UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

# Deep Learning

## Week 3 : CNN's (Part 2)

TRANSFORMATEC

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

# Deep Learning

## Week 3 : CNN's (Part 2)

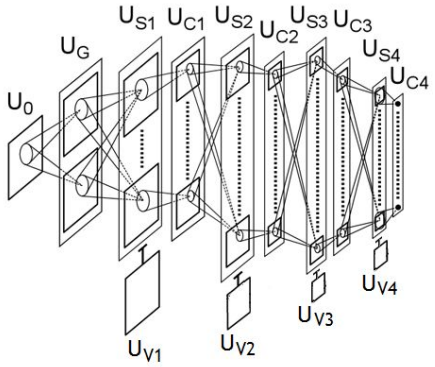
TRANSFORMATEC

# Summary of the Week:

1) AlexNet vs VGG vs ResNet
2) Feature Spaces in CNNs
3) U-Nets and De-Convolution
4) Preamble to Generative Models

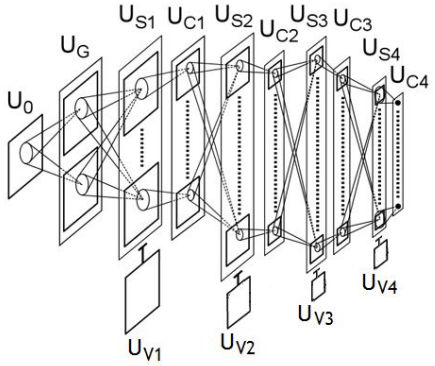
# Deep Architectures

Neurocognitron (Fukushima, 1980)

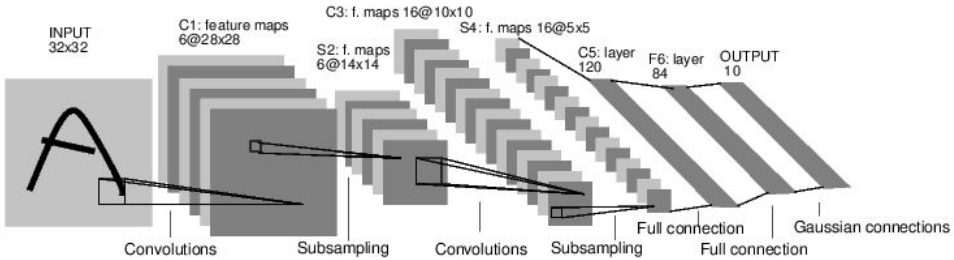


# Deep Architectures

Neurocognitron (Fukushima, 1980)

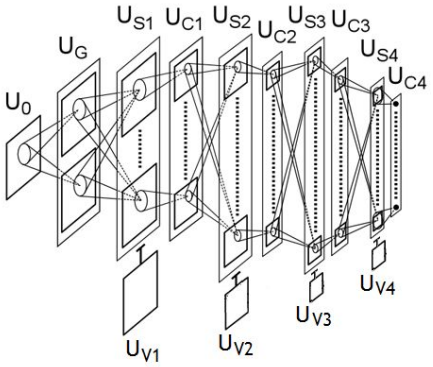



LeNet,1989

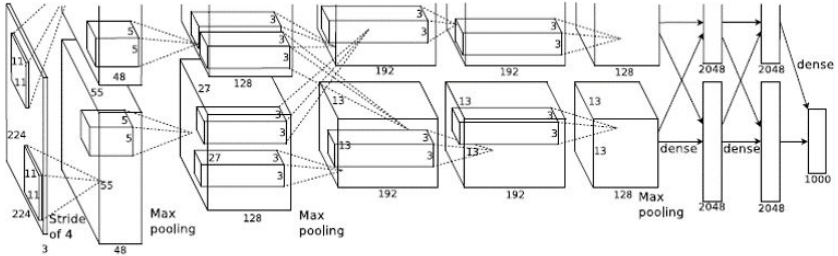


# Deep Architectures

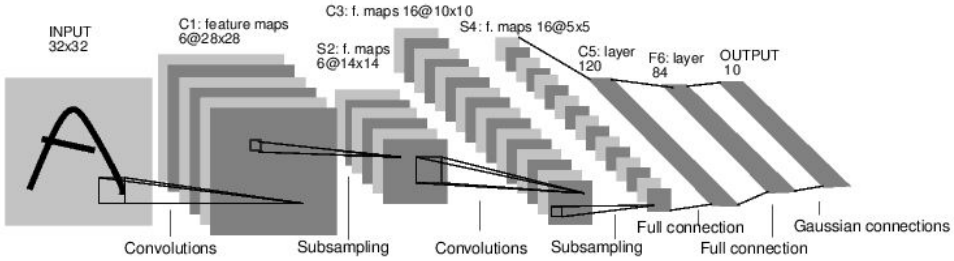
Neurocognitron (Fukushima, 1980)




AlexNet, 2011



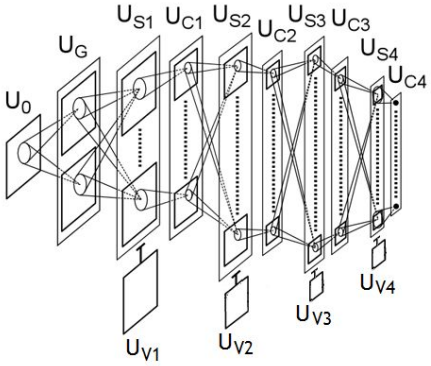

LeNet,1989



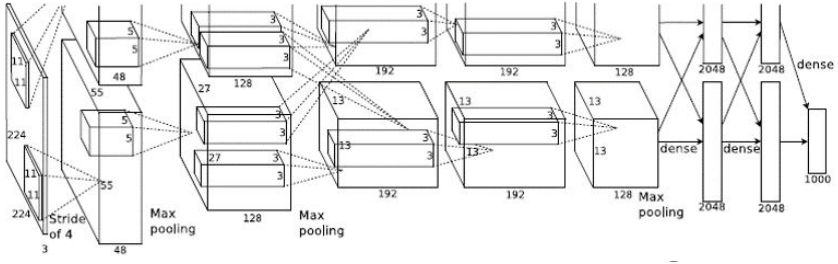
# Deep Architectures

VGGNet (Simonyan & Zisserman, 2014)


Neurocognitron (Fukushima, 1980)

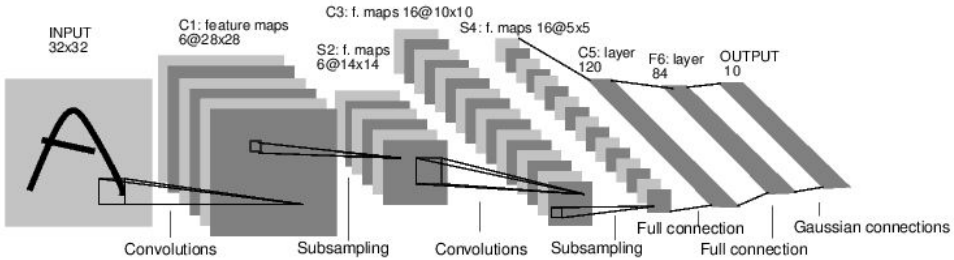


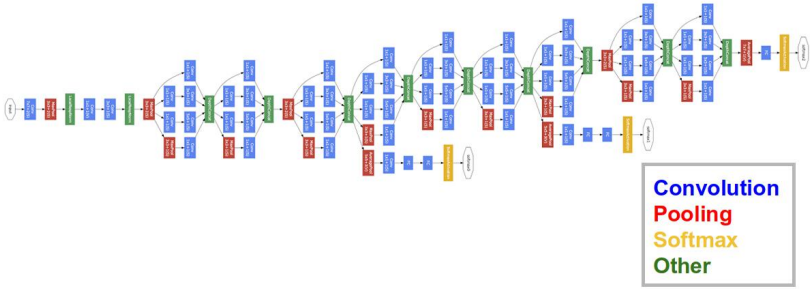

AlexNet, 2011




GoogLeNet, 2014

LeNet,1989

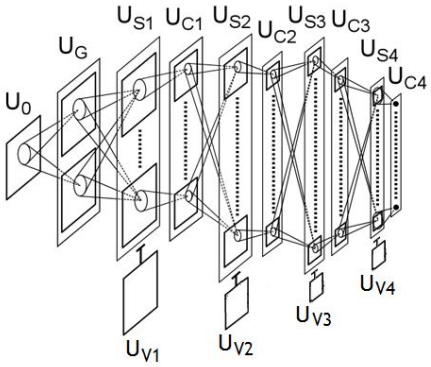


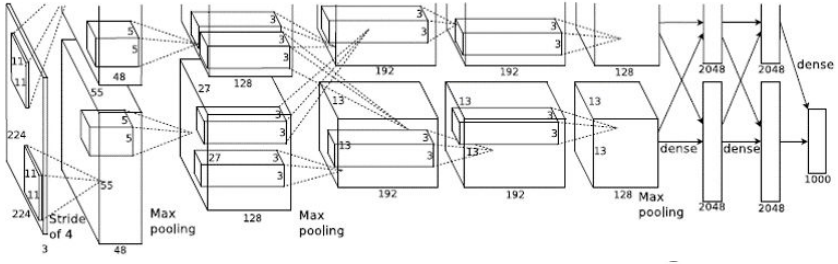
# Deep Architectures

VGGNet (Simonyan & Zisserman, 2014)

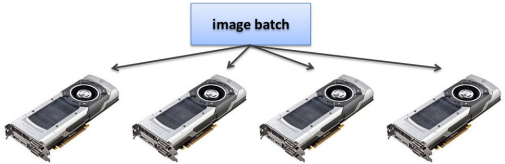

Neurocognitron (Fukushima, 1980)




AlexNet, 2011

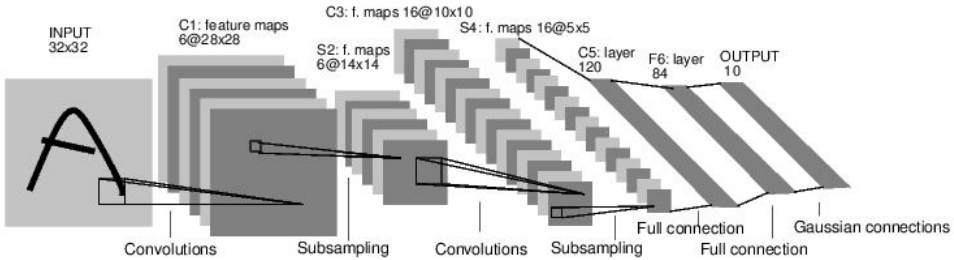


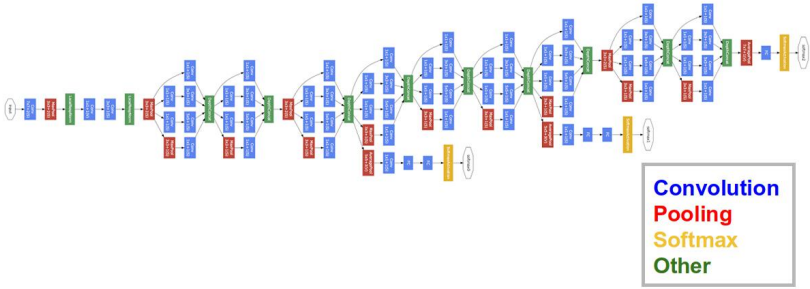

(Ridiculously huge amount of parameters)



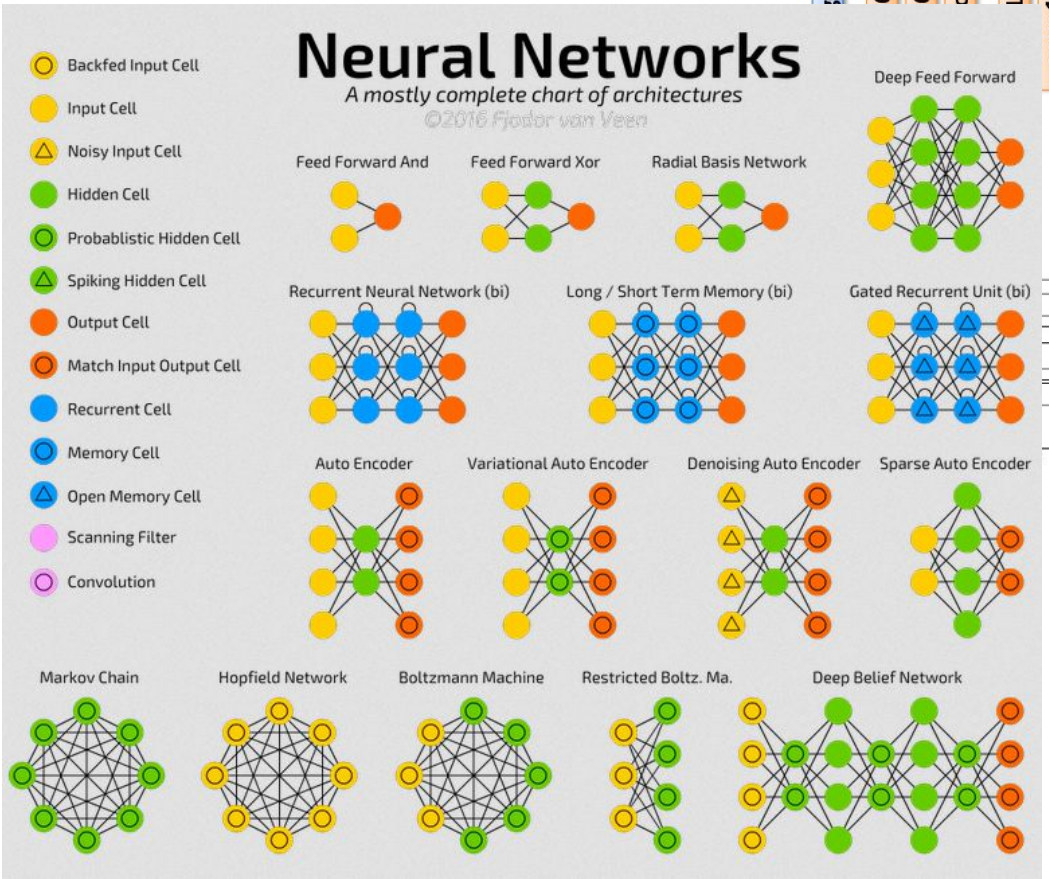

GoogLeNet, 2014

LeNet,1989



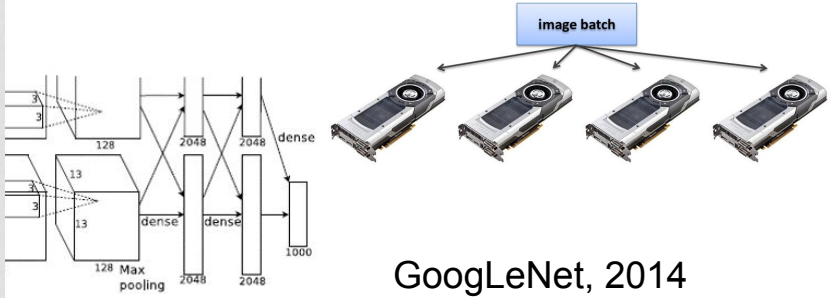



# Deep Architectures




VGGNet (Simonyan & Zisserman, 2014)

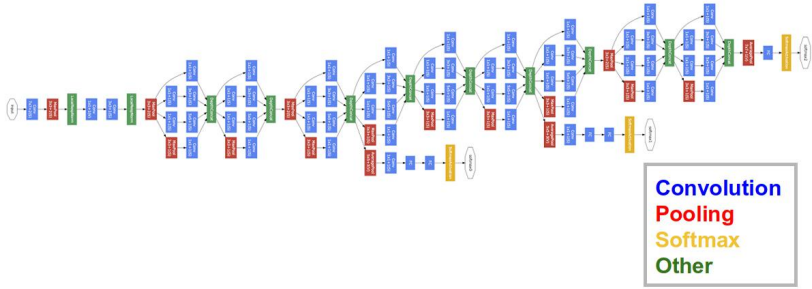

(Ridiculously huge amount of parameters)



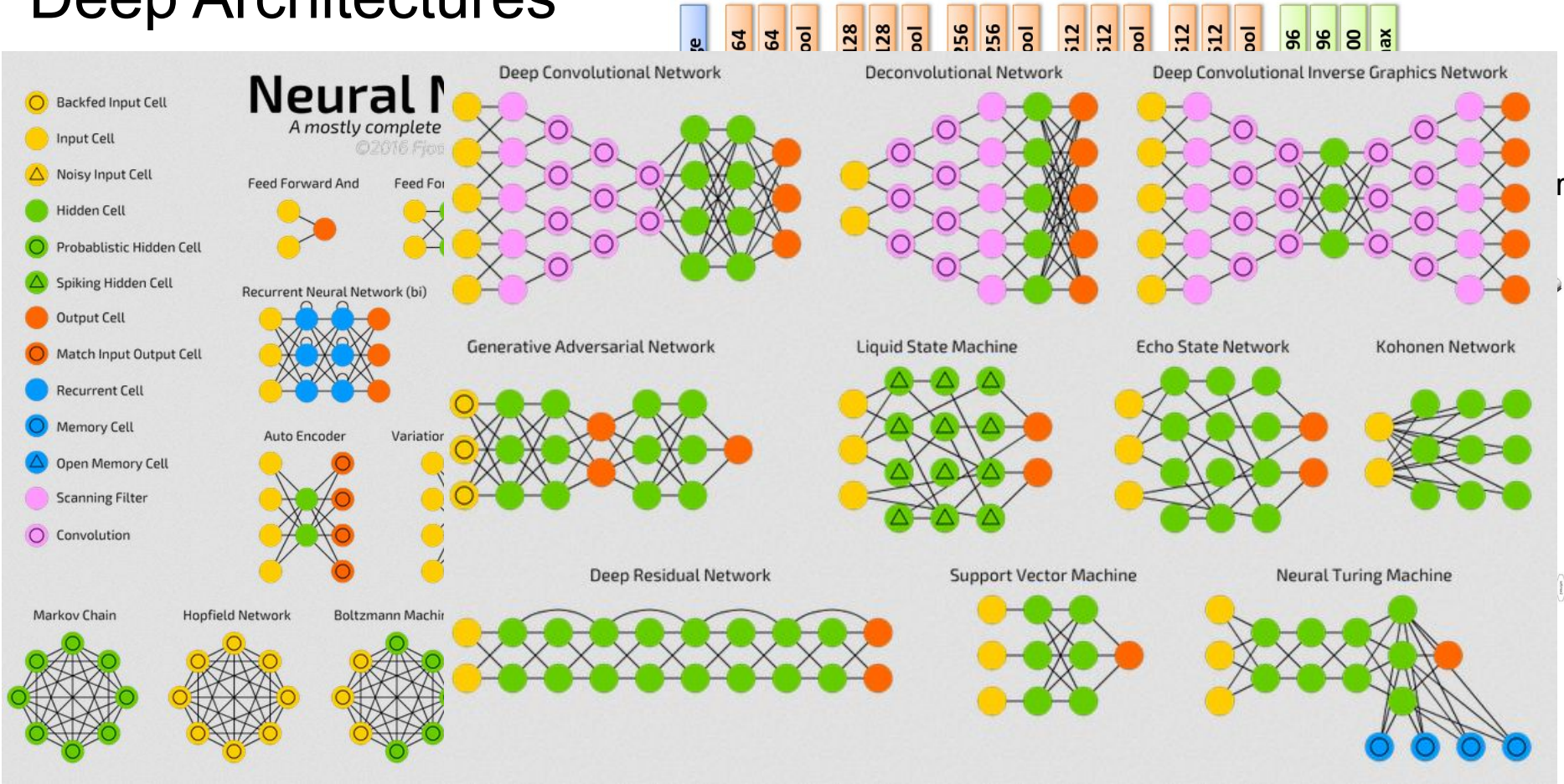




GoogLeNet, 2014

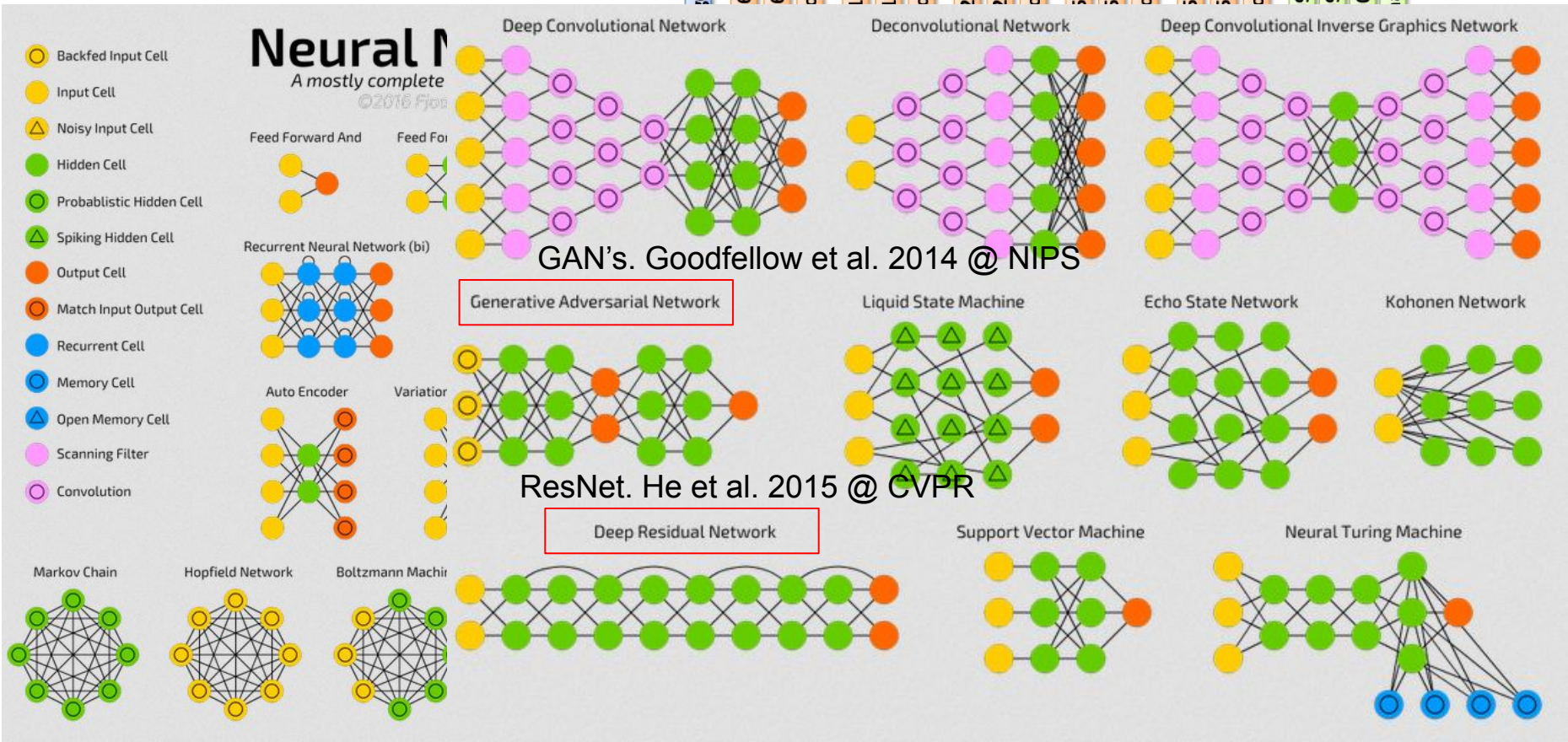


# Deep Architectures



# Deep Architectures

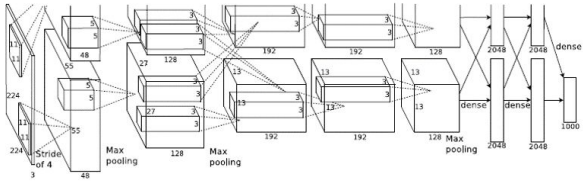
VGGNet (Simonyan & Zisserman, 2014)




rs)

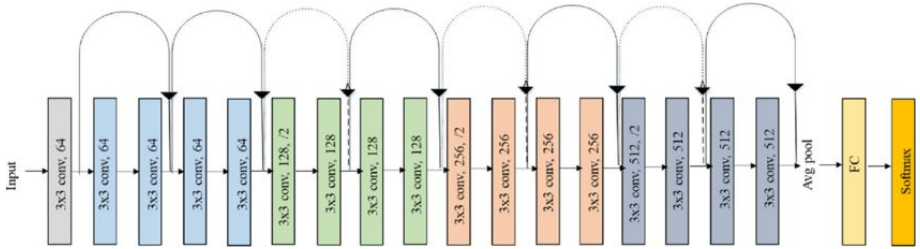
# Similarities & Differences between Architectures

AlexNet




VGG11


ResNet18



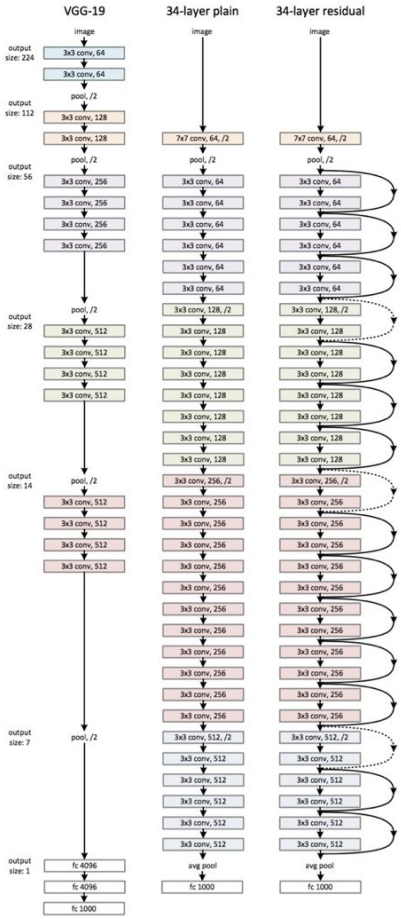
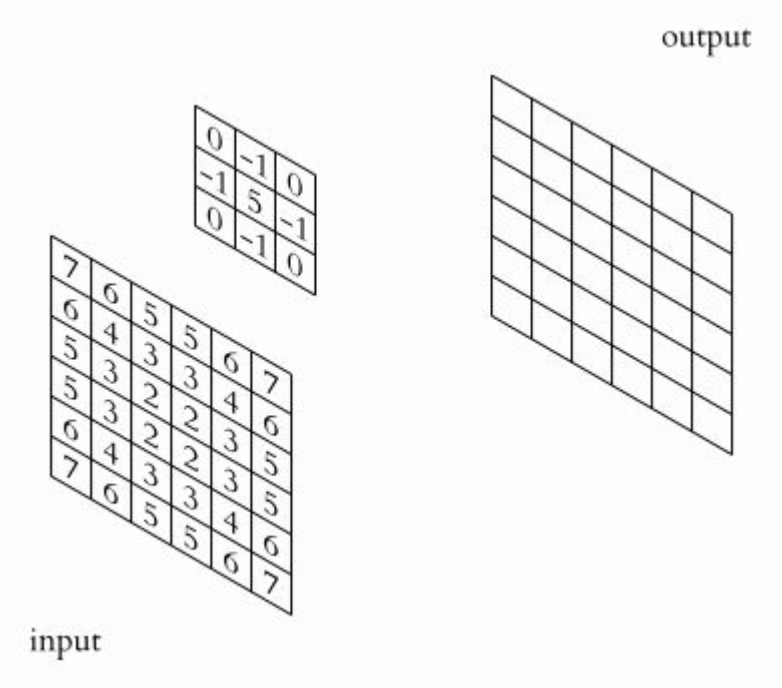


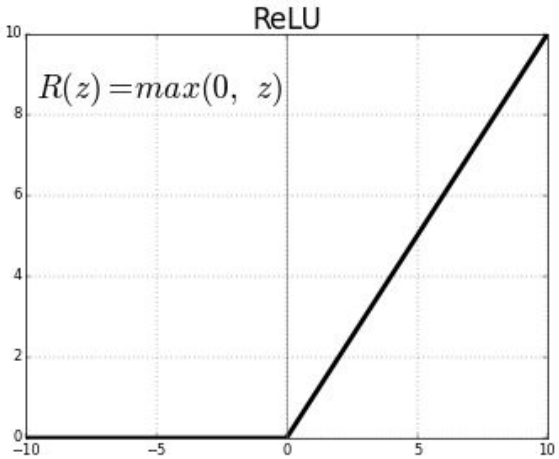

Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

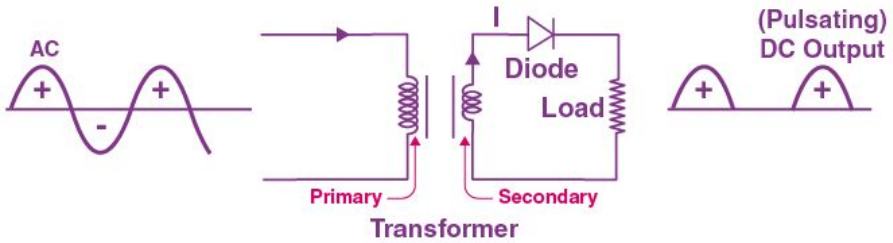
*From He et al. 2015 (Original ResNet paper)*

# Convolution



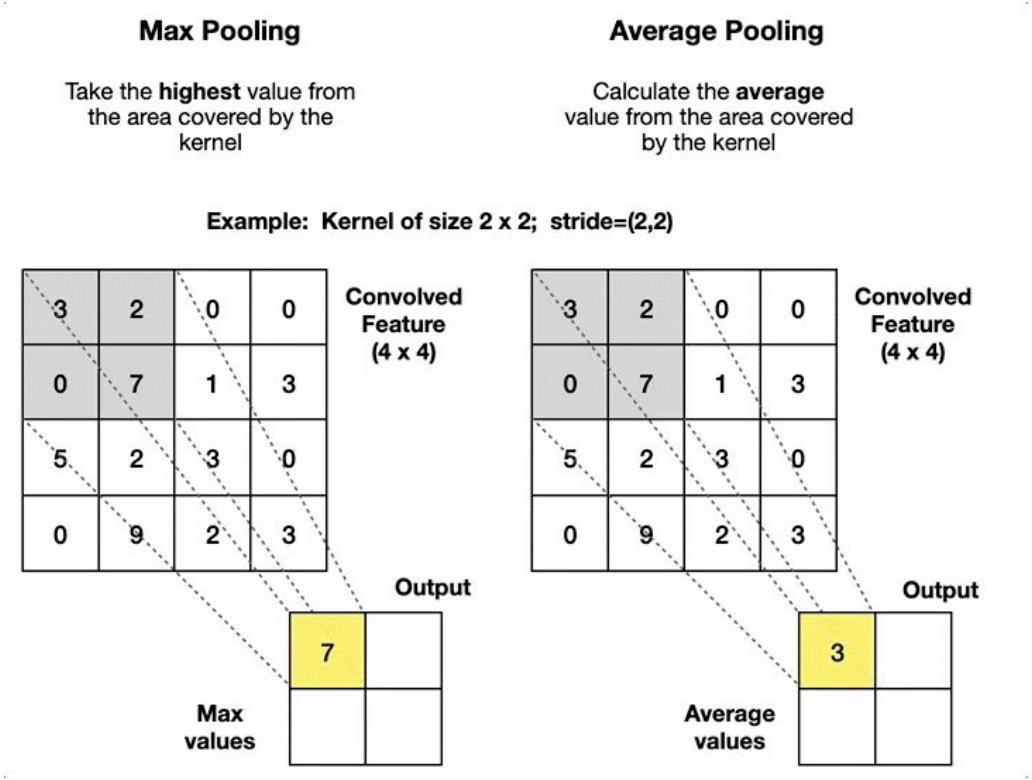
# Half-Wave Rectification



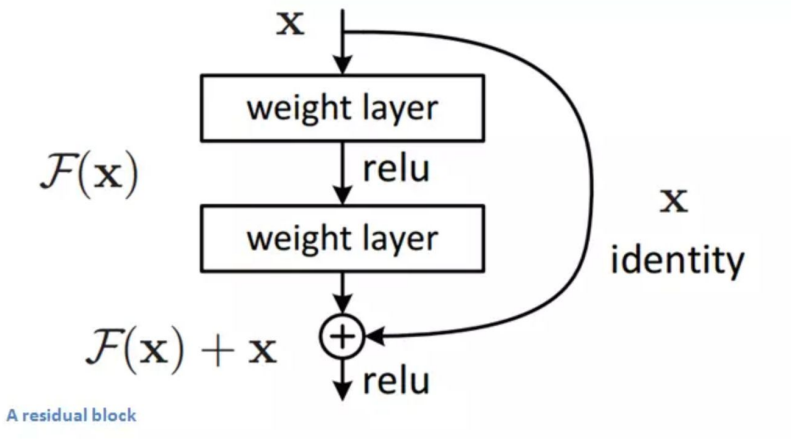




(A very similar idea is used in Power Electronics)

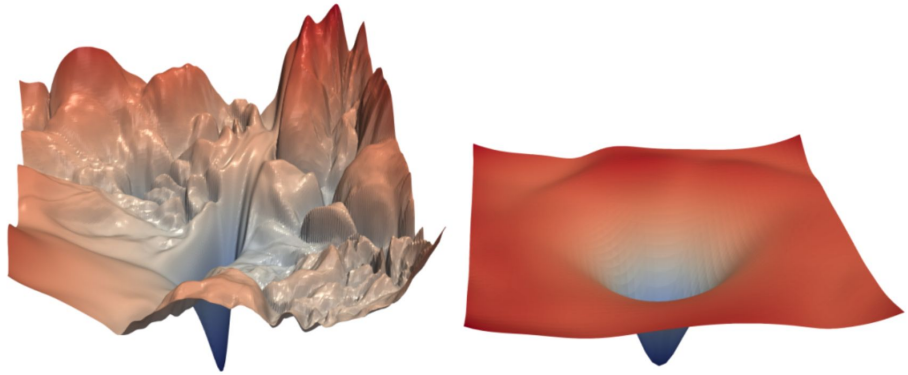
# Pooling


Max Pooling
Take the highest value from the area covered by the kernel
Average Pooling
Calculate the average value from the area covered by the kernel
Example: Kernel of size 2 x 2; stride=(2,2)
3 2 0 0
0 7 1 3
5 2 3 0
0 9 2 3
Convolved Feature (4 x 4)
Output
7
Max values
3 2 0 0
0 7 1 3
5 2 3 0
0 9 2 3
Convolved Feature (4 x 4)
Output
3
Average values

# Skip Connections




A residual block

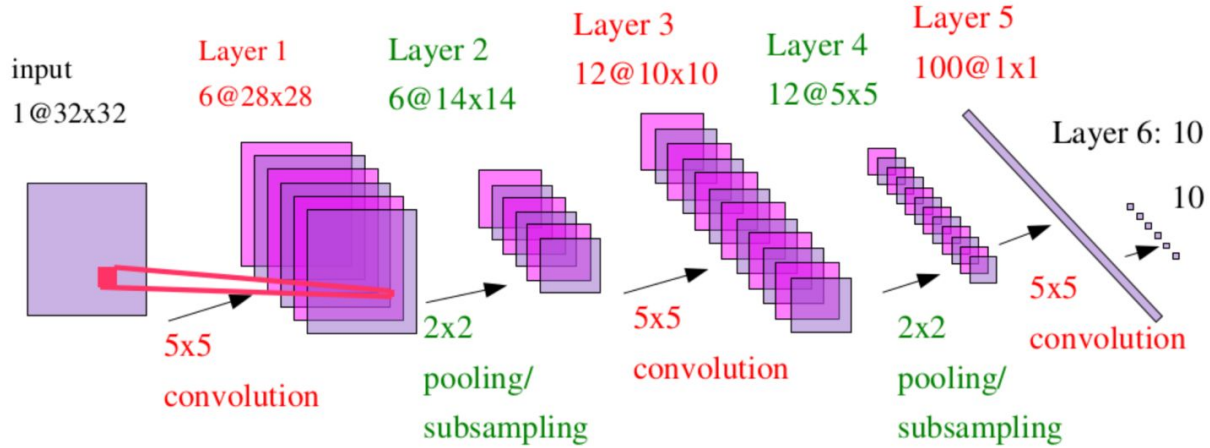


(a) without skip connections (b) with skip connections

**Visualizing the Loss Landscape of Neural Nets**

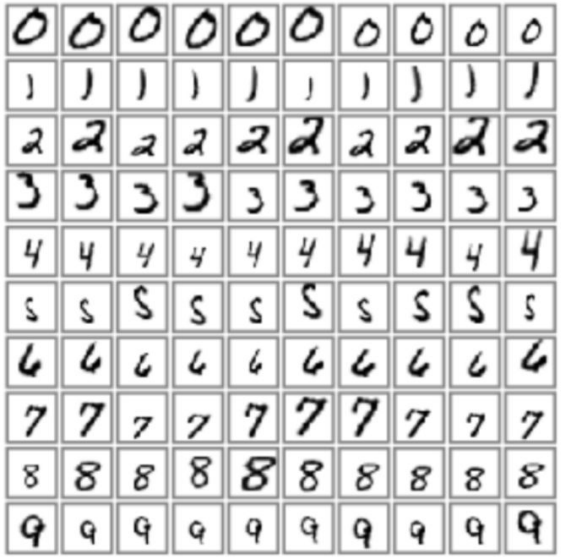
Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein

# Convolutional Net Architecture



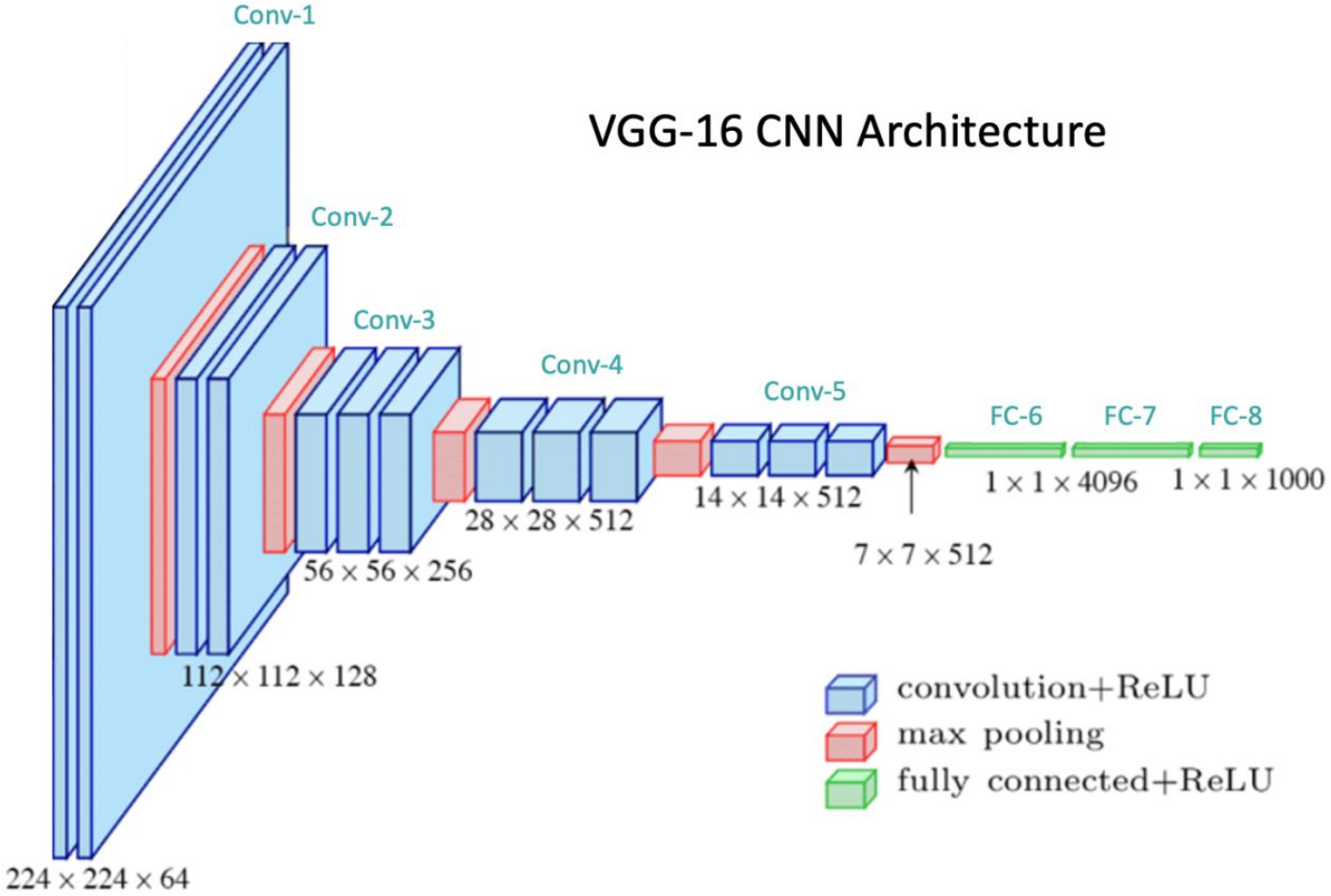

- **Convolutional net for handwriting recognition** (400,000 synapses)
- Convolutional layers (simple cells): all units in a feature plane share the same weights
- Pooling/subsampling layers (complex cells): for invariance to small distortions.
- **Supervised gradient-descent learning using back-propagation**
- **The entire network is trained end-to-end. All the layers are trained simultaneously**.

# MNIST Handwritten Digit Dataset

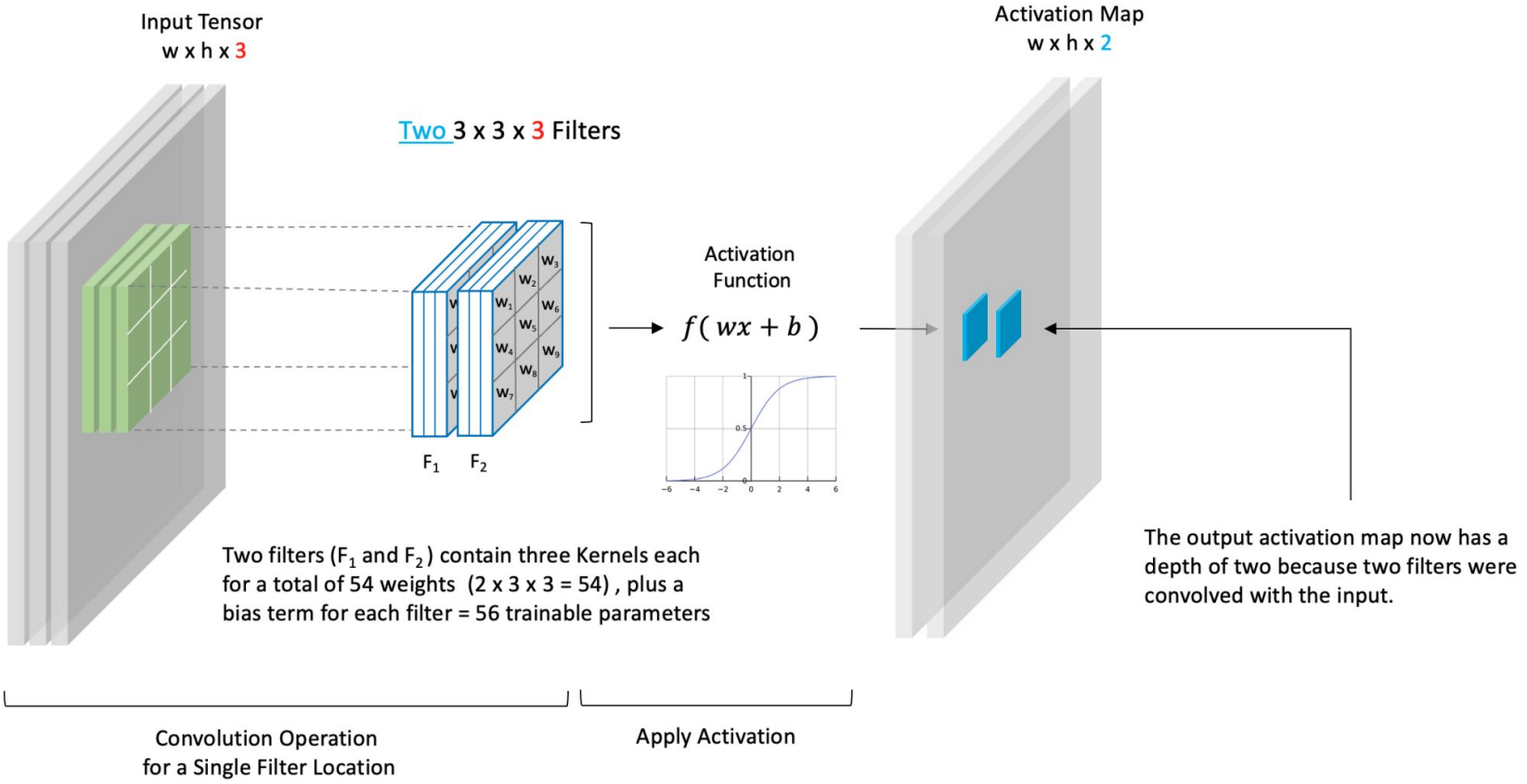


- Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

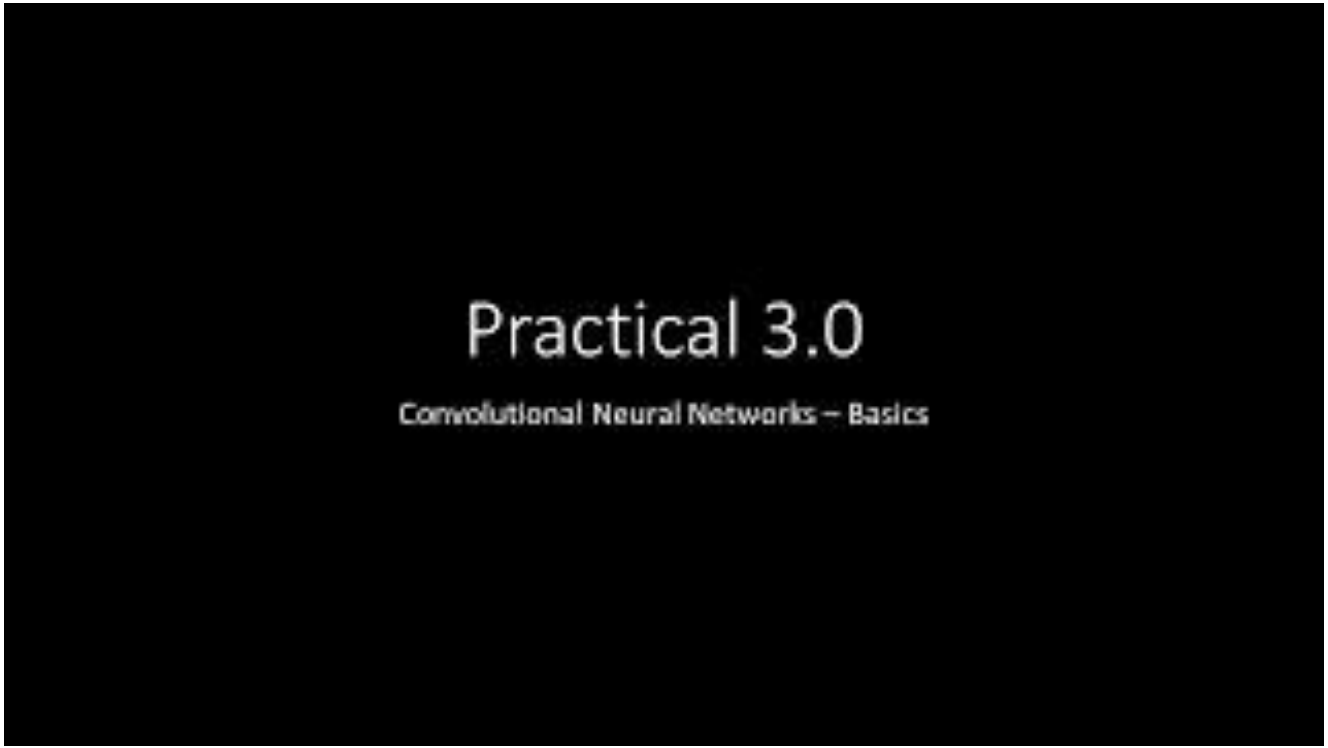
# VGG-16 CNN Architecture



# *Visualization of a Convolutional Layer : Input, Kernel, Activation Map (Output)*



An excellent Tutorial on CNNs can be viewed here:




*Prof. Alfredo Canziani*

NYU

# U-Net

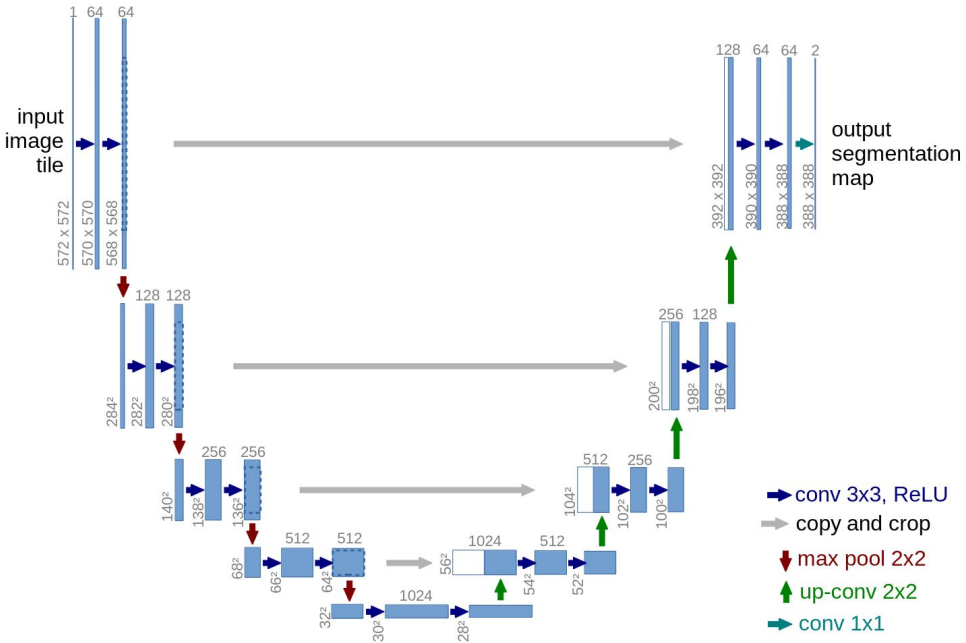
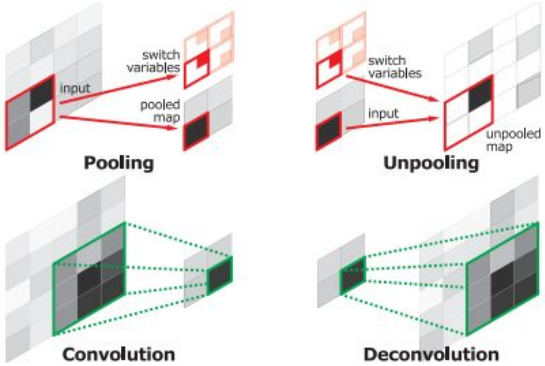



**Fig. 1.** U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.




Unpooling + DeConvolution

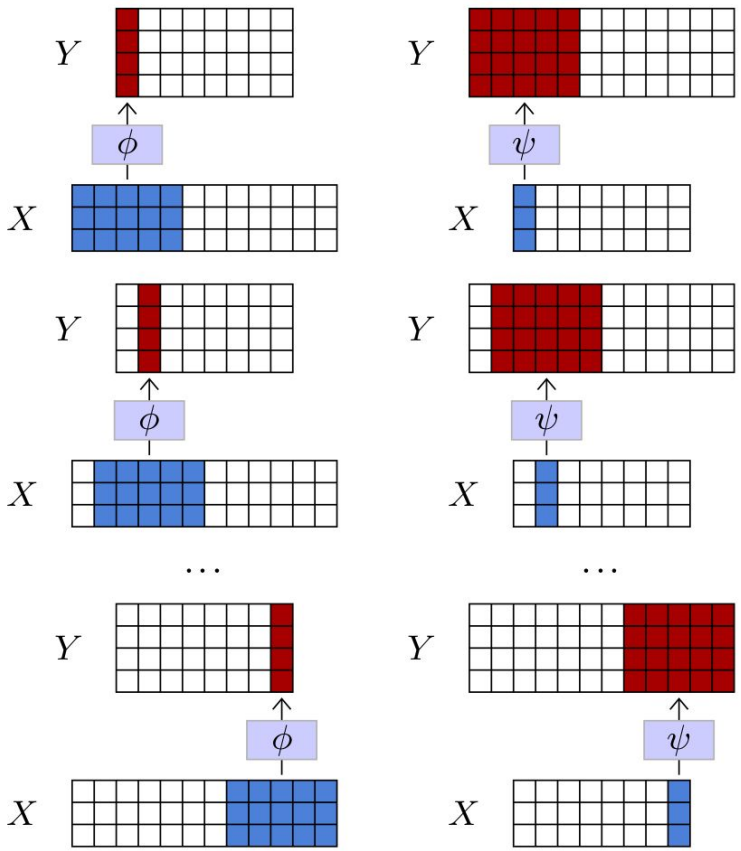



Figure 4.1: *A 1D convolution (left) takes as input a $D \times T$ tensor $X$, applies the same affine mapping $\phi(\cdot; w)$ to every sub-tensor of shape $D \times K$, and stores the resulting $D' \times 1$ tensors into $Y$. A 1D transposed convolution (right) takes as input a $D \times T$ tensor, applies the same affine mapping $\psi(\cdot; w)$ to every sub-tensor of shape $D \times 1$, and sums the shifted resulting $D' \times K$ tensors. Both can process inputs of different sizes.*

Source: The Little Book of Deep Learning

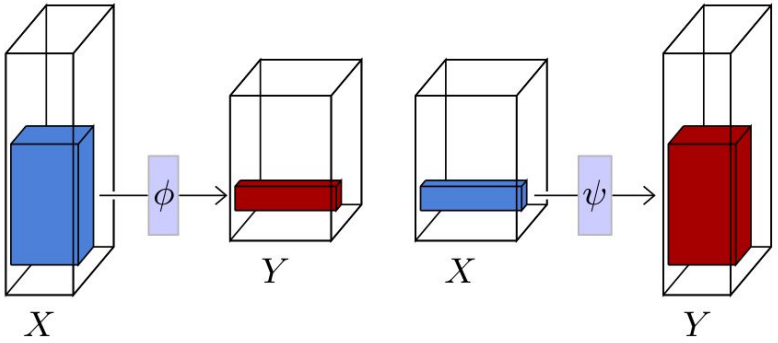
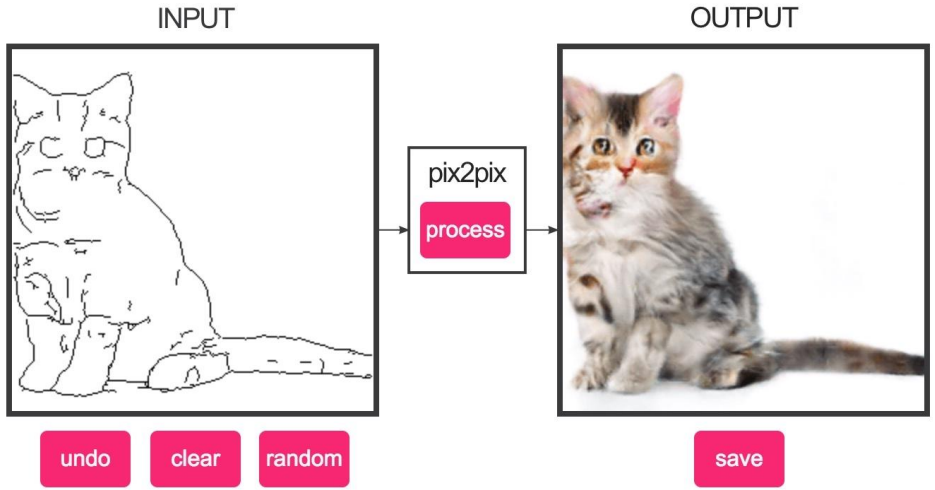



Figure 4.2: *A 2D convolution (left) takes as input a $D \times H \times W$ tensor $X$, applies the same affine mapping $\phi(\cdot; w)$ to every sub-tensor of shape $D \times K \times L$, and stores the resulting $D' \times 1 \times 1$ tensors into $Y$. A 2D transposed convolution (right) takes as input a $D \times H \times W$ tensor, applies the same affine mapping $\psi(\cdot; w)$ to every $D \times 1 \times 1$ sub-tensor, and sums the shifted resulting $D' \times K \times L$ tensors into $Y$.*




Source: *The Little Book of Deep Learning*

# Image-to-Image Translation with Conditional Adversarial Networks

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

Berkeley AI Research (BAIR) Laboratory, UC Berkeley

{isola,junyanz,tinghuiz,efros}@eecs.berkeley.edu

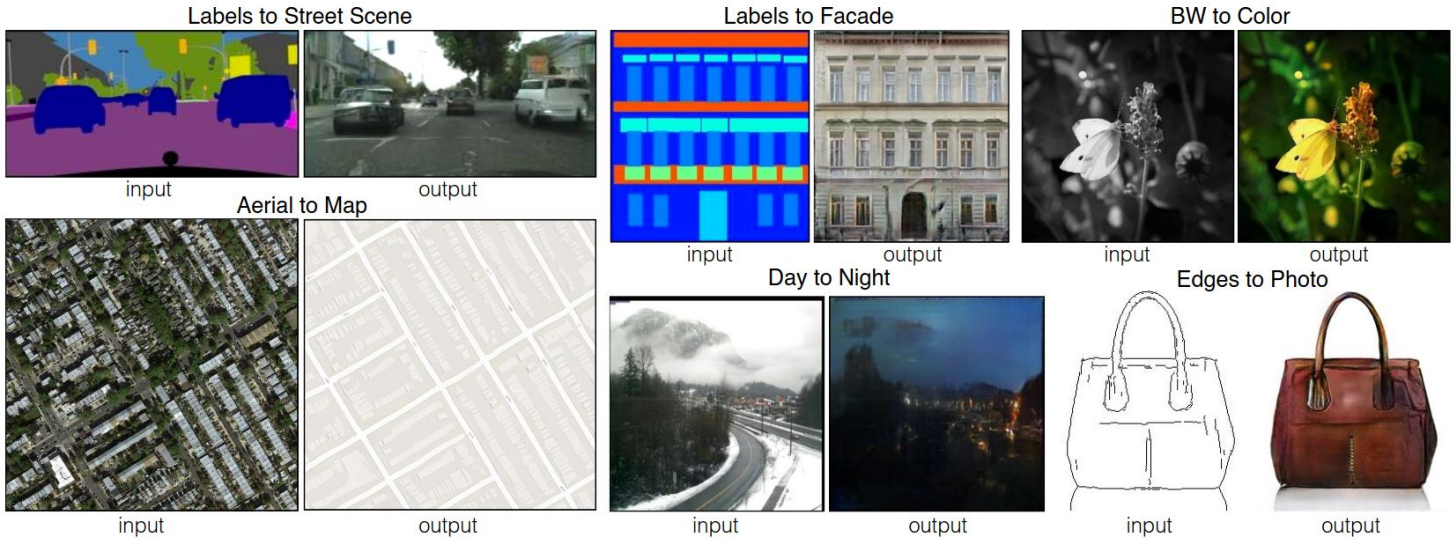



Figure 1: Many problems in image processing, graphics, and vision involve translating an input image into a corresponding output image. These problems are often treated with application-specific algorithms, even though the setting is always the same: map pixels to pixels. Conditional adversarial nets are a general-purpose solution that appears to work well on a wide variety of these problems. Here we show results of the method on several. In each case we use the same architecture and objective, and simply train on different data.

# Applications : Texture Synthesis

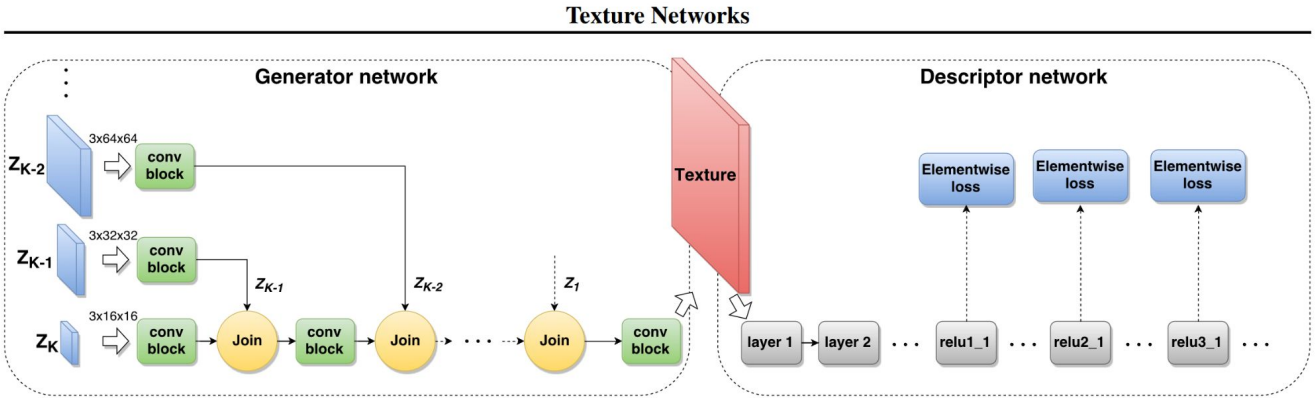
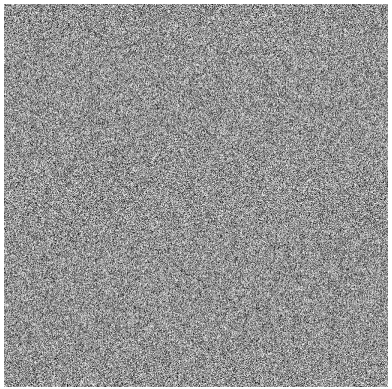



*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.

# Applications : Texture Synthesis



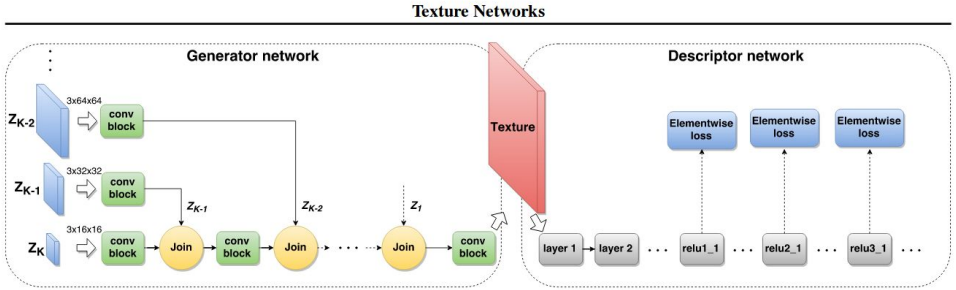
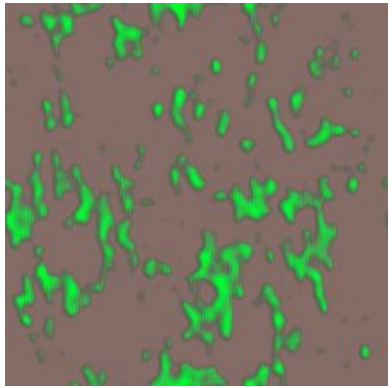


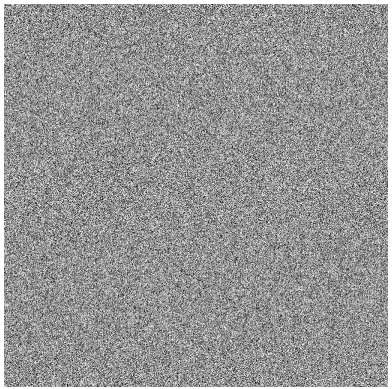

*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.



Ulyanov et al. 2016

# Applications : Texture Synthesis

Forward Pass



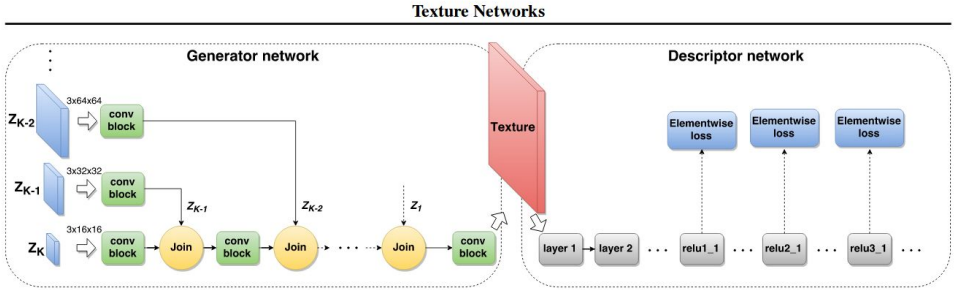
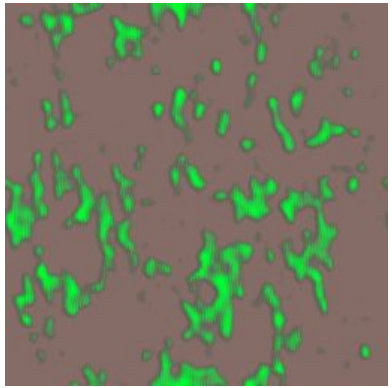


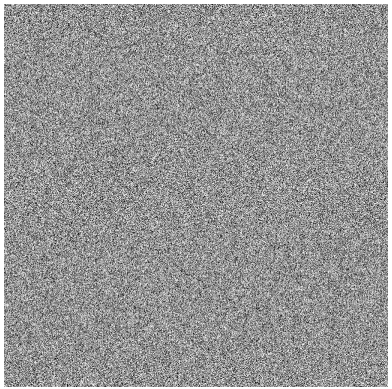

*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.



Ulyanov et al. 2016

# Applications : Texture Synthesis

Forward Pass



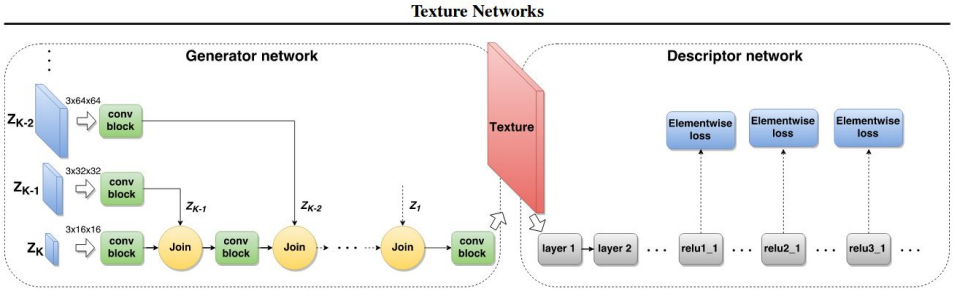
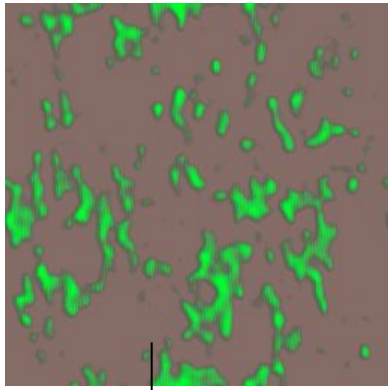



*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.

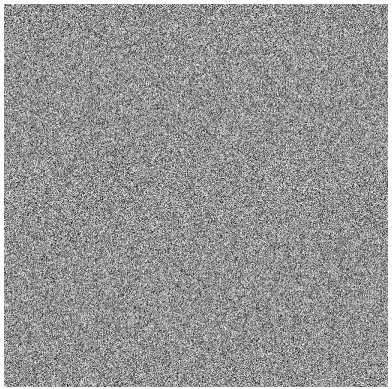


Texture Loss

Ulyanov et al. 2016

# Applications : Texture Synthesis

Forward Pass



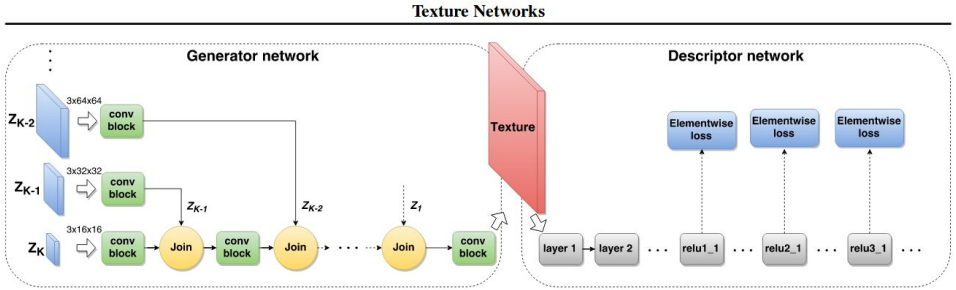
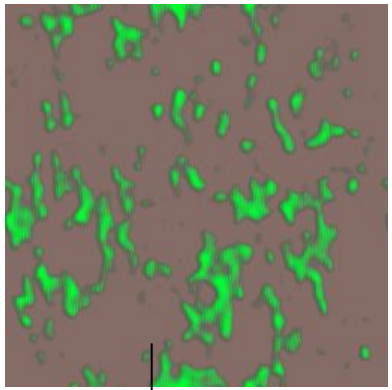



*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.



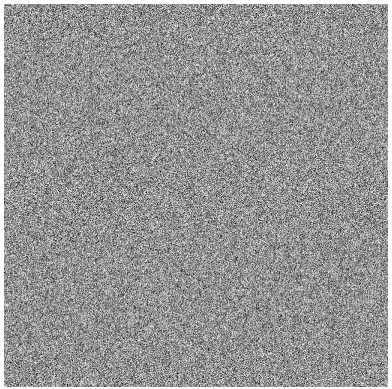
Texture Loss

Backward Pass

Ulyanov et al. 2016

# Applications : Texture Synthesis

Forward Pass



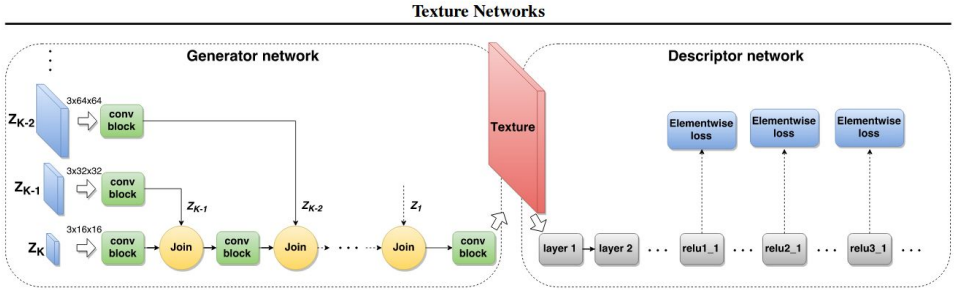
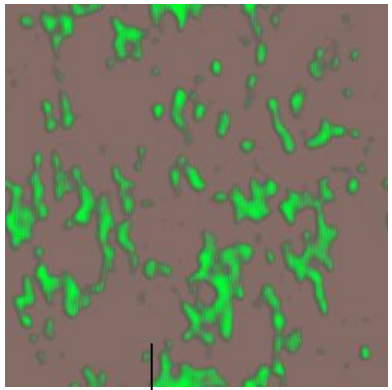



*Figure 2.* Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors $z_i$ of different sizes.



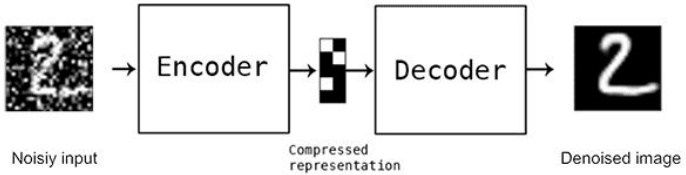
Update Gradients

Texture Loss

Backward Pass

Ulyanov et al. 2016

# Applications : Image De-Noising


*“Fija del Quiz” : Why is Convolution a good idea for image denoising?*

# Applications : Image Compression

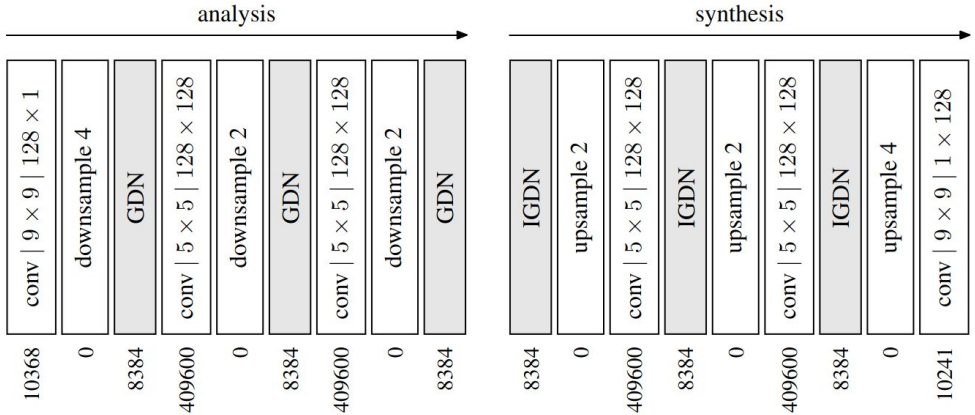


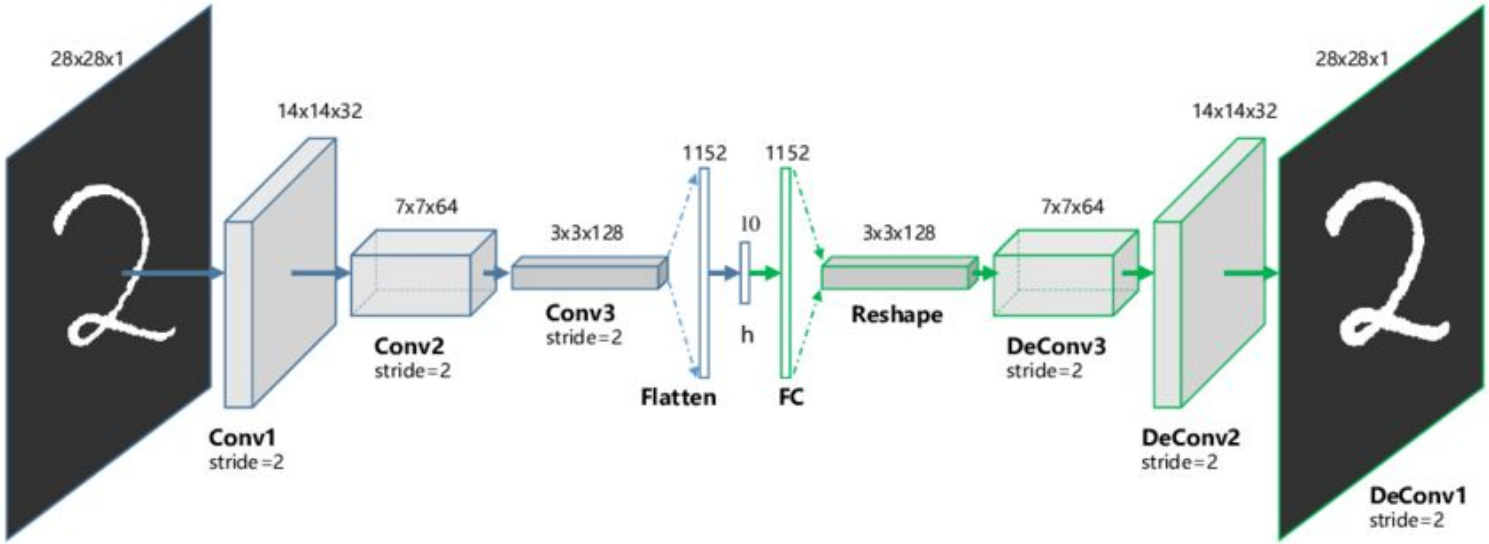

Figure 8: Parameterization of analysis ($g_a$) and synthesis ($g_s$) transforms for grayscale images. *conv*: affine convolution (1)/(6), with filter support ($x \times y$) and number of channels (output×input). *down-/upsample*: regular down-/upsampling (2)/(5) by given factor (implemented jointly with the adjacent convolution). *GDN/IGDN*: generalized divisive normalization across channels (3), and its approximate inverse (4); see text. Number of parameters for each layer given at the bottom.

Balle et al., ICLR 2017.

**Proposed method**, 10130 bytes (0.287 bit/px), PSNR: 25.27 dB, MS-SSIM: 0.9537

**JPEG 2000**, 10197 bytes (0.289 bit/px), PSNR: 24.41 dB, MS-SSIM: 0.9320

# Remembering Convolutional Auto-Encoders



# Examples of the Effects of “bottle-neck” size

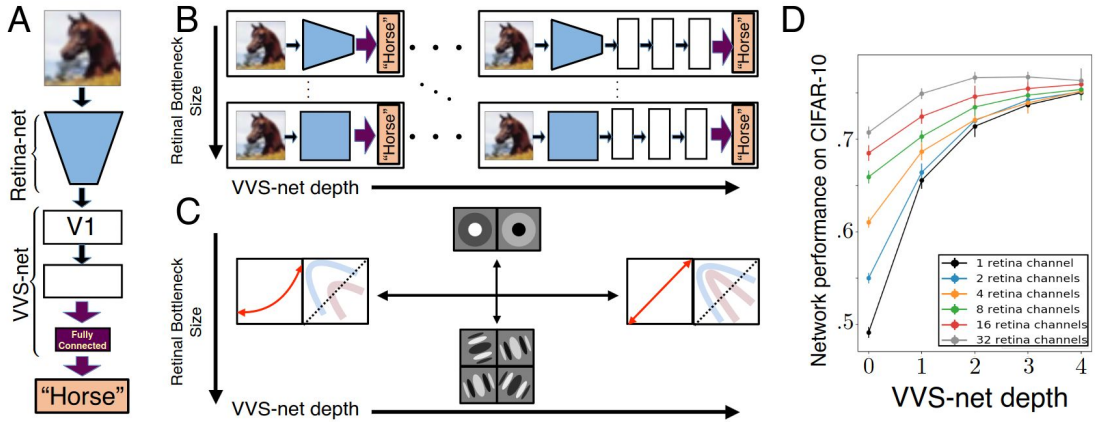



Figure 1: Illustration of the framework we used to model early visual representations. A: We trained convolutional neural networks on an image recognition task (CIFAR-10). The networks were composed of two parts, a retina-net and a ventral-visual-system-net (VVS-net), which receives input from the retina-net. B: We varied the number of layers in the VVS-net (white boxes) and the number of channels at the output of the retina-net (blue box). C: Key results: (1) A bottleneck at the output of the retina yielded center-surround retinal RFs. (2) A shallow VVS-net yielded more nonlinear retinal responses (linearity is schematized by the red arrow), which better disentangled image classes (represented as bent manifolds). D: Test-set accuracy of all model architectures on CIFAR-10, averaged over ten networks with random initial weights for each architecture. Performance increases with VVS-net depth and retinal channel, indicating that both factors are meaningful constraints on the network in the regime tested.

*From : Lindsey et al., ICLR 2019*

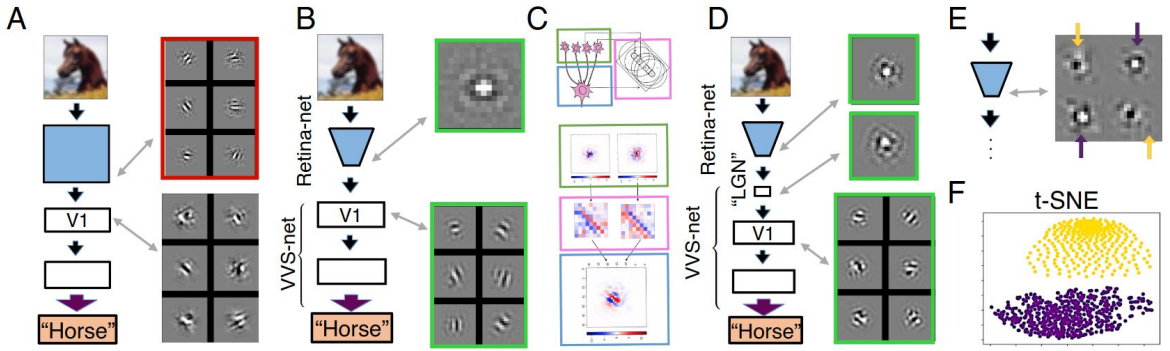



Figure 2: Effects of a bottleneck constraint on receptive fields (RFs). All results are shown for $D_{VVS} = 2$. A: Examples of RFs of cells at selected layers (layers 2 and 3) of a control network with no bottleneck. No center-surround RFs appear. B: Center-surround RFs emerge at the output of the retina-net (layer 2) and oriented RFs emerge in the first layer of the VVS-net when we impose a bottleneck constraint at the output of the retina ($N_{BN} = 1$) C: Top: Hubel and Wiesel's hypothesis on oriented cell formation in V1 (Hubel, 1995). Bottom: A representative example of an orientation-selective neuron (bottom RF) drawing from center-surround channels (top RFs) in the previous layer with weight matrices (center) according to their polarity. Light / dark-selective regions of a receptive field, and positive / negative weights, are represented with red / blue, respectively. D: Examples of RFs in a network with an extra bottleneck layer corresponding to mammalian LGN. Center-surround RFs appear at both the retinal output and LGN layer. E: Examples of ON and OFF center-surround RFs in the untied network ($N_{BN} = 4$). F: t-SNE clustering of the retinal neurons of the untied network (see text). Two distinct cell type clusters form corresponding to ON and OFF center-surround receptive fields.

*From : Lindsey et al., ICLR 2019*