Understanding Paper Notes

1. The Core Motivation: ConvNets vs. Transformers (ConvNeXt)

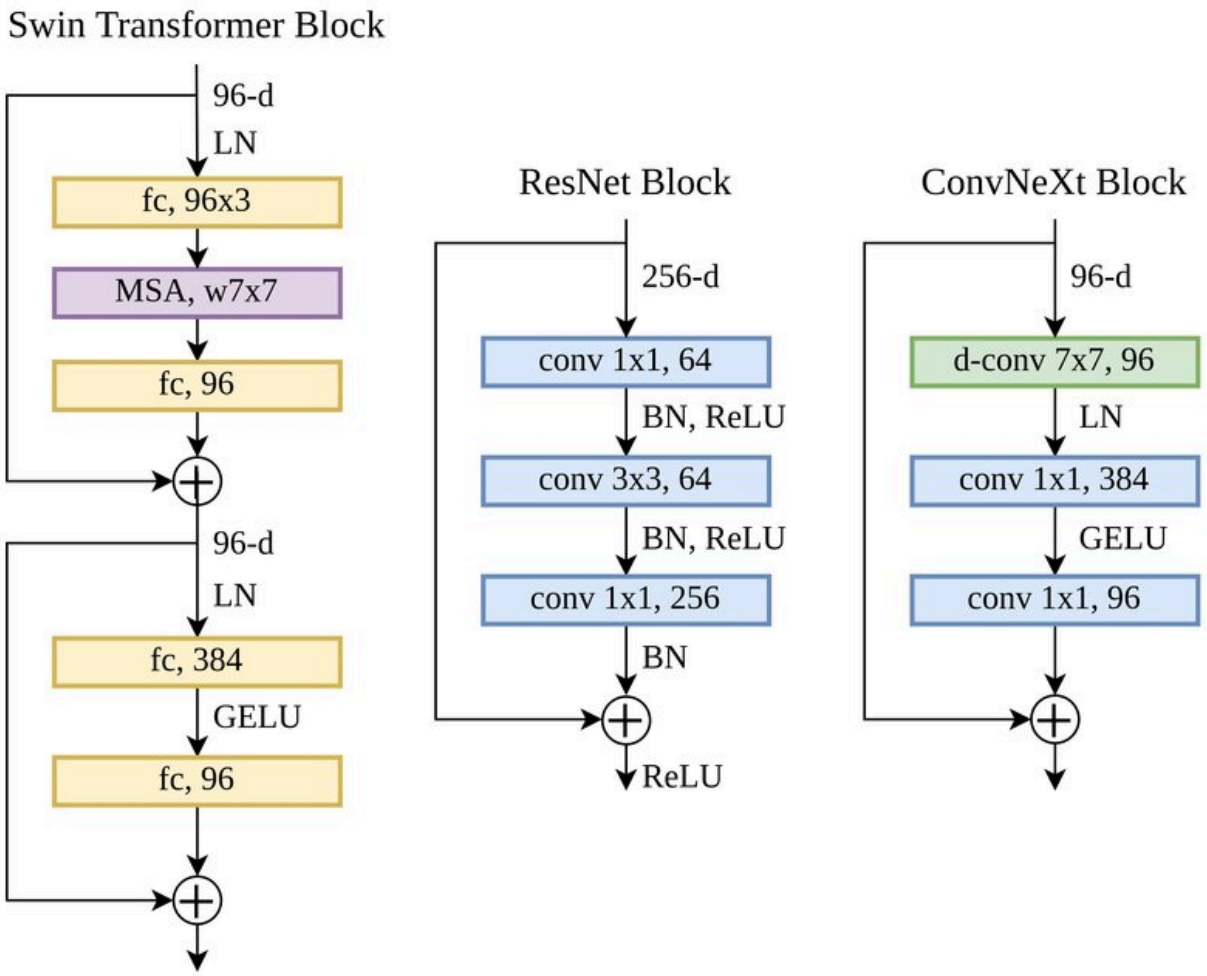
Summary

Researchers investigated whether the dominance of Vision Transformers (ViTs) was due to self-attention or simply modern design choices. They "modernized" a standard ResNet to mimic a Transformer's architecture, resulting in the pure ConvNet model known as ConvNeXt.

Key Points

- **Context:** By the 2020s, Vision Transformers (like Swin) had superseded ConvNets in accuracy and scalability.
- **The Gap:** Vanilla ViTs struggle with tasks like object detection without the inductive biases (like sliding windows) found in ConvNets.
- **Hypothesis:** The performance gap might be due to training techniques and macro/micro architecture designs rather than the attention mechanism itself.
- **Outcome:** ConvNeXt competes favorably with Transformers in accuracy and scalability while maintaining the simplicity and efficiency of standard ConvNets.

Example



This Python code demonstrates a "Mini-ConvNeXt" block that embodies the modernized design: using depthwise convolution, LayerNorm (channels-last), and inverted bottlenecks.

```
class ConvNeXtBlock(nn.Module):
    """
    ConvNeXt block:
      - Depthwise Conv (7x7) for spatial mixing
      - LayerNorm (channels-last) instead of BatchNorm
      - Pointwise 1x1 (expand 4x) -> GELU -> Pointwise 1x1 (project)
    """
    def __init__(self, dim, kernel_size=7, mlp_ratio=4.0):
        super().__init__()
        # Depthwise convolution with large kernel
        self.dwconv = nn.Conv2d(dim, dim, kernel_size=kernel_size,
                                padding=kernel_size//2, groups=dim)

        # LayerNorm applied over channels
        self.ln = LayerNormChannelsLast(dim)

        # Inverted Bottleneck (Expand -> Act -> Project)
        hidden_dim = int(dim * mlp_ratio)
        self.pwconv1 = nn.Conv2d(dim, hidden_dim, kernel_size=1)
        self.act = nn.GELU()
        self.pwconv2 = nn.Conv2d(hidden_dim, dim, kernel_size=1)
```

```
    def forward(self, x):
        residual = x
        x = self.dwconv(x)
        x = self.ln(x)
        x = self.pwconv1(x)
        x = self.act(x)
        x = self.pwconv2(x)
        return x + residual
```

2. The Baseline: Improved Training Techniques

Summary

Before modifying the architecture, researchers significantly boosted ResNet-50 performance by adopting modern training recipes typically used for Vision Transformers. This demonstrated that much of the performance gap between ConvNets and Transformers was due to outdated training methodologies rather than architectural deficiencies.

Key Points

- **Extended Training:** Training duration was increased from 90 to 300 epochs.
- **Modern Optimizer:** Switched from SGD to AdamW.
- **Data Augmentation:** Implemented Mixup, Cutmix, RandAugment, and Random Erasing.
- **Regularization:** Applied Stochastic Depth and Label Smoothing.
- **Result:** Accuracy improved from 76.1% to 78.8% without any architectural changes.

3. Macro Design: Changing the "Skeleton"

Summary

The authors adjusted the macroscopic structure of the ResNet to align with the hierarchical design of the Swin Transformer. This involved changing the distribution of computation across stages and modifying the input stem to mimic the non-overlapping patch embedding of Transformers.

Key Points

- **Stage Compute Ratio:** Adjusted the number of blocks in each stage from (3, 4, 6, 3) to (3, 3, 9, 3) to match Swin-T's heavy third stage.
- **"Patchify" Stem:** Replaced the traditional 7x7 convolution and MaxPool with a 4x4 convolution with stride 4.

- **Benefit:** These changes simplified the design and aligned the ConvNet's feature map resolution and processing stages with modern Transformers.

```
class PatchifyStem(nn.Module):
  """
  Replaces standard 7x7 conv + maxpool.
  Splits image into 4x4 non-overlapping patches.
  """
  def __init__(self, in_ch=3, out_ch=96, patch_size=4):
      super().__init__()
      self.conv = nn.Conv2d(in_ch, out_ch, kernel_size=patch_size,
stride=patch_size)

  def forward(self, x):
      return self.conv(x)
```

4. The "ResNeXt-ification" (Depthwise Convolution)

Summary

Researchers adopted depthwise convolutions (grouped convolutions) to separate spatial mixing from channel mixing, a strategy that mirrors the mechanism of self-attention in Transformers while improving efficiency.

Key Points

- **Depthwise Convolution:** A special case of grouped convolution where the number of groups equals the number of channels.
- **Separation of Concerns:** This technique separates spatial mixing (mixing information within feature maps) from channel mixing (mixing between feature maps using 1×1 convs).
- **Transformer Similarity:** This separation is mathematically similar to the self-attention mechanism in Vision Transformers, where weighted sums mix spatial information per channel1.
- **Network Width:** Depthwise convolutions significantly reduce FLOPs, allowing the network width (number of channels) to be increased from 64 to 96 to match the capacity of Swin-T2.

Example

In PyTorch, "ResNeXt-ification" is achieved by setting the `groups` argument equal to the number of input channels (`dim`).

```
import torch.nn as nn

# Standard Convolution ( Spatial + Channel mixing)
standard_conv = nn.Conv2d(in_channels=96, out_channels=96, kernel_size=7)

# Depthwise Convolution (Spatial mixing only)
# groups=96 means each channel is processed independently
depthwise_conv = nn.Conv2d(in_channels=96, out_channels=96, kernel_size=7,
groups=96)
```

5. The Inverted Bottleneck

Summary

The authors adopted an "inverted bottleneck" design, where the internal hidden dimension is wider than the input, aligning the ConvNet structure with the MLP blocks found in Transformers and efficient networks like MobileNetV2.

Key Points

- **Shape Inversion:** Standard ResNet blocks use a "Wide $\rightarrow$ Narrow $\rightarrow$ Wide" structure. ConvNeXt uses "Narrow $\rightarrow$ Wide $\rightarrow$ Narrow".
- **Expansion Ratio:** The hidden dimension is expanded by a factor of 4 (e.g., $96 \rightarrow 384 \rightarrow 96$), mirroring the expansion ratio in Transformer MLP blocks.
- **Efficiency:** Despite the wider internal dimension, the use of depthwise convolutions keeps the FLOPs manageable, and the 1×1 shortcuts become cheaper.
- **Performance:** This design change slightly improved performance on ImageNet (80.5% to 80.6%) and provided larger gains for bigger models.

```
    def __init__(self, dim, mlp_ratio=4.0):
        super().__init__()
        hidden_dim = int(dim * mlp_ratio) # Expansion: Narrow -> Wide

        # 1. Pointwise conv (Expand)
        self.pwconv1 = nn.Conv2d(dim, hidden_dim, kernel_size=1)

        # 2. Depthwise conv (Spatial Mixing)
        self.dwconv = nn.Conv2d(hidden_dim, hidden_dim, kernel_size=7,
groups=hidden_dim)
```

```
        # 3. Pointwise conv (Project/Compress)
        self.pwconv2 = nn.Conv2d(hidden_dim, dim, kernel_size=1) #
Compression: Wide -> Narrow

    def forward(self, x):
        # ... (activations and norms omitted for clarity)
        return self.pwconv2(self.dwconv(self.pwconv1(x))) + x
```

6. Large Kernel Sizes (The "Global" Receptive Field)

Summary

To mimic the global receptive field of Vision Transformers, the authors rearranged the block structure and significantly increased the kernel size, finding that 7×7 kernels offered the best balance of performance and efficiency.

Key Points

- **Global Context:** Unlike standard ConvNets with small 3×3 kernels, Transformers process information globally (or in large windows). To compete, ConvNets need larger receptive fields.
- **Moving Up:** The depthwise convolution layer was moved to the top of the block (before the 1×1 layers). This mirrors the position of the Multi-Head Self-Attention (MSA) module in a Transformer block.
- **7x7 Kernels:** The kernel size was increased from 3×3 to 7×7. This captures larger context, similar to the 7×7 windows used in Swin Transformers.

---

7. Micro Design: Activation and Normalization

Summary

The researchers fine-tuned the layer-level details by adopting "Transformer-style" choices: simpler activation functions (GELU), fewer activations overall, and substituting Batch Normalization with Layer Normalization8.

Key Points

- **GELU Activation:** Replaced ReLU with GELU (Gaussian Error Linear Unit), a smoother activation function used in advanced Transformers like BERT and ViT.
- **Fewer Activations:** Instead of appending an activation to every convolutional layer, the block now contains only a single activation function placed between the two 1×1 layers.
- **Layer Normalization (LN):** Replaced Batch Normalization (BN) with Layer Normalization. While BN is standard for ConvNets, LN is the standard for Transformers and simplifies training dynamics.
- **Fewer Norms:** Reduced the number of normalization layers, retaining only one LN per block instead of one after every convolution.

---

8. Final Results: ConvNeXt Performance

Summary

The final ConvNeXt model, constructed entirely from standard ConvNet modules, competed favorably with state-of-the-art Transformers, achieving 87.8% accuracy on ImageNet and outperforming Swin Transformers on downstream tasks.

Key Points

- **ImageNet Accuracy:** ConvNeXt-XL achieved 87.8% top-1 accuracy on ImageNet-1K, demonstrating scalability comparable to large Transformers.
- **Throughput:** ConvNeXt exhibited higher inference throughput (images/second) than Swin Transformers, particularly on A100 GPUs, due to the efficiency of standard convolutions.
- **Object Detection:** On COCO object detection, ConvNeXt outperformed Swin Transformers (e.g., +1.0 AP for large models).
- **Segmentation:** Achieved competitive performance on ADE20K semantic segmentation, validating the effectiveness of the architecture beyond classification.