The Secret Blueprints of AI Vision

Evolving CNNs: From Grayscale MNIST to Deep ImageNet, a Comparative Study of Architectures

Minor In AI, IIT Ropar 5th June, 2025

Welcome, budding AI enthusiasts!

You've taken your first steps into the fascinating world of Artificial Intelligence, and perhaps you've started exploring Convolutional Neural Networks (CNNs). These powerful tools are the backbone of many modern image and video applications, from recognizing faces on your phone to self-driving cars.

But building an effective CNN isn't always straightforward. You might have already experimented with adding layers, changing settings, and seen that sometimes, despite your best efforts, the model doesn't improve, or even gets worse! It can feel a bit like fumbling in the dark.

So, how do the experts do it? Do they just randomly try things until something works? While experimentation is definitely a part of the process, successful AI practitioners often rely on something more structured: **architectures**.

The Blueprint: Learning from Experience

Imagine a time long ago, when our ancestors lived in the open. There were threats everywhere – wild animals, harsh weather. To protect themselves, they started seeking shelter. Perhaps first in trees, then building simple shelters with just four walls.

As time passed, people learned more about living comfortably and efficiently. They realized that having a separate space for cooking kept the rest of the house clean, and a separate space for sleeping provided privacy and rest. They started designing homes with different rooms: a bedroom, a kitchen, a living area.

Figure 1: The evolution of shelter: From open forests to structured homes with separate rooms, demonstrating the concept of architecture.

This design, this blueprint of how the different parts of the house are arranged and function together, is the **architecture** of the home.

And guess what? Once someone figured out that having separate rooms made life better, others started copying that successful design. They didn't have to reinvent the idea of a kitchen every time they built a house! Of course, the architecture evolved further – maybe an office space needs a different layout than a home, with a large open area and smaller meeting rooms.

This same idea applies directly to building AI models, particularly CNNs. People have spent years experimenting, trying different combinations of layers, settings, and techniques. When they discover a configuration that works really well for a specific type of problem, that configuration becomes a well-known **architecture** – a blueprint that others can learn from and use as a starting point.

Why Architectures Matter in AI

Think back to building a simple CNN. You combine Convolutional layers, Pooling layers, Activation functions, and Dense layers. But how many layers? What size filters? What kind of pooling? Should you add something else?

Randomly changing these pieces can lead to issues:

1. **Overfitting:** The model becomes too good at recognizing only the specific images it was trained on, failing on new, unseen images.
2. **Training Time:** Adding more layers significantly increases the time and computational power needed to train the model.
3. **Suboptimal Performance:** Your carefully crafted layers might not extract the best features or learn the most effective patterns from the data.

This is where established architectures come in. They represent years of research and experimentation, offering proven ways to combine layers and techniques to achieve high performance on challenging tasks. Instead of starting from scratch, you can often use or adapt these existing blueprints.

Beyond the Basics: New Tools for Better Training

As researchers built more complex CNNs, they also developed new techniques to handle the challenges that arose. Two important ones are:

- **Dropout:** Remember the Dense or Fully Connected layers where every neuron in one layer is connected to every neuron in the next? This can contribute to overfitting. Dropout is a technique where, during training, we randomly "drop out" (temporarily ignore) a percentage of these connections. By randomly removing connections, the network can't become overly reliant on any single connection or small group of neurons. This forces the network to learn more robust features and helps prevent overfitting. If you don't specify the percentage, it often defaults to 50% of the connections.
- **Batch Normalization:** As data flows through many layers in a deep network, the distribution of the outputs of each layer can change dramatically. This can slow down training. Batch Normalization is a technique applied *within* the network layers to keep the outputs of each layer within a stable distribution (often normalized to have a mean close to 0 and a standard deviation close to 1). Normalizing the outputs of layers makes the training process more stable and significantly faster, allowing us to train deeper networks effectively.

These techniques, along with others like data augmentation, became crucial components in the evolution of CNN architectures.

The Pioneers: A Comparative Study of Architectures

Now, let's look at three landmark CNN architectures that showcase this evolution and serve as foundational blueprints: LeNet, AlexNet, and VGG16.

The following table provides a detailed breakdown of their key differences.

Table 1: Summary of Landmark CNN Architectures

Feature	LeNet-5	AlexNet	VGG-16
Year	1998	2012	2014
Primary Dataset	MNIST (Handwritten Digits)	ImageNet (1000 Object Classes)	ImageNet (1000 Object Classes)
Input Size	32x32 pixels	224x224 pixels	224x224 pixels
Color Channels	Grayscale (1)	RGB (3)	RGB (3)
Conv Layers	2	5	13
Kernel Sizes	5x5	Varied (11x11, 5x5, 3x3)	Uniform (3x3 only)
Activation	Tanh	ReLU	ReLU
Pooling	Average Pooling	Max Pooling	Max Pooling
Dropout	×	✓	✓
Batch Norm	×	× (Original)	× (Original)
Key Contribution	First practical, successful CNN. Proved the concept.	Proved deep networks on GPUs work. Popularized ReLU & Dropout.	Showed extreme depth with small, uniform filters is effective.
Limitation	Only for small, simple grayscale images.	Computationally expensive. Memory intensive.	Extremely slow to train. Very high parameter count.
Parameters	∼60 Thousand	∼60 Million	∼138 Million

1. LeNet (1998): The Grandparent of CNNs

- **Built For:** Grayscale images (like the MNIST dataset), typically small (32x32 pixels).
- **Structure:** Relatively simple and shallow, with only a few convolutional layers (typically 2).
- **Parameters:** Relatively few (around 60,000).

2. AlexNet (2012): The Breakthrough

- **Built For:** Large, colorful RGB images (like ImageNet), typically 224x224 pixels.
- **Structure:** Much deeper than LeNet, featuring 5 convolutional layers.
- **Novelty:** Demonstrated the power of *deep* CNNs. Introduced ReLU and Dropout. *Crucially*, it showed that training large models was possible using GPUs.
- **Parameters:** Substantially more than LeNet (around 60 million).

3. VGG16 (2014): Pushing the Boundaries of Depth

- **Built For:** Large, colorful RGB images (ImageNet), typically 224x224 pixels.
- **Structure:** Very deep (16 layers, hence VGG*16*), with 13 convolutional layers.
- **Novelty:** Demonstrated the effectiveness of increased depth with uniform small filters.
- **Parameters:** One of the largest models at the time (around 138 million).

Putting Architectures to Work: Using Pre-trained Models

One of the biggest advantages of these established architectures is that you can often take a pre-trained model and adapt it for your specific task – a technique called **transfer learning**. Let's look at how you might define a simple CNN structure versus using a pre-built one in code.

```
# This is a simplified example to show the structure
import torch.nn as nn

class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        self.features = nn.Sequential(
            # First Convolutional Block
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # Second and Third blocks would follow...
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 512), # Size depends on image dimensions
            nn.ReLU(inplace=True),
            nn.Linear(512, num_classes) # Output layer
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Listing 1: Example 1: Defining a Simple CNN (Layer by Layer)

```
# This is a simplified example showing how to load the model
import torchvision.models as models
import torch.nn as nn

# Load the AlexNet architecture with pre-trained weights from ImageNet
model = models.alexnet(pretrained=True)

# To adapt for a new task (e.g., 10 classes instead of 1000),
# you can replace the final layer of the classifier.
num_classes = 10
model.classifier[6] = nn.Linear(4096, num_classes)

# Now the model is ready for fine-tuning on your new dataset!
```

Listing 2: Example 2: Using a Pre-built Architecture (AlexNet)

The demonstration on the CIFAR-10 dataset highlighted the power of this. While the simple CNN struggled, the pre-trained AlexNet model, even in its first training *epoch*, achieved significantly higher accuracy (81%) compared to the simple CNN's starting accuracy (46%).

Choosing Your Blueprint

How do you decide which architecture to use?

- **Dataset Type:** Grayscale or color? LeNet is for grayscale, while AlexNet/VGG are for RGB.
- **Dataset Complexity:** Simple images might only need a shallow network. Complex scenes need deeper architectures.
- **Available Resources:** Do you have GPUs? Deeper models like AlexNet and VGG require them for reasonable training times.
- **Performance Needs:** Do you need state-of-the-art accuracy or is speed more critical?

Conclusion

Just as architects design blueprints for buildings, researchers have designed powerful architectures for CNNs. Understanding these architectures – their structure, strengths, limitations, and the key techniques they employ – is crucial. It allows you to:

1. Appreciate the evolution of the field.
2. Choose appropriate existing architectures for your tasks.
3. Leverage pre-trained models for faster development.
4. Potentially even design *your own* novel architectures.

This journey from simple, grayscale models to deep, colorful image recognition is a testament to the power of structured experimentation and building upon past successes.

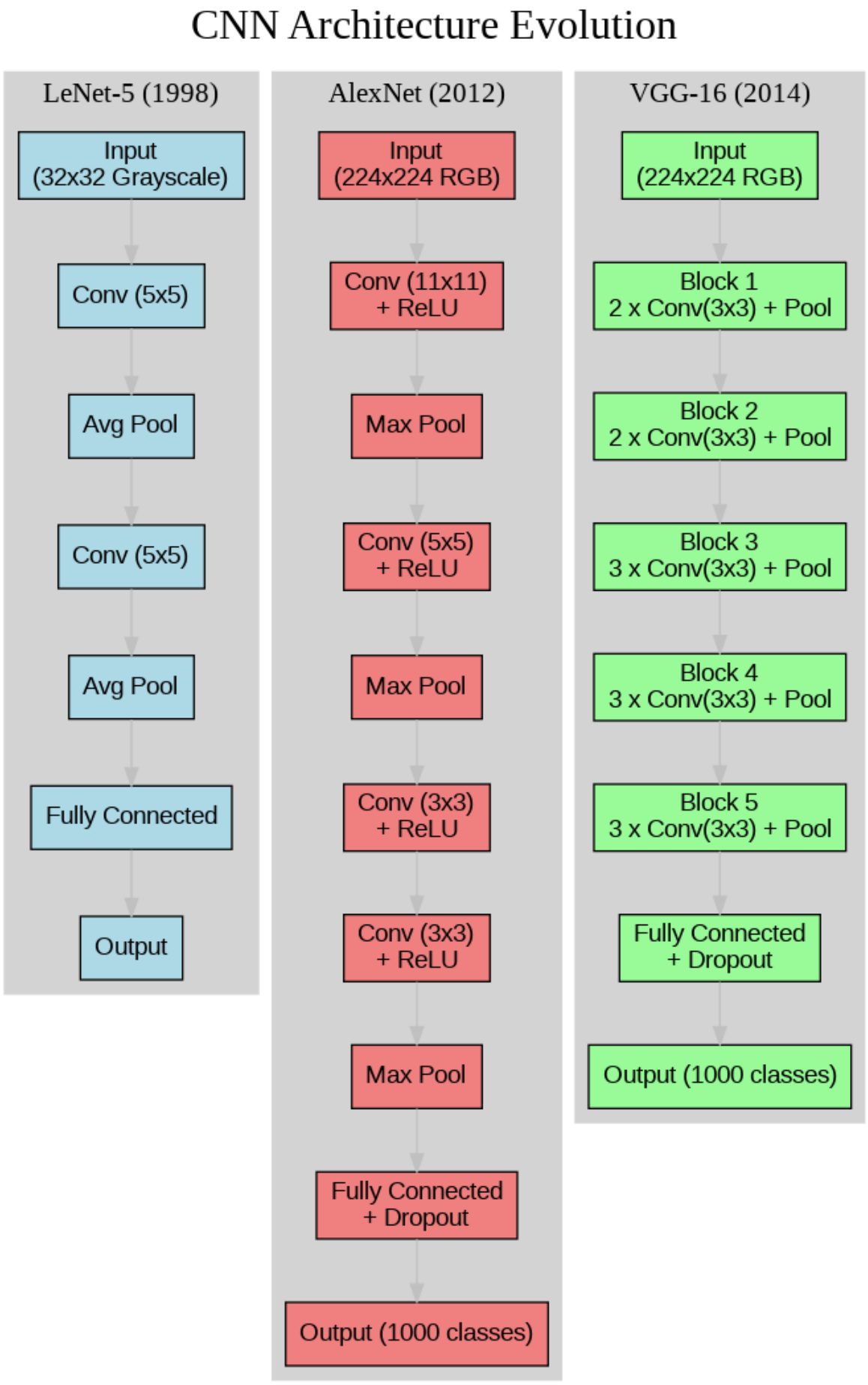

Figure 2: A visual comparison of the architectures: LeNet (shallow, for simple tasks), AlexNet (deeper, introducing modern techniques), and VGG16 (very deep, focused on uniform structure).

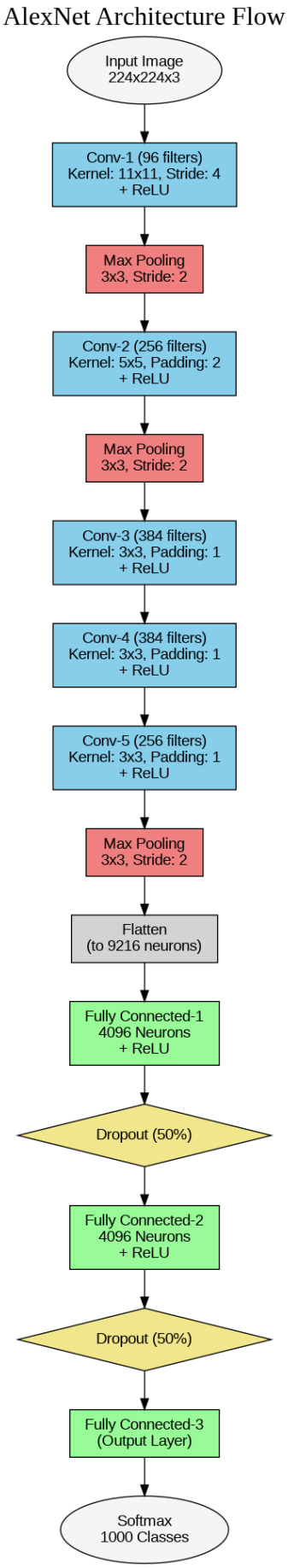

Figure 3: Simplified diagram of the AlexNet architecture, showing the flow from a large RGB input through multiple convolutional and pooling layers to the final classification output.