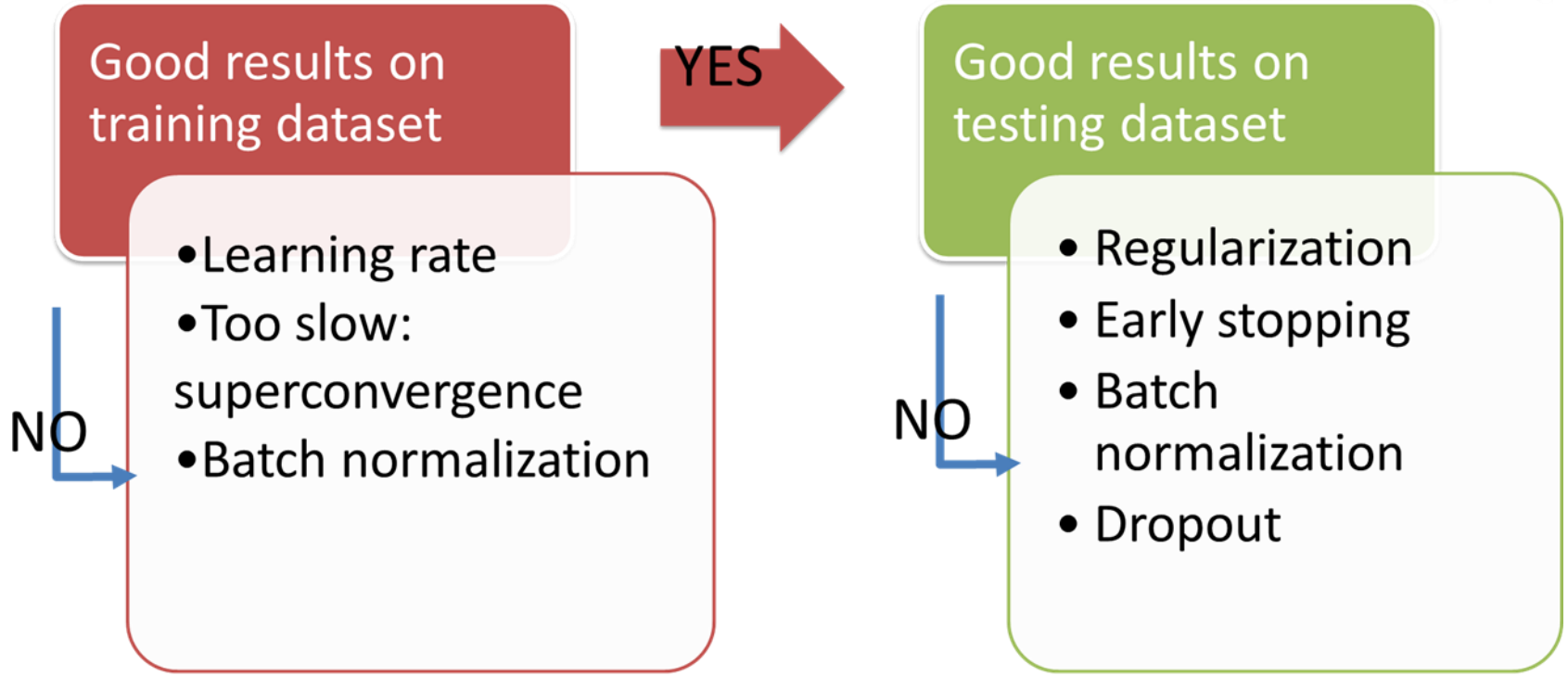


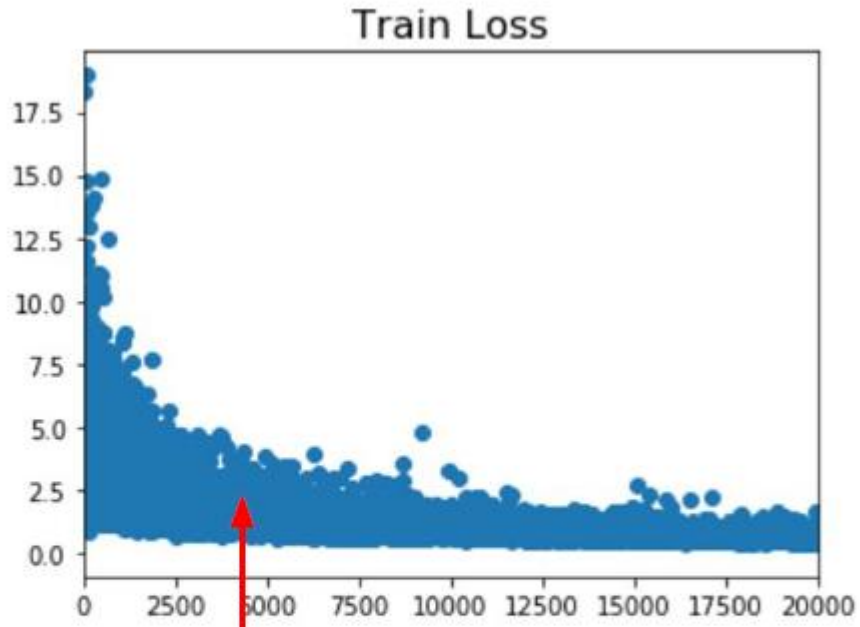
REGULARIZATION

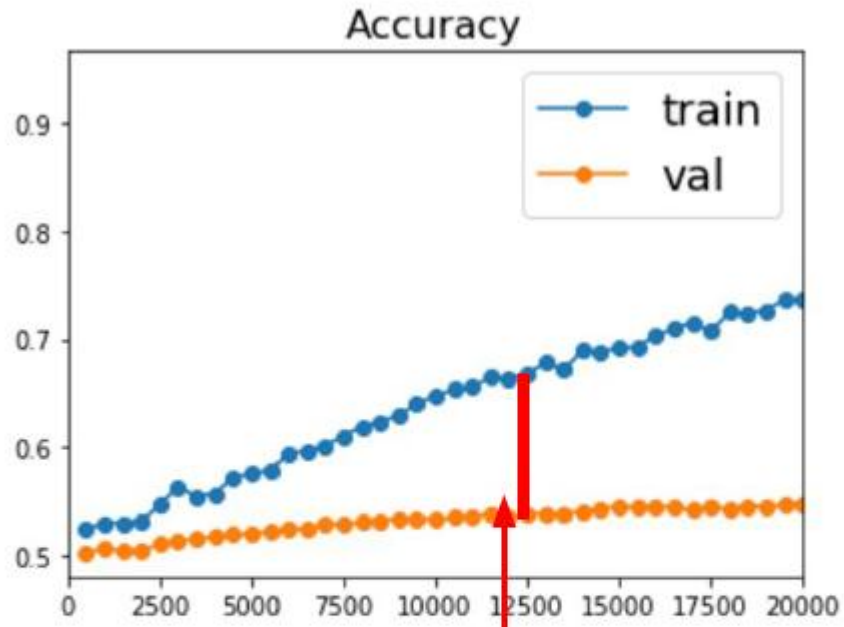
限制模型「不要想得太複雜」的傾向，逼它找到較平滑、可泛化的解

How to Reduce Generalization Error?

Regularization

- Modification made to the learning algorithm to reduce generalization error but not its training error (usually at the cost of higher training error)





Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Source: cs231n

Regularization Methods

- 參數層級
  - Weight Decay（L2 regularization, AdamW）
  - L1 regularization
- 結構層級:在模型的訓練過程中引入隨機性或噪聲
  - Dropout
  - Stochastic Depth / DropPath / LayerDrop
  - R-Drop
    - 對同一輸入做兩次 dropout 前向，最小化兩次輸出的 KL 差異，等於把「隨機性」變成一致性約束。NLP 任務上很常見
  - Batch normalization
- Data 層級: data augmentation
- 最佳化層級 (optimizer)
  - Early stop
  - Label smoothing
  - SAM（Sharpness-Aware Minimization)
  - SWA（Stochastic Weight Averaging）

Regularization: Add term to loss function

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Can be any data loss

Regularization loss

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Source: cs231n

L2 Regularization

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

- Loss function
  - ½ term for gradient

$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \frac{\alpha}{2}||\theta||_2^2$$

- Gradient of L2 regularized objective

$$\nabla \hat{L}_R(\theta) = \nabla \hat{L}(\theta) + \alpha\theta$$

- Effect on gradient descent update

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) = \theta - \eta\, \nabla \hat{L}(\theta) - \eta\alpha\theta = (1 - \eta\alpha)\theta - \eta\, \nabla \hat{L}(\theta)$$

Toward zero

  - Weight decay linearly toward zero for a single step
  - What will happen over the entire course of training?

Source: deep learning, 7-1~7-3, deep learning basics, Yingyu Liang

L2 Regularization

- What will happen over the entire course of training?
- Approximate loss function with quadratic approximation and some derivation, we can get the regularized weight

$$\tilde{w} = (H + \alpha I)^{-1} H w^* \qquad H = Q \Lambda Q^T$$

  - $\hat{w} = Q(\Lambda + \alpha I)^{-1} \Lambda Q^T w^*$
- effect on optimal solution: weight decay
  - Rescale w^* by $\frac{\lambda_i}{\lambda_i + \alpha}$ along the axes defined by the eigenvectors of H
  - decay away components of w along unimportant directions with $\lambda_i << \alpha$
    - Directions do not contribute to reduce cost
  - effect of regularization is relatively small (Keep w almost unchanged) if $\lambda_i >> \alpha$
    - Directions contribute to reduce cost

Source: : deep learning, 7-1~7-3, deep learning basics, Yingyu Liang

L1 Regularization

- Loss function $\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \alpha||\theta||_1$
- Gradient of loss function $\nabla\hat{L}_R(\theta) = \nabla\hat{L}(\theta) + \alpha\ \mathrm{sign}(\theta)$
- Gradient descent update

$$\theta \leftarrow \theta - \eta\nabla\hat{L}_R(\theta) = \theta - \eta\ \nabla\hat{L}(\theta) - \eta\alpha\ \mathrm{sign}(\theta)$$

Regularization contribution
: Constant factor

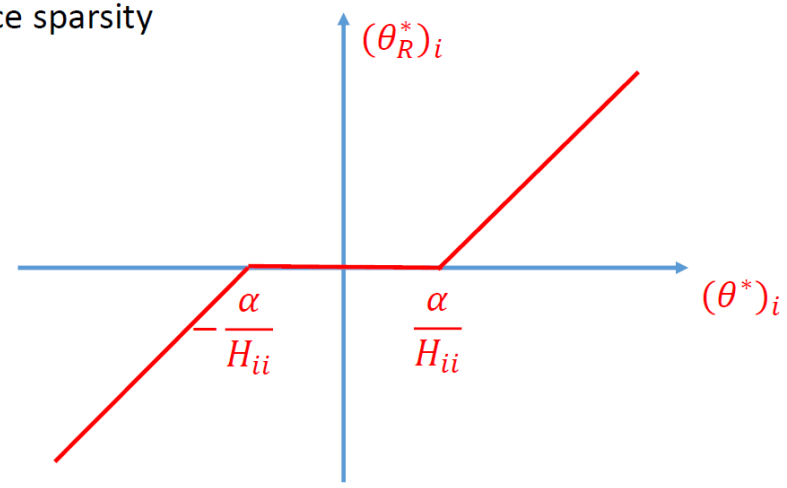
Source: deep learning, 7-1~7-3, deep learning basics, Yingyu Liang

L1 Regularization

- What will happen over the entire course of training?
- Approximate loss function with quadratic approximation and some derivation, we can get the regularized weight
- Optimal weight
  - More zero weight. weight becomes sparse

$$(\theta_R^*)_i \approx \begin{cases} \max\left\{\theta_i^* - \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* \geq 0 \\ \min\left\{\theta_i^* + \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* < 0 \end{cases}$$

- Effect: induce sparsity



Source: deep learning basics, Yingyu Liang

Regularization: Dropout

- In each forward pass, randomly set some neurons to zero
  - Probability of dropping is a hyperparameter; 0.5 is common
- Only at training, not in testing

Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Source: cs231n

Regularization: Dropout

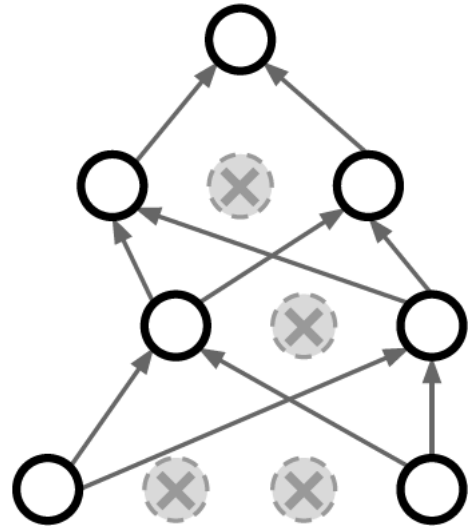
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

- 3-layer NN with dropout

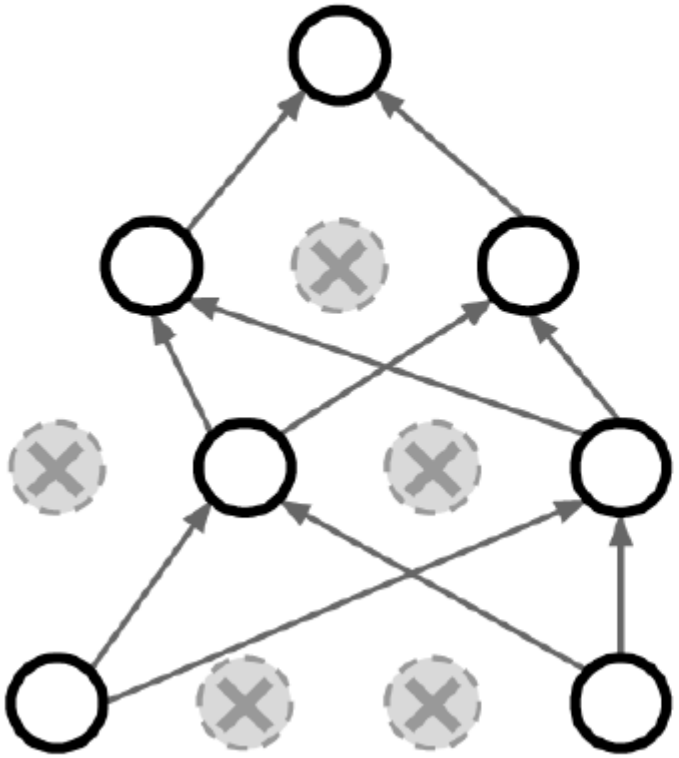


p: prob. of keeping unit active

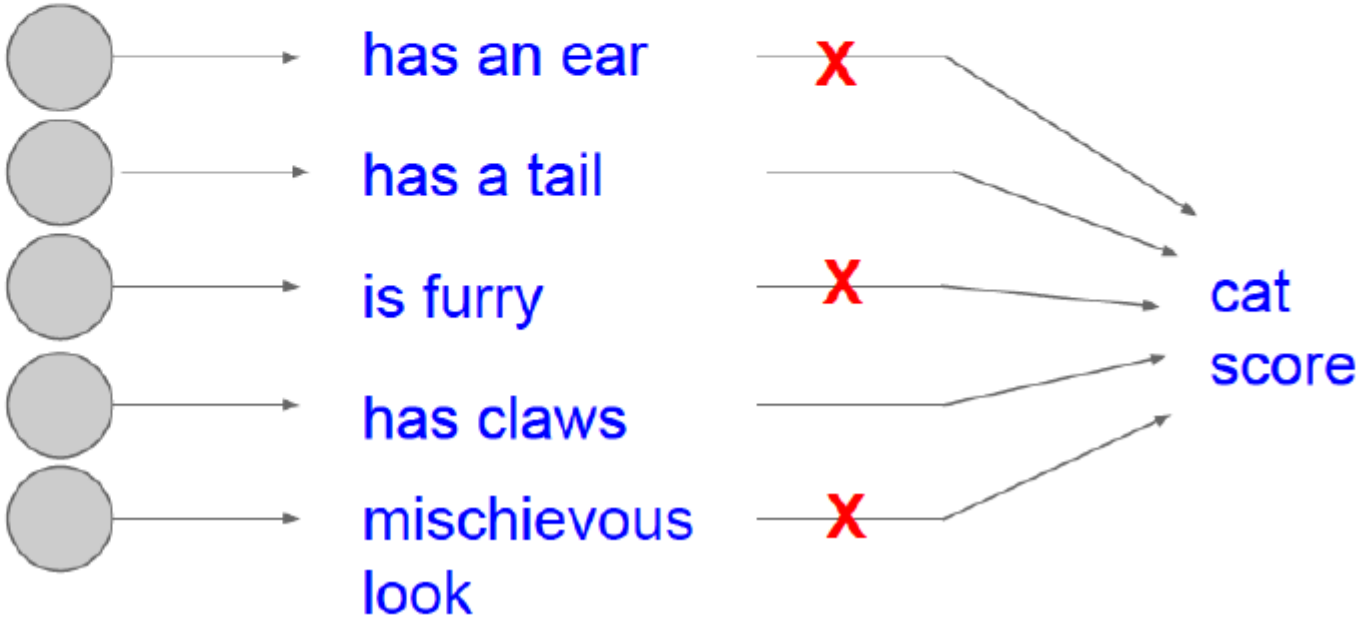
Source: cs231n

Regularization: Dropout

How can this possibly be a good idea?



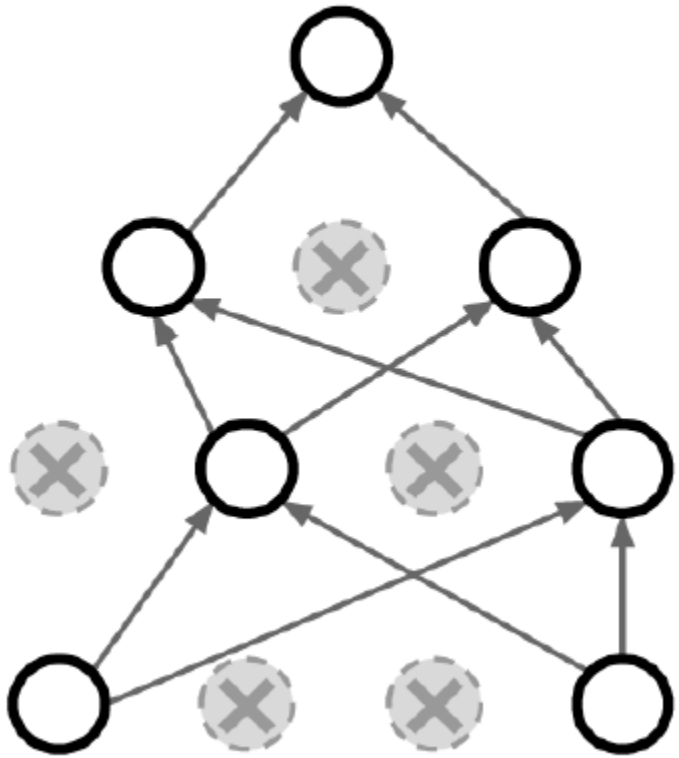
Forces the network to have a redundant representation;
Prevents co-adaptation of features



Source: cs231n

Regularization: Dropout

How can this possibly be a good idea?



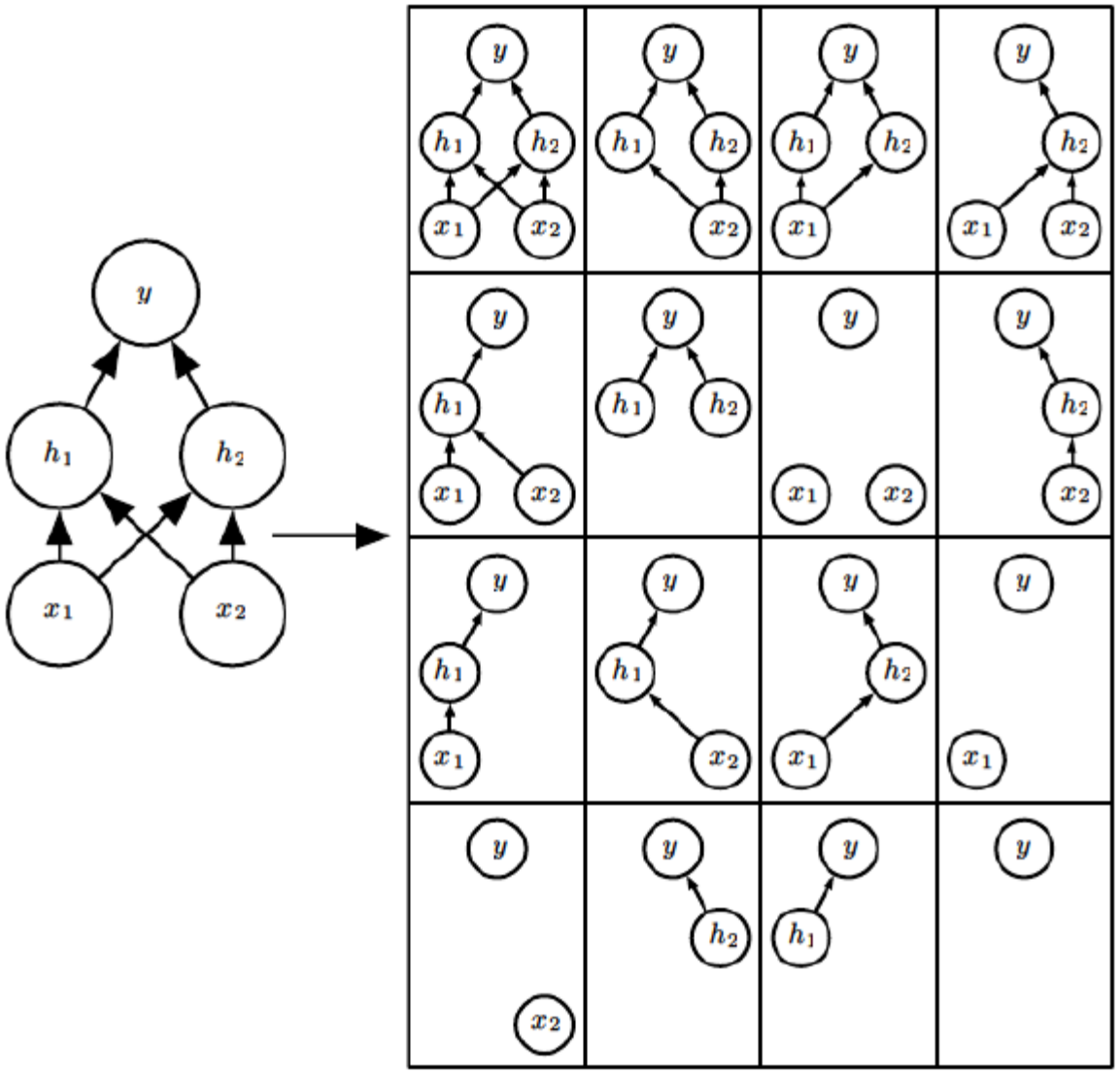
Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Source: cs231n


y
h1
h2
x1
x2

Dropout: Test time

Output (label) Input (image)

Dropout makes our output random!

$$y = f_W(x, z)$$

Random mask

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

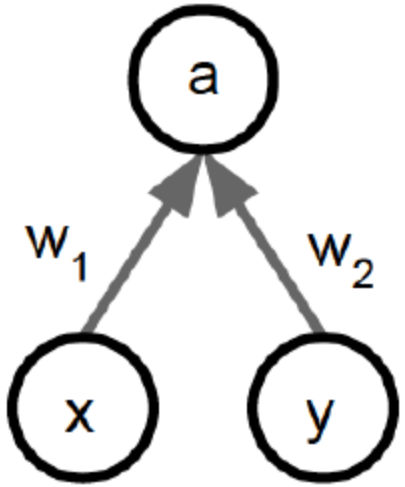
But this integral seems hard …

Source: cs231n

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$



Consider a single neuron.

At test time we have: $E\left[a\right] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E[a] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

At test time, **multiply** by dropout probability

Source: cs231n

Dropout: Test time

*p active probability

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

Source: cs231n

Not Recommended Implementation

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Dropout Summary

drop in forward pass

scale at test time

Source: cs231n

More common: "Inverted dropout"

```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

- Scaling at train time, leave forward pass at test time unchanged
  - Faster test time
  - Easy to turn on/off dropout as wish

test time is unchanged!

Source: cs231n

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

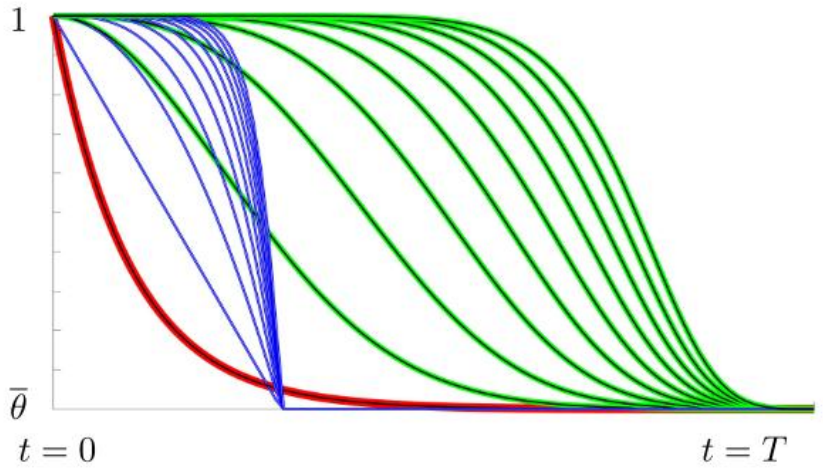
Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

Source: cs231n

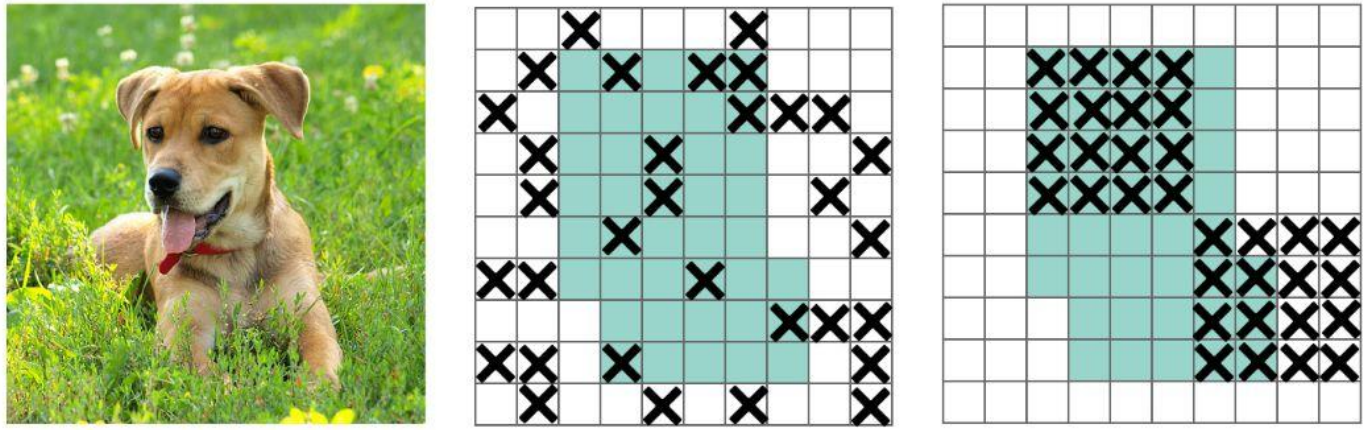
Where to Put Dropout Layers?

- After activation of conv layer or FC layer
  - Some common parameters
  - P=0.5 or 0.4 for FC layers
  - P=0.1 or 0.2 for conv layers, P=0.7 in Hinton's paper(conv layer)
- Watch out to set training and test mode in Pytorch or Tensorflow if applying dropout
  - Keras seems to handle this automatically
- Curriculum Dropout



Dropout Variants

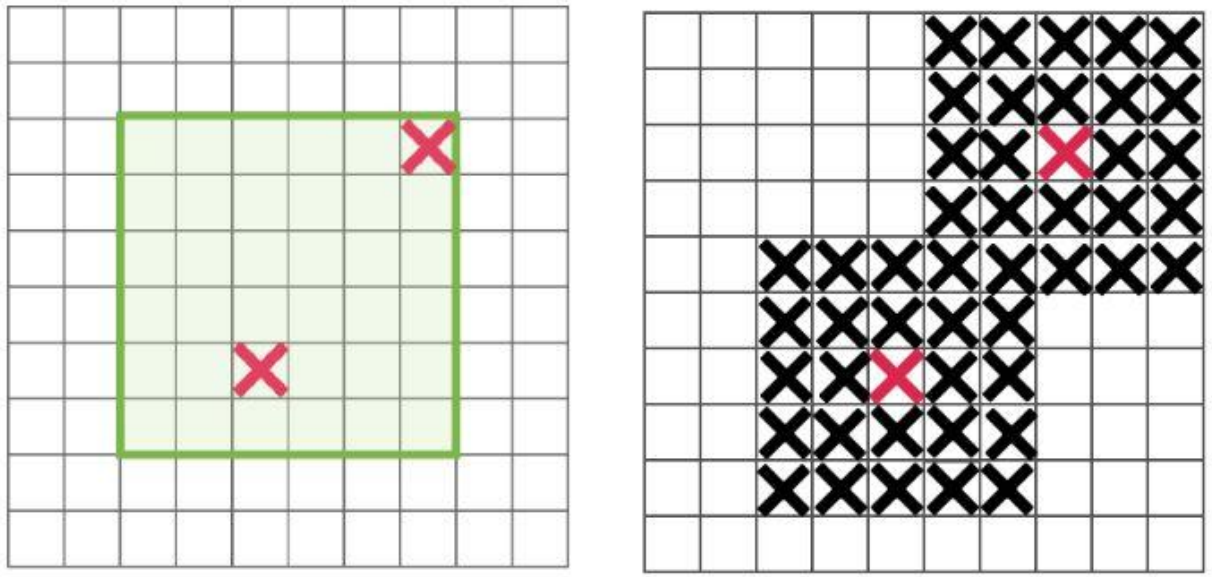
- Dropout ：完全隨機扔 (neuron level)
- SpatialDropout：按channel隨機扔
- Stochastic Depth：按res block隨機扔
- DropBlock：每個feature map上按spatial塊隨機扔
- Cutout：在input層按spatial塊隨機扔
- DropConnect：只在連接處扔，神經元不扔。



b: dropout; c: dropblock

Algorithm 1 DropBlock

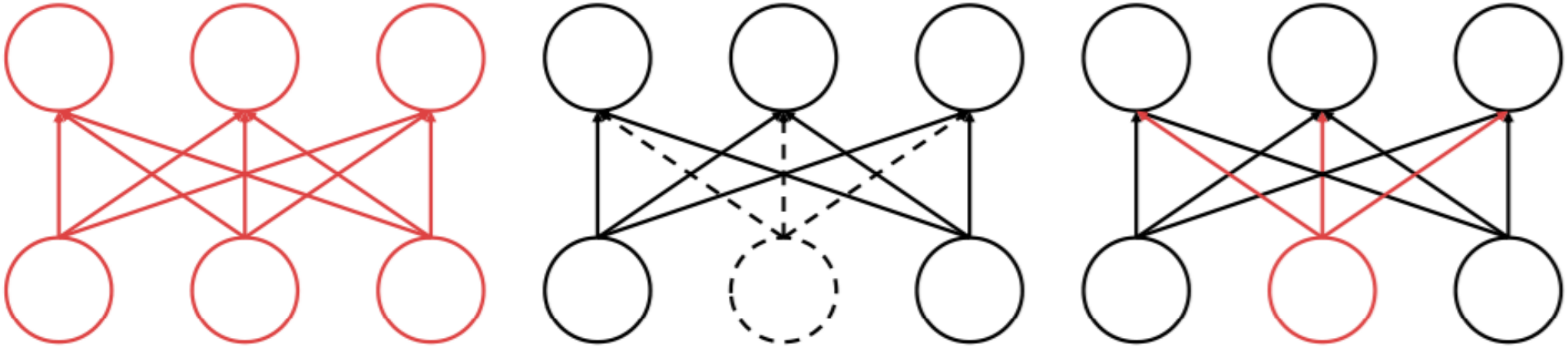
1: **Input:**output activations of a layer (A), $block\_size$, γ, $mode$
2: **if** $mode$ == $Inference$ **then**
3: return A
4: **end if**
5: Randomly sample mask M: $M_{i,j} \sim Bernoulli(\gamma)$
6: For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being $block\_size$ and set all the values of M in the square to be zero (see Figure 2).
7: Apply the mask: $A = A \times M$
8: Normalize the features: $A = A \times \textbf{count}(M)/\textbf{count\_ones}(M)$



(a) (b)

MixOut

Pretrained models

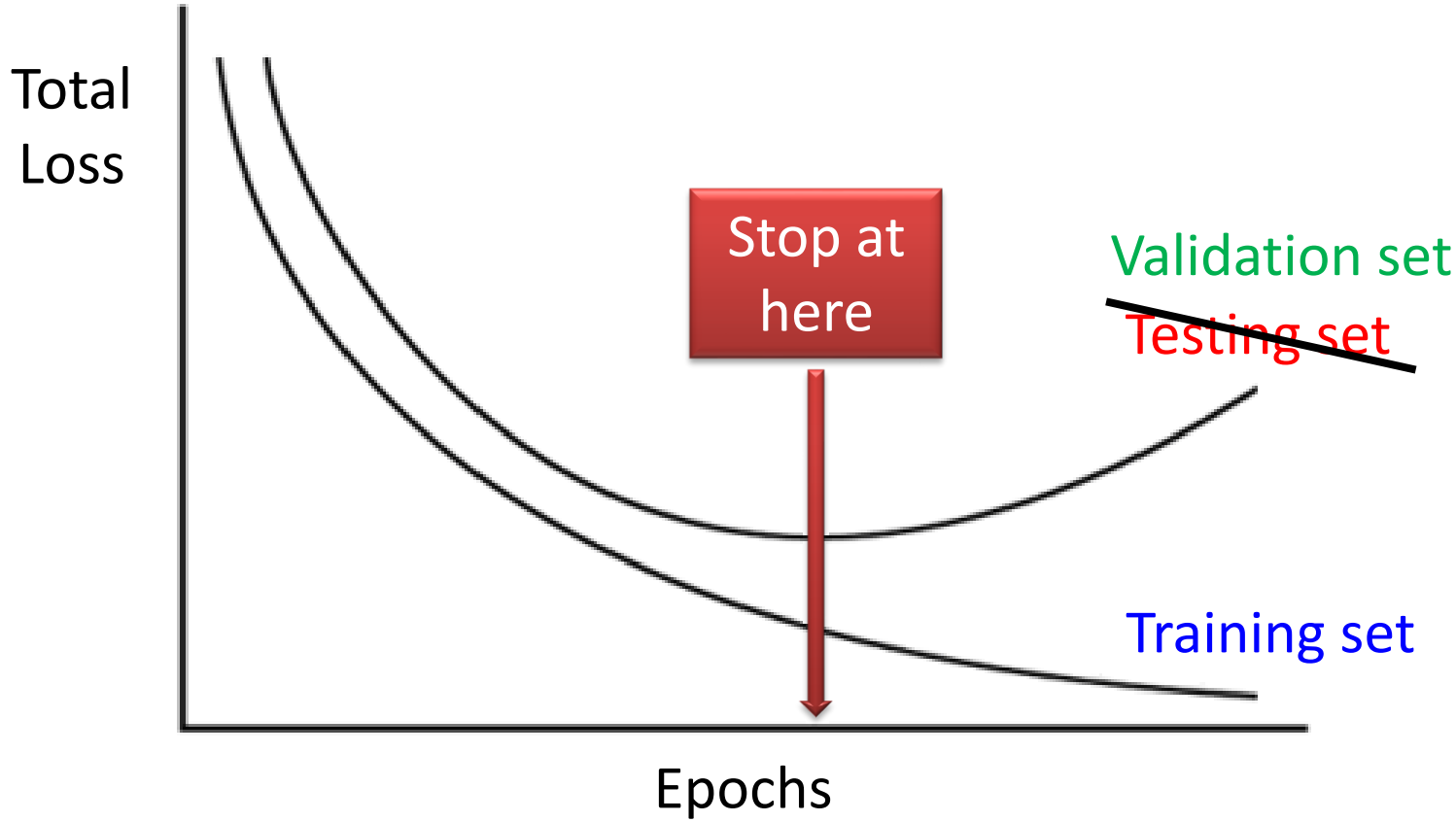


(a) Vanilla network at $\boldsymbol{u}$ (b) Dropout network at $\boldsymbol{w}$ (c) $\texttt{mixout}(\boldsymbol{u})$ network at $\boldsymbol{w}$

Figure 1: Illustration of $\texttt{mixout}(\boldsymbol{u})$. Suppose that $\boldsymbol{u}$ and $\boldsymbol{w}$ are a target model parameter and a current model parameter, respectively. (a): We first memorize the parameters of the vanilla network at $\boldsymbol{u}$. (b): In the dropout network, we randomly choose an input neuron to be dropped (a dotted neuron) with a probability of p. That is, all outgoing parameters from the dropped neuron are eliminated (dotted connections). (c): In the $\texttt{mixout}(\boldsymbol{u})$ network, the eliminated parameters in (b) are replaced by the corresponding parameters in (a). In other words, the $\texttt{mixout}(\boldsymbol{u})$ network at $\boldsymbol{w}$ is the mixture of the vanilla network at $\boldsymbol{u}$ and the dropout network at $\boldsymbol{w}$ with a probability of p.

Early Stopping

```
# Set callback functions to early stop training and save the best model so far
callbacks = [EarlyStopping(monitor='val_loss', patience=2),
ModelCheckpoint(filepath='best_model.h5', monitor='val_loss', save_best_only=True)]
```



Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

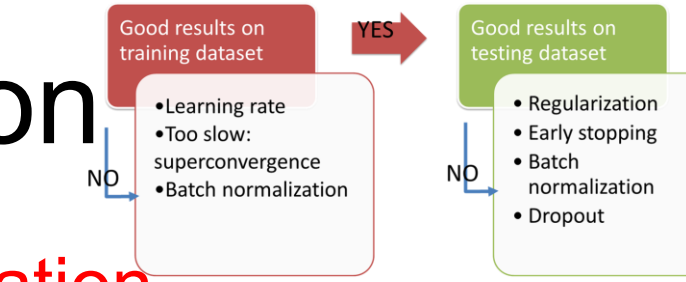
Early Stopping As Regularization

- How to do early stopping
  - When training, also output validation error
  - Store a copy of the weights when validation error improves
  - Stop if error is not improved for some time
- Early Stopping as Regularization
  - Hyperparameter selection
    - Training step is a hyperparameter. Limiting steps implies limited possible capacity
  - Advantages
    - Efficient: only store one extra copy of the weights
    - Simple: no change to the model and algorithm
  - Disadvantages
    - Need validation data

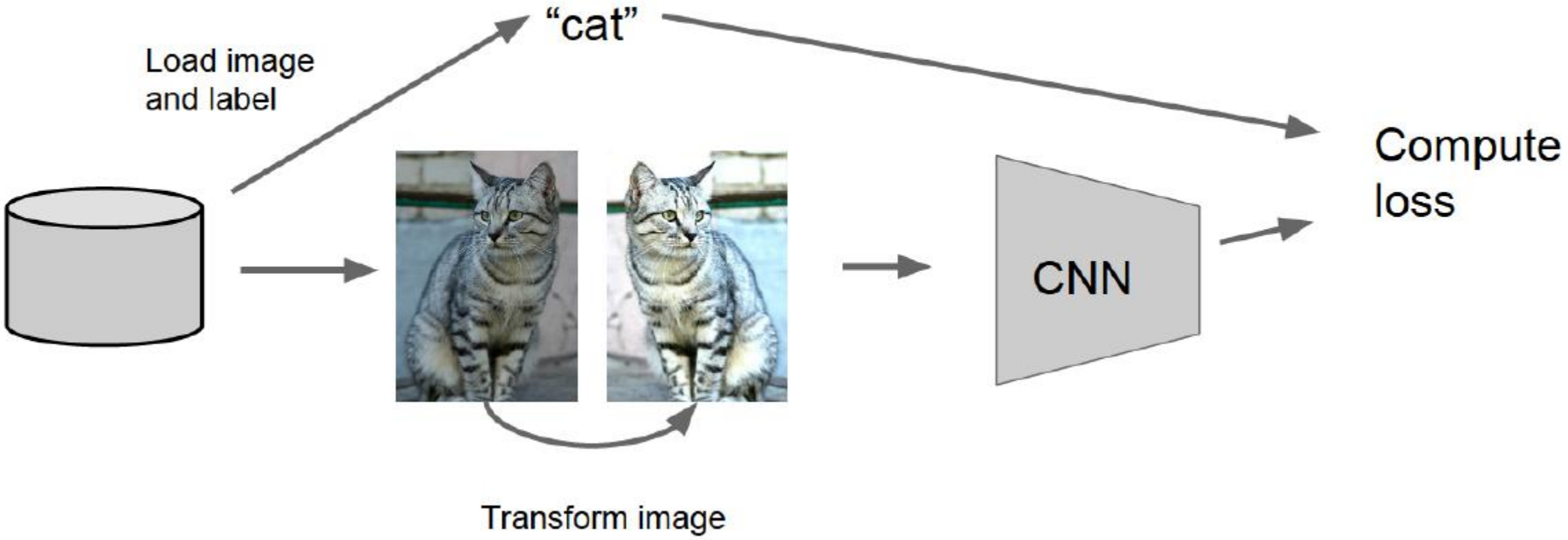
- **小資料**：
  - Label smoothing 小幅、Dropout（FC/頭部）、SWA；配合強一點的資料增強。
- **深網路/ViT**：
  - Weight decay + Stochastic depth（隨深度遞增 drop rate）；視情況加 SAM。arXiv
- **NLP/LLM 微調**：
  - Mixout/LayerDrop、輕量 weight decay、R-Drop；若資料很少，早停、凍結底層也常見。

DATA AUGMENTATION

Regularization: data argumentation



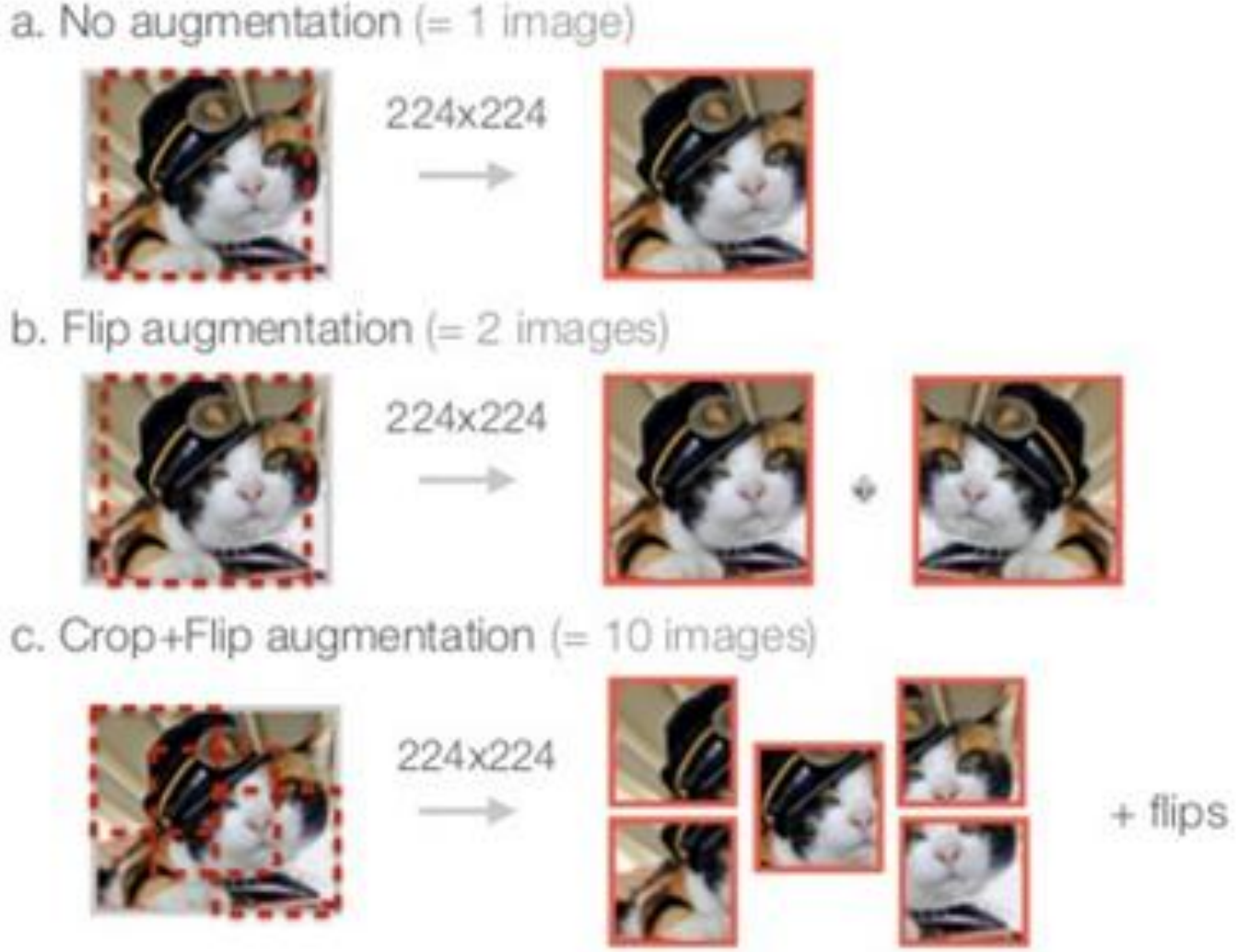
- Data argumentation: invariant to certain transformation
  - A cheap way to increase the amount of your training data



Source: cs231n

Data Augmentation

- Increase data size and robustness
- Be careful about transformation applied
  - E.g. ‘b’ and ‘d’
  - E.g. ‘6’ and ‘9’



Source: augmentation, Eva Mohedano

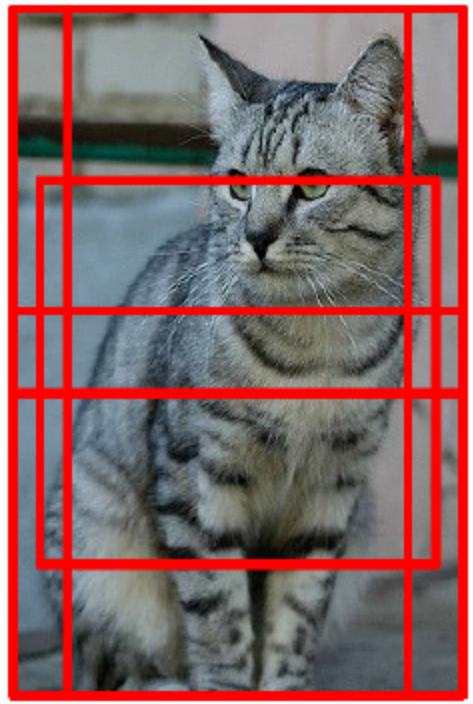
Data Augmentation for ImageNet Training

Random crops and scales

Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Source: cs231n

Data Augmentation

Color Jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Source: cs231n

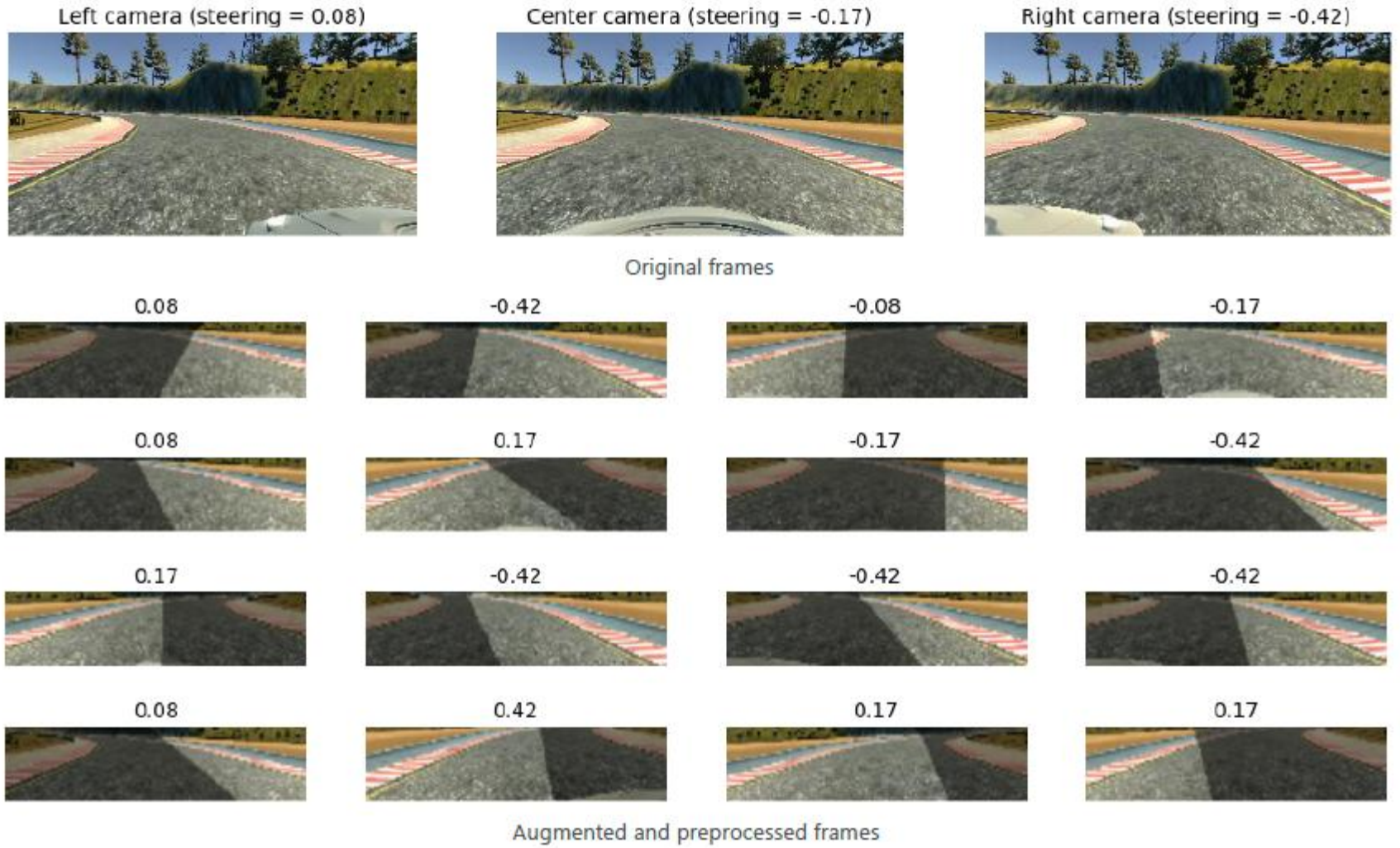
Data Augmentation

- Random mix/combinations of
  - Translation/rotation/stretching/shearing/lens distortions,
  - Noise injection in inputs, hidden units, outputs, and weights
    - Be careful about noise amount
- Particular useful for classification problems
  - No label changes
- Not as readily applicable to many other tasks, e.g. density estimation
- Certain tasks such as steering angle regression require dataset augmentation to perform well.

Source: cs231n

Data Argumentation: Steering Angles

- Use left and right cameras
- Horizontal flip
- Vertical shift
- Random shadow
- 1 sample =>16

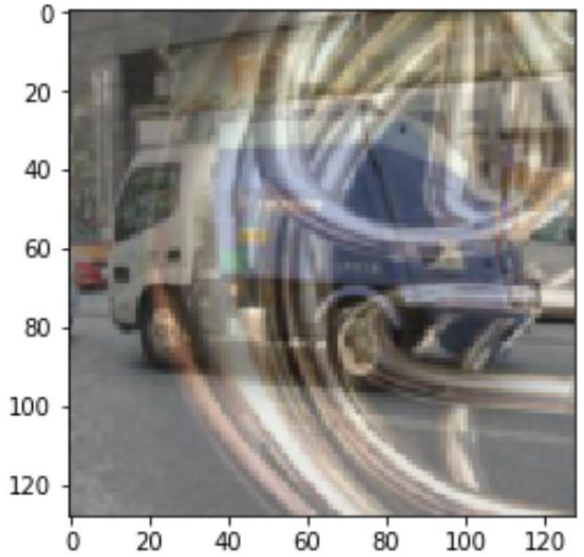


Source: https://navoshta.com/end-to-end-deep-learning/

Mixup

- new_image = t * image1 + (1-t) * image2
- new_target = t * target1 + (1-t) * target2
  - assuming your targets are one-hot encoded (which isn't the case in pytorch usually)

```
mixed_up = ll.train.x[0]*0.5 + ll.train.x[4000]*0.5
plt.imshow(mixed_up.permute(1,2,0));
```



French horn or truck? The right answer is 50% french horn and 50% truck ;)

Spatial Mixup

(a) vertical stitching

(b) horizontal stitching

(c) random rectangle stitching

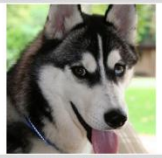
Dataset	Model	ERM	mixup-C	mixup-H	mixup-HV	mixup-HC
CIFAR-10	PreAct ResNet-18	5.6	3.9	3.9	3.4	3.5
	DenseNet-BC-190	3.7	2.7	2.7	2.3	2.3
	ResNext29-8-64	3.7	2.7	-	2.4	-
CIFAR-100	PreAct ResNet-18	25.6	21.1	21	19.8	19.9
	ResNext29-8-64	17.4	16.8	-	14.5	-
ImageNet	ResNet-50	23.5	23.3	23.3	23	23.1
	ResNet-101	22.1	21.5	21.7	20.3	20.5
	ResNeXt-101 64×4d	20.4	19.8	19.9	19.3	19.3

Mixup

- 這些合成的training data的作用
  - 流行的解釋是“**增強模型對某種變換的invariance**”。
  - 這句話反過來說，就是機器學習裡經常提到的“減少模型估計的variance”，也就是控制了模型的複雜度

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \quad \text{where } x_i, x_j \text{ are raw input vectors}$$

$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \quad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

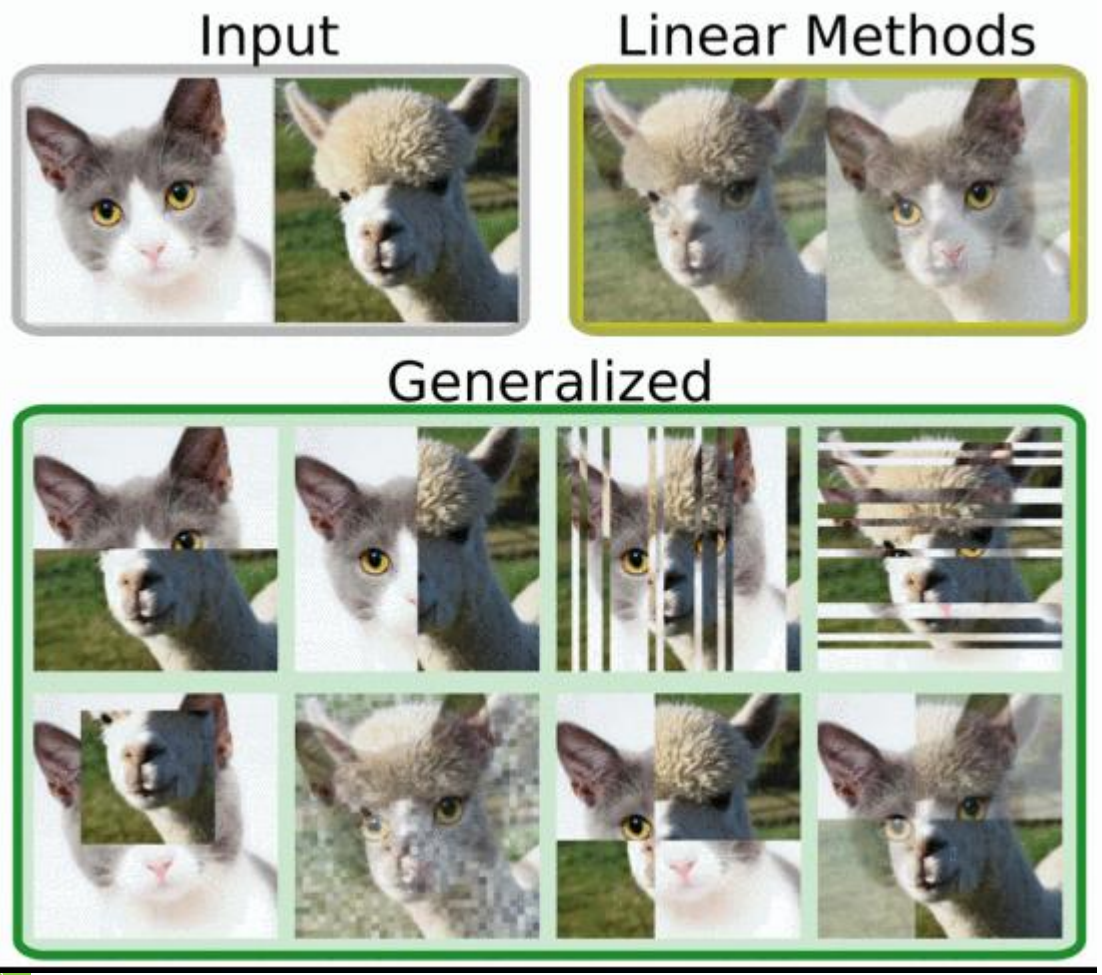
0.4 × data[1]:		cat: 1.0
+ 0.6 × data[2]:		dog: 1.0
= mixup:		cat: 0.4 dog: 0.6

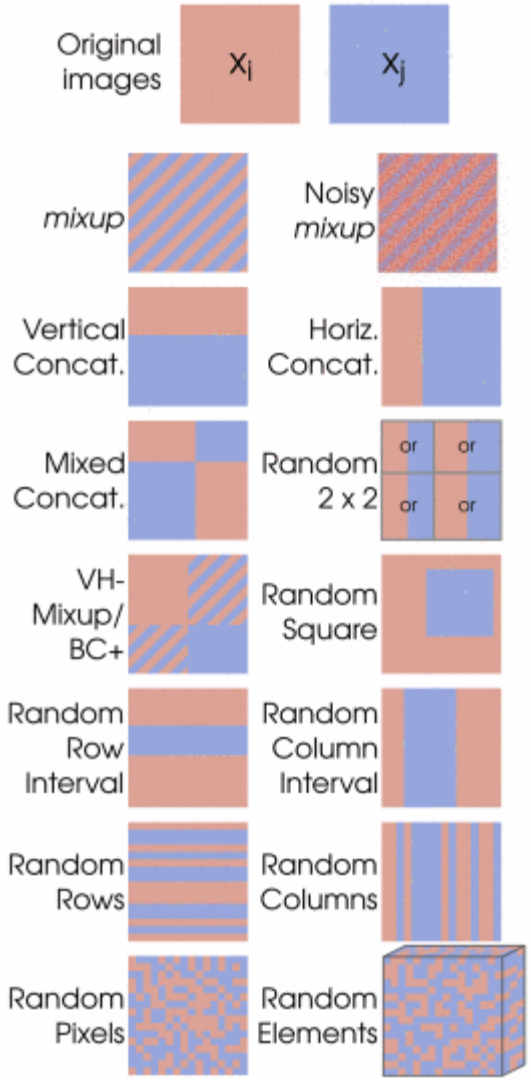
mixup: BEYOND EMPIRICAL RISK MINIMIZATION

CutMix

	ResNet-50	Mixup	Cutout	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	**78.4** **(+2.1)**
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	**47.3** **(+1.0)**
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	**76.7** **(+1.1)**

Improved Mixed-Example Data Augmentation





	Method	Error (%)
CIFAR-10	ResNet-18	5.4
	mixup[27]	4.3
	BC+[24]	4.2
	Rand. Elems.	6.2
	Rand. Pixels	5.7
	Rand. Col. Int.	5.1
	Rand. Cols	4.8
	Horiz. Concat.	4.7
	Rand. Rows	4.6
	Noisy Mixup	4.5
	Rand. Row Int.	4.5
	Vert. Concat.	4.4
	Mixed. Concat.	4.4
	Rand. Square.	4.3
	Rand. 2×2	*4.1*
	VH-BC+	*3.8*
	VH-Mixup	*3.8*

Augmix

提升圖像分類模型的魯棒性（robustness）和不確定性估計（uncertainty estimates），而且非常容易嵌入目前的訓練流程中。AugMix的原理很簡單：隨機對圖像進行不同的資料增強（Aug），然後混合（Mix）多個資料增強後的圖像；同時在分類器上施加對同一圖像的不同增強後的一致性約束。

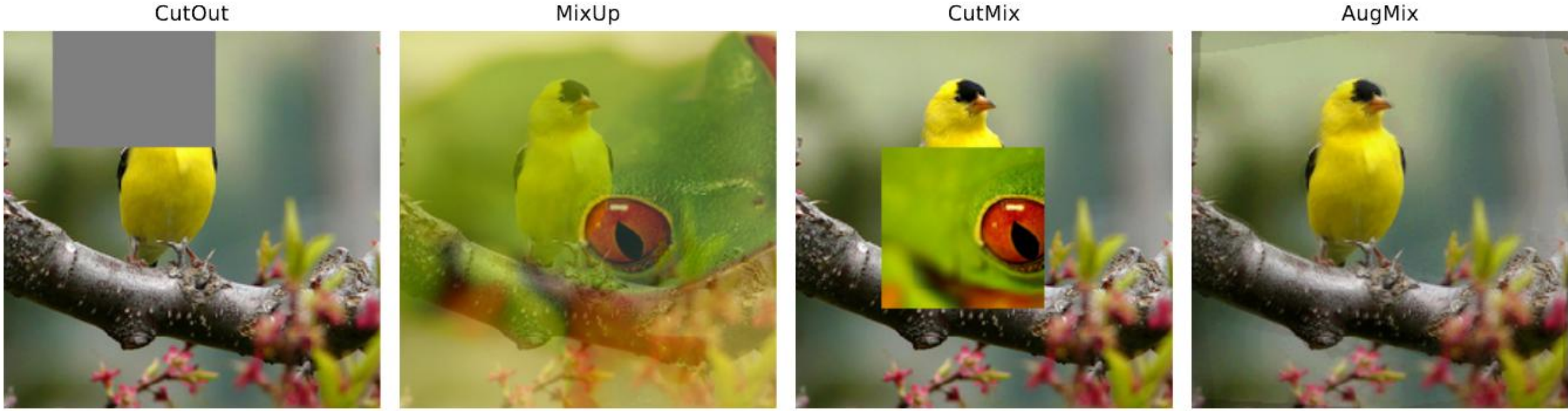


Figure 1: A visual comparison of data augmentation techniques. AUGMIX produces images with variety while preserving much of the image semantics and local statistics.

https://arxiv.org/abs/1912.02781

混合同一個影像的經過不同資料增強得到的影像

AugMix由於是混合同一個圖像，相比CutMix和MixUp，其得到的圖像更自然一些；如果只是串聯不同的數據增強，最後得到的圖像往往偏離原圖，變得失真

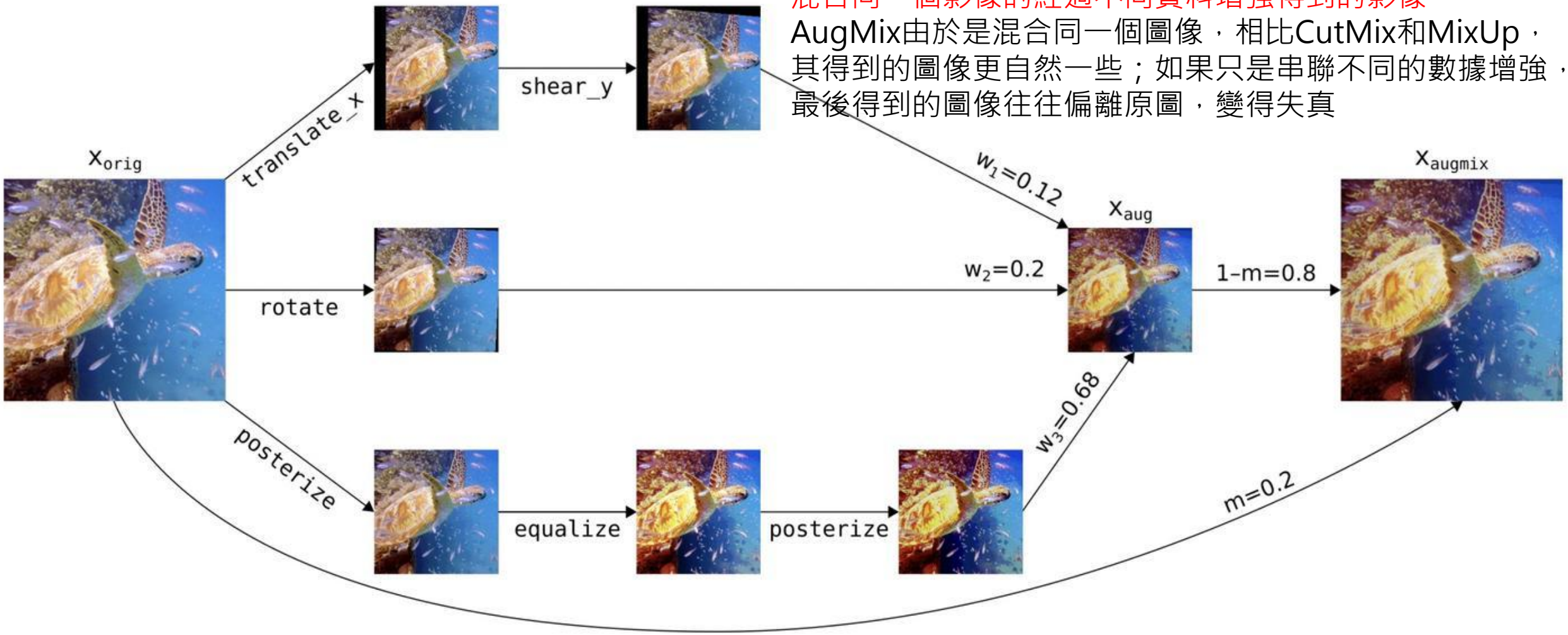
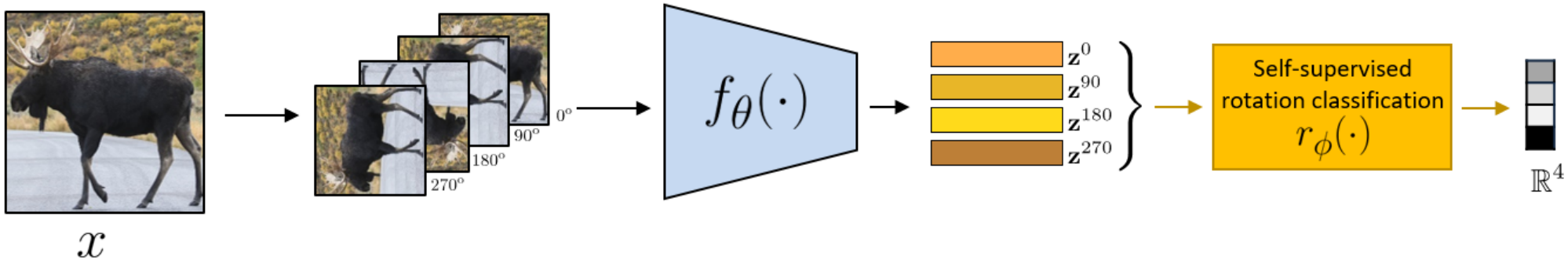


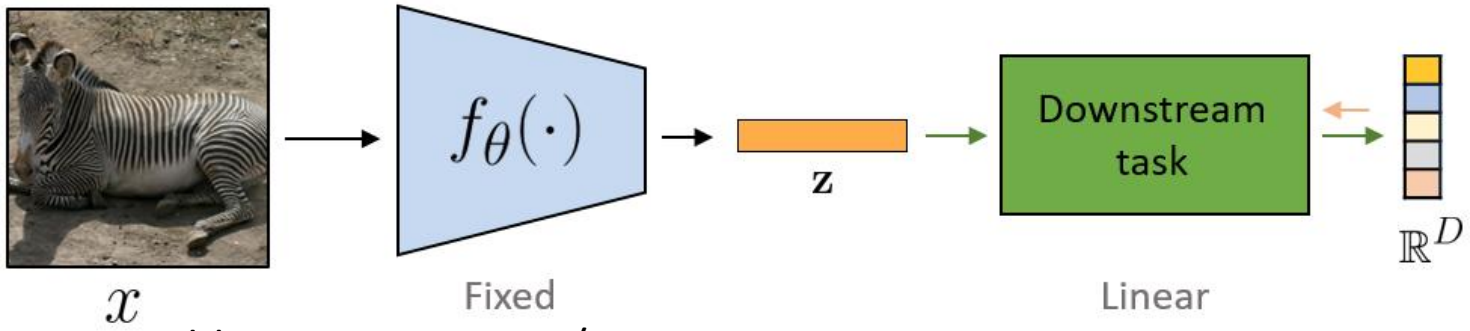
Figure 4: A realization of AUGMIX. Augmentation operations such as translate_x and weights such as m are randomly sampled. Randomly sampled operations and their compositions allow us to explore the semantically equivalent input space around an image. Mixing these images together produces a new image without veering too far from the original.

Self Supervised Learning

- Stage 1: Train network on pretext task (without human labels)



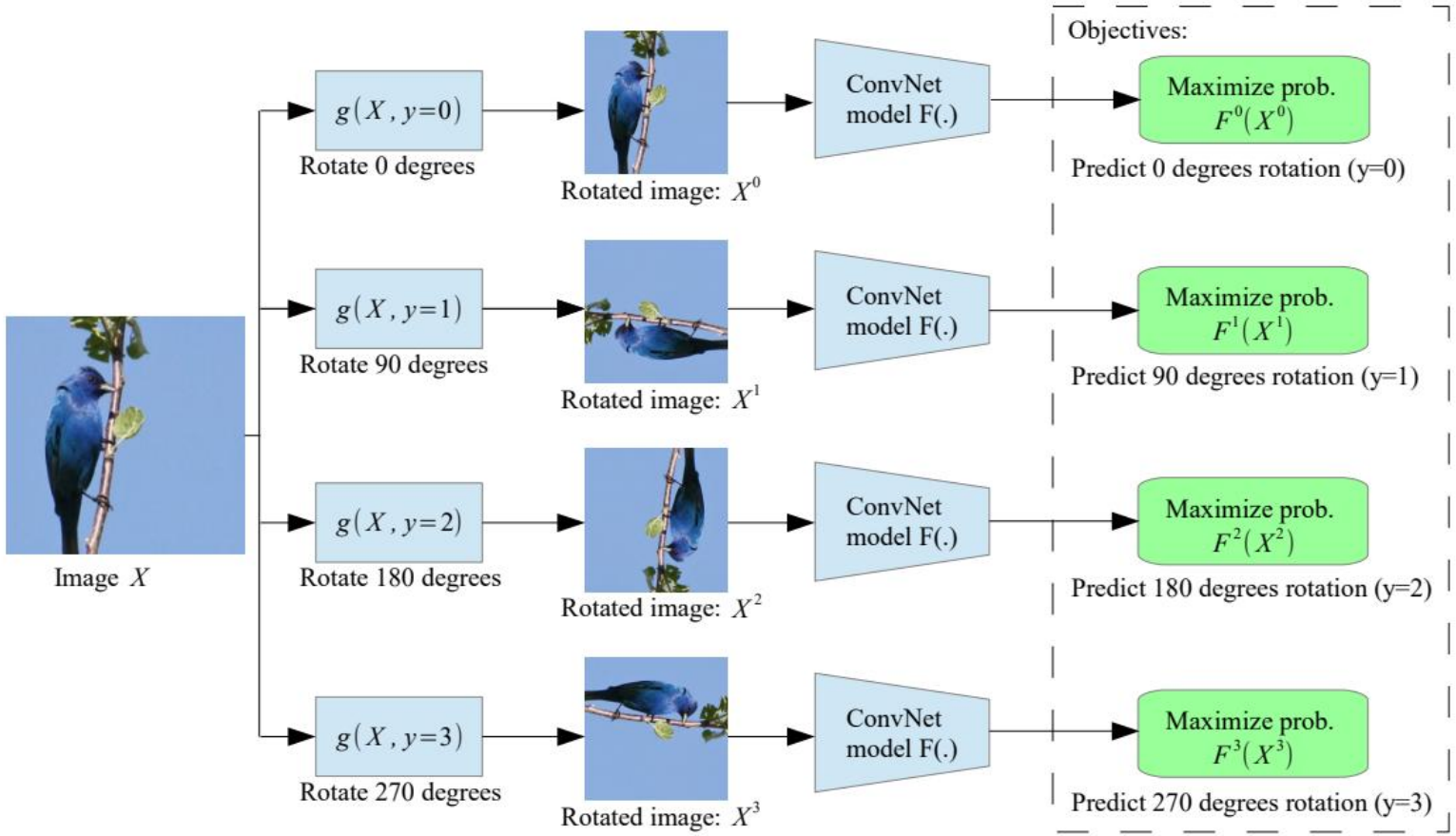
- Stage 2: Train classifier on learned features for new task with fewer labels



https://gidariss.github.io/self-supervised-learning-cvpr2021/

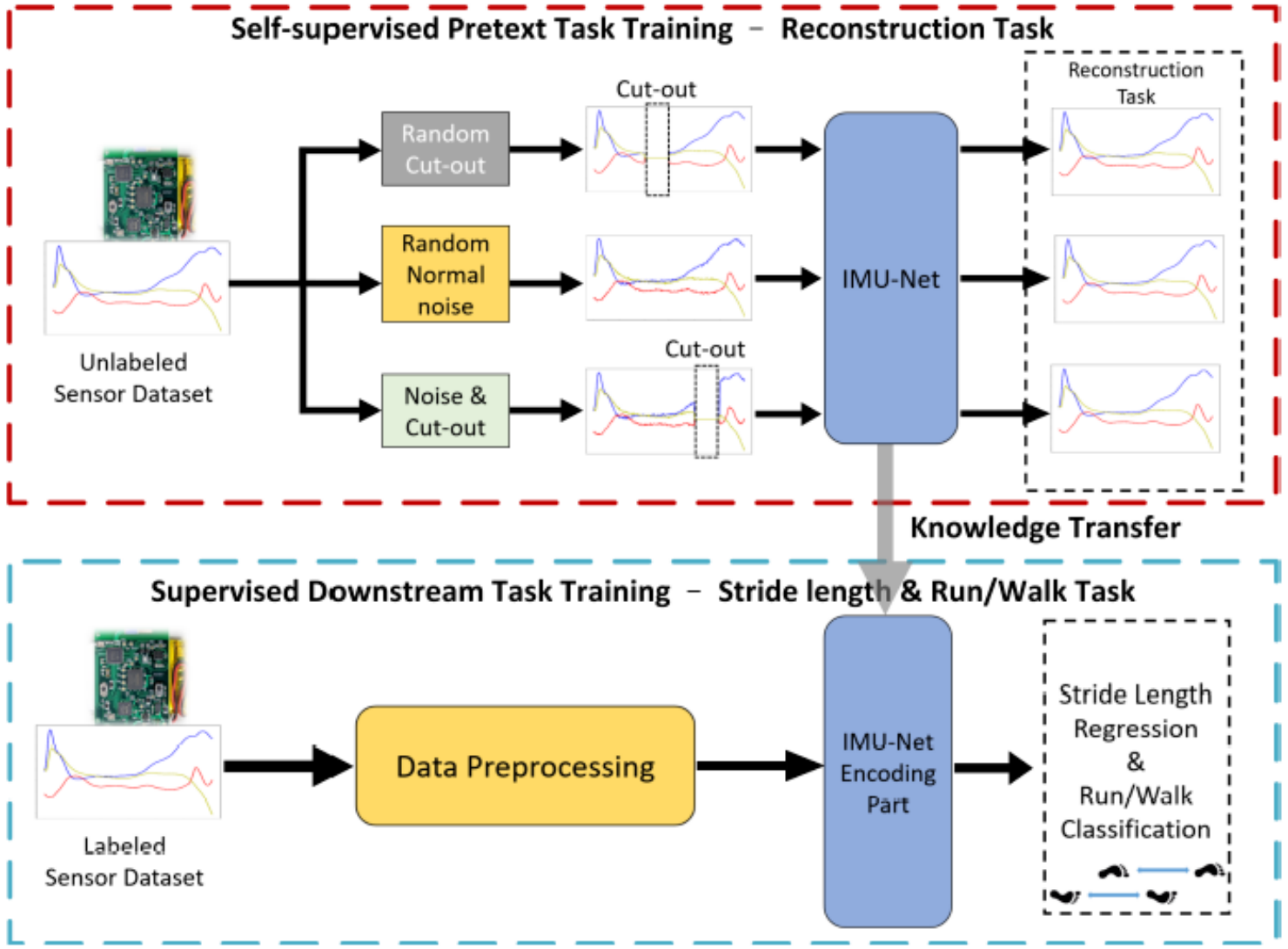
Example: Rotation prediction

Predict the orientation of the image



S. Gidaris et al., Unsupervised Representation Learning by Predicting Image Rotations, ICLR 2018

Example: IMU Sensor Based Prediction



Summary: Data Augmentation

- **CV：基本到自動化**
  - **幾何/色彩/裁切/隨機擦除**：課堂起手式；強度要配合資料。
  - **Mixup / CutMix**：將兩張圖（與標籤）線性/區塊混合；CutMix 保留空間線索，學到更完整目標區域。CVF 開放存取
  - **AutoAugment / RandAugment / TrivialAugment**：從搜尋到**零參數**策略（TA）。課堂可對照：AA（需費時搜尋）→ RA（2 參數）→ **TA（免調）**。proceedings.neurips.ccCVF 開放存取
  - **AugMix**：把多條增強管線結果混合，外加**一致性正則**，對**腐蝕/分佈偏移**有韌性。openreview.net
  - **偵測/分割特化**：Mosaic、Copy-Paste、以及把 CutMix/CutOut 調整到半監督分割也有效。arXiv
- **CV：生成式（2024–2025 趨勢）**
  - **Diffusion-based Augmentation**：用擴散模型產生或編輯樣本（如 Diff-Mix、GenMix、AGA）；要麼在**類間插值**、要麼**保留前景/改變背景**以兼顧**真實度（faithfulness）與多樣性（diversity）**。可提升少樣本、長尾與魯棒性，但需控**標籤歧義/偽影**。CVF 開放存取 +1arXiv
  - 也有直接把**預訓練擴散模型作為增強算子**的做法（image-to-image 變換），實證顯著提升分類表現。arXiv

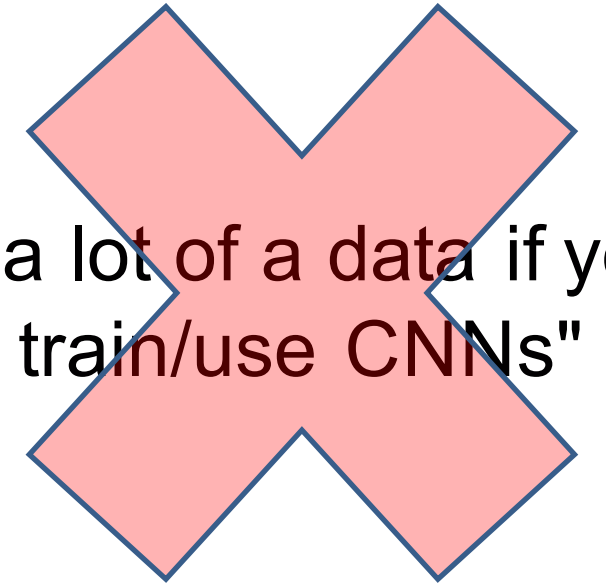
Data Augmentation

- **NLP / LLM**
  - **EDA**：同義詞取代、插入、對調、刪除，適合小資料文本分類。aclanthology.org
  - **Back-Translation**：翻譯→再翻回，保語意而造變體；到 2024–2025 仍常見在低資源與魯棒性研究。aclanthology.orgACM Digital Library
  - **LLM 生成資料**：如 **Self-Instruct** 自舉指令資料，已成對齊/指令微調的主流資料擴增策略。arXivaclanthology.org
- **語音（ASR）**
  - **SpecAugment**：對梅爾頻譜做時間/頻率遮罩與扭曲，簡單有效，Whisper/Conformer 教材都可複現。arXiv
- **測試時增強（TTA）**
  - 在推論端對輸入做多種增強並集成，可改善長尾/分佈移轉與不確定性估計（近年也見於推薦、校準/合規估計研究）。成本是推論延遲↑。CVF 開放存取

- CV
  - **何時少用/避免**：醫影或需要嚴謹幾何/亮度物理意義的任務（過強顏色/幾何變形會破壞真相）；檢測中若標註無法同步變換要小心（如關鍵點）。

TRANSFER LEARNING

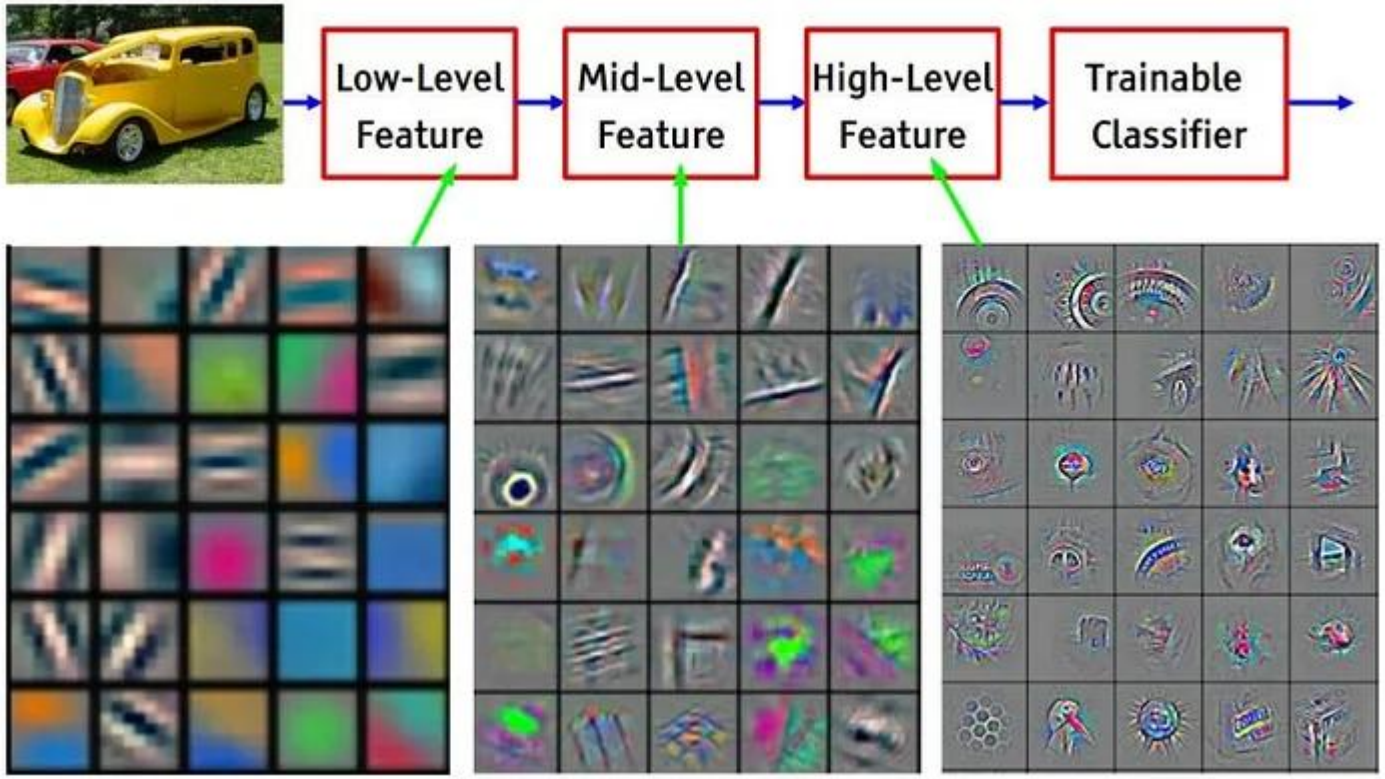
Transfer Learning



Source: cs231n

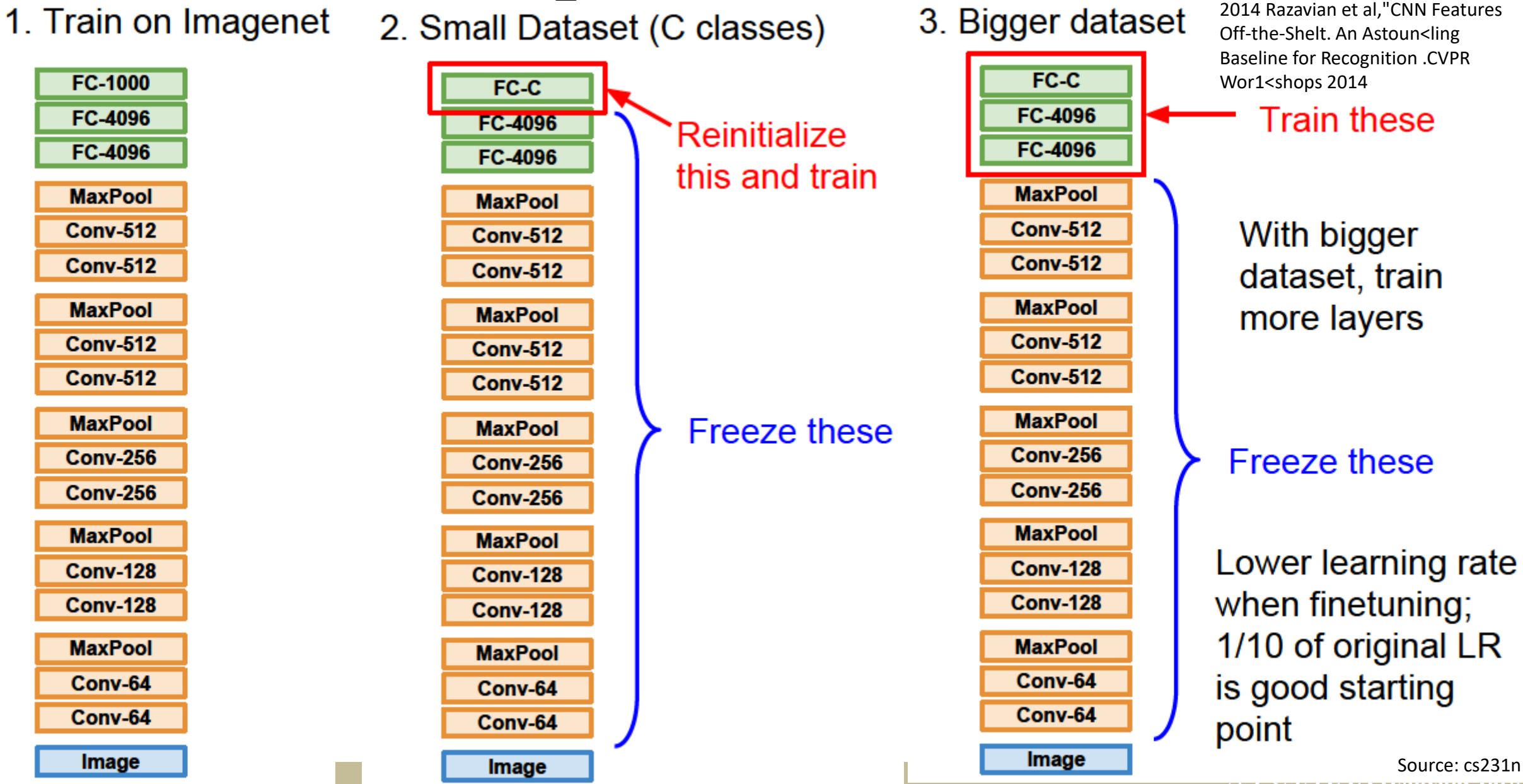
Recall

- NN在學習的時候，
  - 在淺層會學會抽取較為“粗淺”（Low level）的特徵 (task independent)
  - 而越深層則會學會越"複雜"（High level）的特徵。(task specific)



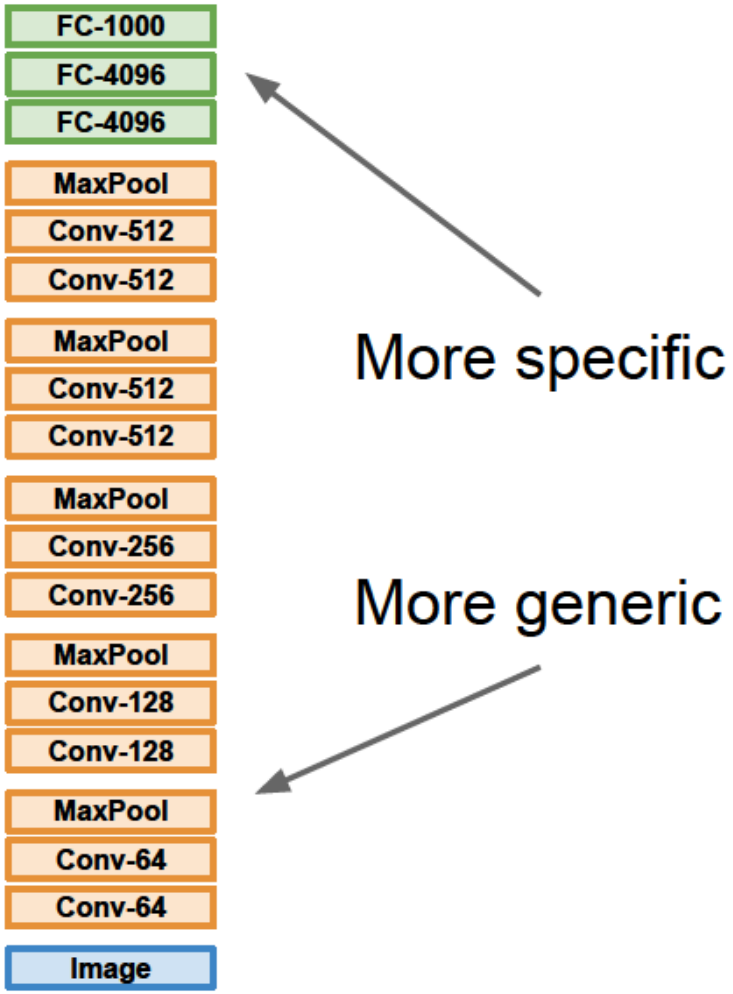
Transfer Learning with CNNs

Donahue et al,"DeCAF:A Deep ConvolulionalAc ivation Feature for Generic Visual Recognmon .ICML 2014 Razavian et al,"CNN Features Off-the-Shelt. An Astoun<ling Baseline for Recognition .CVPR Wor1<shops 2014



Source: cs231n

Transfer Learning

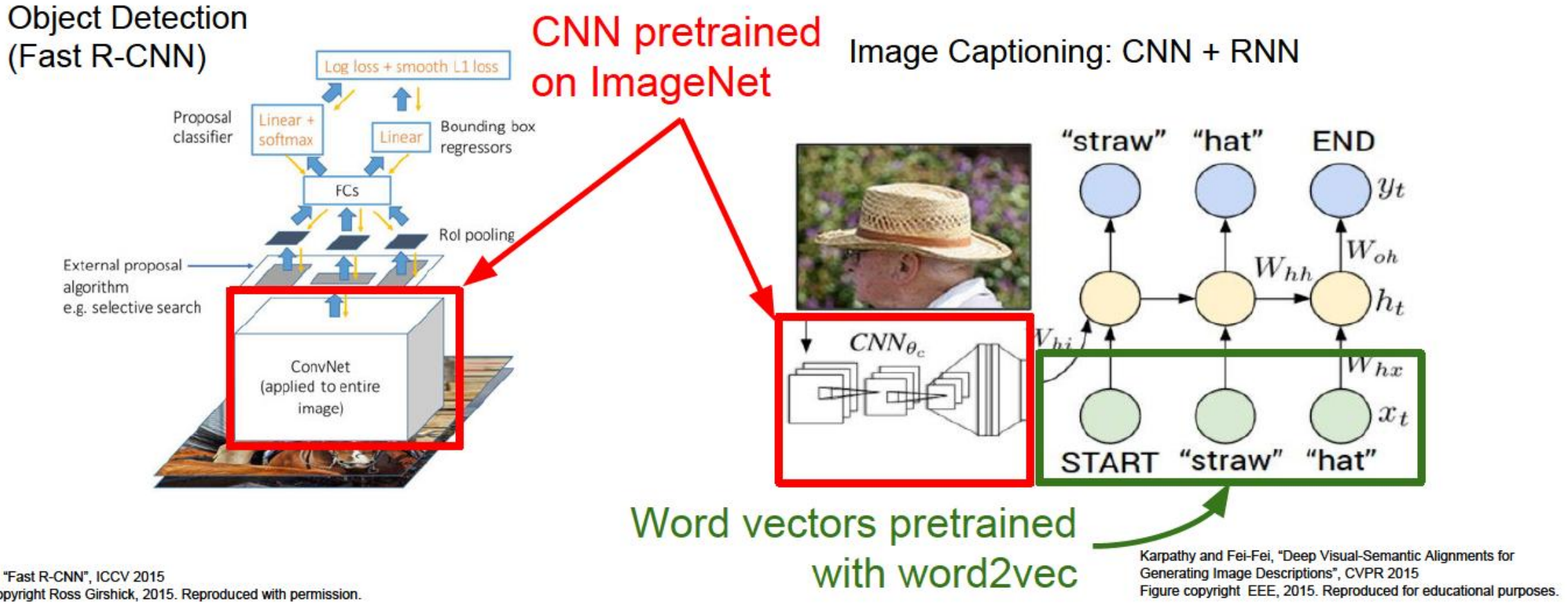


	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Source: cs231n

Transfer Learning with CNNs is pervasive

- It is the norm, not an exception



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright EEE, 2015. Reproduced for educational purposes.

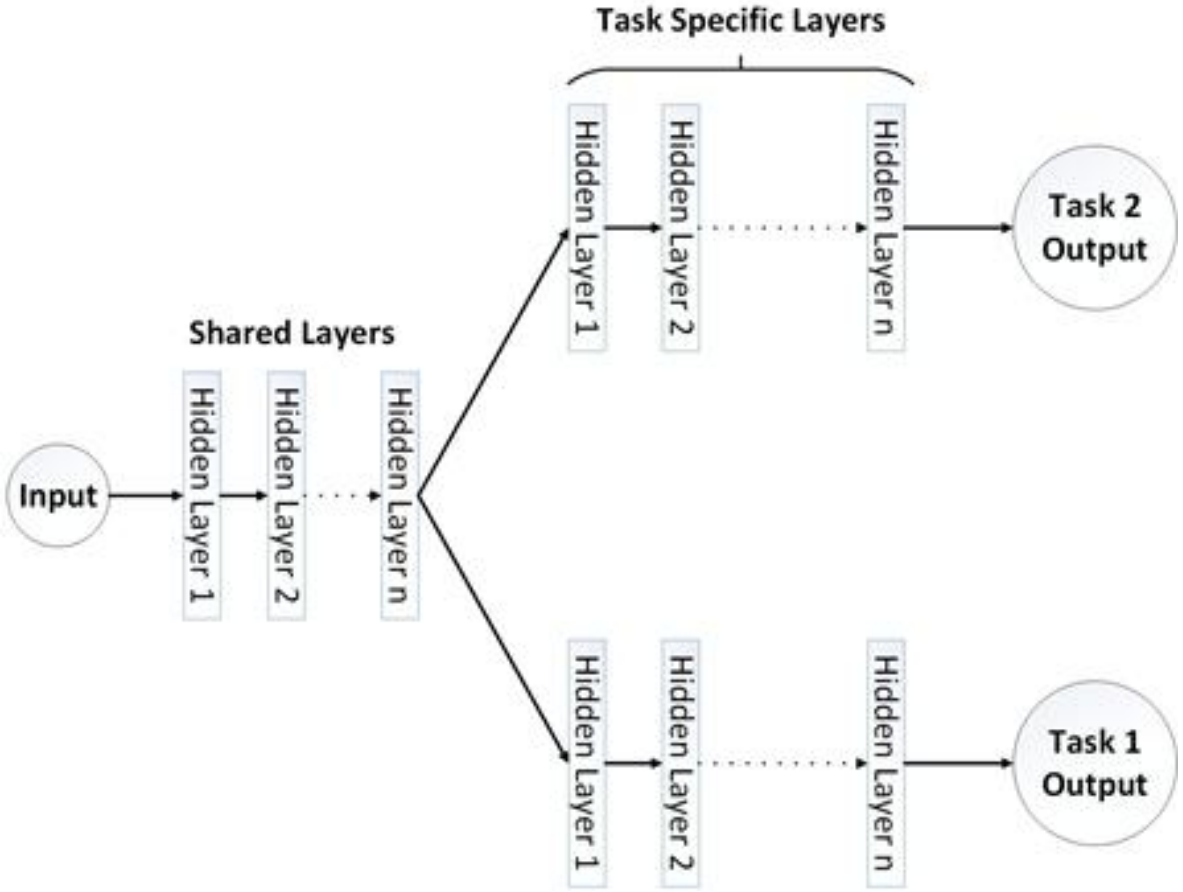
Source: cs231n

Transfer Learning

- Transfer a model trained on source data A to target data B
  - Task transfer: in this case, the source and target data can be the same
    - Image classification -> image segmentation
    - Machine translation -> sentiment analysis
    - Time series prediction -> time series classification
    - …
  - Data transfer:
    - Images of everyday objects -> medical images
    - Chinese -> English
    - Physiological signals of one patient -> another patient
    - …
- Rationale: similar feature can be useful in different tasks, or shared by different yet related data.

Multi-Task Learning

- improve generalization by pooling the examples arising out of several tasks
- Two parts
  - Shared layers
  - Task specific layers



Takeaway for your projects and beyond

- Have some dataset of interest but it has < 1M images?

- 1. Find a very large dataset that has similar data, train a big ConvNet there
- 2. Transfer learn to your dataset

- Deep learning frameworks provide a "Model Zoo" of pretrained models so you don't need to train your own
  - Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
  - TensorFlow: https://github.com/tensorflow/models
  - PyTorch: https://github.com/pytorch/vision

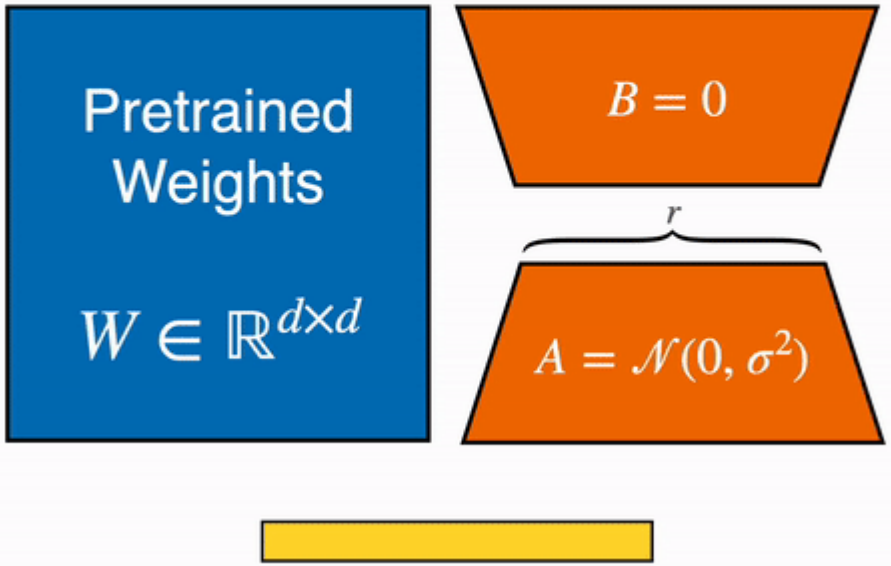
Source: cs231n

- **PEFT（Parameter-Efficient Fine-Tuning，參數高效微調）：**
  - 僅新增很少參數（如 **Adapters**、**LoRA**、**Prefix/Prompt Tuning**、**BitFit**），大幅降低顯存與算力需求。近年的綜述與系統化比較可參考 2024–2025 的 PEFT survey。
- **Domain Adaptation / Domain Generalization：**
  - 面對**分佈轉移（distribution shift）**，利用對抗或統計對齊等方法提升跨域泛化；
- **Test-Time Adaptation（TTA）：**
  - **測試階段即時自適應**（無標註），如 TENT 以**熵最小化**更新 normalization 參數，近年有大量擴展與批判性調查。E.g. 只更新 BN

https://arxiv.org/abs/2411.03687v1

LoRA: low rank adaption

- 將大模型的權重凍結起來，不去訓練他，並在旁邊放一個小模型，Forward 時將大模型與小模型的輸出合併，但 Backward 時只計算小模型的梯度。



reference

- https://dgresearch.github.io/
  - A Tutorial on Domain Generalization

MODEL ENSEMBLES

Model Ensemble

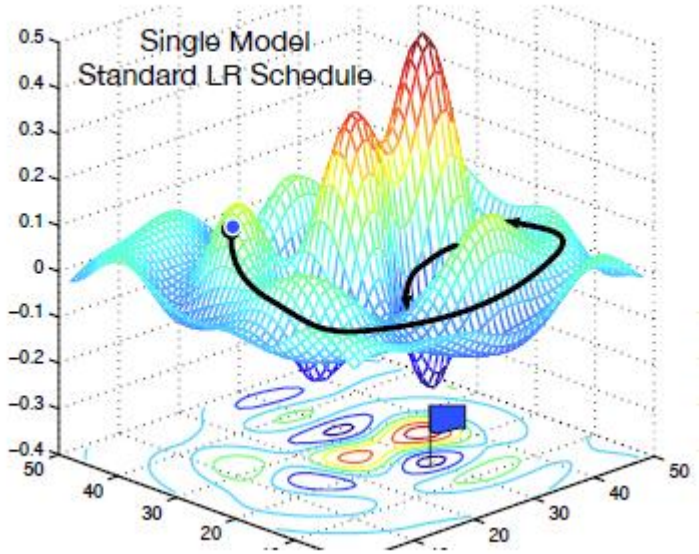
1. Train multiple independent models
2. At test time average their results
   - Enjoy 2% extra performance
   - Widely used in comptitions

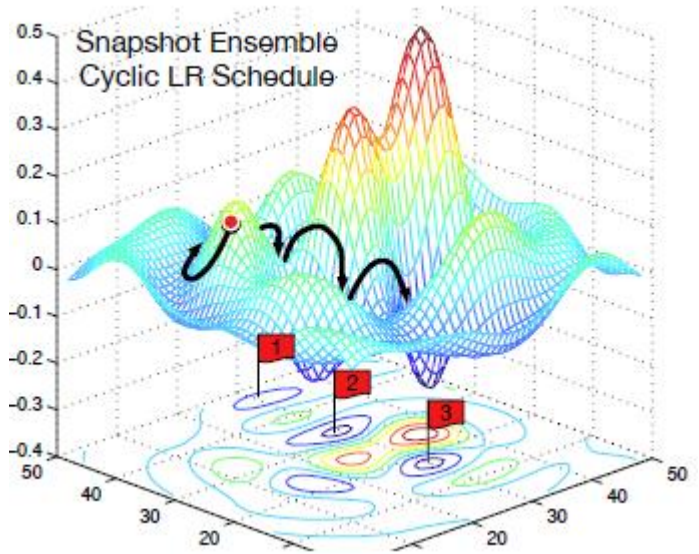
- How to form ensemble beyond multiple independent models?
  - Same model, different initialization
    - Variety only due to initialization
  - Top few models discovered during cross-validation
  - Different checkpoints of a single model
    - Cheap to implement
  - Running average of parameters during training
    - Cheap to implement

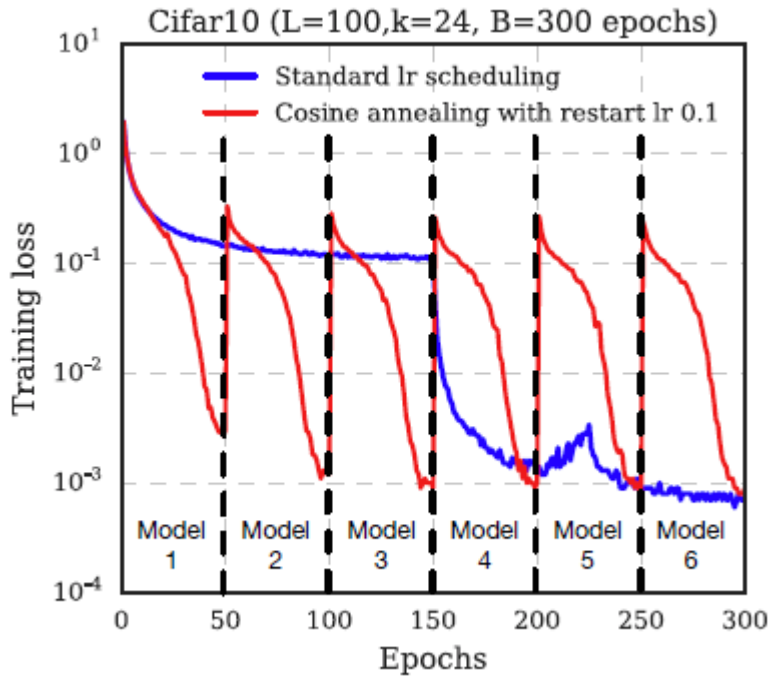
Source: cs231n

Model Ensembles: Tips and Tricks

- Instead of training independent models, use multiple snapshots of a single model during training!







Cyclic learning rate schedules can make this work even better

Loshchilov and Hutter,"SGDR: Stochastic gradient descent with restarts",arXiv 2016
Huang et al,"Snapshot ensembles: train 1, get M for free",ICLR 2017
Source: cs231n

Source: cs231n

Model Ensembles: Tips and Tricks

- Polyak averaging
  - Instead of using actual parameter vector , keep a moving average of the parameter vector and use that at test time
  - Concept
    - optimization algorithm may leap back and forth across a valley several times without ever visiting a point near the bottom of the valley. The average of all of the locations on either side should be close to the bottom of the valley though.
  - How: Exponential decay moving average for nonconvex function

```
while True:
  data_batch = dataset.sample_data_batch()
  loss = network.forward(data_batch)
  dx = network.backward()
  x += - learning_rate * dx
  x_test = 0.995*x_test + 0.005*x # use for test set
```

Polyak and Juditsky, "Acceleration of stochastic approximation by averaging", SIAM Journal on Control and Optimization, 1992.

Source: cs231n

Stochastic Weight Averaging

- Support in Pytorch 1.6 and above

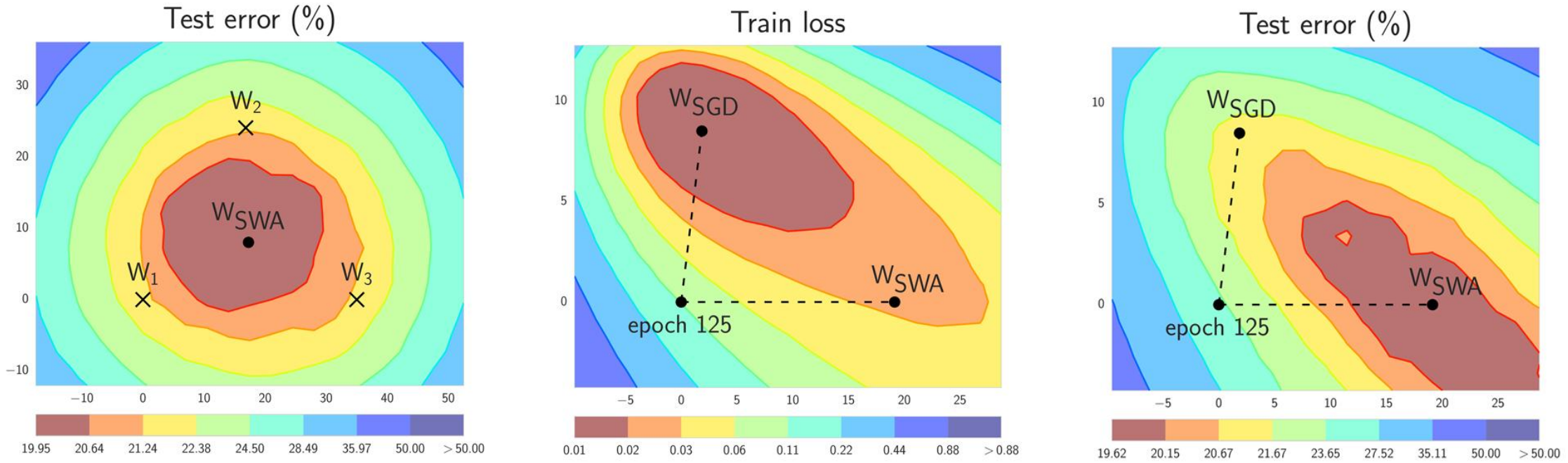
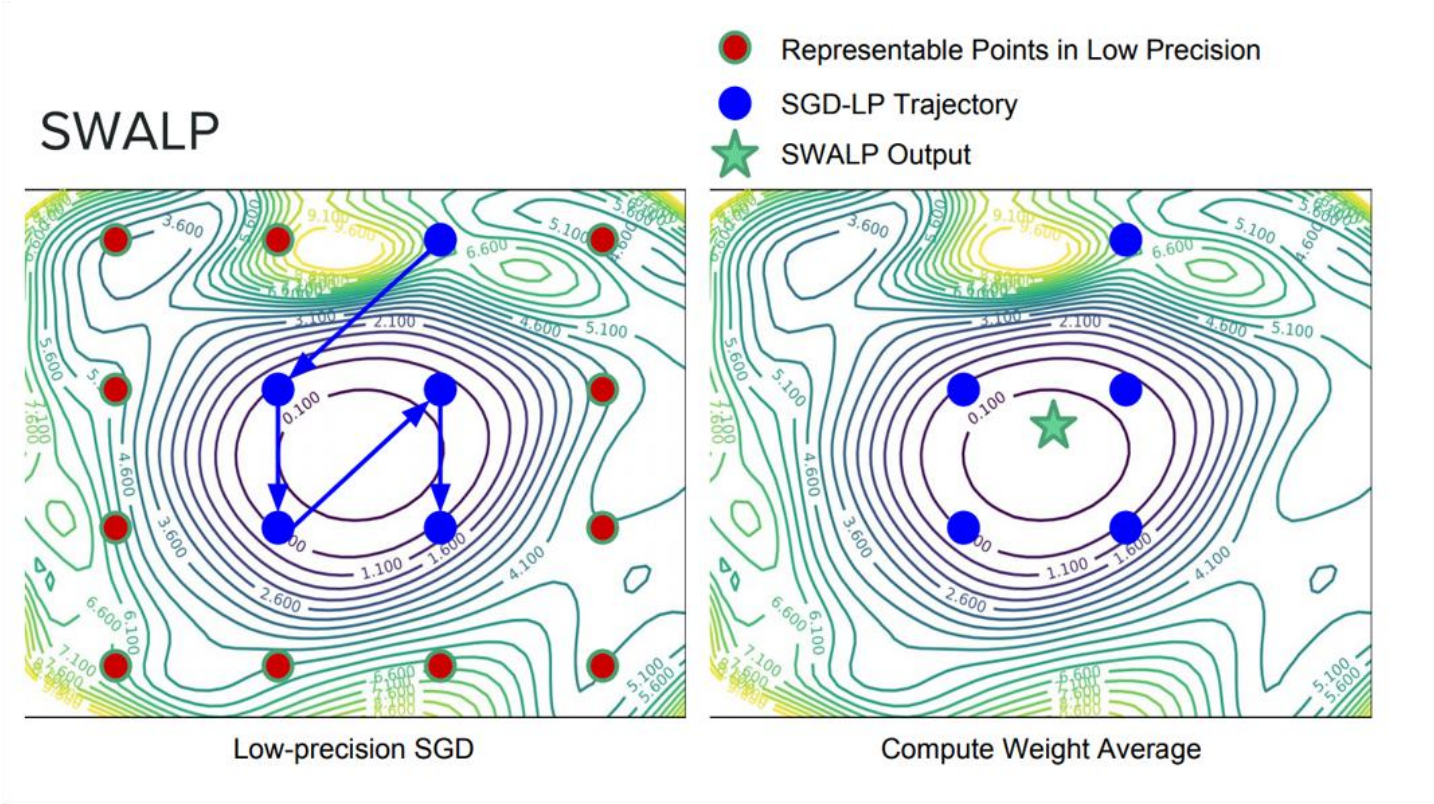


Figure 1. Illustrations of SWA and SGD with a Preactivation ResNet-164 on CIFAR-100 [1]. Left: test error surface for three FGE samples and the corresponding SWA solution (averaging in weight space). Middle and Right: test error and train loss surfaces showing the weights proposed by SGD (at convergence) and SWA, starting from the same initialization of SGD after 125 training epochs.



The difference between standard low precision training and SWALP