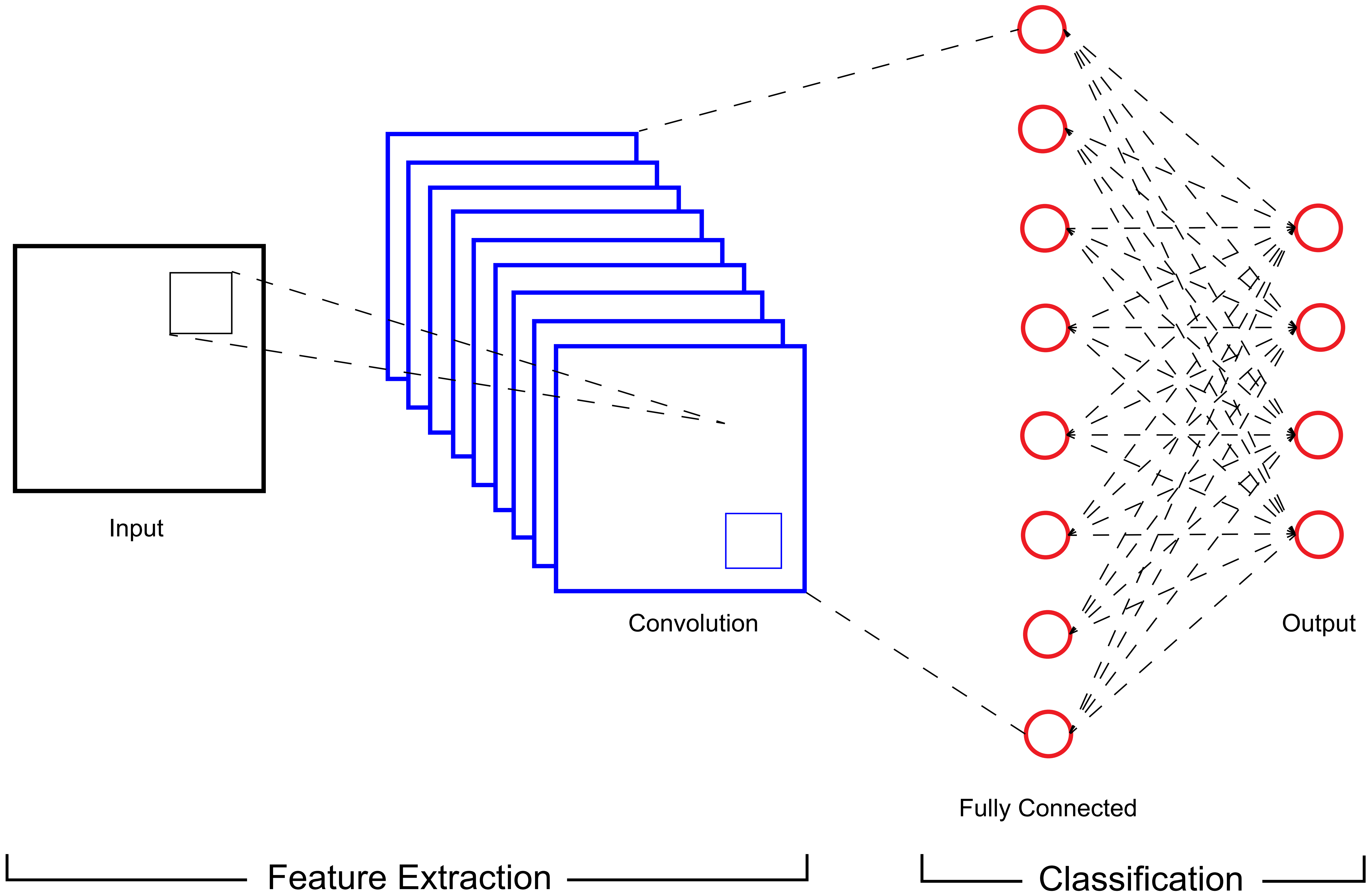
CNN Classification with Blending Mode Data Augmentation

- CNNs
- DenseNet
- Data Augmentation
- Blending Modes
- Dataset
- Testing
- Conclusions


Input
Convolution
Fully Connected
Output
Feature Extraction
Classification

Convolution Feature Maps

- Extracts patches from the input feature map and applies the same transformation to all these patches, producing an output feature map.
- Feature maps are size (width, height, color channels).
- Feature maps are defined by two parameters:
    1. Size of the patches usually 3x3 or 5x5.
    2. Depth of the output feature maps equals number of filters.
- Every spatial location in the input feature map corresponds to the same location in the output feature map.
- 3x3 patch, the output feature maps [i , j , :] comes from the input feature map [i - 1: i + 1, j - 1 : j + 1, :]

Edge Detector

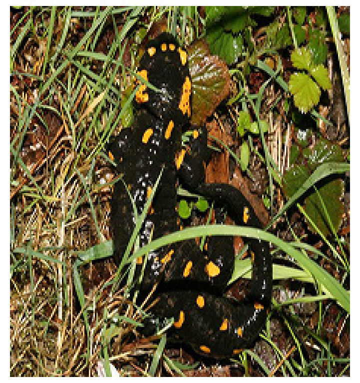
$$Kernel\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix} X \begin{bmatrix}255 & 226 & 153\\226 & 153 & 0\\153 & 0 & 0\end{bmatrix} Image$$

(255 * 1) + (226 * 1) + (156 * 1) + (226 * 0) + (153 * 0) + (0 * 0)
+ (153 * -1) + (0 * -1) + (0 * -1) = 481 (max 255)

Blur Filter

$$Kernel \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} X \begin{bmatrix} 255 & 226 & 153 \\ 226 & 153 & 0 \\ 153 & 0 & 0 \end{bmatrix} Image$$

(255 * 1/9) + (226 * 1/9) + (153 * 1/9) + (226 * 1/9) + (153 * 1/9)
+ (0 * 1/9) + (153 * 1/9) + (0 * 1/9) + (0 * 1/9) = 129.555 ≈ 130



→

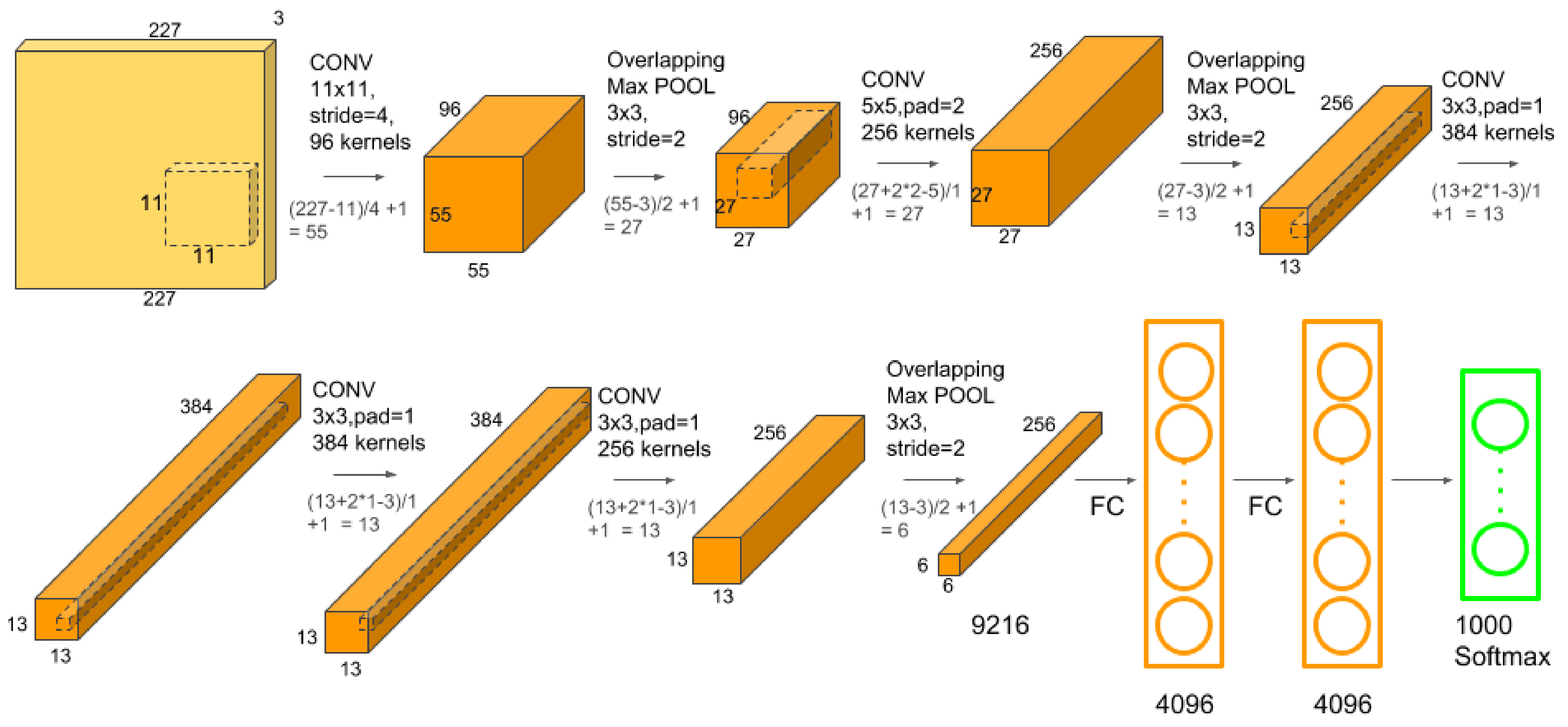
→

Model Progression

- AlexNet
- VGG16
- ResNet50
- DenseNet121

AlexNet

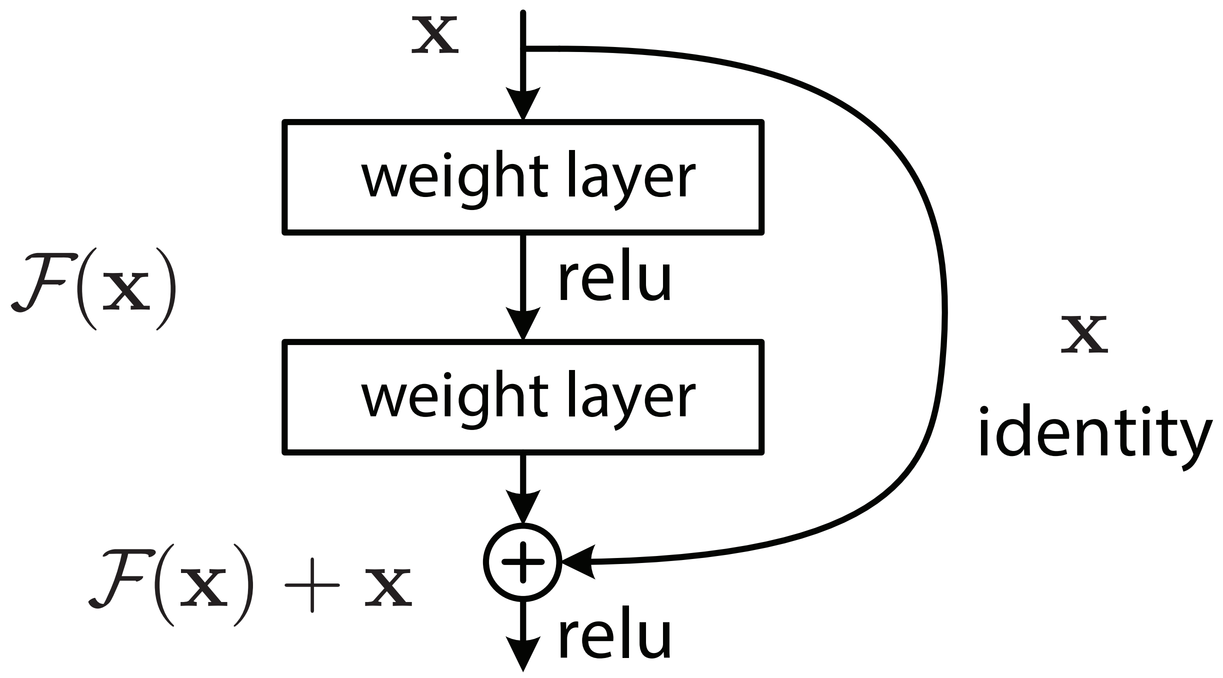
input (227 X 227 X 3 RGB image)
conv11 - Stride: 4 - 55 x 55 x 96 - ReLu
maxpool - Stride: 2 - 27 x 27 x 96
conv5 - Pad: 2 - 27 x 27 x 256 - ReLu
maxpool - Stride: 2 - 13 x 13 x 96
conv3 - Pad: 1 - 12 x 12 x 384 - ReLu
conv3 - Pad: 1 - 12 x 12 x 384 - ReLu
conv3 - Pad: 1 - 12 x 12 x 256 - ReLu
maxpool - Stride: 2 - 6 x 6 x 256
flatten
Densely Connected layers



VGG16

input (224 X 224 X 3 RGB image)
conv3 - Stride: 1 - 224 x 224 x 64 - ReLu **conv3** - Stride: 1 - 224 x 224 x 64 - ReLu **maxpool** - Stride: 2 - 112 x 112 x 128
conv3 - Stride: 1 - 112 x 112 x 128 - ReLu **conv3** - Stride: 1 - 112 x 112 x 128 - ReLu **maxpool** - Stride: 2 - 56 x 56 x 256
conv3 - Stride: 1 - 56 x 56 x 256 - ReLu **conv3** - Stride: 1 - 56 x 56 x 256 - ReLu **conv3** - Stride: 1 - 56 x 56 x 256 - ReLu **maxpool** - Stride: 2 - 28 x 28 x 512
conv3 - Stride: 1 - 28 x 28 x 512 - ReLu **conv3** - Stride: 1 - 28 x 28 x 512 - ReLu **conv3** - Stride: 1 - 28 x 28 x 512 - ReLu **maxpool** - Stride: 2 - 14 x 14 x 512
conv3 - Stride: 1 - 14 x 14 x 512 - ReLu **conv3** - Stride: 1 - 14 x 14 x 512 - ReLu **conv3** - Stride: 1 - 14 x 14 x 512 - ReLu **maxpool** - Stride 2 - 7 x 7 x 512
flatten
Densely Connected layers

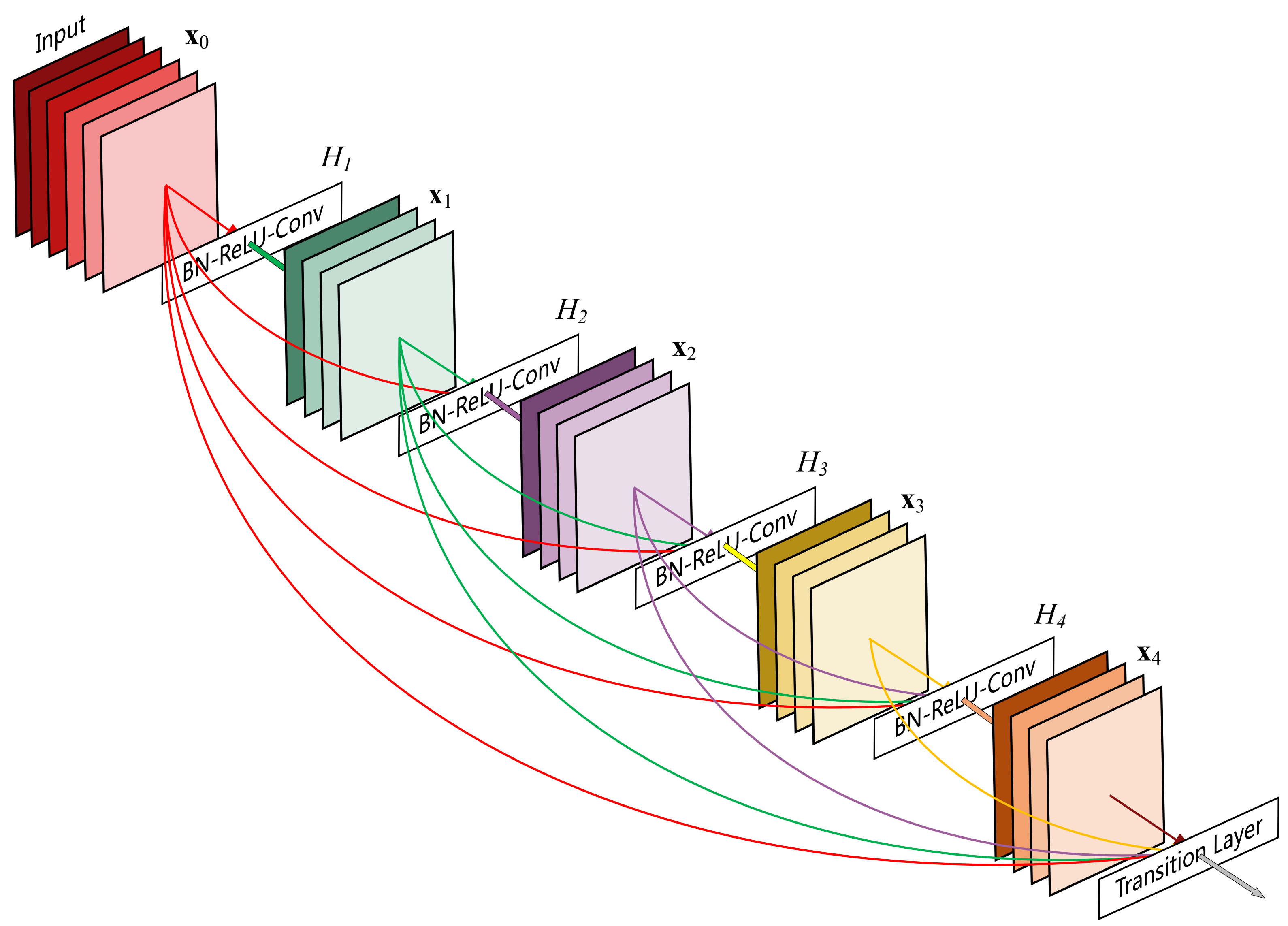
Residual Connection



ResNet50

input (224 X 224 X 3 RGB image)
conv7 - Stride: 2 - 112 x 112 x 64 - ReLu
maxpool - Stride: 2 - 56 x 56 x 64
conv1 - Stride: 1 - 56 x 56 x 64 - ReLu **conv3** - Stride: 1 - 56 x 56 x 64 - ReLu **conv1** - Stride: 1 - 56 x 56 x 256 - ReLu
Add skip connection
Repeated 3 times
conv1 - Stride: 2 - 28 x 28 x 128 - ReLu **conv3** - Stride: 2 - 28 x 28 x 128 - ReLu **conv1** - Stride: 2 - 28 x 28 x 512 - ReLu
Add skip connection
Repeated 4 times
conv1 - Stride: 2 - 14 x 14 x 256 - ReLu **conv3** - Stride: 2 - 14 x 14 x 256 - ReLu **conv1** - Stride: 1 - 14 x 14 x 1024 - ReLu
Add skip connection
Repeated 6 times
conv1 - Stride: 2 - 7 x 7 x 512 - ReLu **conv3** - Stride: 2 - 7 x 7 x 512 - ReLu **conv1** - Stride: 2 - 7 x 7 x 2028 - ReLu
Repeated 3 times
average pooling
Densely Connected layers

DenseNet121



DenseNet121

input (224 X 224 X 3 RGB image)
conv7 - Stride: 2 - 112 x 112 x 64 - ReLu
maxpool - Stride: 2 - 56 x 56 x 64
conv1 - Stride: 1 - 56 x 56 x 256 - ReLu **conv3** - Stride: 1 - 56 x 56 x 256 - ReLu
Repeated 6 times
conv1 - Stride: 1 - 56 x 56 x 128 **avgpool2** - Stride: 2 - 28 x 28 x 128 - SoftMax
conv1 - Stride: 1 - 28 x 28 x 512 - ReLu **conv3** - Stride: 1 - 28 x 28 x 512 - ReLu
Repeated 12 times
conv1 - Stride: 1 - 28 x 28 x 256 **avgpool2** - Stride: 2 - 14 x 14 x 256 - SoftMax
conv1 - Stride: 1 - 14 x 14 x 1024 - ReLu **conv3** - Stride: 1 - 14 x 14 x 1024 - ReLu
Repeated 24 times
conv1 - Stride: 1 - 14 x 14 x 512 **avgpool2** - Stride: 2 - 7 x 7 x 512 - SoftMax
conv1 - Stride: 1 - 7 x 7 x 1024 - ReLu **conv3** - Stride: 1 - 7 x 7 x 1024 - ReLu
Repeated 16 times
average pooling
Densely Connected layers

Model Layer Depth

- AlexNet / VGG16 = L layers with L connections.
- ResNet50 = L layers with L + (L/2) connections.
- DenseNet121 = L layers with $L(L + 1) / 2$ connections.

Data Augmenation

- Spatial and Color Channel
- Safe and Unsafe
- Online and Offline

Spatial

1. **Flipping**
2. Rotating
3. Translating
4. Cropping

Color Channel

1. Brightness
2. Contrast
3. Saturation
4. **Blending**

Blending Modes

- Color channel augmentation where separate images are blended with each other according to a set of rules.
- Stacked Transparency.
- Multiply & screen blends are used.

Linear Interpolation

$$f(a, b) = a + b$$

Where: a = RGB image with values between 0-1.

b = RGB image with values between 0-1.

Linear Interpolation with Opacity

$$f(a, b, \alpha) = (a * \alpha) + (b * (1 - \alpha))$$

Where: α = opacity between 0-1.

Multiply Blending

$$f(a,b) = a * b$$

Input: RGB image of size img[width,height,color channels]
Output: Blended RGB image

```
blendImg = array[width,height,color channels]
// Iterate through individual pixel values
for each row in the image do
    for each pixel in the row do
        for each color in the pixel do
            blendImg[row,pixel,color] = img[row,pixel,color] *
             img[row,pixel,color]
```

Screen Blending

$$f(a,b) = 1 - (1-a)(1-b)$$

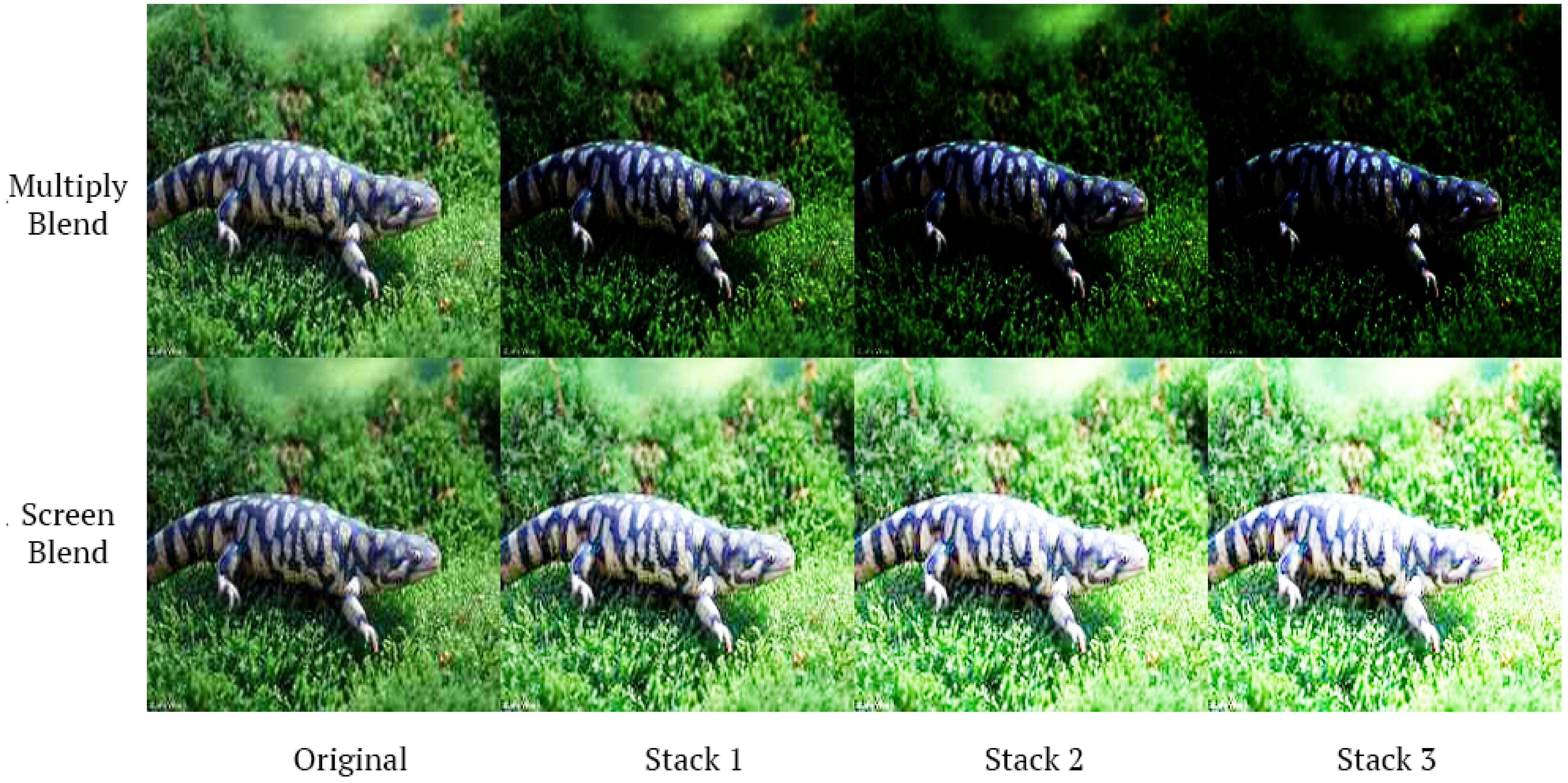
Input: RGB image of size img[width,height,color channels]

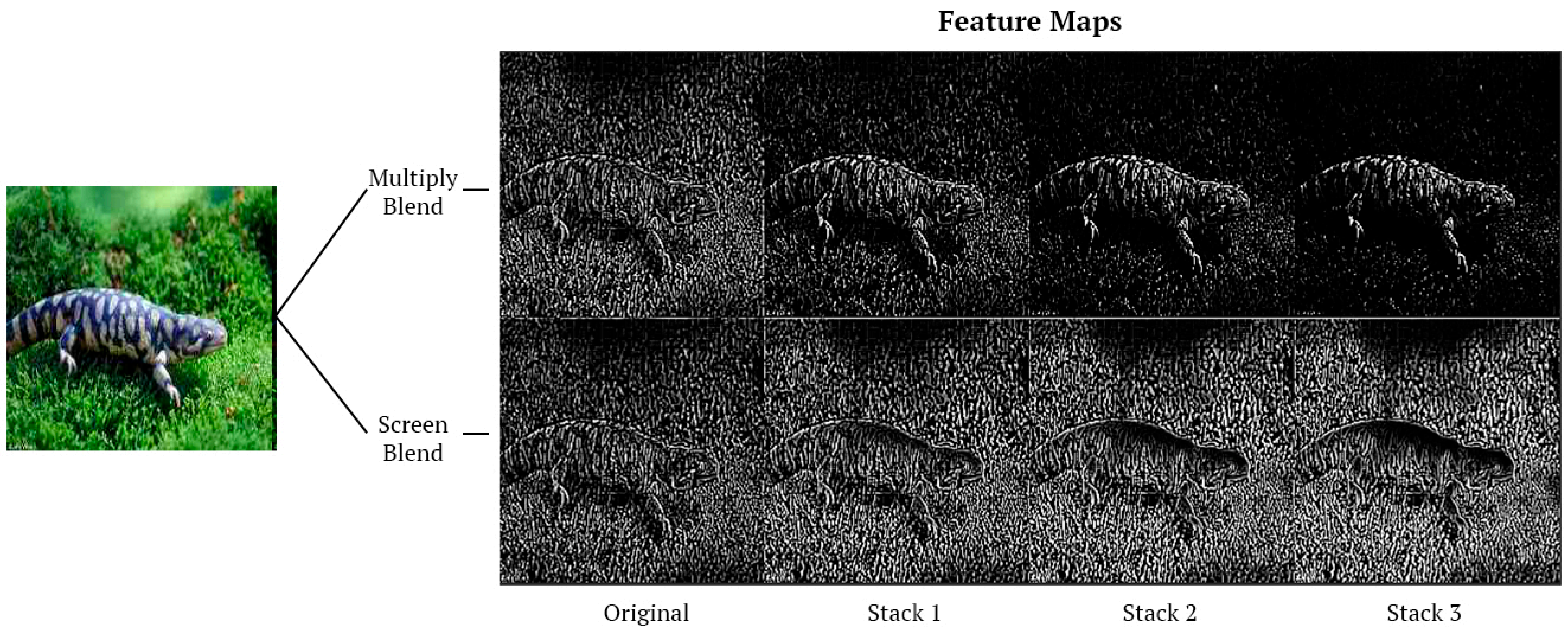
Output: Blended RGB image

```
// Iterate through individual pixel values
blendImg = array[width,height,color channels]
for each row in the image do
    for each pixel in the row do
        for each color in the pixel do
            blendImg[row,pixel,color] = 1 - (1 - img[row,pixel,color]) *
              (1 - img[row,pixel,color])
```

Full Data Aug Layer

```
Input: img = RGB image of size img[width,height,color channels]
       range = Value Range either integer 0 - 255 or float 0 - 1
       α = Opacity float between 0.01 - 1
       stack = Stack depth integer between 1 - 10
Output: Final blended RGB image
chooseBlend = int(random(0,1))
blendImg = array[width,height,color channels]
finalImg = array[width,height,color channels]
// Iterate through individual pixel values
for each row in the image do
    for each pixel in the row do
        for each color in the pixel do
            if chooseBlend = 0 then
                blendImg[row,pixel,color] = img[row,pixel,color] *
                  img[row,pixel,color] ;
            else
                blendImg[row,pixel,color] = 1 - (1 -
                  img[row,pixel,color]) * (1 - img[row,pixel,color]);
// Continue to blend image until stack is zero
while stack > 0 do
    for each row in the image do
        for each pixel in the row do
            for each color in the pixel do
                if chooseBlend = 0 then
                    blendImg[row,pixel,color] =
                      blendImg[row,pixel,color] * img[row,pixel,color] ;
                else
                    blendImg[row,pixel,color] = 1 - (1 -
                      blendImg[row,pixel,color]) * (1 -
                      img[row,pixel,color]);
    stack = stack - 1;
// Linearly Interpolate the blended image and original
   with Opacity
finalImg = (blendImg * α) + (img * (1 - α))
```


Multiply Blend
Screen Blend
Original
Stack 1
Stack 2
Stack 3


Feature Maps
Multiply Blend
Screen Blend
Original
Stack 1
Stack 2
Stack 3

Data Augmentation Pipeline

- Keras API
- Base Augmentation Layer

Input Image
Multiply & Blend
Horizontal Flip
CNN

Input Image
Multiply & Blend
CNN

Input Image
Horizontal Flip
CNN

Dataset Tiny ImageNet

ImageNet

1. 224 x 244 pixels
2. 1,000,000 images
3. 1000 classes

Tiny ImageNet

1. 64 x 64 pixels
2. 100,000 images
3. 200 classes

Model & Training Specfications

DenseNet-121 Model Details.

Train samples	Validation samples	Batch size	Epochs	Optimizer	Learning Rate
100,000	10,000	32	50	SGD	.001 with 0.9 Momentum

Training Computer Specifications.

CPU	RAM	GPU
AMD RYZEN 9 5950X	64 GB DDR4 3600	Nvidia RTX 3090 24 GB RAM

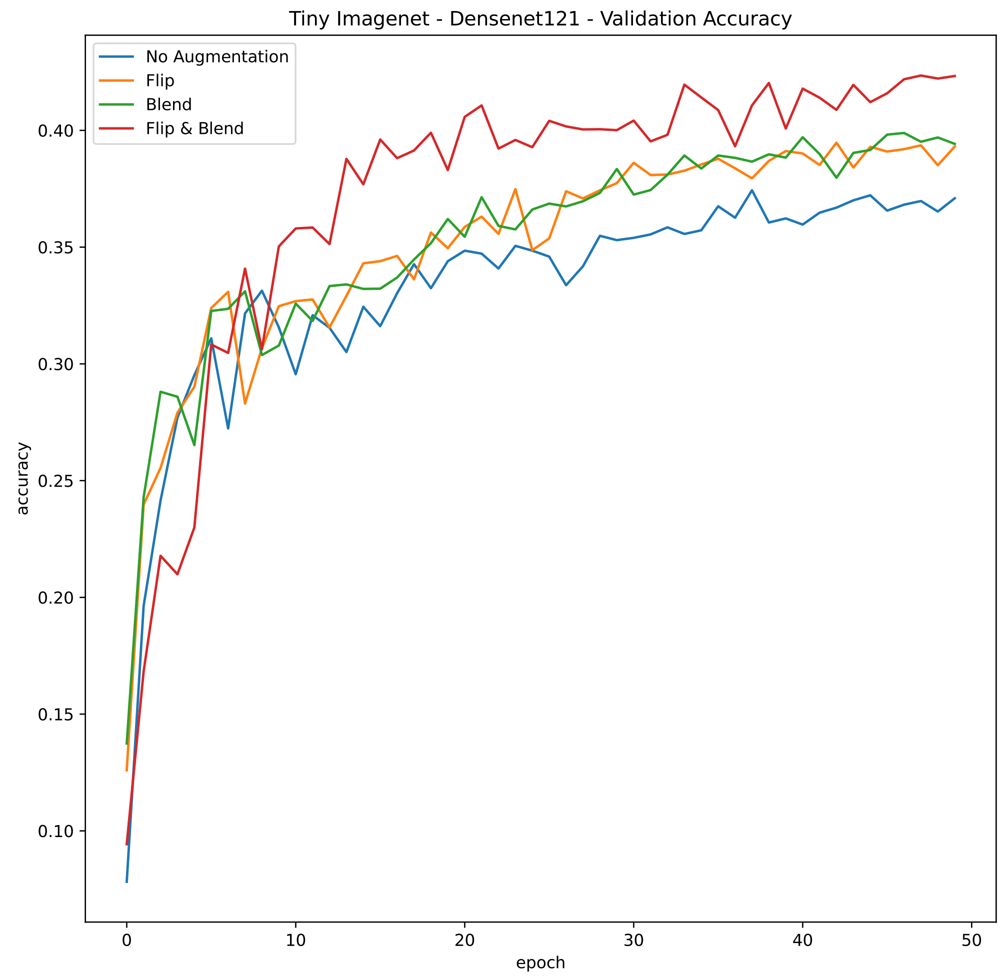


Figure 4.1: Validation Accuracy.

Table 4.3: Validation Accuracy Results.

Data Augmentation Layer(s)	**Top-1**
No Augmentation	37.1%
Horizontal Flip	39.4%
Multiply & Screen Blend	39.6%
Horizontal Flip, Multiply, & Screen	43.1%

Horizontal flip 6.2% increase
Multiply & Screen - 6.7% increase
Both Layers - 16.2% increase

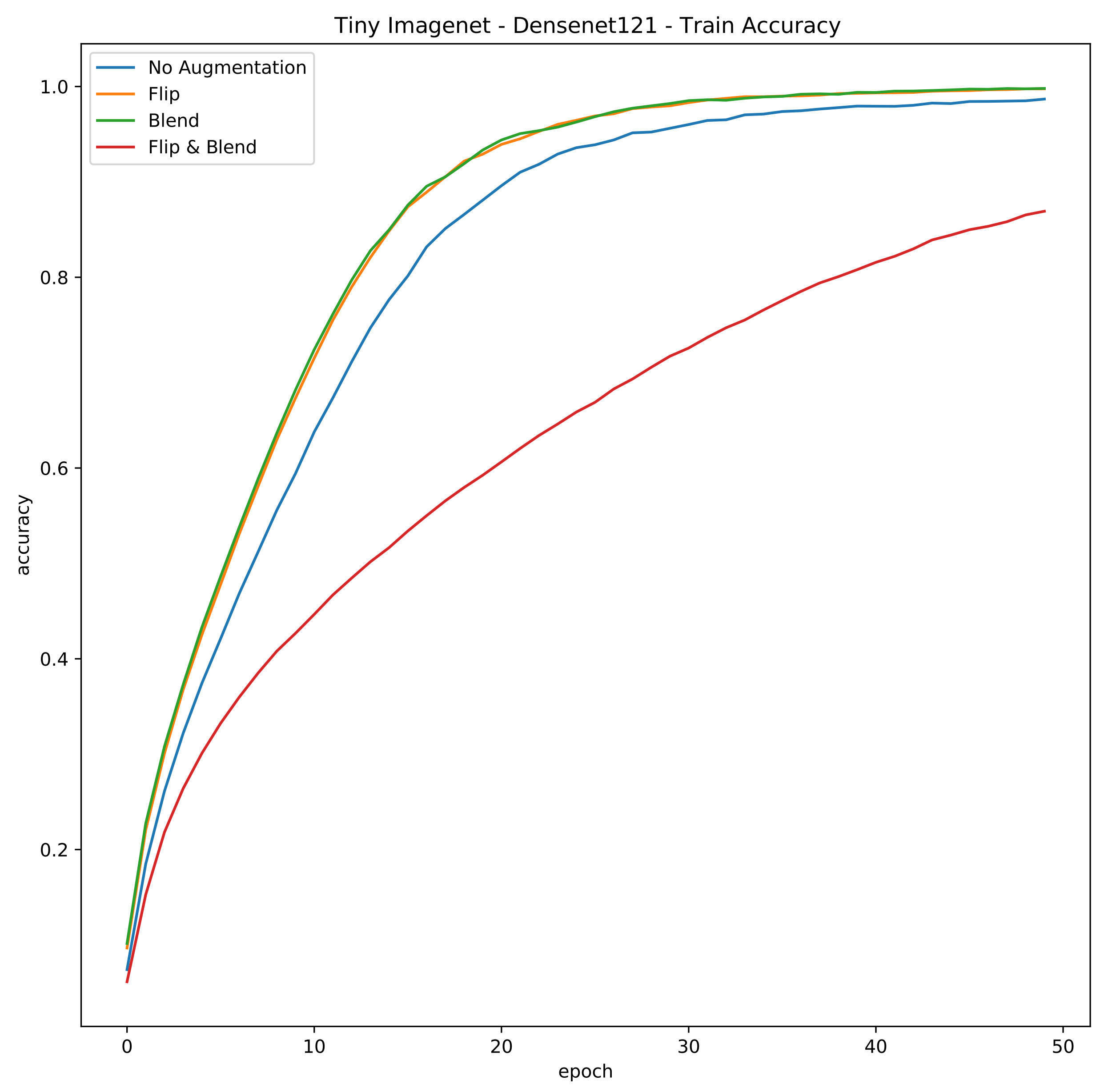


Figure 4.2: Training Accuracy.

Data Augmentation Layer(s)	**Training Time** in seconds
No Augmentation	4992.5
Horizontal Flip	5270.2
Multiply & Screen Blend	5178.5
H. Flip and Multiply and Screen	5295.1

- Conclusion

Future Work

- Finely tune the hyperparameeters.
- Train on the full ImageNet dataset.
- Test on object detection and segmentation.