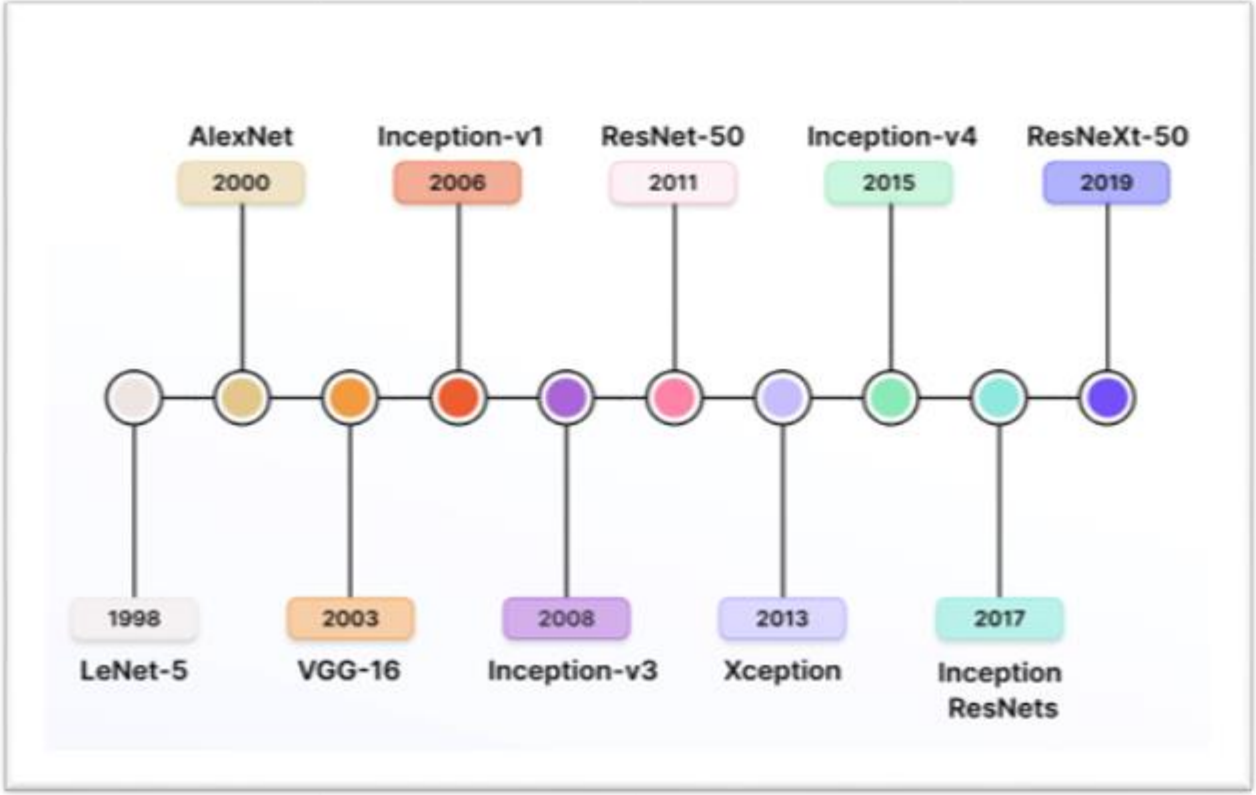
Redes Neuronales y Aprendizaje Profundo

Bloque2: Convolutional Neural Networks

Architectures



Convolutional Neural Networks

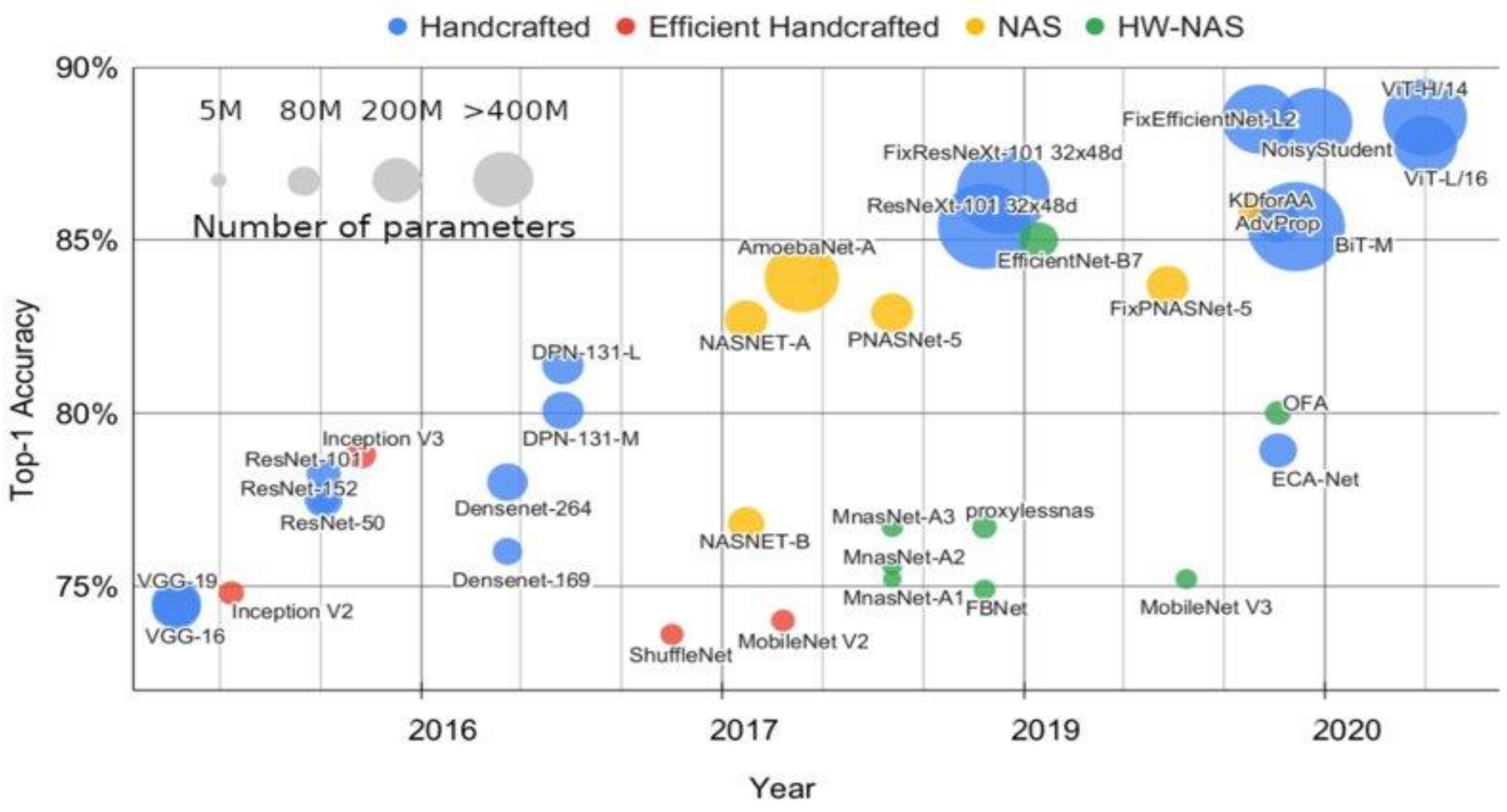
- **Objectives**
  - **Understand the historical evolution** of CNN architectures, from foundational models like LeNet to modern, efficient networks.
  - **Identify key structural components** (convolutional layers, pooling, fully connected layers) and their role in hierarchical feature extraction.
  - **Analyze the trade-offs** between model depth, computational cost, and accuracy across different architectures (AlexNet, VGG, ResNet).

Convolutional Neural Networks

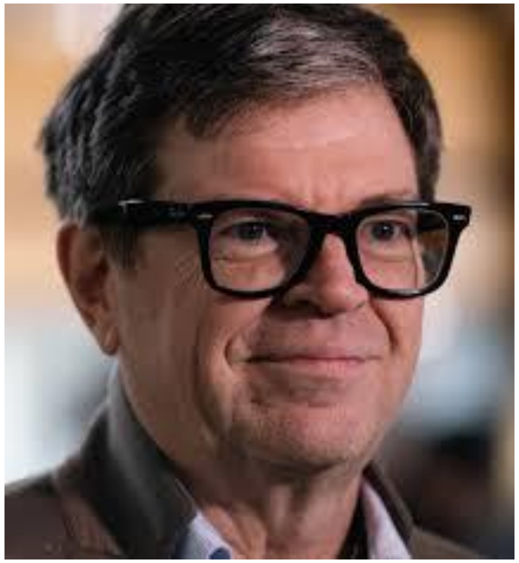
- **Objectives**
  - **Master the concept of Residual Learning** and how shortcut connections solve the vanishing gradient problem in deep networks.
  - **Evaluate efficiency-focused innovations** such as Depthwise Separable Convolutions in MobileNet and Compound Scaling in EfficientNet.
  - **Recognize the impact of Image Augmentation** as a vital technique for improving model generalization and reducing overfitting.

Contents

Evolution of CNNs



Introduction to LeNet



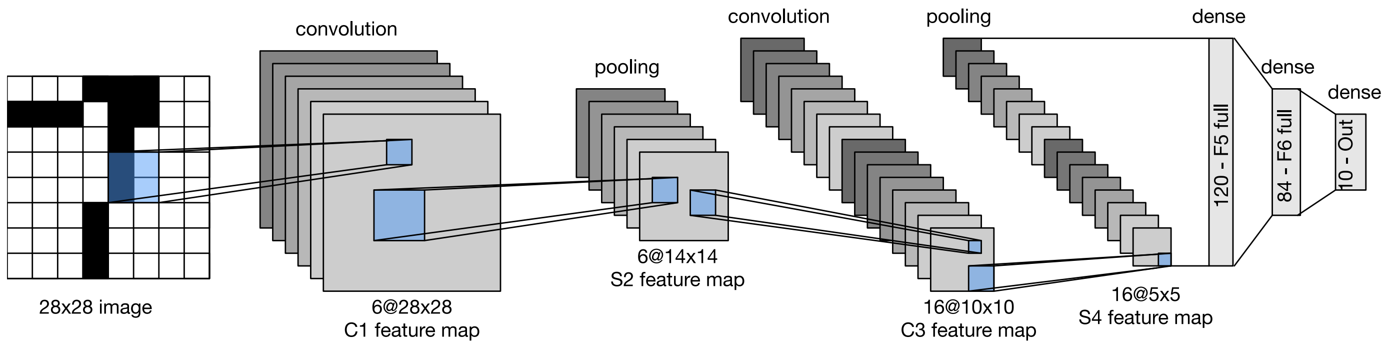
Born in **1960** in **France**, Yann LeCun is a world-renowned **computer scientist**, pioneer of Deep Learning, and the current Chief AI Scientist at Meta.

- Cited by 452731
- Around 400 to 500 publications

- **The LeNet Era (1989-1998):** While at Bell Labs, he developed **LeNet-5**, the first commercially successful CNN.
- **Biological Inspiration:** His work on CNNs was inspired by the biological visual cortex (how the eyes and brain process images).

Introduction to LeNet

- **LeNet Architecture**

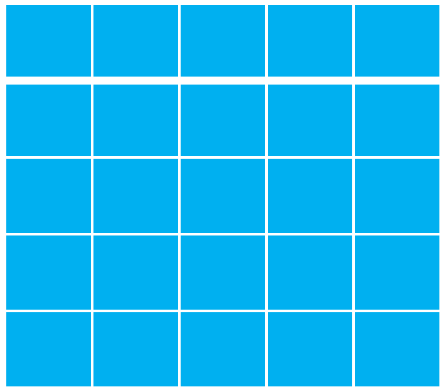


Input Layer -> Convolution Layer -> Pooling Layer -> Fully Connected Layer -> Fully Connected Layer -> Output Layer

Introduction to LeNet

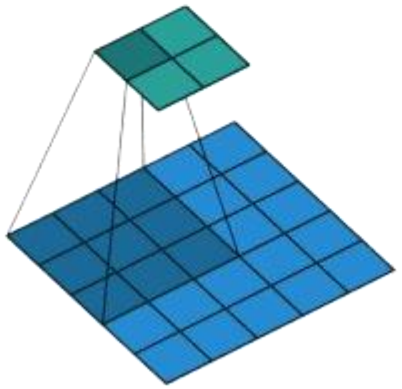
- **LeNet Architecture (Features)**

```
class LeNet(d2l.Classifier):  #@save
    """The LeNet-5 model."""
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.Sigmoid(),
            nn.LazyLinear(84), nn.Sigmoid(),
            nn.LazyLinear(num_classes))
```

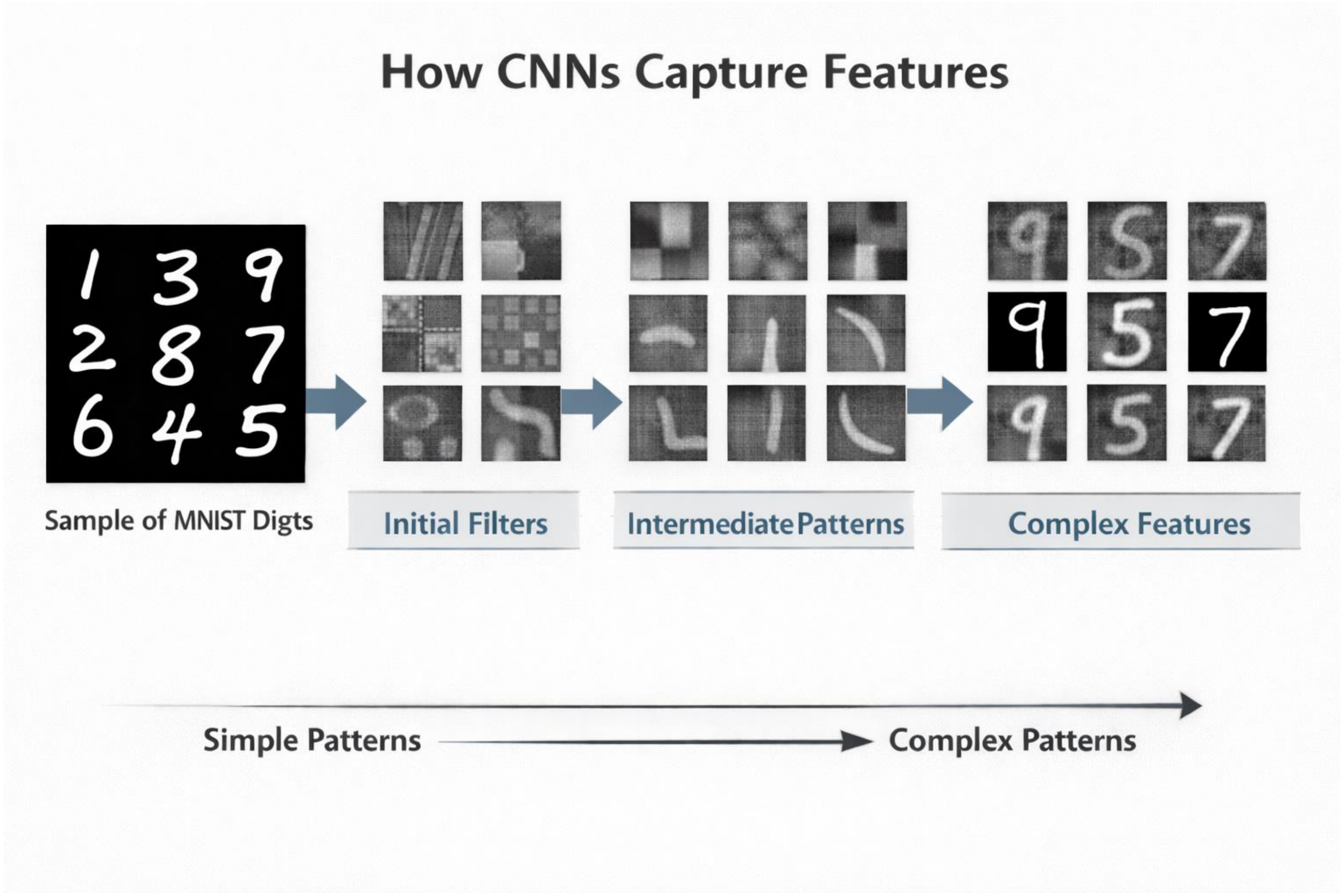


5x5

Stride = 2



Introduction to LeNet



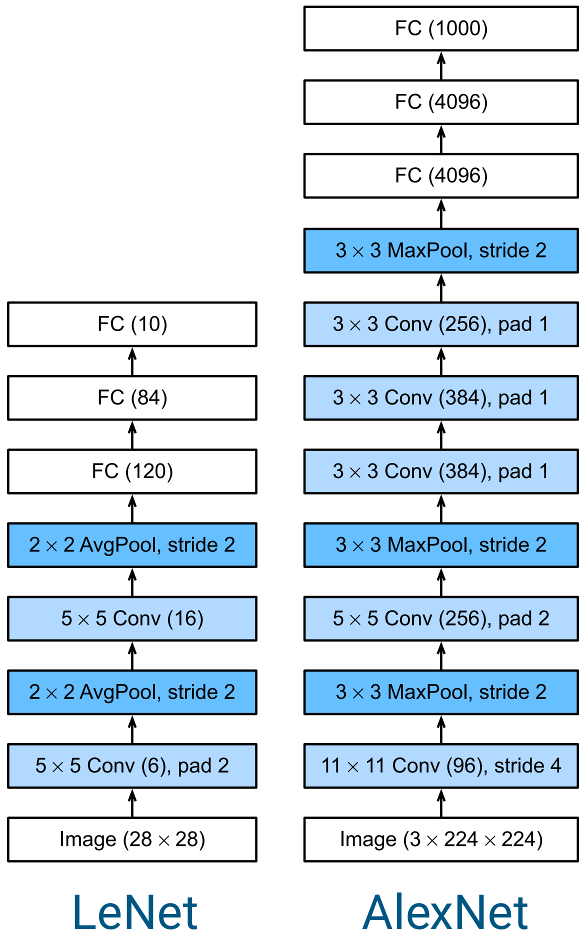
The Classics: AlexNet & VGG-16

- **AlexNet**
  - AlexNet represents a significant milestone in the development of convolutional neural networks and played a pivotal role in demonstrating the power of deep learning for image recognition tasks.
  - It was significantly larger and deeper than its predecessors like LeNet (with about 60M parameters). This increase in scale allowed AlexNet to capture more complex and abstract features from images, contributing to its superior performance.

The Classics: AlexNet & VGG-16

• AlexNet Architecture

```
class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(96, kernel_size=11, stride=4, padding=1),
            nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(256, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2), nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(p=0.5),
            nn.LazyLinear(4096), nn.ReLU(),nn.Dropout(p=0.5),
            nn.LazyLinear(num_classes))
        self.net.apply(d2l.init_cnn)
```

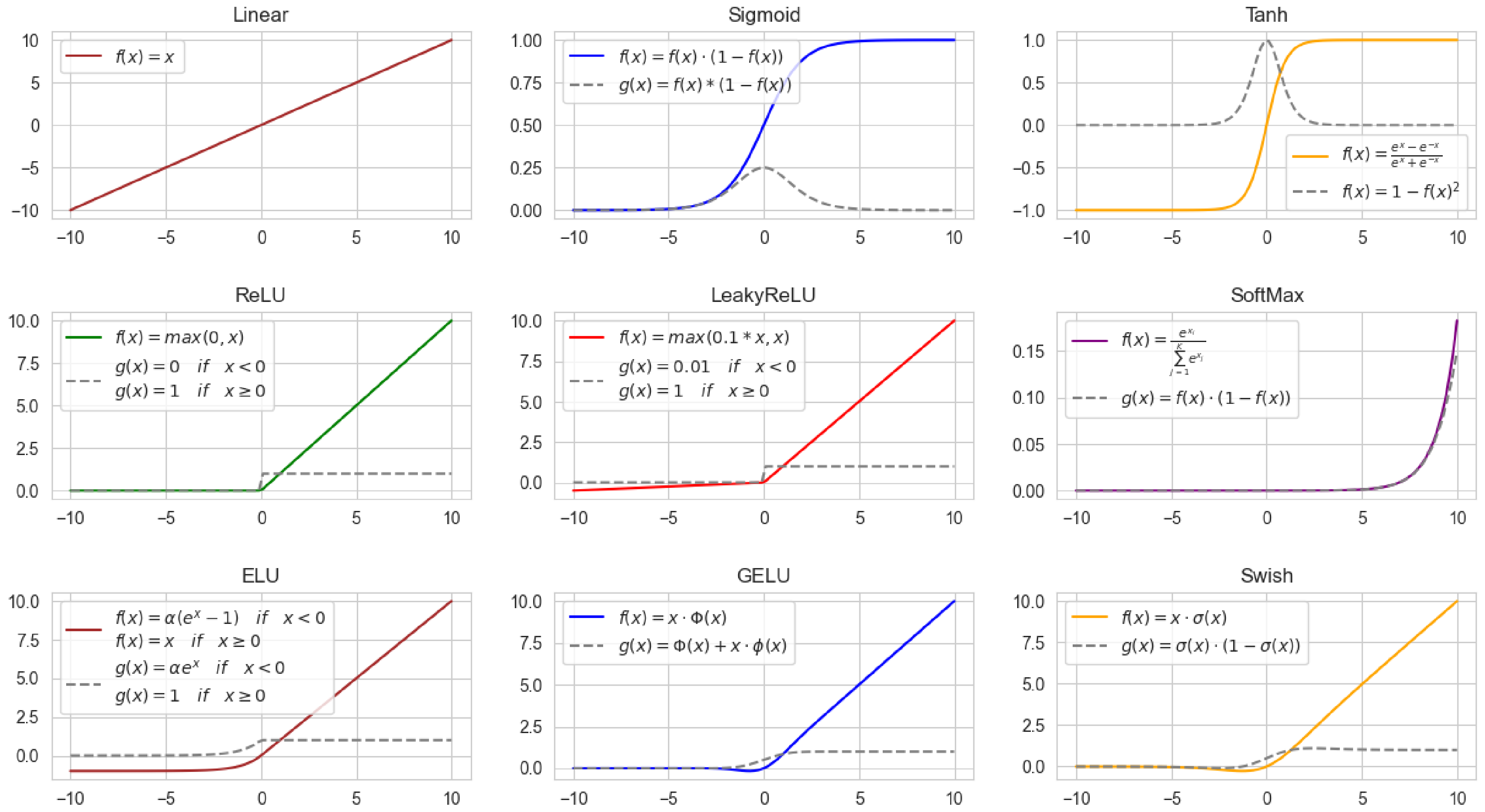


The Classics: AlexNet & VGG-16

- **AlexNet**
  - It was one of the first CNNs to successfully use ReLU activation functions instead of the sigmoid or tanh functions that were common at the time. ReLUs help to alleviate the vanishing gradient problem, allowing deeper networks to be trained more effectively.
  - It introduced overlapping pooling, where the pooling windows overlap with each other, as opposed to the non-overlapping pooling used in earlier architectures like LeNet. This was found to reduce overfitting and improve the network's performance.

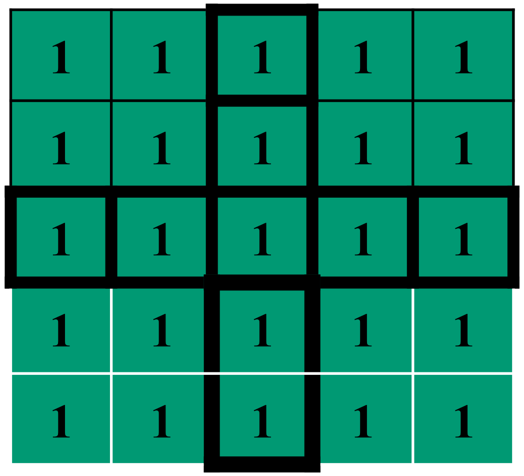
The Classics: AlexNet & VGG-16

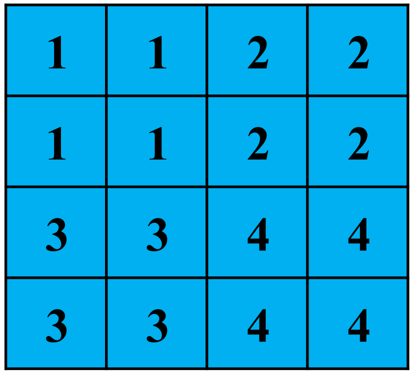
- **AlexNet (Features)**

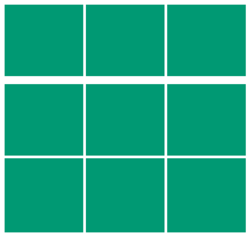


The Classics: AlexNet & VGG-16

- **AlexNet (Features)**
  - Overlapping
  - Non-overlapping

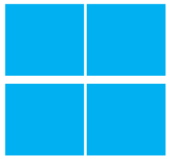






Kernel size = 3

Stride = 2

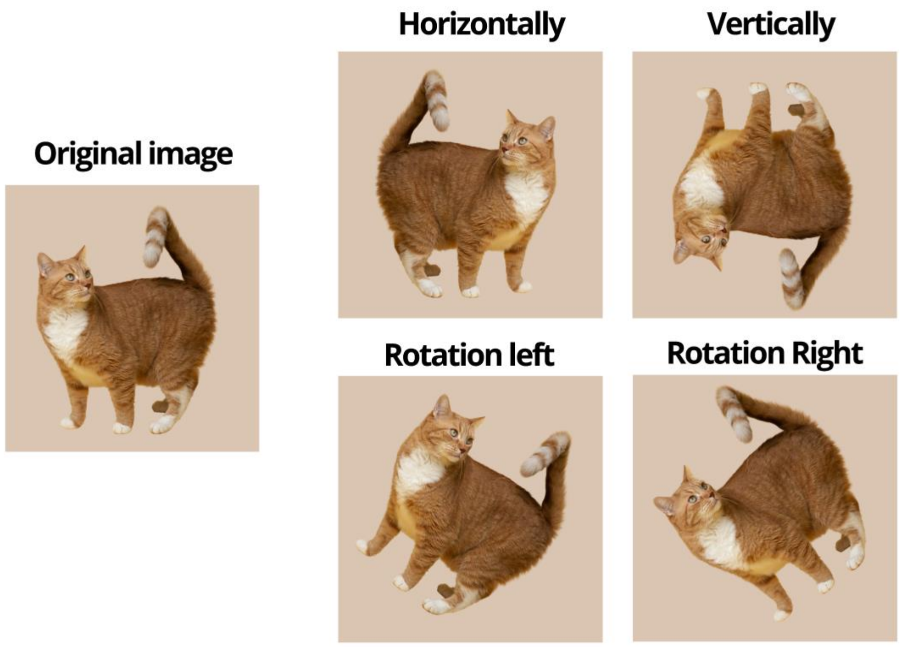


Kernel size = 2

Stride = 2

The Classics: AlexNet & VGG-16

- **AlexNet**
  - It introduced data augmentation techniques such as image translations, horizontal reflections, and alterations to the intensities of the RGB channels.



The Classics: AlexNet & VGG-16

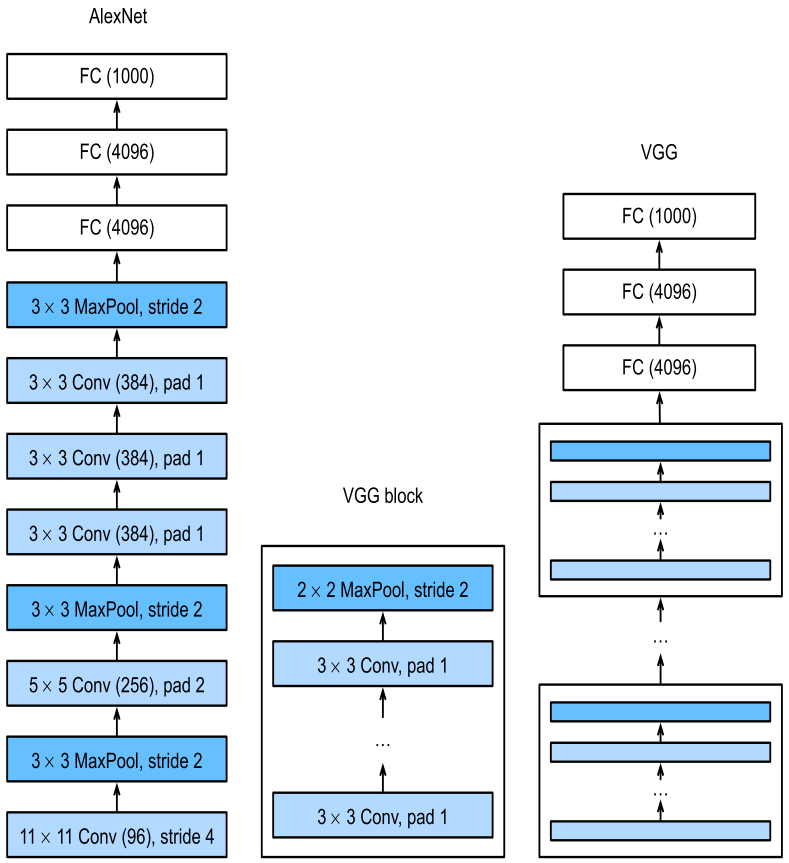
- **VGG-16**
  - VGG-16, developed by the Visual Graphics Group (VGG) at Oxford, is known for its simplicity and depth. It was a runner-up in the 2014 ImageNet competition.
  - The architecture's significant number of parameters (138M) makes it prone to overfitting, which is mitigated by using dropout and data augmentation techniques.
  - The model is characterized by its use of 3x3 convolutional layers stacked on top of each other in increasing depth.

The Classics: AlexNet & VGG-16

- **VGG-16 Architecture**

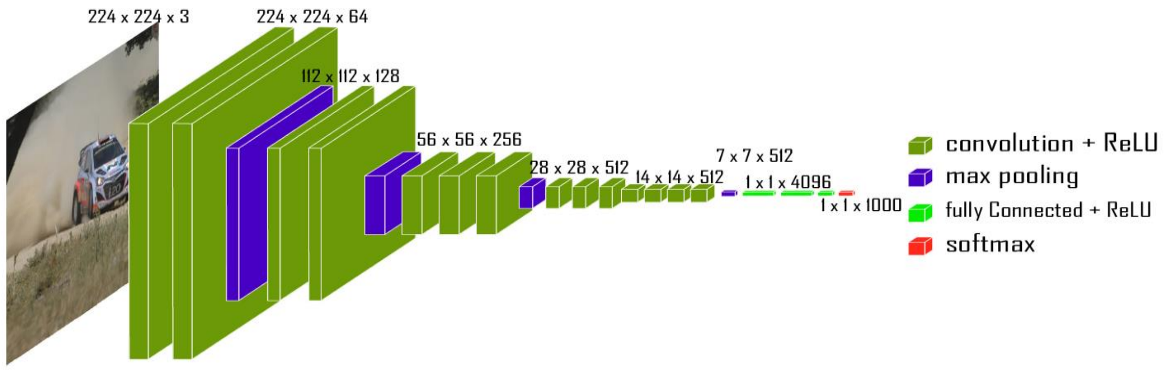
```
def vgg_block(num_convs, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.LazyConv2d(out_channels, kernel_size=3, padding=1))
        layers.append(nn.ReLU())
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)

class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        conv_blks = []
        for (num_convs, out_channels) in arch:
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.net = nn.Sequential(
            *conv_blks, nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
            nn.LazyLinear(num_classes))
        self.net.apply(d2l.init_cnn)
```

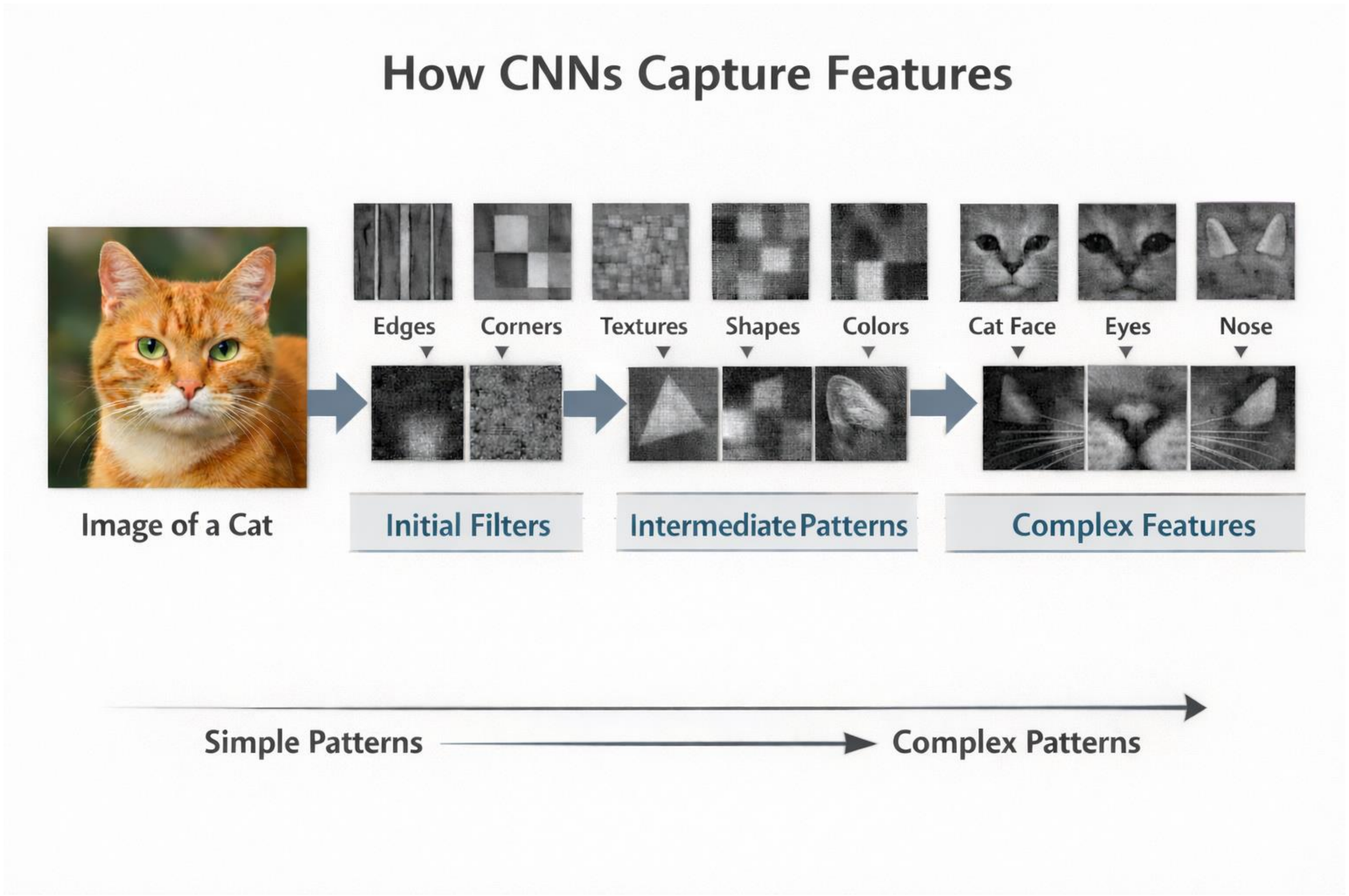


The Classics: AlexNet & VGG-16

- **VGG-16**
  - It has an uniform architecture consistent of using 3x3 convolutional filters and 2x2 max-pooling layers throughout the network.
  - It duplicate filters, starting at 64, doubles after each max-pooling layer, following the sequence 64, 128, 256, 512, 512.



The Classics: AlexNet & VGG-16



Modern Architectures: ResNet & Inception

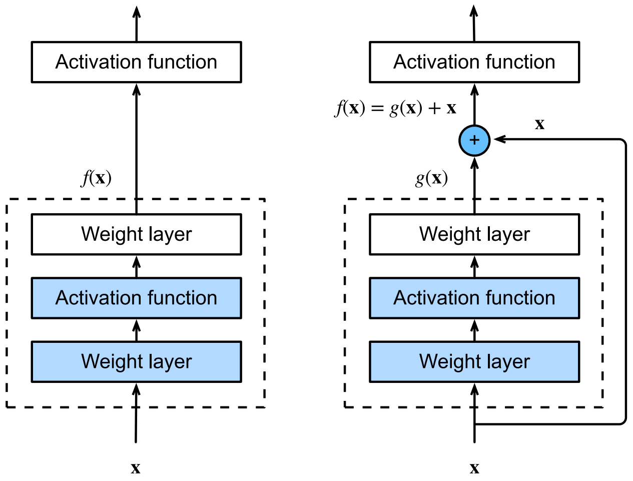
- **ResNet**
  - ResNet (Residual Network) was introduced by He et al. in 2015 and won the ImageNet competition by a significant margin.
  - It addresses the vanishing gradient problem in deep neural networks through the use of residual blocks, enabling the training of networks that are much deeper than previous architectures. This is by using shortcut connections that allow gradients to flow through the network more effectively. his approach allows for the construction of very deep networks (ResNet variants come in depths of 50, 101, 152 layers, and more).

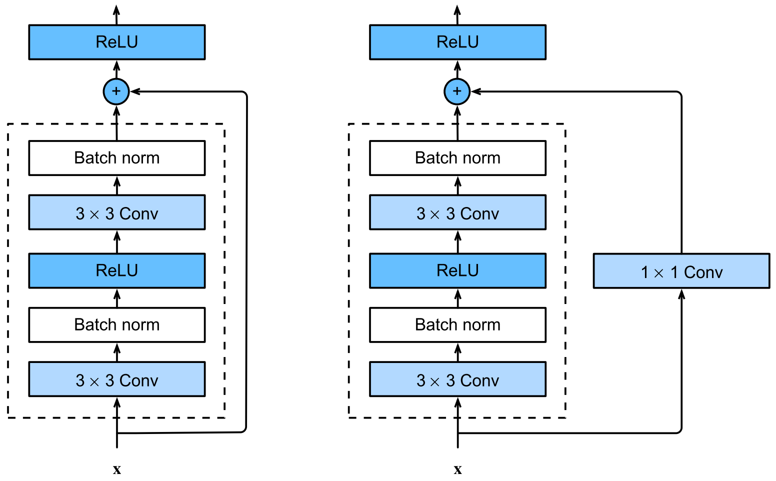
Modern Architectures: ResNet & Inception

- ## ResNet Architecture

```
class Residual(nn.Module):  #@save
    """The Residual block of ResNet models."""
    def __init__(self, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=strides)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return F.relu(Y)
```





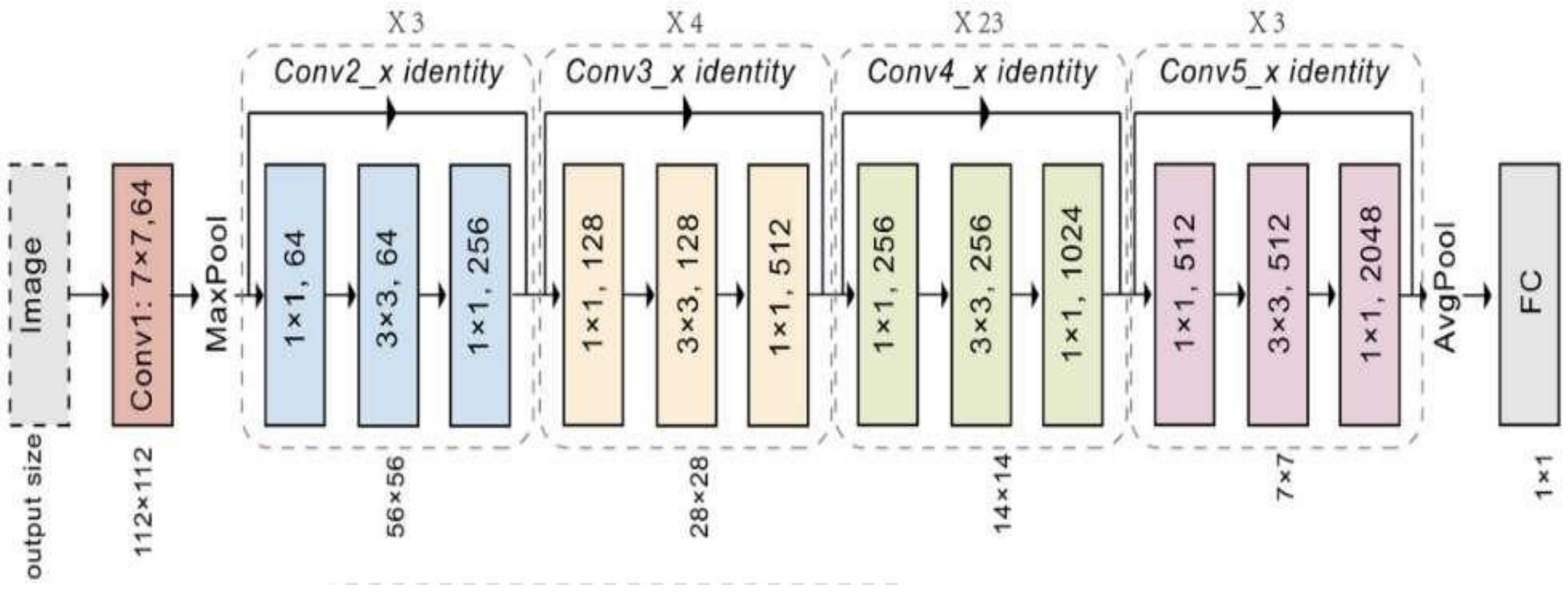
Modern Architectures: ResNet & Inception

- **ResNet**
  - A **residual block** allows the input to a series of layers to be added to their output, facilitating the learning process by allowing the network to learn modifications to the identity mapping rather than the entire transformation from scratch. It is composed by:
    - **Shortcut Connection** that skips one or more layers.
    - **Two or three convolutional layers**, each followed by batch normalization and a ReLU activation function.
    - The output of the weighted layers is added to the shortcut connection's output. If the input and output dimensions are the same, the shortcut connection directly adds the input x to the output of the convolutional layers F(x), resulting in F(x)+x. **If the dimensions of x and F(x) do not match**, a linear projection Ws is applied to x through a convolutional operation to match the dimensions. The resulting output is **F(x)+Wsx**.

Modern Architectures: ResNet & Inception

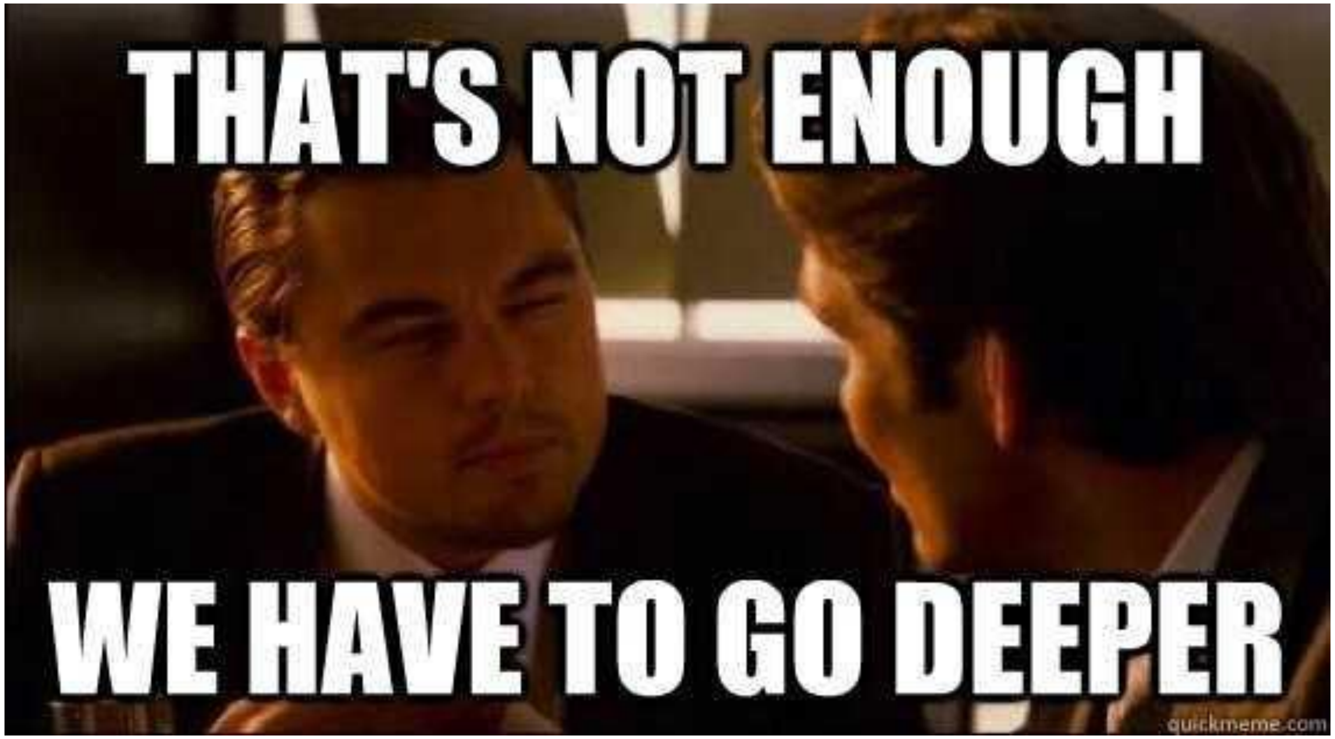
⑩ ResNet

- **1x1 convolution**: 1X1 Conv are used to increase/reduce the number of channels while introducing non-linearity.



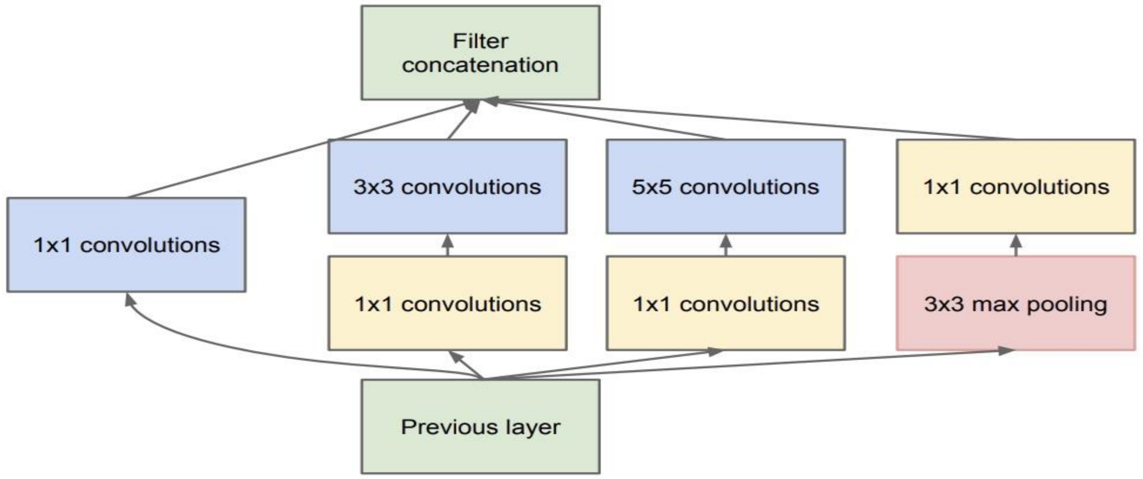
Modern Architectures: ResNet & Inception

- **Inception**



Modern Architectures: ResNet & Inception

- **Inception**
  - The Inception architecture revolutionizes the design of convolutional layers by incorporating multiple filter sizes within the same module, allowing the network to adapt to various spatial hierarchies of features in images.



Modern Architectures: ResNet & Inception

- **The Inception Module**

  - **1x1 Convolutions**: To perform dimensionality reduction or increase, reducing the computational cost and the number of parameters in the network and to increase the network's ability to capture nonlinearities without a significant increase in computational complexity.

  - **Multiple Filter Sizes**: Within each Inception module, convolutional operations with different filter sizes (e.g., 1x1, 3x3, and 5x5) are performed in parallel to capture information from various spatial extents. The outputs of these parallel paths are concatenated along the channel dimension, allowing the network to decide which filters to emphasize for each new input.

Modern Architectures: ResNet & Inception

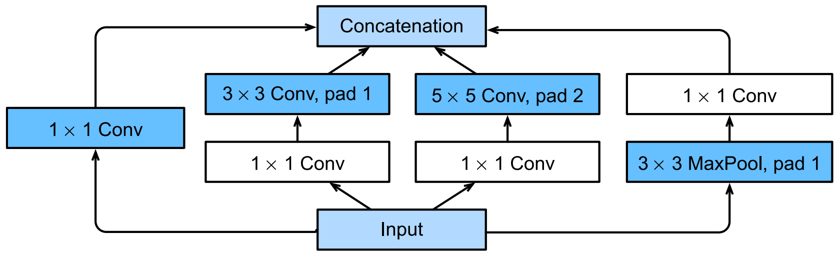
- **The Inception Module**
  - **Pooling**: Inception modules also include a parallel pooling path, typically max pooling, followed by 1x1 convolutions to reduce dimensionality before concatenation.
  - **Dimensionality Reduction**: Before applying larger convolutions (e.g., 3x3 and 5x5), 1x1 convolutions are used for dimensionality reduction, decreasing the computational burden.

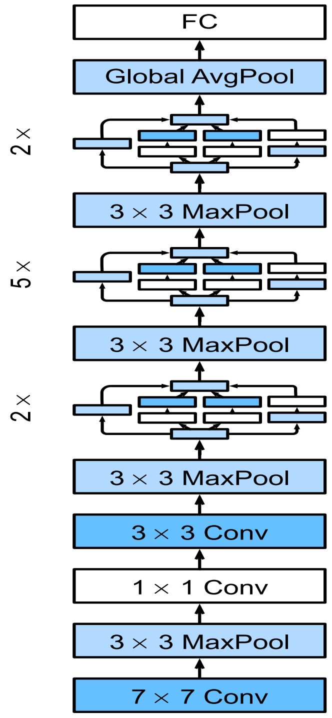
Modern Architectures: ResNet & Inception

- ## Inception Architecture

```
class Inception(nn.Module):
    # c1--c4 are the number of output channels for each branch
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)
        # Branch 1
        self.b1_1 = nn.LazyConv2d(c1, kernel_size=1)
        # Branch 2
        self.b2_1 = nn.LazyConv2d(c2[0], kernel_size=1)
        self.b2_2 = nn.LazyConv2d(c2[1], kernel_size=3, padding=1)
        # Branch 3
        self.b3_1 = nn.LazyConv2d(c3[0], kernel_size=1)
        self.b3_2 = nn.LazyConv2d(c3[1], kernel_size=5, padding=2)
        # Branch 4
        self.b4_1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.b4_2 = nn.LazyConv2d(c4, kernel_size=1)

    def forward(self, x):
        b1 = F.relu(self.b1_1(x))
        b2 = F.relu(self.b2_2(F.relu(self.b2_1(x))))
        b3 = F.relu(self.b3_2(F.relu(self.b3_1(x))))
        b4 = F.relu(self.b4_2(self.b4_1(x)))
        return torch.cat((b1, b2, b3, b4), dim=1)
```





GoogLeNet

Modern Architectures: ResNet & Inception

- **The Inception Module**
  - The Inception module's combination of parallel convolutional paths with differents filter sizes and 1x1 convolutions for dimensionality management allows the network to be both wide (in terms of capturing a broad range of features) and deep (in terms of layers), while maintaining computational efficiency. This design philosophy has been extended and refined in subsequent versions of the Inception architecture, such as Inception v2, v3 and v4, each introducing further optimizations and improvements.

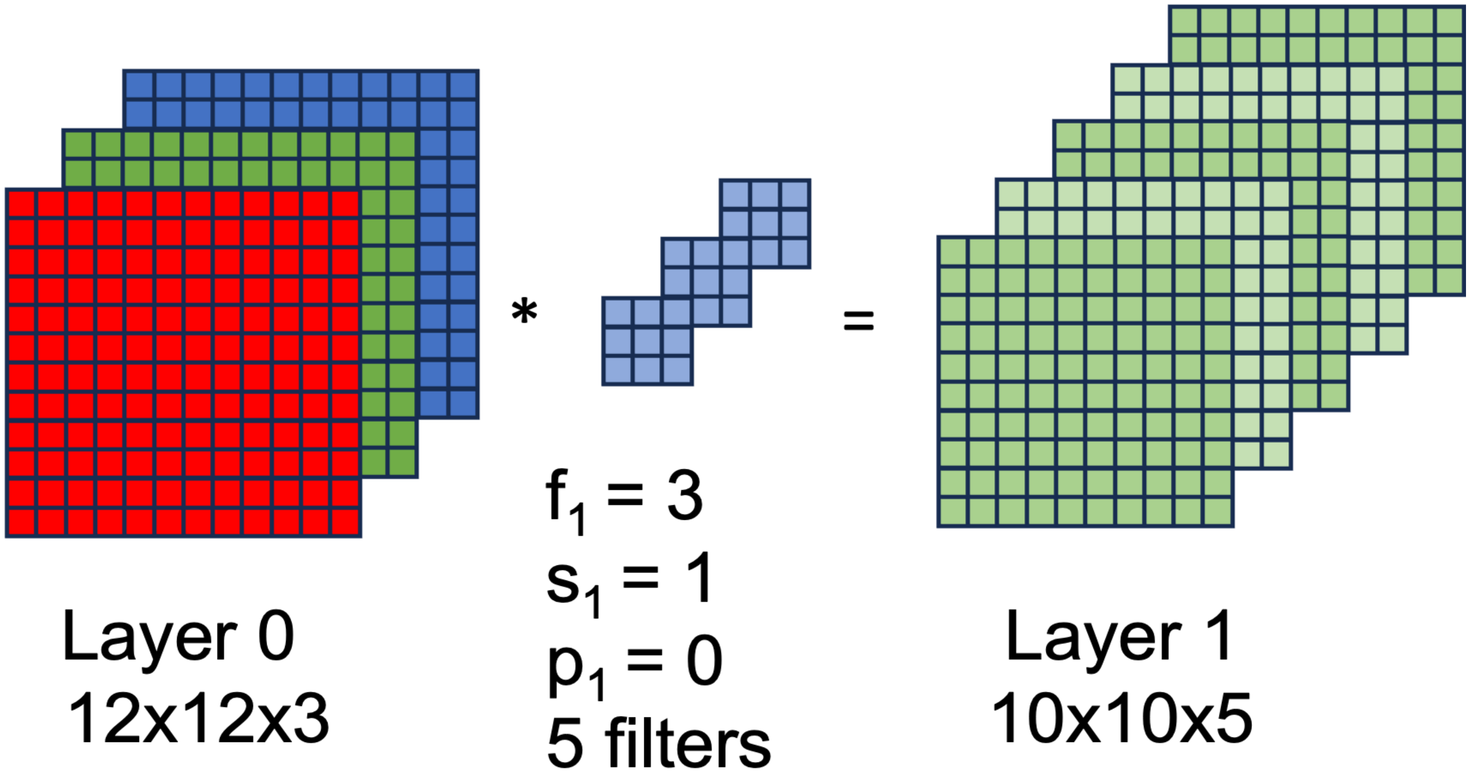
Computational Efficiency: MobileNet & EfficientNet

- **MobileNet**
  - The key innovation is in their efficient architectural design, aimed at **reducing computational cost while maintaining high performance**, especially on mobile and embedded devices.
  - Instead of using standard convolutions, it employs **depthwise separable convolutions**, which use a standard convolution into a depthwise convolution and a 1x1 pointwise convolution. *The input is first processed by a depthwise convolution (nw·nh·nc), applying a single filter per input channel. This is followed by a pointwise convolution (1·1·n'c) convolutions that combines the outputs of the depthwise convolution, adjusting the depth as necessary.
  - The computational cost is significantly reduced compared to standard convolutions.

Computational Efficiency: MobileNet & EfficientNet

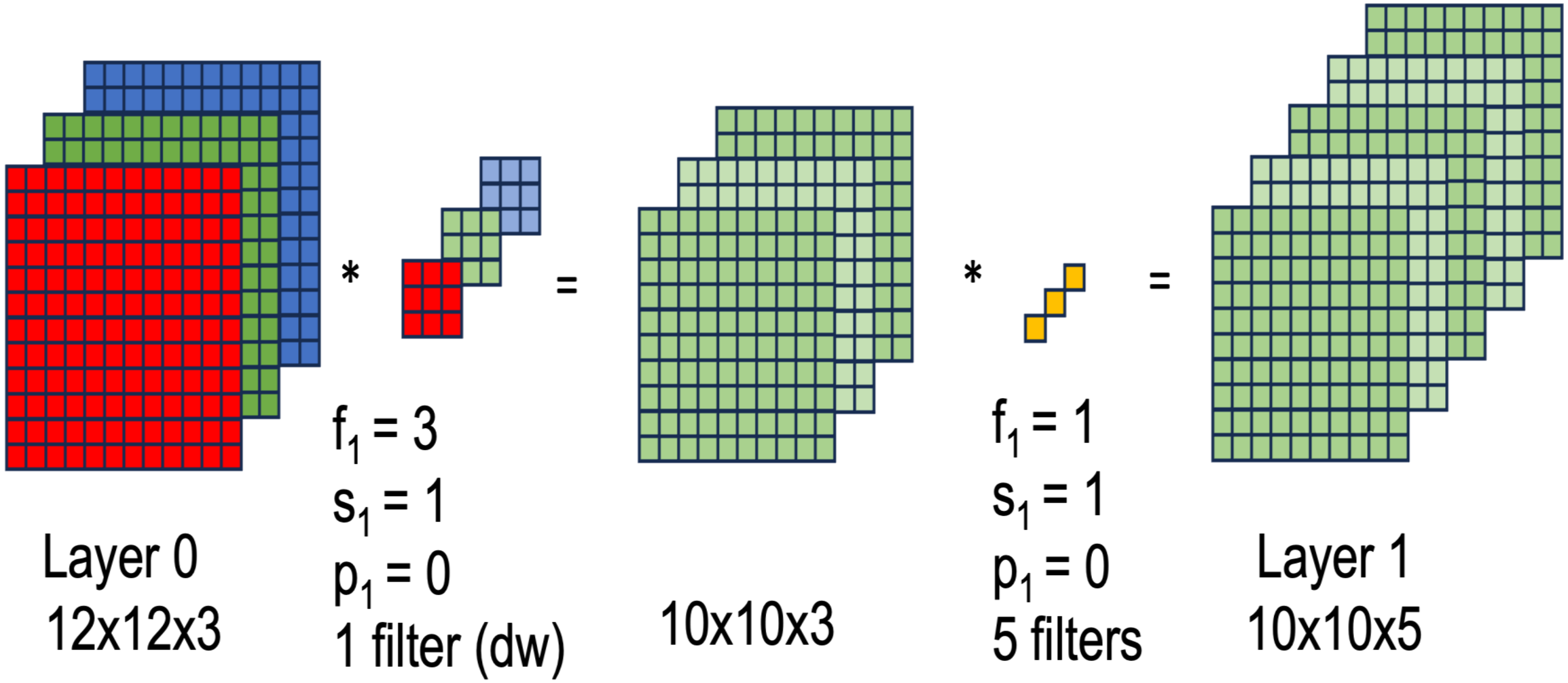
- **MobileNet**
  - **Standard Convolution**: filter parameters *x* filter positions *x* number of filters.

    (3 x 3 x 3) x (10 x 10) x 5 = 13.500 operations



Computational Efficiency: MobileNet & EfficientNet

- **MobileNet**
  - **Depthwise separable convolution**: filter parameters *x* filter positions *x* number of filters. (3 x 3) x (10 x 10) x 3 = 2.700 operations. (1 x 1 x 3) x (10 x 10) x 5 = 1.500 operations. Total = 4.200 operations (reduction of 30%)

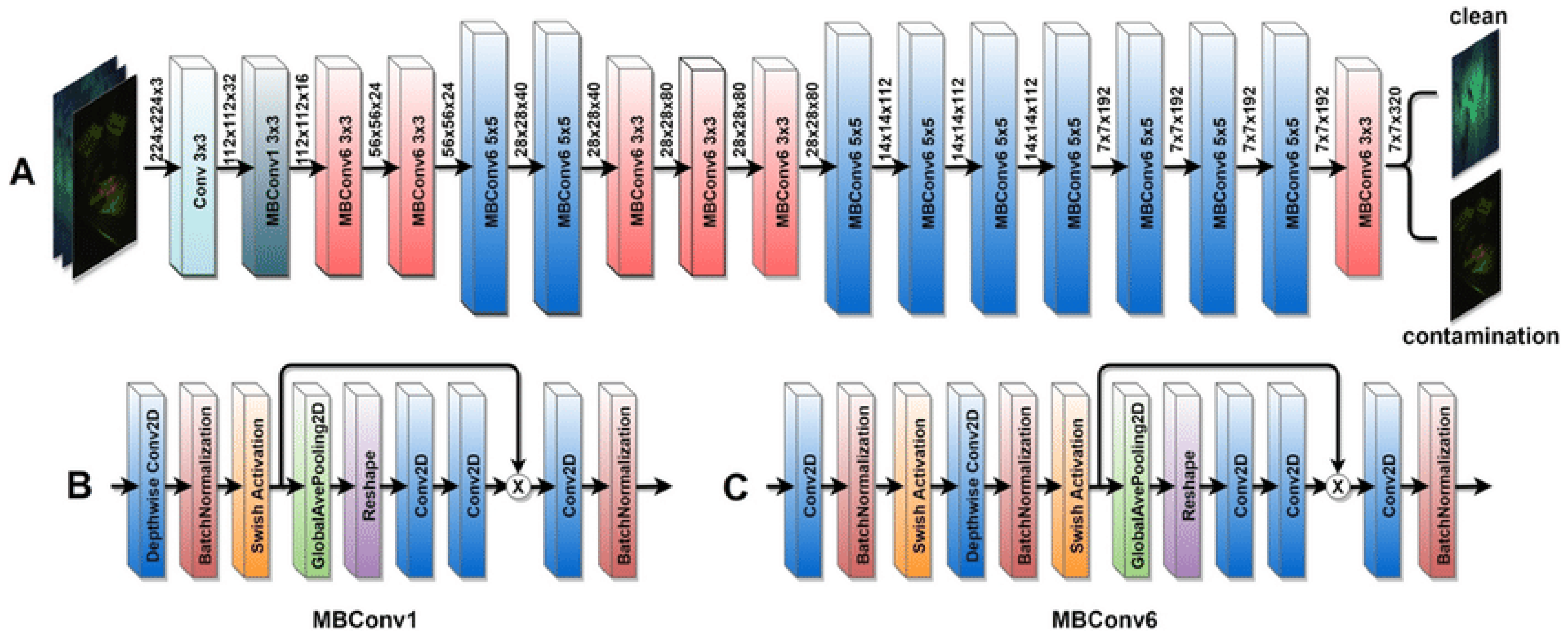


Computational Efficiency: MobileNet & EfficientNet

- **MobileNetV2**
  - It introduces residual connections similar to those in ResNet, but within the framework of inverted residual blocks. These connections allow the input to bypass one or more layers, facilitating the flow of gradients during training and mitigating the vanishing gradient problem.
  - Inverted Residual Blocks
    - **Pointwise convolution**: Each block starts with a 1x1 convolution that expands the input's depth, increasing the representation capacity and allowing the network to learn more complex functions.
    - **Depthwise convolution:** Follows the expansion layer, applying spatial filtering within each channel.
    - **Projection layer:** A 1x1 convolution that projects the expanded feature map back to a lower dimension, reducing the size and computational cost of the feature map.
  - This expansion-projection strategy increases the network's expressiveness while keeping the computational cost low by expanding the feature space only temporarily within the block.

Computational Efficiency: MobileNet & EfficientNet

- **EfficientNet**
  - These architectures systematically scale up CNNs in a more structured manner to achieve better efficiency and accuracy. The key innovation of EfficientNet is the use of a compound scaling method that uniformly scales network **width**, **depth**, and **resolution** with a set of fixed scaling coefficients.
  - It is able to achieve state-of-the-art accuracy with significantly fewer parameters and FLOPs (floating-point operations per second) compared to previous architectures.

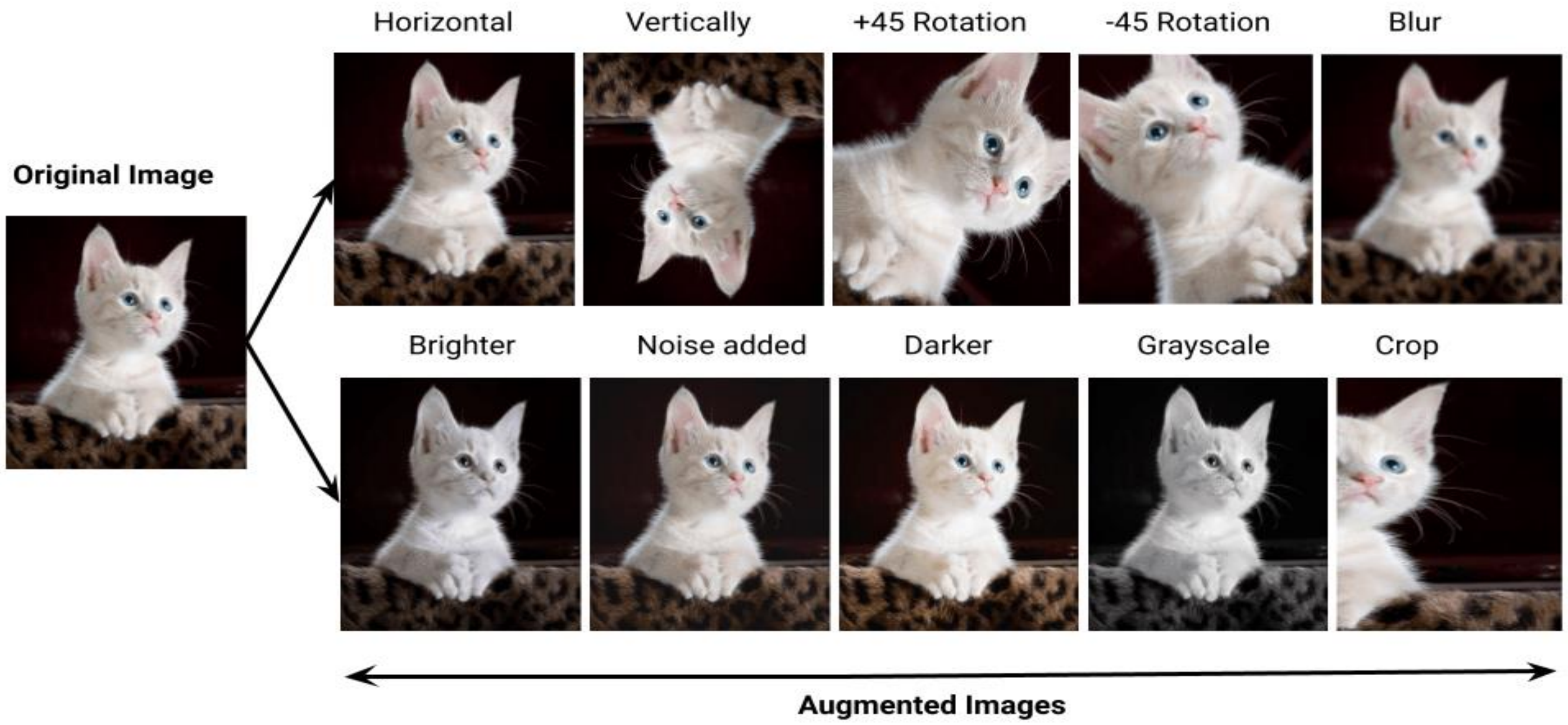


Data Strategy: Image Augmentation

- **Data Strategy: Image Augmentation**
  - **Image Augmentation** is a technique used to enhance the diversity of a training dataset without actually collecting new images. This is achieved by applying a series of random transformations to the existing images in the dataset, such as rotations, translations, flips, scaling, shearing, and color variations. These transformations produce altered versions of the images, which help the model generalize better from the training data, making it more robust to variations it might encounter in real-world data.
  - The benefits are: * **Enhanced Generalization**: Augmentation increases the diversity of the training set, helping the model generalize better to unseen data. * **Reduced Overfitting**: By providing varied examples, it prevents the model from memorizing specific images. * **Improved Robustnes**: Models become more robust to variations in input data, such as different angles, lighting conditions, and occlusions.

Data Strategy: Image Augmentation

- **Data Strategy: Image Augmentation**



Original Image
Horizontal
Vertically
+45 Rotation
-45 Rotation
Blur
Brighter
Noise added
Darker
Grayscale
Crop
Augmented Images