AI Learning Series

Kaustubh Khatri

What to expect

This is not one lecture, but a series.

Starting directly with CNN

Then with **GenAI** using CNN

Later we will look at concepts of NN in dept

We will continue our Journey with GAN, LSTM, RNN and further

Starting with math light and heavier concepts will come as needed

The more we experiment the more learn

Through this series we will be focusing on **Why?**

Key Principles

Starting with **math light** and heavier concepts will come as needed

The more we experiment the more **learn**

Through this series we will be focusing on **Why?**

The Artist vs. The Critic

The Core Question:

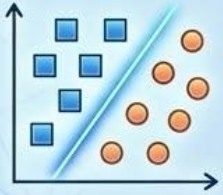
Discriminative AI asks:
"Which side of the line does this data point fall on?"

Generative AI asks:
"Where do the data points actually live, and can I make a new one?"

1. **Discriminative Models** (The Critic/Classifier)

Goal: To distinguish between classes.

The Math:
Models the conditional probability $P(Y|X)$.
Given the Data (X), what is the probability it belongs to Label (Y)?



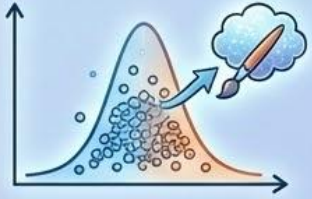
Mechanism:
It learns a Decision Boundary (a line or hyper-plane) that separates data. It doesn't care what a "dog" looks like fundamentally; it only cares what even:
what makes a "dog" different from a "cat."

Example: CNN takes an image and outputs "Class A" or "Class B."

2. **Generative Models** (The Artist/Creator)

Goal: To understand how the data is created.

The Math:
Models the joint probability $P(X,Y)$ or just the distribution $P(X)$.
Given the Label (Y), what is the probability of seeing this Data (X)?



Mechanism:
It learns the Distribution Shape of the data itself. It tries to map the entire space where "dogs" exist.

Example: A GAN or VAE. It takes a random seed and outputs a full image of a dog.

Why We Don’t Just Want the ‘Best’ Image

Discriminative AI asks: ‘Which side of the line does this data point fall on?’

Generative AI asks: ‘Where do the data points actually live, and can I make a new one?’

1. The Misconception: Maximization (The ‘Average’)

Goal: To find the mathematical ‘peak’ probability.

The Math: $\hat{x} = \text{argmax } P(x|y)$.

Mechanism: The model minimizes error to ALL data, resulting in the average of all possibilities.

Example: Averaging every dog photo creates a blurry, unrealistic blob. It's technically the 'most likely' but looks terrible.

Analogy: Rolling dice and always getting '7' (a terrible simulation).

2. The Reality: Sampling (Stochasticity/Randomness)

Goal: To pick a random point from the ‘high probability region’.

The Math: $x \sim P(x|y)$.

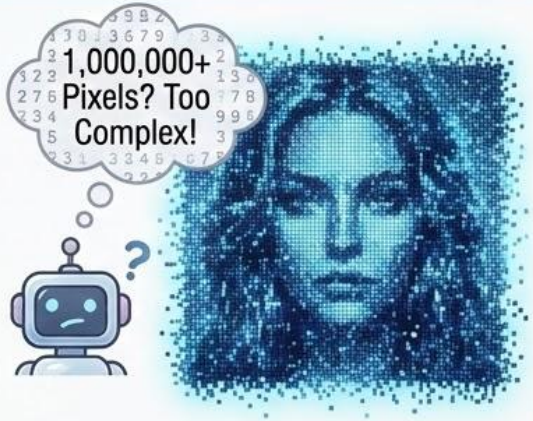
Mechanism: The model introduces **Stochasticity (Randomness)** to generate diverse, realistic examples.

Example: You might get a Golden Retriever, or a Poodle, depending on the random seed.

Analogy: Simulating the full distribution (mostly 6-8, but occasionally 2 or 12) for accuracy.

Latent Space & Manifolds: High-Dimensional Data to Low-Dimensional Features

1. The Problem: High-Dimensional Data (Pixels)



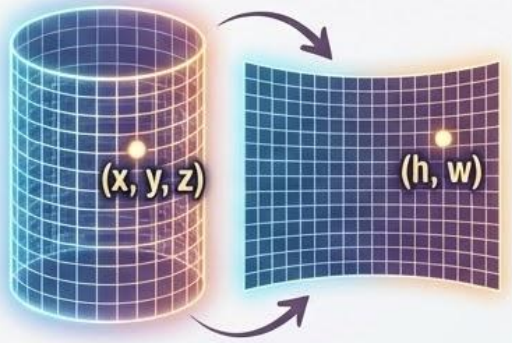
Raw data is high-dimensional and complex.

Learning pixel-by-pixel is inefficient and redundant.

2. The Solution: Latent Space (Features)

We map observations from High-Dimensional Space to Low-Dimensional Latent Space, learning meaningful features instead of individual pixels.

3. The Cylinder Analogy: Unrolling Complexity



A 3D point needs **(x, y, z)**. 'Unrolling' the cylinder to 2D requires only **(h, w)**.

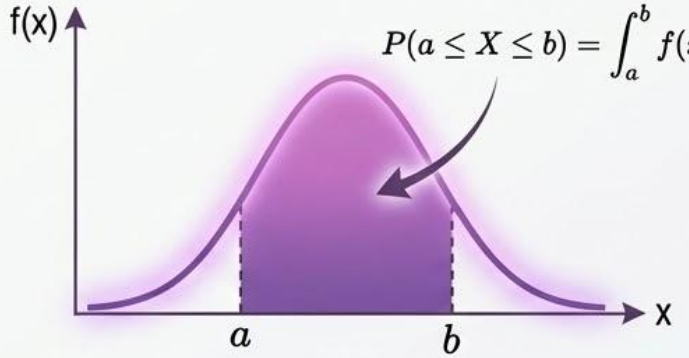
Deep learning 'unrolls' data to find these simple latent dimensions.

Mathematical Foundations: Probability Fundamentals

1. Sample Space (Ω)

- **Definition:** The set of all possible outcomes or data points we could possibly see.
- **Example:** In face generation, the sample space is "all possible arrangements of pixels." Most of this space is noise (static); a tiny fraction is actual faces.

2. Probability Density Function (PDF), $f(x)$

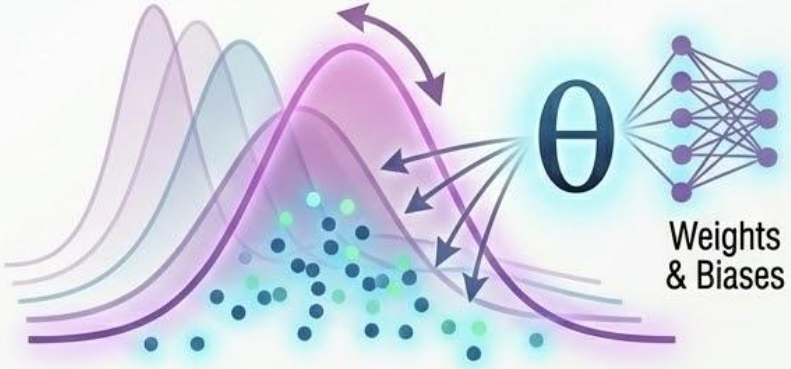


- **Definition:** A function that specifies the probability of the random variable falling within a particular range of values.
- **Concept:** It tells us how "dense" the data is at a specific point. High density = high probability of seeing this data.
- **Formula:** For a continuous variable x:

$$P(a \leq X \leq b) = \int_a^b f(x)\,dx$$

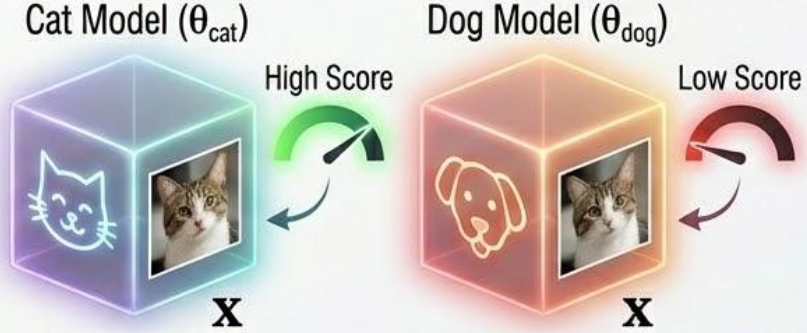
How Machines Learn – Parametric Modeling
Finding the Best Parameters

3. Parametric Modeling (Assuming a Family of Distributions)



- **Definition:** Instead of learning from scratch, we assume a family of distributions (like a Gaussian/Bell curve) defined by parameters θ.
- **In Deep Learning:** The "Parameters" θ are the weights and biases of the neural network.
- **Goal:** Change θ until the curve fits our data points.

4. Likelihood $\mathcal{L}(\theta \mid \mathbf{x})$ (How Likely is the Data Given the Parameters?)



- **Definition:** Given a fixed set of data points $\mathbf{x}$, how "likely" is it that they came from a specific set of parameters θ?
- **Intuition:** If a model generates cat pictures, a dog picture will have a very low likelihood score.
- **Formula:** For independent data samples $\mathbf{x}_1, x_2, \ldots, \mathbf{x}_n$:

$$L(\theta) = \prod_{i=1}^{n} P_\theta(x_i)$$

The Difference: Probability vs. Likelihood

Probability ($\mathcal{P}$(Data|Model))

Fair Coin
(Model)

10 Heads
(Data)

- **Definition:** I have a fair coin (Model). What is the chance I flip 10 heads (Data)?
- **Context:** We know the Truth (Model); we want to predict the Future (Data).
- **In Short:** Fixed Parameters → Variable Data.

Likelihood ($\mathcal{L}$(Model|Data))

10 Heads
(Data)

Fair Coin
(Model)?

- **Definition:** I just flipped 10 heads (Data). What is the chance the coin was fair (Model)?
- **Context:** We know the Past (Data); we want to discover the Truth (Model).
- **In Short:** Fixed Data → Variable Parameters.

A Concrete Mathematical Example

The Coin Toss Mystery

The Data & Question

10 Heads
(Data)

The Question: Is this a fair coin? (p=0.5?)

Scenario Comparison

Scenario A:
Assume Fair Coin
(p=0.5)

$$\mathcal{L}(p{=}0.5) = 0.5^{10}$$

$$\approx \mathbf{0.0009}$$

Verdict:
Extremely Unlikely.
Fits data poorly.

Scenario B:
Assume Trick Coin
(p=0.9)

$$\mathcal{L}(p{=}0.9) = 0.9^{10}$$

$$\approx \mathbf{0.348}$$

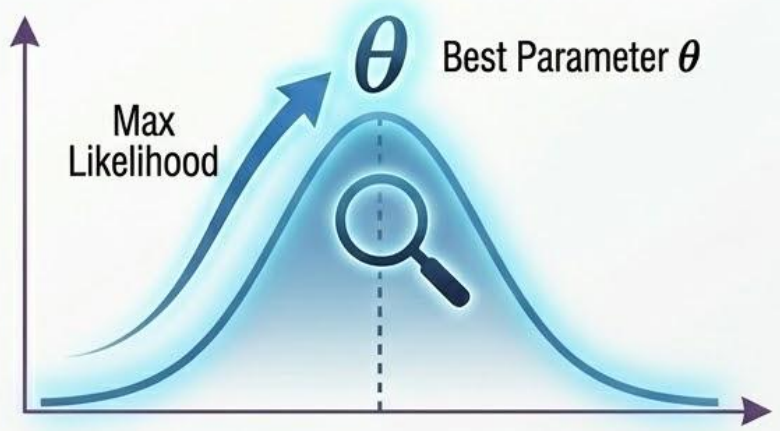
Verdict:
Much Higher Likelihood!
Explains data better.

Conclusion: The AI's Choice

AI chooses **Scenario B**
(Higher Score).
It learns the coin is likely rigged.

The Optimization Loop
Maximizing Success, Minimizing Error

5. Maximum Likelihood Estimation (MLE)



- **Definition:** The method of finding the specific θ that maximizes the Likelihood function. We want the setting that makes our observed data "most probable."
- **Formula:** $\hat{\theta}_{\text{MLE}} = \text{argmax}_\theta \mathcal{L}(\theta)$

6. Log-Likelihood & 7. Negative Log-Likelihood (NLL)

$P_1 \times P_2 \times \cdots < 1$ $\quad$ $\log(P_1) + \log(P_2) + \cdots$

Underflow Risk $\quad$ Stable Computation

- **Definition:** The natural logarithm of the likelihood.
- **Why use it?** Avoids underflow from multiplying small probabilities; turns multiplication into stable addition.
- **Formula:** $\log \mathcal{L}(\theta) = \sum_{i=1}^{n} \log P_\theta(x_i)$

Loss Function

Minimize NLL
= Maximize Likelihood

- **Definition:** The negative of the Log-Likelihood.
- **Why use it?** Deep Learning frameworks minimize loss. Maximizing Likelihood is mathematically identical to Minimizing NLL.

Formula (Loss Function): $J(\theta) = -\sum_{i=1}^{n} \log P_\theta(x_i)$

Summary: Minimize NLL → Maximize Likelihood → Generate better data.

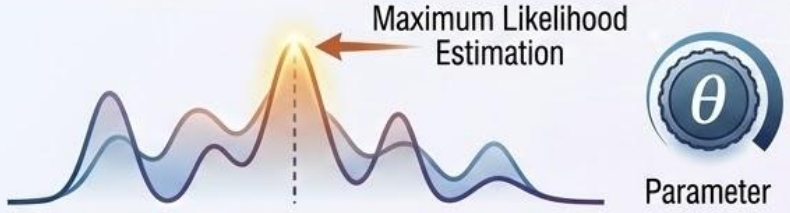
The Big Picture – Putting It All Together

From Theory to Reality

1. The Goal (Generative AI)

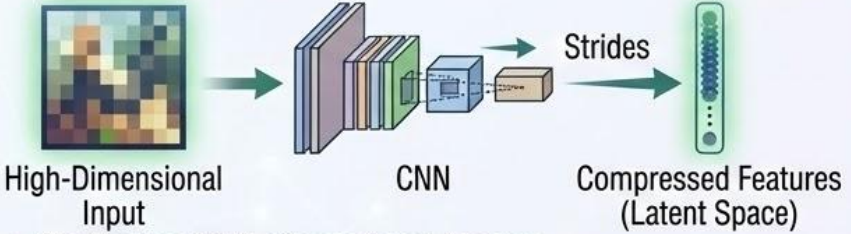
- Don't just classify, we create.
- Move from observing to modeling.

2. The Math (Likelihood & PDFs)



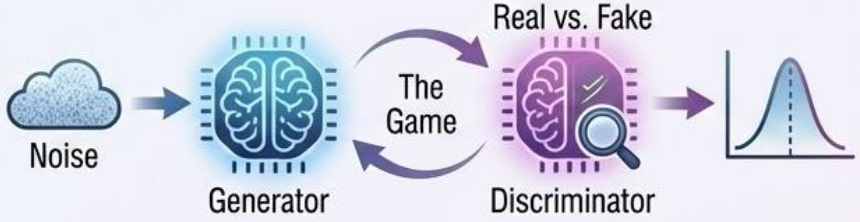
- **The World**: A complex PDF.
- **Strategy:** Tune parameters (θ) to maximize likelihood of training data.

3. The Tool (CNNs & Strides)

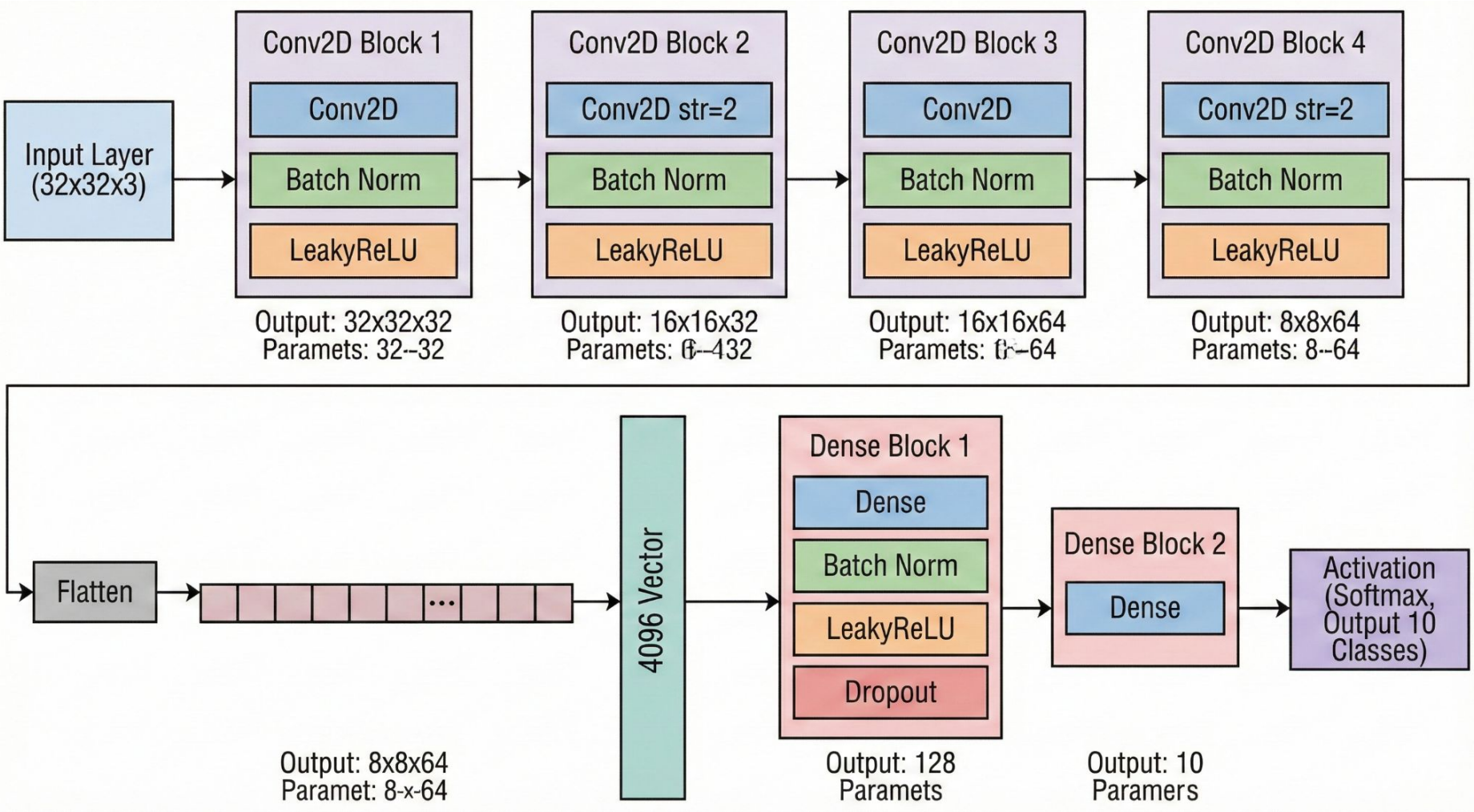


- **Challenge:** High-dimensional images.
- **Solution:** CNNs & Strides compress 'Pixels' into 'Features'.

4. The Architecture (GANs)



- **Method:** A game between Generator & Discriminator.
- **Outcome:** Generator implicitly learns true data distribution.


Input Layer
(32x32x3)
Conv2D Block 1
Conv2D
Batch Norm
LeakyReLU
Output: 32x32x32
Paramets: 32--32
Conv2D Block 2
Conv2D str=2
Batch Norm
LeakyReLU
Output: 16x16x32
Paramets: 6--432
Conv2D Block 3
Conv2D
Batch Norm
LeakyReLU
Output: 16x16x64
Paramets: 8--64
Conv2D Block 4
Conv2D str=2
Batch Norm
LeakyReLU
Output: 8x8x64
Paramets: 8--64
Flatten
Output: 8x8x64
Paramet: 8-x-64
4096 Vector
Dense Block 1
Dense
Batch Norm
LeakyReLU
Dropout
Output: 128
Paramets
Dense Block 2
Dense
Output: 10
Paramers
Activation
(Softmax,
Output 10
Classes)

Block One – Initial Feature Extraction

The Input Processor

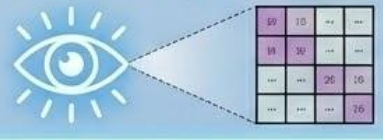
The Goal & Shapes

- **The Goal:** Convert raw RGB pixels into meaningful mathematical features.
- **Input Shape:** (32, 32, 3) → A color image.

- **Output Shape:** (32, 32, 32)
  → A stack of 32 feature maps.

The Components

1. Conv2D: The "Eye" (Finds patterns)



2. BatchNormalization: The "Stabilizer"
(Adjusts the range)

3. LeakyReLU: The "Activator"
(Decides what is important)

Deep Dive – Conv2D Mechanics (The Filter)

How the Convolution Works

1. The Kernel (Filter) & The Code

```
layers.Conv2D(filters=32, kernel_size=3, ...)
```

We create a small 3x3 matrix of weights. This matrix slides over the image like a magnifying glass.

Dot Product

(P1*W1 + P2*W2 + …) → 1

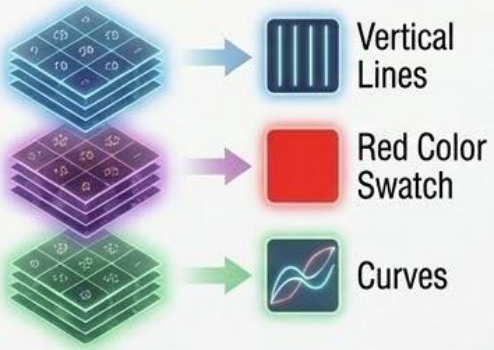
One Single

Operation: It performs a Dot Product, multiplying image pixels by filter weights and summing them up to get one single number.

2. The Filters (32) & Transformation

We don't just use one filter; we use **32 independent filters.**

Filter 1 might learn to see vertical lines.
Filter 2 might learn to see the color red.
Filter 3 might learn to see curves.



3 Channels (RGB)

32 Channels (Features)

Controlling the Geometry

Stride and Padding – Managing Spatial Dimensions

```
layers.Conv2D(.., strides=1, padding="same")
```

1. Stride = 1 (The Step Size)

Definition: How many pixels the filter moves after each calculation.

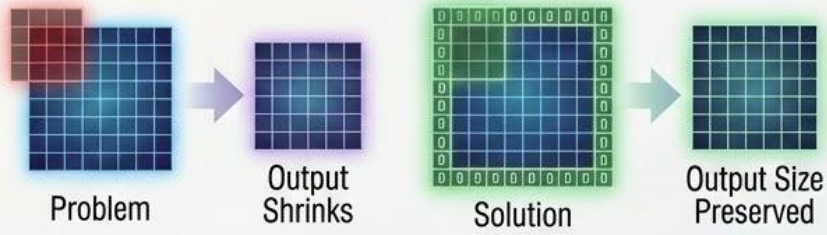
Logic: We move 1 pixel at a time.

Effect: We examine every single possible location. High detail, no downsampling.

2. Padding = "same" (The Border)

Problem: If a 3x3 filter is at the corner of a 32x32 image, it hangs off. Output shrinks to 30x30.

Solution ("Same"): The code automatically adds a border of zeros.



Math: Input Width (32) = Output Width (32)

Why? Preserves spatial dimensions for stacking many layers.

The "Why" – Who Chooses the Numbers?

Hand-Crafted vs. Learned Filters

Question: Do we choose the numbers [-1, 0, 1]?
Answer: In the past, Yes. In Deep Learning, No.

1. The Old Way (Computer Vision ~1990s)

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Blur Filter
(Averages pixels)

0	-1	0
-1	5	-1
0	-1	0

Sharpen Filter
(Highlights differences)

Limitation: We had to guess what features were important.

2. The Deep Learning Way (Your Code)

Initialization
(Random Noise)

0.01	-0.4	0.2
-0.2	0.1	-0.4
0.01	-0.4	0.2

Backpropagation:
Adjusts numbers to fix error

Training
(Backpropagation)

Predict "Dog" (Fail)

Dog Ear

-1	0	1
-1	0	1
-1	0	1

The Magic (Automatic Discovery)
Network automatically discovers useful filters over thousands of steps.

Summary of Filter Evolution

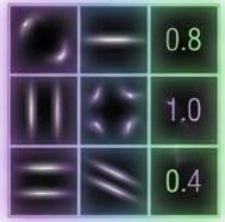
From Randomness to Understanding

1. Start (Epoch 0): Randomness

Filter Values
Random

What it "Sees":
Nothing
(Static noise).

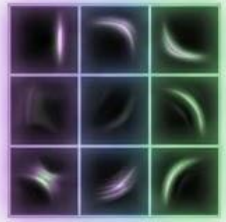
2. Middle (Epoch 10): Rough Patterns



Filter Values
Rough
Patterns

What it "Sees":
Simple edges
and blobs.

3. End (Epoch 100): Optimized Structures



Filter Values
Optimized
Structures

What it "Sees":
Specific textures
(Fur, Eyes, Curves).

Key Takeaway: We define the **Size** (3×3) and the **Quantity** (32 filters), but the **Data** defines the actual **Numbers** inside.

The Math – A Concrete Example

Calculating the Convolution

1. The Setup: Image Patch (X) & Filter (K)

Image Patch (X)

0	0	0
0	0	0
10	10	10

Filter (K)

-1	0	1
-1	0	1
-1	0	1

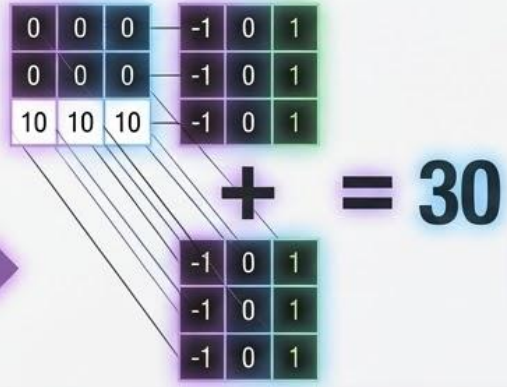
A tiny part of a photo (Dark Wall / Bright Sky)

(0 = Dark/Black, 10 = Bright/White)

A "Vertical Edge Detector"

(Negative on left, Positive on right)

2. The Operation: Dot Product & Sum



Result = (0·-1)+(0·0)+(10·1)+(0·-1)
+(0·0)+(10·1)+(0·-1)+(0·0)+(10·1)
= 10+10+10 = 30

3. The Interpretation: What It Means

The result is 30 (a high number).

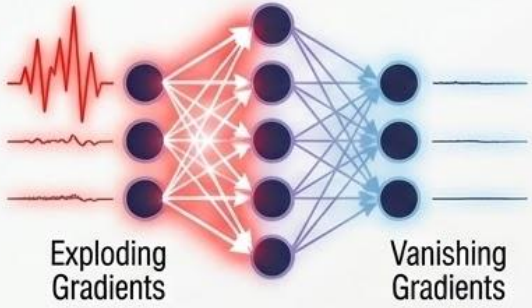
This tells the network:
"Hey! There is a strong vertical edge right here!"

If the image was all black (all 0s), the result would be 0 (No feature found).

Deep Dive – Batch Normalization

Stabilizing the Learning

1. The Problem: Instability & Covariate Shift



Exploding/Vanishing Gradients

Feature A (Large Scale, 0-1000)

Network Gets Confused (Internal Covariate Shift)

2. The Solution: Batch Normalization (BN) & Formula

Batch of 32 Images

Batch Norm

Σ

Mean (μ)

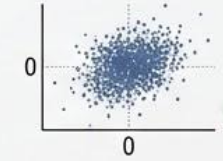
σ^2

Variance (σ^2)

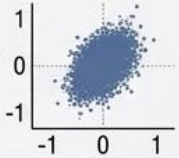
$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

Learnable Parameters

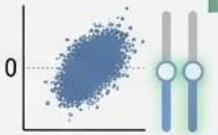
3. The Mechanism (Steps) & Benefits



1. **Subtract Mean** (Center at 0)



2. **Divide by Variance** (Scale Range)



3. **Scale by γ & Shift by β** (Learnable)

γ (Scale) β (Shift)

Why it helps:

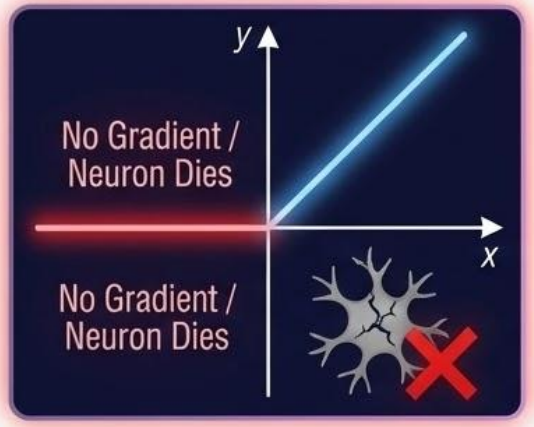
Faster Learning

Prevents Internal Covariate Shift

Deep Dive – LeakyReLU

The Activation Function

1. The Standard ReLU Problem ("Dying ReLU")

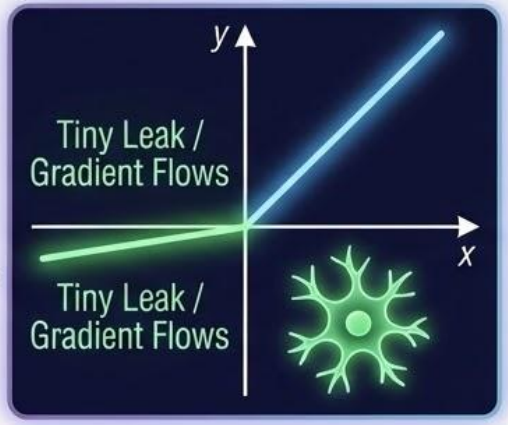


Normal ReLU: f(x)=max(0,x)

Negative Input → Output is 0

Gradient is 0 → Learning Stops (Neuron "dies")

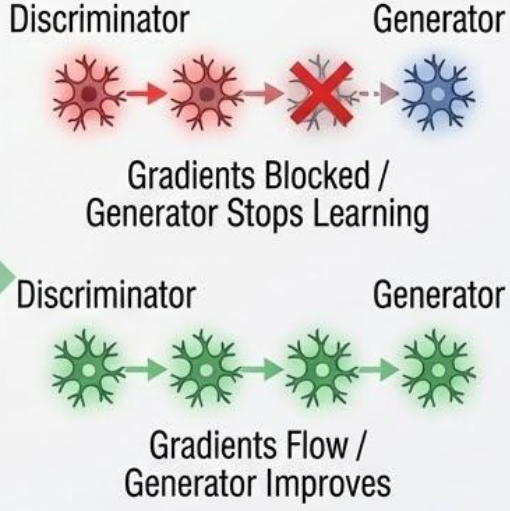
2. The Solution: Leaky ReLU & Formula



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

α (e.g., 0.3) — Small "Leak" Parameter

3. Why it Matters for Generative AI (GANs)



In GANs, gradients must flow backwards. Leaky ReLU ensures information always flows.

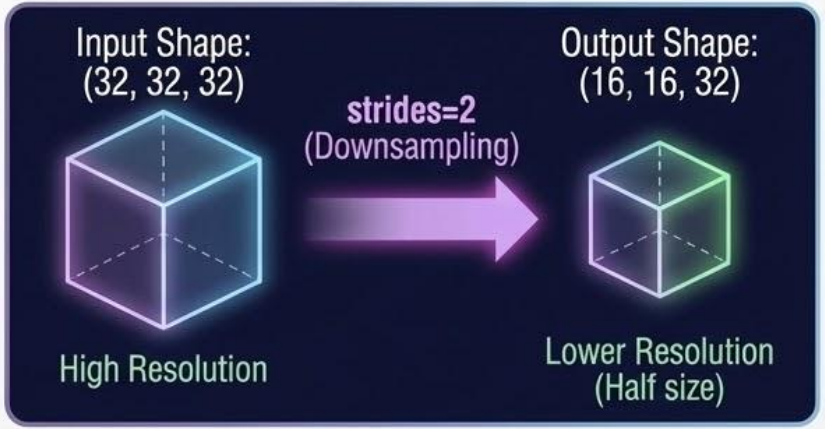
Block Two – The Downsampler

Reducing Resolution, Increasing Focus

The Code & Transformation

```python
x = layers.Conv2D(filters=32, kernel_size=3,
                  strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU()(x)
```

Visual Transformation



The Goal & Key Actor

The Main Actor: strides=2

It acts as the compression engine, skipping pixels to reduce dimensions.

The Goal: Compress & Focus

We no longer care about the exact pixel location of an edge; we just want to know if the edge exists in that general area.

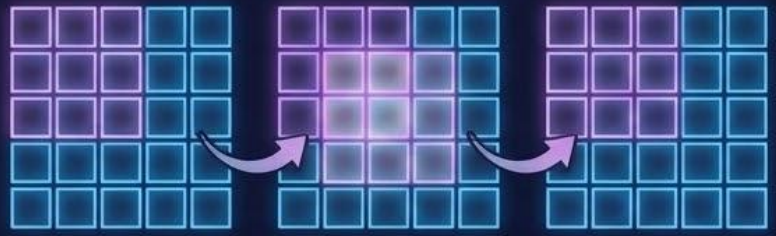
Key Takeaway: Downsampling compresses the image, trading spatial resolution for feature abstraction and computational efficiency.

Deep Dive – Stride 2 Mechanics

Skipping Steps to Compress Data

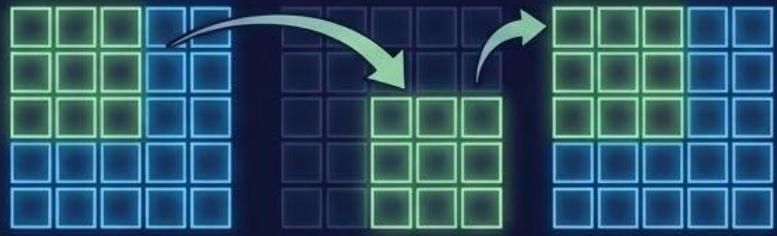
Comparison: Stride 1 vs. Stride 2

Stride 1 (Previous Block): **High Overlap**



Center at pixel 1, then 2, then 3. High overlap.

Stride 2 (This Block): **Skipping Steps**



Center at pixel 1, SKIP pixel 2, Center at pixel 3. Lower overlap.

The Math: Calculation Efficiency

$$\text{Output Size} = \frac{\text{Input Size}}{\text{Stride}} = \frac{32}{2} = 16$$

Because we skip every other pixel, we perform the calculation half as many times horizontally and vertically.

☑ Key Takeaway: Why not just resize the image?

Traditional Resizing

Loses data & detail.

Convolutional Resizing

Learns to keep important features.

The Convolution is learning how to resize. It learns which pixels to keep and which to ignore to best preserve important features.

The Concept – "Information Density"

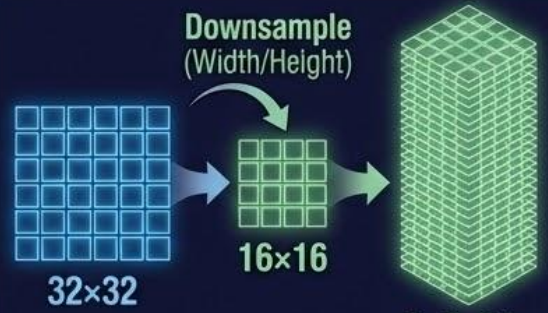
Trading Space for Meaning

The Problem: Large & Shallow

⚠ Too Much Detail

At the start, the image is **large**, but the information is **shallow** (just colors). Too much raw detail.

The Solution: Shrink & Deepen



As we go deeper, we shrink the Width/Height, but increase the **Depth** (Channels). Trading space for meaning.

The Analogy: From Bricks to Concepts

Input: Every Single Brick (Pixels)

Block 2: Streets (Downsampled)

Block 4: Neighborhoods (Concepts)

☑ By the end, the network doesn't see "pixels," it sees "concepts."

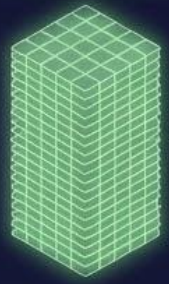
Result: The network trades spatial resolution for semantic understanding, effectively compressing data while increasing its meaningful content.

The Bridge – Flattening

Preparing for the Decision

The Code & The Situation

```
x = layers.Flatten()(x)
```

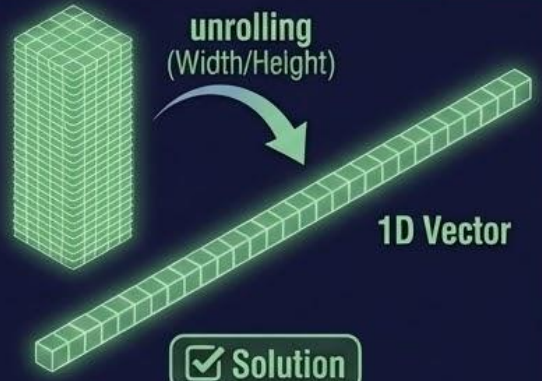


Input: 8×8×64
(3D Volume)

← Represents 64 specific feature maps (textures, shapes), each on an 8×8 grid.

⚠ **Problem:** Standard "decision making" neurons (Dense layers) expect a simple list, not a 3D block.

The Solution (Flatten)



☑ **Solution**

We take the 3D block and "unroll" it into a single long line of numbers.
Math: 8×8×64=4,096 values.

The Analogy: From Stack to Strip

It's like taking a stack of 64 photos, cutting them into strips, and laying them all out in one long row so the computer can look at everything at once.

Key Takeaway: Flattening is the essential bridge that transforms complex 3D spatial data into a 1D format suitable for the final decision-making layers of the network.

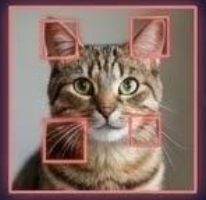
The Brain – The Dense Layer

Finding Global Relationships

The Code & The Shift

```
x = layers.Dense(128)(x)
```

CNN (Local Patterns)



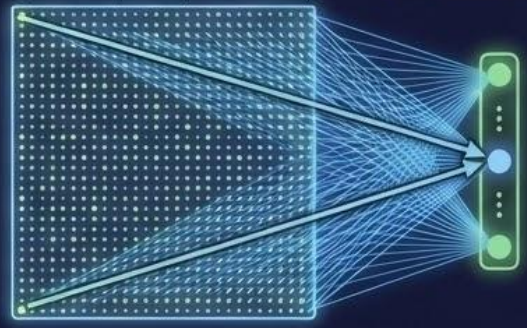
CNNs look at local patterns (eyes, ears, whiskers)

Dense Layer (Global Combinations)

Dense Layers look at global combinations (Does "Whiskers" + "Pointy Ears" + "Small Size" = Cat?)

How it works (Dense Connectivity)

4,096 Inputs (Flattened) → **128** Neurons



Each of the 128 neurons in this layer connects to ALL 4,096 inputs. This allows the network to combine information from the top-left corner of the image with information from the bottom-right corner.

Why 128? (The Bottleneck)

4,096 Raw Features

128 High-Level Concepts

This is a "bottleneck." We force the network to summarize those 4,096 raw features into just 128 high-level concepts.

Key Takeaway: The Dense layer is the brain's core, making sense of the flattened data by finding global relationships and summarizing complex inputs into a concise set of high-level concepts.

The Verdict – Softmax

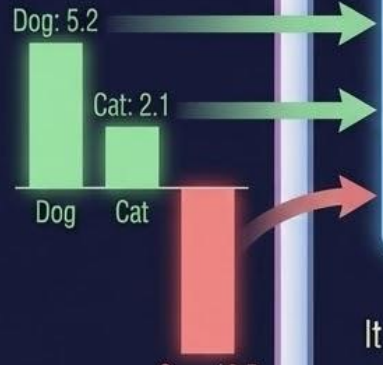
Converting Scores to Probabilities

The Code & Raw Scores (Logits)

```
x = layers.Dense(NUM_CLASSES)(x) (e.g., 10 classes)
output_layer = layers.Activation("softmax")(x)
```

Raw Output (Logits)

The Dense layer outputs raw scores, which can be negative or huge numbers.



The Transformation (Softmax)

Softmax Function

$$S(y_i) = \frac{e^{y_i}}{\sum e^{y_j}}$$

It squashes these numbers so they all sum up to 1 (100%).

Why Softmax? (The Amplifier)

It amplifies the winner. If one score is slightly higher than the others, Softmax pushes it close to 1 and pushes the others close to 0, giving a clear, confident prediction.

Key Takeaway: Softmax is the final judge, converting raw, uninterpretable scores into clear, confident probabilities that sum to 100%, highlighting the most likely prediction.