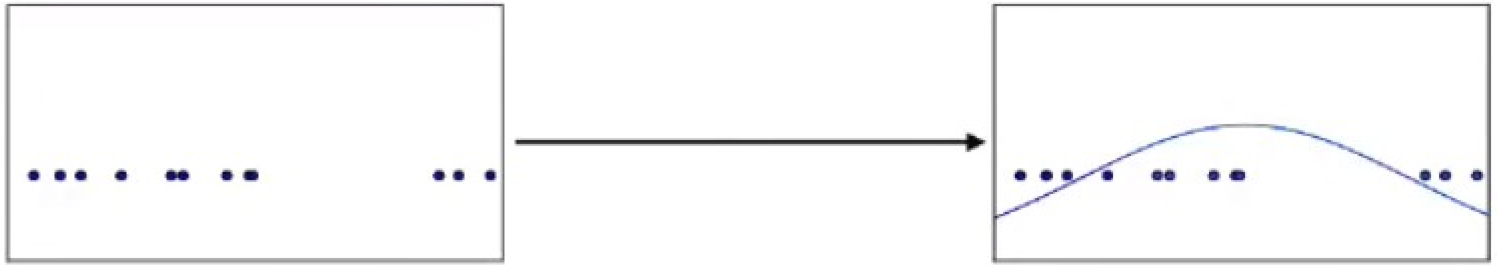
Generative Adversarial Networks

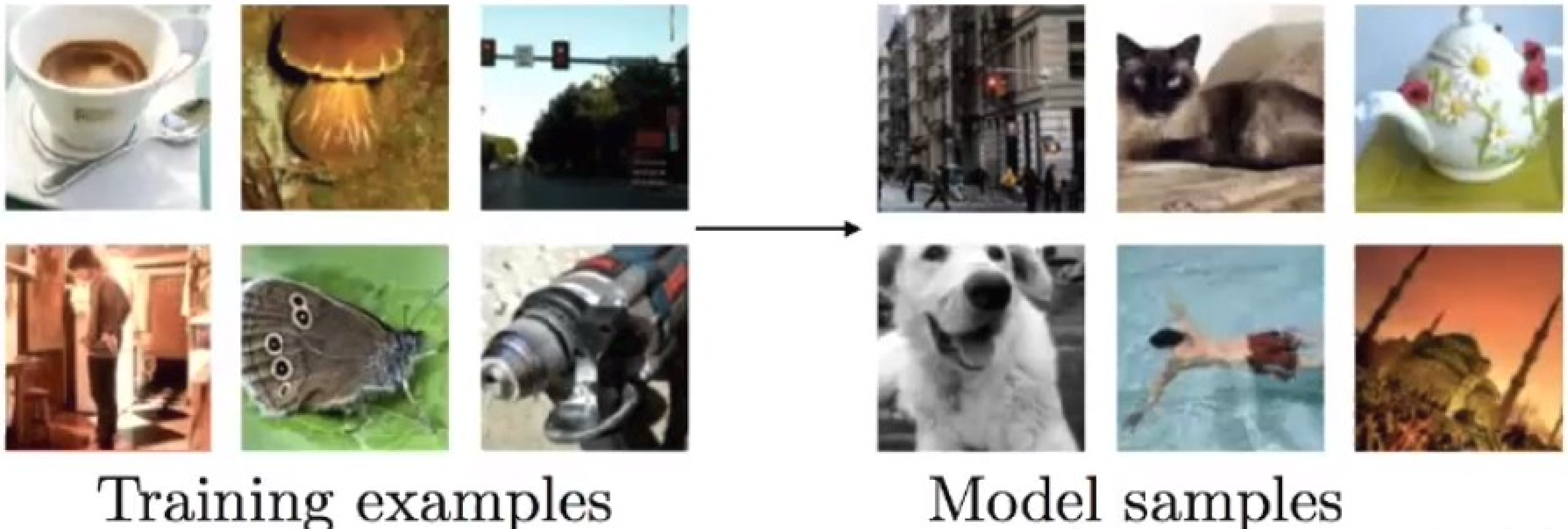
Nhat Nguyen
University of Alberta

Generative Modeling

- Density estimation



- Sample generation



(Goodfellow 2016)

Applications of GANs

- Generate realistic environment: help in reinforcement learning, imitation learning.
  - Easy to parallelize.
  - No cost for making mistakes.
- Semi-supervised learning: can be trained to fill missing labels.
- Many tasks require generating realistic samples
- ...

Examples

- GANs can act as an experience replay buffer to prevent catastrophic forgetting in RL

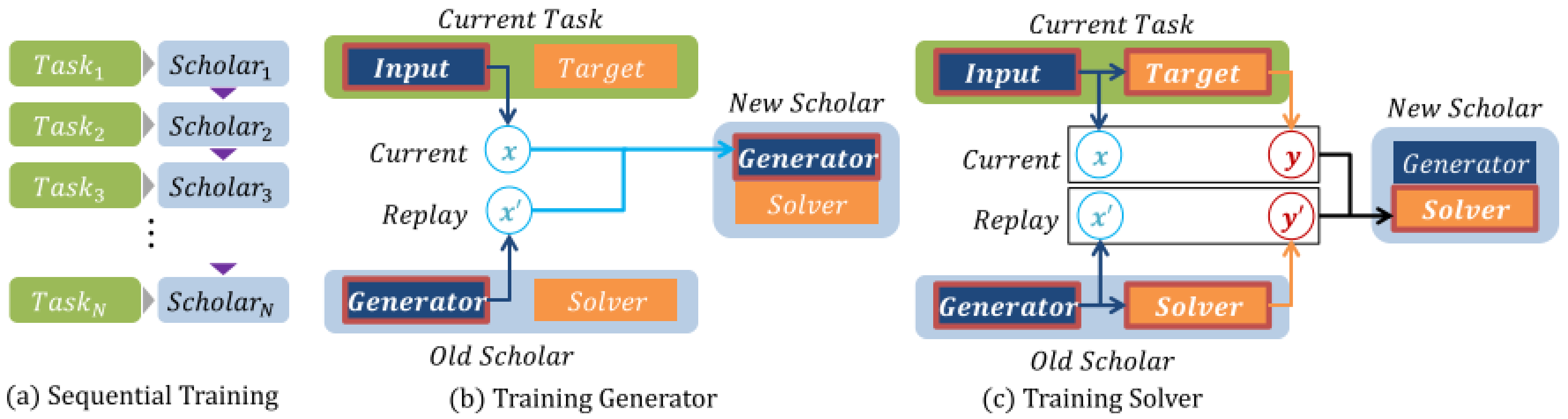


Figure 1: Sequential training of scholar models. (a) Training a sequence of scholar models is equivalent to continuous training of a single scholar while referring to past copy of a self. (b) A new generator is trained to mimic a mixed data distribution of real samples x and replayed inputs x' from previous generator. (c) A new solver learns from real input-target pairs (x, y) and replayed input-target pairs (x', y'), where replayed response y' is obtained by feeding generated inputs into previous solver.

Examples

- Super image resolution

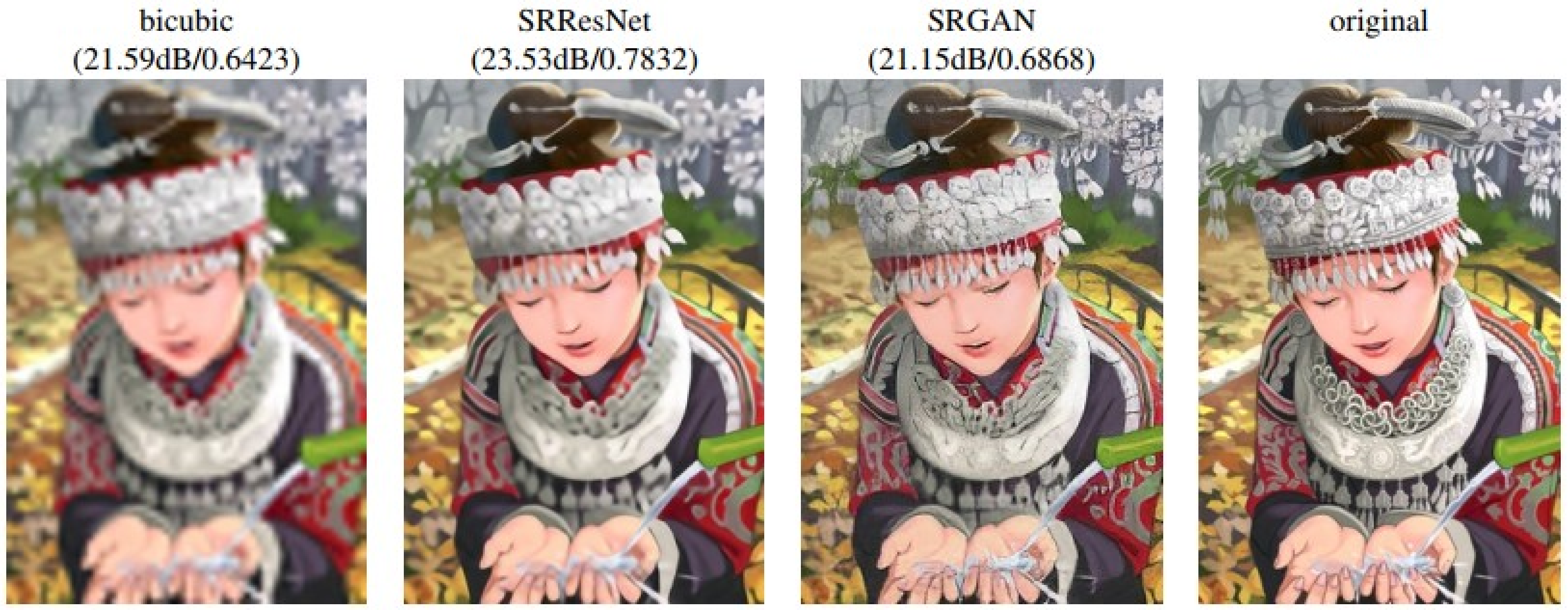
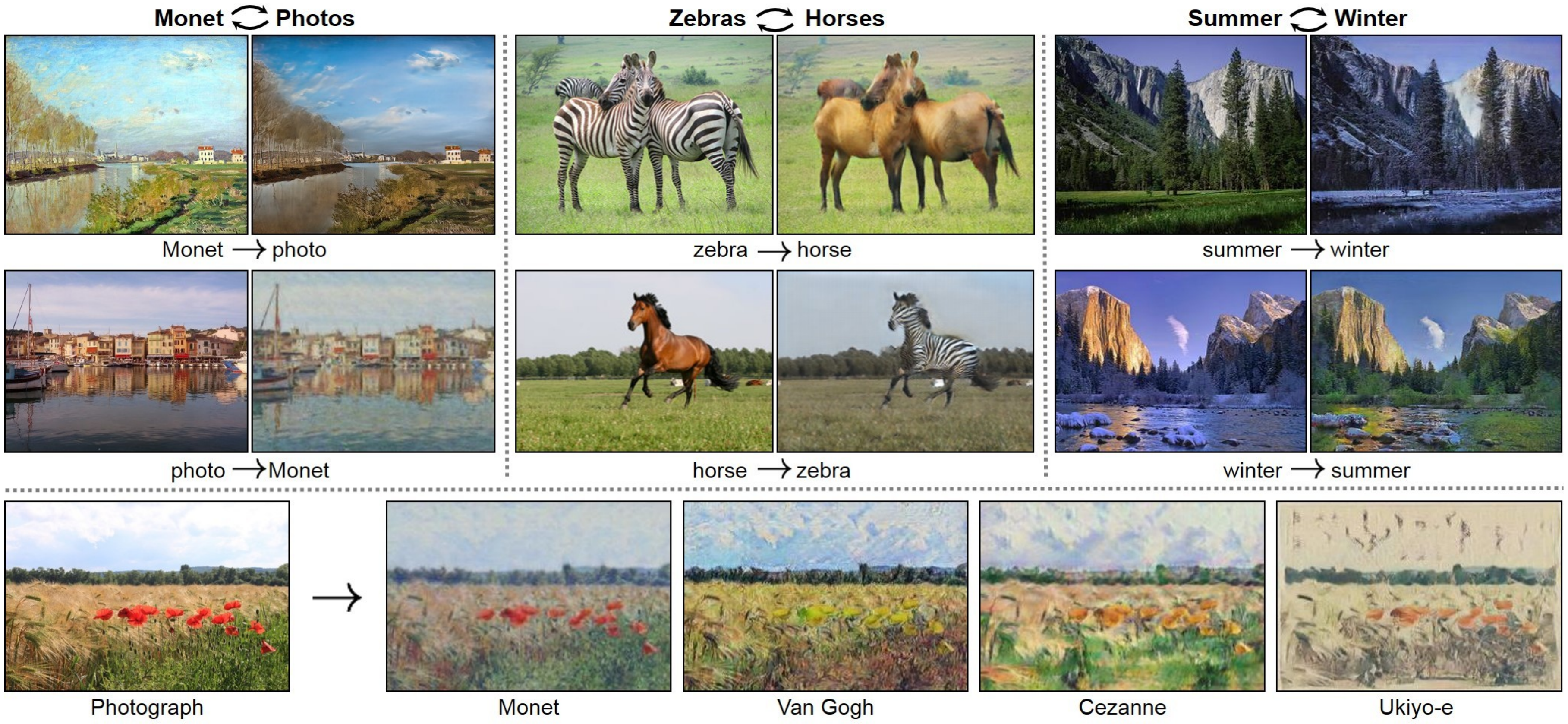


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., … Shi Twitter, W. (n.d.). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.

Examples

- Style transfer:



Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." arXiv preprint arXiv:1703.10593 (2017).

Examples

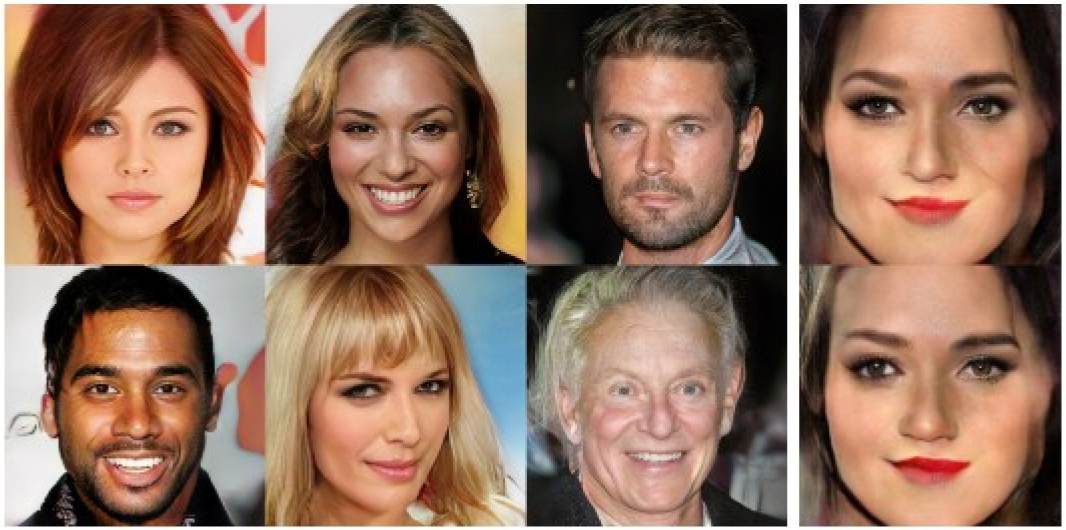


Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations. On the right, two images from an earlier megapixel GAN by Marchesi (2017) show limited detail and variation.

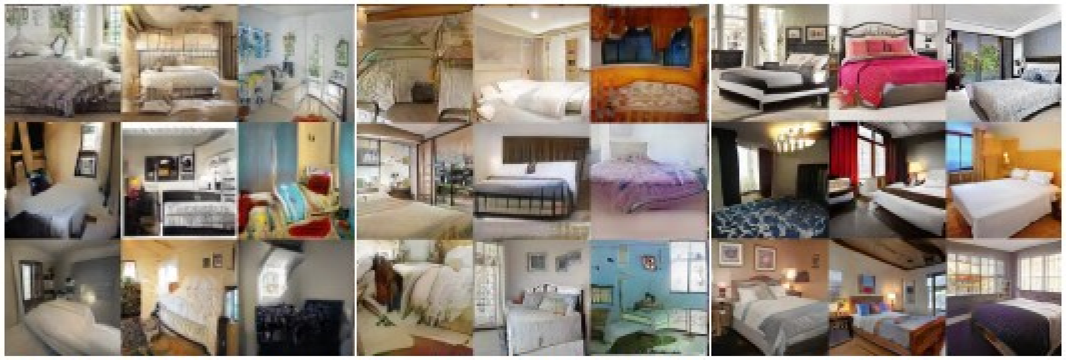

Mao et al. (2016b) (128 × 128)
Gulrajani et al. (2017) (128 × 128)
Our (256 × 256)

Figure 6: Visual quality comparison in LSUN BEDROOM; pictures copied from the cited articles.

Karras, Tero, et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation." arXiv preprint arXiv:1710.10196 (2017).

https://www.youtube.com/watch?v=XOxxPcy5Gr4

Examples

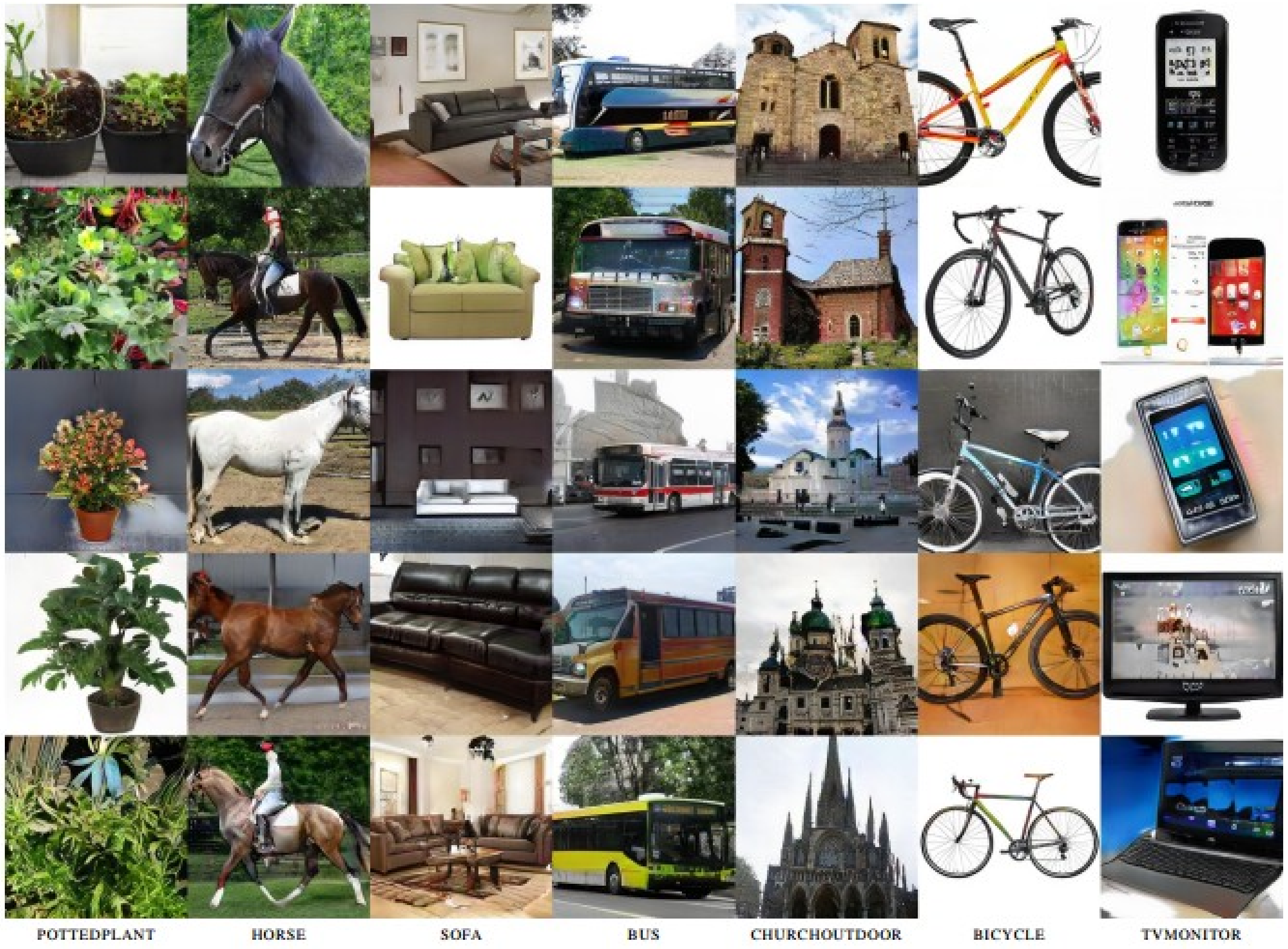
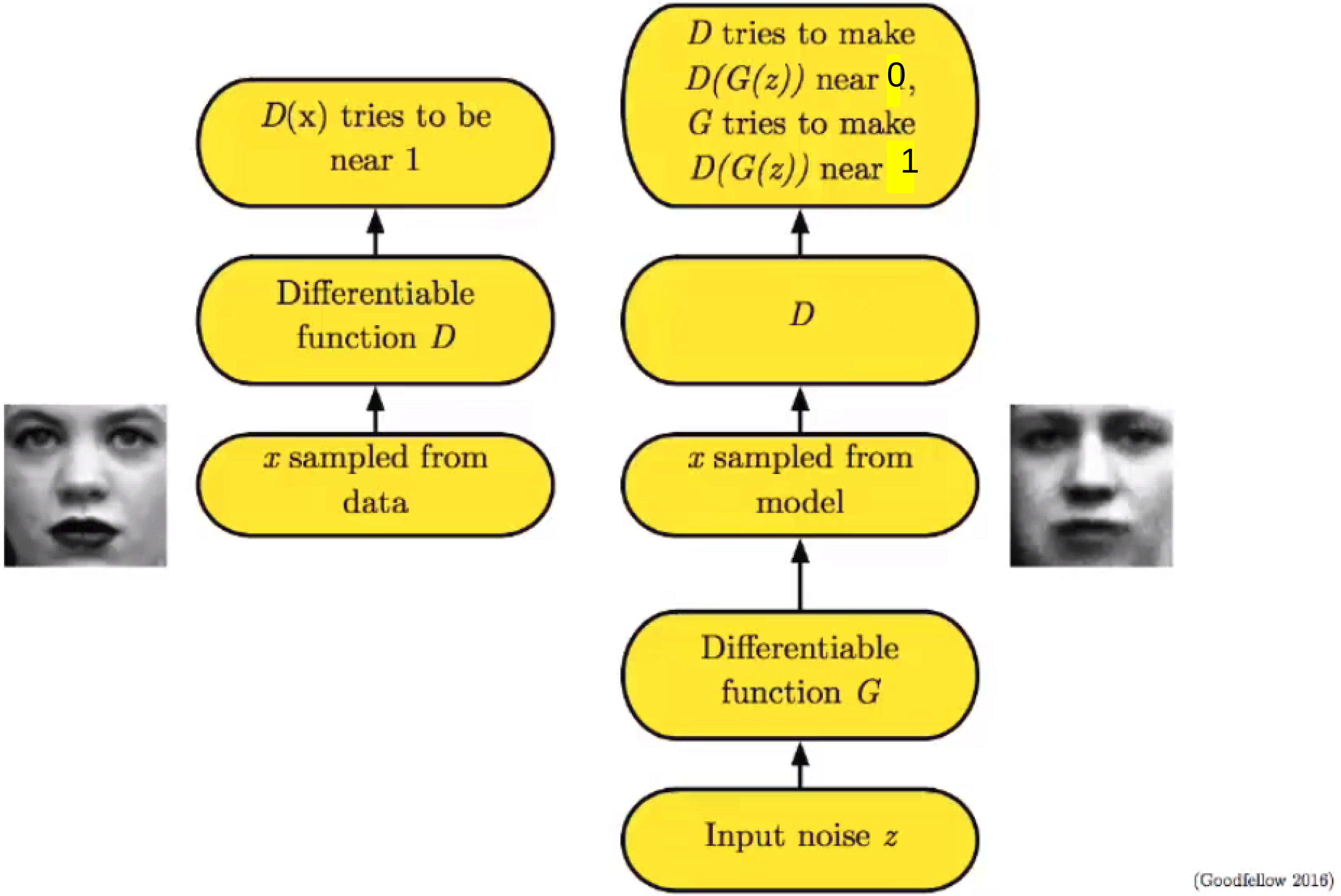


Figure 7: Selection of 256 × 256 images generated from different LSUN categories.

Adversarial Nets Framework



$D(\mathrm{x})$ tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

(Goodfellow 2016)

Generative Adversarial Networks

Minimax game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim q_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_x(z)}[\log(1 - D(G(z))]$$

$q_{data}(x)$ is the real data distribution on $\mathbb{R}^n$

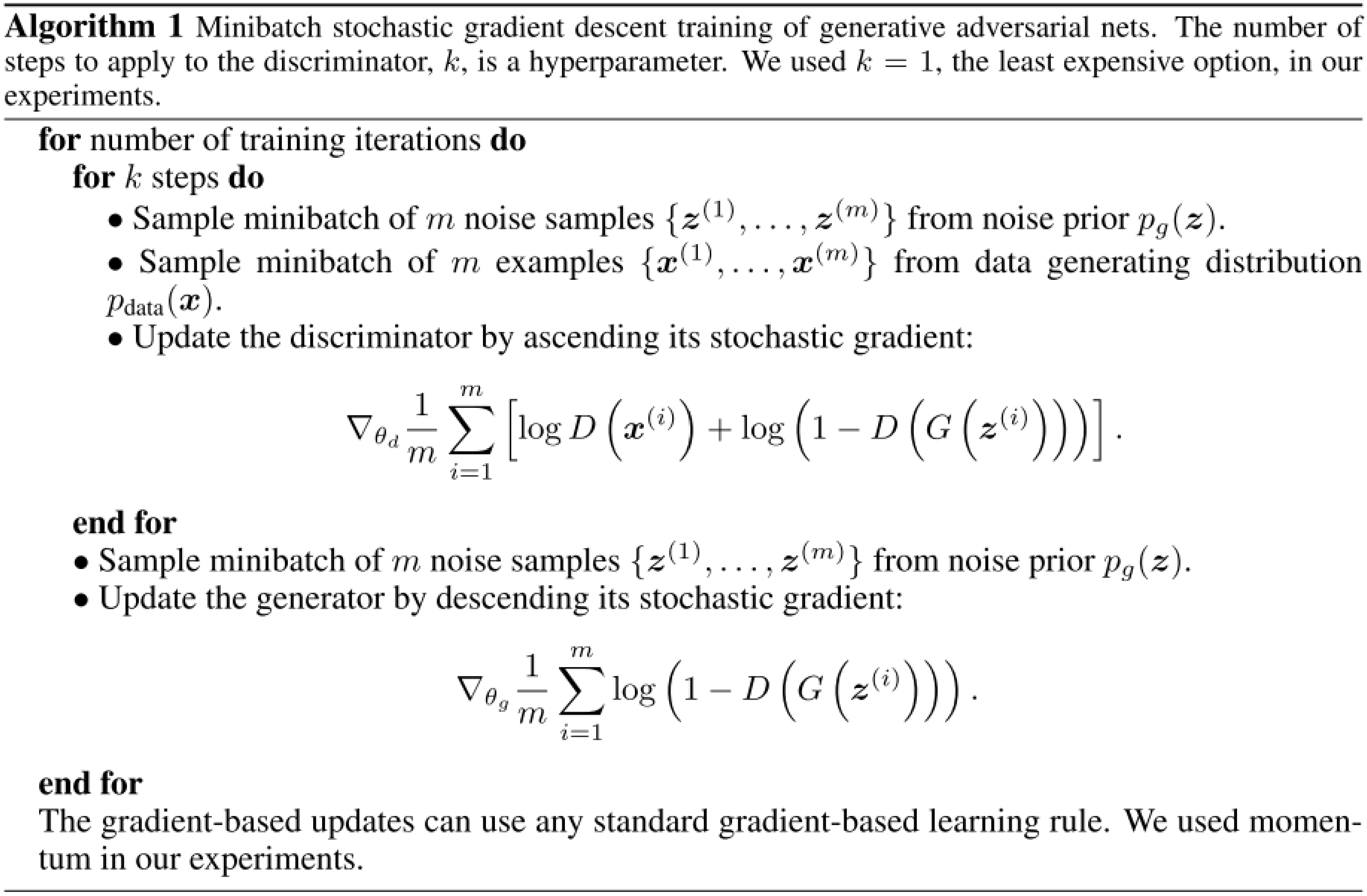
$z \in \mathbb{R}^m$ is a random noise variable drawn from $p_x(z)$

$p_x(z) : \mathcal{N}(0, I)$

$G : \mathbb{R}^m \to \mathbb{R}^n$

$D : \mathbb{R}^n \to [0,1]$

Training GAN



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Difficulties in training GANs

- Training unstablities
- Sensitivity to hyper-parameters and architectures choice
- Mode collapse
- Poorer results when trained on visually diverse datasets: CIFAR-10, STL-10, ImageNet

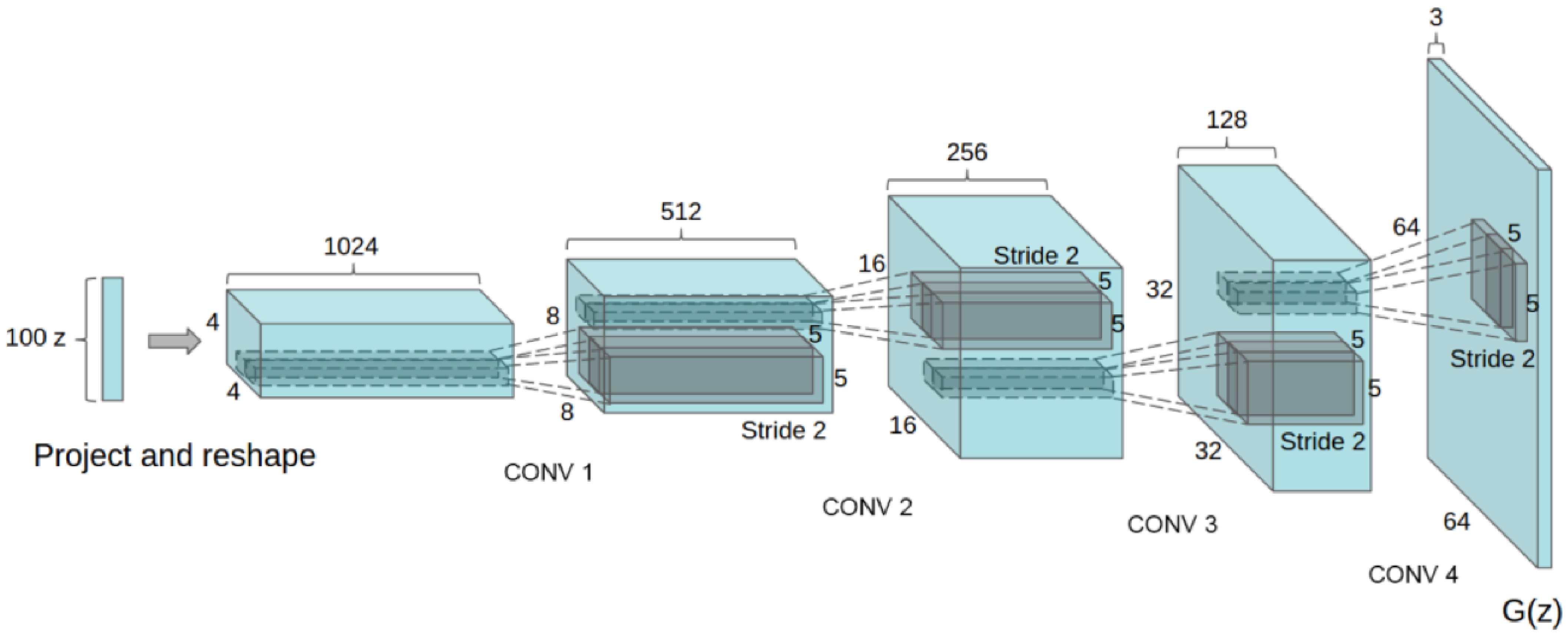
Spectral Norm GAN

- Enforce 1-Lipschitz condition on the discriminator to make training stable.
- Do this by dividing each weight matrix in the discriminator by its own largest singular value after each training step.
- Fast! The largest singular value can be approximated efficiently using an iterative algorithm.

Generative Adversarial Networks using Adaptive Convolution

Motivation

- Most GANs generators upsample low resolution feature maps toward higher resolution using fixed convolutions or resize-convolution



Motivation

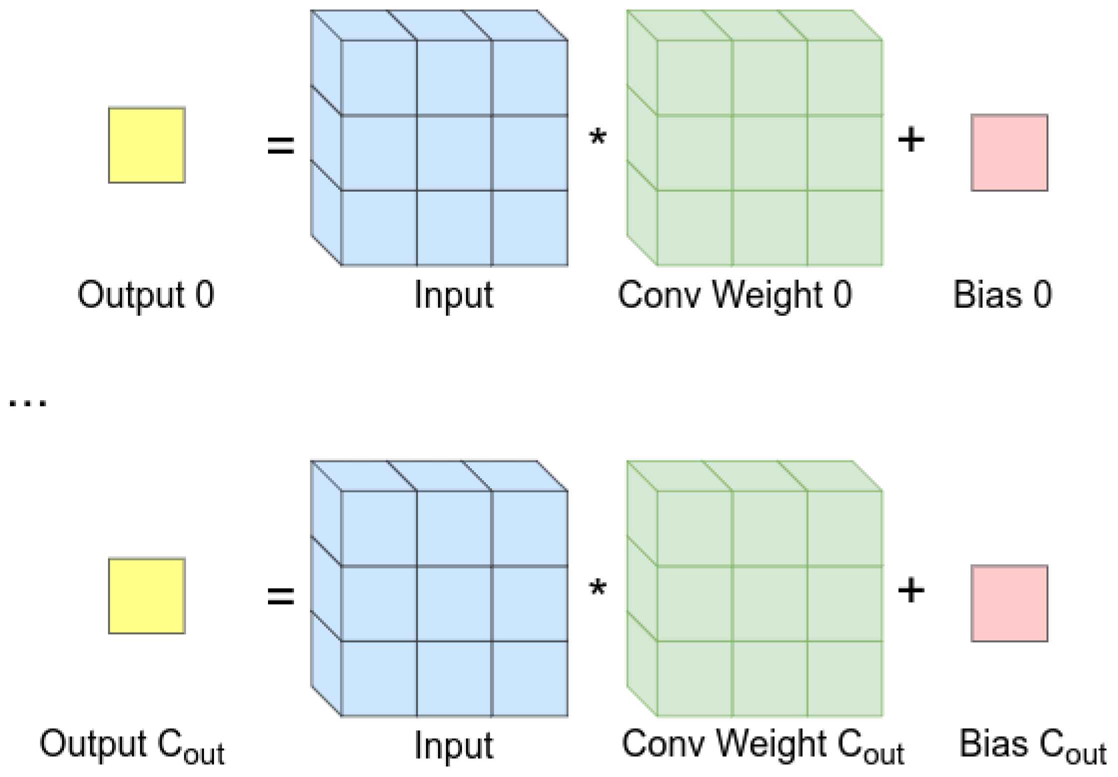
- Such fixed upsampling operations are unintuitive
- In visually diverse datasets, pixel locations can have different local contexts and belong to diverse object categories
- Consequently, different pixel locations should have different upsampling algorithms.

Motivation

- Instead of learning a fixed convolution for the upsampling of all pixels from the lower to the higher resolution feature map, one should learn to generate the algorithm (i.e. the convolution weights and biases) adaptively based on the local feature map at each pixel location.
- When a human draws something: the same thought process is used in the whole drawing but the style of each stroke should vary and depend on the local context around each pixel position.
- Our methods is orthogonal to other method and can be combined to yield better results.
- This kind of algorithm is called adaptive convolution.

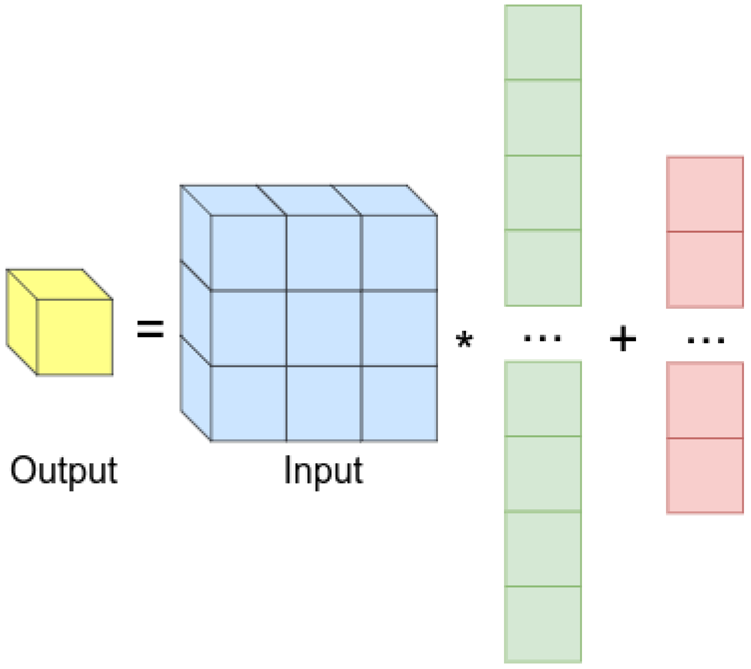
Normal Convolution

Consider a normal convolution: C_{in}, C_{out}, K_{filter} x K_{filter}



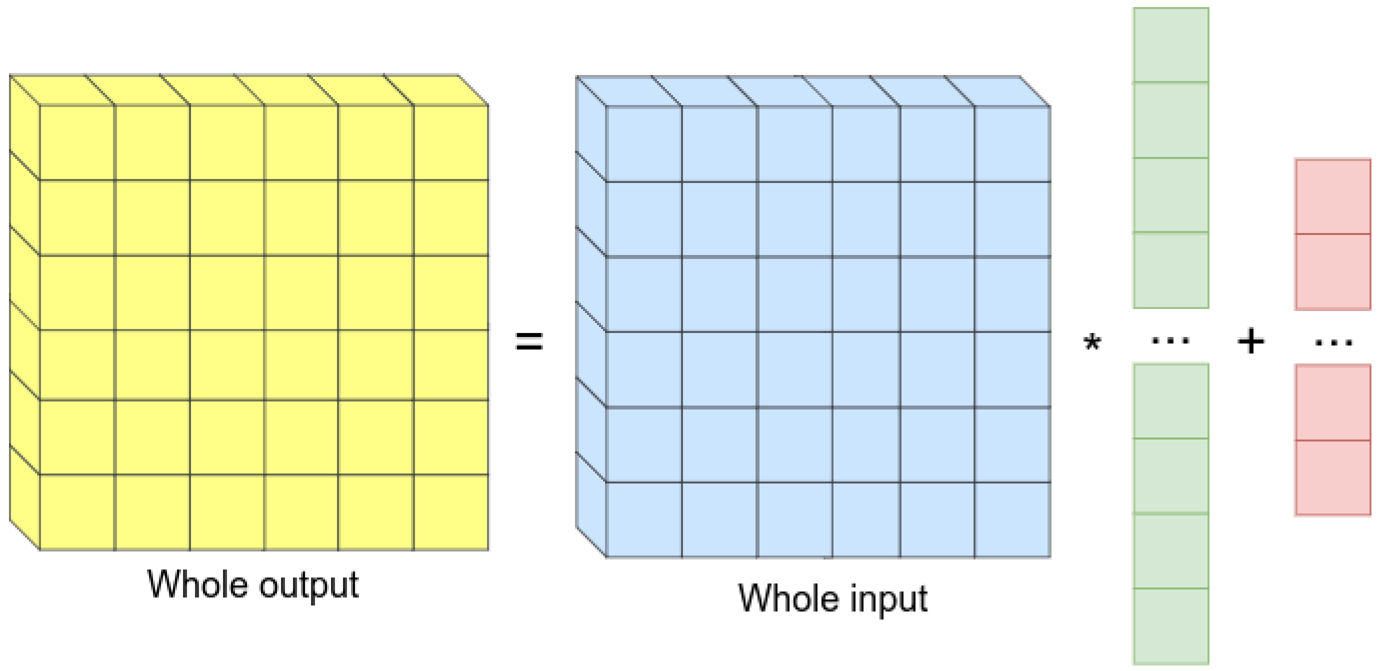
Normal Convolution

Which can be written as:



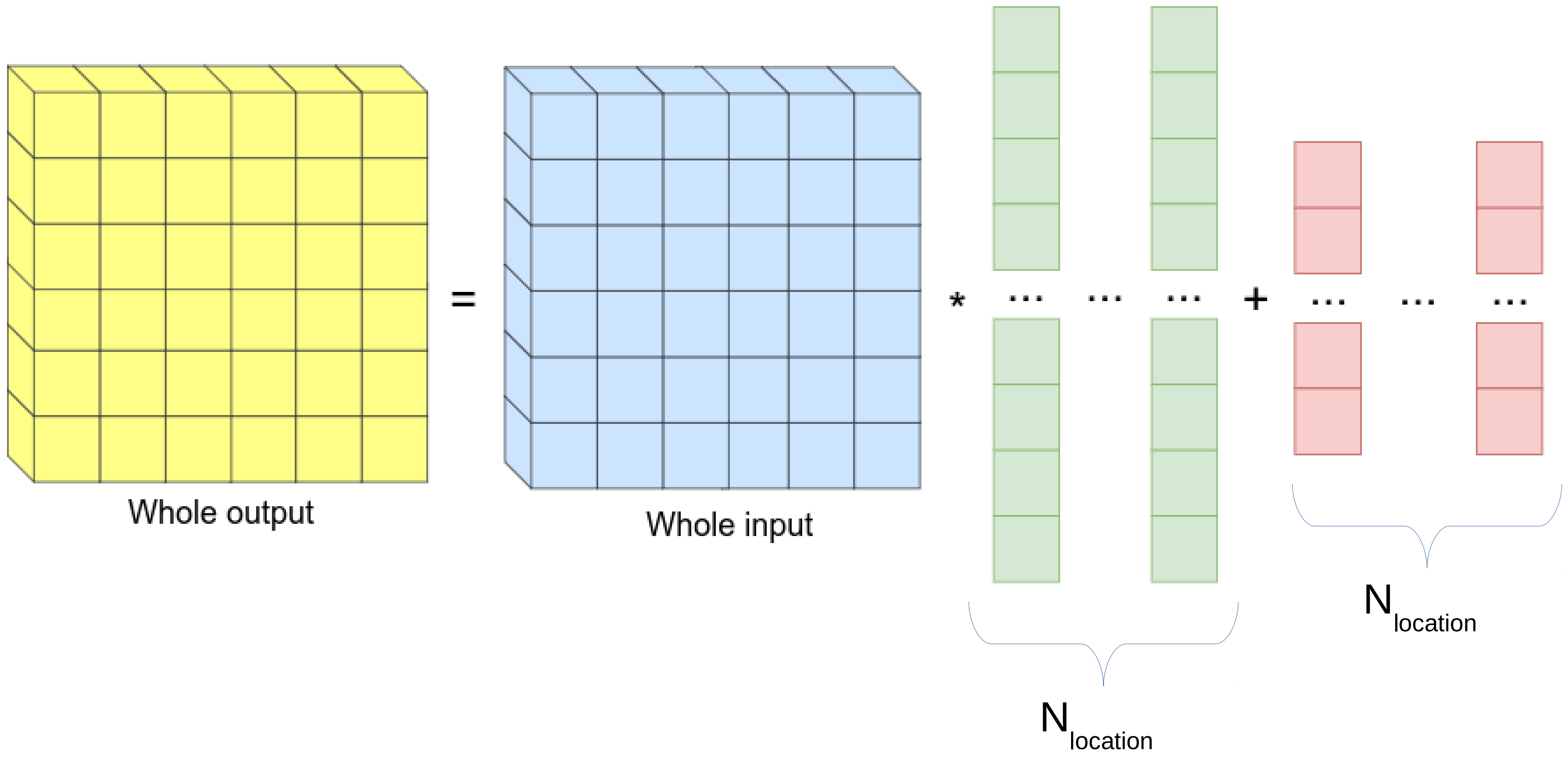
Normal Convolution

Shared weights and biases for all pixel locations.



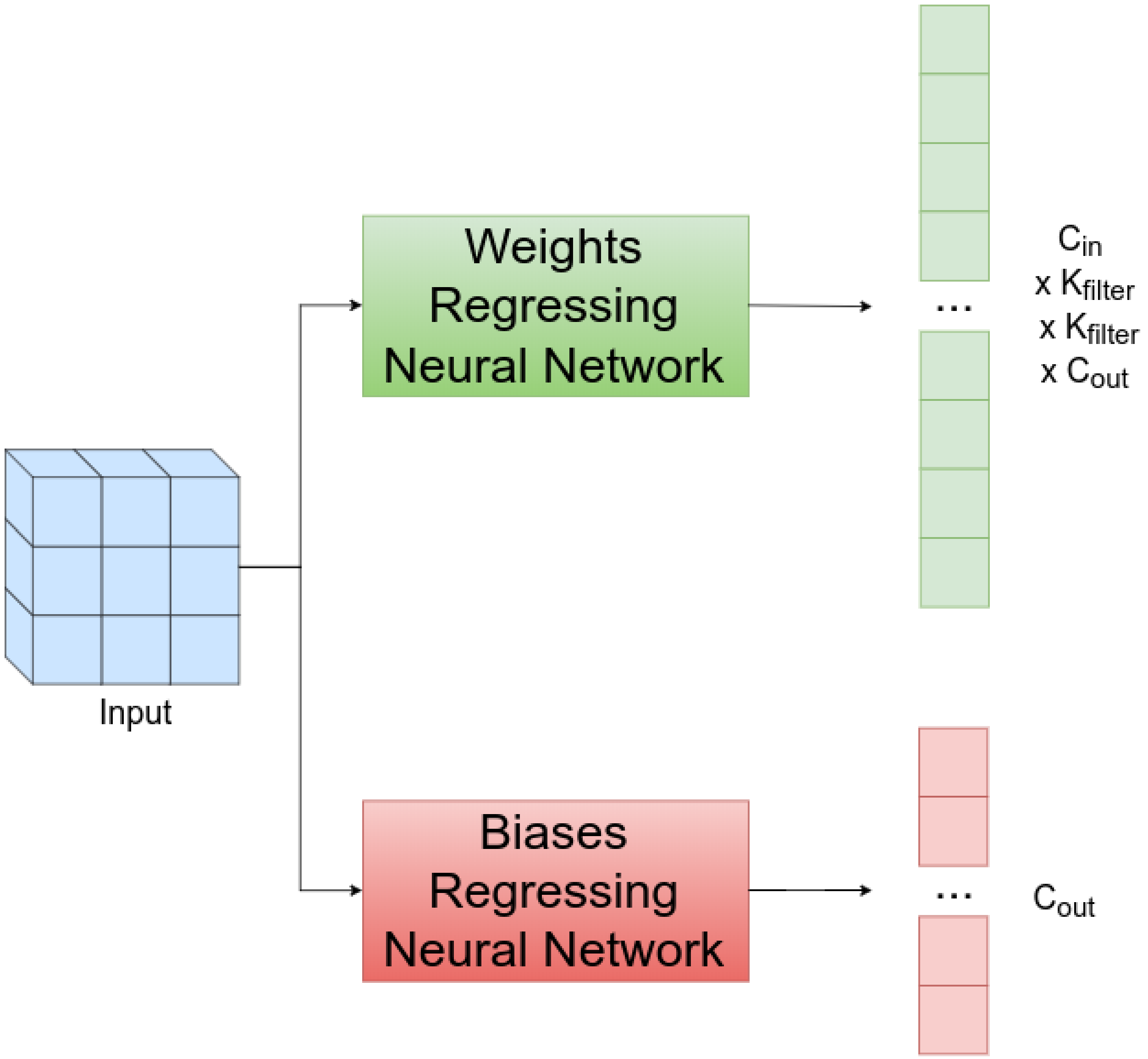
Adaptive Convolution

No weights and biases sharing. Learn weights and biases for each location instead. $N_{location}$ weights and biases pairs



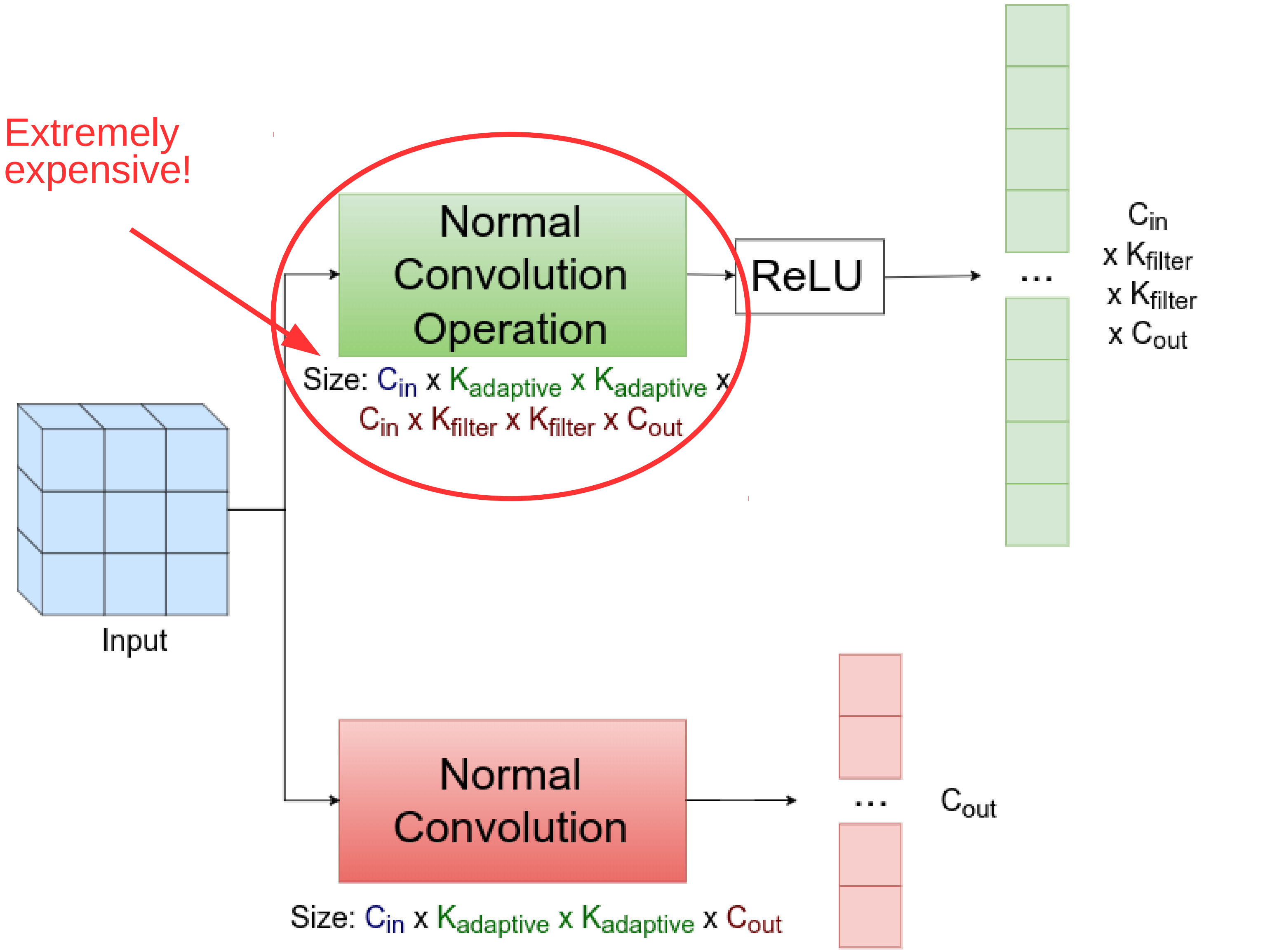
Adaptive Convolution

Use neural networks to learn to generate the weights and biases


Weights Regressing Neural Network
C_{in} x K_{filter} x K_{filter} x C_{out}
Input
Biases Regressing Neural Network
C_{out}

Adaptive Convolution

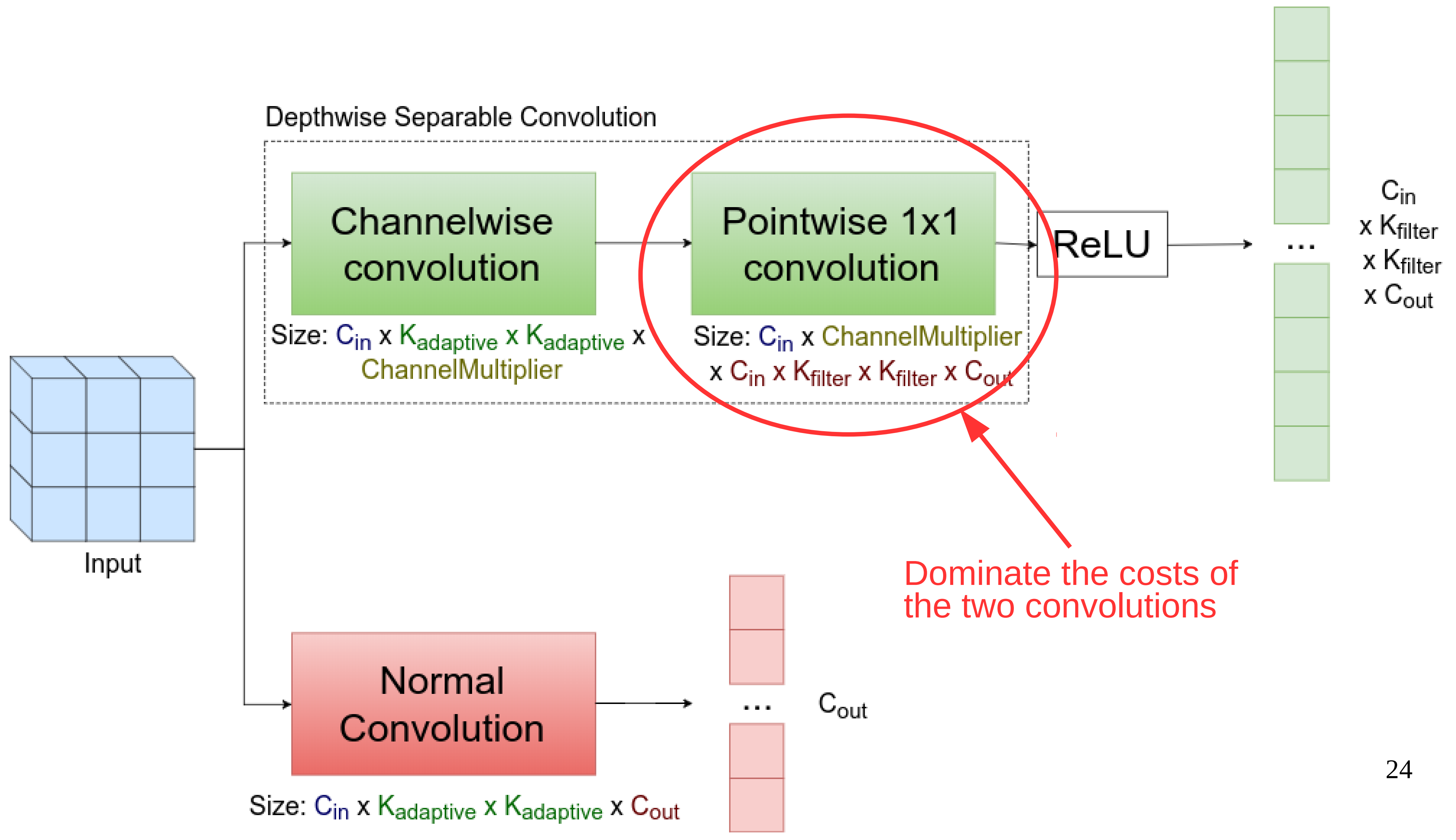
Use two normal convolutions to learn to generate the weights and biases: $K_{adaptive}$ = Local Window size to regress the weights and biases. Naive version:



Extremely expensive!
Normal Convolution Operation
ReLU
Size: $C_{in} \times K_{adaptive} \times K_{adaptive} \times C_{in} \times K_{filter} \times K_{filter} \times C_{out}$
$C_{in} \times K_{filter} \times K_{filter} \times C_{out}$
...
Input
Normal Convolution
Size: $C_{in} \times K_{adaptive} \times K_{adaptive} \times C_{out}$
C_{out}

Adaptive Convolution

Reduce memory and computation cost by using depthwise separable convolution.



Depthwise Separable Convolution
Channelwise convolution
Size: C_{in} x $K_{adaptive}$ x $K_{adaptive}$ x ChannelMultiplier
Pointwise 1x1 convolution
Size: C_{in} x ChannelMultiplier x C_{in} x K_{filter} x K_{filter} x C_{out}
ReLU
C_{in} x K_{filter} x K_{filter} x C_{out}
Input
Dominate the costs of the two convolutions
Normal Convolution
Size: C_{in} x $K_{adaptive}$ x $K_{adaptive}$ x C_{out}
C_{out}

Depthwise Separable Convolution

- Channelwise convolution: Convolution that is performed independently on each input channel.
- ChannelMultiplier = Number of temporary output channels for each input channel.
- Pointwise convolution: 1x1 convolution
- No nonlinearity between the two convolution

Depthwise Separable Convolution

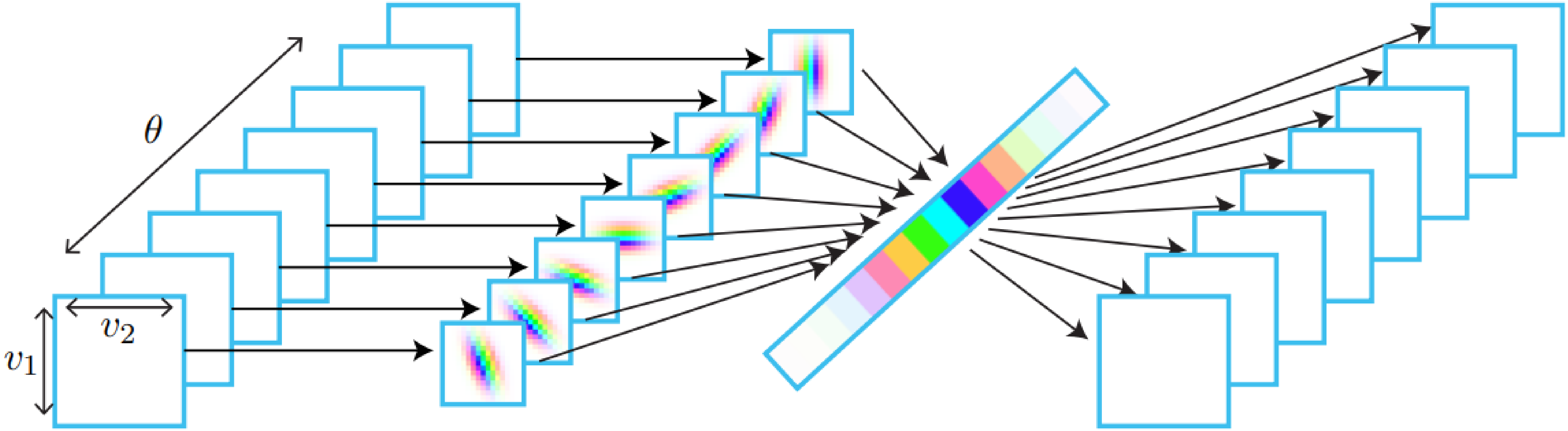
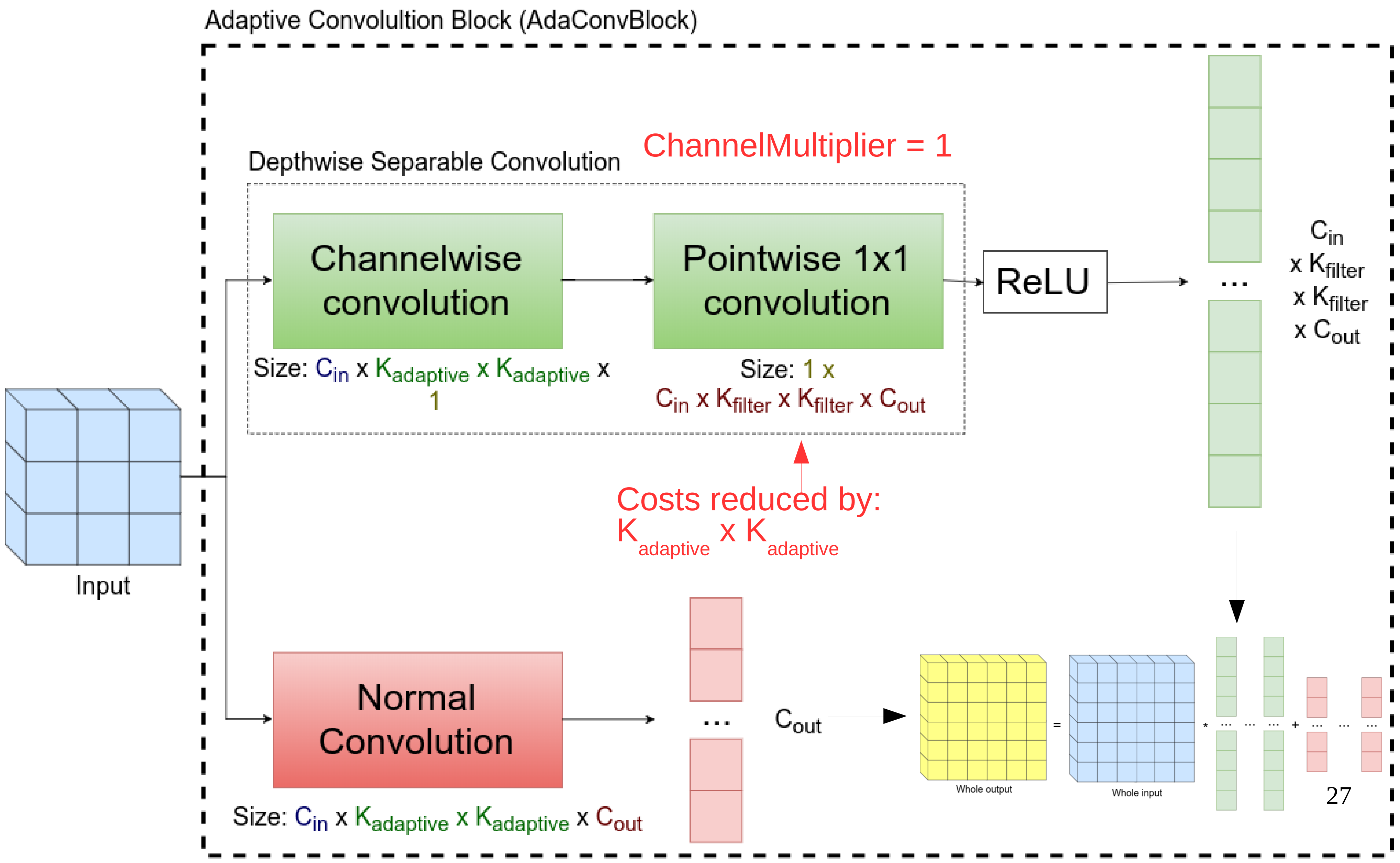


Image source: Laurent Sifre and Stephane Mallat. Rigid-motion scattering for texture classification. ´ arXiv preprint arXiv:1403.1687, 2014.

Adaptive Convolution



Adaptive Convolultion Block (AdaConvBlock)
Depthwise Separable Convolution
ChannelMultiplier = 1
Channelwise convolution
Pointwise 1x1 convolution
ReLU
Size: C_{in} x $K_{adaptive}$ x $K_{adaptive}$ x 1
Size: 1 x C_{in} x K_{filter} x K_{filter} x C_{out}
C_{in} x K_{filter} x K_{filter} x C_{out}
Costs reduced by: $K_{adaptive}$ x $K_{adaptive}$
Input
Normal Convolution
Size: C_{in} x $K_{adaptive}$ x $K_{adaptive}$ x C_{out}
C_{out}
Whole output
Whole input

Design Choices

- No Batch Normalization: No reasons to believe that the adaptive weights and biases should follow any distribution
- No non-linearity after biases regress: Doing so will limit the output range of AdaConvBlock $\rightarrow$ Unintended effect
- ReLU after weights regression: from experiments

Experiments

- Take a weak baseline generator G in a GAN, replace normal convolutions with AdaConvBlocks.
  - In a progressive manner

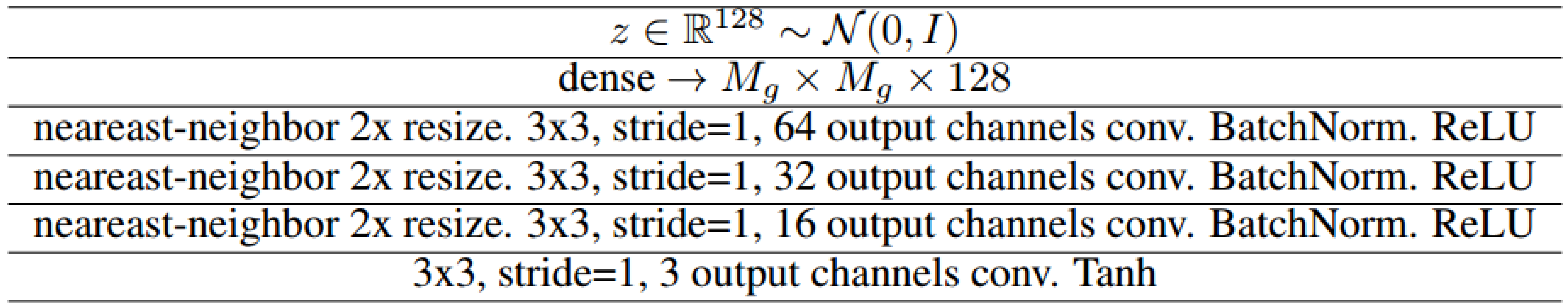


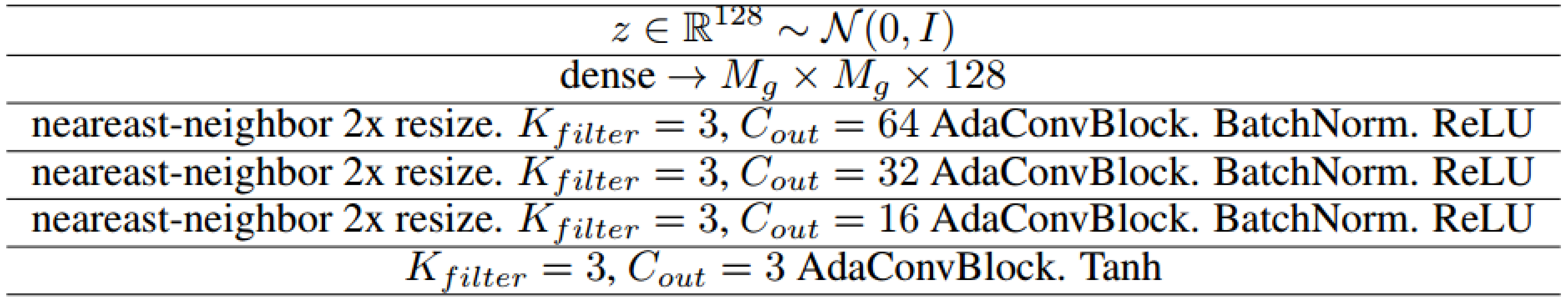
Table 1: Architecture of the baseline generator. $M_g = 4$ for CIFAR-10 and $M_g = 6$ for STL-10.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense $\rightarrow M_g \times M_g \times 128$
neareast-neighbor 2x resize. 3x3, stride=1, 64 output channels conv. BatchNorm. ReLU
neareast-neighbor 2x resize. 3x3, stride=1, 32 output channels conv. BatchNorm. ReLU
neareast-neighbor 2x resize. 3x3, stride=1, 16 output channels conv. BatchNorm. ReLU
3x3, stride=1, 3 output channels conv. Tanh

Experiments

- Architecture of generator with all normal convolutions replaced with AdaConvBlock
- No hyper parameters tuning.

Table 2: Architecture of AdaGAN. $M_g = 4$ for CIFAR-10 and $M_g = 6$ for STL-10. $K_{adaptive}$ for each AdaConvBlock are not specified.



$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense $\rightarrow M_g \times M_g \times 128$
neareast-neighbor 2x resize. $K_{filter} = 3$, $C_{out} = 64$ AdaConvBlock. BatchNorm. ReLU
neareast-neighbor 2x resize. $K_{filter} = 3$, $C_{out} = 32$ AdaConvBlock. BatchNorm. ReLU
neareast-neighbor 2x resize. $K_{filter} = 3$, $C_{out} = 16$ AdaConvBlock. BatchNorm. ReLU
$K_{filter} = 3$, $C_{out} = 3$ AdaConvBlock. Tanh

Architecture names: AdaGAN-1-3x3, AdaGAN-2-3x3, AdaGAN-3-3x3, AdaGAN-3x3 AdaGAN-5x5, AdaGAN-7x7

Experiments

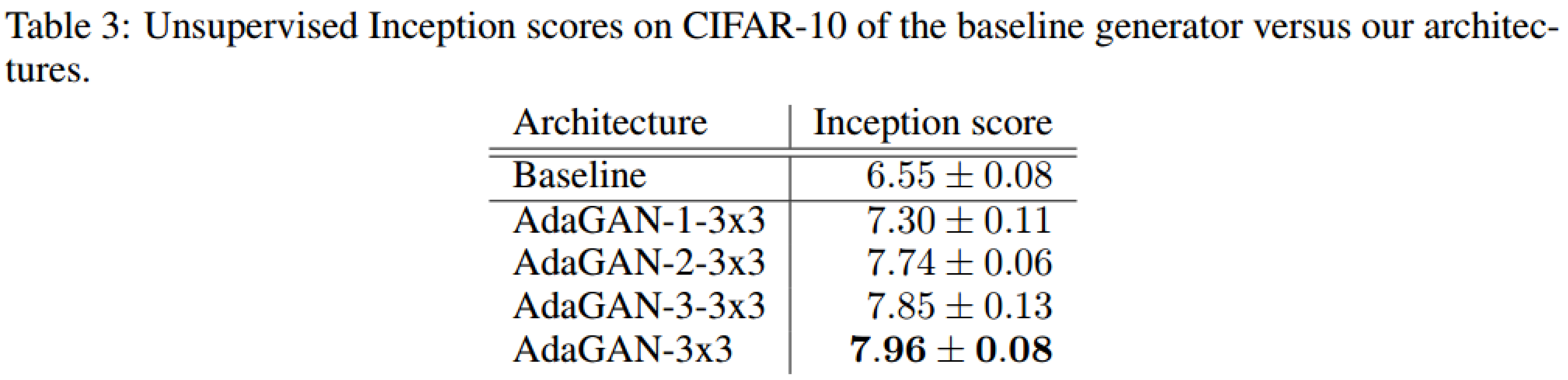


Table 3: Unsupervised Inception scores on CIFAR-10 of the baseline generator versus our architectures.

Architecture	Inception score
Baseline	6.55 ± 0.08
AdaGAN-1-3x3	7.30 ± 0.11
AdaGAN-2-3x3	7.74 ± 0.06
AdaGAN-3-3x3	7.85 ± 0.13
AdaGAN-3x3	$\mathbf{7.96 \pm 0.08}$

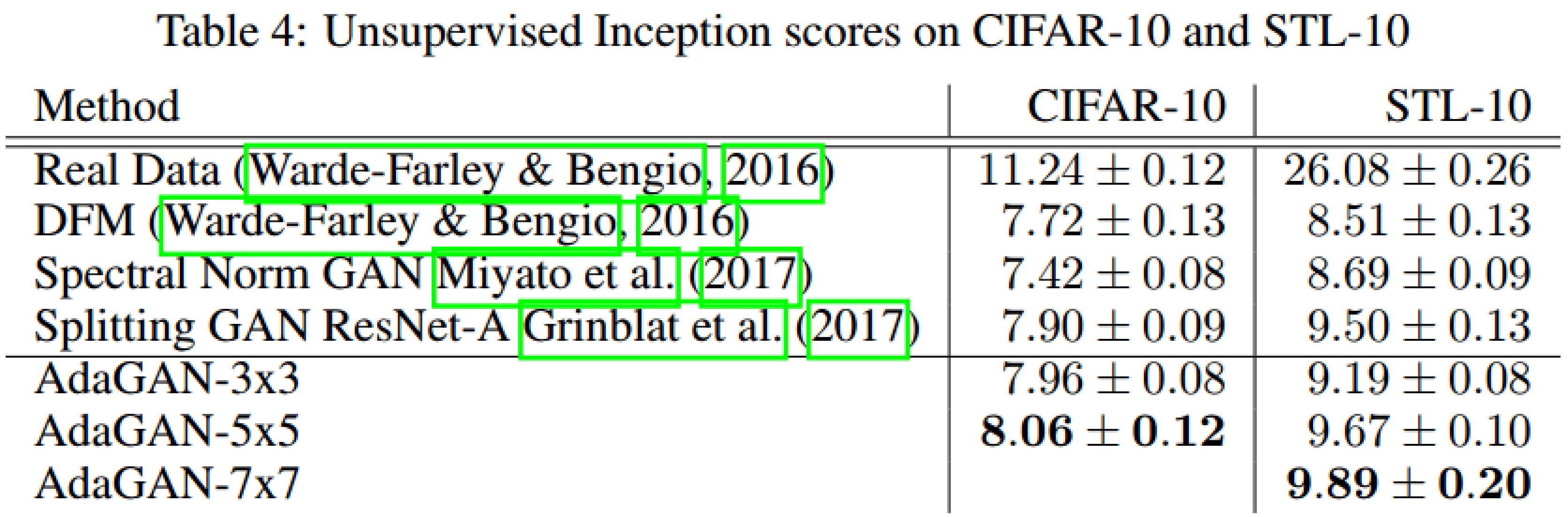


Table 4: Unsupervised Inception scores on CIFAR-10 and STL-10

Method	CIFAR-10	STL-10
Real Data (Warde-Farley & Bengio, 2016)	11.24 ± 0.12	26.08 ± 0.26
DFM (Warde-Farley & Bengio, 2016)	7.72 ± 0.13	8.51 ± 0.13
Spectral Norm GAN Miyato et al. (2017)	7.42 ± 0.08	8.69 ± 0.09
Splitting GAN ResNet-A Grinblat et al. (2017)	7.90 ± 0.09	9.50 ± 0.13
AdaGAN-3x3	7.96 ± 0.08	9.19 ± 0.08
AdaGAN-5x5	$\mathbf{8.06 \pm 0.12}$	9.67 ± 0.10
AdaGAN-7x7		$\mathbf{9.89 \pm 0.20}$

Generated Samples

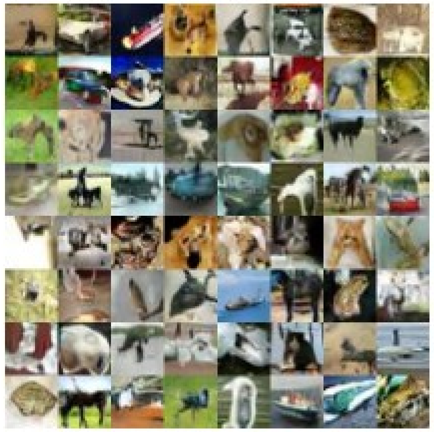


Figure 2: Samples generated by AdaGAN-3x3 on CIFAR-10 dataset

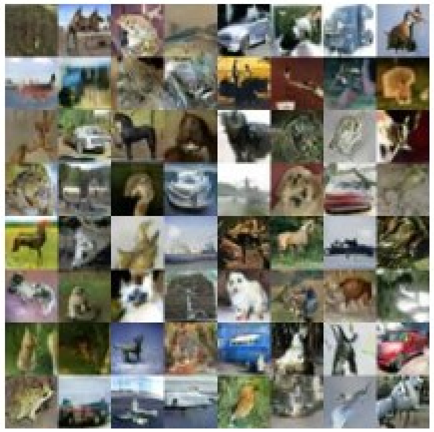


Figure 3: Samples generated by AdaGAN-5x5 on CIFAR-10 dataset

Generate Samples

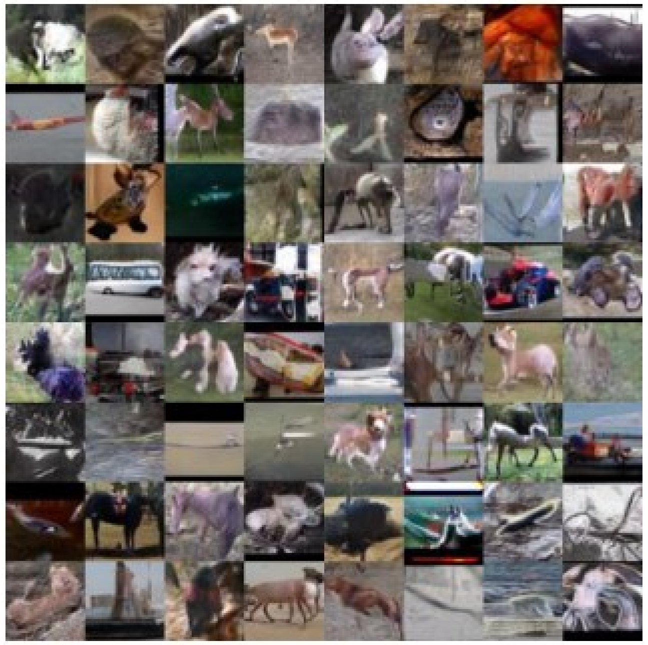


Figure 4: Samples generated by AdaGAN-3x3 on STL-10 dataset

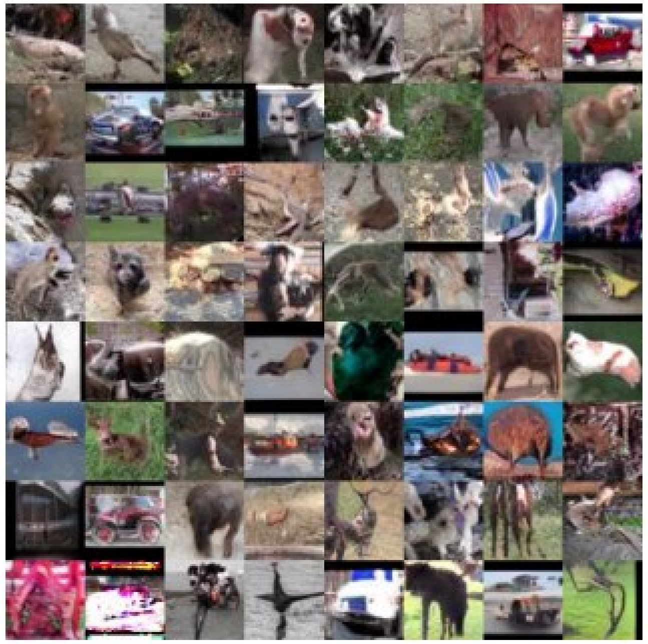


Figure 5: Samples generated by AdaGAN-5x5 on STL-10 dataset

Generated Samples

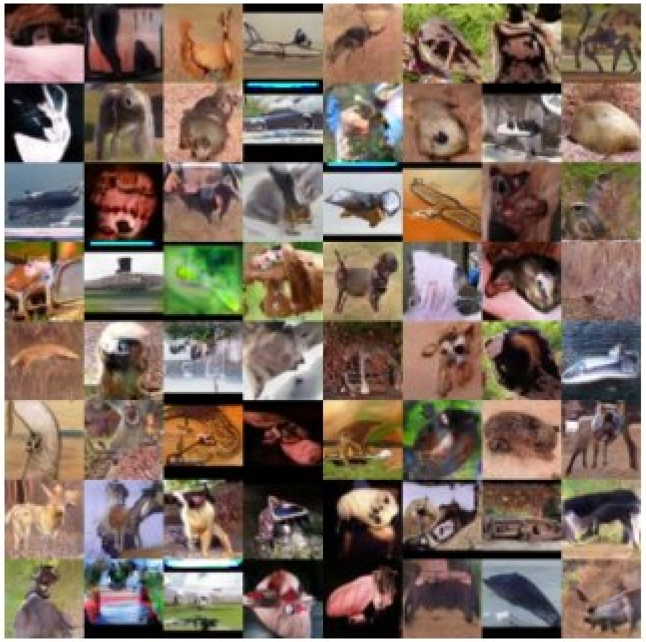


Figure 6: Samples generated by AdaGAN-7x7 on STL-10 dataset

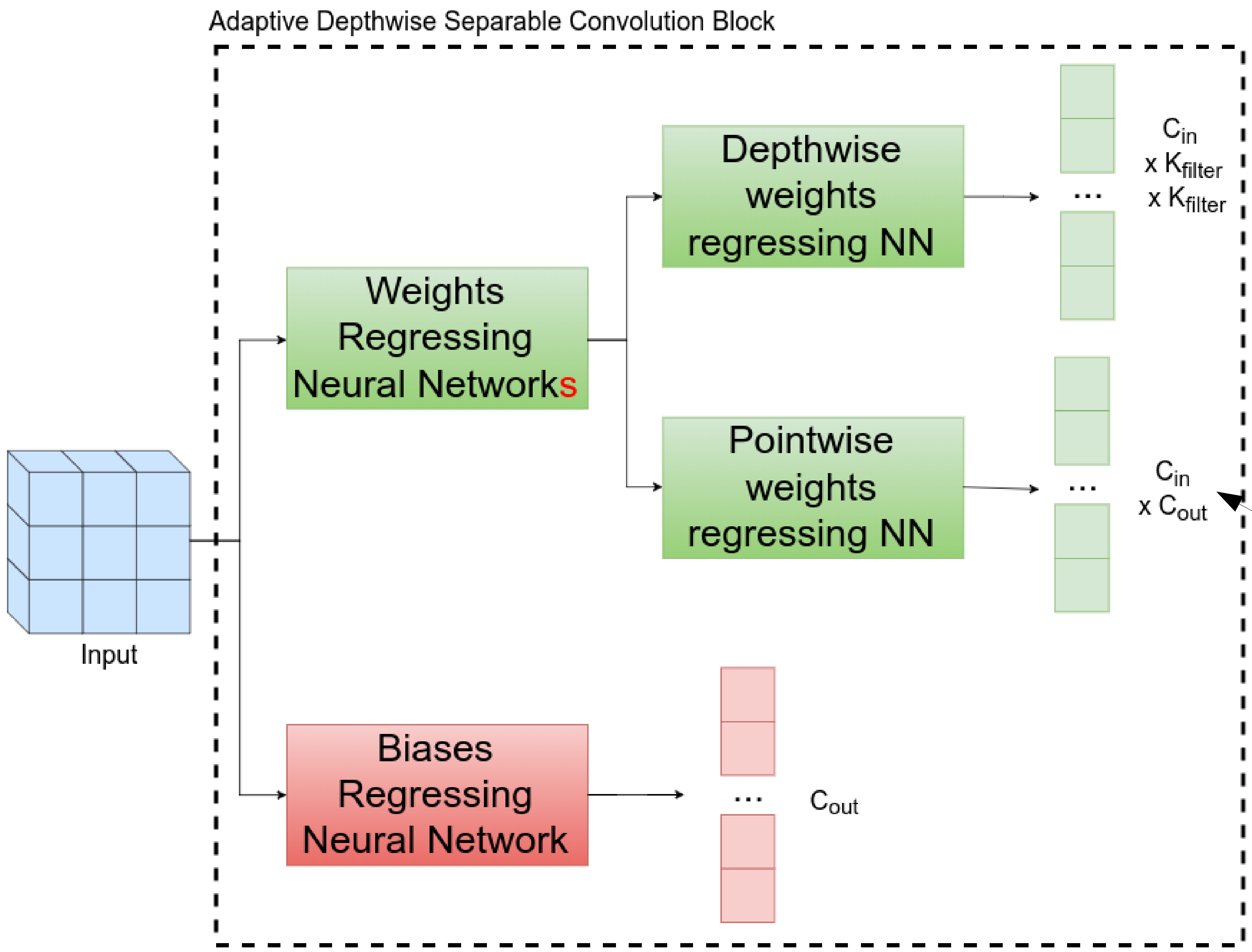
Future Directions

- Explore the effectiveness of AdaConvBlock in other tasks.
  - Will it help?
    - No improvement in datasets like MNIST.
    - CIFAR-10, CIFAR-100 and ImageNet?
- Make AdaConvBlock a feasible drop-in replacement for normal convolution.
  - Costs scales cubically with number of input channels
  - Need to further improve the memory and computation costs

Reduce memory and computation costs of AdaConvBlock

- Adaptive Depthwise Separable Convolution Block: Learn to generate weights and biases for a depthwise separable convolution instead of a normal convolution.

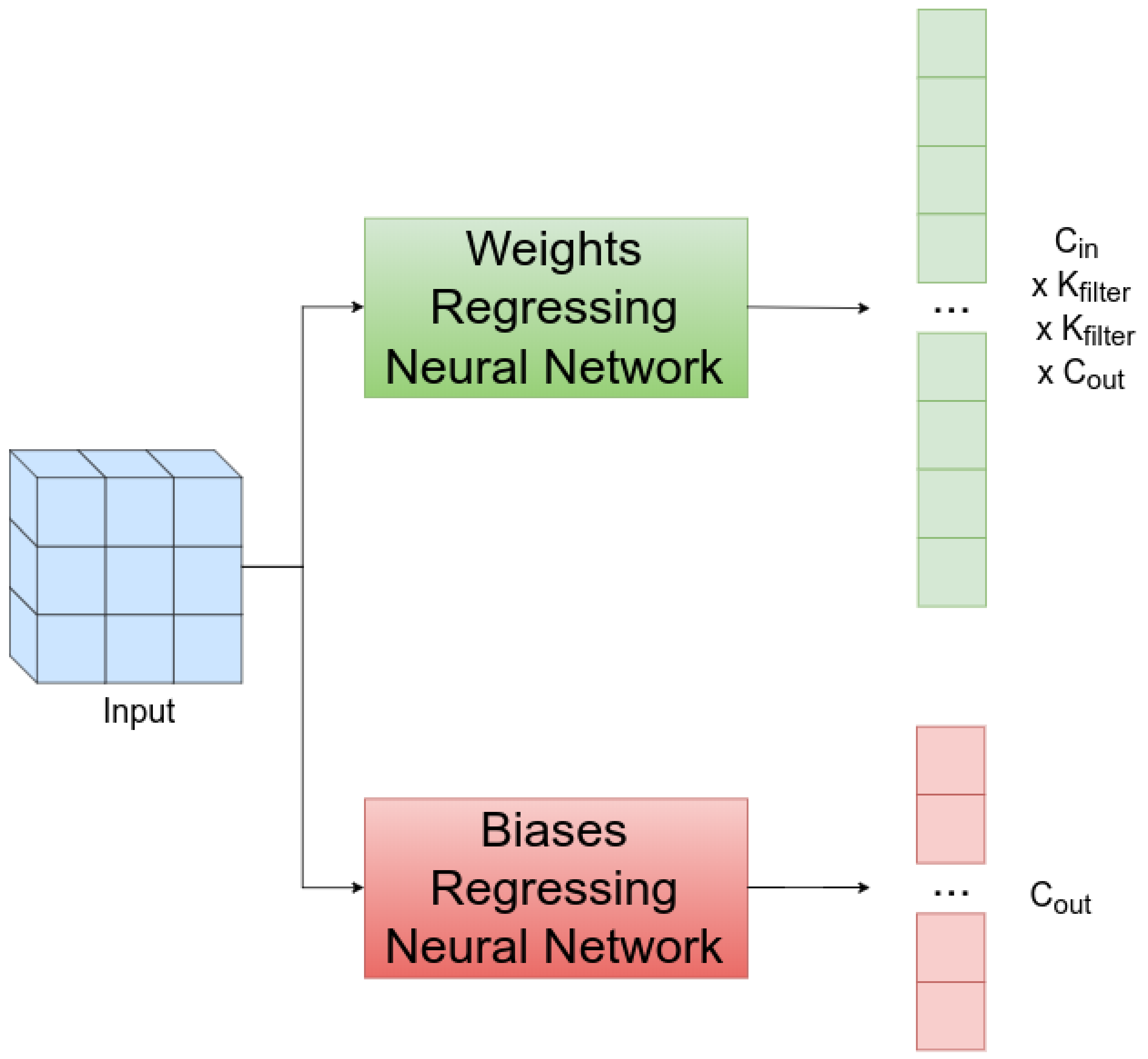
Reduce memory and computation costs of AdaConvBlock



Adaptive Depthwise Separable Convolution Block
Input
Weights Regressing Neural Networks
Depthwise weights regressing NN
Pointwise weights regressing NN
Biases Regressing Neural Network
C_{in} x K_{filter} x K_{filter}
C_{in} x C_{out}
C_{out}

Costs additionally reduced by: K_{filter} x K_{filter}

Reduce memory and computation costs of AdaConvBlock

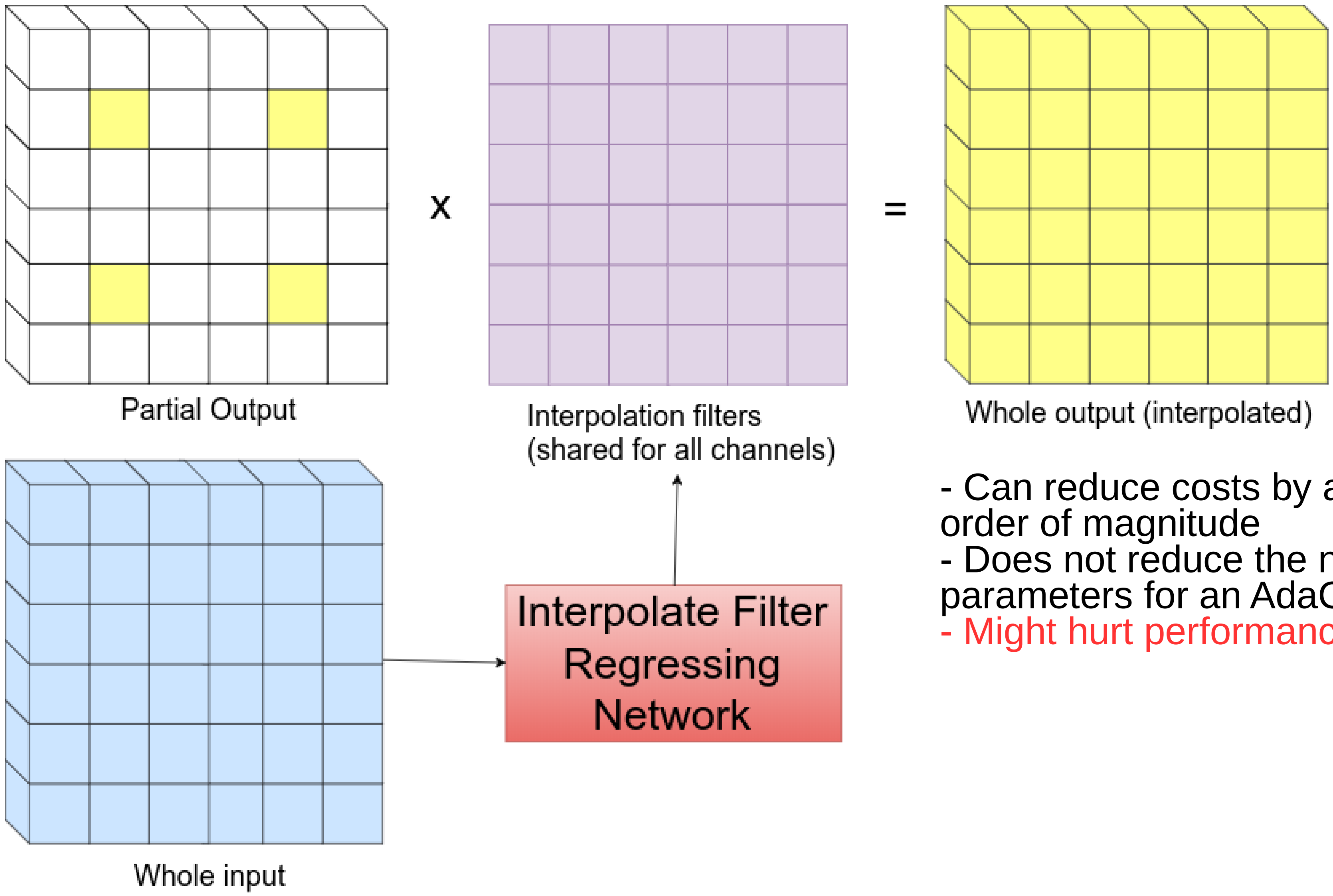


Input
Weights Regressing Neural Network
$C_{in} \times K_{filter} \times K_{filter} \times C_{out}$
...
Biases Regressing Neural Network
C_{out}
...

Reduce memory and computation costs of AdaConvBlock

- Exploit locality:
  - Sparsely compute adaptive weights and biases at some certain locations.
  - Use neural networks to learn the interpolate filters for every location. At each location, one single interpolation filter is used for all channels.
  - Interpolate to get the weights and biases for all locations.

Reduce memory and computation costs of AdaConvBlock



Partial Output
x
Interpolation filters
(shared for all channels)
=
Whole output (interpolated)
Interpolate Filter Regressing Network
Whole input

- Can reduce costs by around an order of magnitude
- Does not reduce the number of parameters for an AdaConvBlock
- Might hurt performance greatly