Deep Learning Frameworks

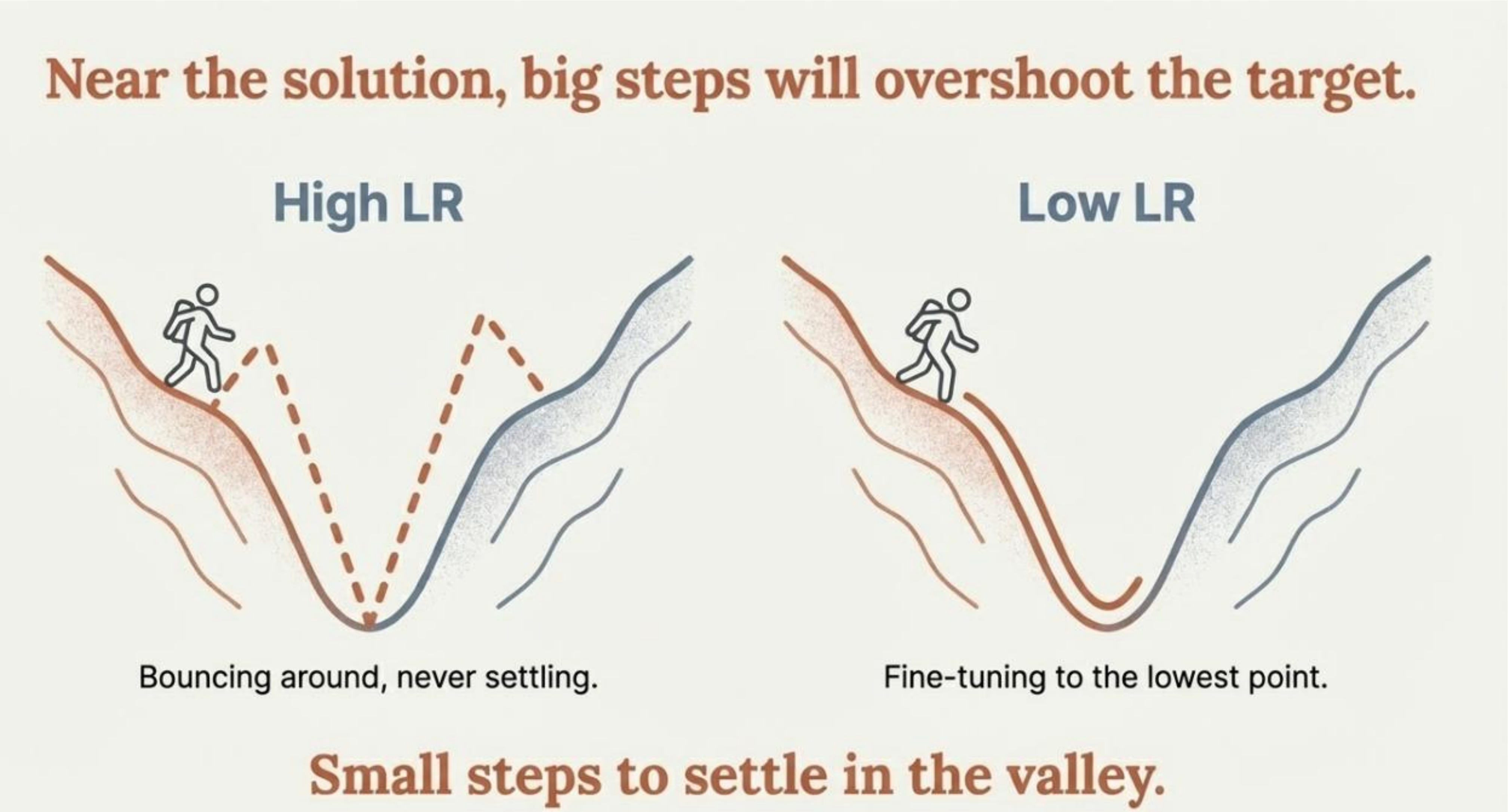
Pretrained Vision Models – VGG and ResNet, Residual Connections, Transfer Learning, Finetuning Pretrained Models, Object Detection, LR Scheduler, Gradient Accumulation

By Sakharam

https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

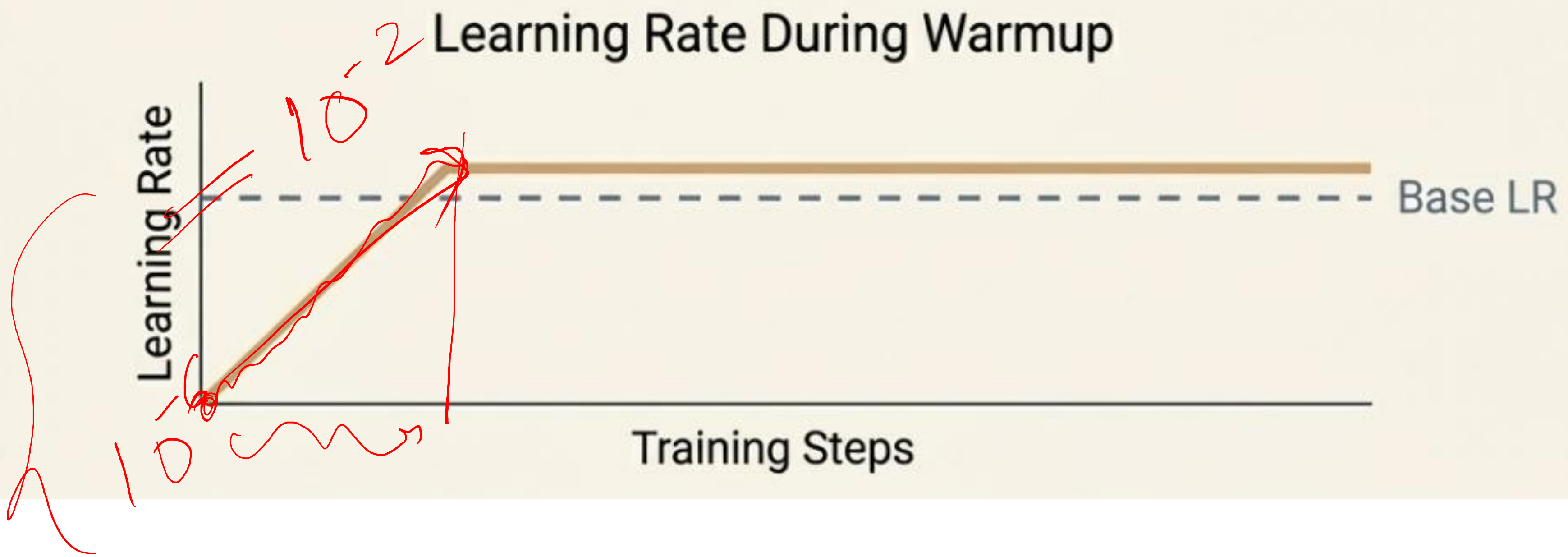
Learning Rate Schedulers: Motivation

Learning Rate Schedulers: Motivation



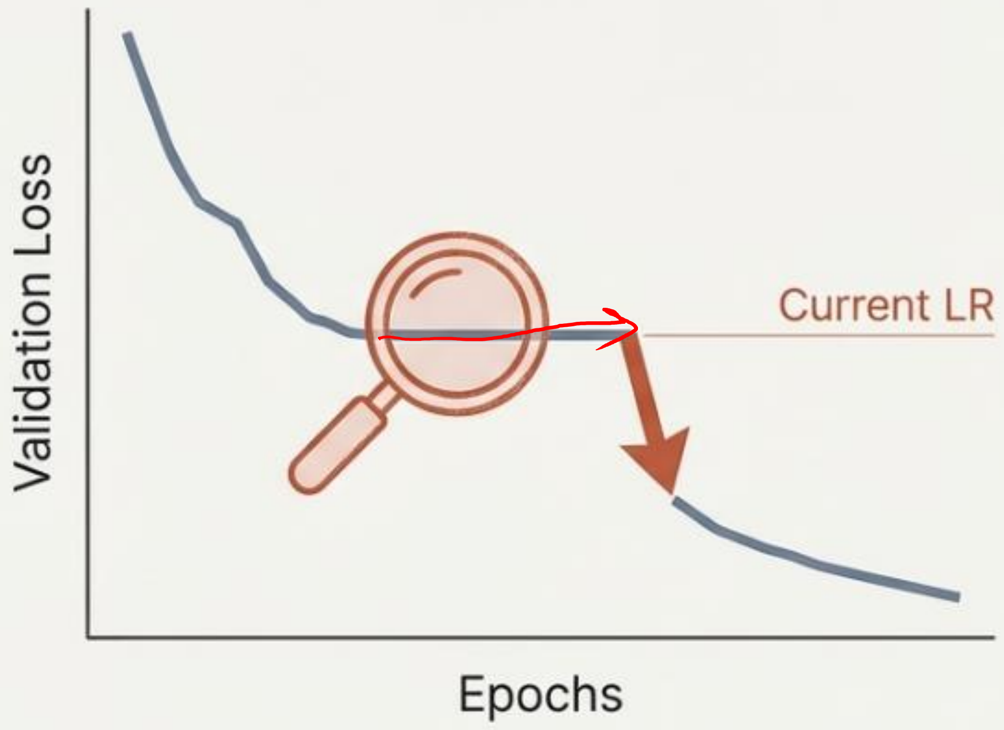
Warm-Up: Motivation

Jumping immediately to a high learning rate can destabilize the model in the very first steps. A "warmup" prevents this by starting with a very small LR and gradually ramping it up to your target base LR over a few epochs. Think of it as gently pressing the accelerator instead of flooring it.

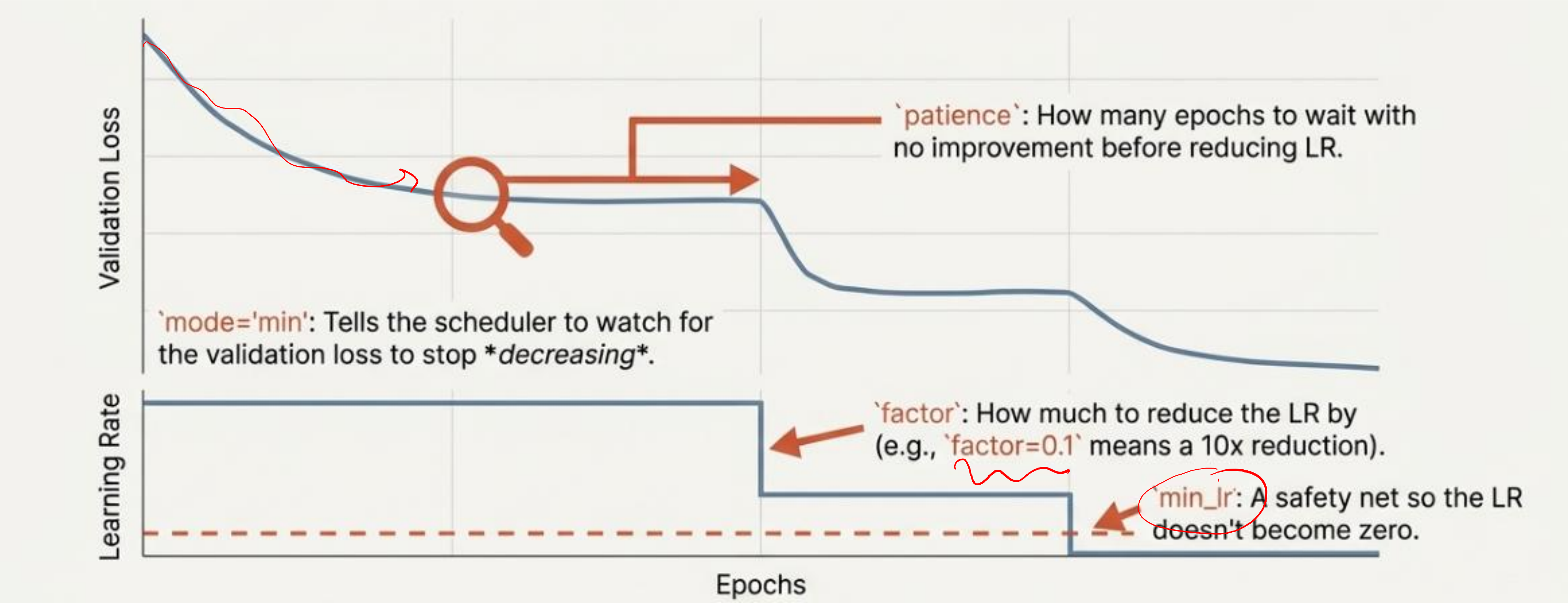


Learning Rate Schedulers: ReduceLROnPlateau

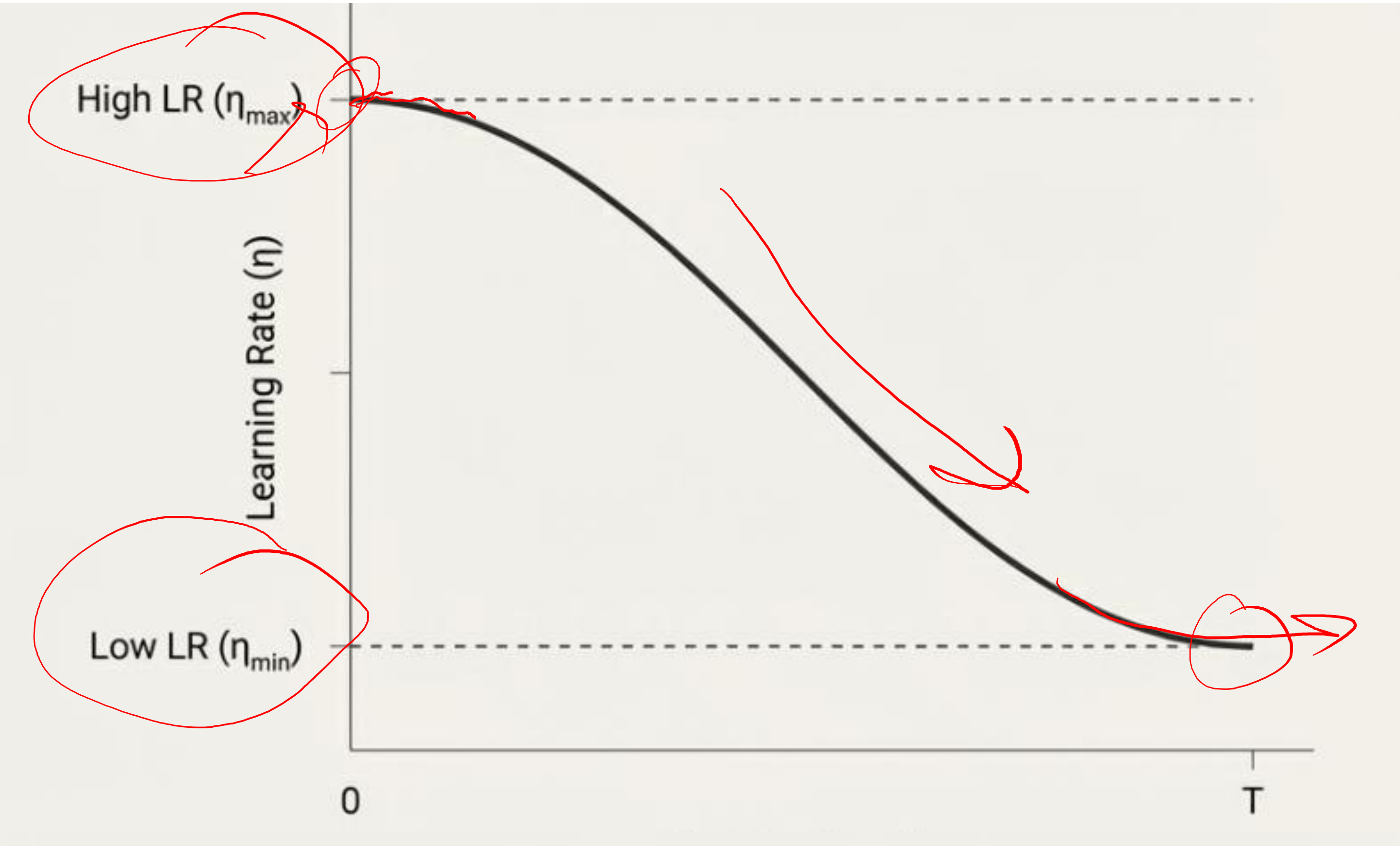
> If our progress on the validation set stalls for a while, it's a sign we're on a **plateau**. Time to reduce our step size and look for a steeper path down.



ReduceLROnPlateau: Working



Learning Rate Schedulers: CosineAnnealing

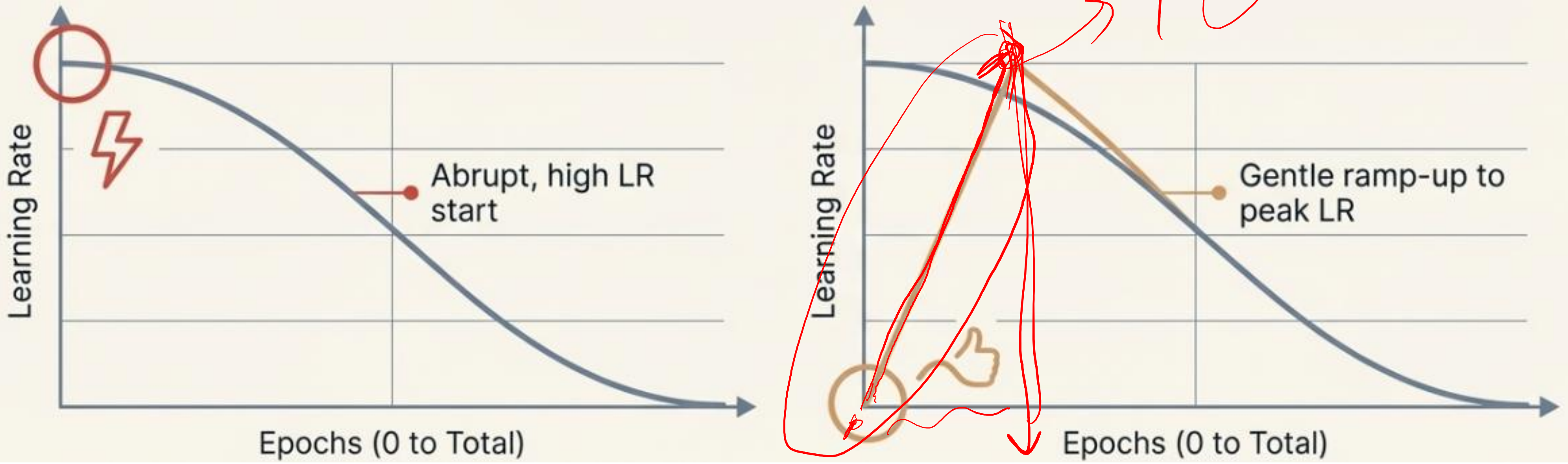


$$\eta(t) = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos(\pi\frac{t}{T})\right)$$

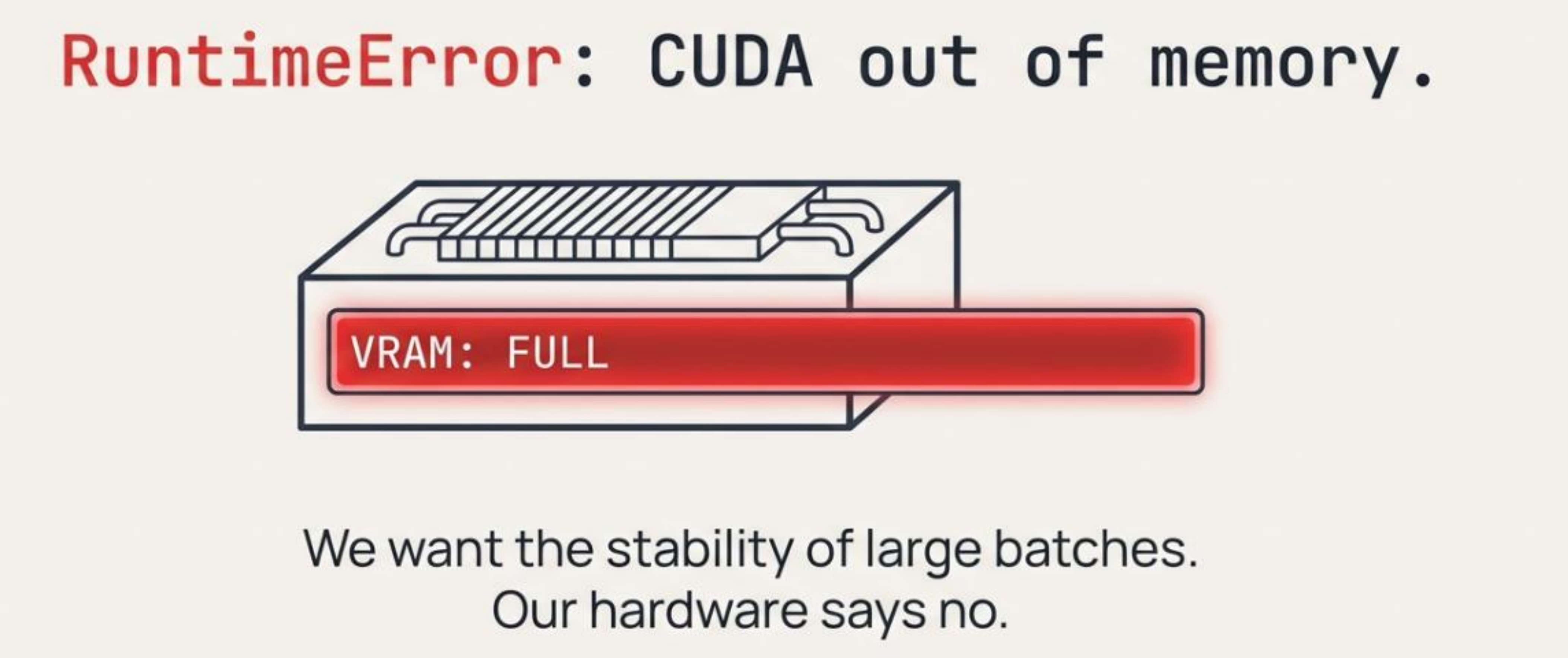
CosineAnnealing with WarmUp: Working

A pure cosine schedule starts at its peak learning rate on the very first step. This can be too aggressive. The warmup provides a short, gentle ramp-up *to* that peak, preventing an initial shock to the model and ensuring a more stable start to the smooth decay.

$5\,10^{-2}$



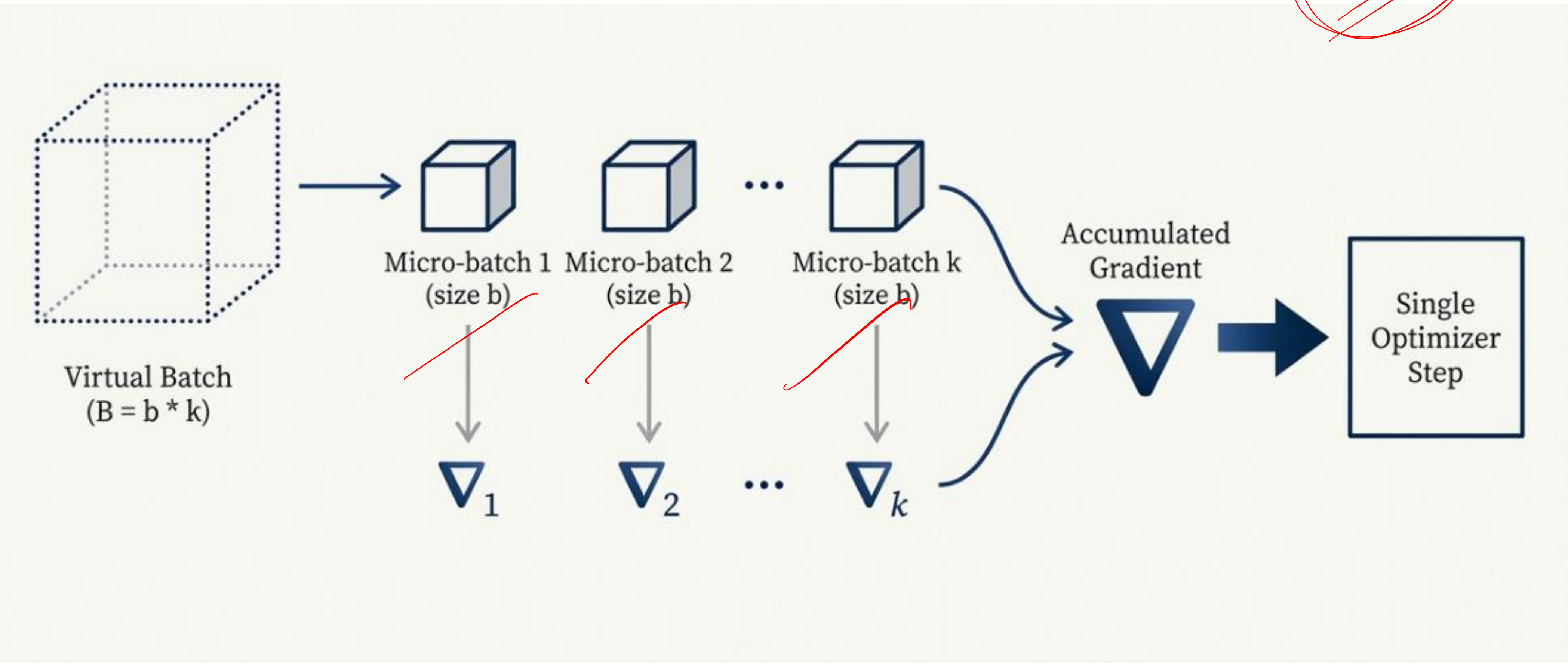
Gradient Accumulation: Motivation



Gradient Accumulation: Working

256 m

64 M



Gradient Accumulation: Code

loop

```
# Standard loop: one step per batch
optimizer.zero_grad()

outputs = model(inputs)
loss = criterion(outputs, labels)
loss.backward()

optimizer.step()
```

256

64
64
64
64

loss
loss
loss
loss

step

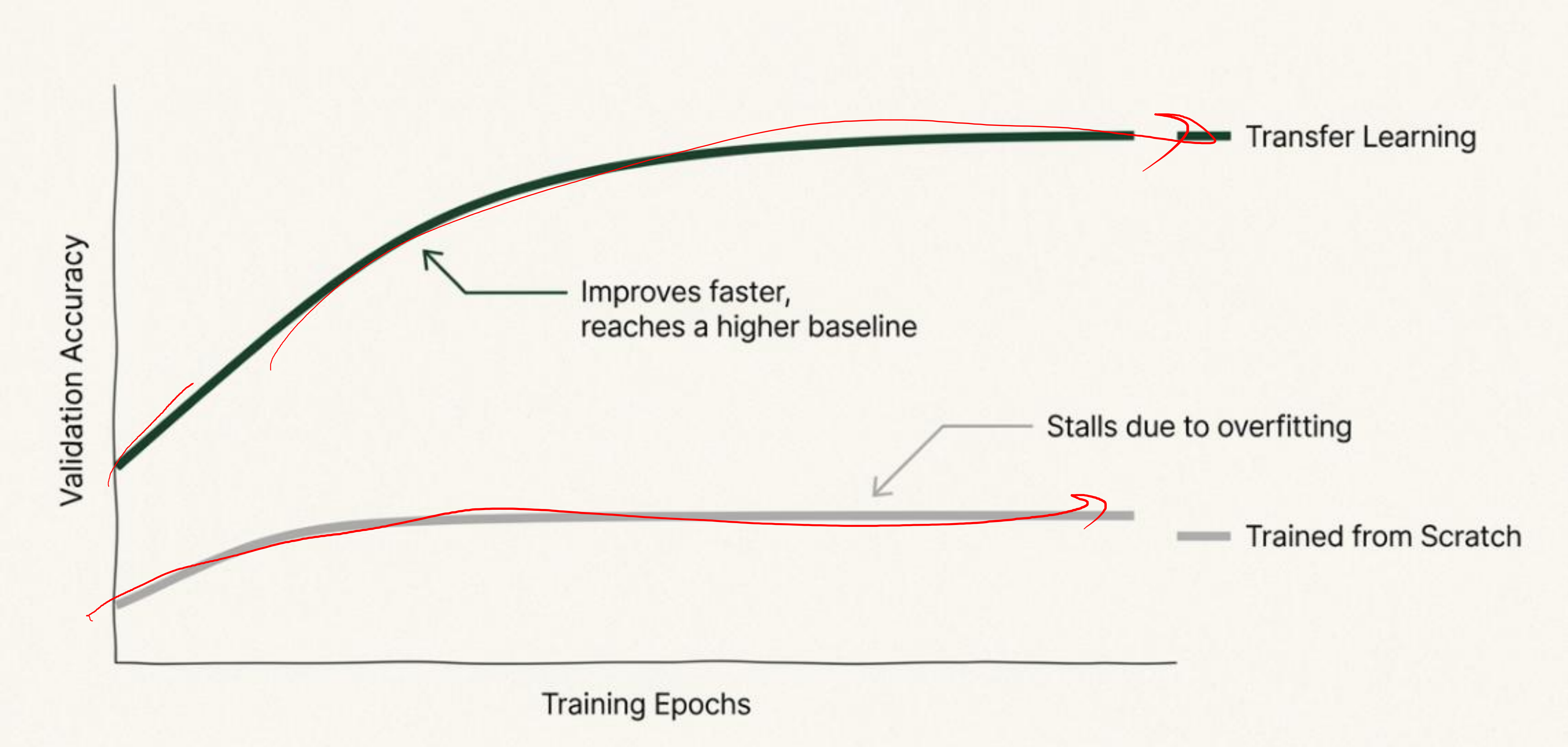
```
# Loop over micro-batches
for i, (inputs, labels) in enumerate(dataloader):
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()   # Grads ADD on each call

    # Perform step after k micro-batches
    if (i + 1) % k == 0:
        optimizer.step()
        optimizer.zero_grad()
```

k=4th

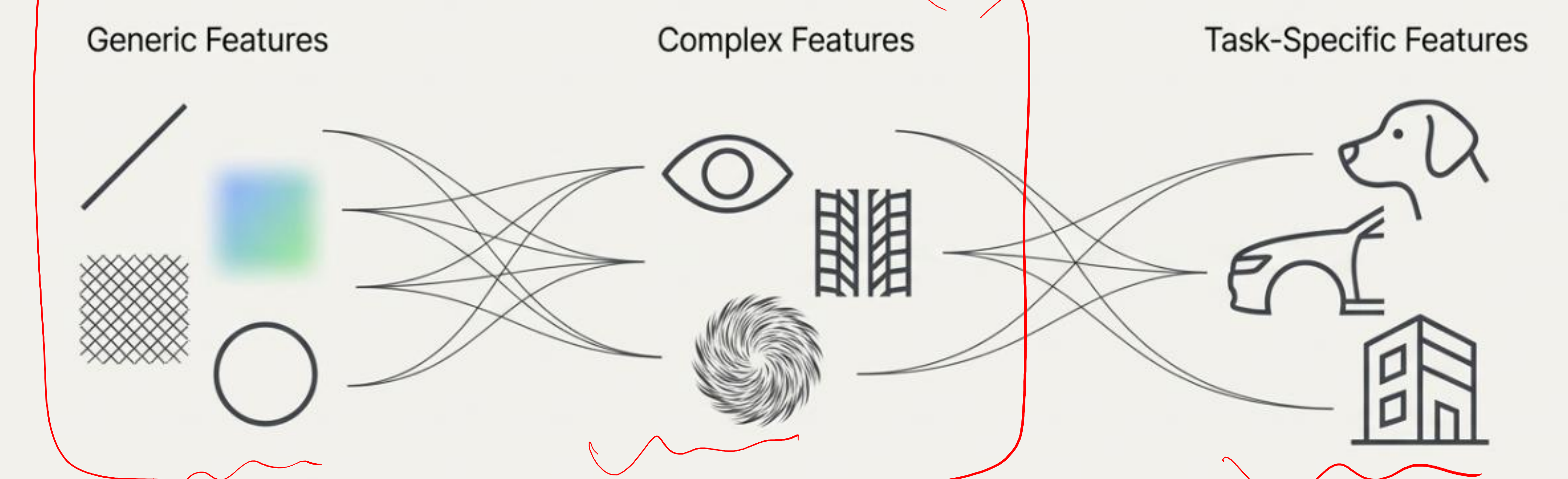
Called `k` times. Gradients are summed in `.grad`.

Optimizer updates and resets only once per 'virtual' batch.

Transfer Learning



Leveraging Pretrained Models



Early layers learn universal visual concepts like edges and textures. By reusing them, we save enormous amounts of data and compute, focusing our effort on learning only the final, task-specific patterns.

Strategy 1: Feature Extraction — The Frozen Backbone

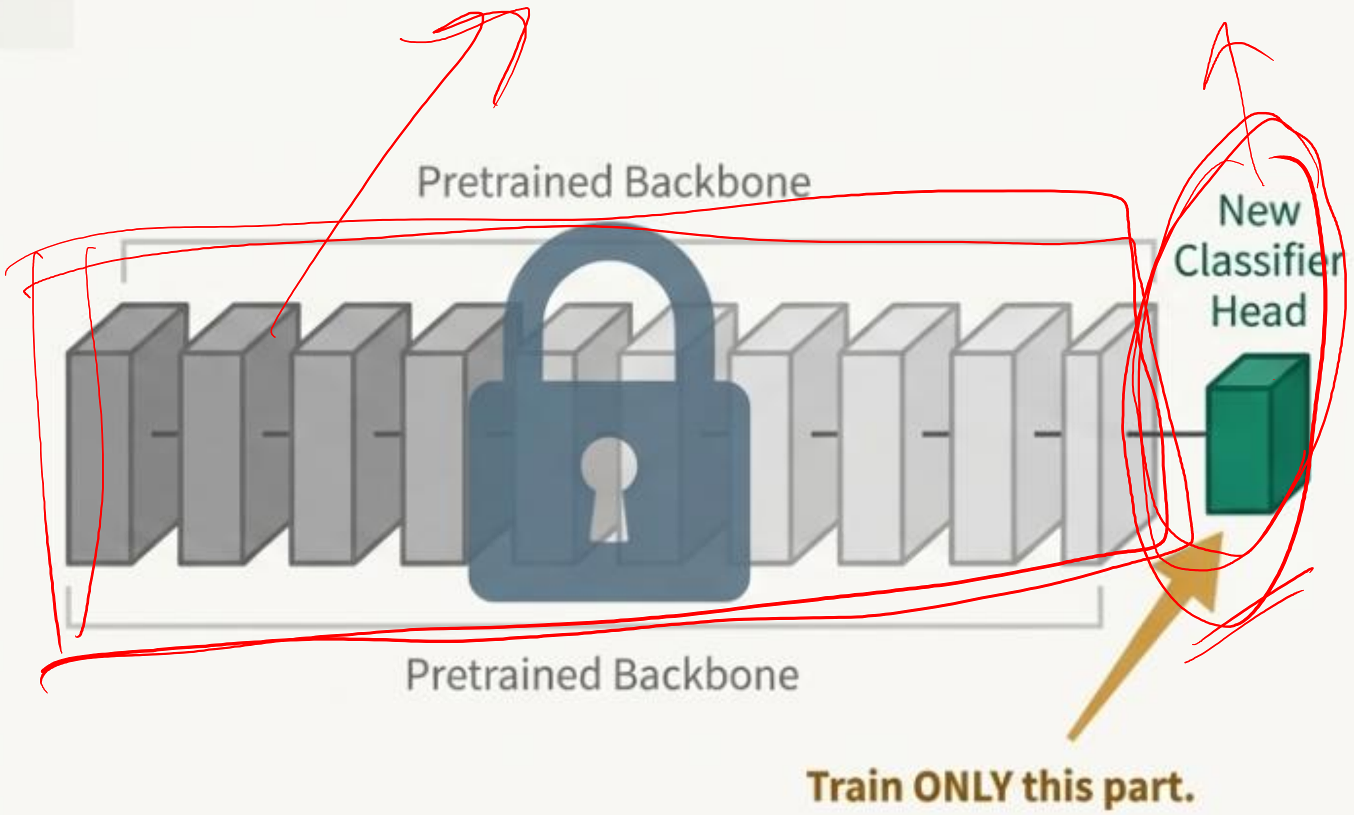
How It Works

1. Take a pretrained backbone (e.g., ResNet50).
2. **Freeze all its weights.** They will not be updated during training.
3. Remove the original final classifier layer.
4. Add a new classifier "head" with outputs matching your number of classes.
5. **Train only the new head.**

When to Use It

- When your dataset is small.
- When compute resources are limited. This method is very fast.

> The backbone becomes a fixed feature generator; your head simply learns the final mapping from these powerful features to your classes.



Strategy 2: Fine-Tuning — Adapting Features with a Small Learning Rate

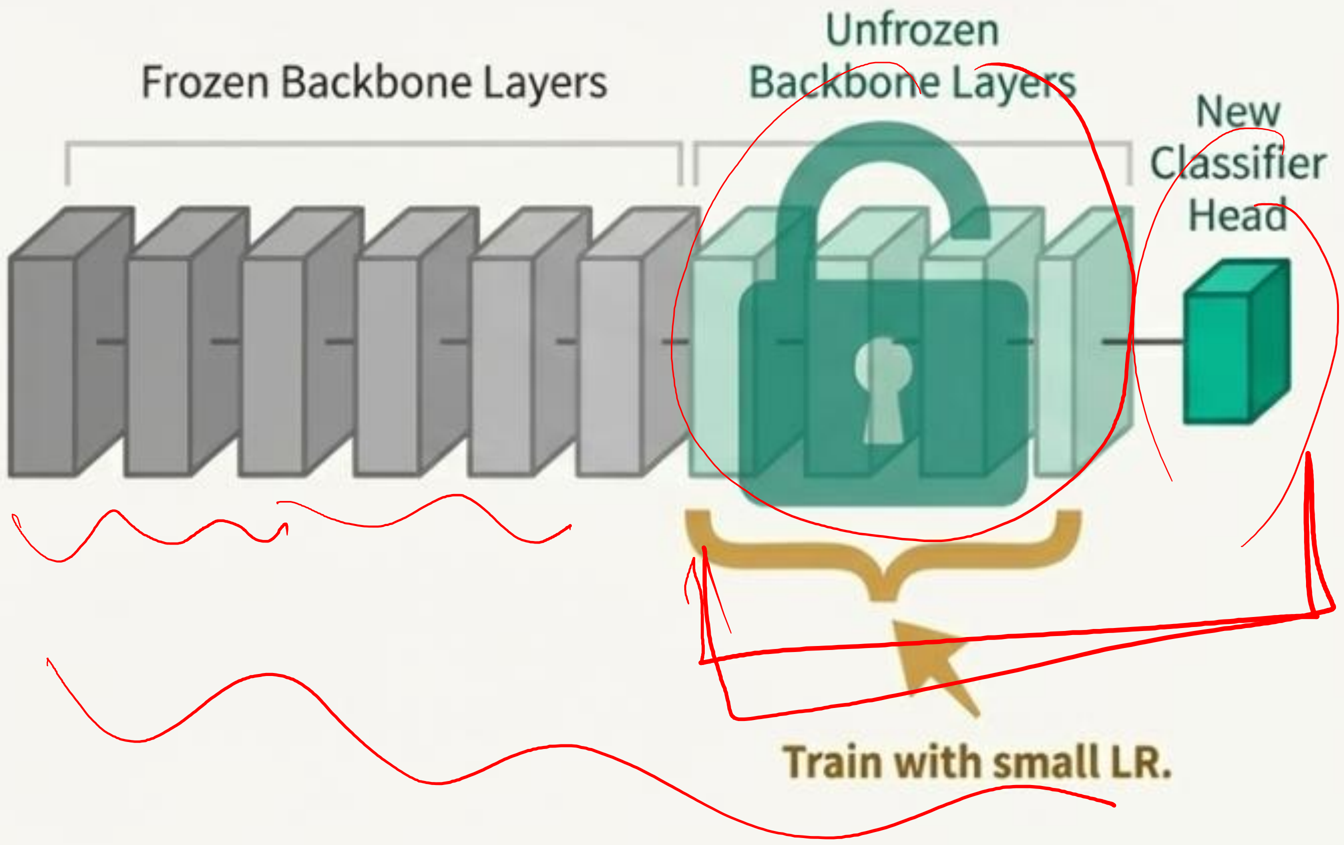
How It Works

1. Start with the same pretrained backbone and new head.
2. **Unfreeze the final block (or more) of the backbone.**
3. **Train the entire network** (head and unfrozen blocks) **with a very small learning rate.**

When to Use It

- When you have more data.
- When your domain differs significantly from ImageNet (e.g., medical scans, satellite imagery).

> Fine-tuning adapts the pretrained features to the specific nuances of your dataset. The small learning rate prevents catastrophic forgetting of the valuable pretrained knowledge.

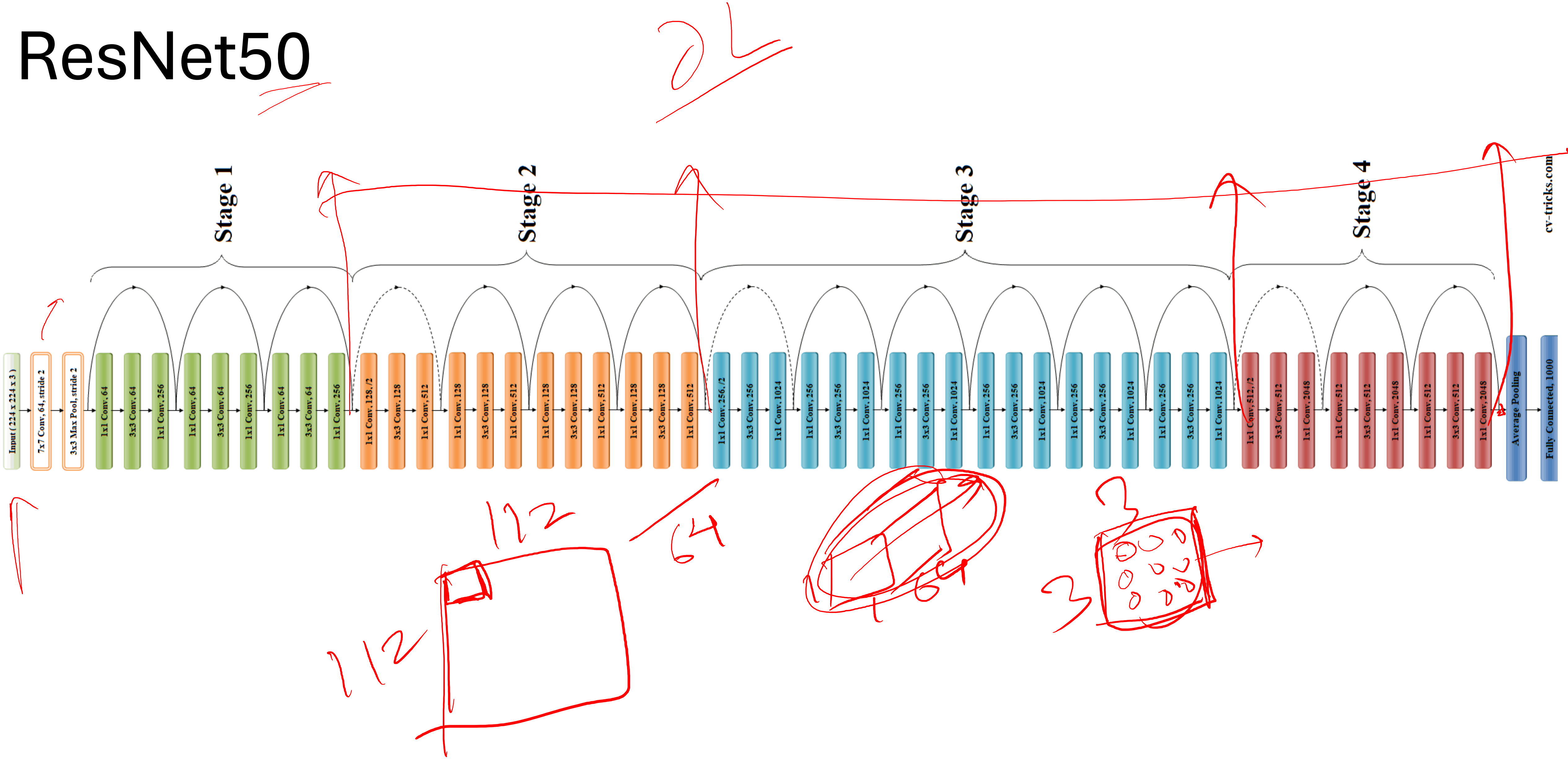


VGG and ResNet

VGG16


3x3 conv, 64
3x3 conv, 64
pool/2
3x3 conv, 128
3x3 conv, 128
pool/2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool/2
fc 4096
fc 4096
fc 4096
Size:224
Size:112
Size:56
Size:28
Size:14
Size:7

ResNet50



Stage 1
Stage 2
Stage 3
Stage 4
cv-tricks.com
Input (224 x 224 x 3)
7x7 Conv, 64, stride 2
3x3 Max Pool, stride 2
1x1 Conv, 64
3x3 Conv, 64
1x1 Conv, 256
1x1 Conv, 128, /2
3x3 Conv, 128
1x1 Conv, 512
1x1 Conv, 256, /2
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 512, /2
3x3 Conv, 512
1x1 Conv, 2048
Average Pooling
Fully Connected, 1000

Thank You

H x W

K x K

RCNN → FasterRCNN

Region

Object Obj

Resnet

Appendix