본 수업자료는 2025년도 과학기술 정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.

ComputerVision

Week3

2025-2

Mobile Systems Engineering

Dankook University

Recap — What We Learned So Far

- **Key Topics Reviewed**
  - **What is a Convolutional Neural Network (CNN)?**
    - A type of deep neural network designed to process data with grid-like topology (e.g., images). It automatically learns spatial hierarchies of features.
  - **Basic Structure of LeNet-5**
    - LeNet-5 was one of the earliest CNNs designed for digit recognition (MNIST). It consists of alternating **convolution**, **pooling**, and **fully connected layers**.
  - **Feature Extraction in CNNs**
    - Through convolution and pooling, CNNs transform raw pixel values into **high-level features**, such as edges, textures, or object parts.

Recap — What We Learned So Far

- **Key Topics Reviewed**
  - **Typical CNN Layer Structure (from LeNet-5)**

Layer	Type	Output Shape	Key Details
C1	Convolution	6@28×28	5×5 filters, stride=1
S2	Avg Pooling	6@14×14	2×2 pooling, tanh activation
C3	Convolution	16@10×10	Not fully connected – grouped connections
S4	Avg Pooling	16@5×5	2×2 pooling, learnable weights
C5	Convolution	120@1×1	5×5×16 filters (full connection)
F6	Fully Connected	84	Each neuron gets all 120 inputs
Output	RBF Layer	10	Radial Basis Function units

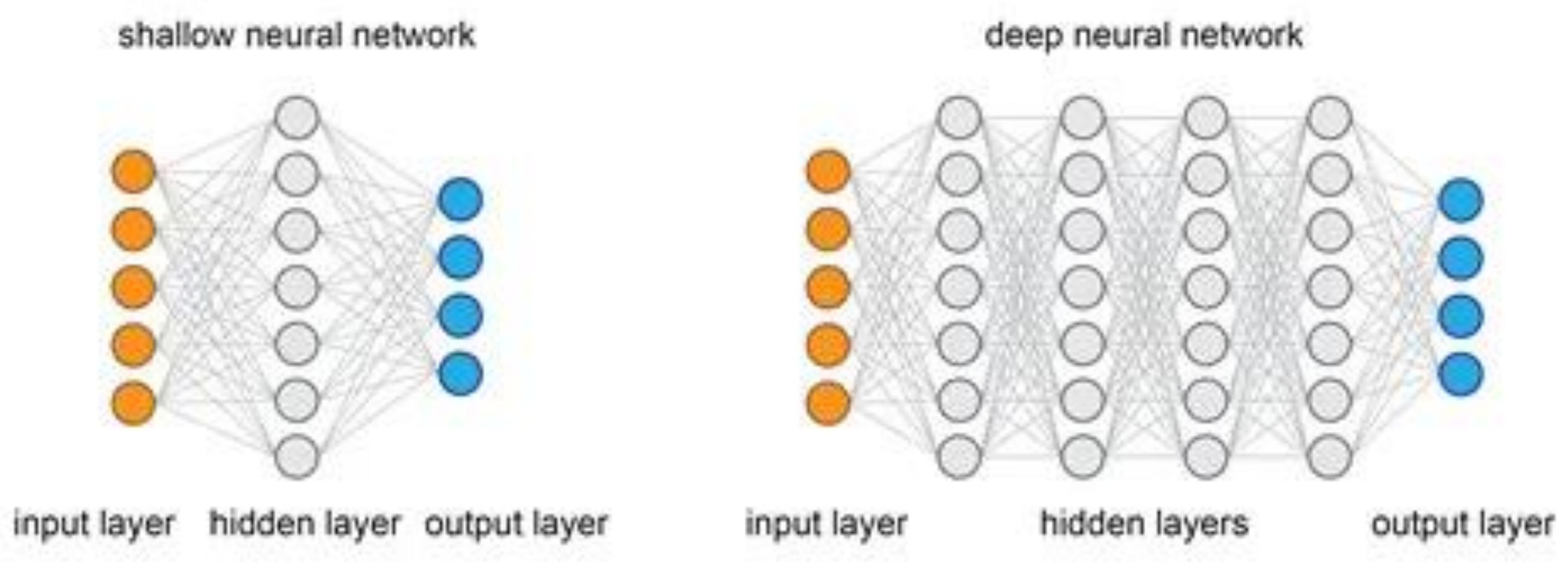
  - **Convolution Layer**: Detects local patterns using filters (kernels)
  - **Pooling Layer**: Reduces spatial dimensions and helps generalization
  - **Fully Connected Layer**: Performs classification based on extracted features

Motivation for Going Deeper

- **Why Do We Need Deeper Networks?**
  - **Shallow Networks**
    - Work well for **simple tasks** like digit recognition (e.g., MNIST)
    - Can only capture **low-level features** like edges or simple textures
    - Quickly **saturate** in performance as task complexity increases
  - **Complex Datasets Need Abstraction**
    - Real-world images (e.g., ImageNet) include complex variations:
      - ✓Background clutter
      - ✓Different lighting, poses, textures
    - Models need to learn **hierarchies of features**:
      - ✓Low-level: edges, corners
      - ✓Mid-level: textures, patterns
      - ✓High-level: object parts, semantic meaning

Motivation for Going Deeper

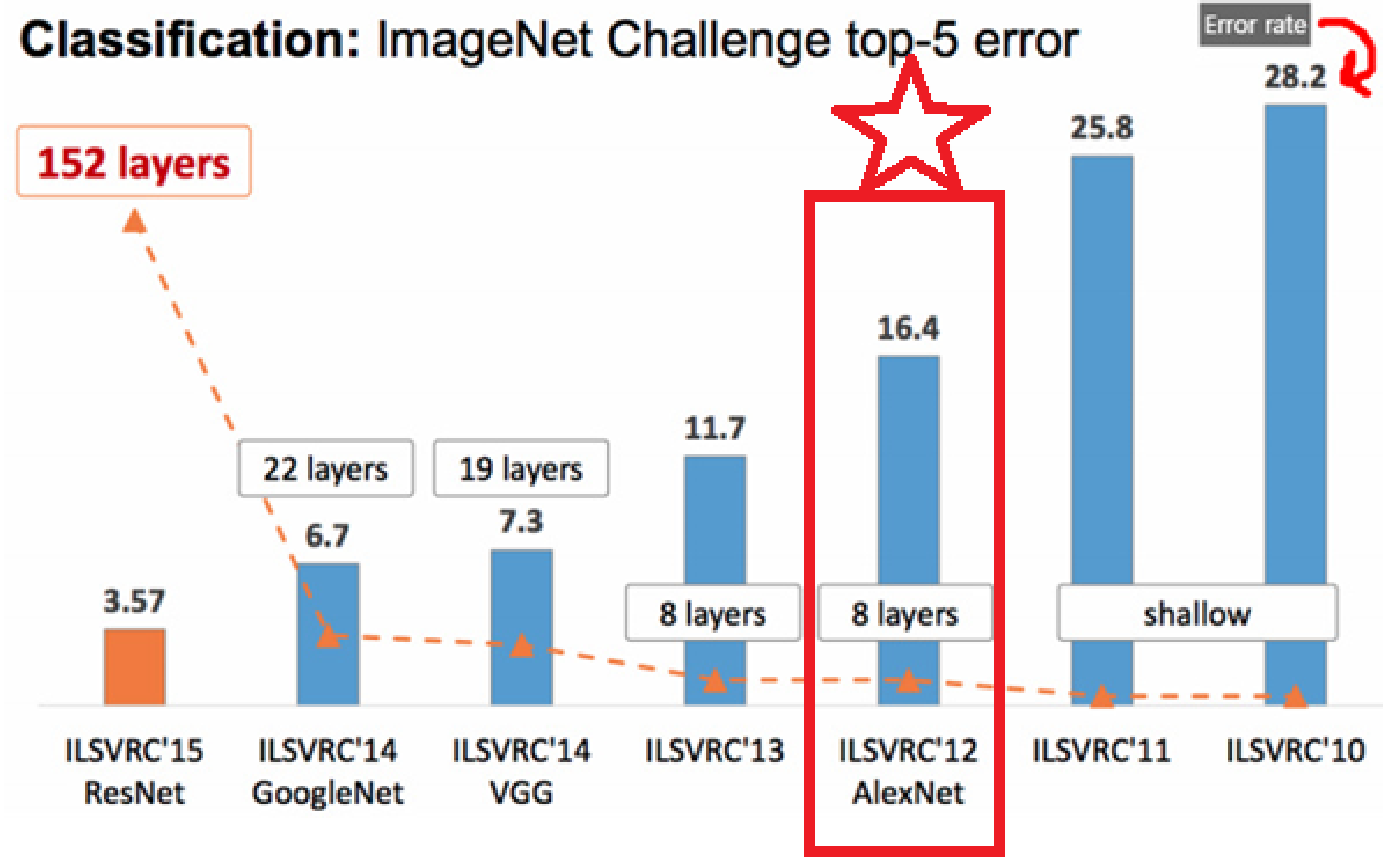
- **Why Do We Need Deeper Networks?**
  - **Deeper Networks Can Build Feature Hierarchies**



  - Each additional layer can capture more abstract concepts
  - Final layers make decisions based on **high-level understanding**
  - This leads to better generalization and classification accuracy

Introduction to AlexNet

- **The Turning Point — ImageNet 2012**



- **AlexNet** won the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** in 2012.
- Reduced top-5 error from ~26% to **16.4%** — a groundbreaking result.
- *“Marked the start of the deep learning era in computer vision.”*

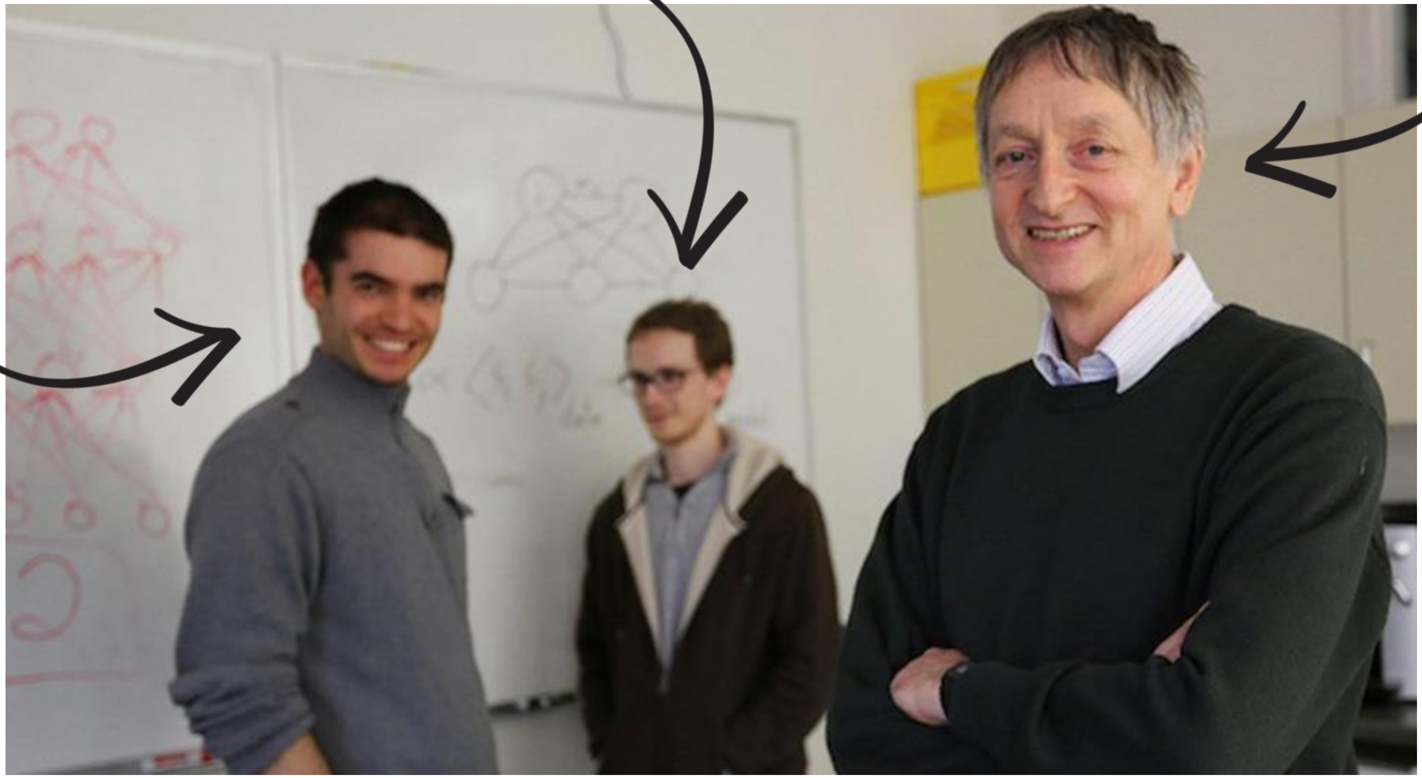
Introduction to AlexNet

- **Who Built AlexNet?**

Alex Krizhevsky

Geoffrey Hinton

Ilya Sutskever



- Developed by **Alex Krizhevsky**, a student of **Geoffrey Hinton**, with support from **Ilya Sutskever**.
- Hinton motivated Alex with an unusual bet: **"If you improve the accuracy by 1%, I'll let you skip the qualifying exam."**
- AlexNet exceeded expectations and made history.

Introduction to AlexNet

- **What is ImageNet?**

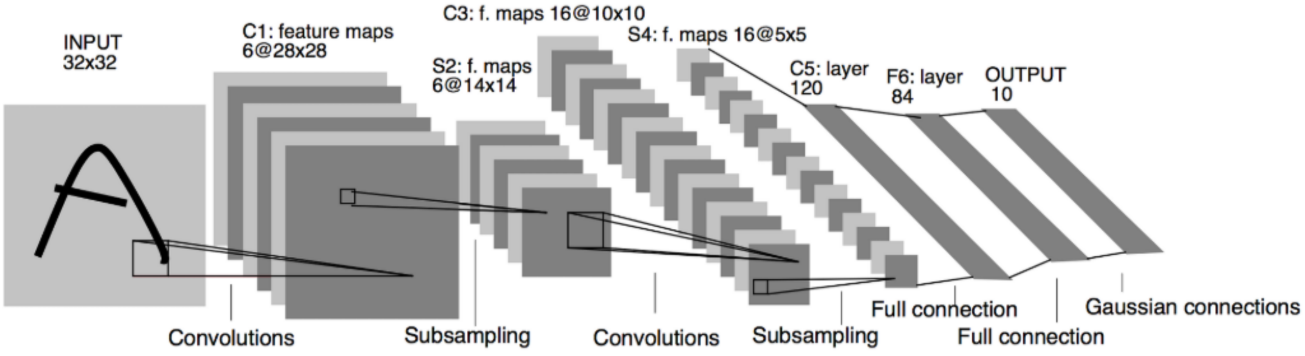
- Created by Fei-Fei Li in 2006.
- Contains **14 million+ labeled images**, classified into **20,000+ categories**.
- Enabled large-scale training of deep models.
- Used for the annual ILSVRC competition (2010–2017).

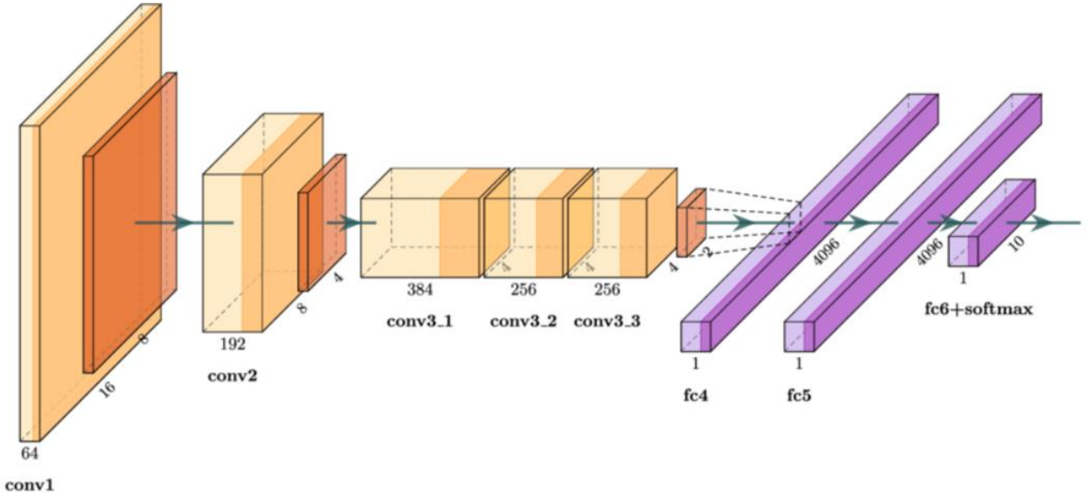
Introduction to AlexNet

▪ What Made AlexNet Special?

- Architecture: 5 convolutional layers + 3 fully connected layers
- ReLU instead of sigmoid or tanh → **faster training**
- Dropout → **reduced overfitting**
- Trained on **2 NVIDIA GTX 580 GPUs (3GB)** for scalability

▪ Architectural Diagram of AlexNet





- Add visual comparison of LeNet and AlexNet (already attached by user)
- Highlight how AlexNet expands LeNet's ideas with deeper layers and more filters

Introduction to AlexNet

- **Technical Innovations**
  - **ReLU** non-linearity made training faster
  - **Split training over two GPUs** to fit in memory (model parallelism)
  - **Data augmentation**: random crops, flips to prevent overfitting
  - **Local Response Normalization (LRN)** to encourage competition between neurons

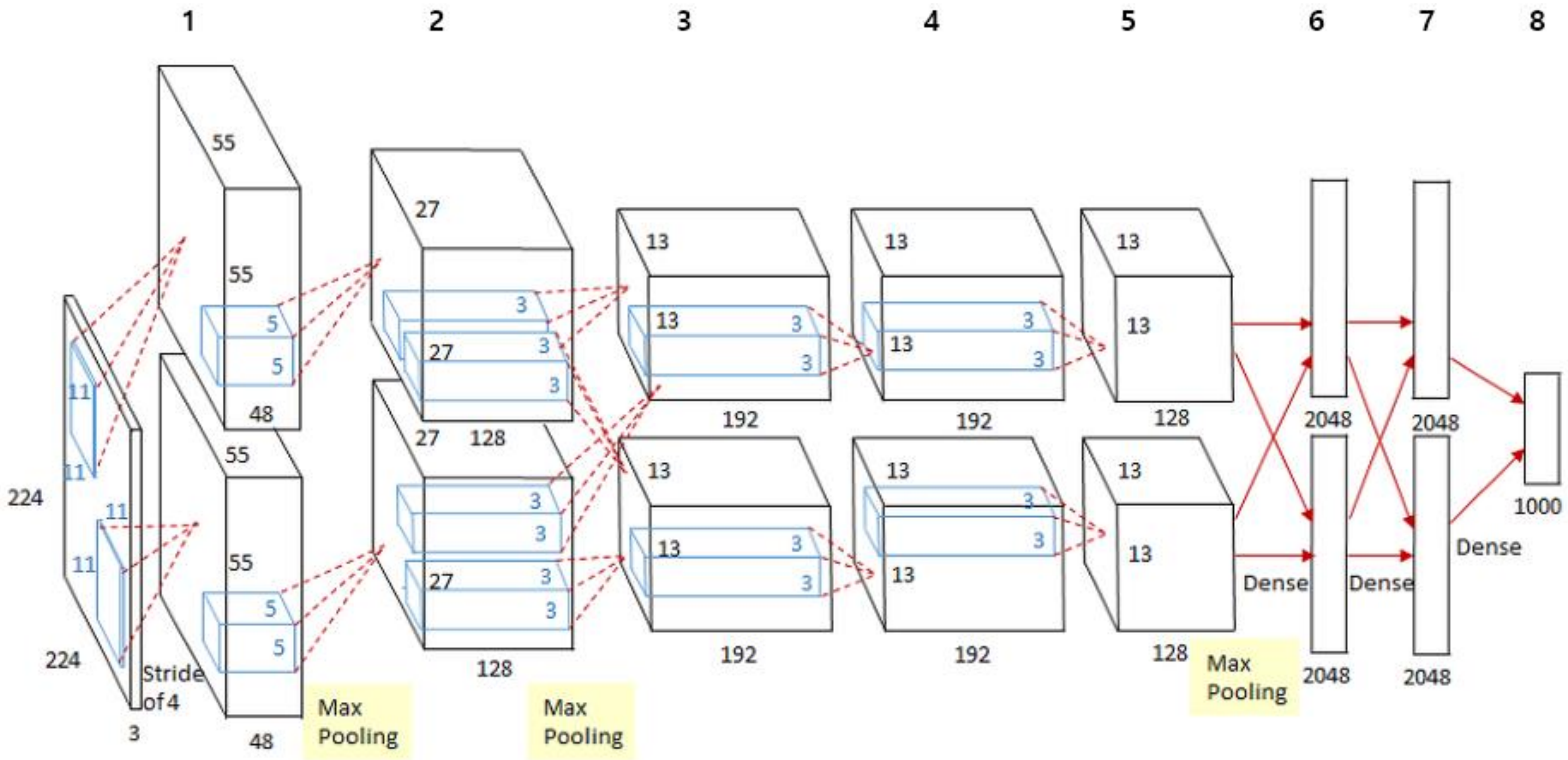
- **Why AlexNet Was a Breakthrough**
  - Demonstrated that deep CNNs could beat traditional hand-crafted features (e.g., SIFT, HOG)
  - Proved that **data + computing power** unlocks deep learning's potential
  - Inspired new models: VGG, GoogLeNet, ResNet

Legacy of AlexNet

1. Opened the era of modern deep learning
2. Changed how AI research and industry approached vision tasks
3. GPUs became essential hardware for training deep models

AlexNet Overall Architecture

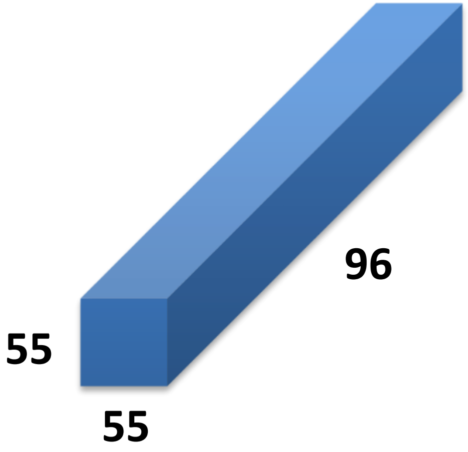
▪ What Does the Architecture of AlexNet Look Like?

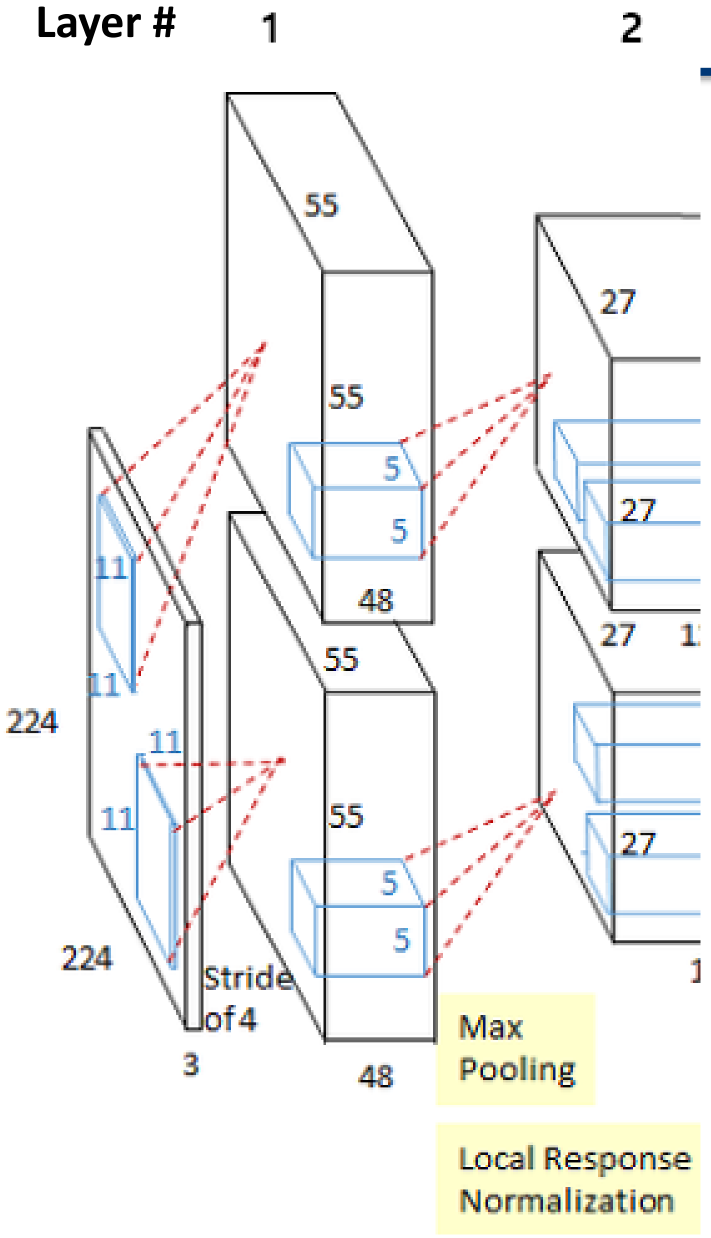


- 8 layers with trainable parameters: **5 Convolutional layers** and **3 Fully Connected layers**
- **Input:** RGB image, 227 × 227 × 3 channels (Red, Blue, Green)
- **Output:** 1000-class softmax
- Uses **two GPUs** to split the computation

AlexNet

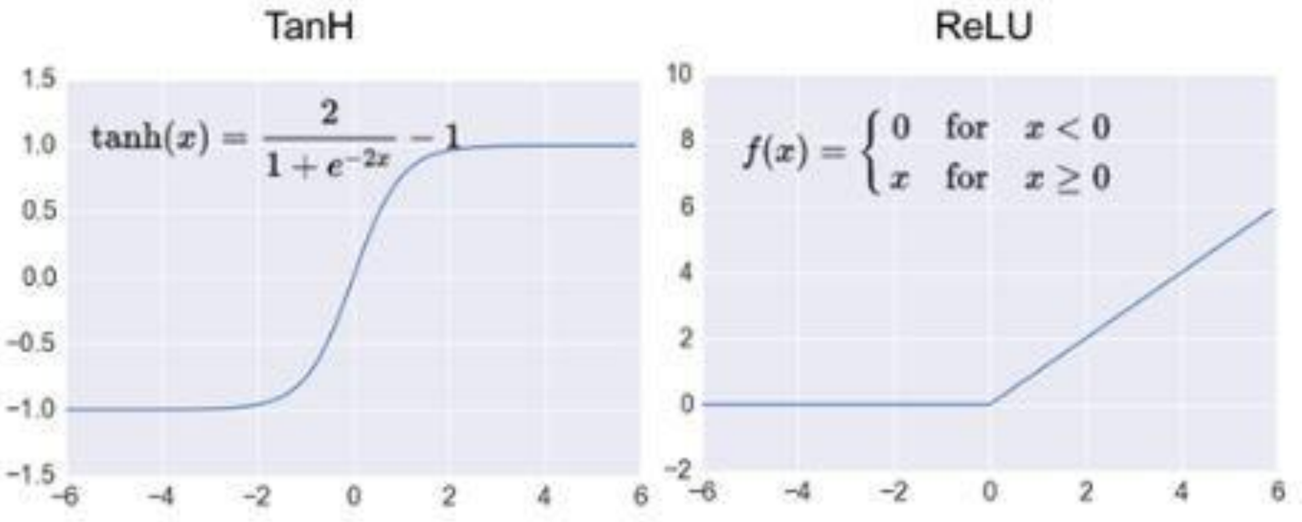
- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - **Convolution layer**
    - Filter (i.e., kernel): 96 filters of size 11×11×3
      → **96 output channels**
  - **Stride = 4, No padding**
  - **Output size:** 55×55×96 channels
  - ***ReLU*** activation
  - ***Overlapping max pooling***: 3×3 window, stride 2
  - ***Local Response Normalization (LRN)***





AlexNet

- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***ReLU*** and Why Was It a Game-Changer?
    - **What is ReLU?**



✓ReLU stands for **Rectified Linear Unit**

✓Formula: $\boldsymbol{ReLU(x) = max(0, x)}$

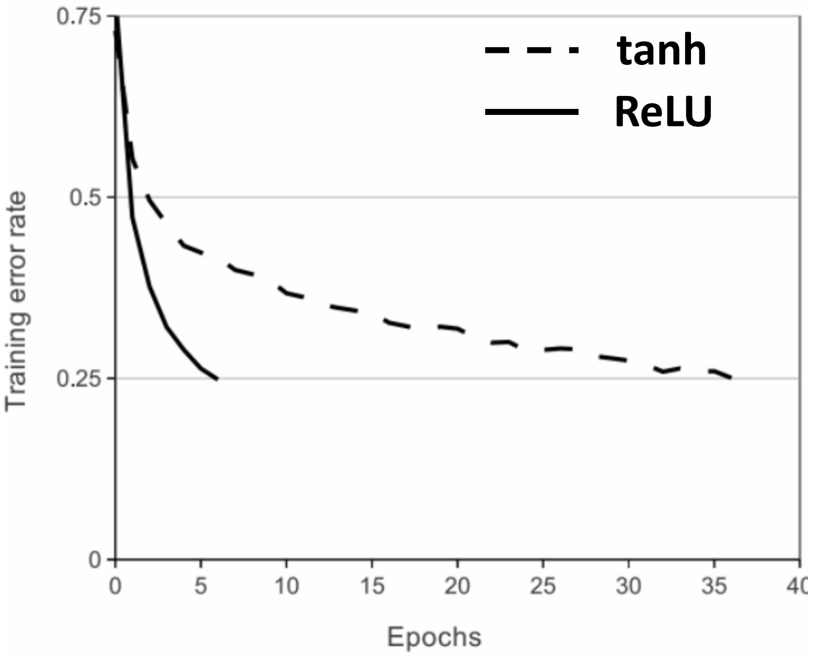
✓Graph:

➢Negative input → 0

➢Positive input → linear increase

AlexNet

- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***ReLU*** and Why Was It a Game-Changer?
    - **Why ReLU Helped AlexNet Succeed**



Property	Sigmoid / tanh (Saturating)	ReLU (Non-saturating)
Output range		
Gradient near 0		
Computation		
Convergence speed		

✓**Non-saturating nonlinearity →** Avoids the vanishing gradient problem

✓**Faster training →** ReLU allowed AlexNet to converge **6x faster** than tanh

✓Helped AlexNet handle a **very deep architecture** with **60M** parameters

✓Without ReLU, large-scale training on ImageNet would have been **impractical**

AlexNet

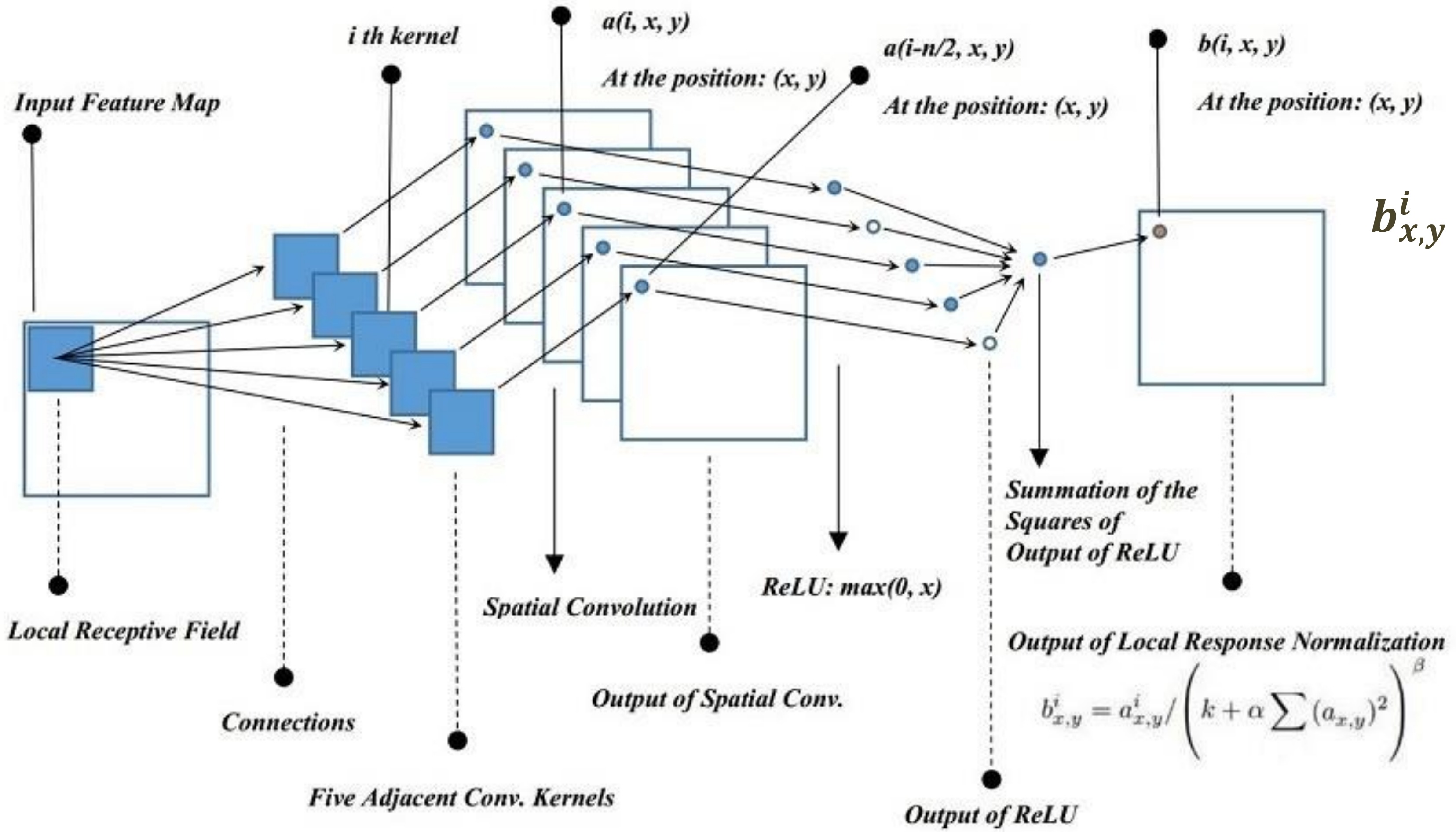
- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***Local Response Normalization (LRN)*** and Why Was It a Game-Changer?
    - **What is Local Response Normalization?**
      - ✓**LRN** is a normalization technique applied **across channels** at the same spatial location (x, y)
      - ✓Inspired by **lateral inhibition** in neuroscience → strong activations suppress weaker neighbors
      - ✓Applied **after ReLU** to encourage feature diversity
    - **Mathematical Formula of LRN**

$$b_{x,y}^{i} =$$

      - ✓$a_{x,y}^{i}$: ReLU output at location (x, y) in the i**-th** channel
      - ✓N: total number of feature maps
      - ✓n: local window size (typically 5)
      - ✓$k = 2, \alpha = 10^{-4}, \beta = 0.75$ (as used in the AlexNet paper)

AlexNet

- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***Local Response Normalization (LRN)*** and Why Was It a Game-Changer?
    - **Visual Explanation of LRN**



$b^i_{x,y} =$

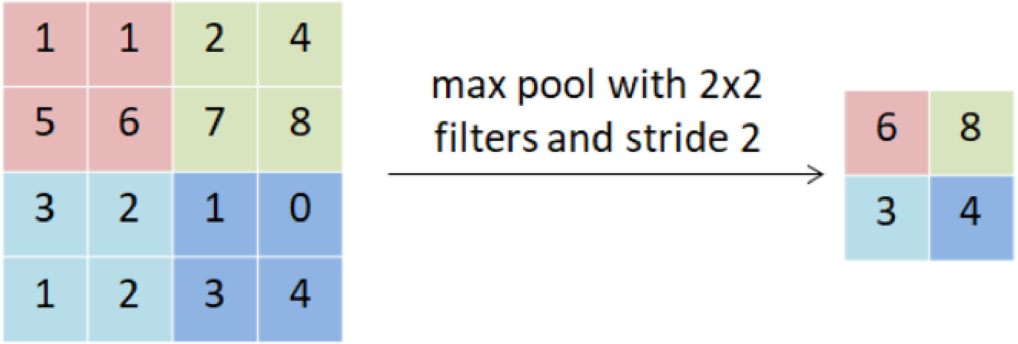
- ✓ n: local window size (typically 5)
- ✓ $k = 2, \alpha = 10^{-4}, \beta = 0.75$
- ✓ $a^i_{x,y}$: ReLU output at location (x, y) in the ***i*-th** channel
- ✓ N: total number of feature maps

AlexNet

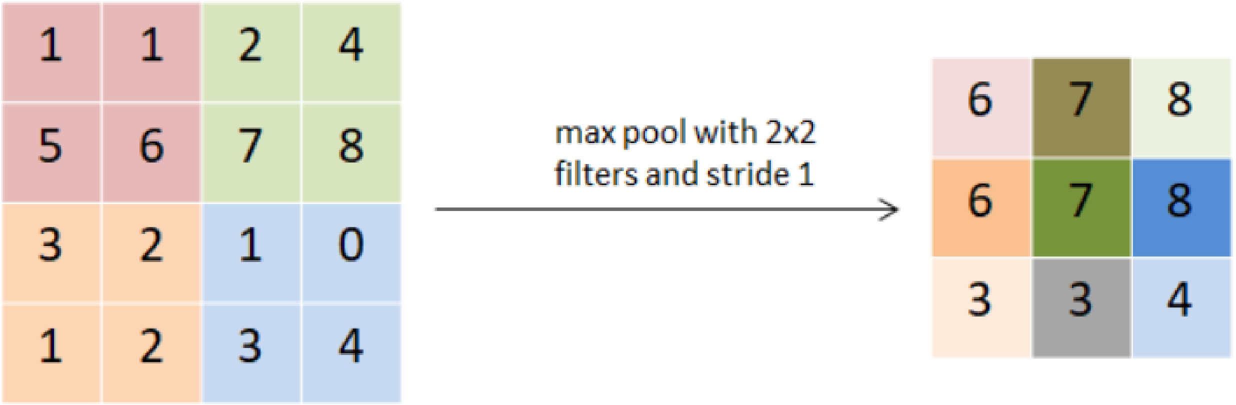
- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***Local Response Normalization (LRN)*** and Why Was It a Game-Changer?
    - **Why Use LRN?**
      - ✓After ReLU, some neurons have **very large outputs**, which can dominate learning
      - ✓LRN **suppresses extreme activations** by normalizing each response with nearby channels
      - ✓This promotes **competition** among neurons → more diverse and selective features
    - **Where Was LRN Used?**
      - ✓Not applied in every layer
      - ✓Used only in **Conv1 and Conv2** in AlexNet
      - ✓Helped reduce
        - ➢**Top-1 error** by **1.4%**
        - ➢**Top-5 error** by **1.2%**

AlexNet

- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***Overlapping Max Pooling*** and Why Was It a Game-Changer?
    - **What is Pooling in CNNs?**
      - ✓Pooling reduces the **spatial size** of feature maps
      - ✓Helps make representations **more compact** and **robust to translation**
      - ✓Most common type
        - ➢ **Max Pooling** — selects the largest value in a region
      - ✓Traditionally used with **non-overlapping filters** (e.g., kernel=2, stride=2)



    - **What is Overlapping Pooling?**
      - ✓Uses a **smaller stride** than the kernel size
      - ✓Example: kernel = 3×3, **stride = 2**
      - ✓Pooling windows **overlap** instead of being disjoint



AlexNet

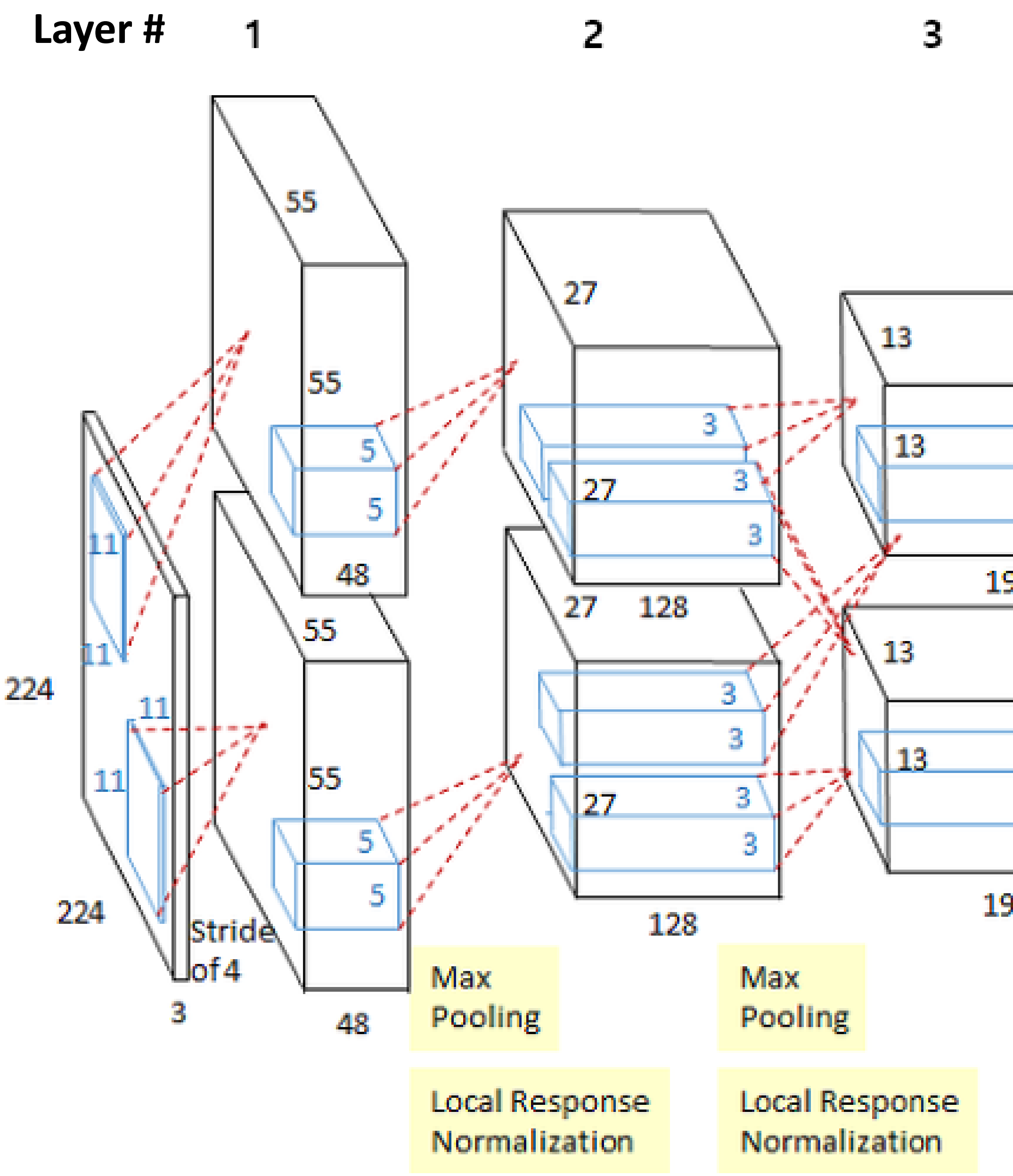
- **Layer 1 — Convolution + ReLU + Pooling + LRN**
  - What is ***Overlapping Max Pooling*** and Why Was It a Game-Changer?
    - **Why Use Overlapping Pooling?**

Metric	Non-Overlapping (Stride = 2)	Overlapping (Stride = 1)
Information		
Receptive field usage		
Accuracy (AlexNet paper)		
Convergence speed		

    - **When to Use Overlapping Pooling?**
      - ✓Useful in **early layers** to retain more spatial detail
      - ✓Helpful when the input is small, and information loss matters
      - ✓More computation than non-overlapping pooling
      - ✓Less commonly used today (often replaced by other techniques like strided convolutions or attention)

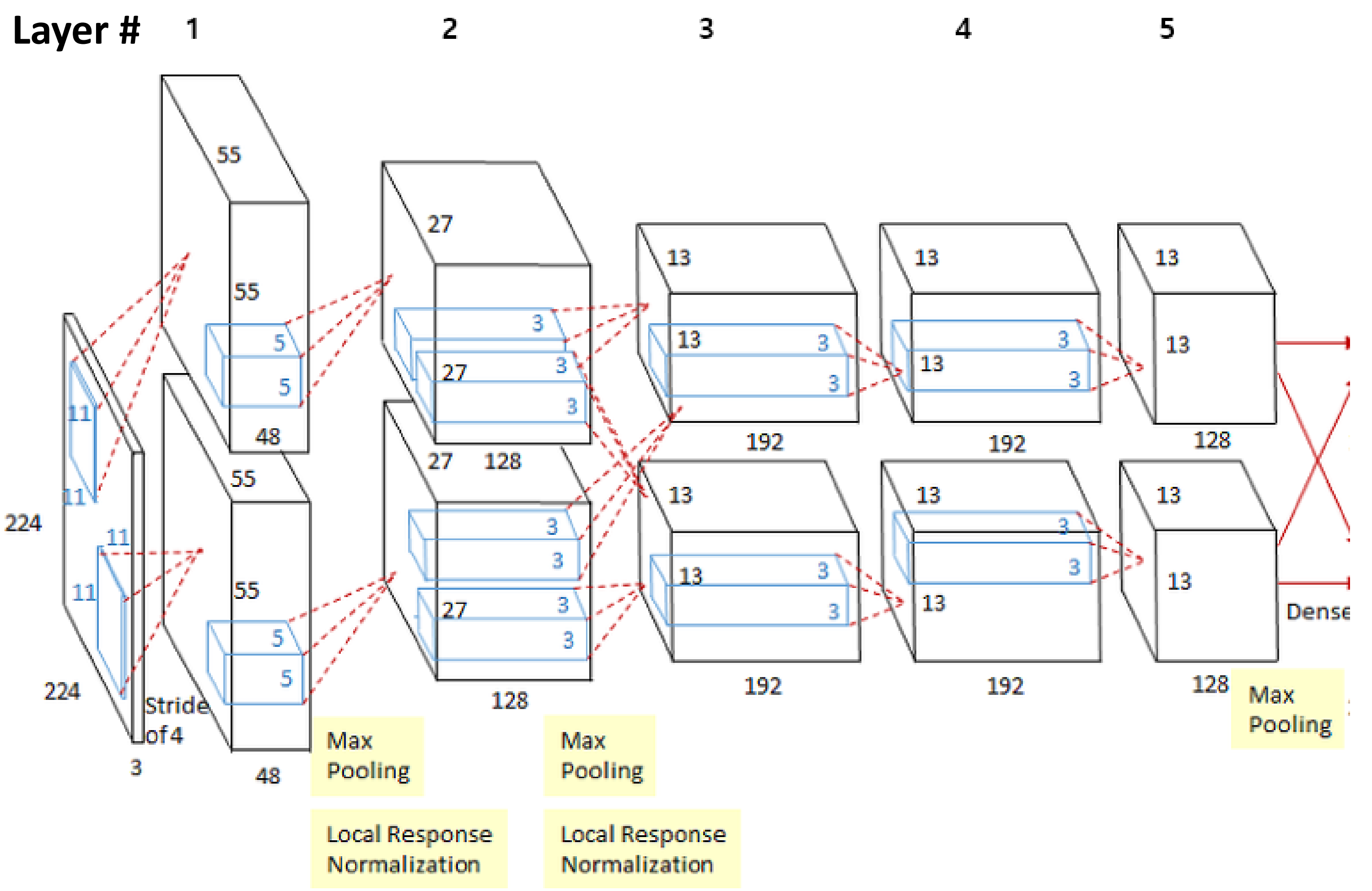
AlexNet

- **Layer 2 — Parallel Convolution + Pooling + LRN**
  - Parallel GPU Processing
    - **128** filters (i.e., kernels) of size 5×5×48 (2 sets due to 2 GPUs)
    - Stride = 1, Padding = 2
    - **Output:** $27 \times 27 \times 256$ $(128 \times 2\ GPUs)$
    - **Overlapping max pooling:** 3×3, stride 2 → 13×13×256
    - **LRN** applied again



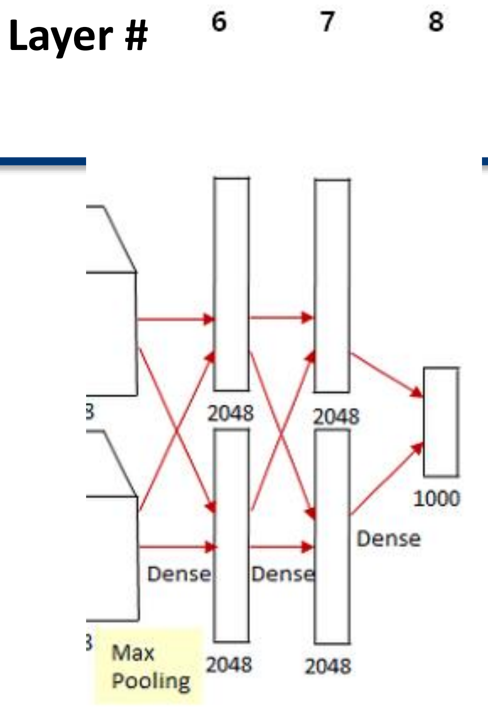
AlexNet

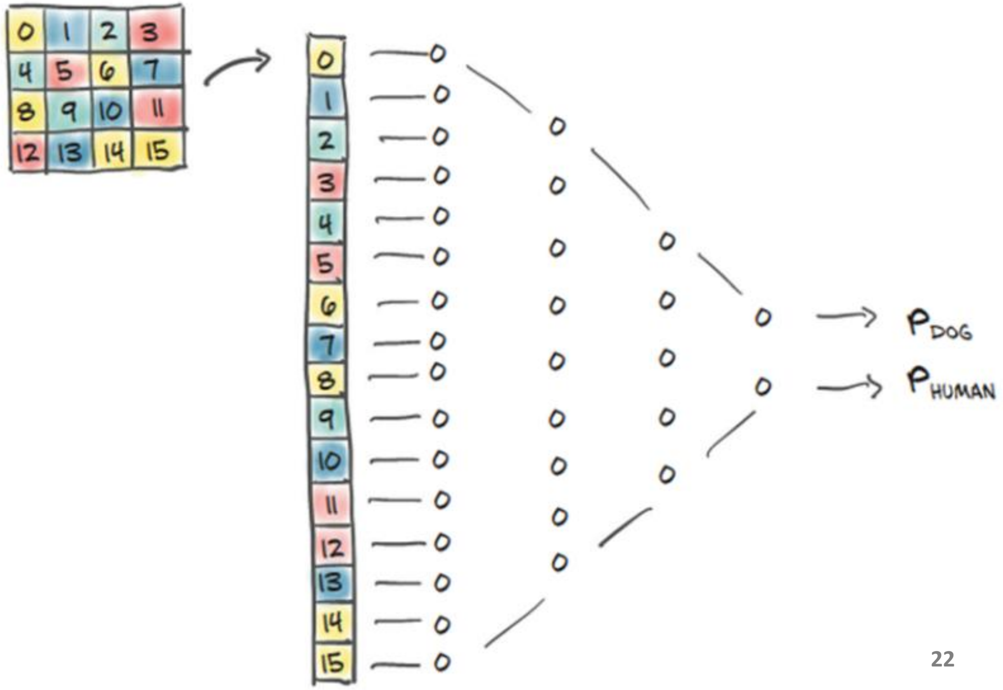
- **Layer 3 to 5 — Depth & Inter-GPU Communication**
  - Deeper Layers for Rich Features
    - ○ **Layer 3**
      - ✓384 filters of size 3×3×256
    - ○ **Layer 4**
      - ✓192 filters of size 3×3×192 (separate on each GPU)
    - ○ **Layer 5**
      - ✓128 filters of size 3×3×192, followed by max pooling → 6×6×256



AlexNet

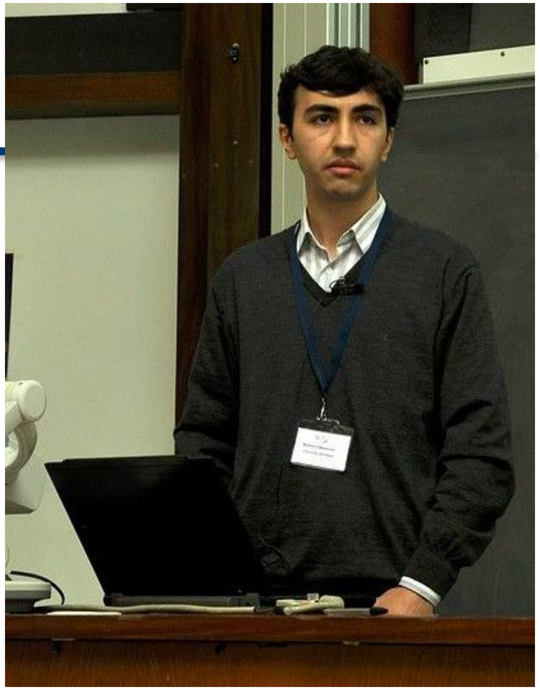
- **Layers 6–8 — Fully Connected & Classification**
  - **Fully Connected Layers**
    - **Layer 6:** FC(9216 → 4096), ReLU, Dropout
    - **Layer 7:** FC(4096 → 4096), ReLU, Dropout
    - **Layer 8:** FC(4096 → 1000), Softmax
  - **What Are Fully Connected Layers?**
    - A **Fully Connected (FC)** layer connects **every neuron** from the previous layer to **every neuron** in the current layer.
    - Equivalent to a **dense matrix multiplication**
      - ✓ $y =$
    - **What Do FC Layers Do?**
      - ✓1.
      - ✓2.
      - ✓3.

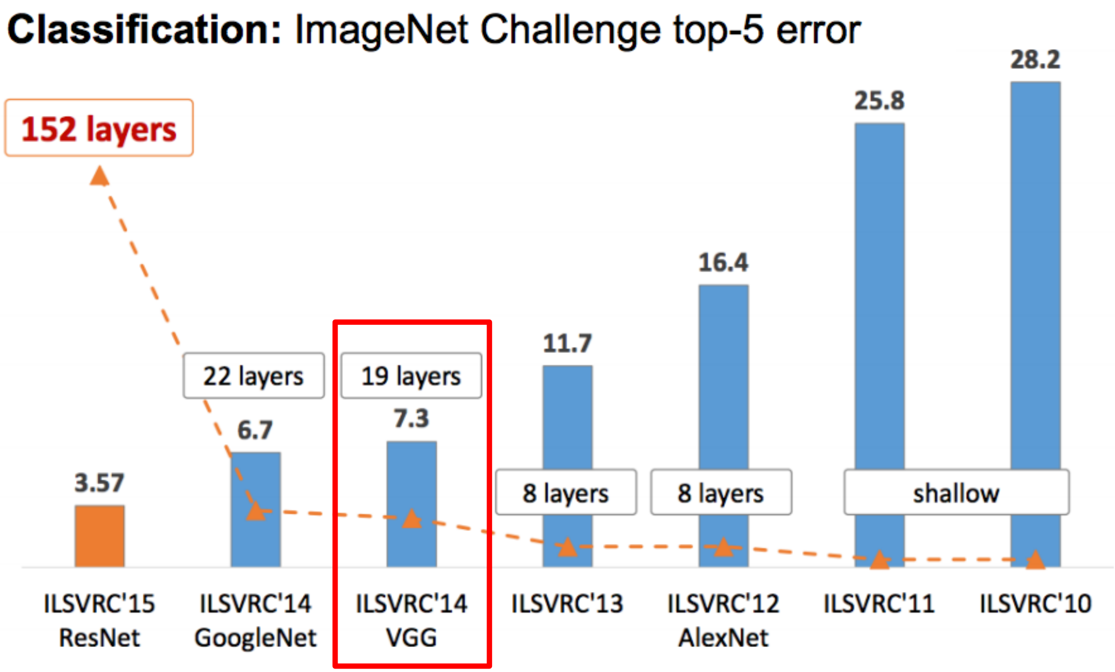




From AlexNet to VGGNet (2014)

- **Introduction to VGGNet (2014)**
  - Developed by **Simonyan** and Zisserman at Oxford's Visual Geometry Group (VGG)
  - Runner-up at **ImageNet Challenge 2014**
  - Known for simplicity and depth: VGG16 and VGG19
  - Uses ***only 3×3 convolution*** filters throughout the network





VGGNet

▪ Motivation and Design of VGGNet

- **Research Goal**
  - Investigate how **network depth** affects image recognition accuracy.

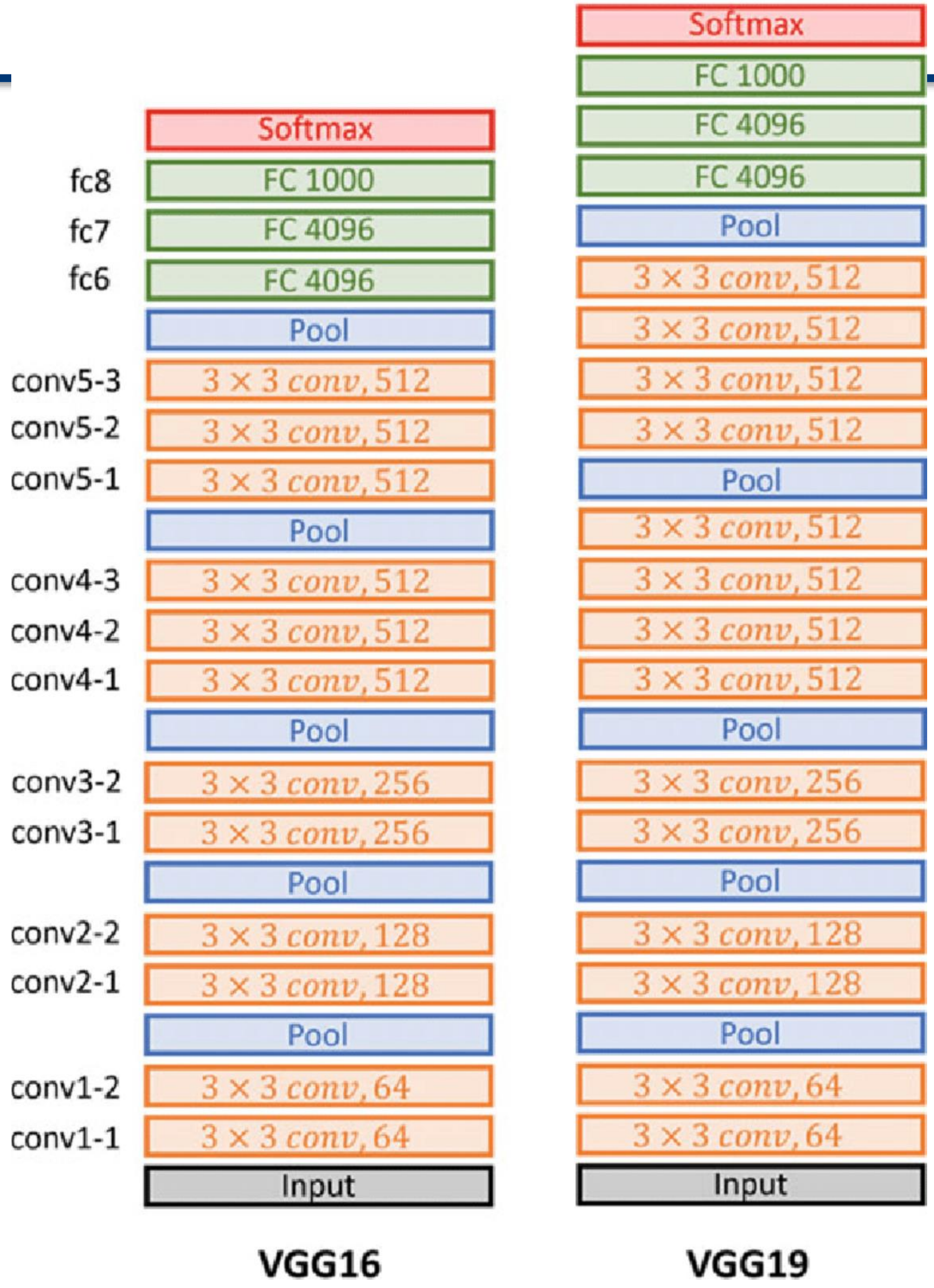
- **Fixed Design Choice**
  - All convolutional layers use **3×3 filters** (small receptive field), with **stride = 1**, and **padding = 1**.
    This choice allows deeper stacking while keeping spatial resolution stable.

- **Progressive Deepening**
  - VGG models are built with **increasing depth** — from 11, 13, 16 to 19 weight layers.
    (These are referred to as configurations A–E in the paper.)

- **Observation**
  - As the depth increased, **classification error consistently decreased** on ImageNet.
    This confirmed that deeper networks can capture more complex patterns and improve accuracy.

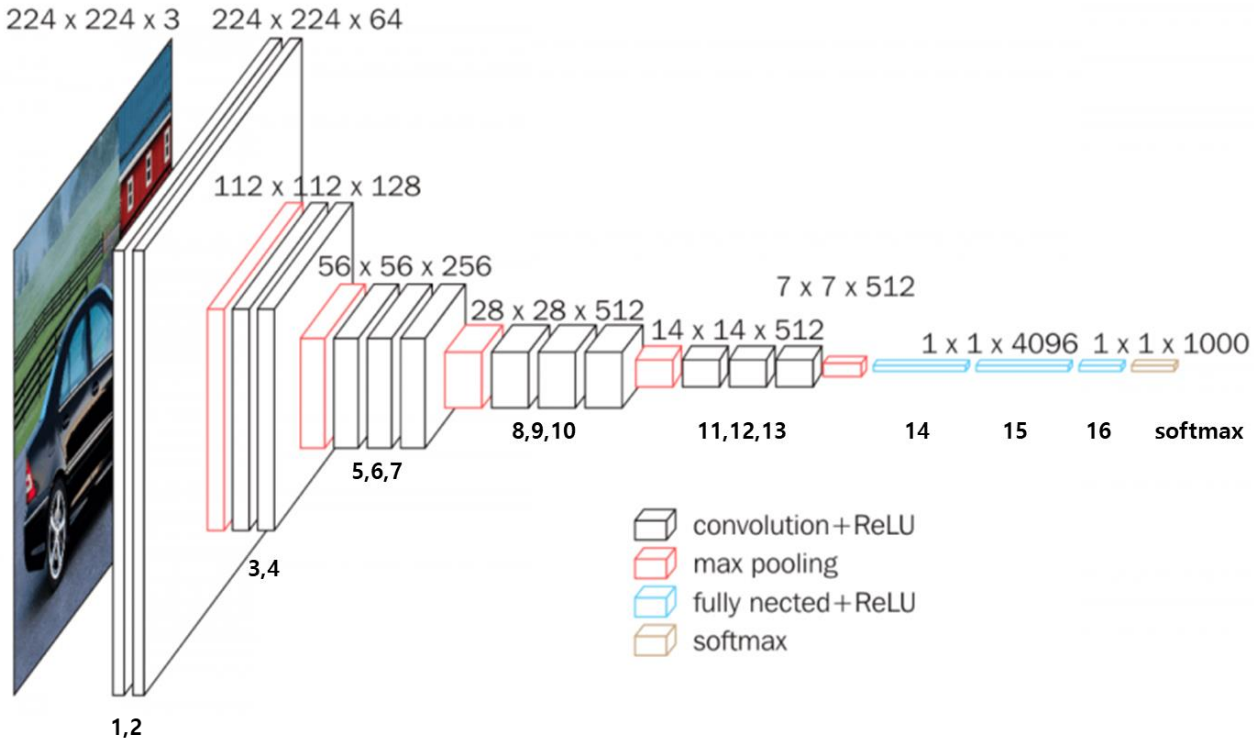
VGGNet

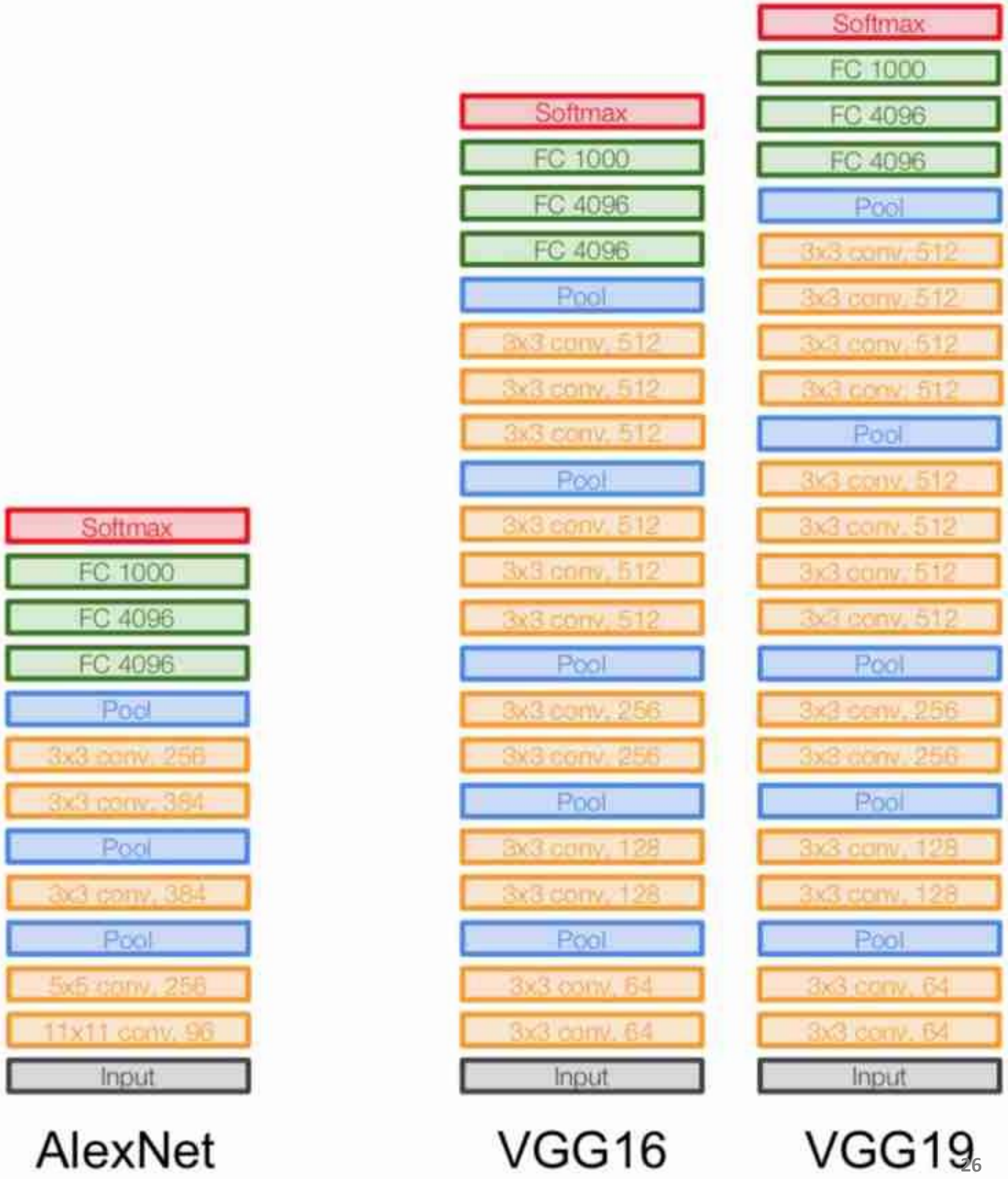
- **Comparison of VGG16 and VGG19 Architectures**
  - ***Common Traits***
    - ○ **Input:** Both **take a 224×224 RGB** image as input
    - ○ **Conv Filter: Use only 3×3 convolution filters** throughout
    - ○ **Pooling: Include 5 max pooling layers**
    - ○ **FC: End with 3 fully connected layers**: 4096 → 4096 → 1000
    - ○ **Activation: ReLU activation** used after each conv layer
    - ○ ***No Local Response Normalization (LRN)***
  - **VGG16**
    - ○ **13 convolutional layers + 3 fully connected layers**
    - ○ Total 16 weight layers
    - ○ Conv blocks: 2-2-3-3-3 pattern
  - **VGG19**
    - ○ **16 convolutional layers + 3 fully connected layers**
    - ○ Total 19 weight layers
    - ○ Conv blocks: 2-2-4-4-4 pattern



VGGNet

- **Comparison of VGG16 and VGG19 Architectures**
  - **Difference from AlexNet**
    - Much deeper (8 layers in AlexNet vs. 16/19 in VGGs)
    - Simpler and more uniform design (only 3×3 filters)
    - Better performance despite higher depth due to small filters and regular structure





VGGNet

- **VGG16**
  - **Architecture Overview**
    - **Input:** 224×224 RGB image
    - **Layers:** 13 convolutional layers + 3 fully connected layers
    - **Conv Filter:** Convolution layers use 3×3 filters, stride = 1, padding = 1
    - **Pooling:** Max pooling: 2×2 with stride 2 after certain conv blocks
    - **Activation:** ReLU used after each conv layer
    - **No Local Response Normalization (LRN) in deeper models**
  - **VGG16 Layer-by-Layer Breakdown**
    - **Conv1_1, Conv1_2**: filters → MaxPooling →
    - **Conv2_1, Conv2_2**: filters → MaxPooling →
    - **Conv3_1 to Conv3_3**: filters → MaxPooling →
    - **Conv4_1 to Conv4_3**: filters → MaxPooling →
    - **Conv5_1 to Conv5_3**: filters → MaxPooling →
    - **FC1, FC2**: neurons each
    - **FC3**: neurons → Softmax

VGGNet

- **VGG16**
  - **Why Stack 3×3 Convolutions?**
    - **Key Idea**
      - ✓Instead of using large filters (e.g., 5×5 or 7×7), VGGNet stacks multiple 3×3 convolution layers to achieve the same **receptive field** but with **more non-linearity** and **fewer parameters**.
    - Stacking two 3×3 convs = 5×5 receptive field
    - Three 3×3 convs = 7×7 receptive field

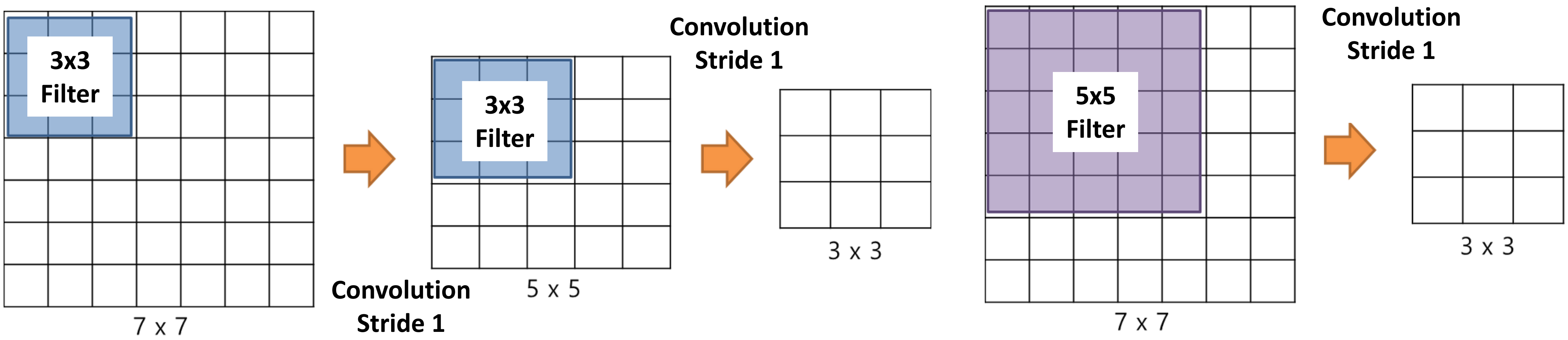
  - **Comparison of Receptive Fields**

Filter Strategy	Effective Receptive Field	Nonlinear Layers	Parameters (per channel)
One 5×5 convolution			
Two stacked 3×3 convolutions			
One 7×7 convolution			
Three stacked 3×3 convolutions			

    - Stacking 3×3s achieves **same receptive field** with:
    - **More non-linearity (ReLU)** → better learning capacity
    - **Fewer parameters** → less overfitting risk and faster training

VGGNet

- **VGG16**
  - **Benefits of 3×3 Stacking**



- **(1) Increased Nonlinearity:** Each convolution is followed by ReLU, increasing the model's capacity to learn complex patterns.
- **(2) Reduced Parameters:** For same output depth C, using 3×3×C×C multiple times requires fewer weights than one large 5×5 or 7×7 convolution.
- **(3) Faster Training:** Fewer parameters mean faster convergence and less memory usage.
- **(4) Modular Design:** Allows deeper networks without exploding parameter count.

VGGNet

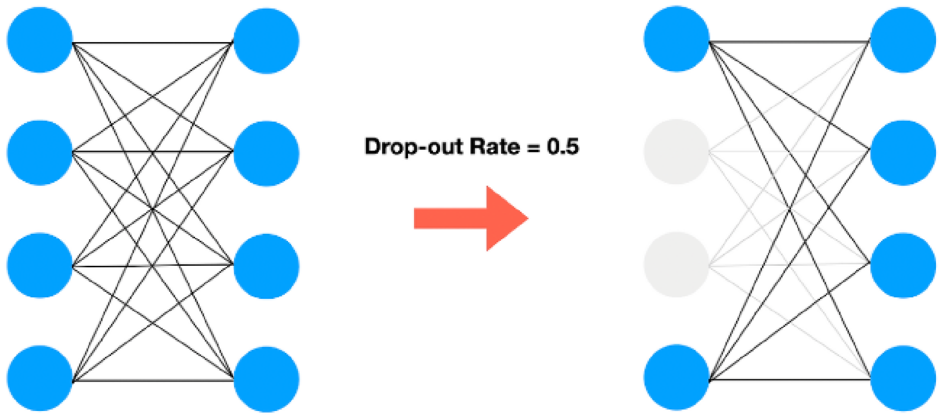
▪ Fully Connected Layers in VGGNet

- **Structure**
  - **Flattened output** from conv layers: $7 \times 7 \times 512$ = **25,088**
  - **FC1**: 4096 units + **ReLU** + **Dropout (p = 0.5)**
  - **FC2**: 4096 units + **ReLU** + **Dropout (p = 0.5)**
  - **FC3**: 1000 units (for classification) + **Softmax**

- **Additional Notes**
  - The FC layers contain the majority of the model's parameters.
  - ***ReLU*** enables **faster training** and introduces **non-linearity**.
  - ***Dropout*** helps reduce overfitting.
  - These FC layers can later be converted into convolutional layers to form a **Fully Convolutional Network**.

VGGNet

- **What is Dropout?**
  - **Concept**
    - **Dropout** is a regularization technique used in neural networks.
    - It works by **randomly "dropping out" neurons** during training with a certain probability, called the **dropout rate**.
    - Dropped neurons are temporarily removed along with their connections, meaning they do not participate in **forward propagation** or **backpropagation** for that training step.
  - **Example (with Dropout Rate = 0.5)**



    - In the figure, the left side shows a fully connected layer where all neurons are active.
    - On the right side, with dropout rate = 0.5, about **half of the neurons are removed at random**.
    - For instance, in this example, 2 out of 4 neurons are dropped. The specific neurons that are dropped can change in each training iteration.

VGGNet

- **What is Dropout?**
  - ***Key Notes***
    - 1. The **dropout rate** is a **hyperparameter**.
      - ✓A common choice is **0.5** for fully connected layers.
    - 2. During training, dropout makes the network less dependent on specific neurons and forces it to learn more **robust and general features**.
    - 3. At test time, however, dropout is **NOT applied.**
      - ✓Instead, all neurons are used with scaling to keep outputs consistent.
    - 4. Dropout helps **prevent overfitting** by reducing reliance on certain strong features, and it also creates an **ensemble effect**, since each random dropout configuration can be seen as training a different sub-network.
      - ✓At inference, combining all neurons is like averaging many models together, improving **generalization**.

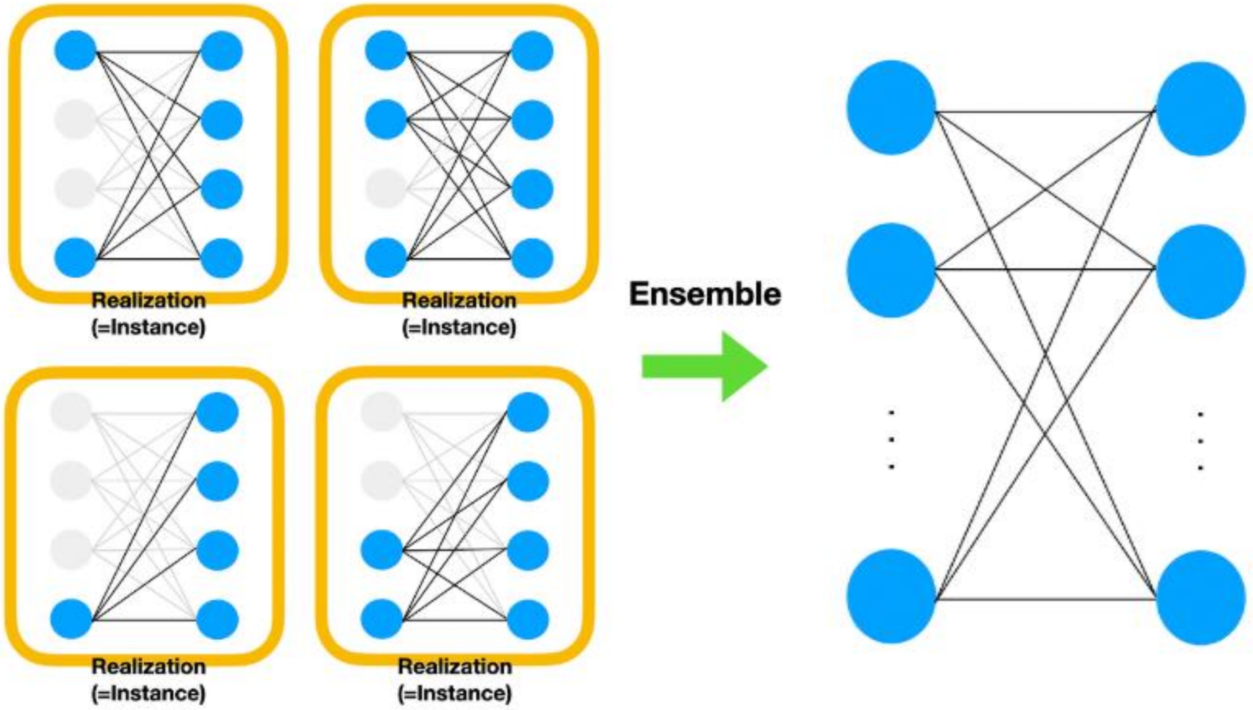
VGGNet

- **Why Do We Use Dropout?**
  - **Preventing Overfitting**
    - Without dropout, a network can easily **overfit**, meaning it relies too heavily on certain strong features.
    - Overfitting reduces the model's ability to generalize to unseen data.
    - Dropout forces the model to learn **robust representations**, since it cannot depend on the presence of specific neurons every time.
  - **Ensemble Effect**
    - Each random dropout configuration creates a **different sub-network**, called a **realization (or instance)**.
    - During training, the network optimizes many different sub-networks in parallel.
    - At test time, all neurons are used together, which is similar to averaging the outputs of many networks.
    - This process acts like an **ensemble of models**, which improves accuracy and reduces bias.

VGGNet

▪ Why Do We Use Dropout?

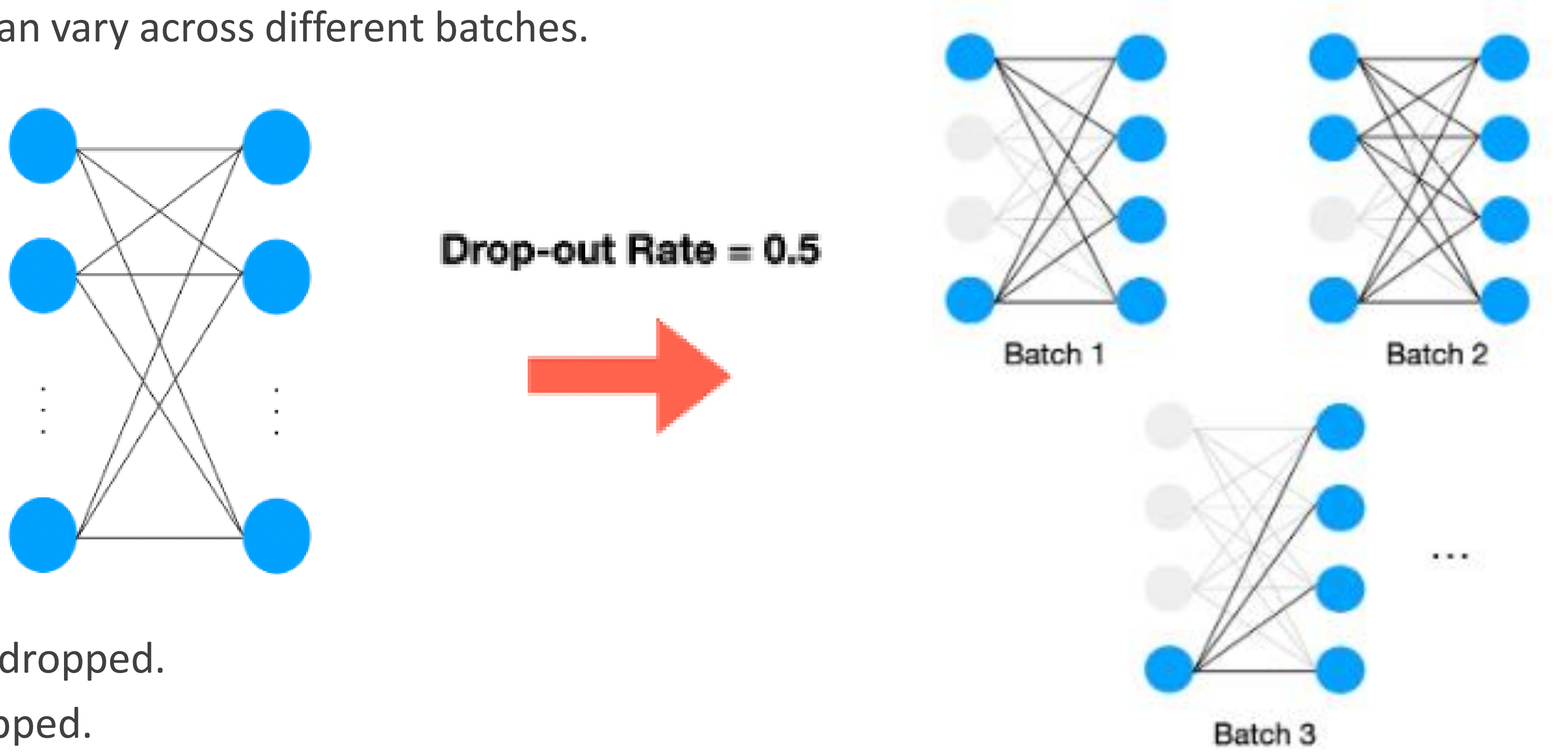
• Key Idea on Ensemble Effect



- On the left, we see multiple realizations where different neurons are dropped in each instance.
- On the right, all of these sub-networks are combined into a single larger network.
- This shows that dropout not only prevents overfitting but also gives the benefits of an **ensemble method** without the need to train many separate models.

VGGNet

- **Dropout in Mini-Batch Training**
  - **How Dropout Works with Mini-Batches**
    - Dropout is applied **independently to each mini-batch** during training.
    - With a dropout rate of **0.5**, each neuron has a 50% chance of being dropped in every batch.
    - The neurons that are dropped can vary across different batches.
  - **Example**



- In **Batch 1**, neurons 2 and 3 are dropped.
- In **Batch 2**, only neuron 3 is dropped.
- In **Batch 3**, neurons 1, 2, and 3 are dropped together.
- This randomness ensures that the network does not rely on specific neurons across all training data.

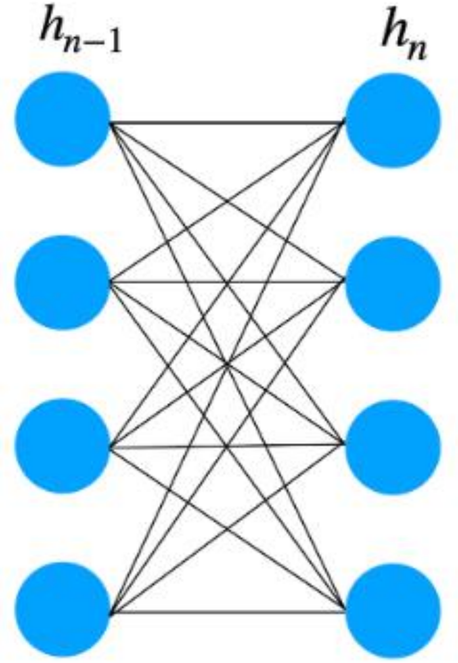
VGGNet

- **Dropout in Mini-Batch Training**
  - **Why This is Important**
    - By changing the dropped neurons in every batch, dropout forces the model to learn **multiple redundant representations**.
    - This leads to stronger **generalization** since no single pathway can dominate the learning process.

    - Because each mini-batch trains a slightly **different sub-network**, dropout creates the effect of training an **ensemble of networks**.
      - ✓ At test time, when all neurons are used together, the model behaves like an **averaged ensemble**, which improves stability and accuracy.

VGGNet

- **Dropout During Testing**
  - **Key Difference from Training**
    - During **training**, some neurons are randomly dropped with probability $\boldsymbol{\alpha}$ (dropout rate).
    - During **testing (inference)**, **all neurons are active**, but their outputs are **scaled** to match the expected values from training.
  - **Scaling Formula**



$$\boldsymbol{h_n} =$$

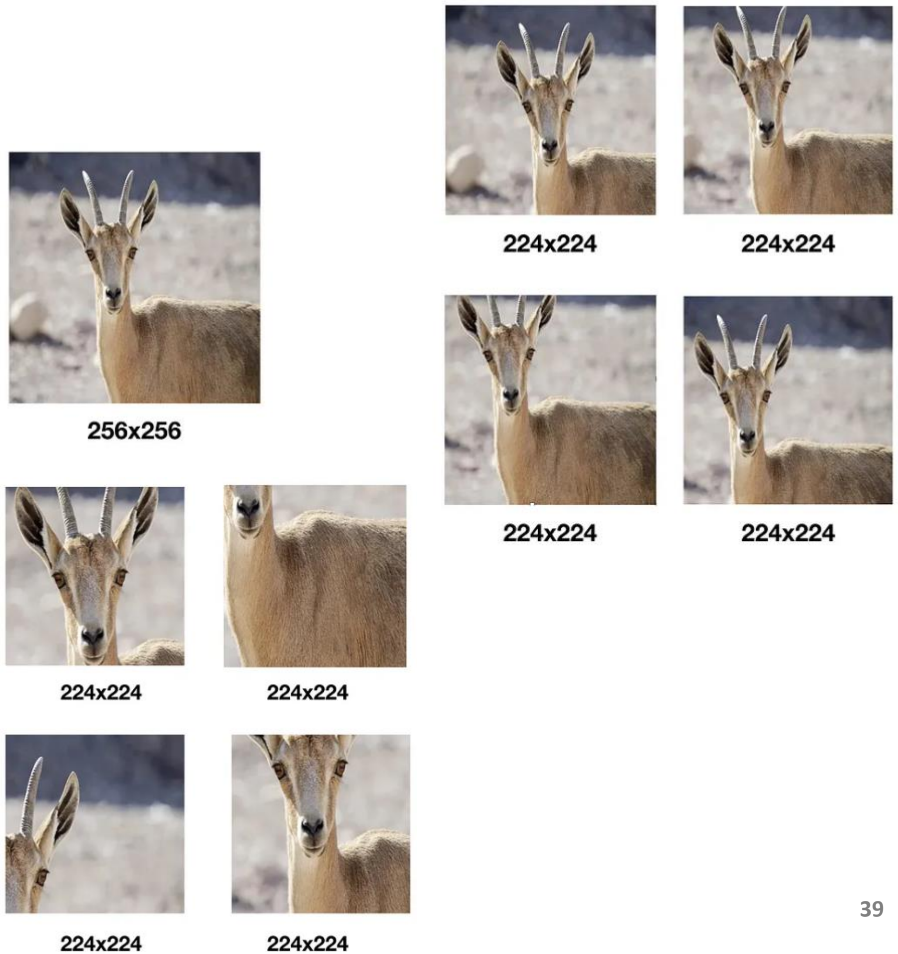
- Here, **a** is the activation function, and $\boldsymbol{\alpha}$ is the dropout rate.
- The factor $(\mathbf{1} - \boldsymbol{\alpha})$ adjusts the output so that the average activation at test time is consistent with training.

VGGNet

- **Dropout During Testing**
  - **Why Scaling is Needed**
    - In training, fewer neurons are active on average due to dropout.
    - Without scaling, test outputs would be **systematically larger**, since all neurons are active.
    - Scaling ensures **fair comparison** between training and testing phases, keeping the same magnitude of outputs.
  - **Key Takeaway**
    - At **training time**: Dropout improves generalization by dropping neurons.
    - At **test time**: No neurons are dropped, but **scaling compensates for dropout**, ensuring consistent behavior and stable predictions.

VGGNet

- **Data Augmentation and Scale Jittering**
  - **Data Augmentation Techniques**
    - ○ **Resize** each training image so that the **shortest side ≥ 256**
    - ○ **Random crop**: 224 × 224 window
    - ○ **Random horizontal flip**
    - ○ **Color jittering**
  - **Scale Jittering**
    - ○ **Single-scale training**: fixed S = 256
    - ○ **Multi-scale training**: S randomly sampled from [256, 512]



VGGNet

- **Data Augmentation and Scale Jittering**
  - **Advantages**
    - Increases dataset diversity
    - Captures different **object contexts**
      - ✓Small crops → entire object
      - ✓Large crops → object parts
    - Helps **reduce overfitting**
    - Experiments showed **multi-scale training improved classification accuracy** over single-scale

isotropically-rescaled training image