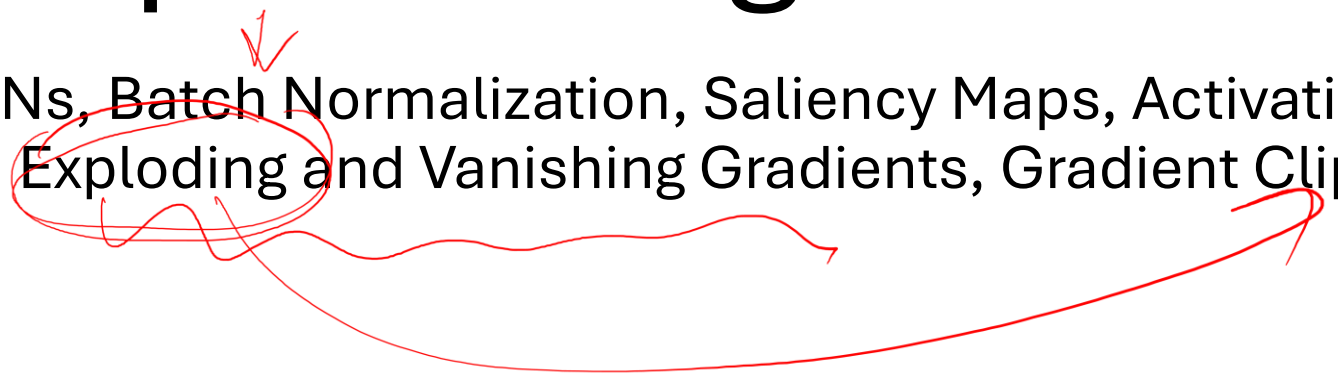
Deep Learning Frameworks

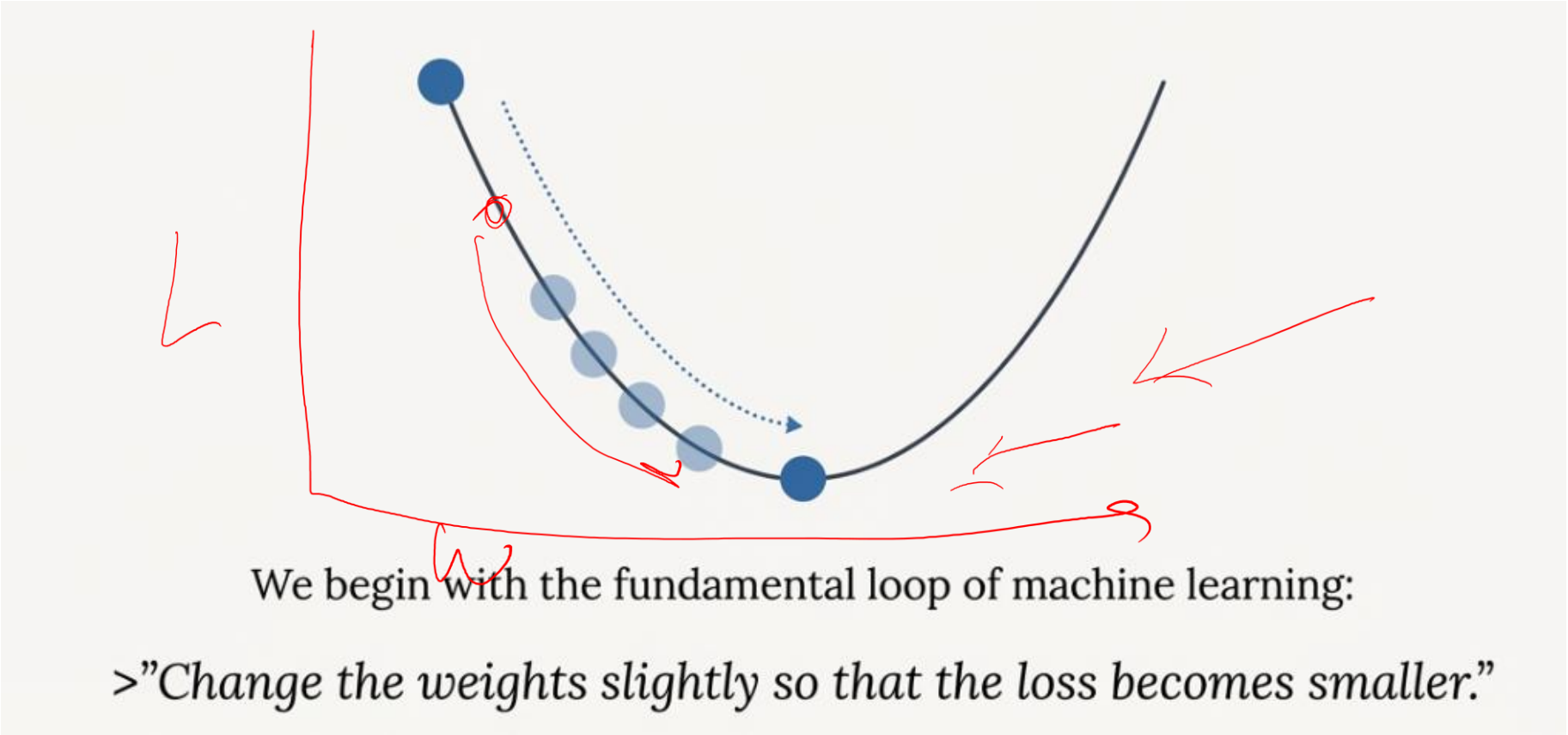
CNNs, Batch Normalization, Saliency Maps, Activation Maps,
Exploding and Vanishing Gradients, Gradient Clipping



By Sakharam

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

Recap: Learning



We begin with the fundamental loop of machine learning:

>"*Change the weights slightly so that the loss becomes smaller.*"

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

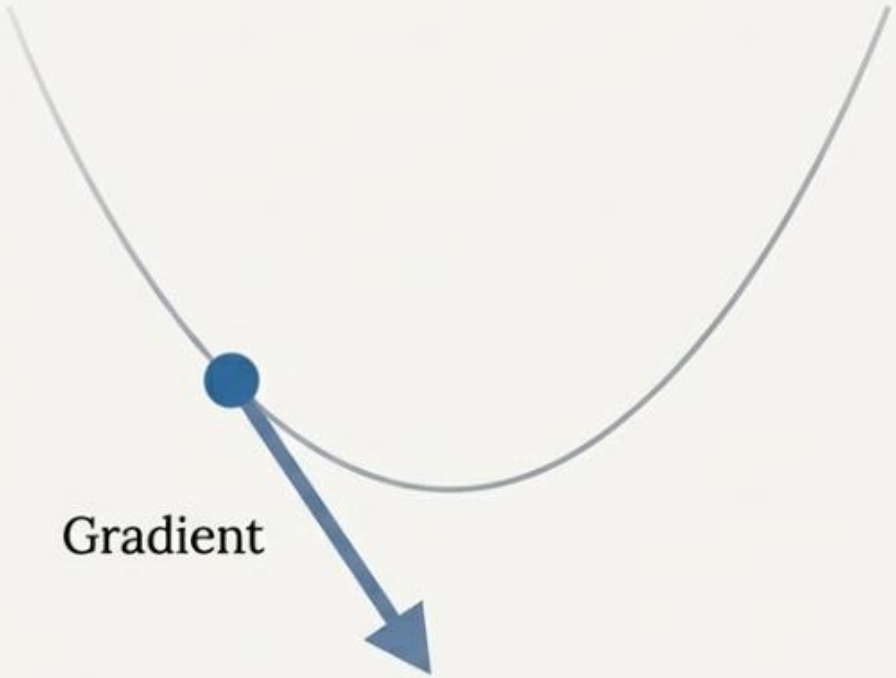
Recap: Gradients

The gradient provides two critical pieces of information for each weight:

1. **Direction**: Which way to change the weight.
2. **Strength**: How much to change it.

$$\texttt{gradient} = \frac{\partial \texttt{loss}}{\partial W}$$

> *"If you increase or decrease this weight a little, how will the loss change?"*



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

Recap: Weight Update

A simple weight update rule like Stochastic Gradient Descent (SGD) uses the gradient directly:

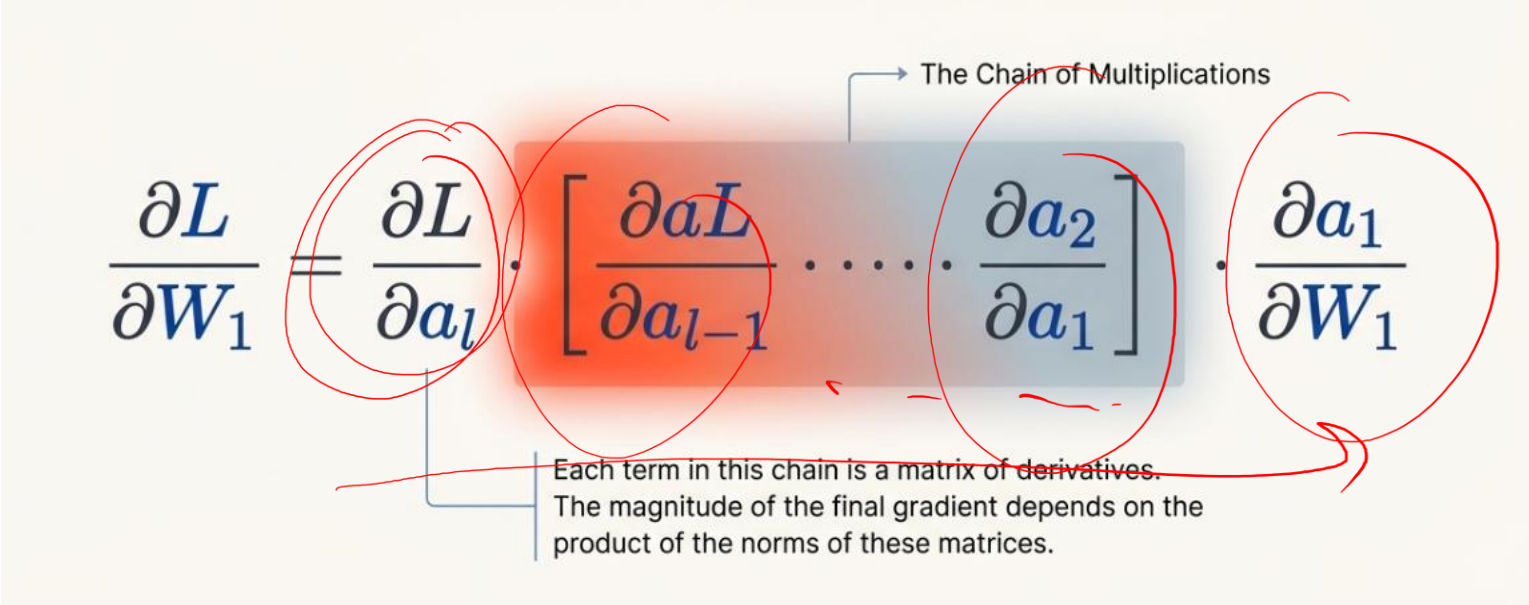
$$W \leftarrow W - \eta \cdot \nabla_W \mathcal{L}$$

The Gradient provides the **direction**.

Learning Rate × Gradient Magnitude determines the **step size**.

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

Recap: Chain Rule



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

This instability creates two critical failure modes.

VANISHING GRADIENTS

Mechanism: The product of derivatives shrinks exponentially towards zero.

Result: Gradients for early layers become too small to provide a meaningful learning signal. The network effectively stops learning at its base.

EXPLODING GRADIENTS

Mechanism: The product of derivatives grows exponentially towards infinity.

Result: Gradients become massive, causing wildly unstable weight updates and leading to numerical overflow (`NaN` values).

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

Exploding Gradients

Exploding gradients cause the model to take giant, chaotic steps and break the learning process.

The Breakdown:

1. **Huge Weight Updates**: Parameters are shifted by massive amounts.
2. **Wild Jumps**: The model is thrown across the parameter space.
3. **Unstable Loss**: The loss value fluctuates wildly and increases.
4. **Divergence**: Training fails, resulting in `nan` or `inf` values.

"The model tries to take a giant step, overshoots, and breaks learning."

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

“Clipping”

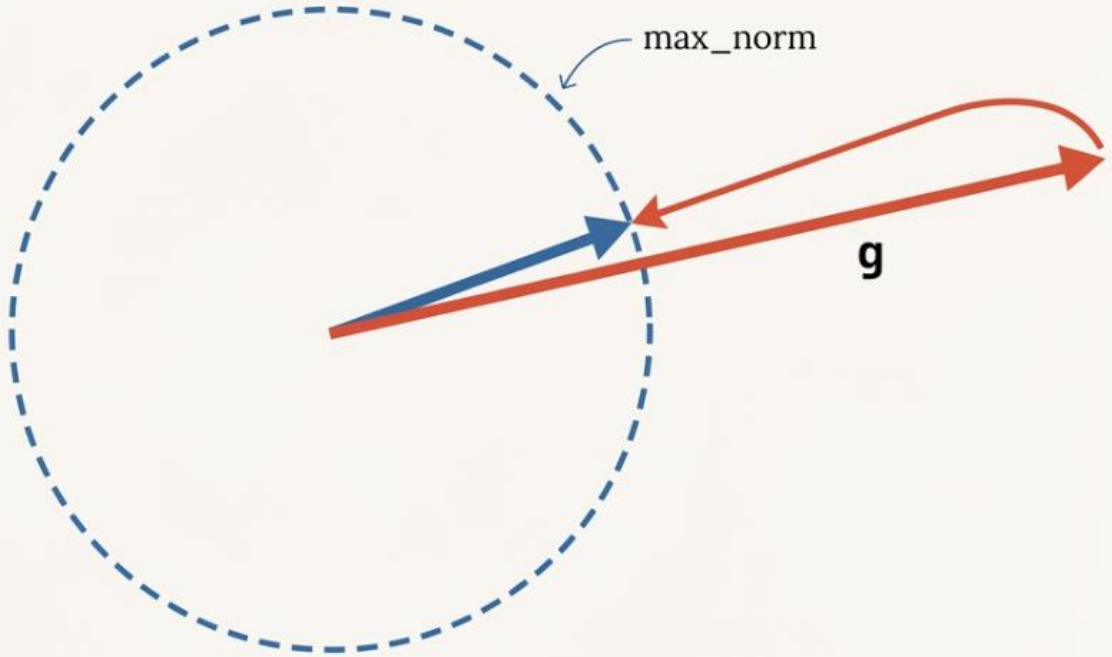
If the gradient vector is too long, we shorten it; otherwise, we leave it untouched.

The Clipping Rule:

If $\|\mathbf{g}\| \leq \text{max\_norm}$ → **Do nothing.**

If $\|\mathbf{g}\| > \text{max\_norm}$ → **Scale down the entire vector.**

$$\mathbf{g} \leftarrow \mathbf{g} \cdot \left(\frac{\text{max\_norm}}{\|\mathbf{g}\|}\right)$$



📌 **Direction stays the same — only magnitude is reduced.**

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

Learning Rate vs Gradient Clipping

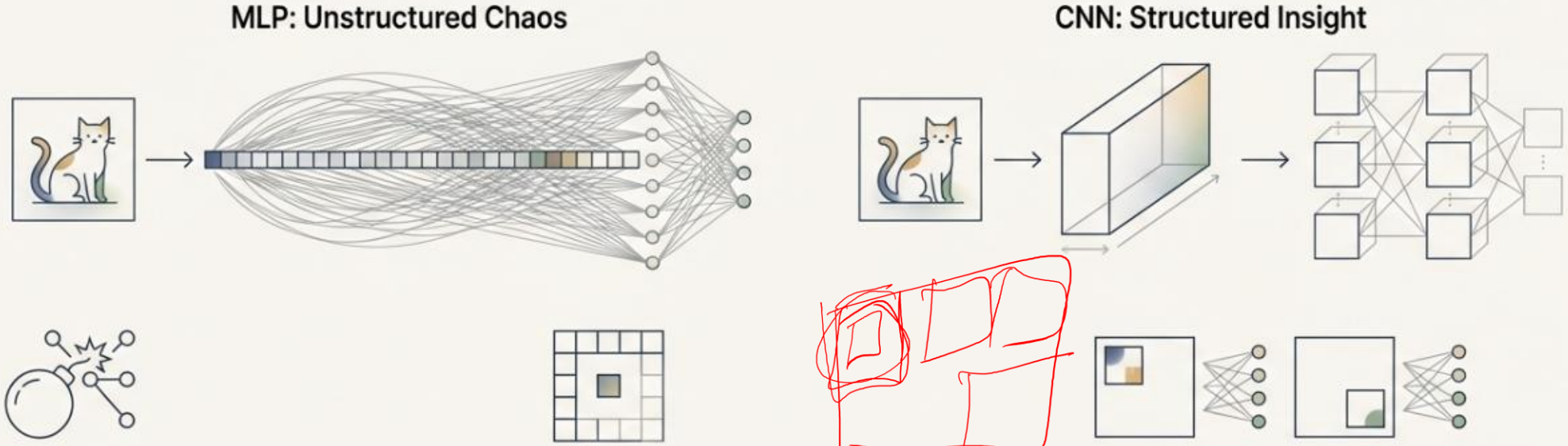
Learning rate decides how fast you learn.

Gradient clipping decides how unsafe you're allowed to be.

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

CNNs

An MLP treats an image as a flat list of numbers, ignoring its inherent structure. This leads to three fundamental breakdowns.



1. Parameter Explosion

A single layer mapping a 224x224 image to 1,000 neurons requires

~150 million weights.

This is inefficient and data-hungry.

2. Ignorance of Locality

MLPs don't understand that nearby pixels are related. They miss the local patterns—edges, textures, corners—that form objects.

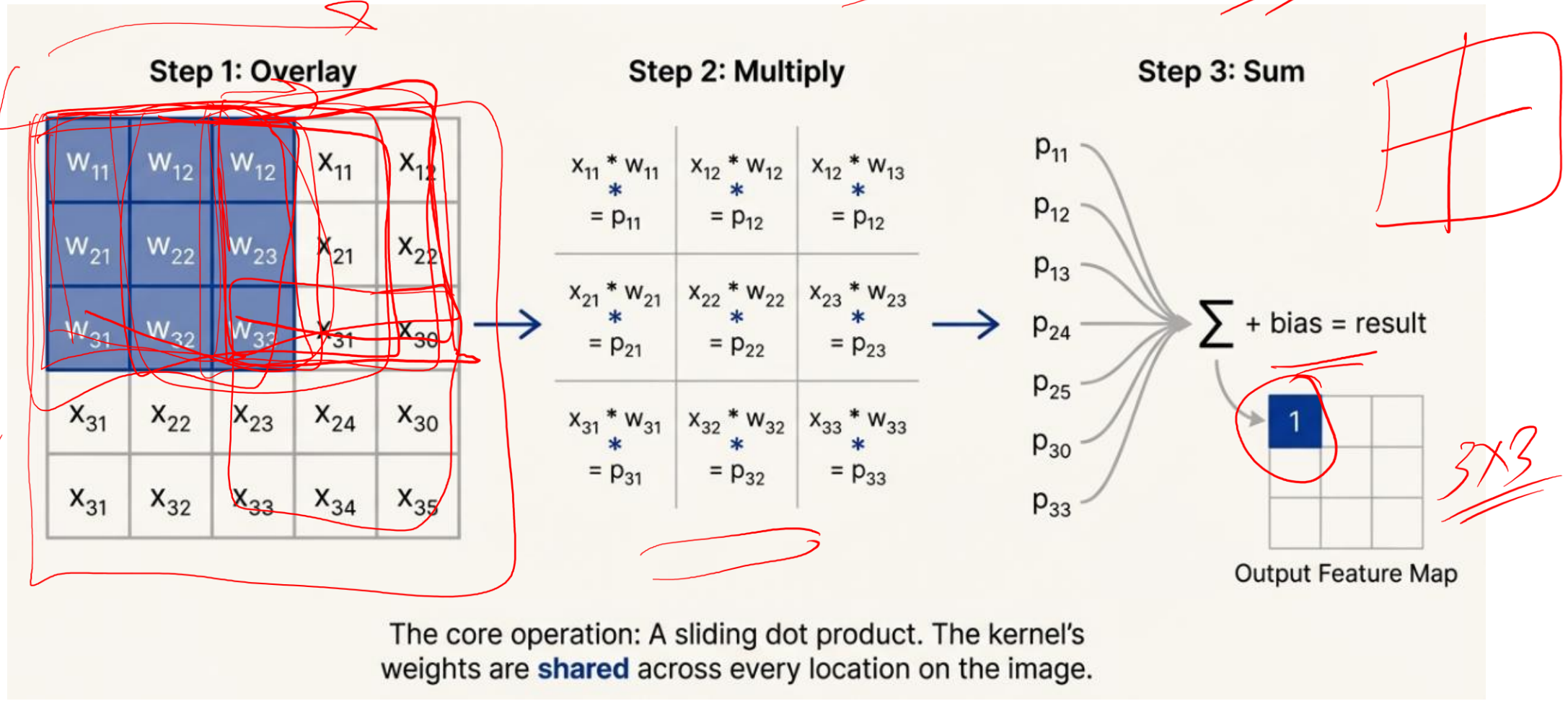
3. Lack of Translation Invariance

An object shifted by a few pixels is a completely new input to an MLP. The network has no built-in understanding that it's the same object in a new place.

https://tinyurl.com/dlframeworks

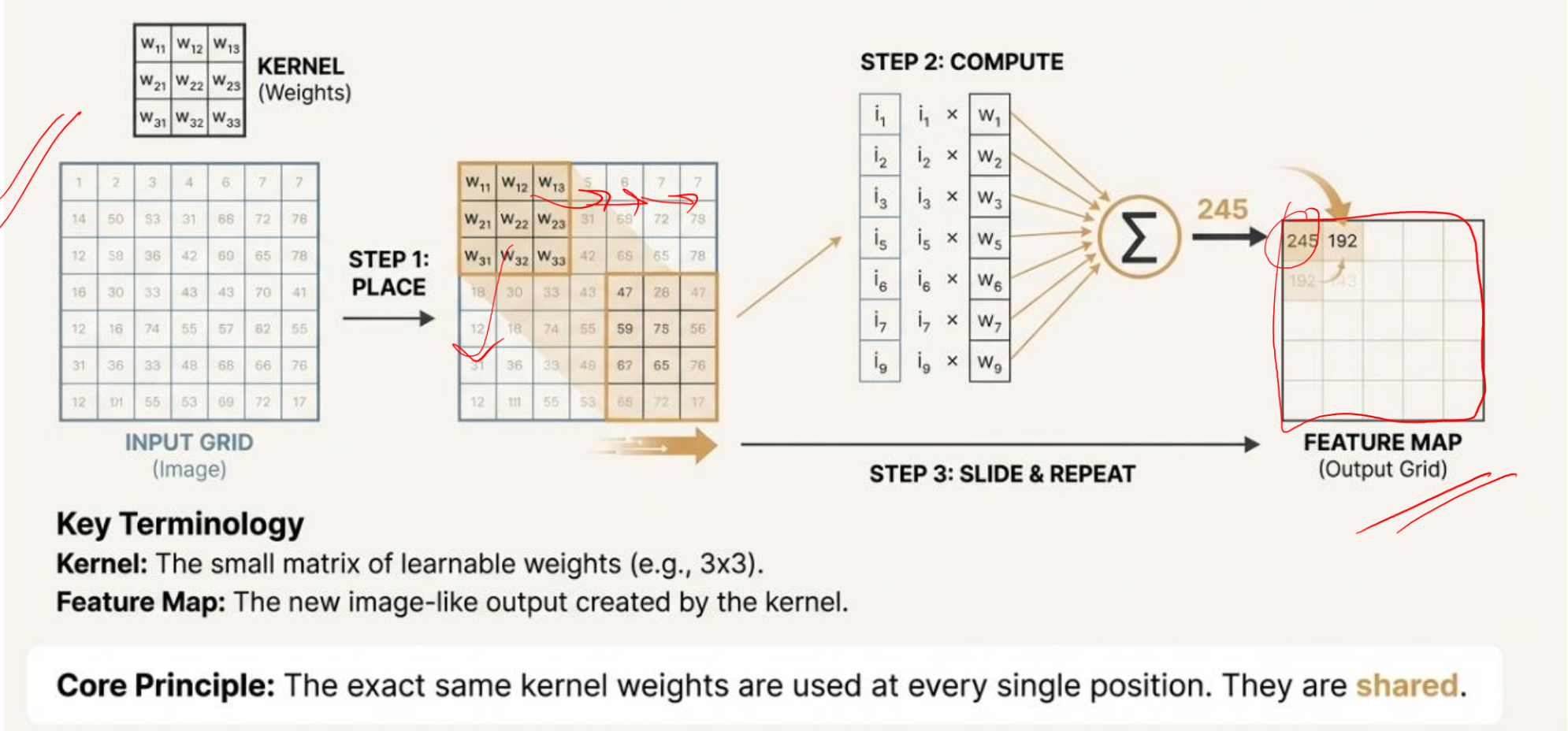
https://github.com/sakharamg/DeepLearningFrameworks

Kernels



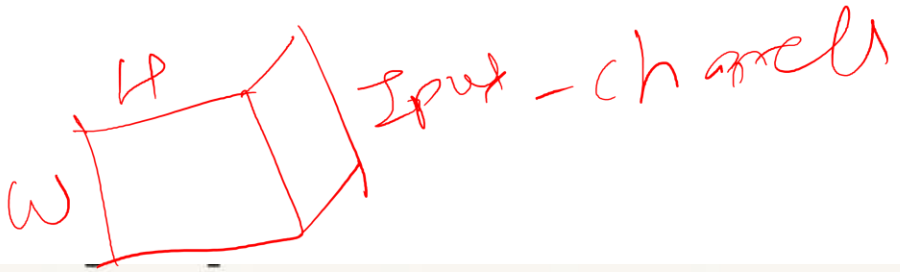
https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

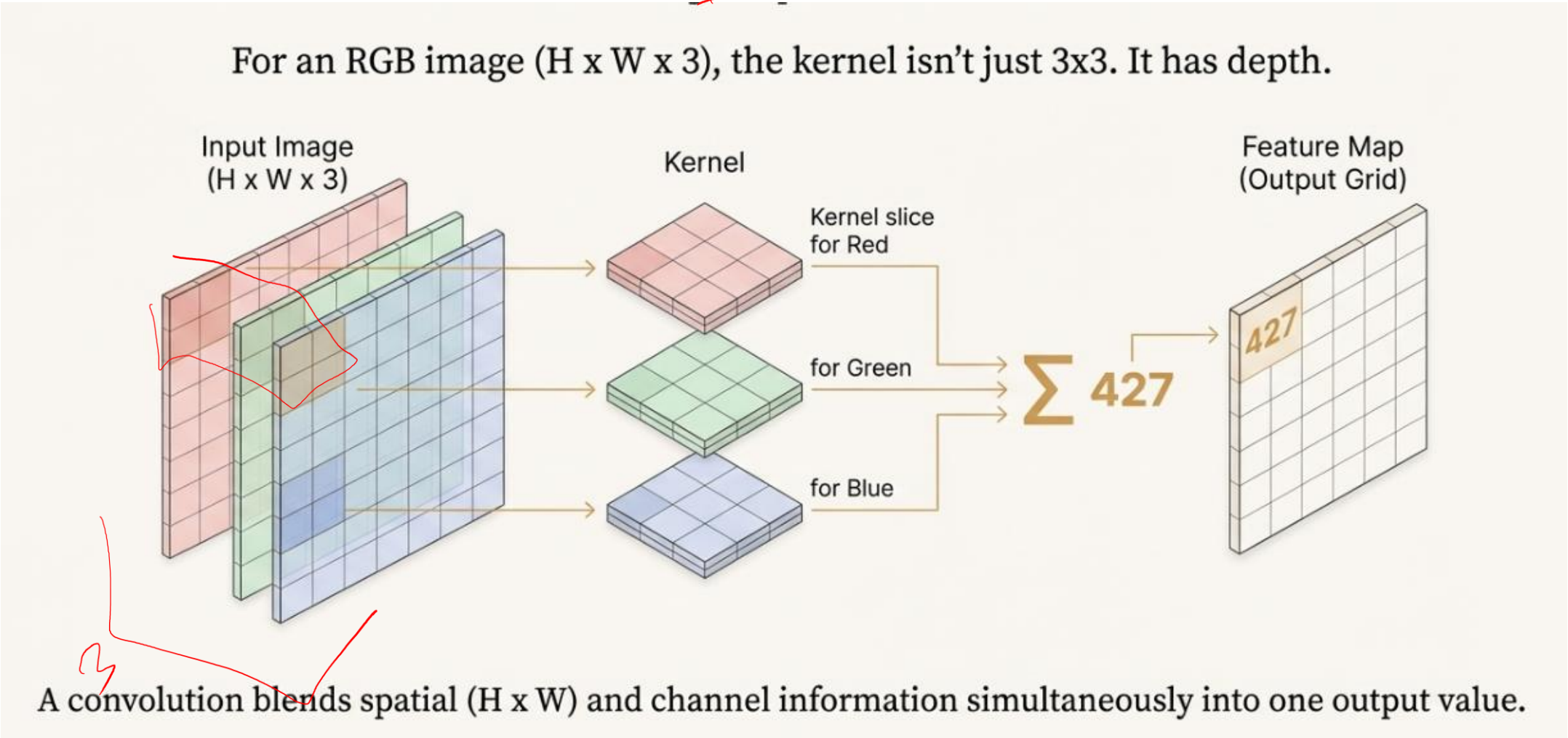
Kernels



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

Kernels



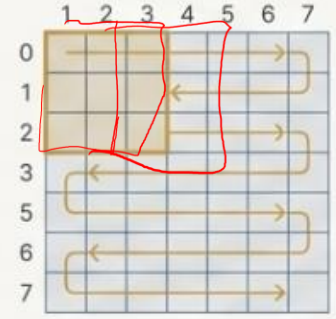


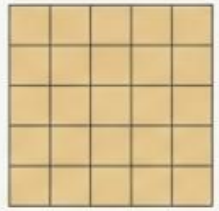
https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Stride and Padding

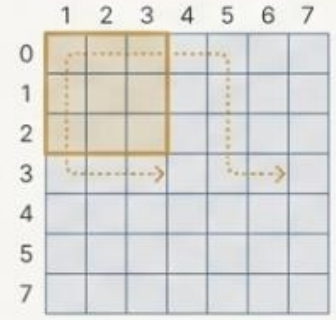
Stride (S) - The step size.

How many pixels the kernel moves at each step.





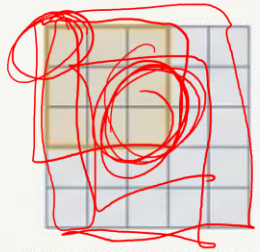
Common for preserving detail.



Used for downsampling.

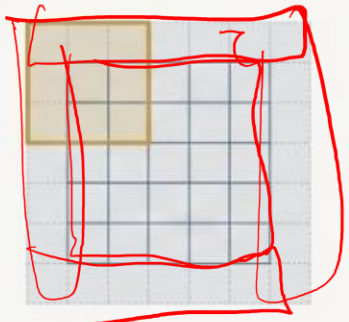
Padding (P) - Handling the borders.

Adding a border (usually of zeros) around the input image.



Valid Convolution

No padding. The output size shrinks.



Common in modern CNNs.

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - K + 2P}{S} \right\rfloor + 1$$

Example: For a 32x32 input, 3x3 kernel (K=3), Padding=1, Stride=1, the output is 32x32.

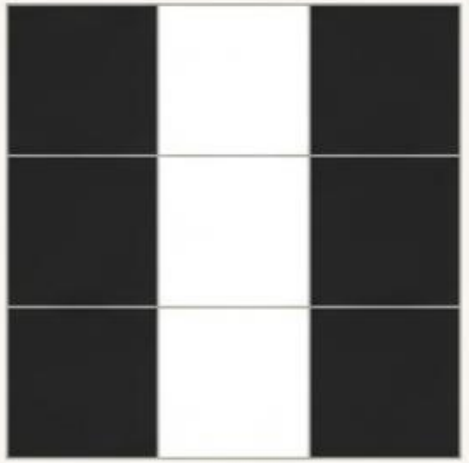
https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Kernel: Vertical Lines

THE KERNEL

$$k_v = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

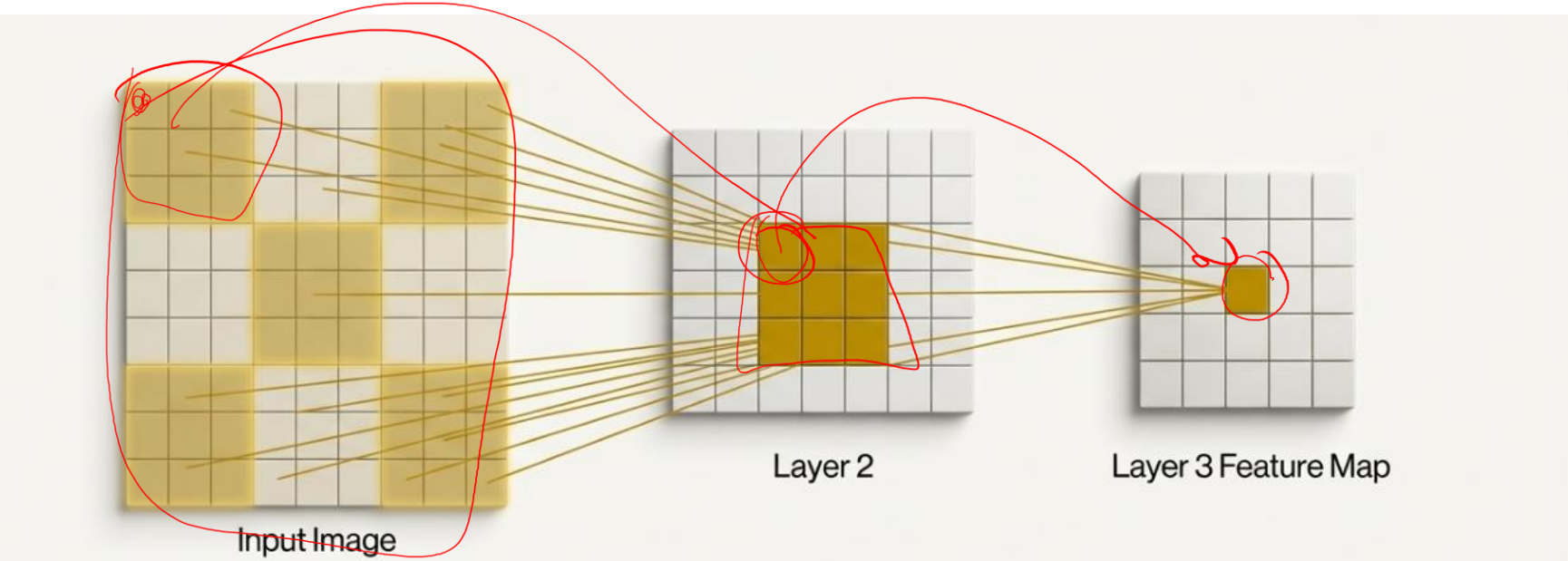
THE PATTERN



This kernel fires when the **middle column is bright** compared to the left and right columns. The logic is identical, just applied across columns instead of rows:

```
y = 2*(sum of middle column) - 1*(sum of left column) - 1*(sum of right column)
```

Local to Global

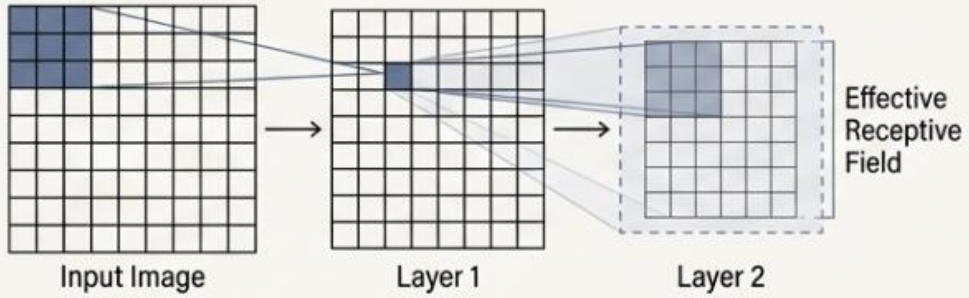


Receptive Field: The specific region of the input image that a particular neuron in the network is "looking at." Stacking layers causes the receptive field to grow, allowing the network to integrate local features into global concepts.

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

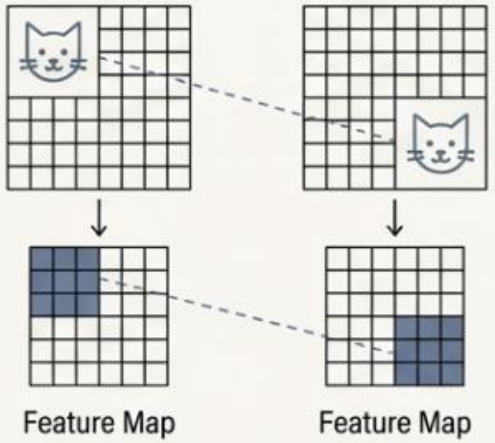
Kernels

(1) Local Receptive Fields



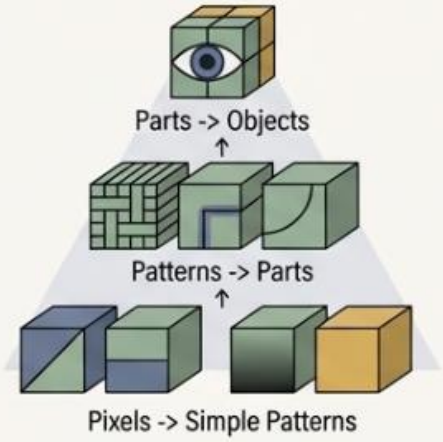
Each neuron in a feature map only 'sees' a small patch of the input. Early layers learn simple patterns (edges, colors). Deeper layers combine these to 'see' larger, more complex structures (eyes, wheels, faces).

(2) Translation Equivariance



If an object shifts in the input, its representation in the feature map shifts by the same amount. The network's understanding moves *with* the object.

(3) Hierarchical Representation

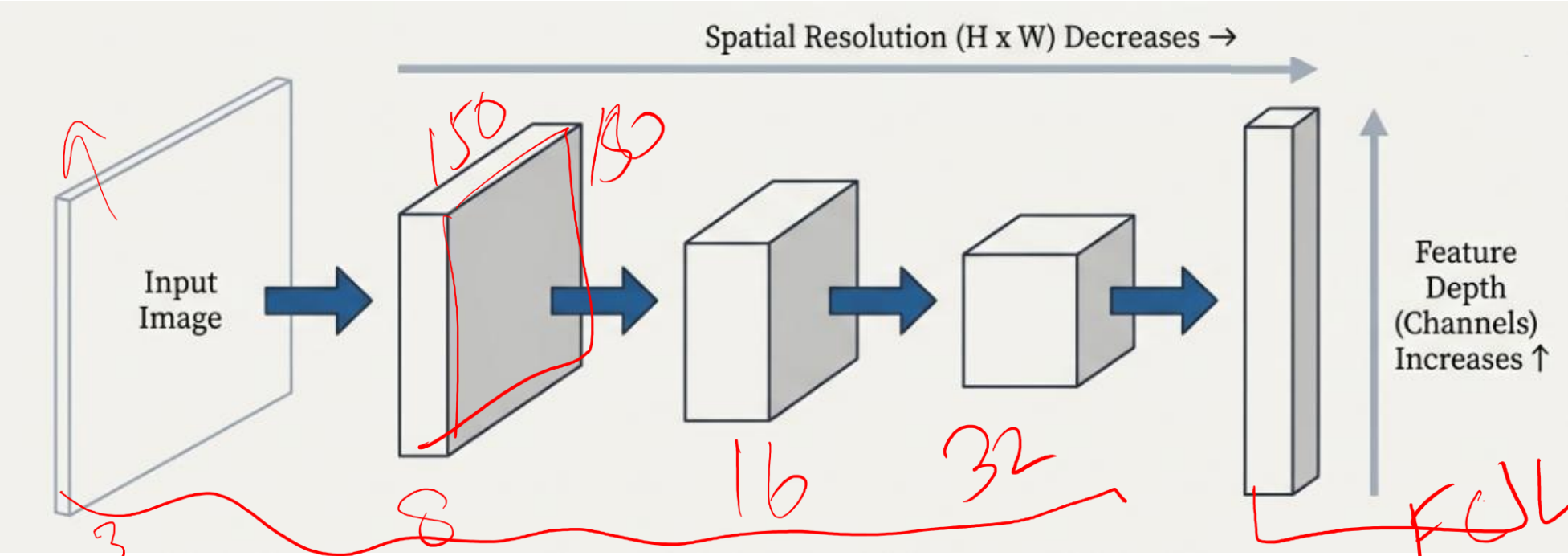


CNNs naturally learn a hierarchy of features, from pixels to patterns to parts to objects.

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

nn.Conv2d

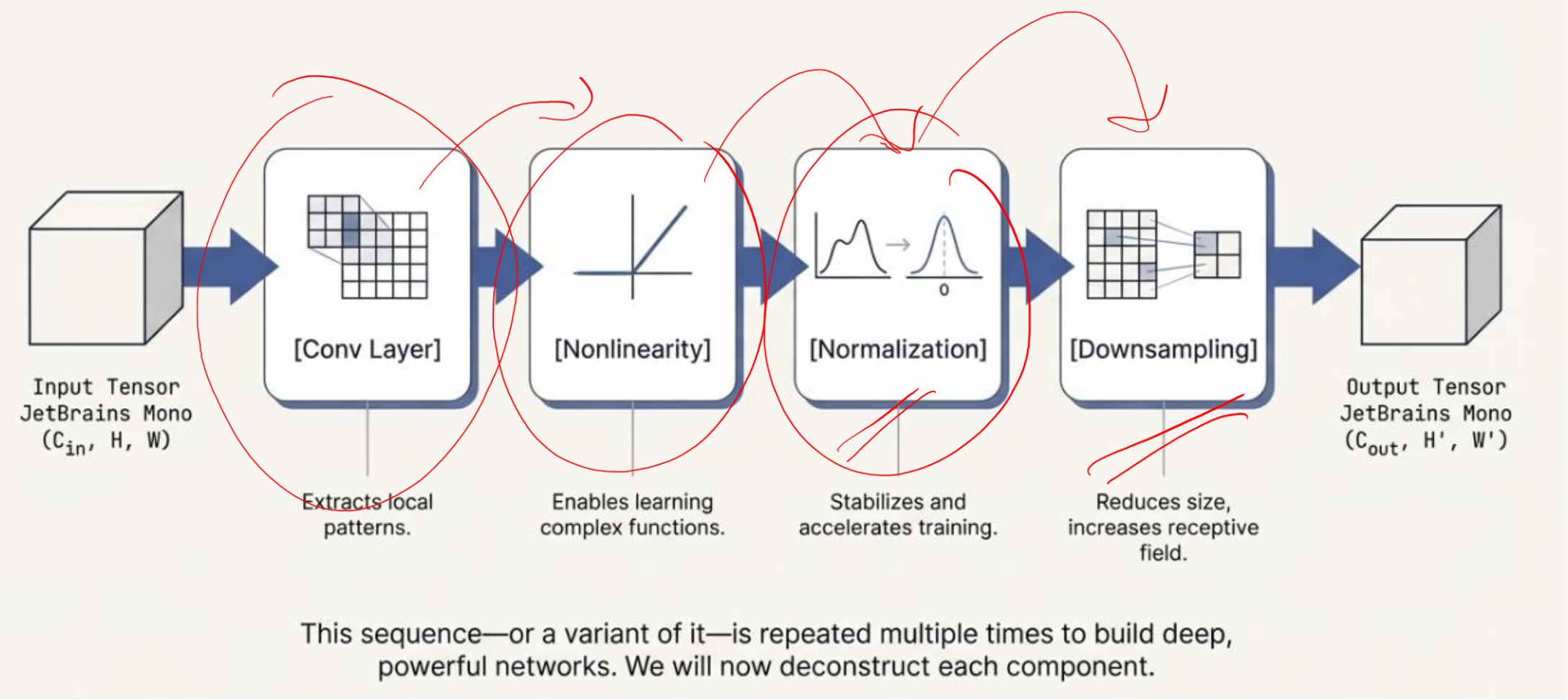
torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride*=*1*, *padding*=*0*)



https://tinyurl.com/dlframeworks

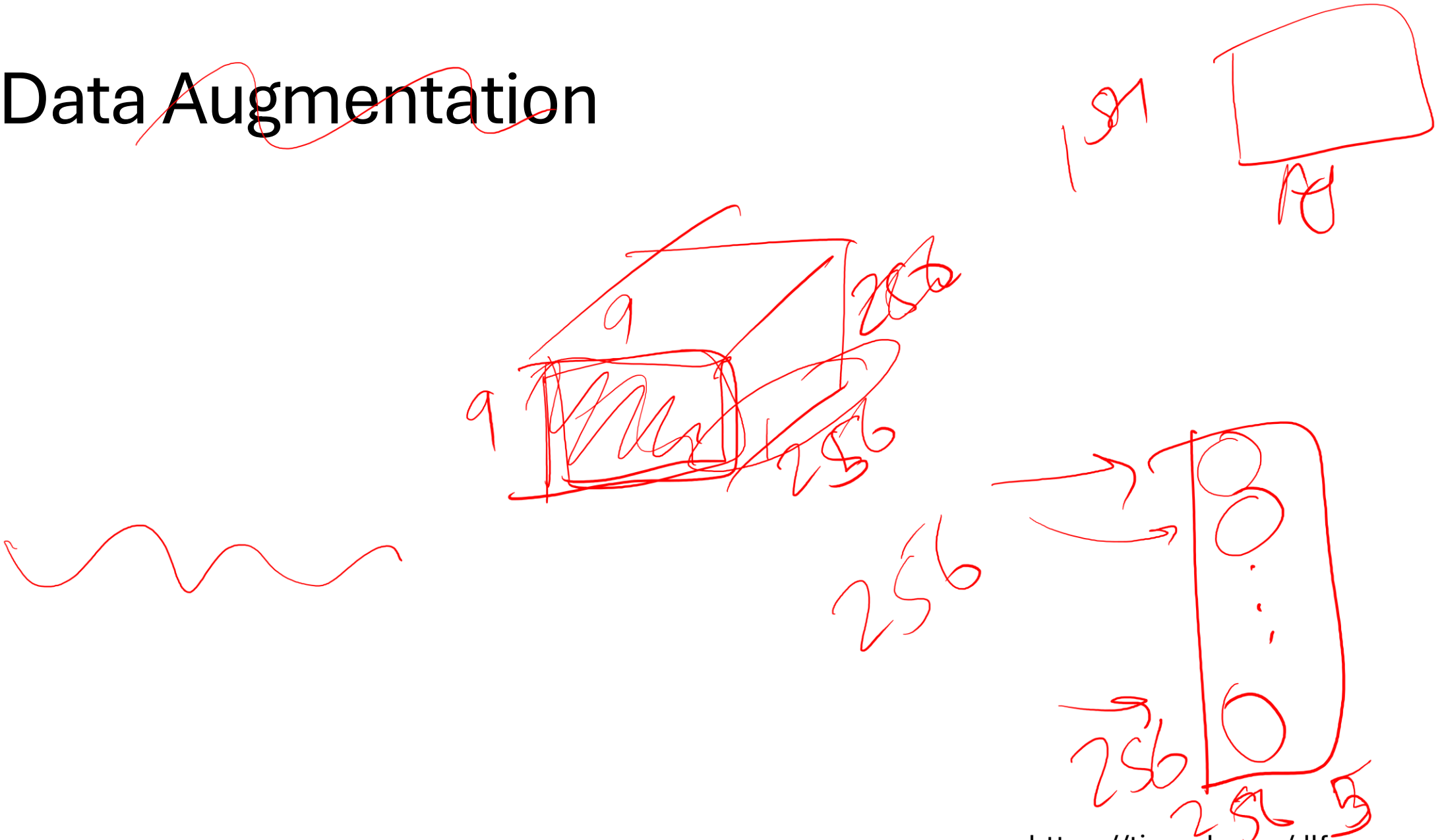
https://github.com/sakharamg/DeepLearningFrameworks

Anatomy of CNN Block



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

Data Augmentation



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

BatchNorm

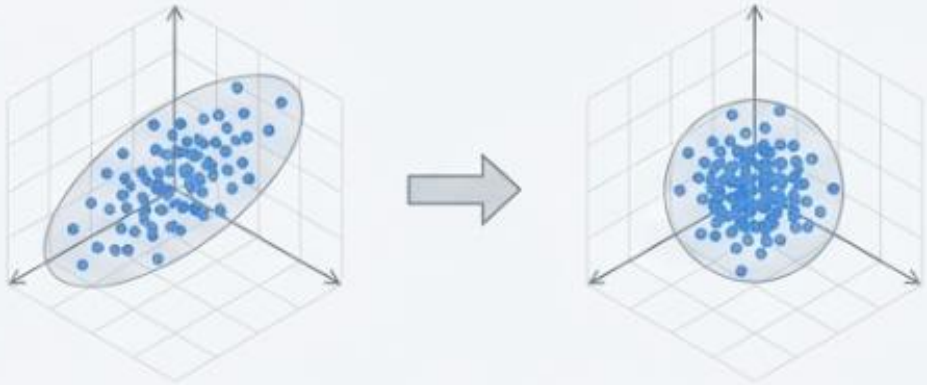
Why

As data passes through deep networks, the distribution of activations can shift dramatically (internal covariate shift), making training slow and unstable.

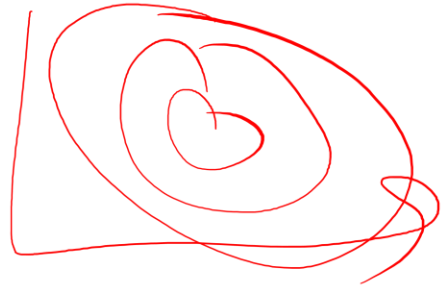
What

Batch Normalization standardizes the activations within a mini-batch to have zero mean and unit variance.

How



Note: Works best with larger batch sizes.
Group Normalization is a strong alternative for smaller batches.



$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

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

detach()

Removes a tensor from computation graph

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

Explaining the Predictions

Our models can achieve superhuman accuracy, but their reasoning is often opaque. When a CNN **outputs a score for a class, (s_c), how do we** understand the basis for that decision?

This deck explores three foundational techniques that answer two critical questions:

1. **Which pixels in the input image most influenced the score?**
2. **Where is the evidence for this class in the image?**

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

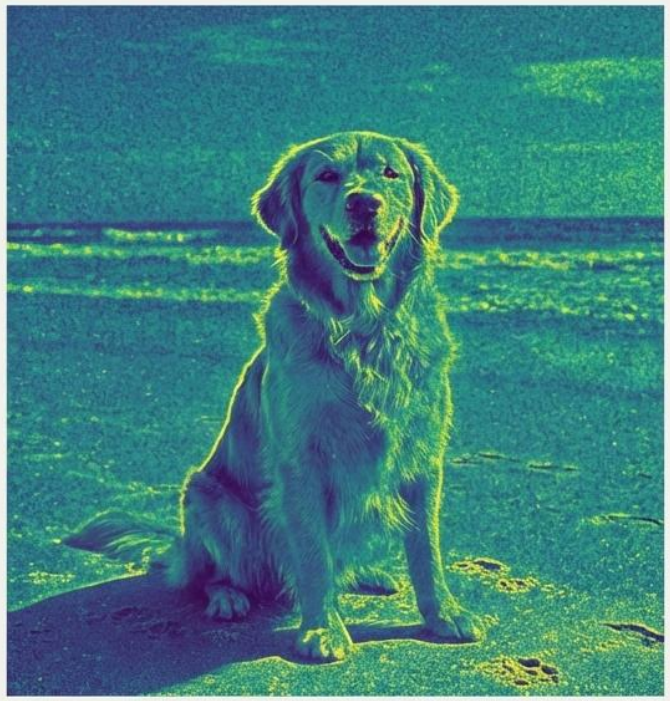
Saliency Maps: Which Pixels are Most Influential

The Concept

Core Idea: The most direct way to find important pixels is to measure how much the output score changes when a pixel's value changes. This is precisely what a gradient does.

Definition: A **Saliency Map** is the magnitude of the **gradient** of the class score (s_c) with respect to the input pixels ($\mathbf{x}$).

$$Saliency(x) = \left| \frac{\partial s_c}{\partial x} \right|$$



Caption: The map highlights pixels where a small change would have a large impact on the 'Golden Retriever' score.

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

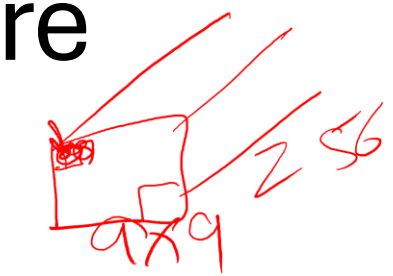
Challenge with Sensitivity

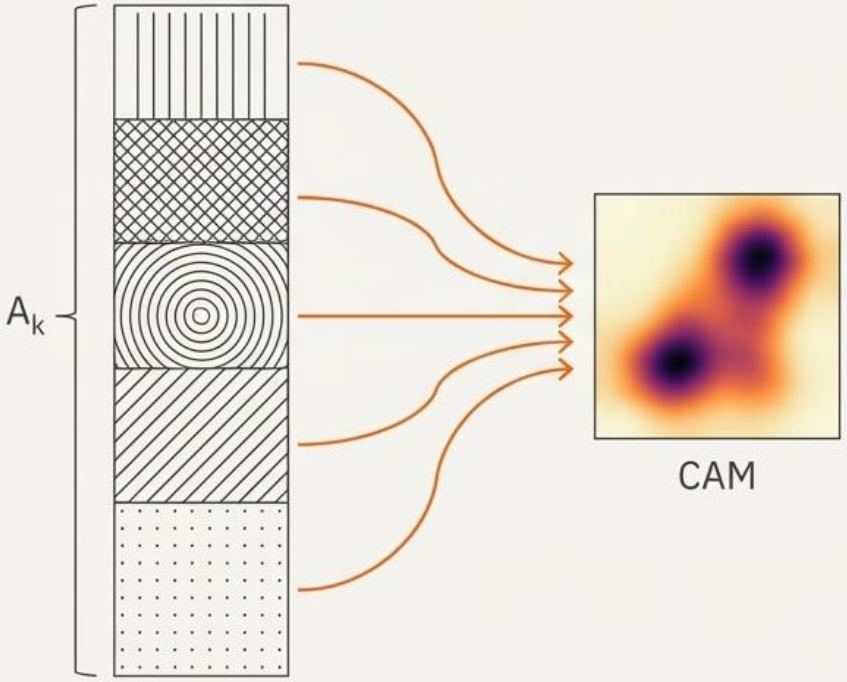
Main Point: Sensitivity is not the same as evidence.

- **Noise:** Raw gradients are often chaotic and look like static, making them hard to interpret cleanly.
- **Focus on Edges/Textures:** They can highlight contours and high-frequency patterns more than the semantic region of the object itself.
- **Gradient Saturation:** For very confident predictions, gradients can become small or vanish, hiding the true importance of key features.

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

Class Activation Maps: Which Pixels are Important during Prediction



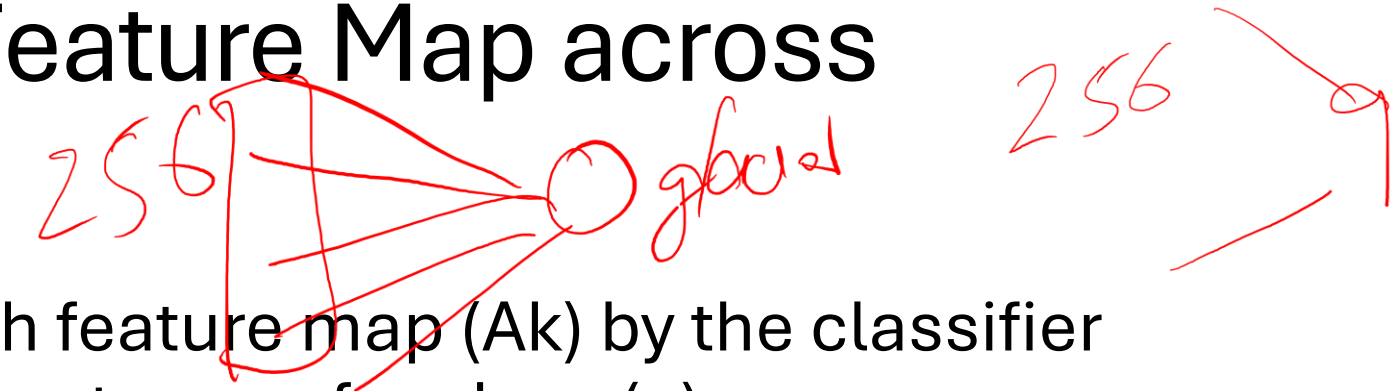


The Core Insight: Instead of looking at input pixels, let's look at the final convolutional feature maps. Each map (A_k) acts as a detector for a specific visual pattern. If we know how important each pattern is for a given class, we can combine them to see where the "evidence" for that class is located.

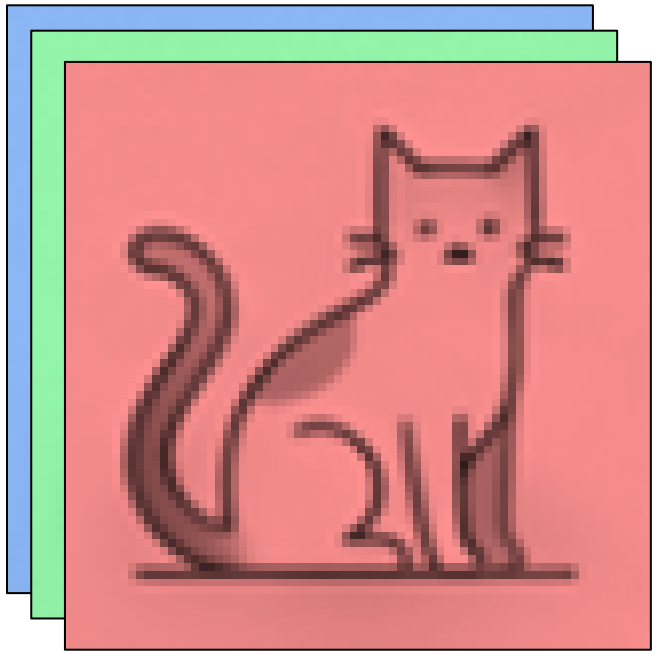
The Result

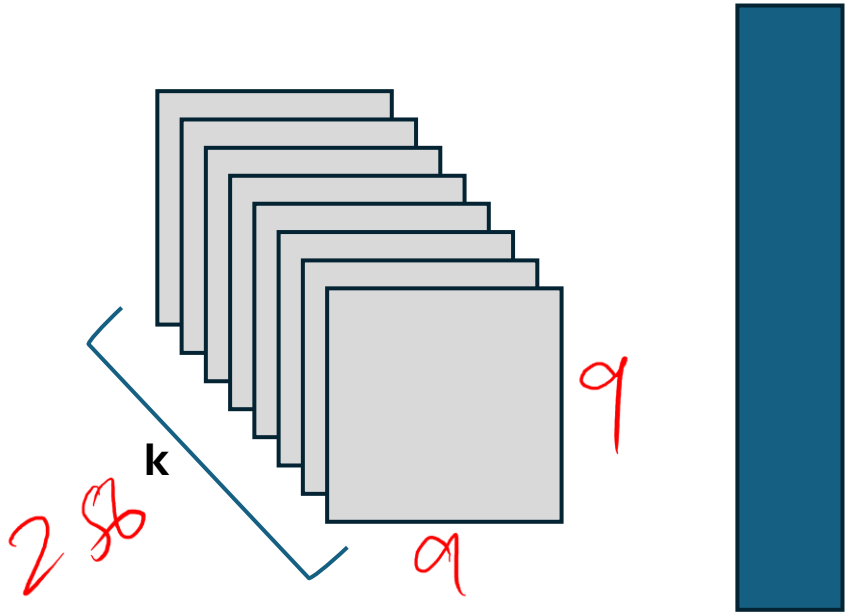
Caption: CAM produces a smooth, high-level map showing the regions the model used as evidence for 'Golden Retriever'.

Weighted Sum of Feature Map across Channels



CAM works by weighing each feature map (Ak) by the classifier weight (w_k^c) corresponding to map for class (c).





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

Thank You

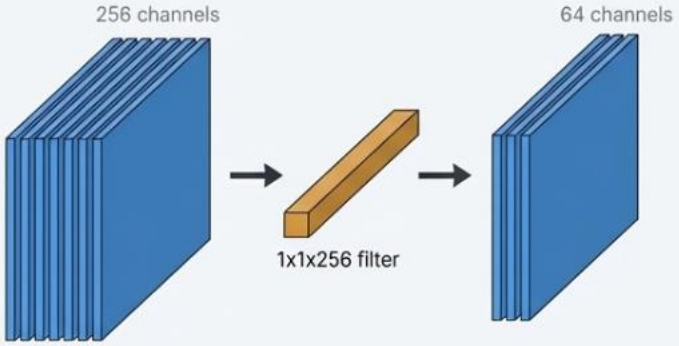
Appendix

Building Smarter and Efficient Blocks

Beyond depth, modern CNNs focus on computational efficiency and richer feature interactions.

1. 1x1 Convolutions ('Pointwise')

- **What it is:** A convolution with a 1x1 kernel size. It doesn't look at spatial neighbors; instead, it operates across the channels at each pixel location.
- **Why it's used:** A powerful tool for channel mixing. It can be used to shrink the number of channels (a 'bottleneck' to reduce computation) or expand them, all while being very computationally cheap.



2. Depthwise Separable Convolutions

- **What it is:** A standard convolution split into two steps:
  1. A *depthwise* convolution: applies a single spatial filter to each input channel independently.
  2. A *pointwise* (1x1) convolution: mixes the channels from the depthwise step.
- **Why it's used:** Massively reduces computation and parameters compared to a standard convolution, making it ideal for mobile and on-device applications (e.g., MobileNets).