VISUAL RECOGNITION

ANUSH SANKARAN
IBM RESEARCH

Most of the slide content are borrowed from CS231n. Thanks to the original authors

EASY LOGISTICS: GITHUB

- Github Repo: https://github.com/goodboyanush/iiit-bangalore-march-april-2019
- Lectures slides, Hands-on notebooks, Assignment solutions
- Have any doubt in my lectures or assignments?
  - Go ahead and create an issue in the repo!
  - I will try to answer them asap!
  - Everyone will be benefitted by the questions asked by one student

ASSIGNMENT #2 (DEADLINE: 21 APRIL) TEAM SIZE (2–3)

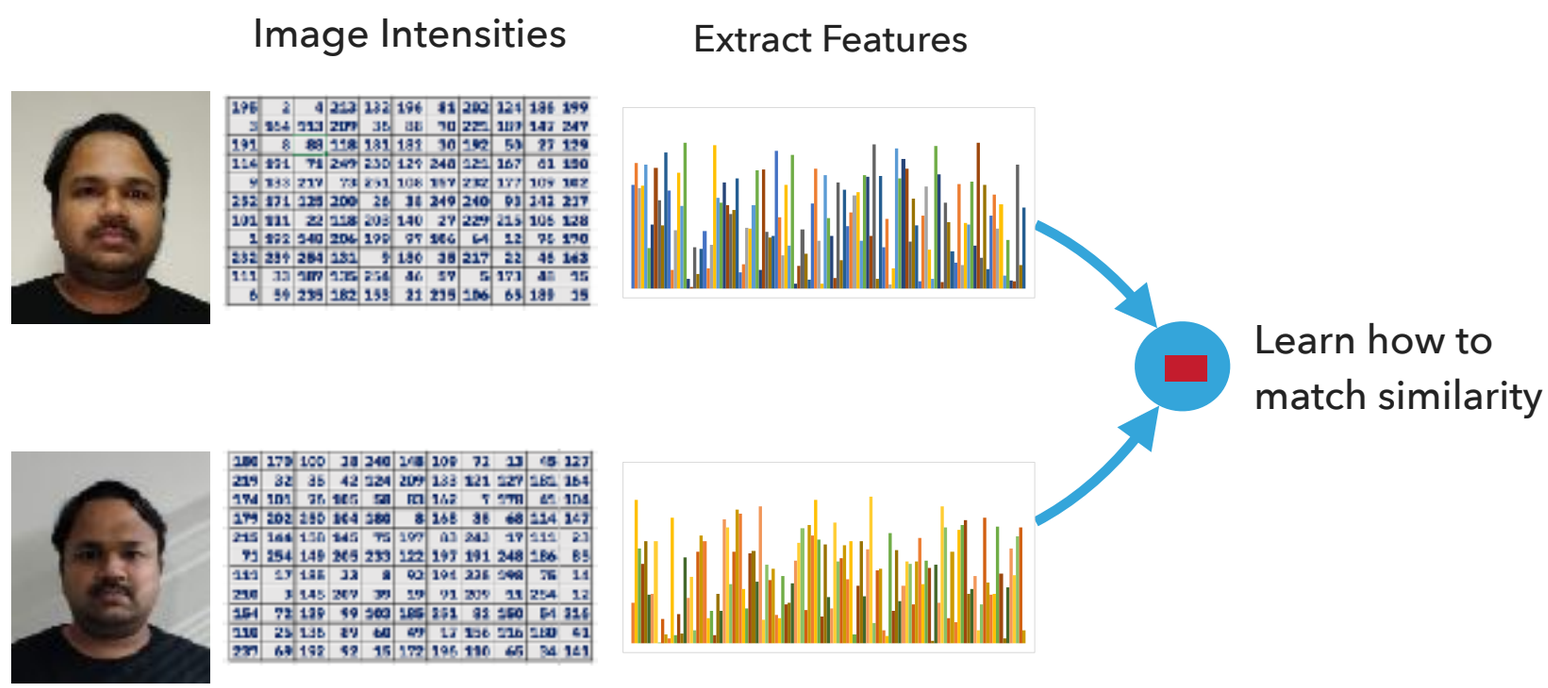
- Consider CIFAR-10 dataset
- For model, please check https://github.com/goodboyanush/iiit-bangalore-march-april-2019/issues/6
- Hyperparameters to explore:
  - Layer activation function
  - Data preprocessing, normalisation
  - Dropout (where to apply, what is the strength)
  - BatchNorm (where to apply)
  - Loss function
  - Learning rate and learning strategy
- Report the final best set of hyper parameters set that gave you the best accuracy.
- You will be evaluated on what values you explored for each of the parameter and in what order? Please do explain how you baby-sit the learning process.
- Please note that there is no single correct answer to this question.

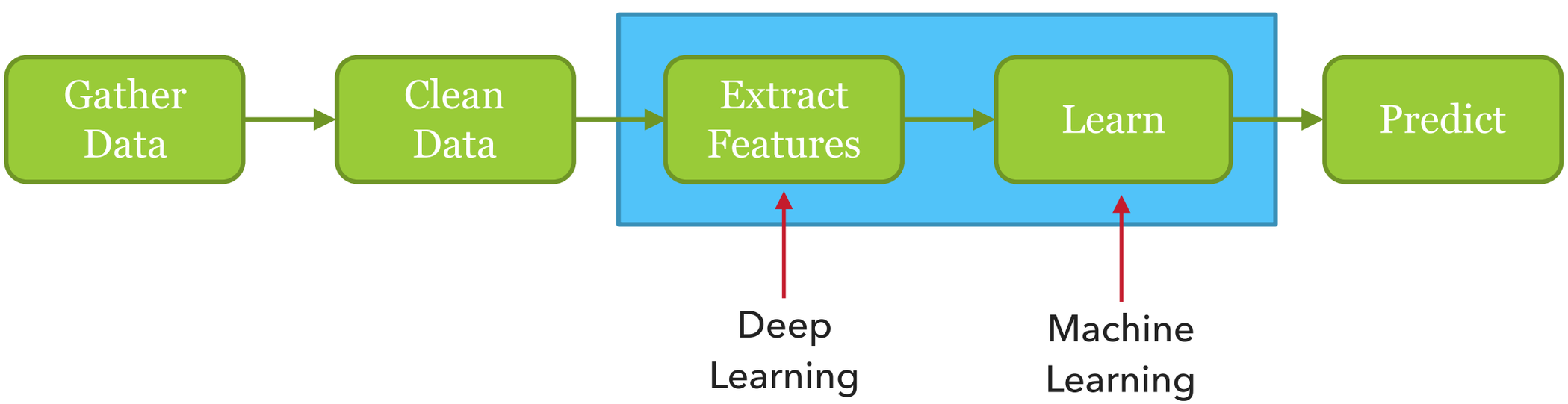
RECAP: WHAT IS THE PROBLEM?

- You are a genius …
- Only if a machine or an algorithm could do it …

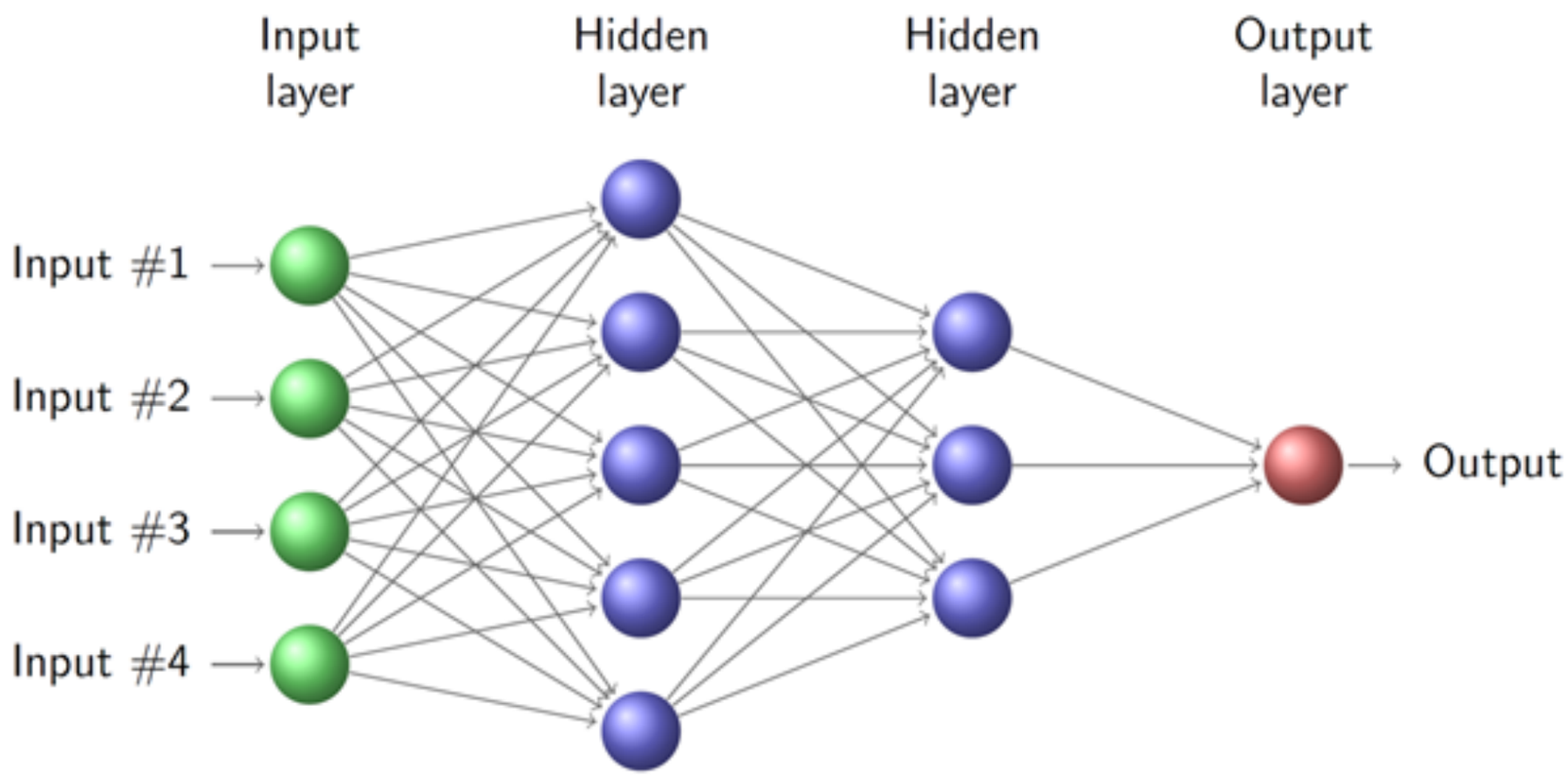
- Make a computer understand the image …
- Understand the background & environment …
- Classify various objects in the image …

RECAP: WHAT IS THE SOLUTION PIPELINE

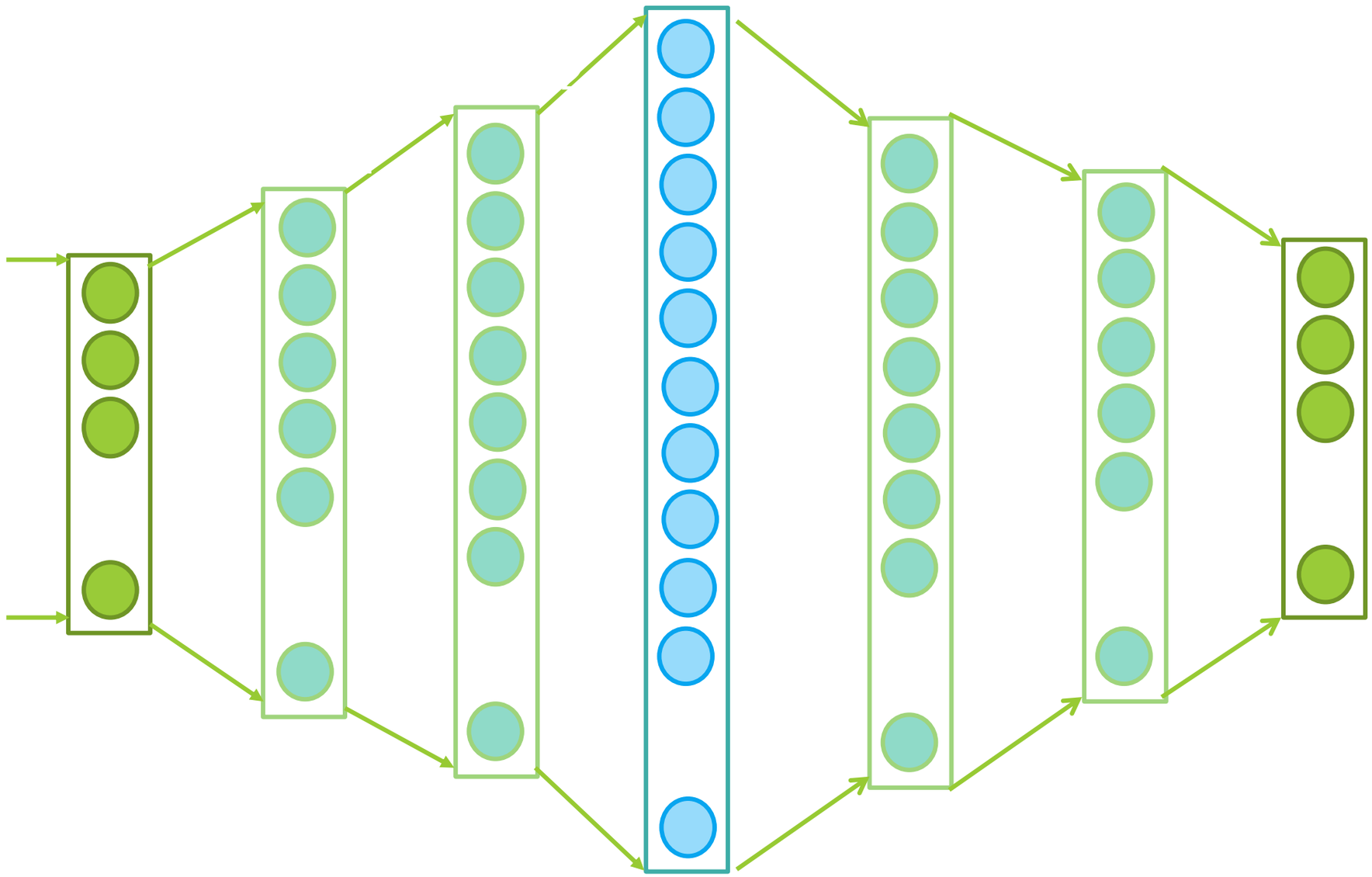




RECAP: MULTI LAYER PERECEPTRON



RECAP: STACKED AUTOENCODER



RECAP: DIFFERENT TYPES OF DEEP LEARNING

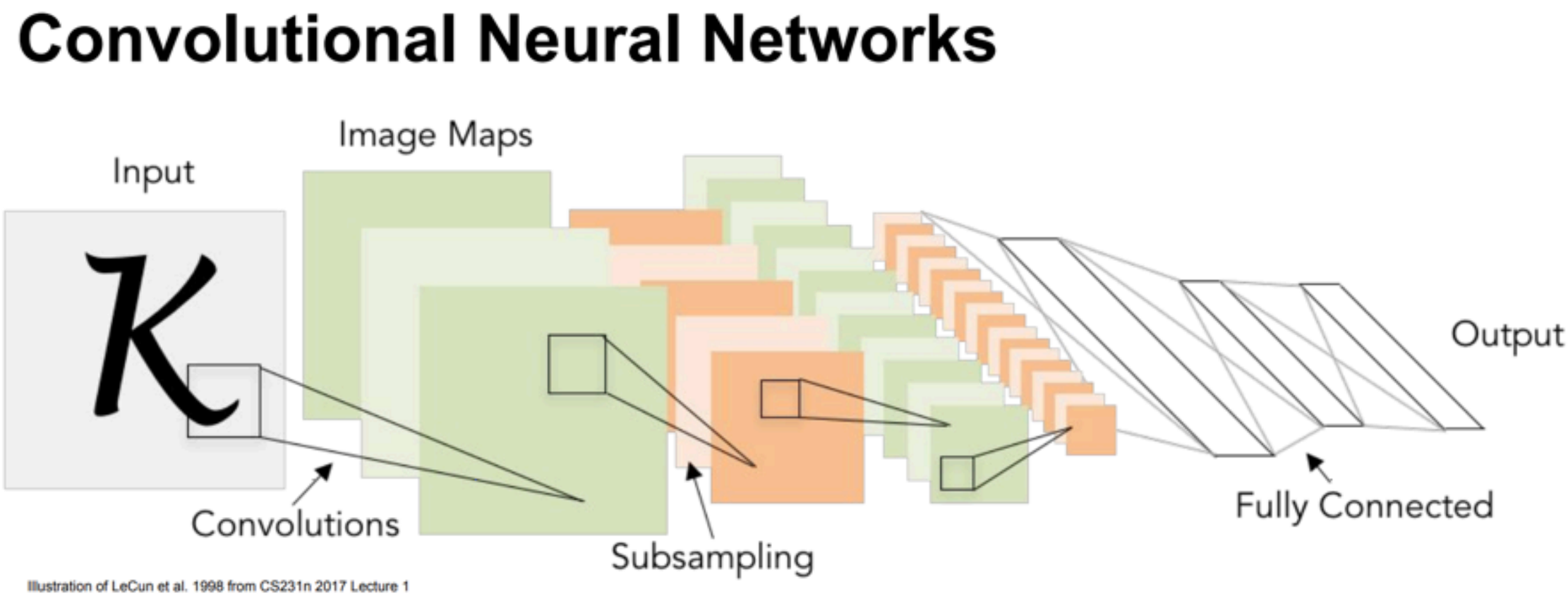
- Input data is a vector Stacked
  - Autoencoder
  - Restricted Boltzmann Machines
  - Deep Belief Networks

- Input data is multi-dimensional
  - Convolutional Neural Network
  - Example: Image analysis, Object recognition

- Input data is time-series
  - Recurrent Neural Network
  - Example: Language Modelling, Speech recognition

RECAP: CONVOLUTIONAL NEURAL NETWORK

Where we are now...



RECAP: WHAT HYPER-PARAMETERS TO PLAY WITH?

- Network architecture
  - How many layers?
  - What sequence of layers?
  - What is the depth of the model?
- Learning rate
  - Decay rate
  - Update strategy
- Regularization
  - L2 / Dropout strength
- Which loss function?
  - hinge/ L2/ Cross-entropy
- Which optimization?
  - SGD/ adam/ RMSPROP

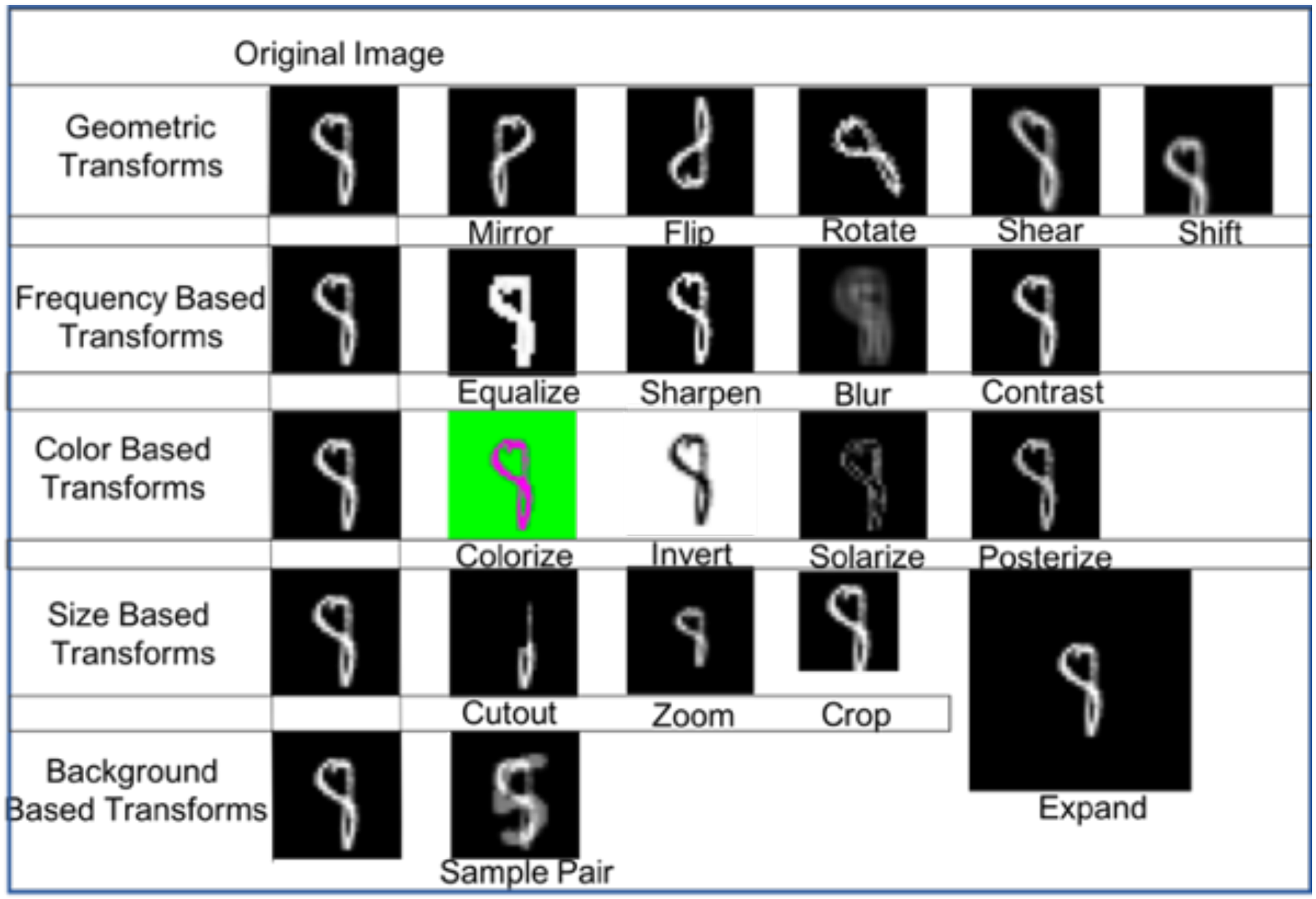
MOST IMPORTANT CHALLENGE

CNN Model has lot of weight parameters to learn…

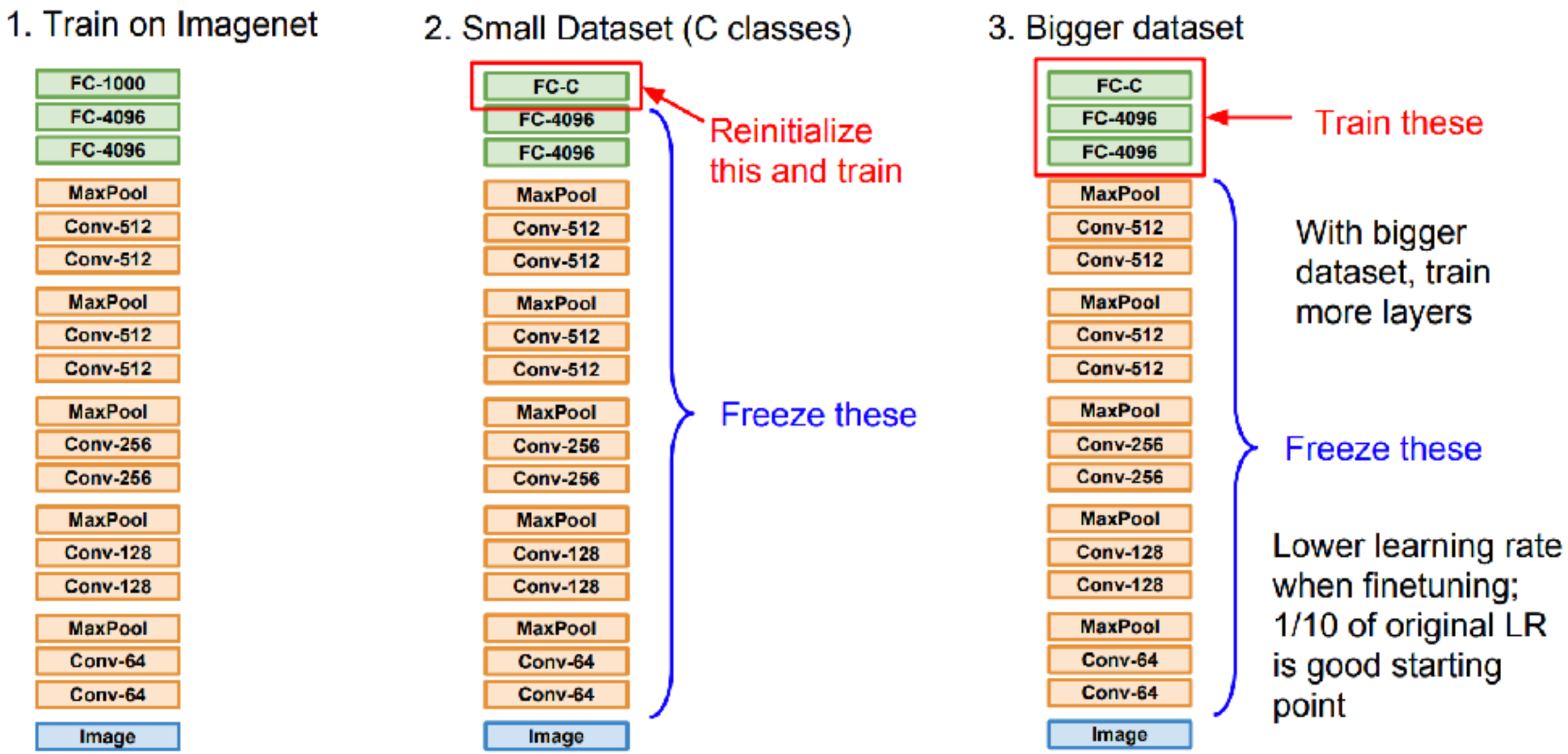
"You need lots of data to train/use a CNN model"

1. Data Augmentation
2. Transfer Learning

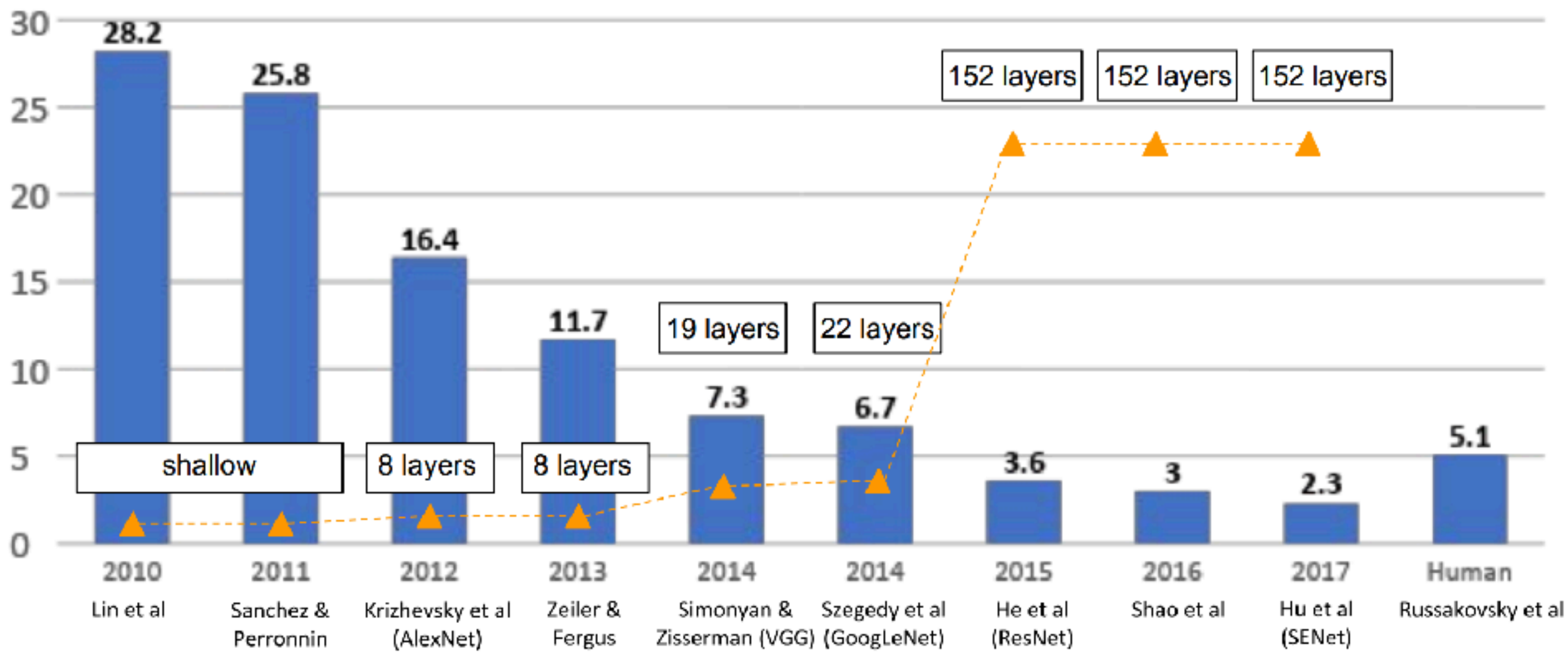
RECAP: DATA AUGMENTATION



RECAP: TRANSFER LEARNING



RECAP: IMAGENET CHALLENGE WINNERS



CNN FOR SUPERVISED LEARNING

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

CNN FOR SUPERVISED LEARNING

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

→ Cat

Classification

CNN FOR SUPERVISED LEARNING

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

DOG, DOG, CAT

Object Detection

CNN FOR SUPERVISED LEARNING

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

GRASS, CAT,
TREE, SKY

Semantic Segmentation

CNN FOR SUPERVISED LEARNING

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

A cat sitting on a suitcase on the floor

Image captioning

CNN FOR UNSUPERVISED LEARNING

Supervised Learning

- **Data**: (x, y)
  - x is the data
  - y is the label
- **Goal**: Learn a *function* to map (x -> y)
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

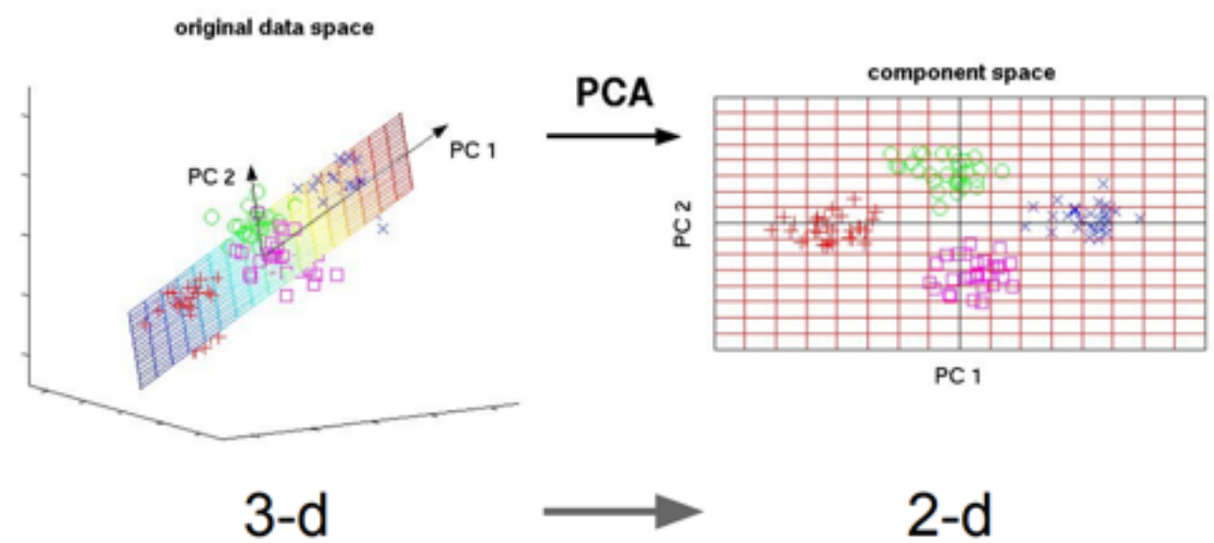
Unsupervised Learning

- **Data**: (x)
  - x is the data
  - No labels are provided
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction,
  - feature learning,
  - density estimation, etc.

CNN FOR UNSUPERVISED LEARNING

Unsupervised Learning

- **Data**: (x)
  - x is the data
  - No labels are provided
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction,
  - feature learning,
  - density estimation, etc.

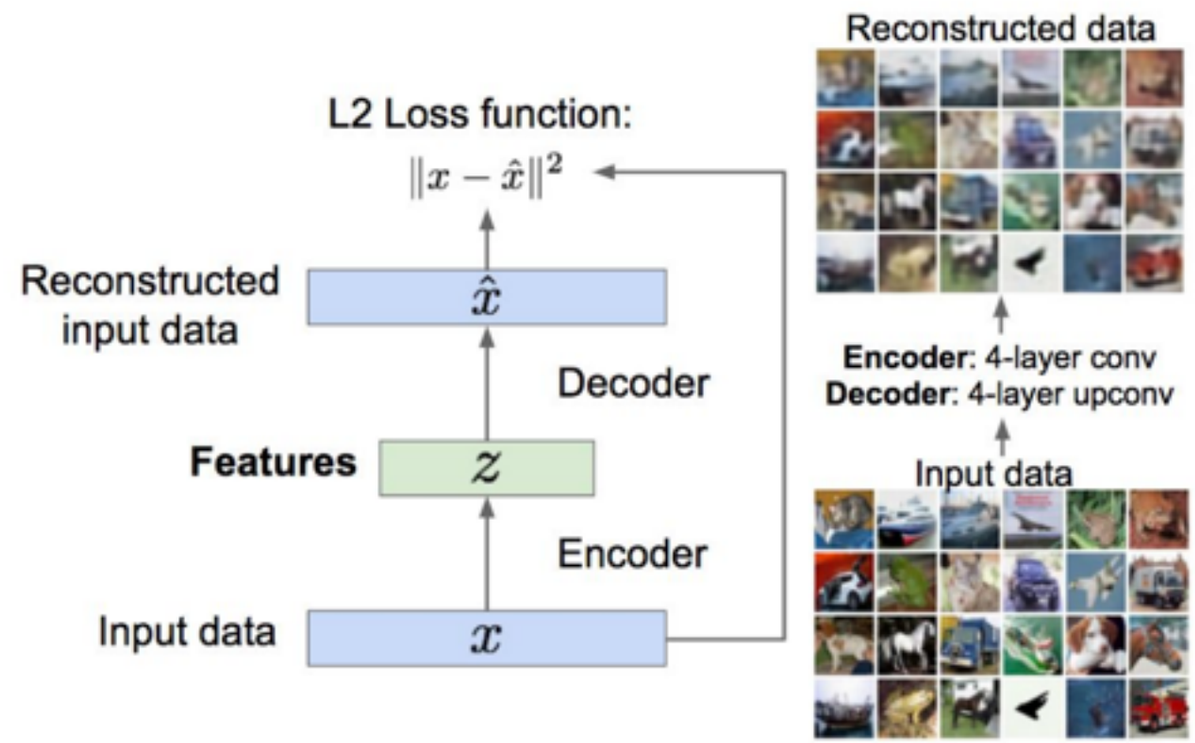


Principal Component Analysis
(Dimensionality reduction)

CNN FOR UNSUPERVISED LEARNING

Unsupervised Learning

- **Data**: (x)
  - x is the data
  - No labels are provided
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction,
  - feature learning,
  - density estimation, etc.



Autoencoders
(Feature learning)

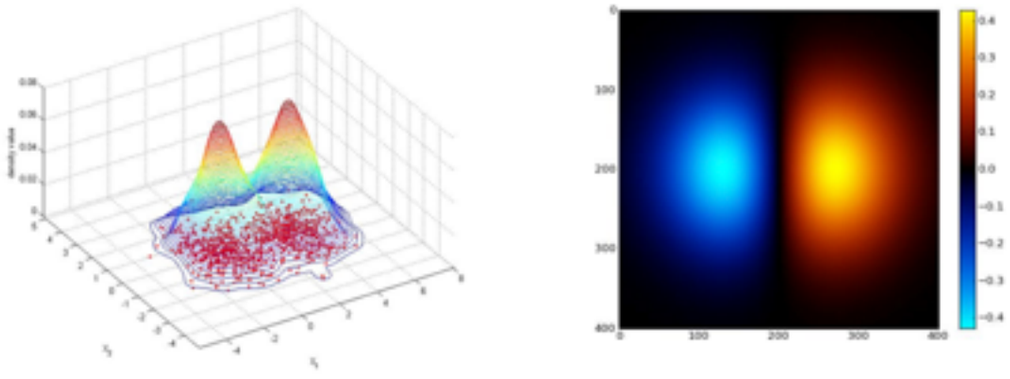
CNN FOR UNSUPERVISED LEARNING

Unsupervised Learning

- **Data**: (x)
  - x is the data
  - No labels are provided
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction,
  - feature learning,
  - density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

CNN FOR UNSUPERVISED LEARNING – GENERATIVE MODELLING

Given training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

MAXIMUM LIKELIHOOD ESTIMATION

- We are given a finite sample from a data distribution

$$X = \{x | x \sim p_{\text{data}}(x)\}, |X| = n$$

- We construct a parametric model $p_{\text{model}}(x; \theta)$ for the distribution, and build a likelihood

$$\mathcal{L}(\theta; X) = \prod_{x \in X} p_{\text{model}}(x; \theta)$$

- In practice, we optimize through MCMC or other means, and obtain

$$\theta_{\text{opt}} = \arg\min_{\theta} \{-\ln \mathcal{L}(\theta; X)\}$$

WHY GENERATIVE MODELS

- Realistic samples for artwork, super-resolution, colorization, etc.

- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

TAXONOMY OF GENERATIVE MODELS

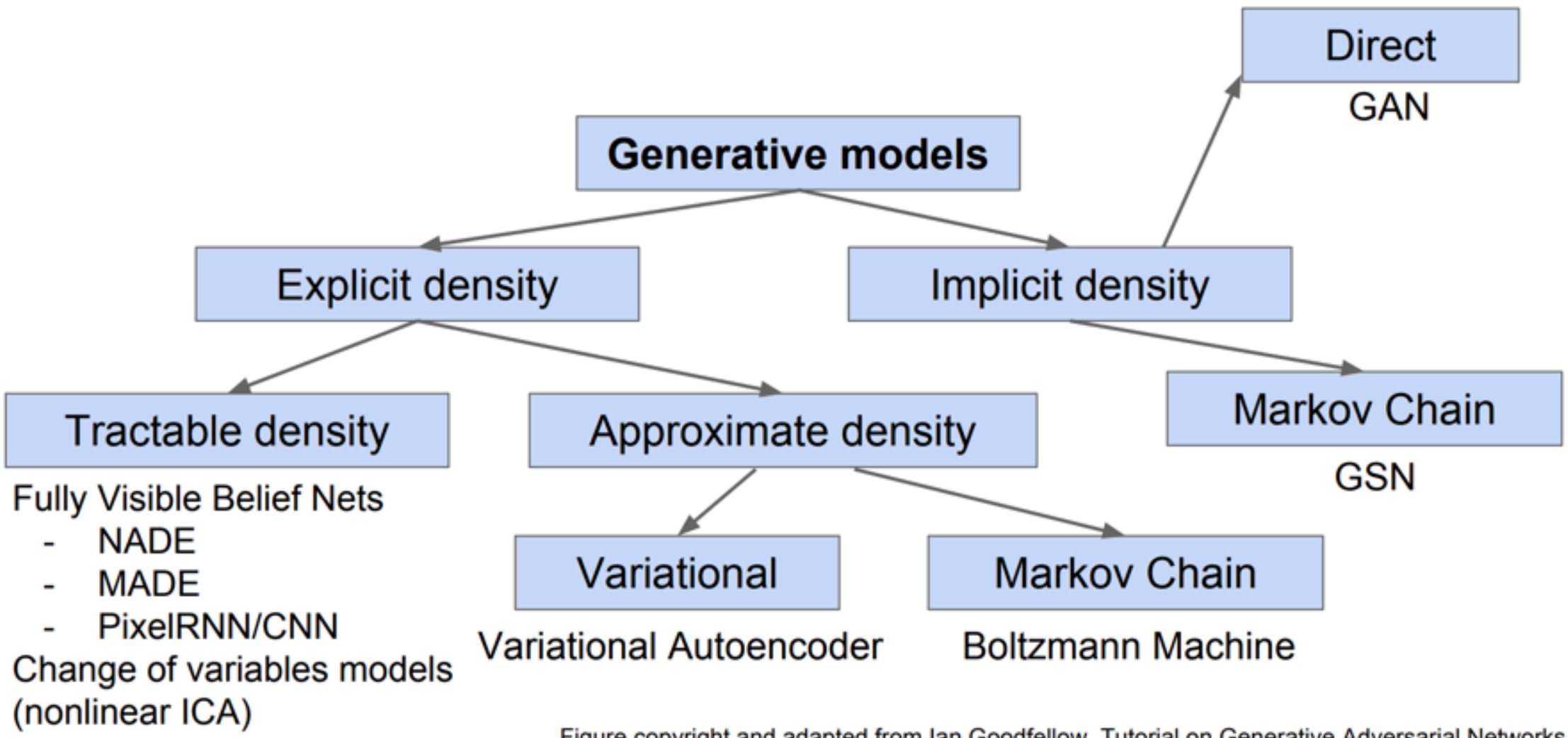


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PIXEL CNN/RNN – BASIC IDEA

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

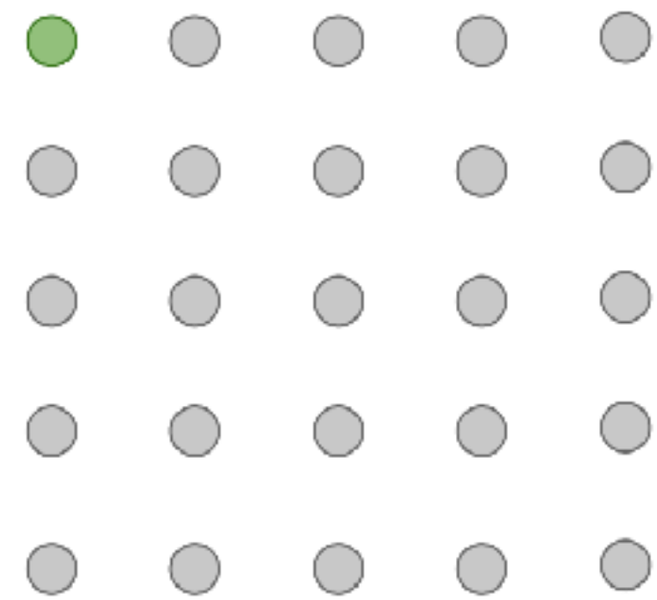
Complex distribution over pixel values => Express using a neural network!

Then maximize likelihood of training data

PIXEL RNN

Generate image pixels starting from corner

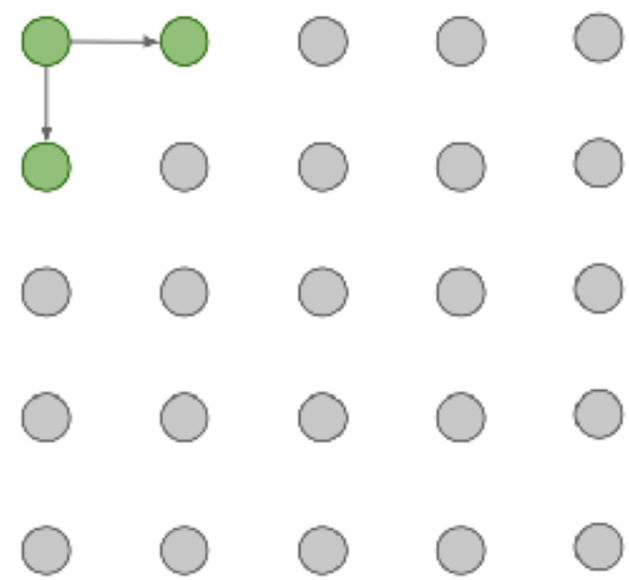
Dependency on previous pixels modeled using an RNN (LSTM)



PIXEL RNN

Generate image pixels starting from corner

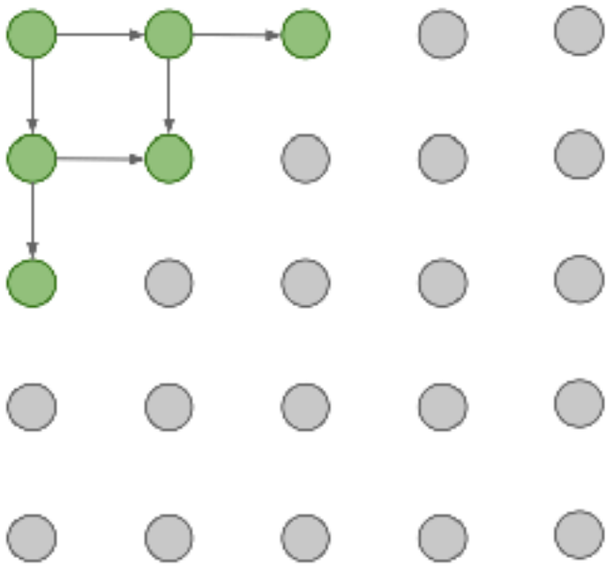
Dependency on previous pixels modeled using an RNN (LSTM)



PIXEL RNN

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

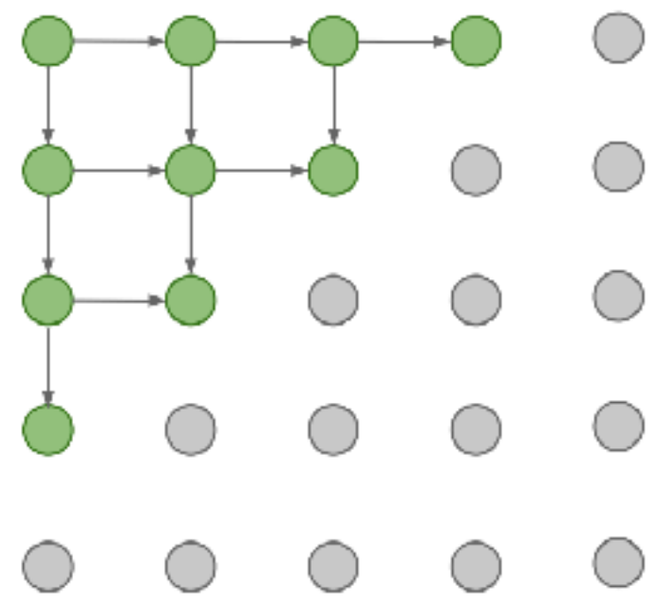


PIXEL RNN

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!



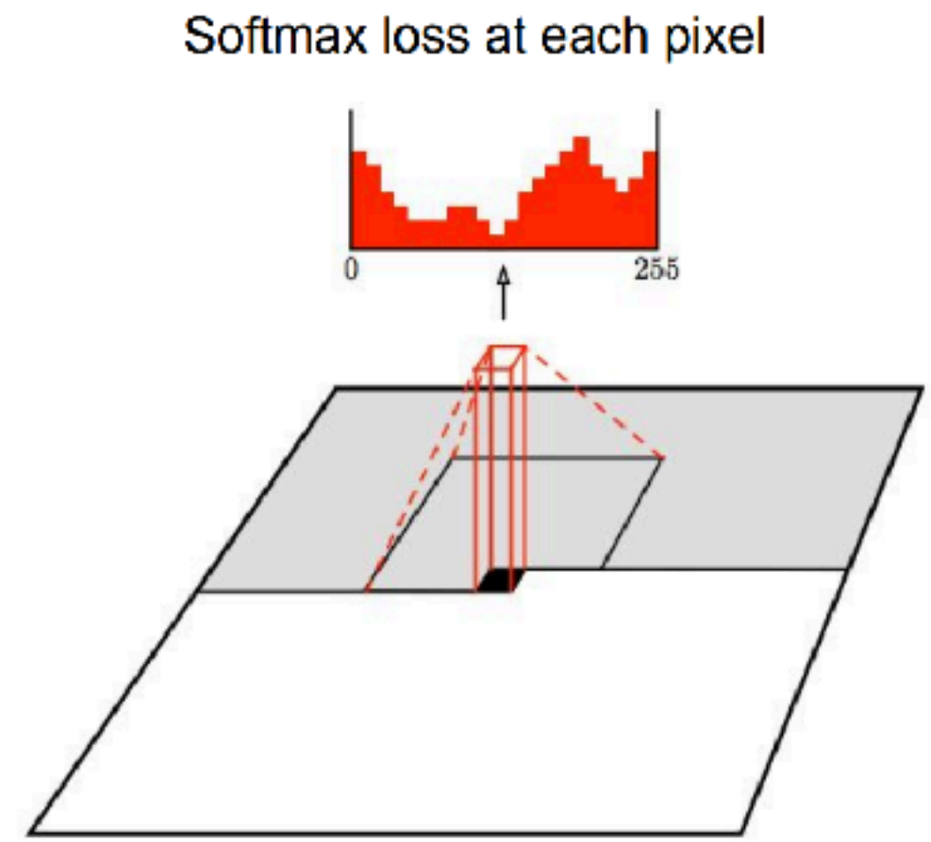
PIXEL CNN

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$



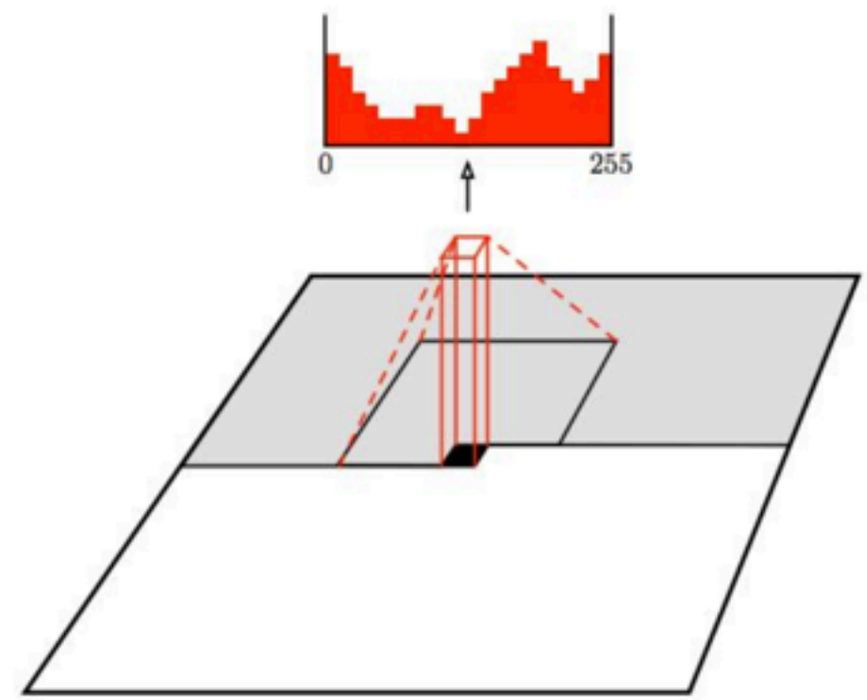
PIXEL CNN

Still generate image pixels starting from corner

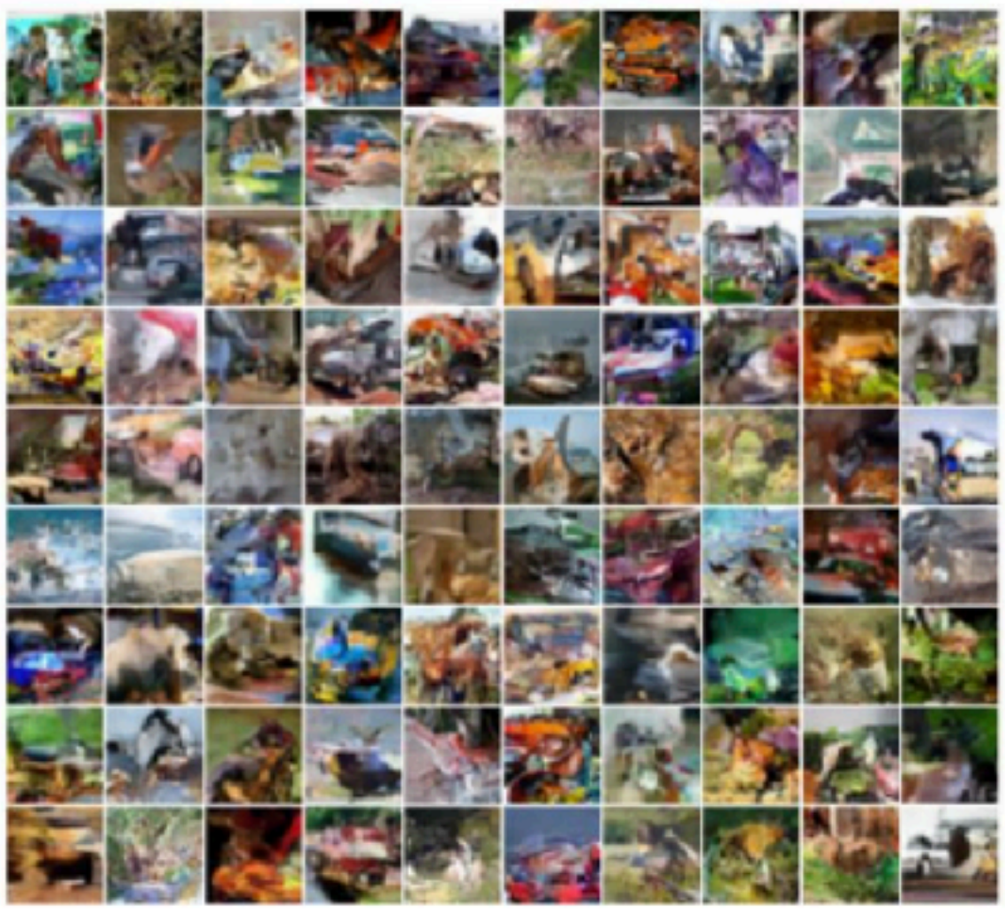
Dependency on previous pixels now modeled using a CNN over context region

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially
=> still slow



GENERATION SAMPLES



32x32 CIFAR-10

32x32 ImageNet

IMPROVEMENTS FOR PIXEL CNN/ PIXEL RNN

Pros:
- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Con:
- Sequential generation => slow

Improving PixelCNN performance
- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

See
- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

VARIATIONAL AUTOENCODER (VAE)

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

AUTOENCODER: RECAP

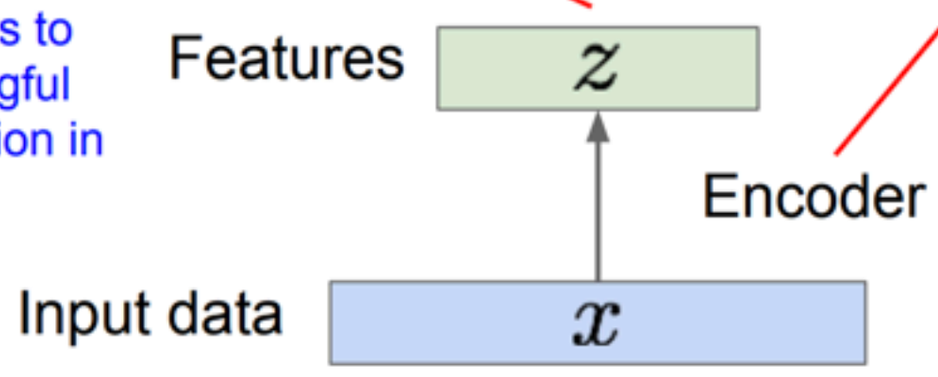
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

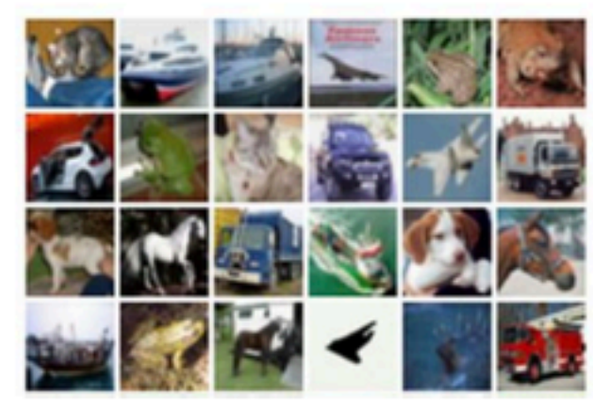
z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





AUTOENCODER: RECAP

Some background first: Autoencoders

Train such that features can be used to reconstruct original data

Doesn't use labels!

L2 Loss function:

$$\|x - \hat{x}\|^2$$

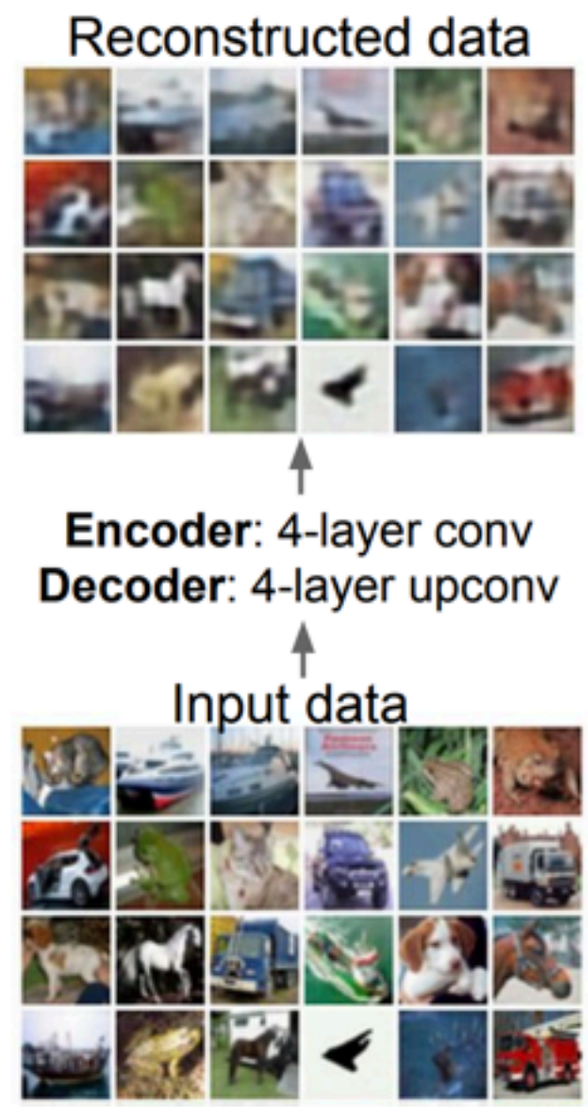
Reconstructed input data $\hat{x}$

Decoder

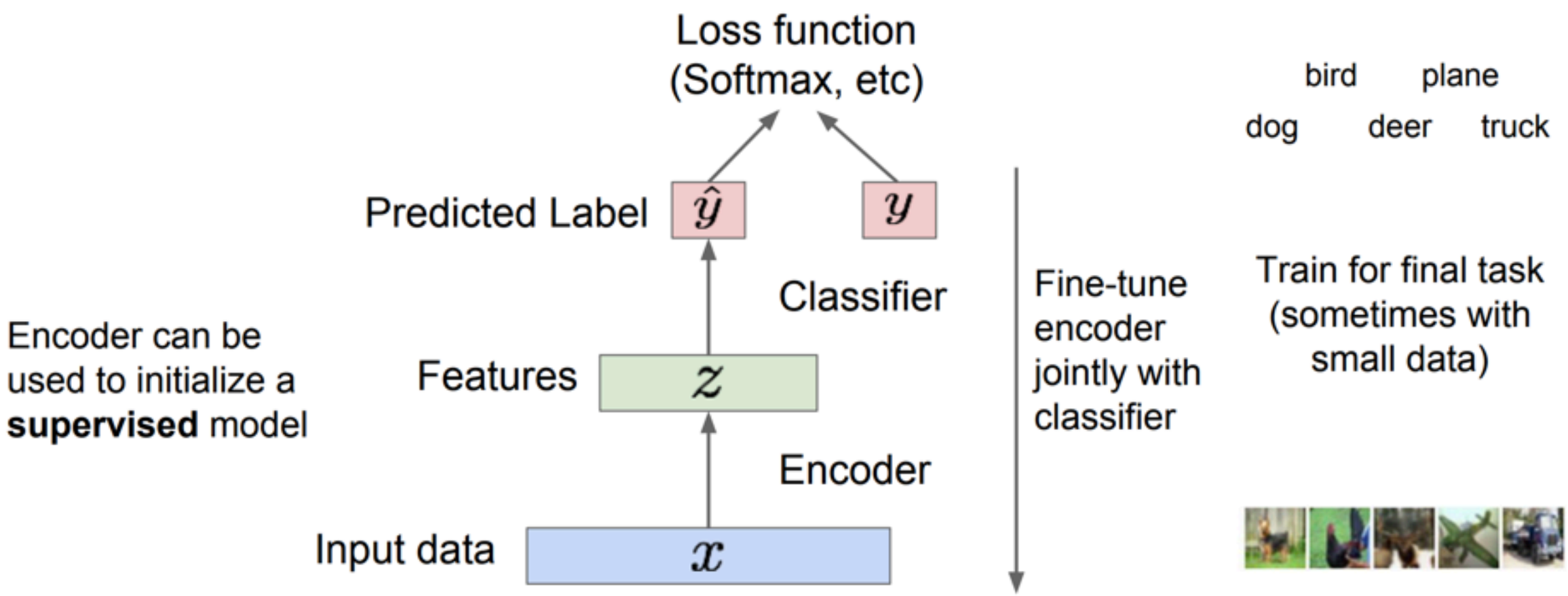
Features z

Encoder

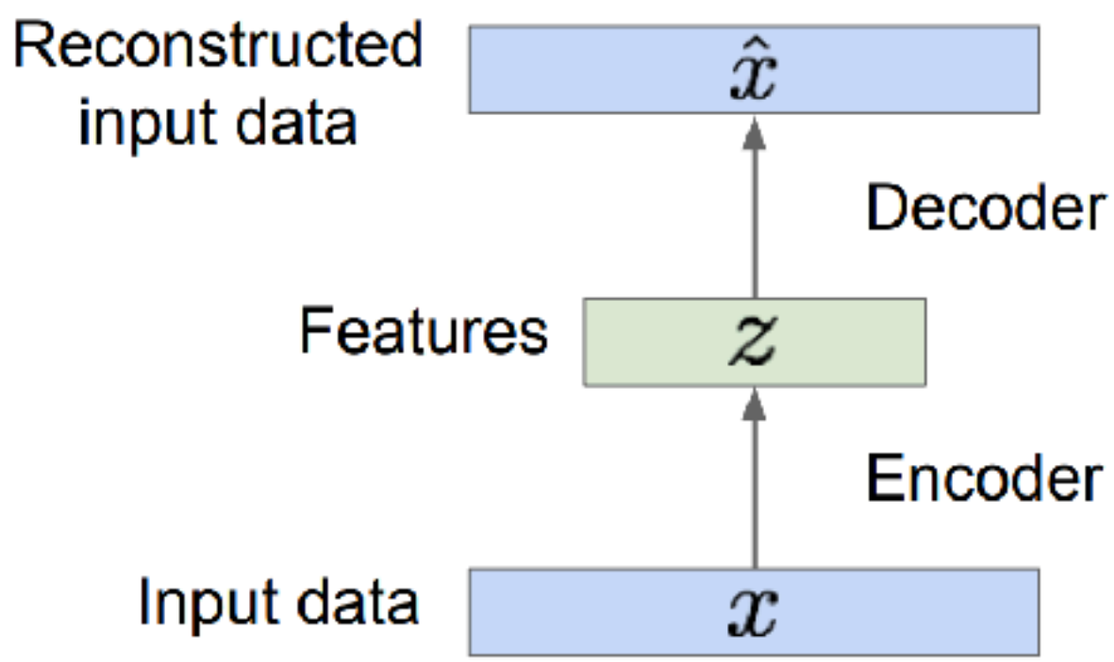
Input data x



AUTOENCODER: RECAP



AUTOENCODER: RECAP



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

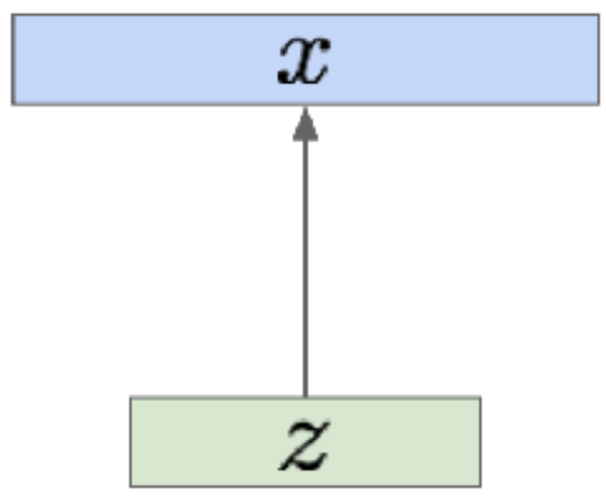
Features capture factors of variation in training data. Can we generate new images from an autoencoder?

IDEA OF VAE

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

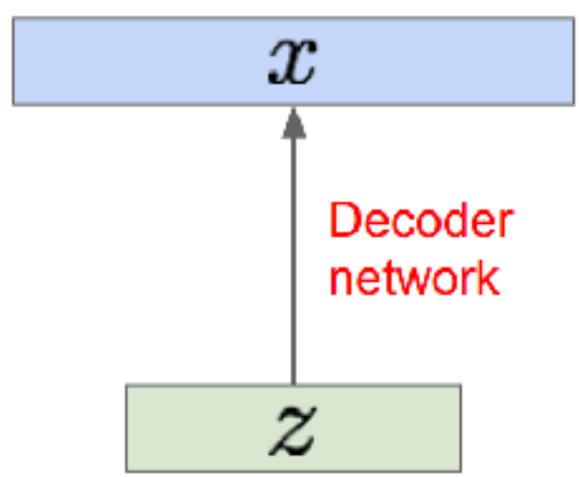


Sample from true prior $p_{\theta^*}(z)$

VARIATIONAL AUTOENCODER

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$



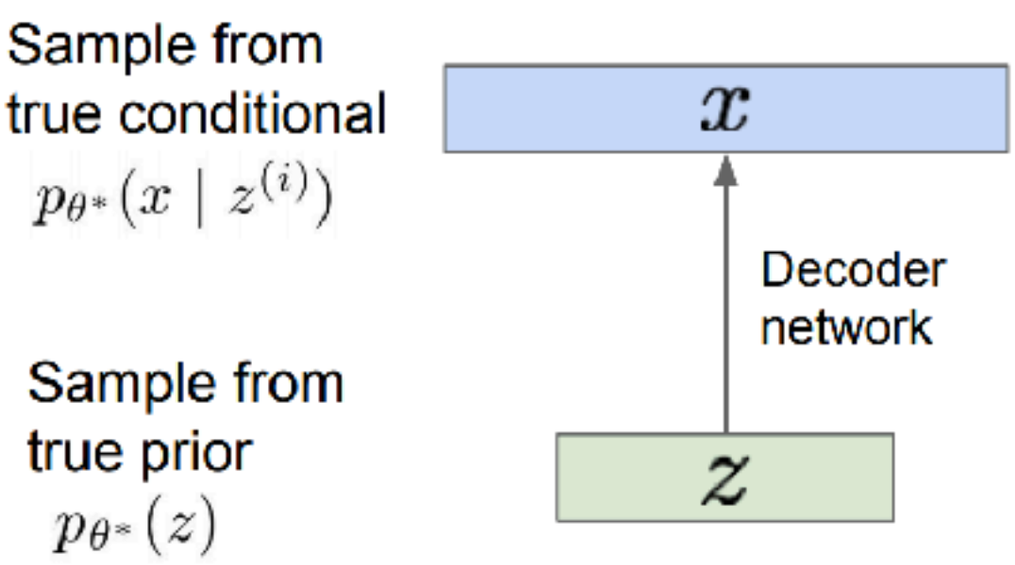
We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

LEARNING VAE IS INTRACTABLE



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Q: What is the problem with this?

Intractable!

VAE: WHY IS IT INTRACTABLE?

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Simple Gaussian prior

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

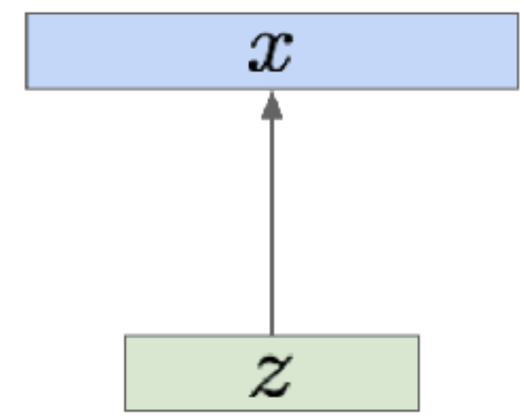
Decoder neural network

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Intractible to compute
p(x|z) for every z!

Sample from
true conditional
$p_{\theta^*}(x \mid z^{(i)})$

Sample from
true prior
$p_{\theta^*}(z)$



VAE: WHY IS IT INTRACTABLE?

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

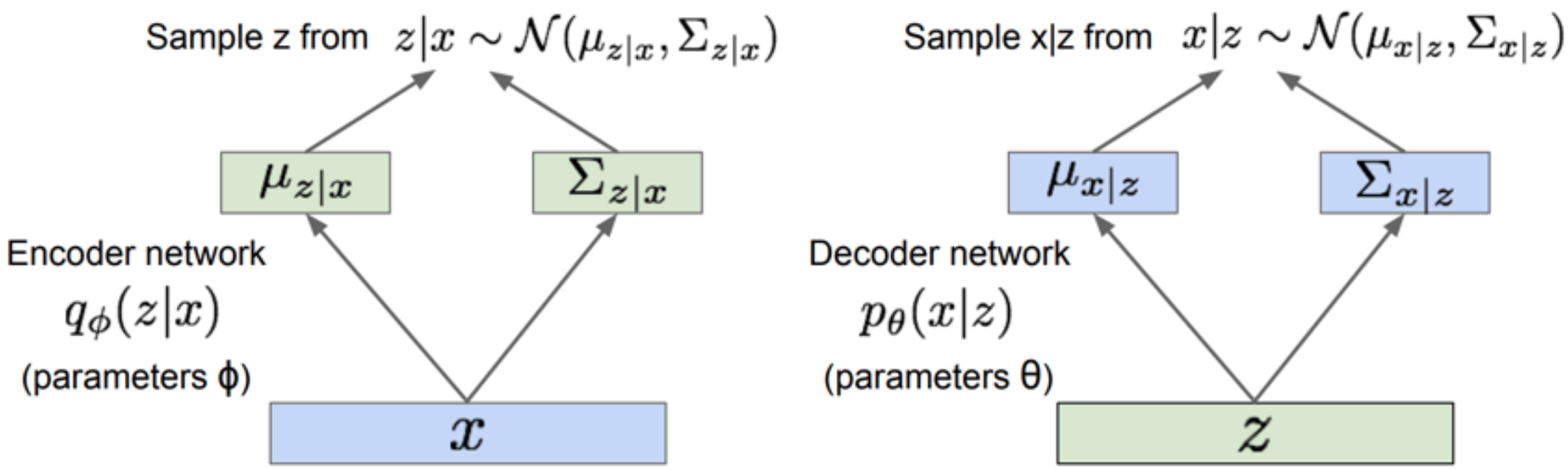
Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

VAE: WHAT IS THE SOLUTION?

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Encoder and decoder networks also called "recognition"/"inference" and "generation" networks

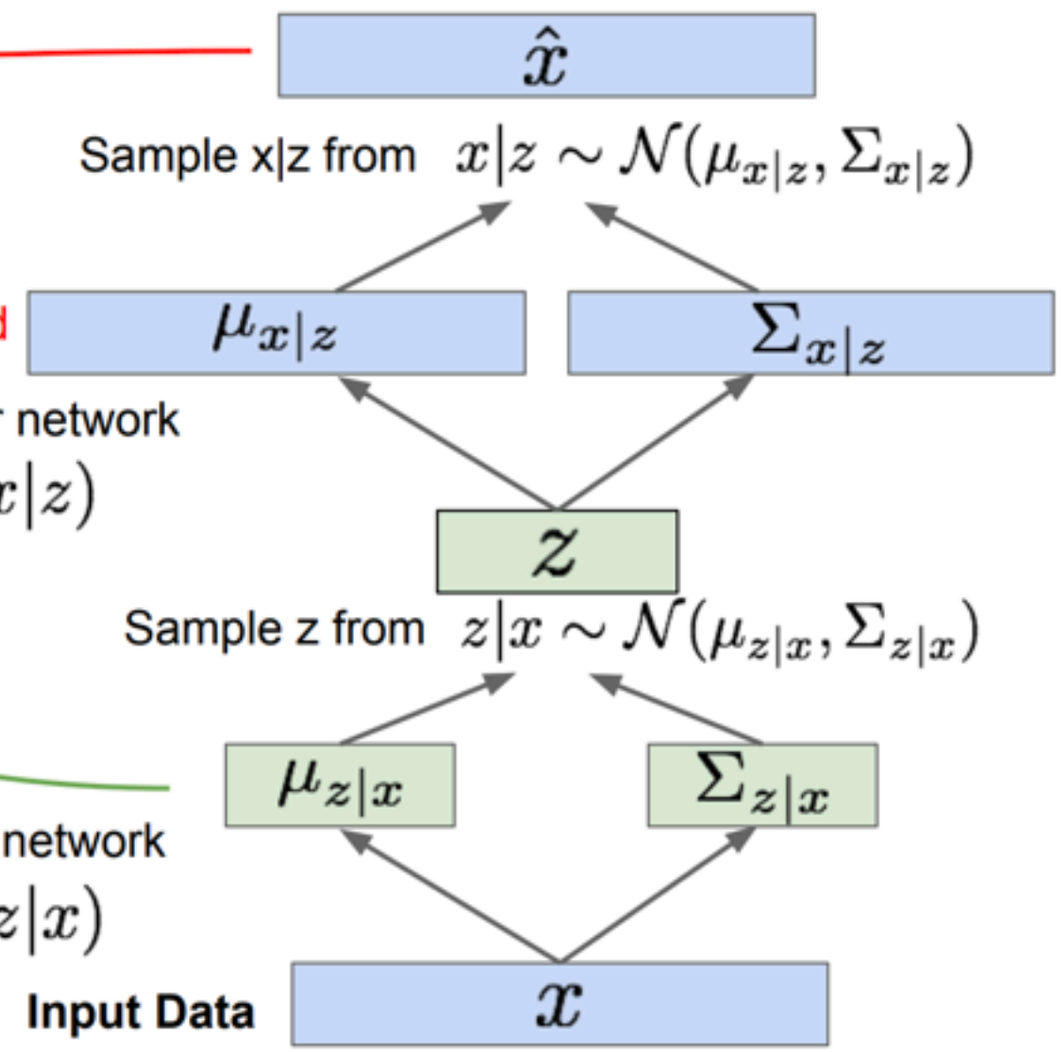
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VAE: WHAT IS THE SOLUTION?

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Maximize likelihood of original input being reconstructed

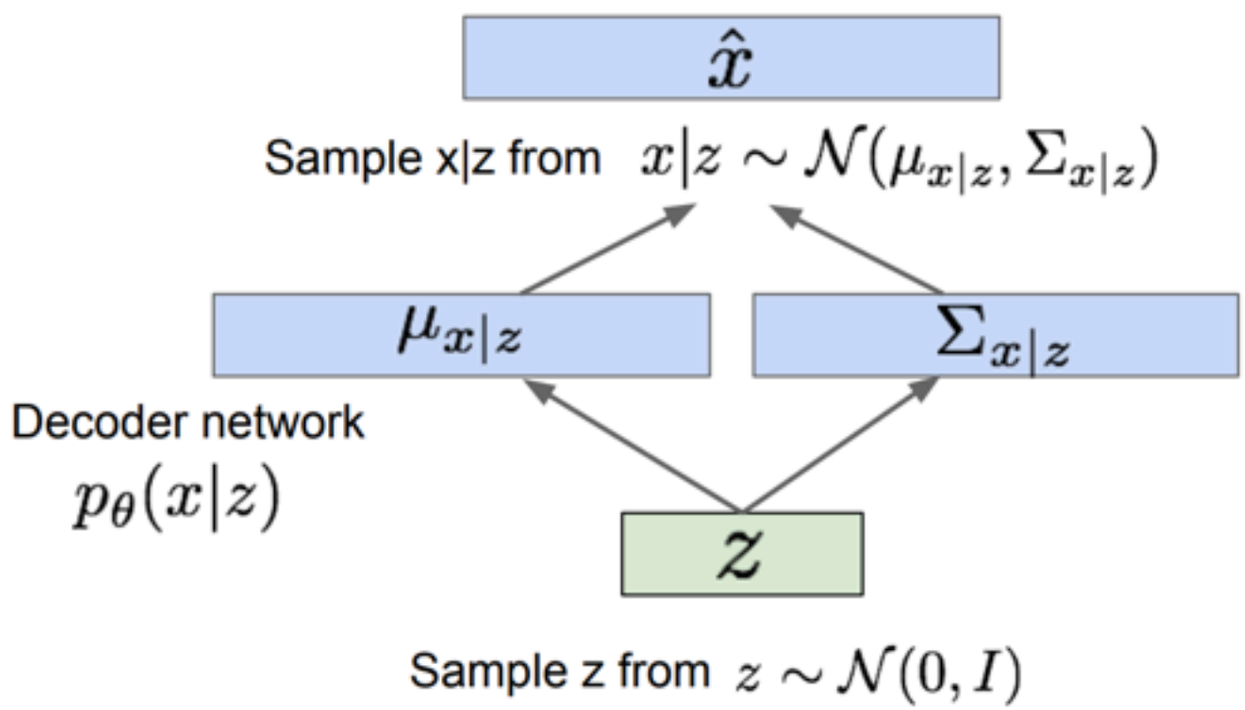


Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!

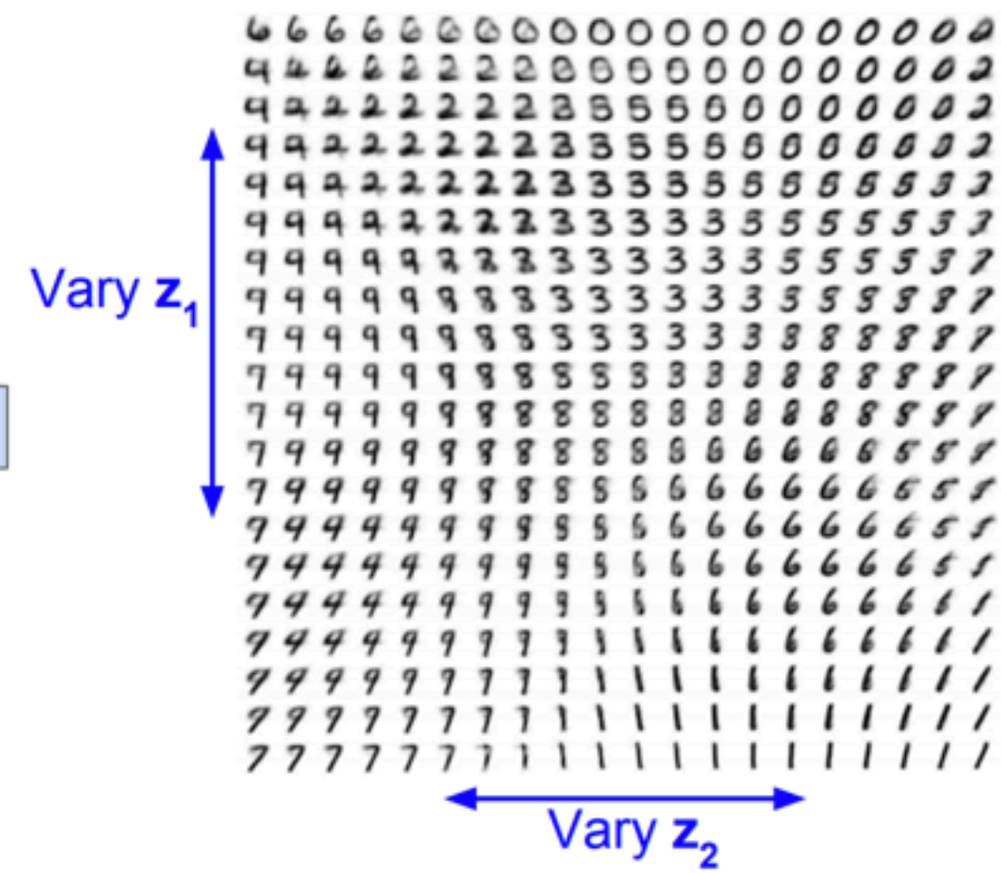
VAE: GENERATING DATA

Use decoder network. Now sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d **z**

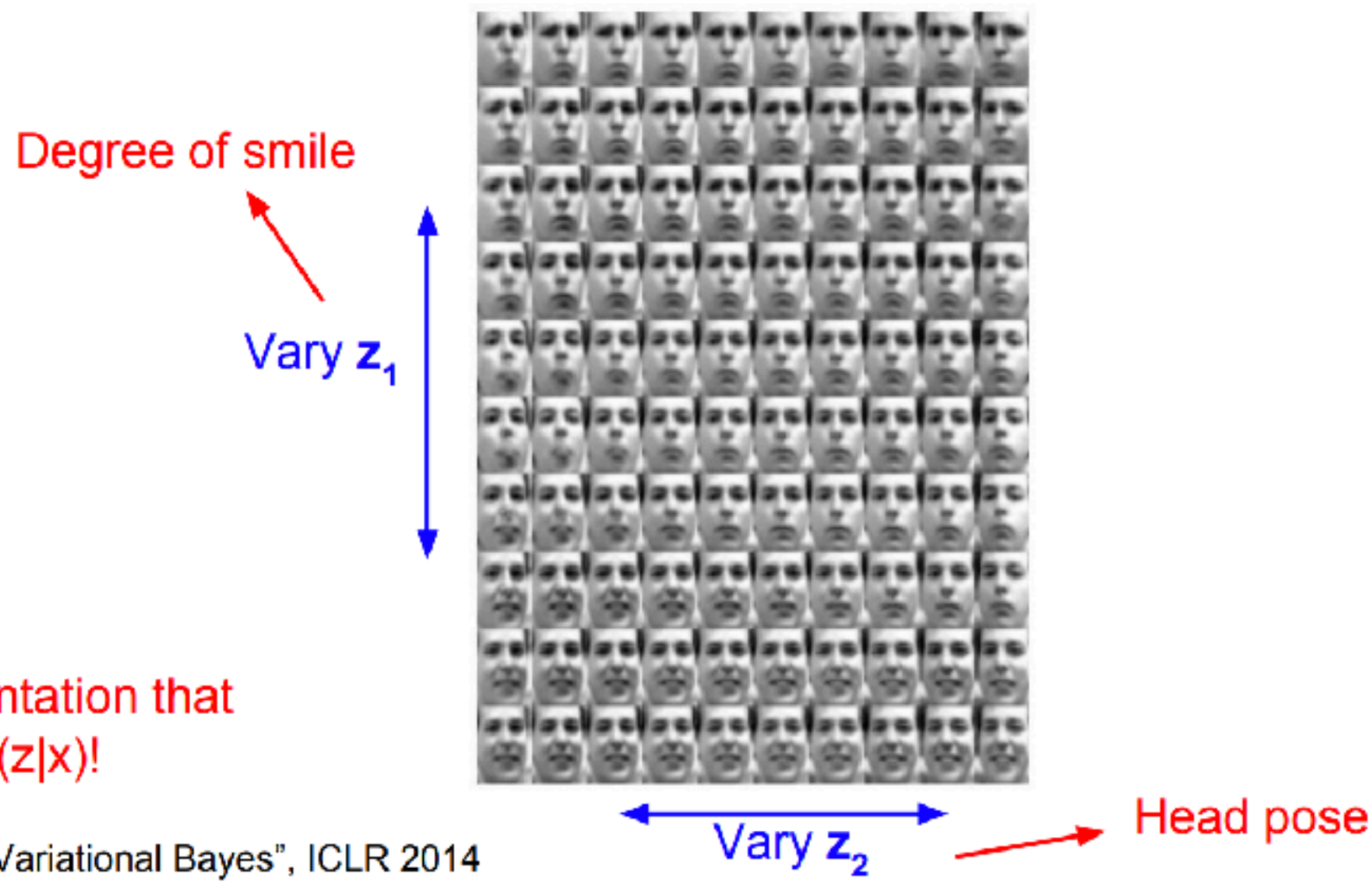


VAE: GENERATING DATA

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

Also good feature representation that
can be computed using $q_\phi(z|x)$!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

IMPROVEMENTS FOR VAE

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Allows inference of q(z|x), can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
- Incorporating structure in latent variables

GENERATIVE ADVERSARIAL NETWORKS (GAN)

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: don't work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

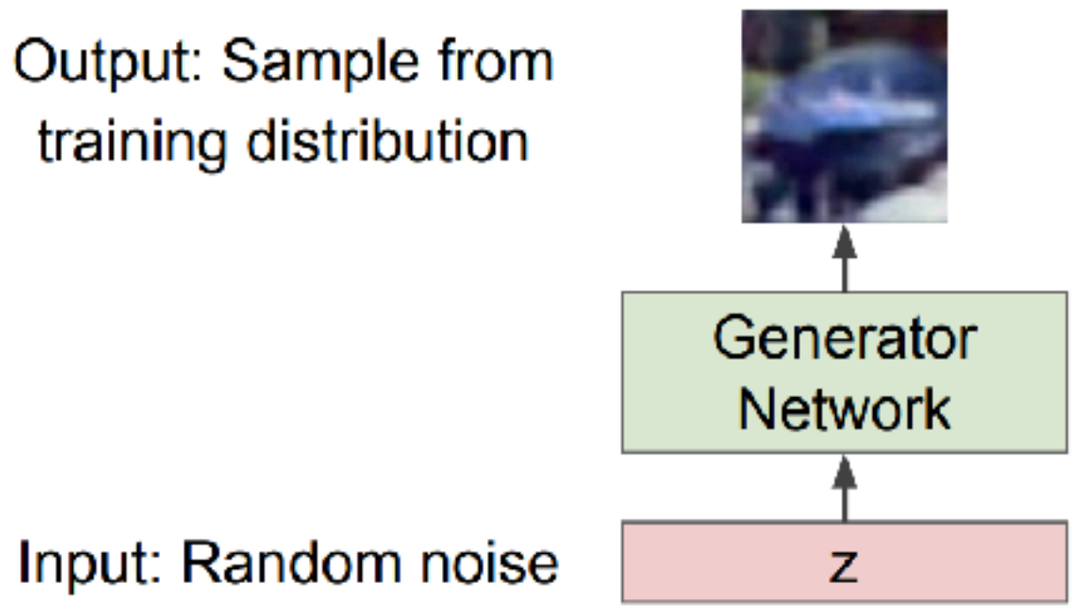
GENERATIVE ADVERSARIAL NETWORKS (GAN)

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

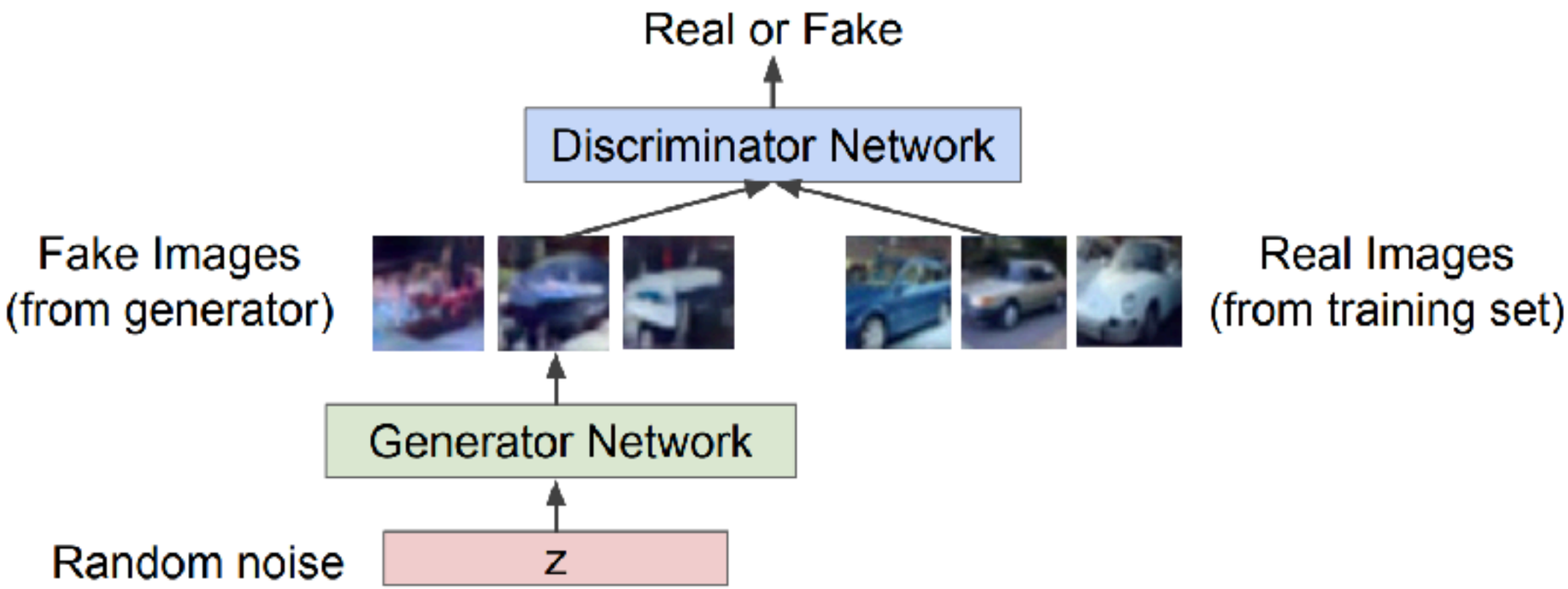
Q: What can we use to represent this complex transformation?

A: A neural network!



GENERATIVE ADVERSARIAL NETWORKS

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



GAN: A TWO PLAYER GAME

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

GAN: A TWO PLAYER GAME

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

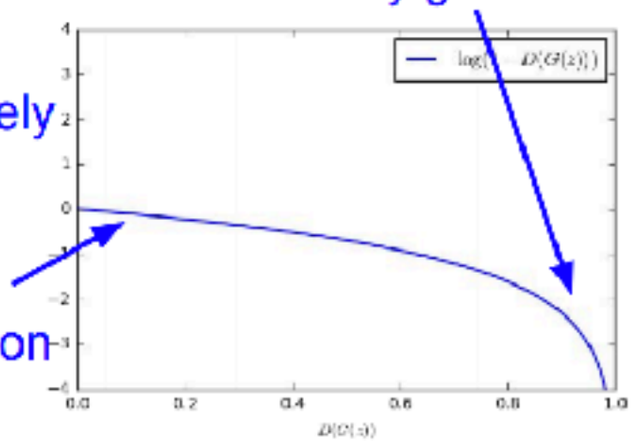
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



GAN: A TWO PLAYER GAME

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

Alternate between:

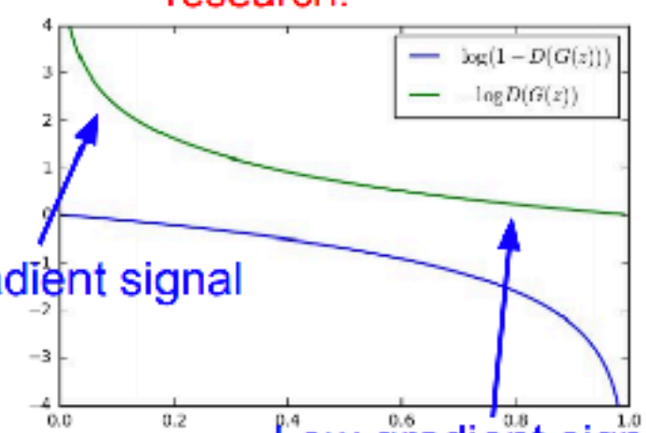
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

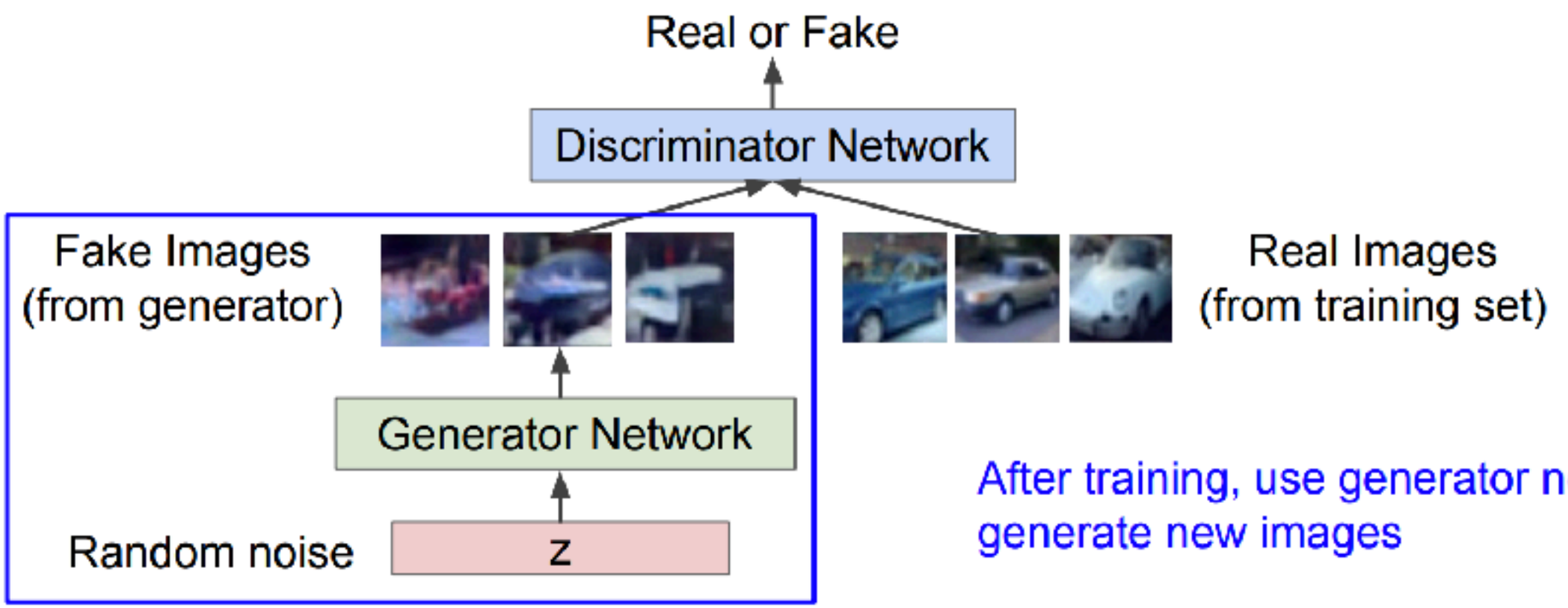
$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



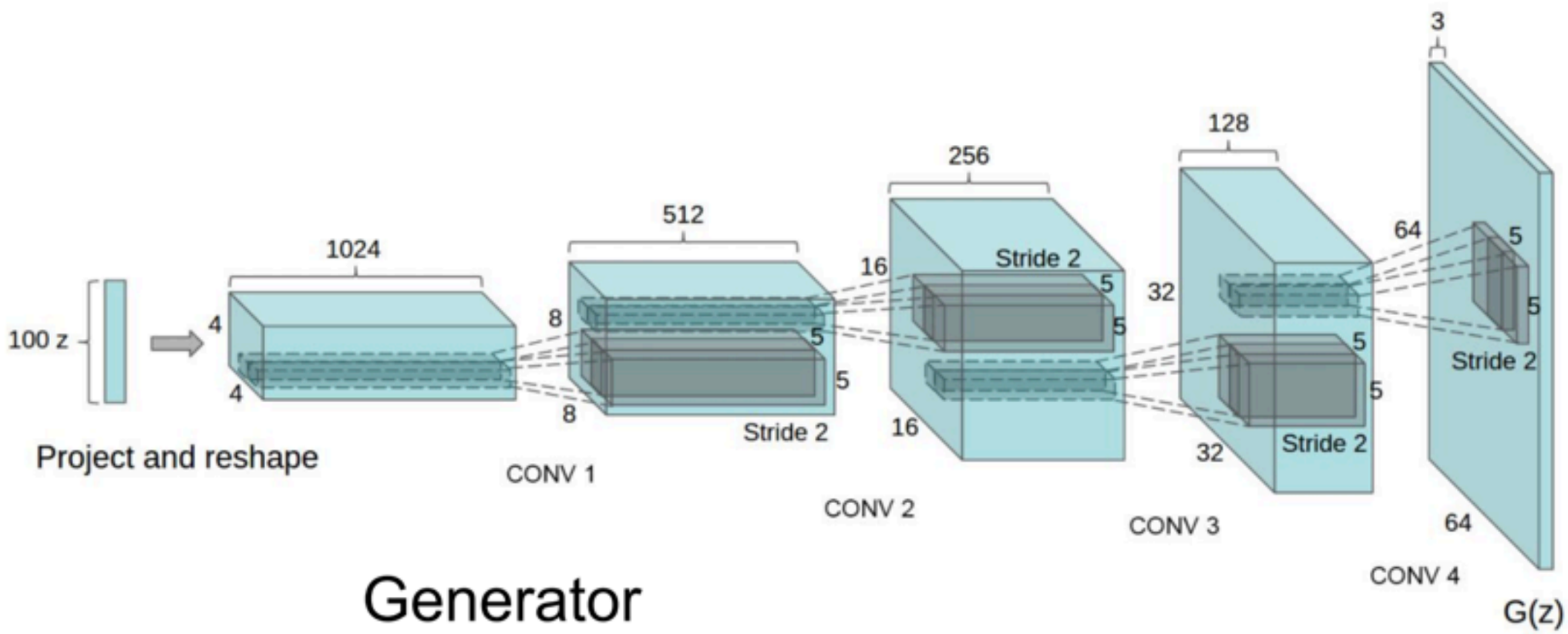
GAN: A GENERATIVE MODEL

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



After training, use generator network to generate new images

DCGAN: DEEP CONVOLUTIONAL GAN



EXPLOSION OF GAN

"The GAN Zoo"

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

https://github.com/hindupuravinash/the-gan-zoo

IMPROVEMENTS OF GAN

Don't work with an explicit density function
Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:
- Beautiful, state-of-the-art samples!

Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$

Active areas of research:
- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

RECAP: GENERATIVE MODELS

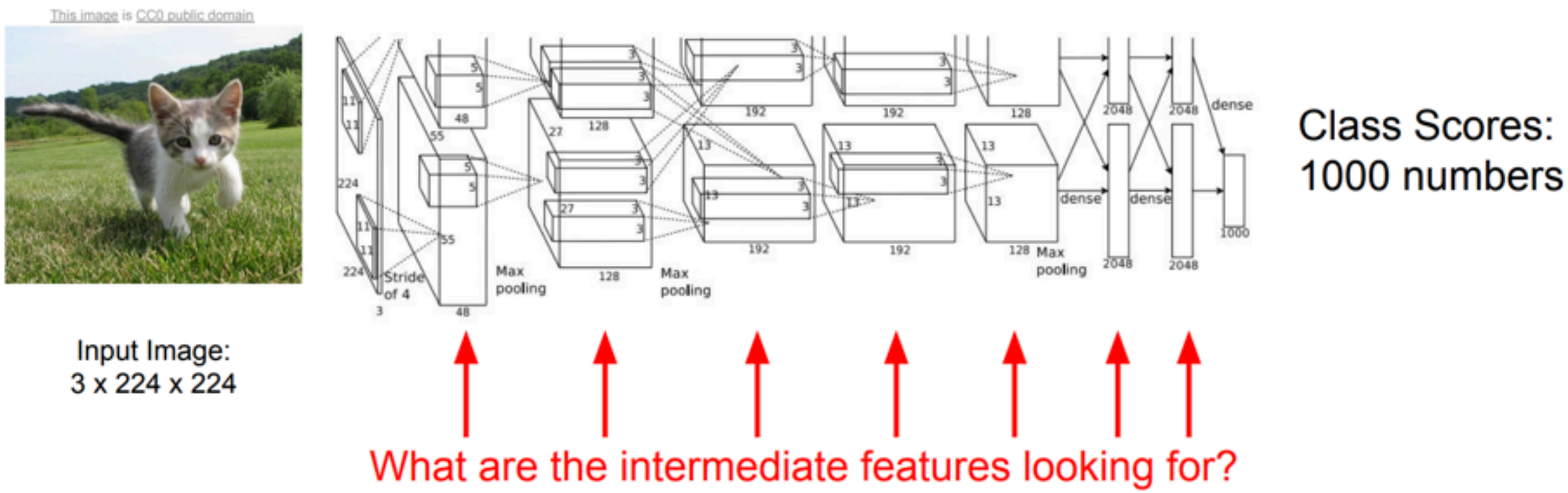
Generative Models

- PixelRNN and PixelCNN Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

Also recent work in combinations of these types of models! E.g. Adversarial Autoencoders (Makhanzi 2015) and PixelVAE (Gulrajani 2016)

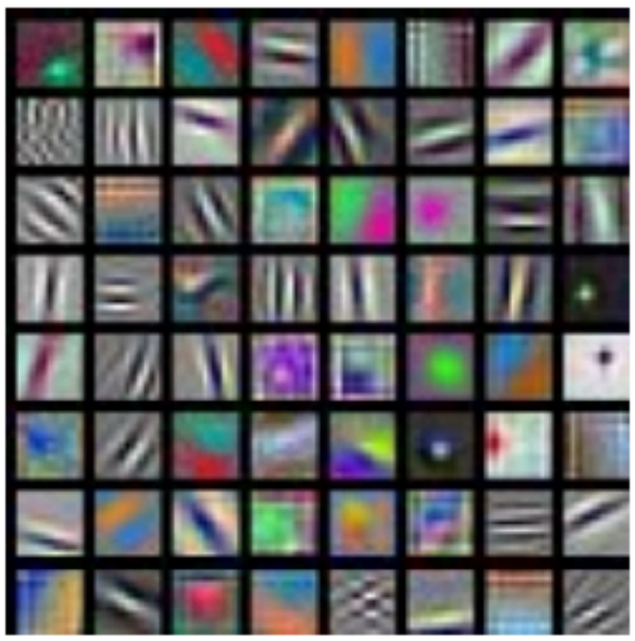
CNN VISUALIZATION

WHAT IS THE CNN LEARNING?



- Visualize Filters/ Kernels
- Visualize Activation Maps

FIRST LAYER FILTER VISUALIZATION



AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

VISUALIZE ALL THE LAYERS

Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)

layer 1 weights

16 x 3 x 7 x 7

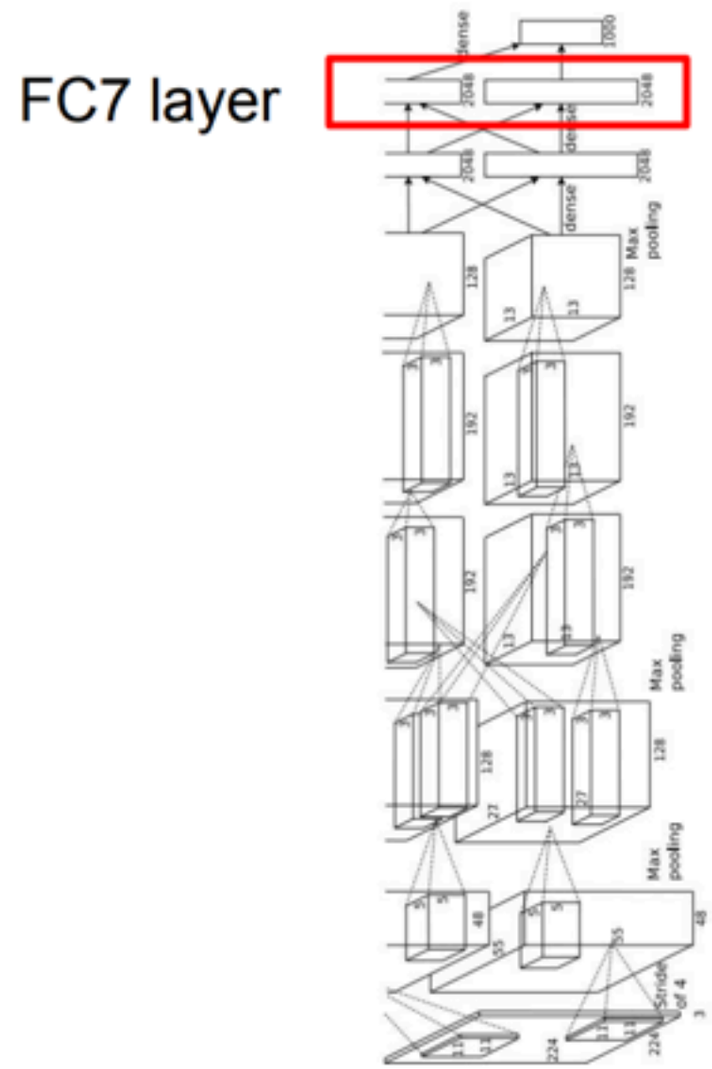
layer 2 weights

20 x 16 x 7 x 7

layer 3 weights

20 x 20 x 7 x 7

FULLY CONNECTED LAYER VISUALIZATION



4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

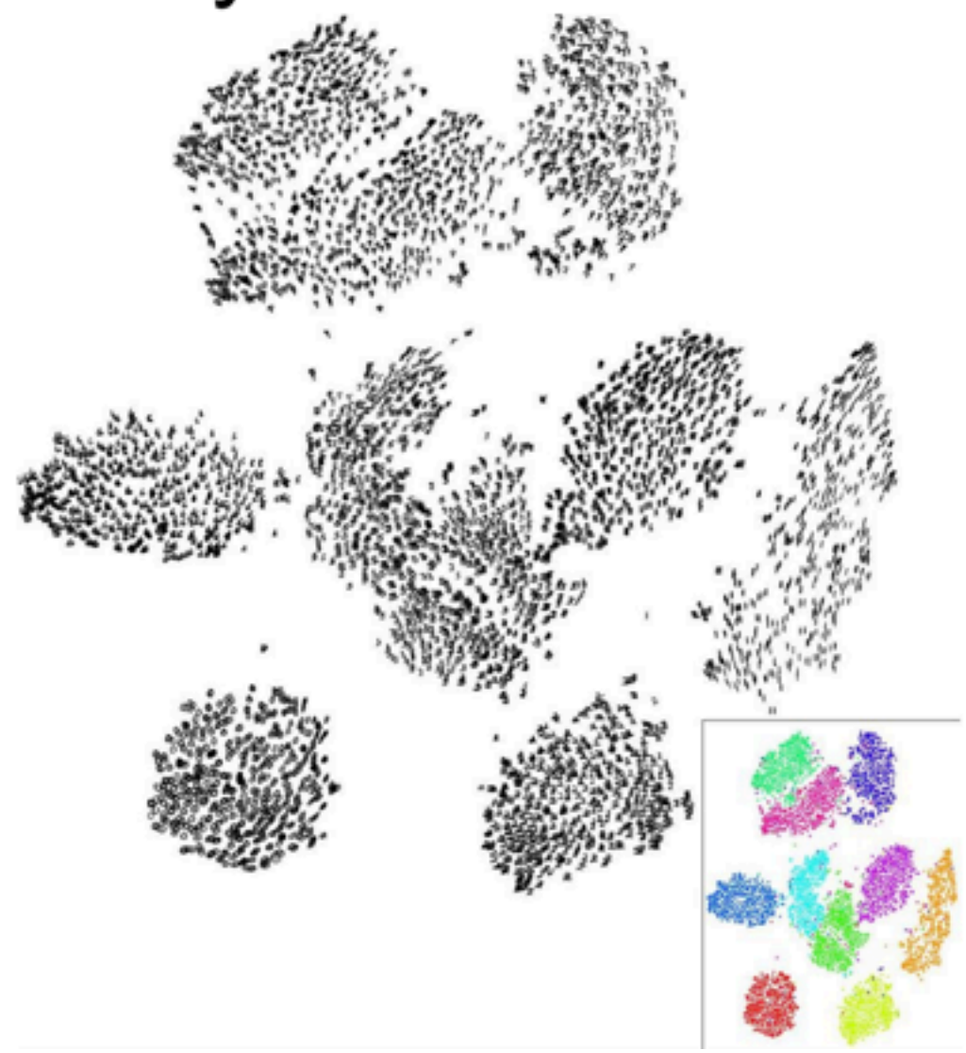
FULLY CONNECTED LAYER VISUALIZATION

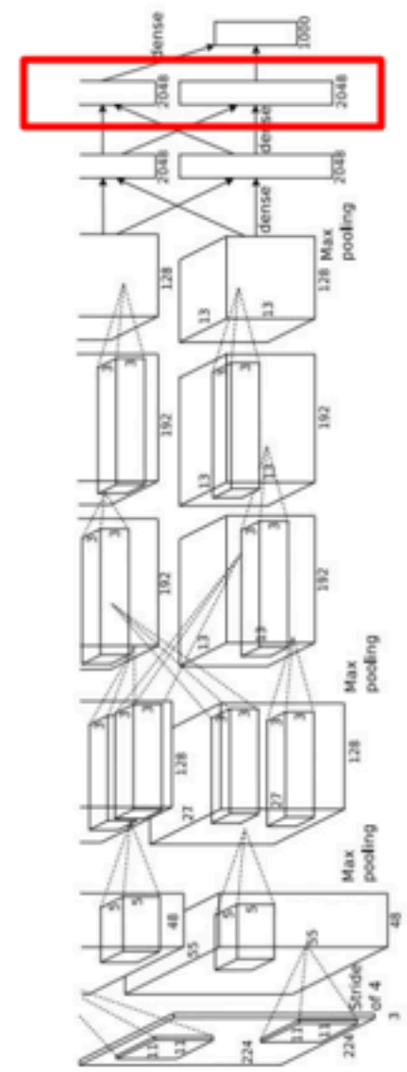
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

More complex: **t-SNE**

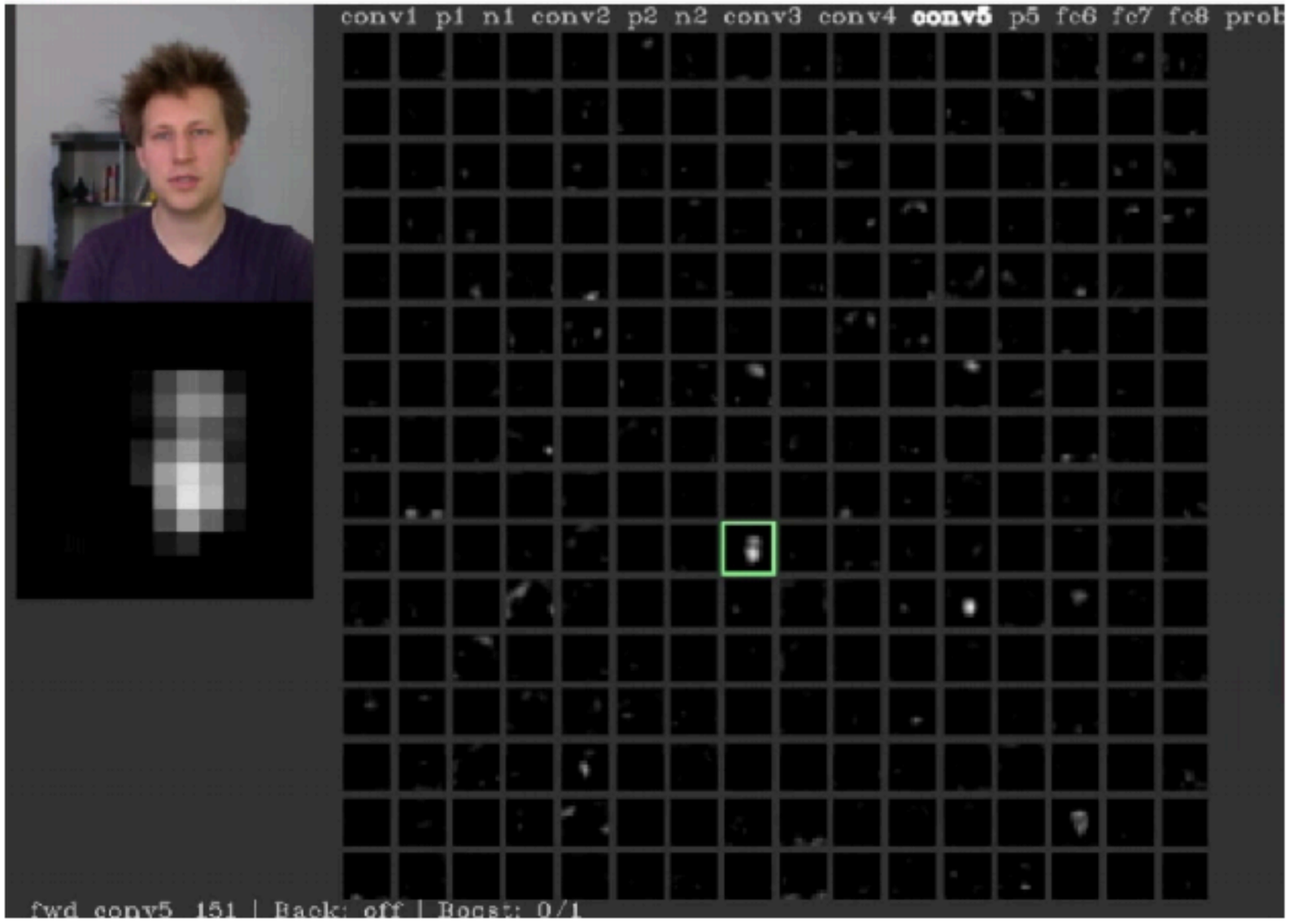




Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Figure copyright Laurens van der Maaten and Geoff Hinton, 2008. Reproduced with permission.

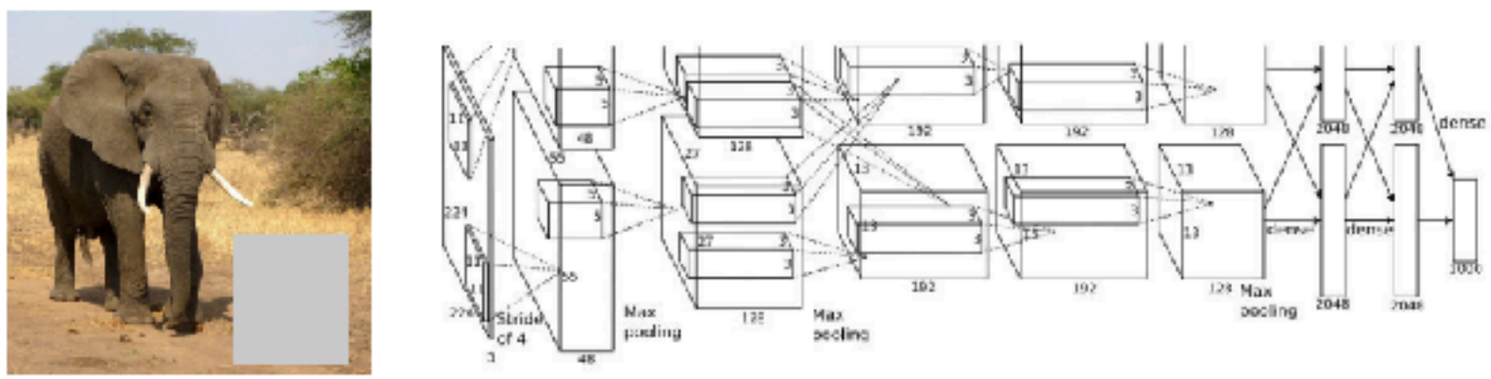
VISUALIZING ACTIVATIONS

conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images

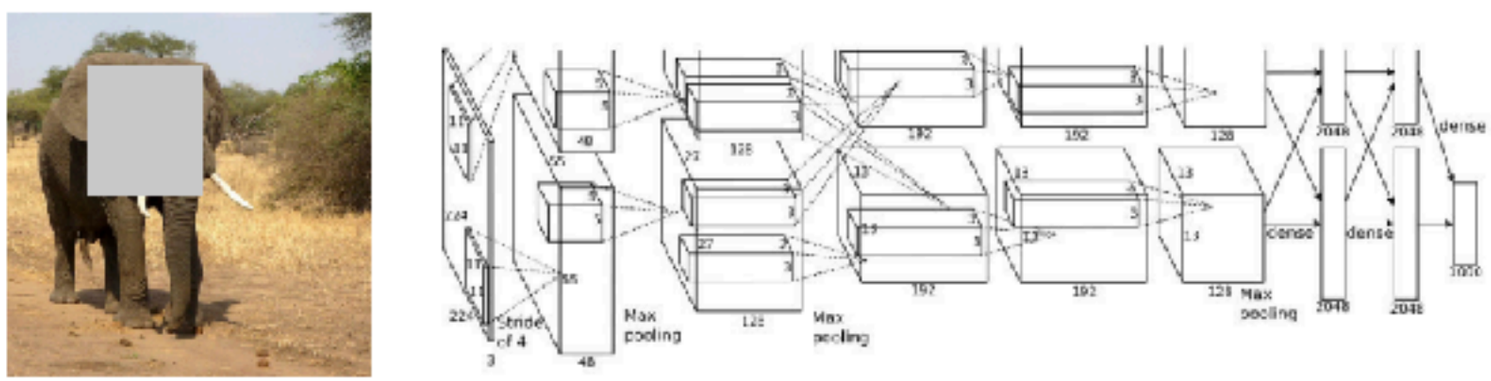


WHICH PIXELS MATTER: SALIENCY

Mask part of the image before feeding to CNN,
check how much predicted probabilities change



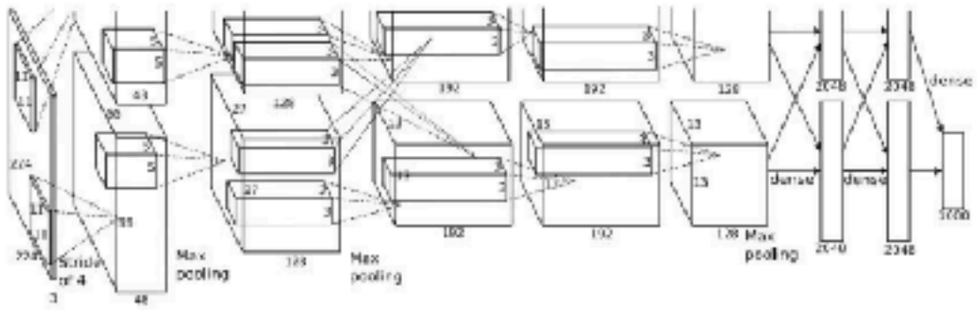
P(elephant) = 0.95



P(elephant) = 0.75

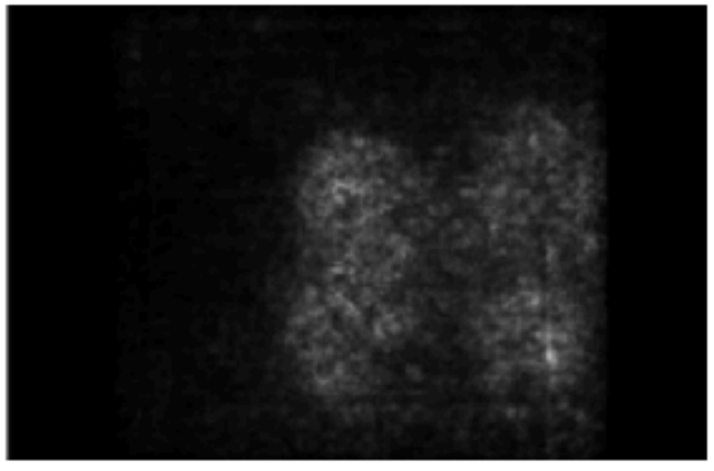
WHICH PIXELS MATTER: SALIENCY VIA BACKPROP

Forward pass: Compute probabilities



Dog

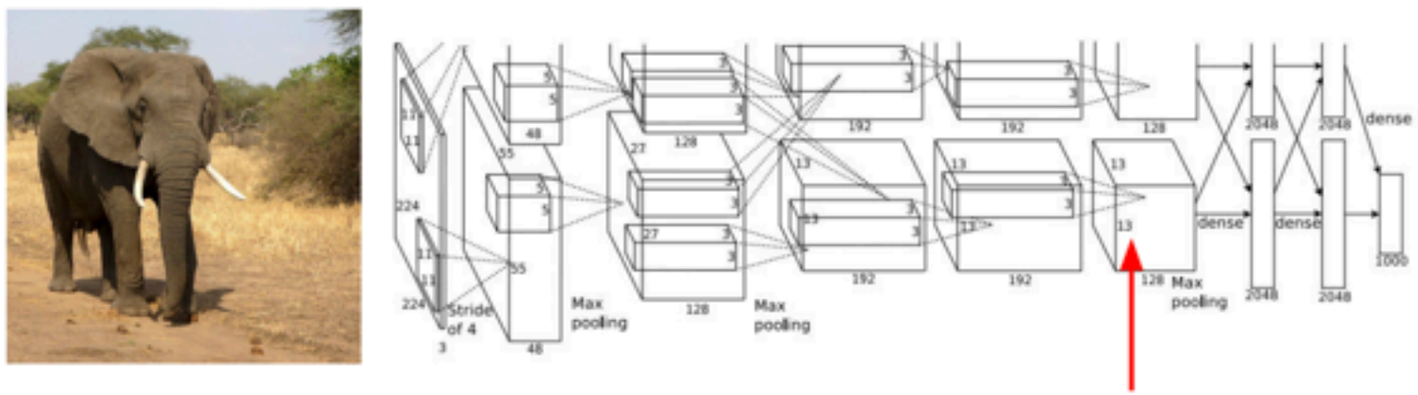
Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

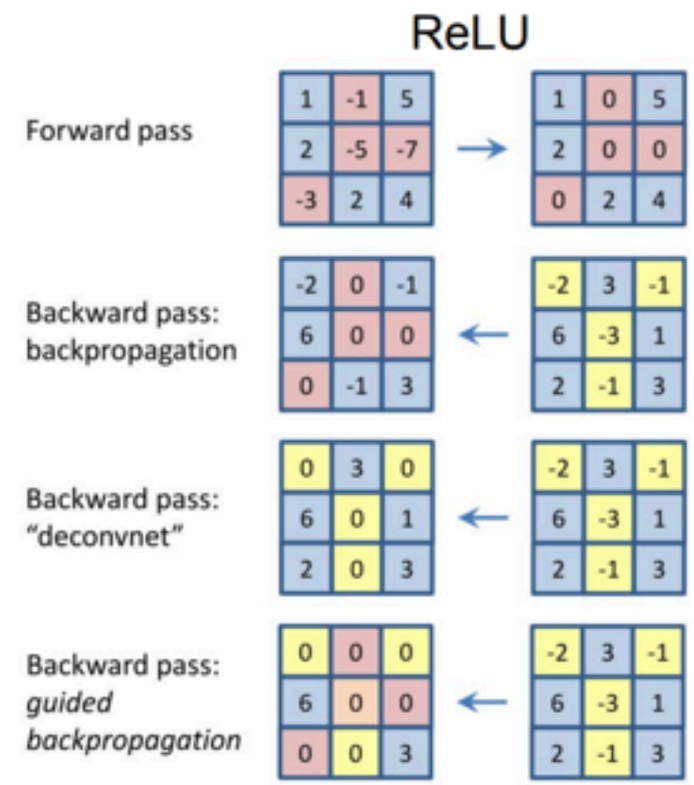
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

GUIDED BACKPROP



Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels



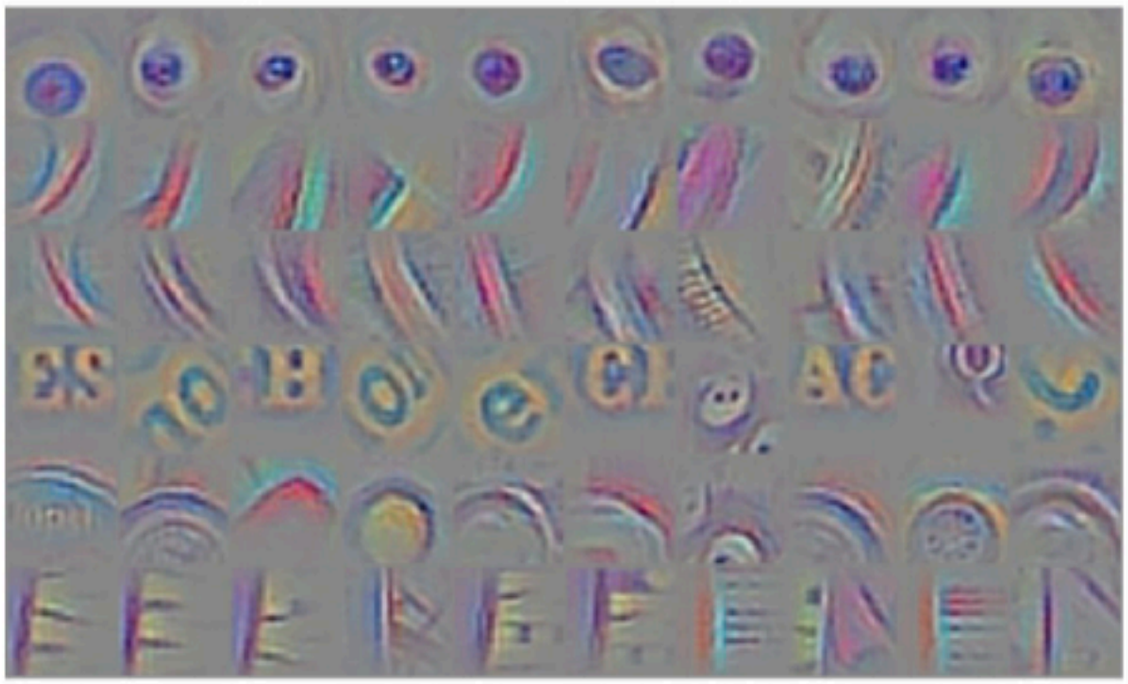
Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

GUIDED BACKPROP

Maximally activating patches
(Each row is a different neuron)



Guided Backprop