Week 9: Deep CNNs

AlexNet, InceptionNet & EfficientNet

Deep Learning for Perception — BSCS, FAST-NUCES

Comprehensive Architecture Analysis with Diagrams

Learning Objectives

By the end of this lecture, you will:

1. Understand the ImageNet dataset and its impact on deep learning
2. Master the concept of transfer learning and fine-tuning
3. Analyze AlexNet architecture and its breakthrough innovations
4. Understand InceptionNet's parallel filter approach
5. Learn EfficientNet's compound scaling methodology
6. Compare architectural evolution from AlexNet to EfficientNet
7. Know when to use each architecture in practice

Prerequisites: Convolutional layers, pooling, activation functions, backpropagation

1 ImageNet and the Deep Learning Revolution

Why It Matters

ImageNet changed everything. Before 2012, computer vision used hand-crafted features. After AlexNet won ImageNet 2012, deep learning became the dominant paradigm. Every modern vision system traces back to this moment. Understanding ImageNet helps you appreciate why these architectures matter.

1.1 The ImageNet Dataset

Definition

ImageNet: A large-scale dataset containing over 14 million images across 20,000+ categories, organized according to the WordNet hierarchy.
ILSVRC (ImageNet Large Scale Visual Recognition Challenge):

- Annual competition from 2010-2017
- 1000 object categories
- 1.2 million training images
- 50,000 validation images
- 100,000 test images

- Task: Classify images into one of 1000 categories
- Metric: Top-5 error rate (prediction correct if true label in top 5 predictions)

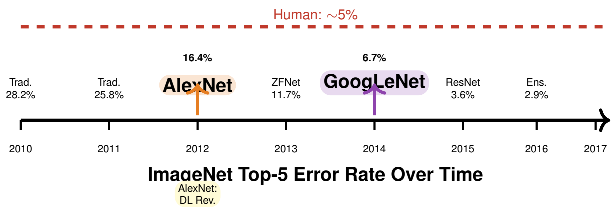


Figure 1: ImageNet competition results showing the breakthrough moment in 2012

Historical Context

The 2012 Breakthrough:
Before AlexNet (2010-2011):

- Traditional methods: SIFT, HOG, hand-crafted features
- Support Vector Machines (SVMs) for classification
- Error rates plateauing around 25-28%
- Incremental improvements year-over-year

AlexNet (2012):

- First deep CNN to win ImageNet
- Error rate: 16.4% (nearly 50% reduction!)
- Proved deep learning works at scale
- Sparked the deep learning revolution

Impact:

- Every winning entry after 2012 used deep CNNs
- By 2015, models surpassed human performance (5%)
- Industry-wide adoption of deep learning
- Birth of modern computer vision

1.2 Why ImageNet Matters

Analogy

ImageNet as the "Olympics" of Computer Vision:
Just like Olympic records drive athletic innovation, ImageNet drove AI innovation:

- **Standardized benchmark:** Everyone competes on same dataset
- **Scale matters:** 1.2M images forced efficient architectures
- **Diverse categories:** 1000 classes test generalization
- **Public leaderboard:** Accelerated research through competition
- **Pre-trained models:** Winners become starting points for all vision tasks

Result: ImageNet-trained models are now the foundation for ALL computer vision applications, from medical imaging to self-driving cars!

2 Transfer Learning and Fine-Tuning

Why It Matters

Training deep CNNs from scratch requires massive datasets (millions of images) and compute (days/weeks on GPUs). Transfer learning lets you use pre-trained ImageNet models and adapt them to YOUR task with just hundreds of images in hours. This is how 99% of real-world deep learning happens!

2.1 The Core Concept

Transfer Learning

Main Idea: Use knowledge learned from one task (ImageNet classification) to solve a different but related task (your specific problem).
Why it works:

- Early layers learn **generic features:** edges, textures, colors
- Middle layers learn **patterns:** shapes, objects parts
- Late layers learn **task-specific features:** class-specific patterns
- Generic features transfer well across tasks!

Key insight: A network trained on ImageNet has already learned to detect edges, textures, and basic shapes — features useful for ANY vision task!

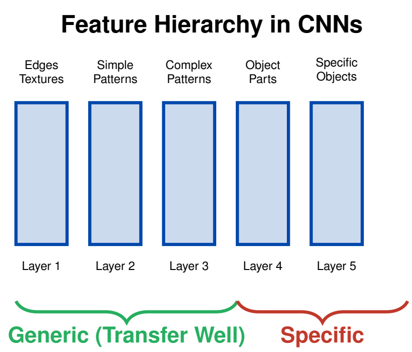


Figure 2: Early layers learn generic features that transfer; late layers are task-specific

2.2 Fine-Tuning Strategies

Fine-Tuning Methods

Strategy 1: Feature Extraction (Frozen Backbone)

- **What:** Freeze all pre-trained layers, train only new classification head
- **When:** Small dataset (hundreds of images), similar to ImageNet
- **How:** Replace final layer, set requires_grad=False for backbone
- **Speed:** Very fast (only training 1 layer)

Strategy 2: Fine-Tuning Top Layers

- **What:** Freeze early layers, fine-tune last few layers + new head
- **When:** Medium dataset (thousands), somewhat different from ImageNet
- **How:** Freeze first N layers, train rest with small learning rate
- **Speed:** Moderate

Strategy 3: Fine-Tuning All Layers

- **What:** Train entire network with very small learning rate
- **When:** Large dataset (10K+ images), quite different from ImageNet
- **How:** Use learning rate ∼10x smaller than training from scratch
- **Speed:** Slower but best accuracy

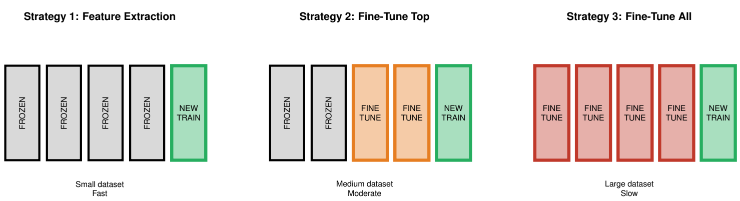


Figure 3: Three fine-tuning strategies depending on dataset size

2.3 Practical Implementation

Fine-Tuning in PyTorch

Step 1: Load Pre-trained Model

```
import torchvision.models as models

# Load ImageNet pre-trained model
model = models.resnet50(pretrained=True)
# or: models.efficientnet_b0(pretrained=True)
```

Step 2: Modify Final Layer

```
num_classes = 10  # Your task's classes

# Replace final layer (1000 -> 10 classes)
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

Step 3: Choose Strategy
Strategy 1: Freeze all except final layer

```
for param in model.parameters():
    param.requires_grad = False
model.fc.requires_grad = True  # Only train new layer
```

Strategy 2: Freeze early, fine-tune late

```
# Freeze first half
for name, param in model.named_parameters():
    if "layer4" not in name and "fc" not in name:
        param.requires_grad = False
```

Strategy 3: Fine-tune all (use small LR!)

```
# All layers trainable (default)
optimizer = torch.optim.Adam(model.parameters(),
                             lr=1e-4)  # 10x smaller!
```

Learning Rate Guidelines:

- Training from scratch: 0.001 - 0.01
- Fine-tuning all layers: 0.0001 - 0.001
- Training only new head: 0.001 - 0.01

Quick Reference

Quick Decision Guide for Fine-Tuning:
Dataset Size:

- 100-1K images: Strategy 1 (feature extraction)
- 1K-10K images: Strategy 2 (fine-tune top layers)
- 10K+ images: Strategy 3 (fine-tune all)

Dataset Similarity to ImageNet:

- Very similar (natural images): Strategy 1 works great
- Somewhat different (medical/satellite): Strategy 2
- Very different (microscopy/X-rays): Strategy 3

Compute Budget:

- Limited (CPU, hours): Strategy 1
- Moderate (single GPU, days): Strategy 2
- Generous (multi-GPU, weeks): Strategy 3

Pro Tip: Always start with Strategy 1, then move to 2 or 3 if underfitting!

3 AlexNet: The Breakthrough (2012)

Why It Matters

AlexNet didn't just win ImageNet — it changed the field forever. This was the moment deep learning proved it works. Every technique we use today (ReLU, dropout, data augmentation, GPU training) was pioneered or popularized by AlexNet. Understanding AlexNet means understanding the foundation of modern deep learning.

3.1 Historical Context and Innovation

Historical Context

The Challenge in 2012:

- ImageNet: 1.2M images, 1000 classes
- Previous best: 25.8% top-5 error (traditional methods)
- Deep learning considered impractical (vanishing gradients, compute limits)
- GPUs just becoming powerful enough (NVIDIA GTX 580)

AlexNet's Result:

- Top-5 error: 16.4% (37% relative improvement!)
- First deep CNN to win ImageNet
- Trained on 2 GPUs for 5-6 days
- 60 million parameters

Authors: Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton (University of Toronto)
Impact: Sparked the deep learning revolution, leading to modern AI breakthroughs in vision, NLP, and beyond.

3.2 Key Innovations

AlexNet's Revolutionary Techniques

1. ReLU Activation Function

- **Before:** Sigmoid, tanh (saturate, slow training)
- **AlexNet:** ReLU (f(x) = max(0,x))
- **Benefit:** 6x faster training, no vanishing gradient
- **Impact:** ReLU became the standard activation

2. Dropout Regularization

- Drop neurons randomly during training (p=0.5)
- Prevents overfitting on large networks
- First large-scale application of dropout

3. Data Augmentation

- Random crops, horizontal flips
- Color jittering (PCA on RGB)
- Effectively increased dataset 2000x!

4. GPU Training

- First ImageNet winner using GPUs
- Split network across 2 GPUs
- Made training feasible (days instead of months)

5. Local Response Normalization (LRN)

- Normalize activations across channels
- Later replaced by Batch Normalization

6. Overlapping Pooling

- 3x3 pooling with stride 2 (instead of 2x2 stride 2)
- Slight accuracy improvement

3.3 AlexNet Architecture

Architecture Details

Architecture Overview:

- **Input:** 224x224x3 RGB images
- **8 learned layers:** 5 convolutional + 3 fully connected
- **Parameters:** $\sim$60 million
- **Output:** 1000 classes (ImageNet)

Layer-by-Layer Breakdown:

1. **Conv1:** 96 filters, 11x11, stride 4 $\rightarrow$ 55x55x96
   - ReLU activation
   - Max pooling 3x3, stride 2 $\rightarrow$ 27x27x96
   - LRN (Local Response Normalization)
2. **Conv2:** 256 filters, 5x5 $\rightarrow$ 27x27x256
   - ReLU activation
   - Max pooling 3x3, stride 2 $\rightarrow$ 13x13x256
   - LRN
3. **Conv3:** 384 filters, 3x3 $\rightarrow$ 13x13x384
   - ReLU activation (no pooling, no LRN)
4. **Conv4:** 384 filters, 3x3 $\rightarrow$ 13x13x384
   - ReLU activation
5. **Conv5:** 256 filters, 3x3 $\rightarrow$ 13x13x256
   - ReLU activation

- Max pooling 3x3, stride 2 $\rightarrow$ 6x6x256

6. **FC6:** Fully connected, 4096 neurons
   - ReLU activation
   - Dropout (p=0.5)
7. **FC7:** Fully connected, 4096 neurons
   - ReLU activation
   - Dropout (p=0.5)
8. **FC8 (Output):** Fully connected, 1000 neurons
   - Softmax activation

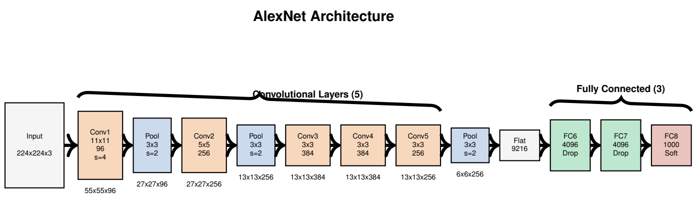


Figure 4: Complete AlexNet architecture showing all 8 layers

Quick Reference

AlexNet Key Takeaways:
Architecture Patterns:

- Start with large filters (11x11), reduce to small (3x3)
- Increase depth gradually (96 $\rightarrow$ 256 $\rightarrow$ 384)
- Pool after first two conv layers to reduce dimensions
- Large FC layers at end (4096 neurons each)

Training Details:

- SGD with momentum (0.9)
- Weight decay: 0.0005
- Dropout: 0.5 in FC6 and FC7
- Learning rate: 0.01, divided by 10 when plateau
- Batch size: 128
- Training time: 5-6 days on 2 GPUs

Why It Worked:

- Deep enough to learn hierarchical features
- ReLU enabled training deep networks
- Dropout prevented overfitting
- Data augmentation increased effective dataset size
- GPU made it computationally feasible

Legacy: Every modern CNN uses ReLU, dropout, and data augmentation — all popularized by AlexNet!

4 InceptionNet (GoogLeNet): Thinking Wider (2014)

Why It Matters

After AlexNet, the race was on: how to make networks better? Most tried going DEEPER (more layers). InceptionNet went WIDER instead, using multiple filter sizes in parallel. This “Inception module” became a fundamental building block, proving that architectural innovation matters as much as scale. InceptionNet won ImageNet 2014 with 6.7% error — half of AlexNet's!

4.1 The Core Problem and Solution

The Multi-Scale Challenge

Problem: Objects appear at different scales in images

- Small objects: need small receptive fields (3x3)
- Large objects: need large receptive fields (5x5, 7x7)
- Traditional approach: pick ONE filter size (compromise!)

InceptionNet's Solution: Use ALL filter sizes in parallel!

- Run 1x1, 3x3, and 5x5 convolutions simultaneously
- Also add max pooling path
- Concatenate all outputs
- Let the network learn which scale matters for each feature

Key Insight: Instead of choosing one filter size, try them all and let the network decide!

Analogy

The Restaurant Menu Analogy:
Traditional CNN (single filter):

- Restaurant offers only one dish size
- Some customers want small portions, others large
- One size fits nobody perfectly

InceptionNet (multiple parallel filters):

- Restaurant offers small, medium, and large simultaneously
- Customer's stomach (network) decides what to eat
- Everyone gets exactly what they need!

Result: Better satisfaction (accuracy) without wasting food (computation)!

4.2 The Inception Module (Naive Version)

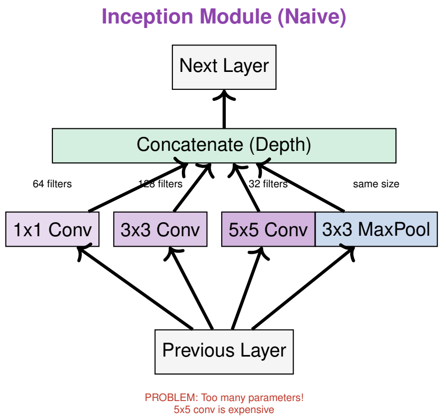


Figure 5: Naive Inception module: parallel filters at multiple scales

4.3 The Dimensionality Reduction Trick

1x1 Convolutions: The Secret Weapon

The Problem: 5x5 convolutions are EXPENSIVE!

- Input: 28x28x256
- Output: 28x28x32 (using 32 filters of 5x5x256)
- Computation: $28 \times 28 \times 32 \times 5 \times 5 \times 256 = 120M$ operations!

The Solution: Add 1x1 conv BEFORE 5x5 to reduce channels!
With 1x1 Bottleneck:

1. 1x1 conv: 28x28x256 $\rightarrow$ 28x28x16 (reduce to 16 channels)
    - Cost: $28 \times 28 \times 16 \times 1 \times 1 \times 256 = 3.2M$ ops
2. 5x5 conv: 28x28x16 $\rightarrow$ 28x28x32
    - Cost: $28 \times 28 \times 32 \times 5 \times 5 \times 16 = 10M$ ops
3. Total: 3.2M + 10M = 13.2M operations

Savings: 120M $\rightarrow$ 13M = **90% reduction!**
Key Insight: 1x1 convolutions reduce channel dimension without losing spatial information, making deeper networks computationally feasible!

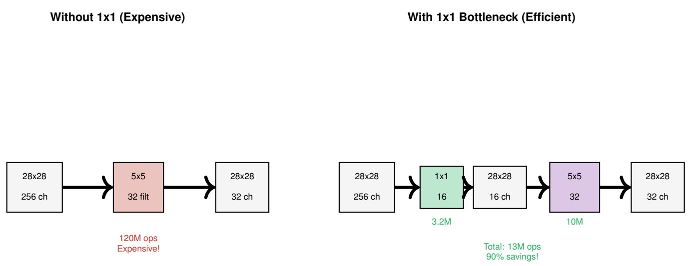


Figure 6: 1x1 convolutions dramatically reduce computation

4.4 Inception Module with Dimensionality Reduction

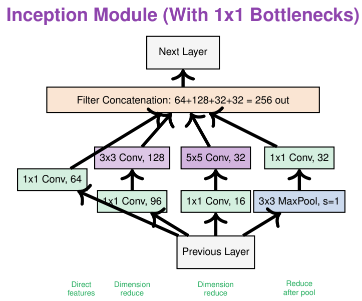


Figure 7: Complete Inception module with dimensionality reduction

4.5 GoogLeNet (Inception V1) Architecture

Architecture Details

GoogLeNet Overview:

- 22 layers deep (only counting layers with parameters)
- 9 Inception modules stacked together
- Only 6.8 million parameters (12x fewer than AlexNet!)
- Input: 224x224x3
- Output: 1000 classes

Architecture Structure:

1. **Stem:** Initial conv + pooling layers
2. **Inception Blocks:** 9 inception modules
3. **Global Average Pooling:** Replace FC layers
4. **Auxiliary Classifiers:** Help training (unique feature!)

Key Innovation - Auxiliary Classifiers:

- Problem: Deep networks suffer from vanishing gradients
- Solution: Add intermediate classifiers at layers 3a and 4a
- During training: Compute loss at 3 points (main + 2 auxiliary)

- Total loss: $L_{total} = L_{main} + 0.3 \times (L_{aux1} + L_{aux2})$
- During inference: Discard auxiliary classifiers, use only main output
- Benefit: Gradients flow better through network

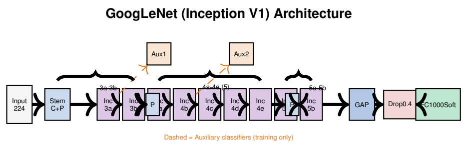


Figure 8: Complete GoogLeNet architecture with 9 Inception modules

Quick Reference

InceptionNet Quick Reference:
Key Innovations:

- Parallel multi-scale convolutions (1x1, 3x3, 5x5, pooling)
- 1x1 bottleneck convolutions for efficiency
- Global average pooling (replaces FC layers)
- Auxiliary classifiers for better gradient flow

Benefits over AlexNet:

- Deeper (22 vs 8 layers)
- Fewer parameters (6.8M vs 60M)
- Better accuracy (6.7% vs 16.4% error)
- More computationally efficient

When to Use InceptionNet:

- Need multi-scale feature extraction
- Limited computational budget
- Objects at varying scales in images
- Good balance of accuracy and efficiency

Variants:

- Inception V2: Batch normalization
- Inception V3: Factorized convolutions (5x5 $\rightarrow$ two 3x3)
- Inception V4: Combined with ResNet connections
- Inception-ResNet: Best of both worlds

5 EfficientNet: Balanced Scaling (2019)

Why It Matters

After years of architectural innovation (Inception, ResNet, DenseNet), EfficientNet asked a fundamental question: *How should we scale networks?* Most just made networks deeper or wider randomly. EfficientNet systematically studied scaling and found that **compound scaling** (balancing depth, width, and resolution) dramatically outperforms scaling any single dimension. Result: State-of-the-art accuracy with 10x fewer parameters!

5.1 The Scaling Problem

Traditional Scaling Approaches

Goal: Improve model accuracy by making it bigger
Method 1: Depth Scaling (More Layers)

- ResNet-50 $\rightarrow$ ResNet-101 $\rightarrow$ ResNet-152
- **Problem:** Diminishing returns, vanishing gradients, hard to optimize
- **Result:** 2x layers $\neq$ 2x accuracy

Method 2: Width Scaling (More Channels)

- Increase filters per layer: 64 $\rightarrow$ 128 $\rightarrow$ 256
- **Problem:** Can't capture fine-grained patterns
- **Result:** Wider but not smarter

Method 3: Resolution Scaling (Larger Images)

- 224x224 $\rightarrow$ 299x299 $\rightarrow$ 331x331
- **Problem:** Expensive computation, network not optimized for it
- **Result:** Slower with minimal gains

Common Mistake: Researchers scale ONE dimension arbitrarily!

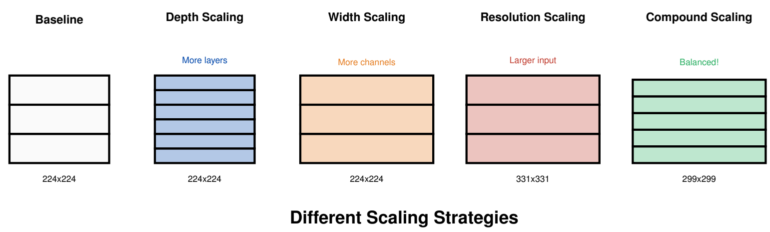


Figure 9: Scaling methods: EfficientNet uses balanced compound scaling

5.2 Compound Scaling: The Core Innovation

Compound Scaling Formula

Key Insight: Scale depth, width, and resolution together in a balanced way!
Scaling Formula:

$$\begin{aligned}\text{depth:} \quad & d = \alpha^\phi \\ \text{width:} \quad & w = \beta^\phi \\ \text{resolution:} \quad & r = \gamma^\phi\end{aligned}$$

Constraint:

$$\boxed{\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2}$$

where:

- ϕ = compound coefficient (user-set, controls overall scaling)
- α, β, γ = scaling coefficients for depth, width, resolution
- Constraint ensures: doubling ϕ approximately doubles total FLOPs

EfficientNet's Optimal Values (found via grid search on B0):

- $\alpha = 1.2$ (depth scaling)
- $\beta = 1.1$ (width scaling)
- $\gamma = 1.15$ (resolution scaling)
- Check: $1.2 \times 1.1^2 \times 1.15^2 \approx 2$ ✓

How it works:

- Set ϕ based on your compute budget
- EfficientNet-B0: $\phi = 0$ (baseline)
- EfficientNet-B1: $\phi = 1$ (2x FLOPs)
- EfficientNet-B7: $\phi = 7$ (128x FLOPs)
- Each dimension scales automatically according to formula!

Analogy

The Construction Analogy:
Building a skyscraper (neural network):
Bad Approach (single-dimension scaling):

- Only make it taller (depth): unstable, needs strong foundation
- Only make it wider (width): wastes space, not more functional
- Only make rooms bigger (resolution): doesn't add capacity

Compound Scaling (balanced):

- Taller building: add more floors (depth)
- Wider footprint: more rooms per floor (width)
- Larger rooms: bigger units for residents (resolution)
- All scale proportionally → structurally sound and functional!

Result: Balanced scaling is like proper architectural planning — everything works together!

5.3 EfficientNet Architecture: MBConv Blocks

MBConv (Mobile Inverted Bottleneck Convolution)

Building Block: EfficientNet uses MBConv blocks (from MobileNetV2)
MBConv Structure:

1. **Expansion:** 1x1 conv to expand channels (typically 6x)
2. **Depthwise Conv:** 3x3 or 5x5 depthwise separable conv
3. **Squeeze-and-Excitation (SE):** Channel attention mechanism
4. **Projection:** 1x1 conv to reduce back to original channels
5. **Skip Connection:** Add input if dimensions match

Key Features:

- **Inverted bottleneck:** Expand $\rightarrow$ process $\rightarrow$ compress
- **Depthwise separable:** Much more efficient than standard conv
- **SE module:** Learns which channels are important
- **Swish activation:** Swish$(x) = x \cdot \sigma(x)$ (better than ReLU)

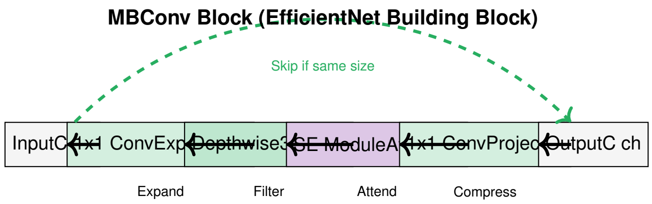


Figure 10: MBConv block: inverted bottleneck with SE attention

5.4 EfficientNet-B0 Baseline Architecture

Architecture Details

EfficientNet-B0: The baseline model designed via Neural Architecture Search (NAS)
Architecture Stages:

Stage	Operator	Resolution	Channels	Layers	Notes
1	Conv 3x3	224x224	32	1	Initial stem
2	MBConv1 (k3x3)	112x112	16	1	Expansion=1
3	MBConv6 (k3x3)	112x112	24	2	Expansion=6
4	MBConv6 (k5x5)	56x56	40	2	Larger kernel
5	MBConv6 (k3x3)	28x28	80	3	-
6	MBConv6 (k5x5)	14x14	112	3	-
7	MBConv6 (k5x5)	14x14	192	4	-
8	MBConv6 (k3x3)	7x7	320	1	Final block
9	Conv 1x1, Pool, FC	7x7	1280	1	Head

Parameters: 5.3 million (B0)
FLOPs: 0.39 billion (B0)

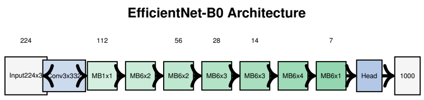


Figure 11: EfficientNet-B0: baseline model with 7 MBConv stages

5.5 EfficientNet Family: B0 to B7

EfficientNet Variants

Model	Input Size	Params	FLOPs	Top-1 Acc	ϕ
B0	224x224	5.3M	0.39B	77.1%	0
B1	240x240	7.8M	0.70B	79.1%	1
B2	260x260	9.2M	1.0B	80.1%	2
B3	300x300	12M	1.8B	81.6%	3
B4	380x380	19M	4.2B	82.9%	4
B5	456x456	30M	9.9B	83.6%	5
B6	528x528	43M	19B	84.0%	6
B7	600x600	66M	37B	84.3%	7

Key Observations:

- Each step up: approximately 2x FLOPs
- Accuracy increases consistently
- B0-B3: Good for mobile/edge devices
- B4-B7: High accuracy for server deployment
- All use same architecture, just scaled!

Quick Reference

EfficientNet Quick Reference:
Core Innovation:

- **Compound scaling:** Balance depth, width, resolution
- Formula: $d = \alpha^\phi, w = \beta^\phi, r = \gamma^\phi$
- Constraint: $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$

Architecture:

- MBConv blocks (inverted bottleneck)
- Squeeze-and-Excitation attention
- Swish activation
- Designed via Neural Architecture Search

When to Use:

- B0-B1: Mobile/embedded devices
- B2-B3: Standard deployment (best efficiency)
- B4-B5: Need high accuracy
- B6-B7: Maximum accuracy (competition/research)

Advantages:

- Best parameter efficiency
- State-of-the-art accuracy
- Easy to scale (just change ϕ)
- Transfer learning friendly

Training Tips:

- RMSProp optimizer (decay 0.9, momentum 0.9)
- Exponential Moving Average (EMA) for weights
- Batch norm momentum: 0.99
- Weight decay: 1e-5
- Dropout: 0.2 (increases with model size)

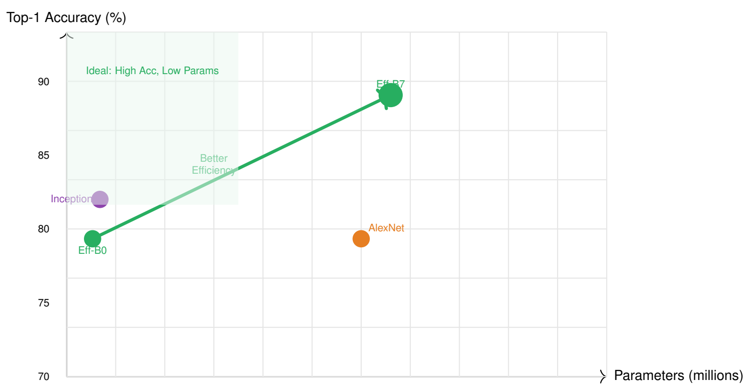
6 Architecture Comparison and Evolution

Complete Comparison Table

Aspect	AlexNet	VGG	InceptionNet	EfficientNet
Year	2012	2014	2014	2019
Depth	8 layers	16-19 layers	22 layers	Varies (B0: 7 stages)
Parameters	60M	138M (VGG16)	6.8M	5.3M (B0) - 66M (B7)
Top-5 Error	16.4%	7.3%	6.7%	2.3% (B7)
Key Innovation	ReLU, Dropout, GPU training	Very deep, simple blocks	Parallel filters, 1x1 conv	Compound scaling
Architecture	Sequential conv + FC	Repeated 3x3 conv blocks	Inception modules	MBConv blocks
Efficiency	Low	Very low	High	Very high
Compute (FLOPs)	1.5B	15.5B	1.5B	0.39B (B0) - 37B (B7)
Main Strength	First to work at scale	Very deep	Multi-scale features	Best accuracy/efficiency
Main Weakness	Many parameters	Massive parameters	Complex architecture	Needs careful tuning
Use Case	Historical importance	Feature extraction	Multi-scale tasks	Production deployment

Evolution Timeline:

- **2012 (AlexNet):** Proof that deep learning works
- **2014 (VGG):** Showed deeper is better (but expensive)
- **2014 (InceptionNet):** Efficiency through clever architecture
- **2015 (ResNet):** Very deep networks with skip connections
- **2017 (MobileNet):** Efficiency for mobile devices
- **2019 (EfficientNet):** Systematic scaling methodology
- **2020+ (Transformers):** Vision Transformers, new paradigm



Accuracy vs Parameters (Lower-left is better!)

Figure 12: Architecture comparison: EfficientNet achieves best accuracy/parameter ratio

Quick Reference

Which Architecture to Use?
For Transfer Learning (Most Common):

- **Limited compute:** EfficientNet-B0 or MobileNetV3
- **Standard tasks:** EfficientNet-B3 or ResNet-50
- **Need best accuracy:** EfficientNet-B7 or ViT-Large
- **Multi-scale objects:** InceptionV3 or FPN

For Training from Scratch:

- **Small dataset (<10K):** Don't! Use transfer learning
- **Medium dataset (10K-100K):** ResNet-34, EfficientNet-B0
- **Large dataset (100K+):** ResNet-50, EfficientNet-B3
- **Huge dataset (ImageNet-scale):** Any modern architecture

For Production Deployment:

- **Mobile/Edge:** MobileNetV3, EfficientNet-B0
- **Server (latency-critical):** EfficientNet-B1/B2
- **Server (accuracy-critical):** EfficientNet-B4+, ResNet-101
- **Batch processing:** Any architecture (latency doesn't matter)

Quick Decision Tree:

```
Do you have <1K labeled images?
  YES -> Use transfer learning (EfficientNet-B0)
  NO  -> Continue

Is inference speed critical?
  YES -> Mobile device?
           YES -> MobileNetV3
           NO  -> EfficientNet-B1
  NO  -> Continue

Need absolute best accuracy?
  YES -> EfficientNet-B7 or ViT
  NO  -> EfficientNet-B3 (best balance)
```

7 Summary and Key Takeaways

Week 9 Summary

ImageNet Revolution

- ImageNet (2012) proved deep learning works at scale
- AlexNet's 16.4% error (vs 25.8% traditional) sparked revolution
- Pre-trained ImageNet models enable transfer learning
- 99% of computer vision uses ImageNet-pretrained weights

Transfer Learning

- Early layers: generic features (edges, textures)
- Late layers: task-specific features
- Three strategies: freeze all, fine-tune top, fine-tune all
- Use based on dataset size and similarity to ImageNet

AlexNet (2012)

- First deep CNN to win ImageNet
- Innovations: ReLU, dropout, data augmentation, GPU training
- 8 layers, 60M parameters
- Legacy: Every modern technique traces back to AlexNet

InceptionNet (2014)

- Parallel filters (1x1, 3x3, 5x5, pooling)
- 1x1 conv for dimensionality reduction (90% savings!)
- 22 layers, only 6.8M parameters
- Auxiliary classifiers help gradient flow
- Error: 6.7% (half of AlexNet!)

EfficientNet (2019)

- Compound scaling: balance depth, width, resolution
- Formula: $d = \alpha^\phi, w = \beta^\phi, r = \gamma^\phi$
- MBConv blocks with SE attention
- B0: 5.3M params, B7: 66M params
- State-of-the-art accuracy with best efficiency

Practical Recommendations

- **Default choice:** EfficientNet-B3 with transfer learning
- **Mobile/edge:** EfficientNet-B0 or MobileNetV3
- **Maximum accuracy:** EfficientNet-B7
- **Multi-scale objects:** InceptionV3
- **Always use pre-trained weights!** (ImageNet)

Evolution of Ideas

1. **AlexNet:** Proved deep learning works

End of Week 9 Notes

Deep Learning for Perception — FAST-NUCES