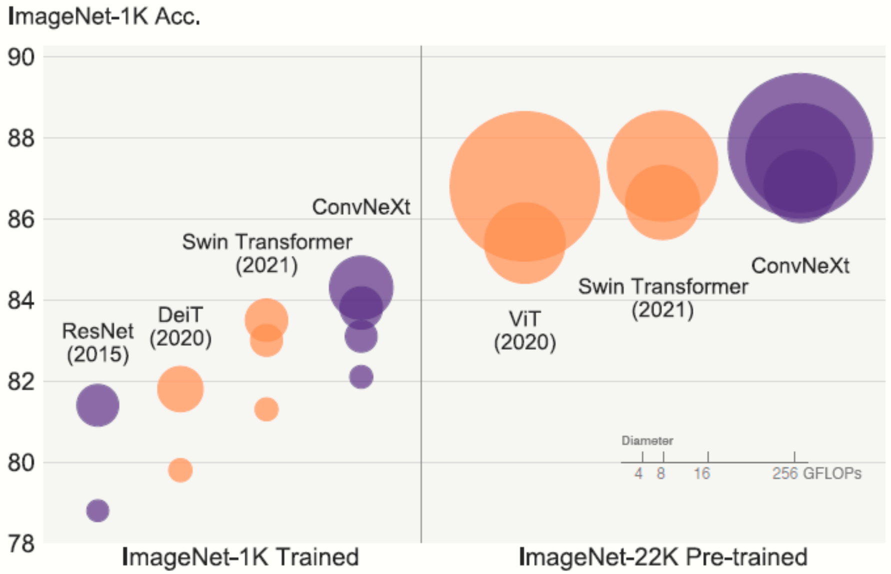


Figure 1. **ImageNet-1K classification** results for ● ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. ResNet and ViT results were obtained with improved training procedures over the original papers. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

Georgia Tech

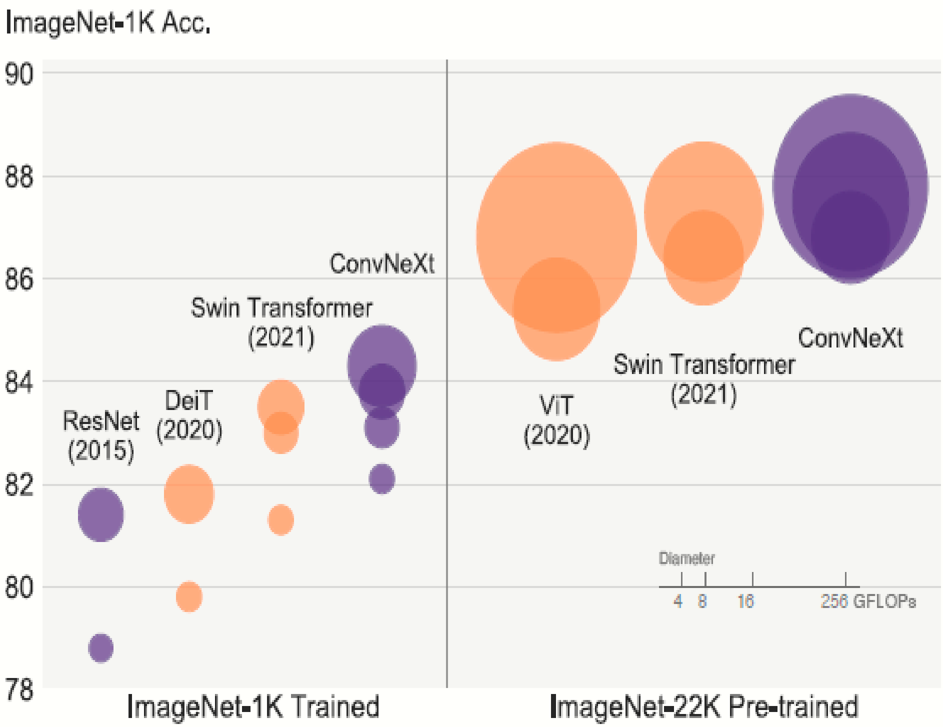
ConvNeXt (2022)

- What approach does this paper take?

Nonlinearities

ConvNeXt (2022)

- **Modern designs added:**
- Macro Design
  - Changing stage compute ratio
  - Changing stem to “patchify”
- Micro Design
  - ReLU -> GELU
  - Fewer activation functions
  - Fewer normalization layers
  - BatchNorm -> LayerNorm
  - Separate downsampling layers

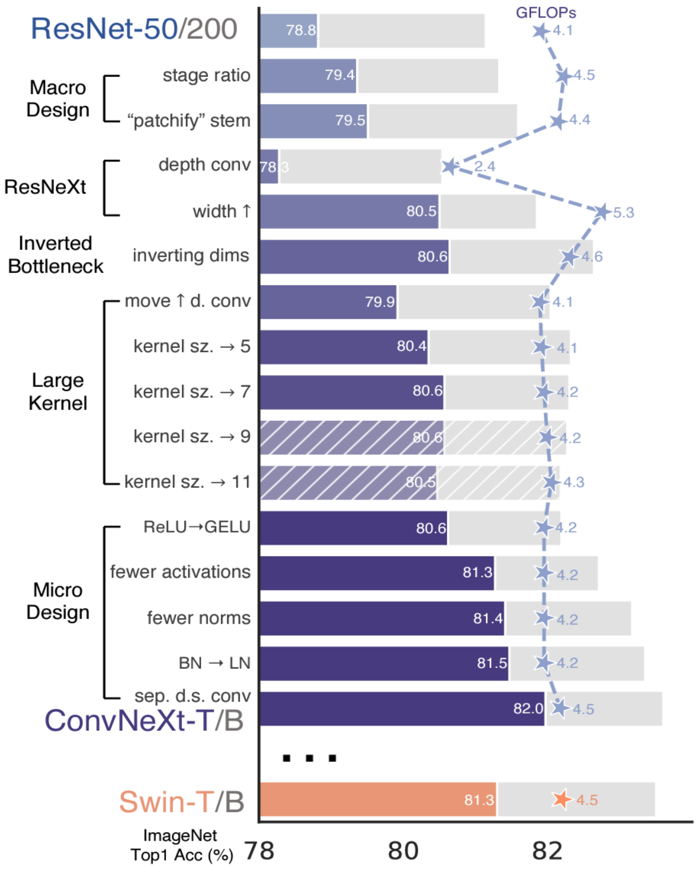


−2 −1

$$f(x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

ConvNeXt (2022)

- **Modern designs added:**
- Use ResNeXt
- Apply Inverted Bottleneck
- Use larger kernel size
- Training strategy:
  - 90 epochs -> 300 epochs
  - AdamW optimizer
  - Data augmentation like Mixup, CutMix
  - Regularization Schemes like label smoothing
  - …



ConvNeXt (2022)

- What prior approaches exist to solve this problem?
  - Will need to explore related work to answer this
- How does this work validate their approach?

ConvNeXt (2022)

- How do they validate their approach?
  - What data do they use?
  - What baselines do they compare against?

ConvNeXt (2022)

- Strengths?

ConvNeXt (2022)

- Weaknesses/limitations?