Deep Learning

Stéphane Gentric, Ph. D.
Research Unit Manager

IDEMIA

Feb 2020

Mini Project

1/ learn your best gender CNN detector.

Accuracy on test db > 85%

2/ find on the web, the face image of a person

- Not androgynous
- Build a 48*48 face image

3/ write a small stand alone script.

- Load your model
- Load a face image
- Estimate gender

ckpt.save("./my_model")

IrfanView

ckpt.restore("./ my_model-1")

Mini Project

4/ Modify your image in order to mislead your CNN detector.

- Load your model
- Load your image and a wrong label
- Add a Variable DX to the input in your graph
  $X_{mod} = X + DX$
- Train only the DX Variable
- Show Modified Image and DX Image
- Save the Modified Image. X_{mod}
- As DX is very small, build an amplified image, A=abs(DX)*50
- save the A Image.

5/ Test the Modified Image with your small script.

6/ Add a L2 constraint to DX to make it small

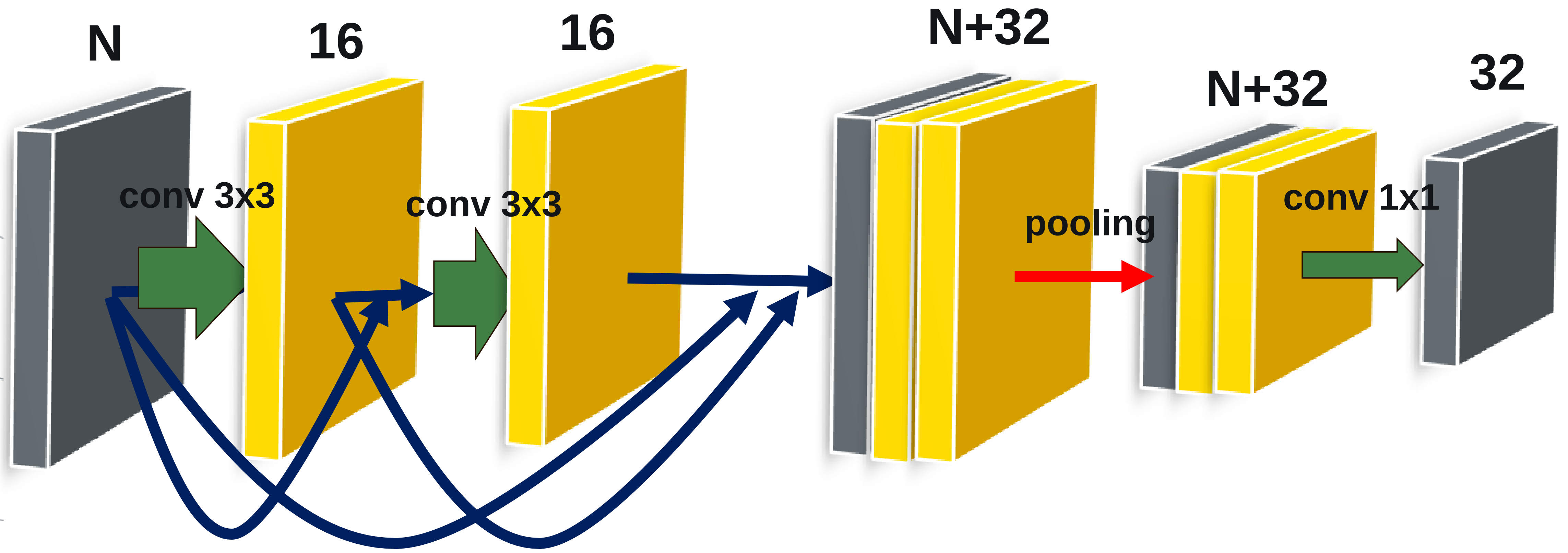
```
y = model(X+DX,False)

grads = tape.gradient(loss, [DX])

Import Pillow
Image.show()
```

Provide 3 images (raw or png)
adv_nom1_nom2_ori.raw
adv_nom1_nom2_mod.raw
adv_nom1_nom2_amp.raw
Send them to
stephane.gentric@telecom-paris.fr

DensNet

nb_conv_per_block = 2
Nbfilter = 16

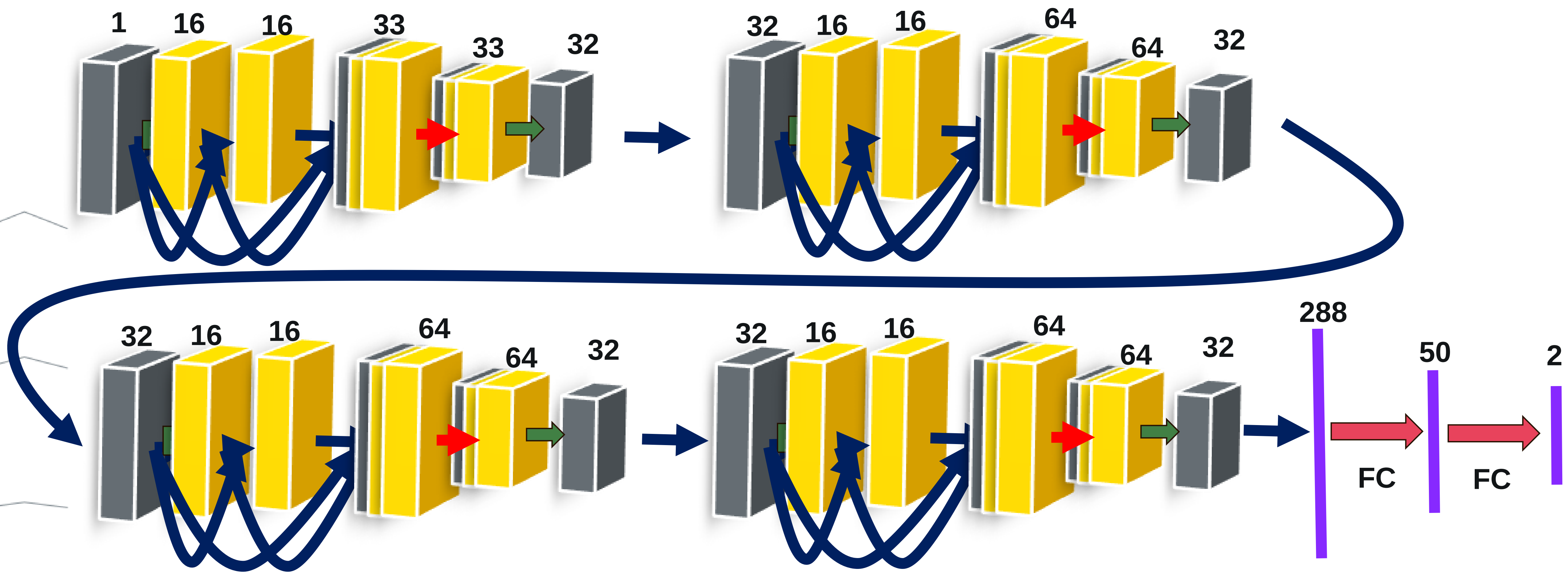
A DenseNet Block



[? , H , W , N] => [? , H/2, W/2 , 32]

DensNet for gender classification

nb_conv_per_block = 2
Nbfilter = 16



[? , 48 , 48 , 1] => [? , 3, 3 , 32] => [? , 2]

DensNet for gender classification

```
class DenseBlock(tf.Module):
   def __init__(self, name, nb_conv, nb_filter,
output_dim, filterSize, stride, dropout_rate=0.0):
      self.block = []
      for i in range(nb_conv):
         self.block.append(conv('%s cv%d'%(name,i),
nb_filter, filterSize, stride, dropout_rate))
      self.mp = maxpool('pool', 2)
      self.cv = conv('%s red' % name, output_dim, 1, 1,
dropout_rate)

   def __call__(self, x, log_summary, training):
      for bl in self.block:
         y = bl(x, log_summary, training)
         x = tf.concat([x, y], axis=3)
      x = self.mp(x)
      x = self.cv(x, log_summary, training)
      return x
```

```
class DenseNet(tf.Module):
   def __init__(self):
      self.unflat = Layers.unflat('unflat', 48, 48, 1)
      self.nbfilter, self.nb_block,
self.nb_conv_per_block, self.outdim = 3, 4, 16 ,32
      self.list = []
      for i in range(self.nb_block):
         self.list.append(Layers.DenseBlock('bl%d' % i,
self.nb_conv_per_block, self.nbfilter,self.outdim, 3,1,
DropOutRate))
      self.flat = Layers.flat()
      self.fc1 = Layers.fc('fc1', 50)
      self.fc2 = Layers.fc('fc2', 2)

   def __call__(self, x, log_summary, training):
      x = self.unflat(x)
      for dense_block in self.list:
         x = dense_block(x, log_summary, training) #
apply denseblock
      x = self.flat(x)
      x = self.fc1(x, log_summary,training)
      x = self.fc2(x, log_summary,training)
      return x
```

Meta Parameters Analysis

net	densnet
use BatchNorm	False
dbsize	100k

	batchsize	64						256								
	nb filter	4		16		32		4		16					32	
		.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.15	0.05	0.0	0.5	0.3
3			86,4%		88,4%											
	2		88,4%		90,9%											
	4		89,4%		91,1%											
	7				[illegible]											
	10				[illegible]			85,6%	92,0%	91,3%	93,1%				88,9%	
	20				[illegible]	54,4%	54,4%	89,4%	92,7%	KO	KO					
	40		91,5%		KO			KO	KO							
4	1		83,2%		87,3%				85,4%		90,5%	[illegible]				
	2	56,0%	84,3%	67,2%	88,9%	65,6%	90,4%	67,3%	88,9%	86,4%	91,9%	92,7%			88,5%	92,5%
	4	60,4%	85,5%	65,4%	91,2%	54,5%	91,1%	88,2%	91,6%	90,3%	92,7%	92,9%			87,9%	92,7%
	7	68,6%	89,7%	66,1%	91,4%	54,4%	89,6%	86,0%	92,0%	90,2%	92,9%	92,9%	92,5%	91,0%	77,9%	92,7%
	10	70,9%	90,3%	55,6%	90,9%	54,4%	90,1%	88,9%	92,2%	87,7%	92,9%	92,9%	92,2%	91,2%	85,7%	93,0%
	20	68,2%	91,2%	61,6%	90,6%	54,4%	90,8%	83,5%	92,6%	KO	KO	KO	KO	KO	KO	KO
	40		91,2%	KO	KO	KO	KO	KO	KO							

Deeper is Better

Tradeoff for DropOut

DropOut kills convergence for small batchsize

Too Big Network or Batch Size

Meta Parameters Analysis

net	convnet
dbsize	100k
block	4

Need BatchNorm for 8+ convolutions

Bigger BatchSize for 80 convolutions

	use BatchNorm	False						True					
		0.5			0.3			0.5			0.3		
		4	16	32	4	16	32	4	16	32	4	16	32
batchSize	nb conv												
64	1				88,2%	90,8%					86,8%	86,7%	
	2				54,3%	54,3%					86,9%	90,5%	
	4				54,3%	54,3%					82,8%	89,5%	
	7	54,4%	54,4%	54,4%	54,3%	54,3%	54,3%	70,9%	88,9%	89,5%		86,1%	86,7%
	10	54,3%	54,3%	54,3%	54,3%	54,4%	54,4%	63,1%	87,5%	89,6%	82,2%	91,3%	91,5%
	20	54,3%	54,4%	54,3%	54,3%	54,4%	54,3%	54,3%	61,2%	65,9%	54,4%	61,1%	58,5%
256	1				89,5%	91,0%					89,3%	89,4%	
	2				54,3%	91,2%					88,4%	91,7%	
	4				54,4%	54,3%					82,6%	91,5%	
	7	[illegible]	[illegible]	,3%	54,4%	54,3%	54,4%	79,2%	89,1%	91,8%	88,2%	92,0%	91,6%
	10	54,3%	54,4%	54,4%	54,4%	54,3%	54,4%	75,6%	90,1%	91,6%	85,5%	91,8%	91,8%
	20	54,4%	54,4%	ko	54,3%	54,4%	ko	54,3%	88,8%	ko	79,7%	91,7%	ko

Inference

See Main_inference.py

Set an Image as a script parameter

```
import sys
image_name = sys.argv[1]
```

Show image

```
from PIL import Image
toshow = np.reshape(ima,(48,48))
im2 = Image.fromarray(toshow)
im2.show()
```

Do forward pass

```
simple_cnn = ConvNeuralNet()
label = simple_cnn(ima)
```

Build an adversarial image

See Layers.py and Main.py

Define a « Noise » variable

```
noise = tf.Variable(tf.constant(0.0, shape=[1,2304]))
```

Prepare and load network

```
optimizer = tf.optimizers.Adam(1e-3)
simple_cnn = ConvNeuralNet()
ckpt = tf.train.Checkpoint(step=tf.Variable(1), optimizer=optimizer,
net=simple_cnn)
ckpt.restore('../nn4/saved_model-1')
```

Learn DX

```
y = model(X+DX)
grads = tape.gradient(loss, [DX])
optimizer.apply_gradients(zip(grads, [DX]))
```

Build an adversarial image

Get modified image

```
residual = noise.numpy()
residual50 = 50.0 * residual
mod_image = ima + residual
```

Add regularization constraints

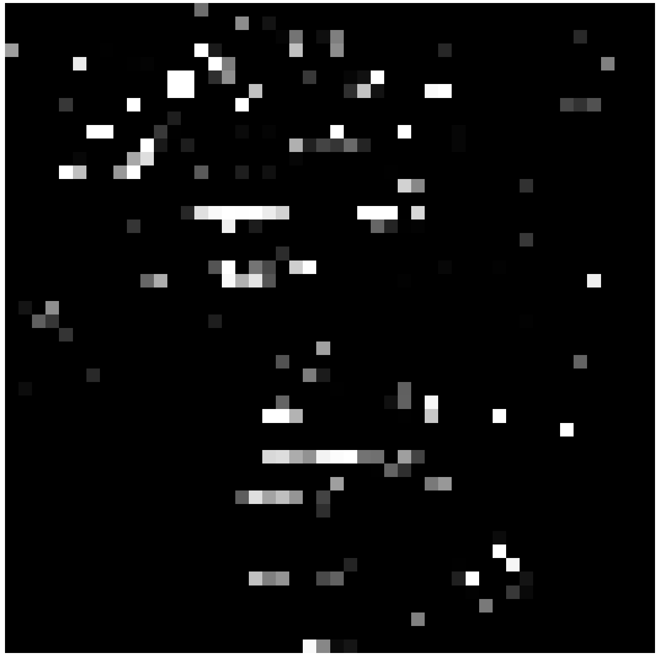
L2 = tf.reduce_mean(tf.square(DX))
L1 = tf.reduce_mean(tf.abs(DX))
Linf = tf.reduce_max(tf.abs(DX))

Learn with constraints

```
loss = loss + L2
```

Build an adversarial image

+

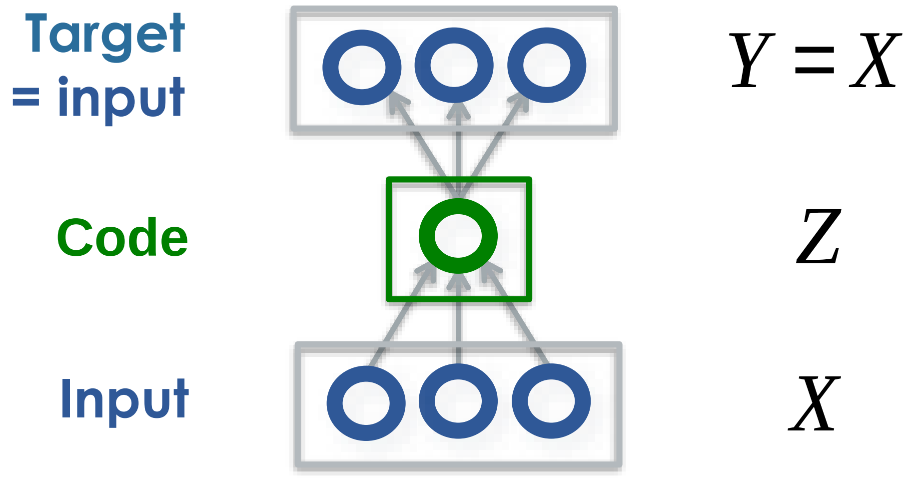


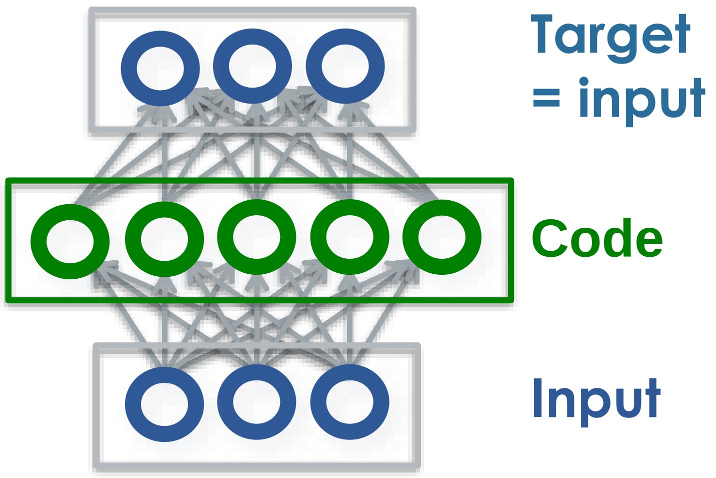
/50=

More Deep Networks

9

Auto-encoder





"Bottleneck" code
i.e., low-dimensional,
typically dense,
distributed
representation

"Overcomplete" code
i.e., high-dimensional,
always sparse,
distributed
representation

Auto-encoder

Usually <784

28 X 28 = 784

NN Encoder

code

Compact representation of the input object

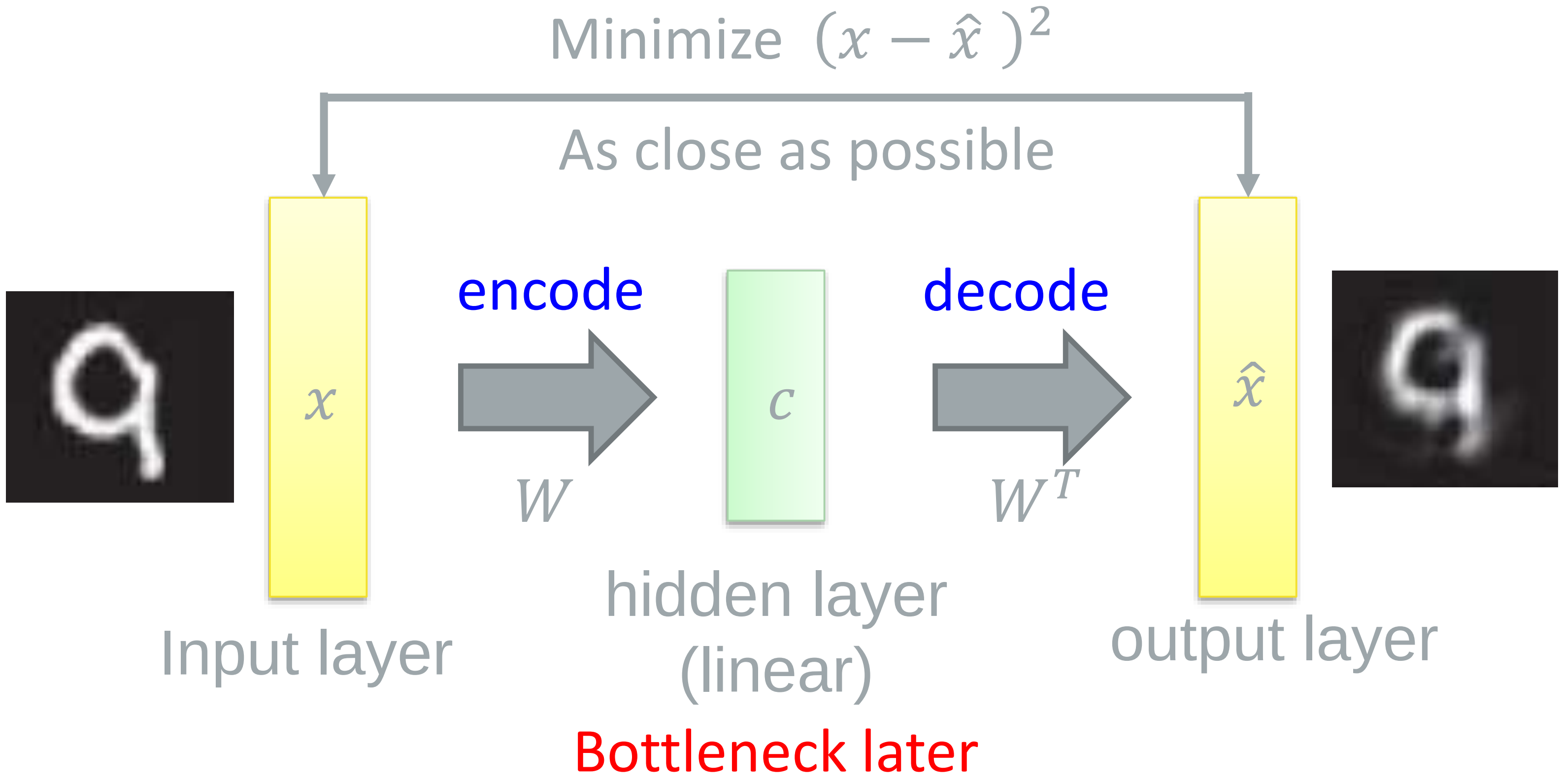
Learn together

code

NN Decoder

Can reconstruct the original object

Recap: PCA

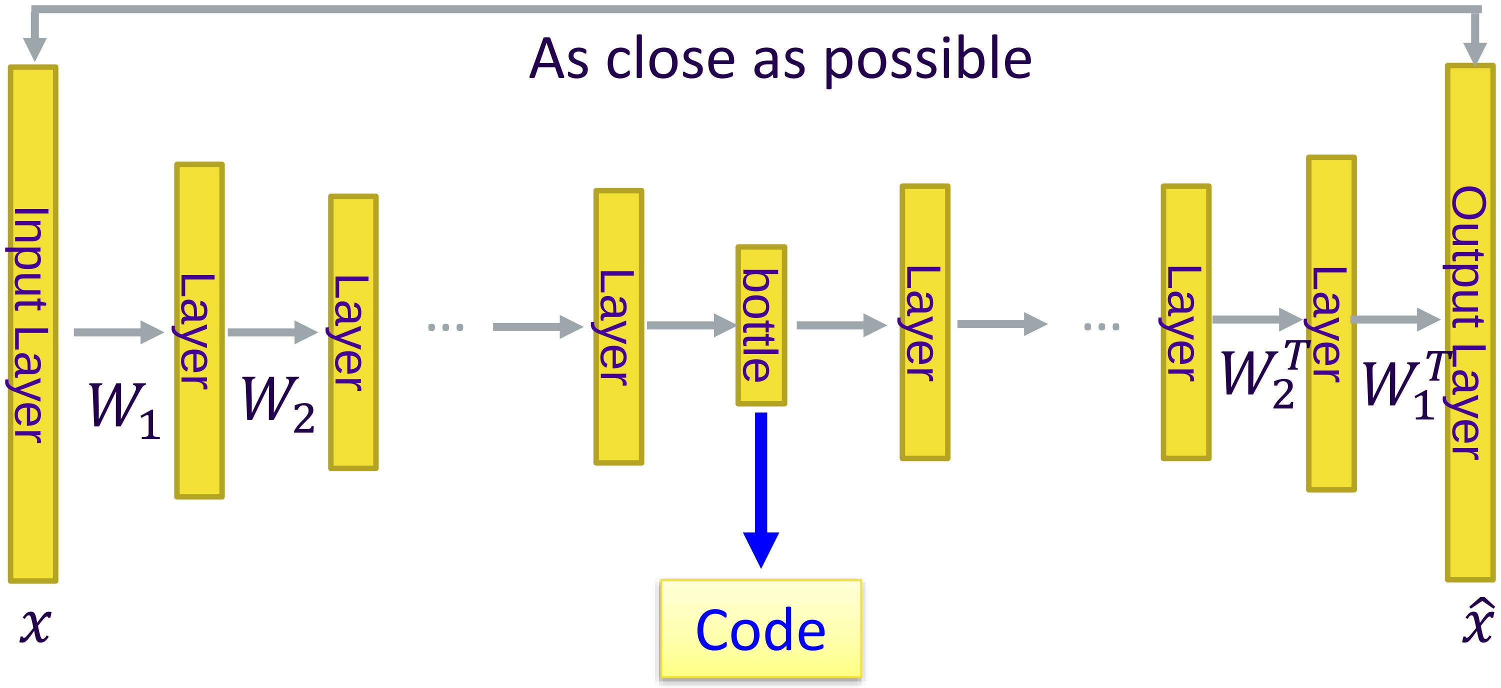


Output of the hidden layer is the code

Deep Auto-encoder

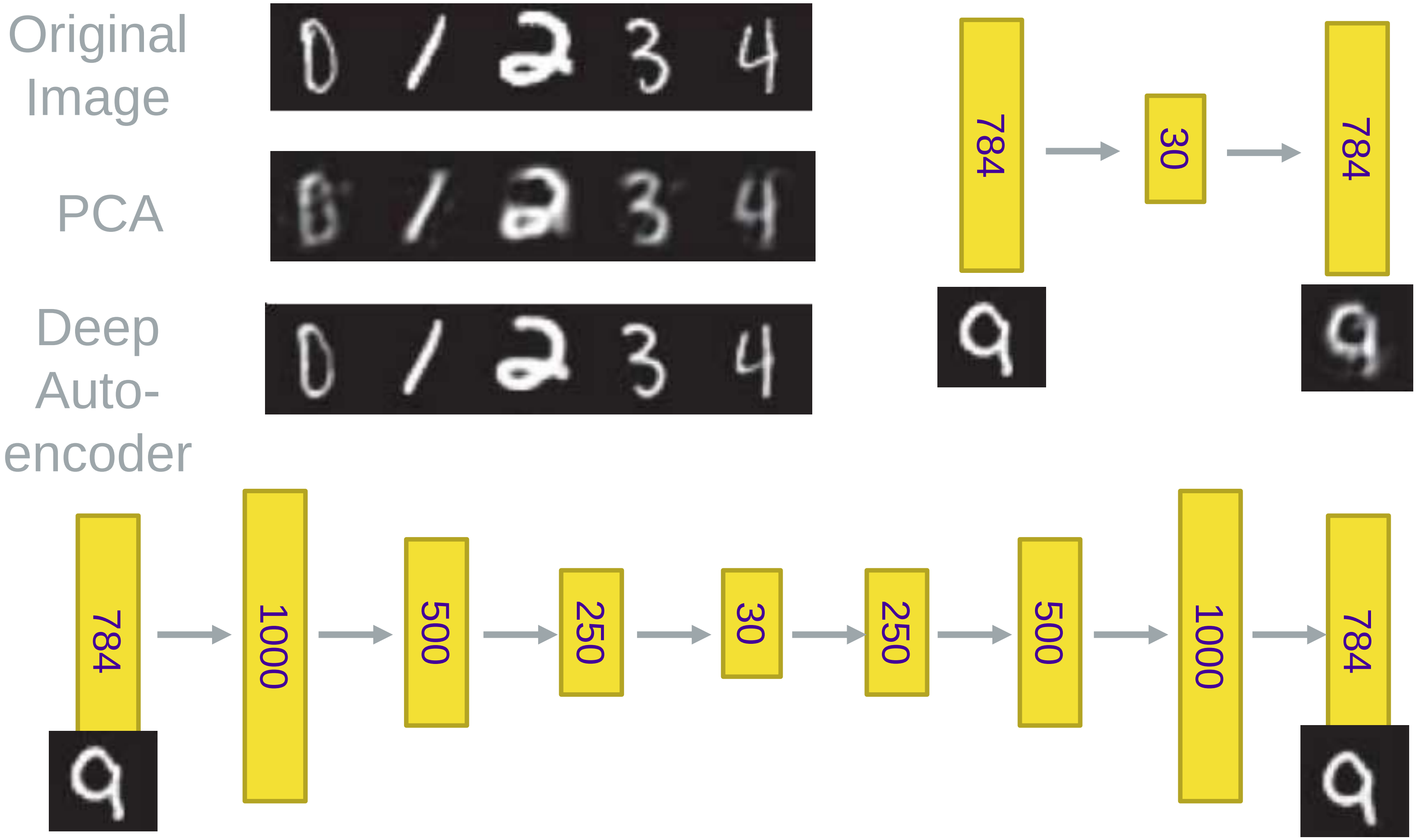
Of course, the auto-encoder can be deep

Symmetric is not necessary.

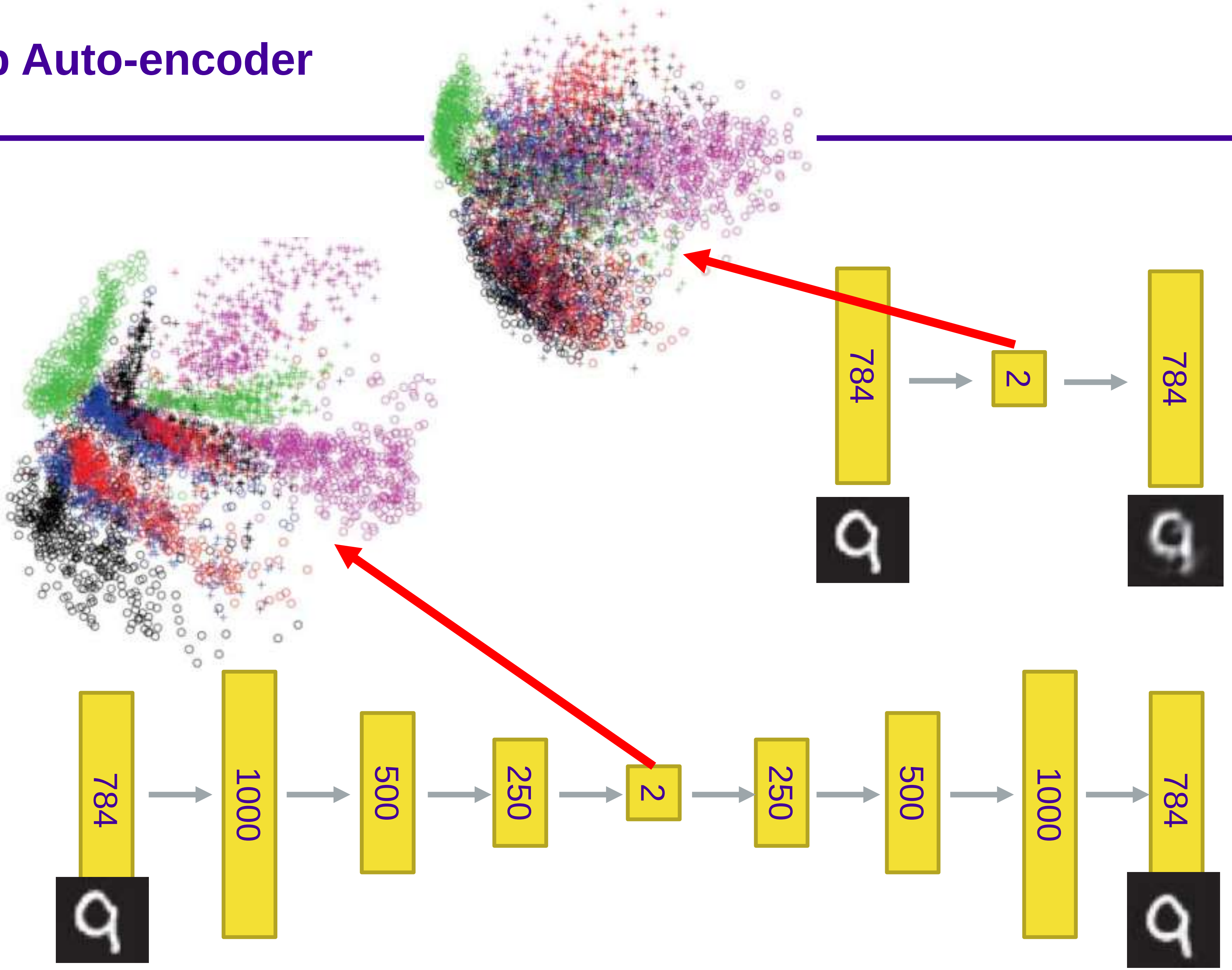


Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

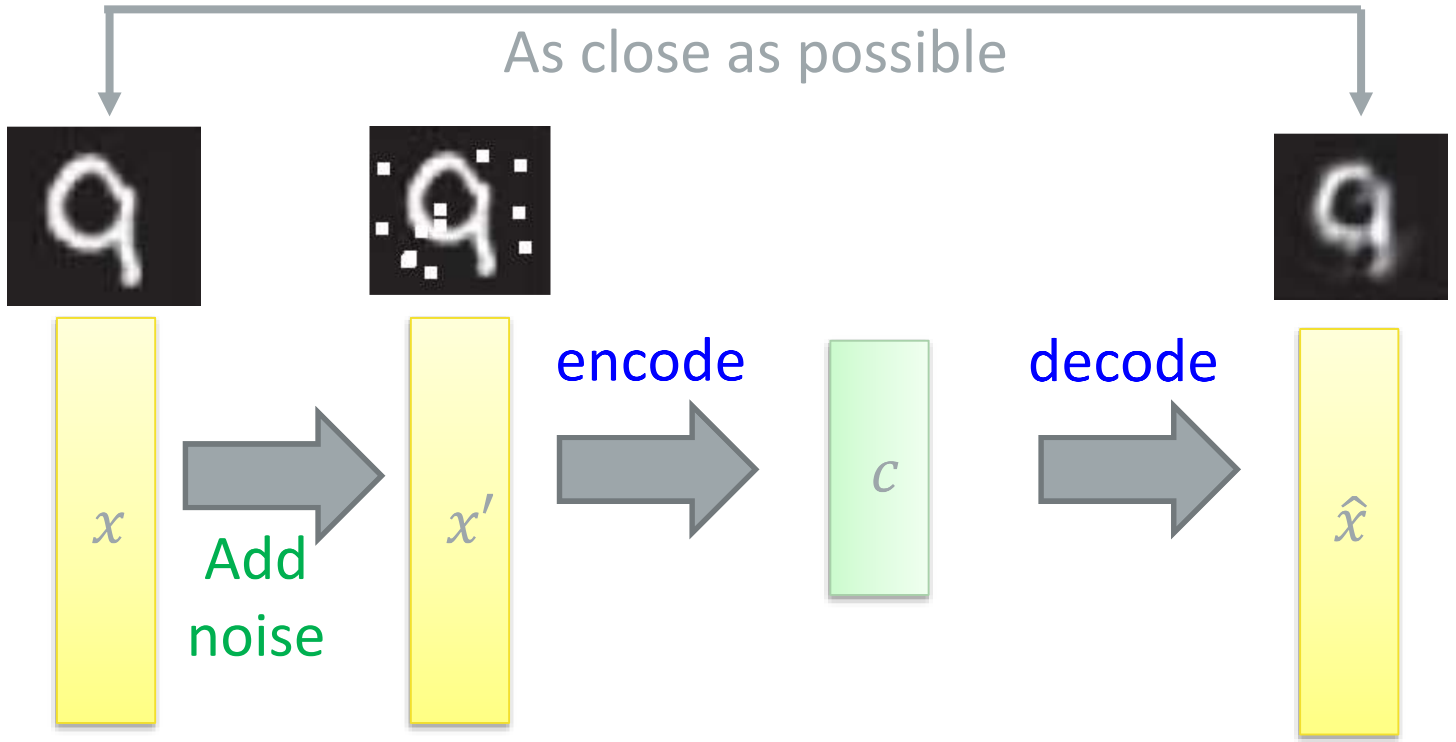
Deep Auto-encoder



Deep Auto-encoder



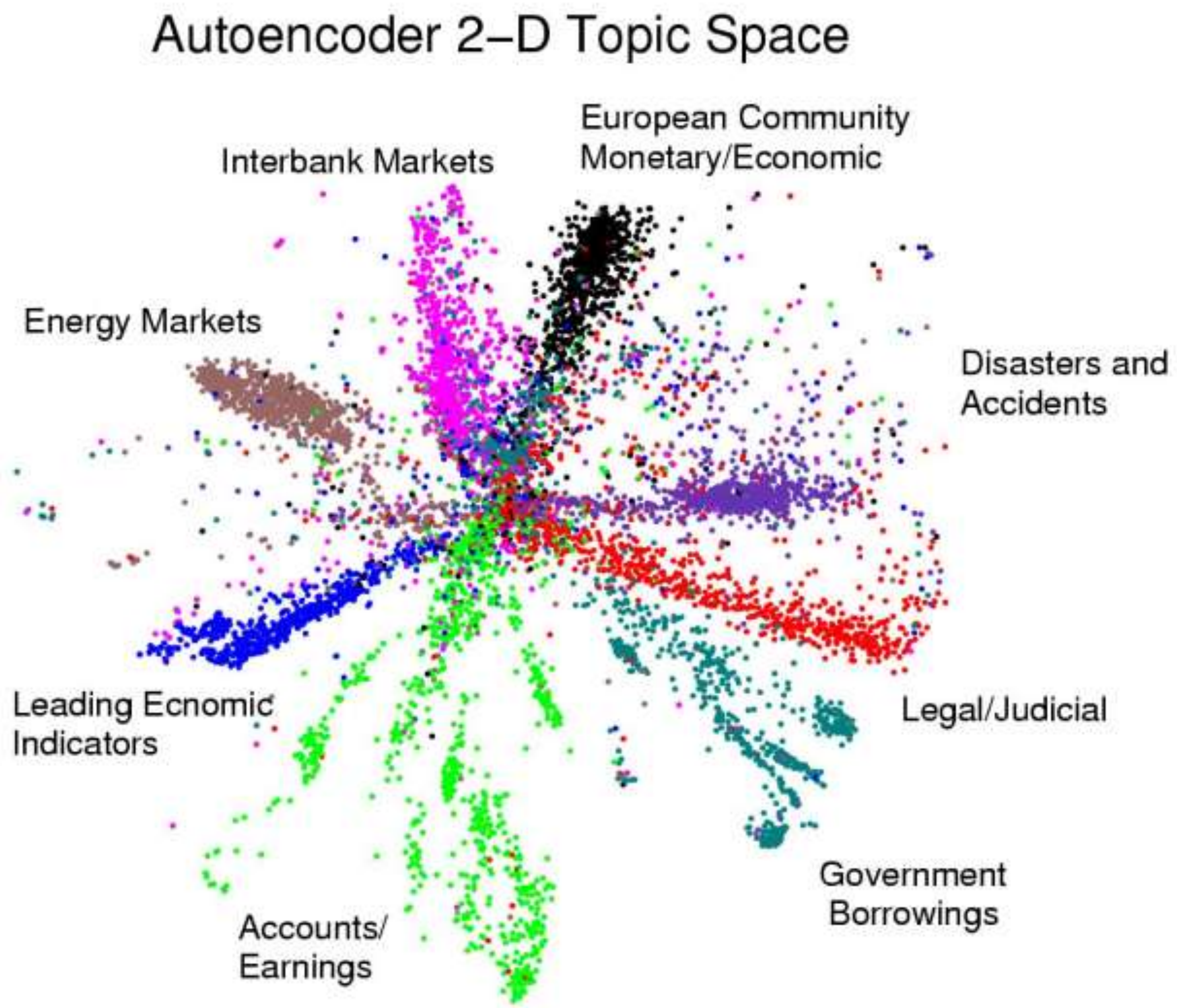
De-noising auto-encoder

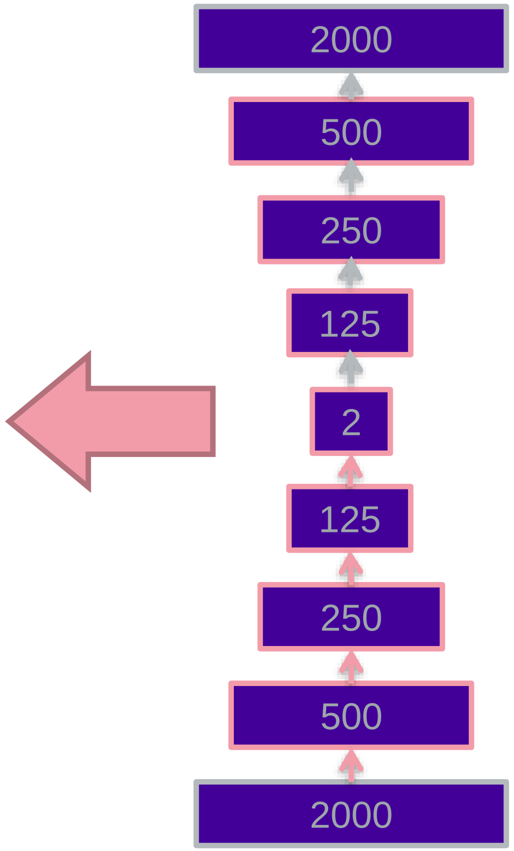


Ref: Rifai, Salah, et al. "Contractive auto-encoders: Explicit invariance during feature extraction.“ *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML,* 2008.

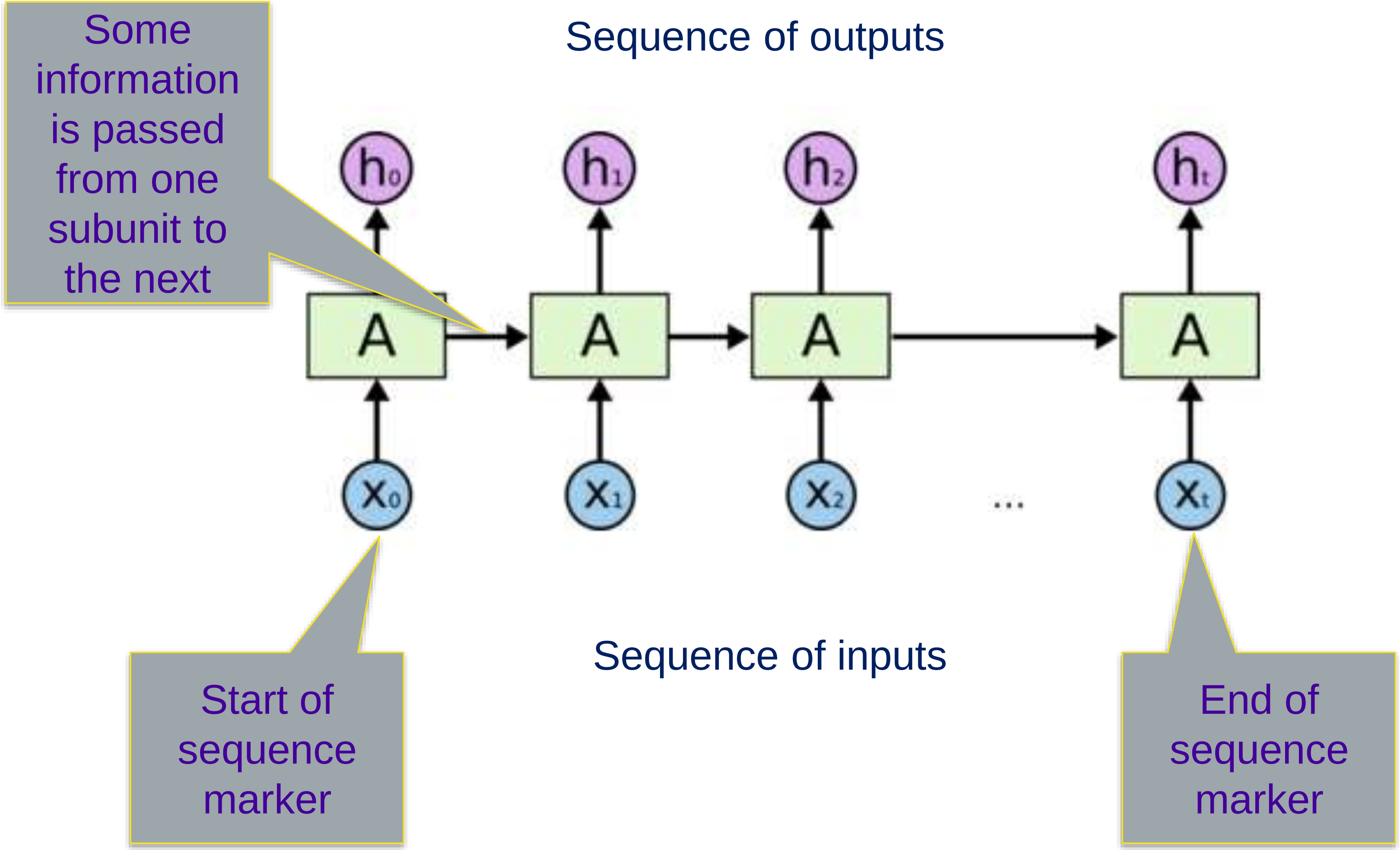
Deep Autoencoders





[Hinton & Salakhutdinov, “Reducing the dimensionality of data with neural networks, *Science*, 2006;
Salakhutdinov & Hinton, “Semantic Hashing”, *Int J Approx Reason*, 2007]

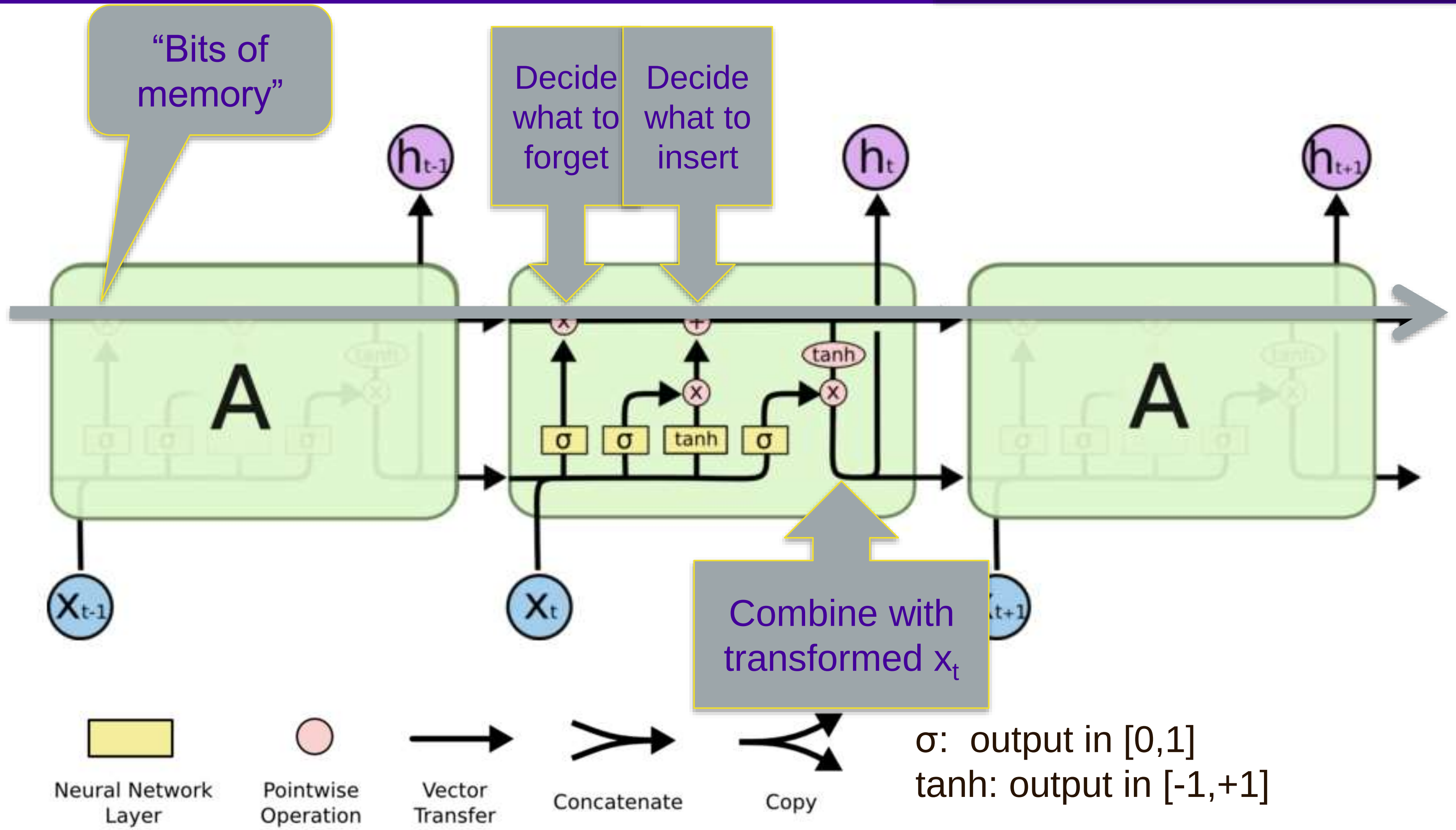
Architecture for an RNN



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Architecture for an LSTM

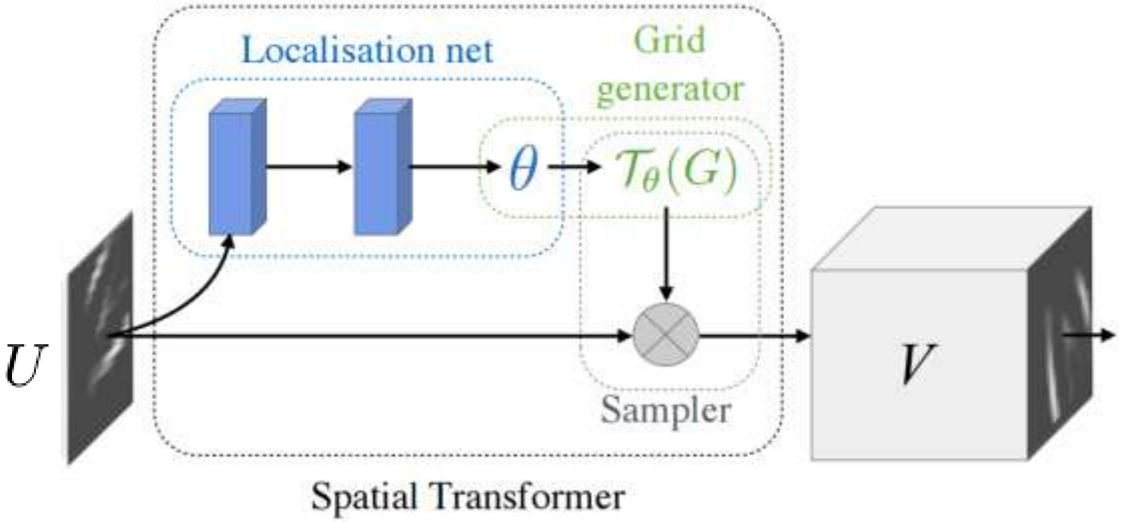
Longterm-short term model

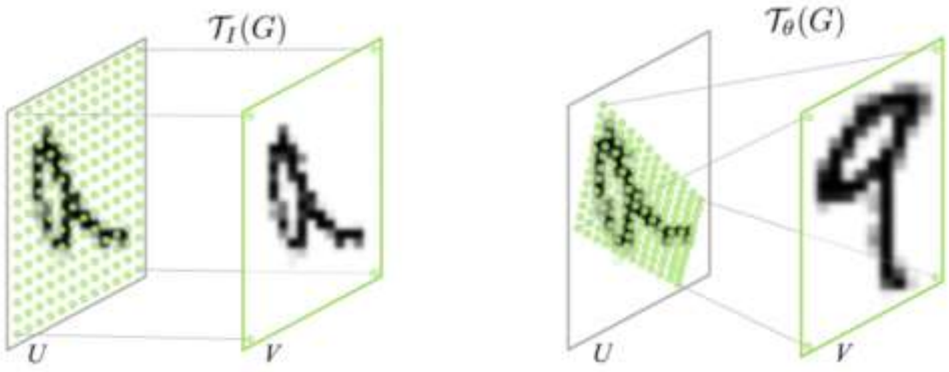


σ: output in [0,1]
tanh: output in [-1,+1]

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Spatial Transformer Network





- A 1st CNN (localisation net) uses the input image to compute the parameters of a geometric transform (similarity, affine, TPS, etc.)
- This transform is applied to the image which is then passed to the task specific network such as a face encoder, a FP detector, etc.
- Both networks are simultaneously trained to minimize a task specific objective function

YOLO : object detection

.02640v5 [cs.CV] 9 May 2016

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

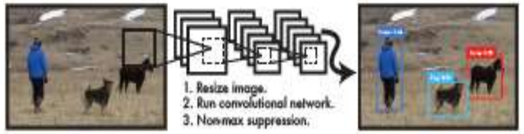


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only

2.08242v1 [cs.CV] 25 Dec 2016

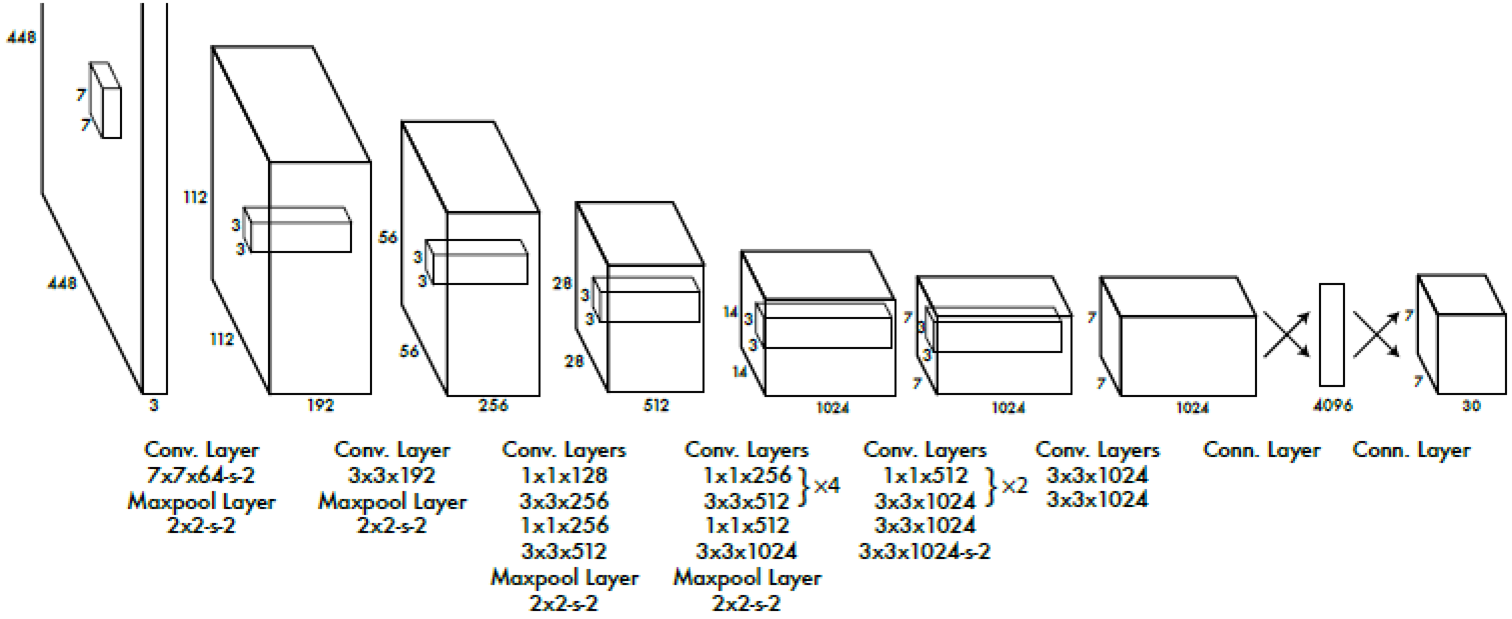
YOLO9000: Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.

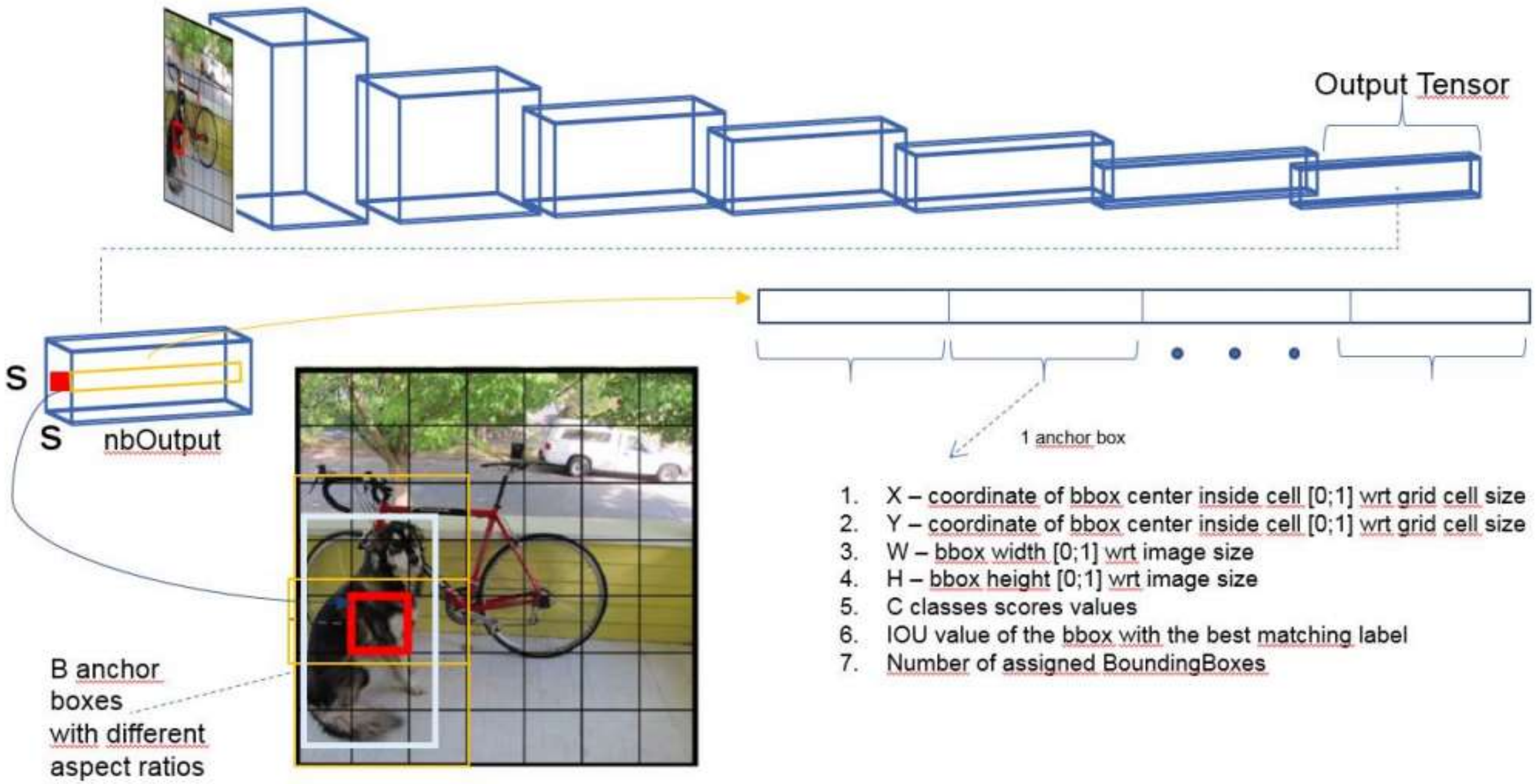
Regression and Classification



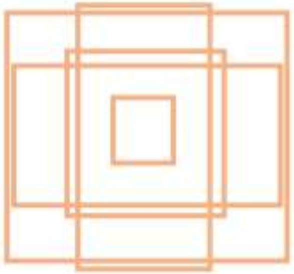
- ✓A neural network predicts bounding boxes and class probabilities directly from full images in one evaluation
- ✓Only convolutional layers
- ✓Predefined boxes named anchors
- ✓Relative object positions in anchor boxes are learned

Object Detection in Video

YOLO: encoding

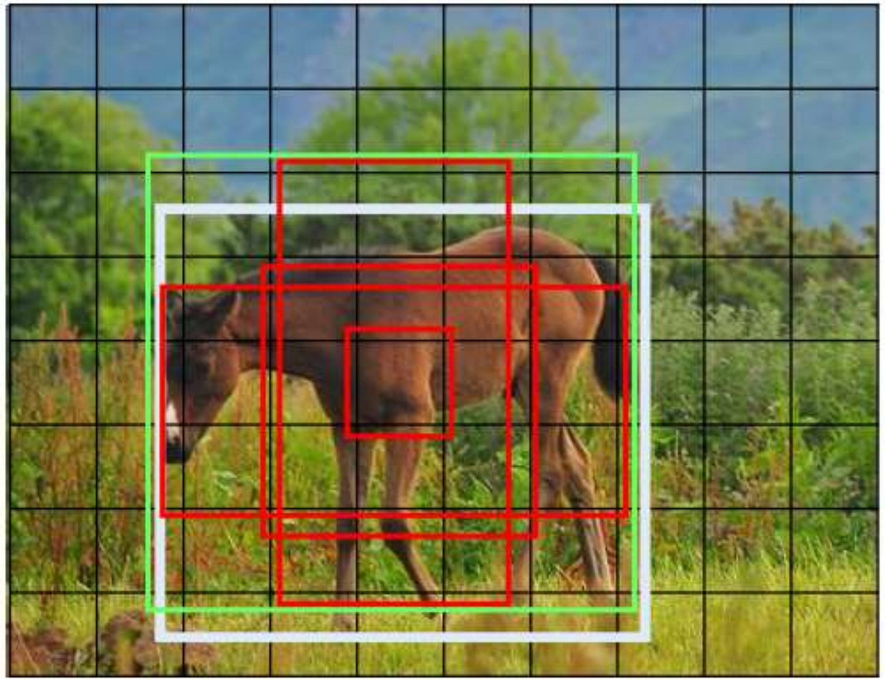


YOLO: labels to anchor association



- For 1 label : Compute matching score with all the anchor boxes
- For each anchor box encode it on the best matching label

NB: 1 label can be encoded on multiple anchor boxes: record the ‘number of assignment’



Convolution padding

tf.nn.conv2d(input, filter, strides,   padding)

• **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use

When stride is 1 (more typical with convolution than pooling), we can think of the following distinction:

•"SAME": output size is the same than input size. This requires the filter window to slip outside input map, hence the need to pad.
•"VALID": Filter window stays at valid position inside input map, so output size shrinks by filter_size - 1. No padding occurs.

With "VALID" padding and appropriate filter size, Fully Connected layers can be implemented as 1x1 convolutions.

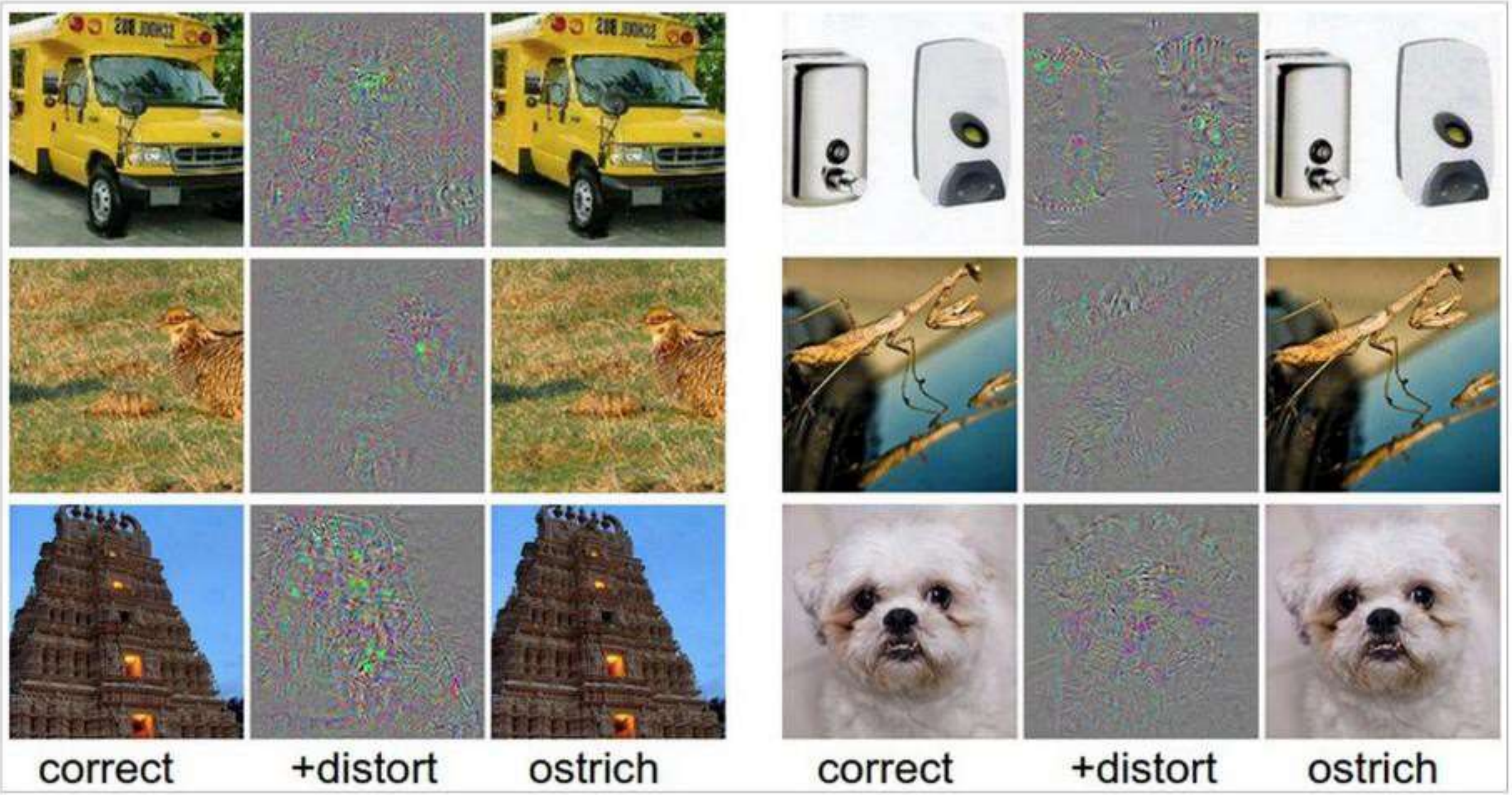
Then, learned Conv networks can be applied of images of any sizes.


YOLO v2
http://pjreddie.com/yolo

ADVERSARIES

10

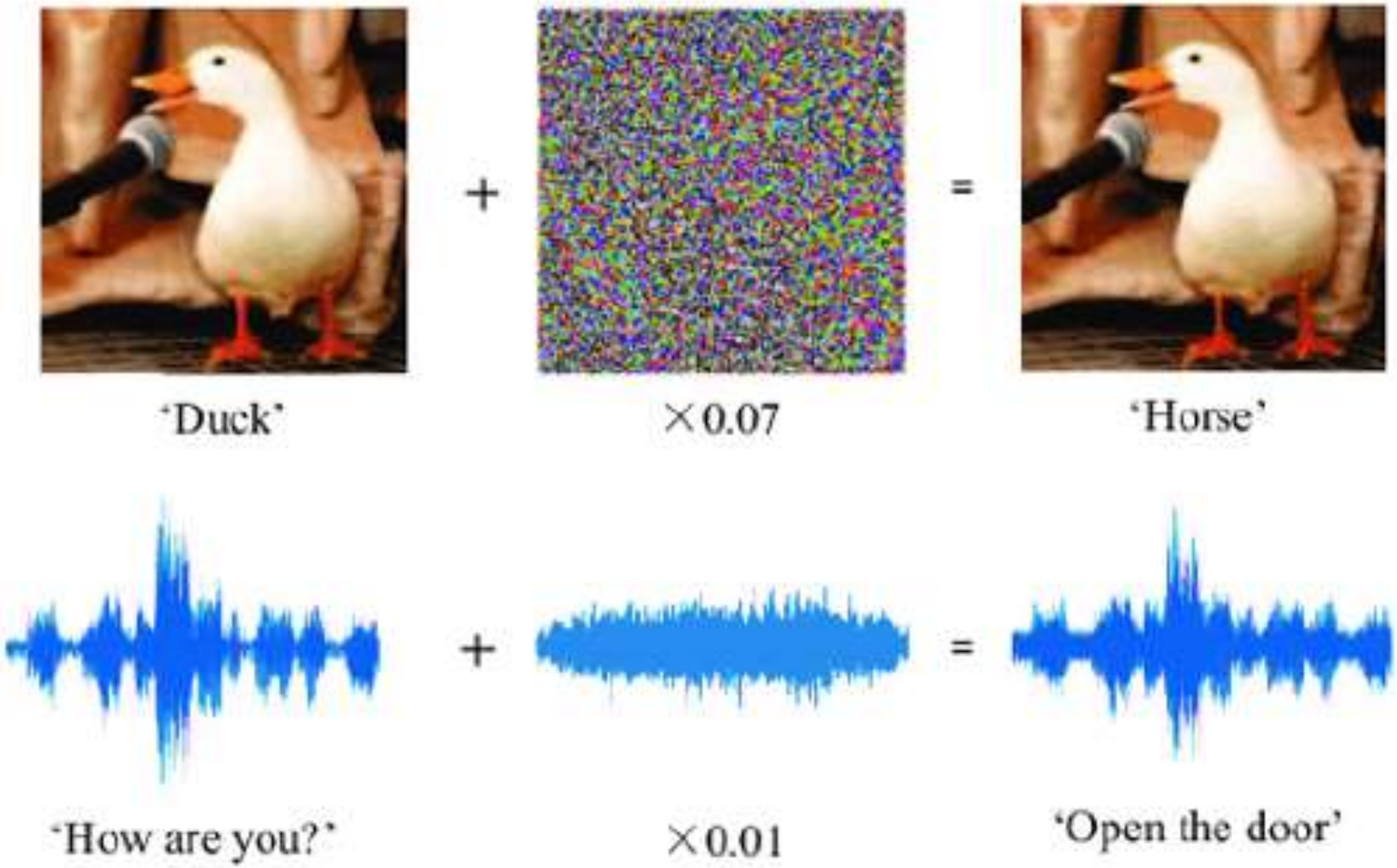
Adversarial Examples



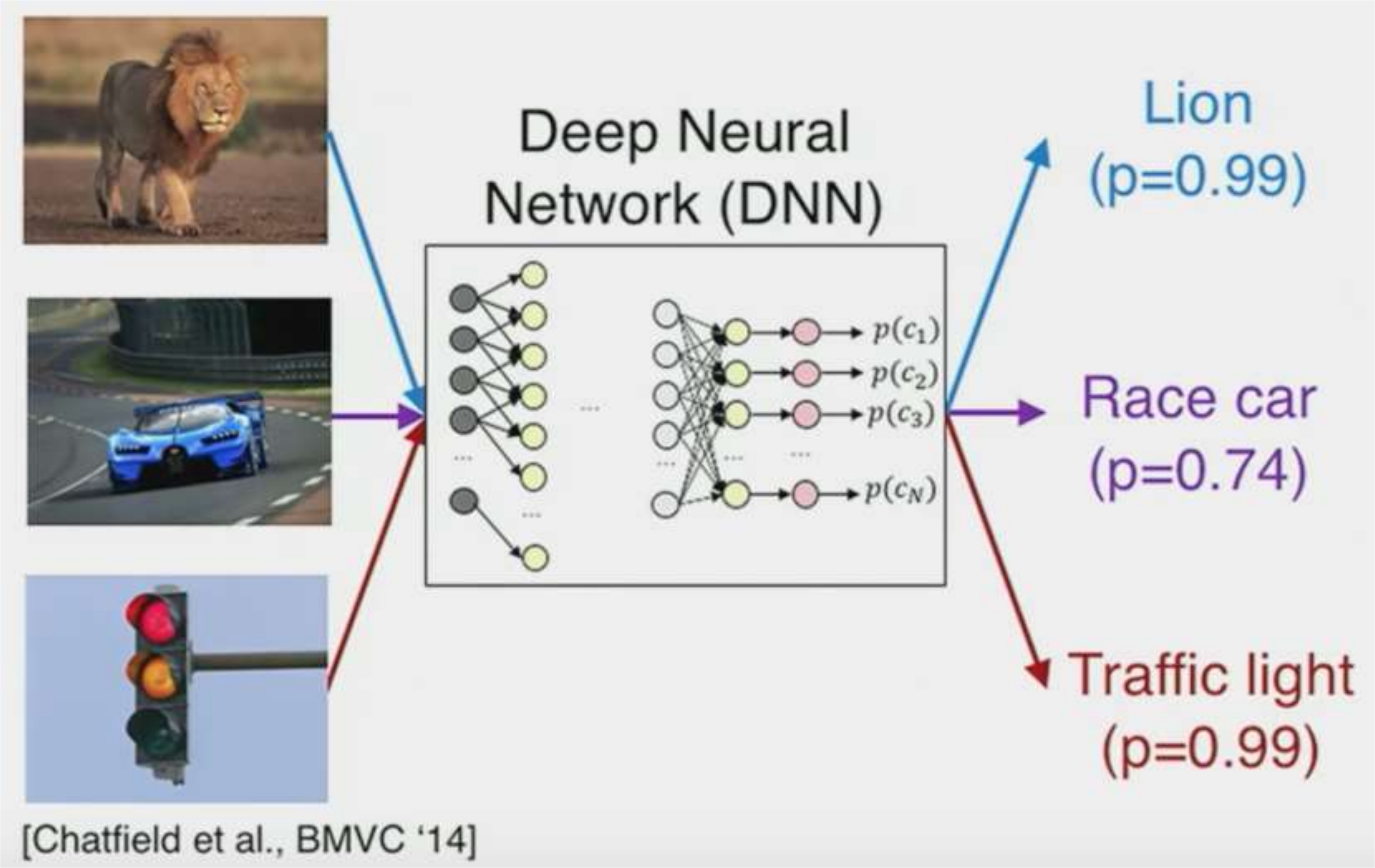
Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [Szegedy ICLR 2014]

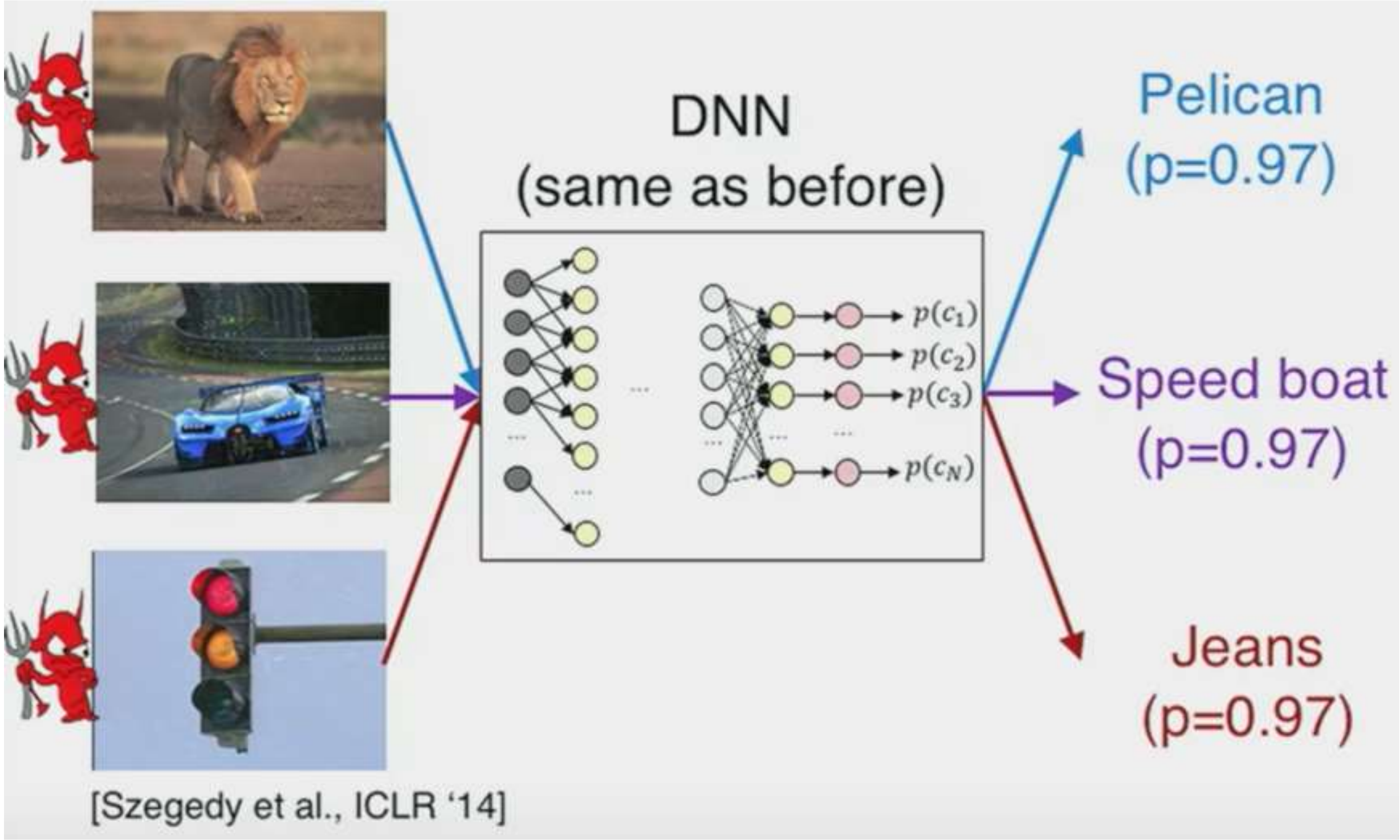
Adversarial Examples



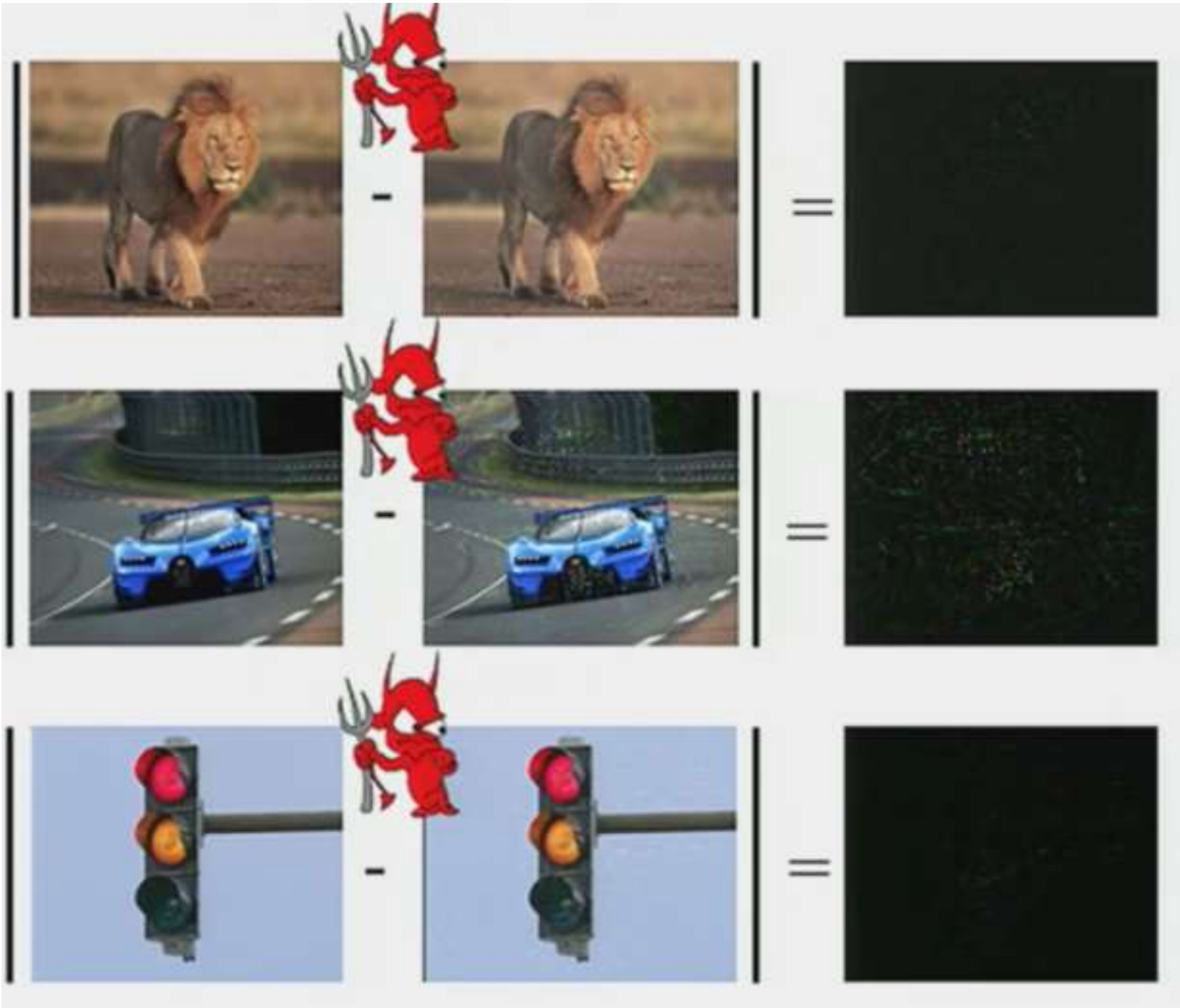
What Do You See



What Do You See Now



What Do You See Now



Adversarial Examples

A real weakness

Stop Sign

Yield Sign

Attack

Evasion vs. poisoning

- **Key issue in AML: bad actors (who do bad things) have *objectives***

  the main one is not getting detected

  they can change their behavior to avoid detection

- **This gives rise to *evasion attacks***

  Attacks on ML, where malicious objects are deliberately transformed to evade detection (prediction by ML that these are malicious)

Evasion vs. poisoning

- **An entirely different class of attacks are *data poisoning attacks***

- **In these, an adversary introduces malicious modifications to the data used for training**
    - Can insert instances (for example, send specially crafted emails, either benign or malicious)
    - Can modify instances in the data (hack one of the servers used to store a part of the data)
    - Can selectively remove some instances

Evasion Attack (Most Common)

- The most common attack. It can be further classified into

- White-Box: Attackers know full knowledge about the ML algorithm, ML model, (i.e., parameters and hyperparameters), architecture, etc.

- Black-Box: Attackers almost know nothing about the ML system (perhaps know number of features, ML algorithm).

White-Box Evasion Attack

- Given a function (LogReg, SVM, DNN, etc) $\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}$, where X is a input feature vector, and Y is an output vector.
- An attacker expects to construct an adversarial sample X* from X by adding a perturbation vector $\delta_{\mathbf{X}}$ such that

$$\arg\min_{\delta_{\mathbf{X}}} \|\delta_{\mathbf{X}}\| \ \textbf{s.t.}\ \mathbf{F}(\mathbf{X} + \delta_{\mathbf{X}}) = \mathbf{Y}^*$$

- where $\mathbf{X}^* = \mathbf{X} + \delta_{\mathbf{X}}$ and Y* is the desired adversarial output.
- Solving this problem is non-trivial, when F is nonlinear or/and nonconvex.

White-Box Evasion Attack

- **Approximate Solution: Jacobian-based Data Augmentation**

  Direction Sensitivity Estimation: Evaluate the sensitivity of model F at the input point corresponding to sample X

$$\nabla \mathbf{F}(\mathbf{X}) = \frac{\partial \mathbf{F}(\mathbf{X})}{\partial \mathbf{X}} = \left[\frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial x_i}\right]_{i \in 1..M, j \in 1..N}$$

  Perturbation Selection: Select perturbation affecting sample X's classification

- **Other Solutions**

  Fast sign gradient method

  DeepFool

White-Box Evasion Attack

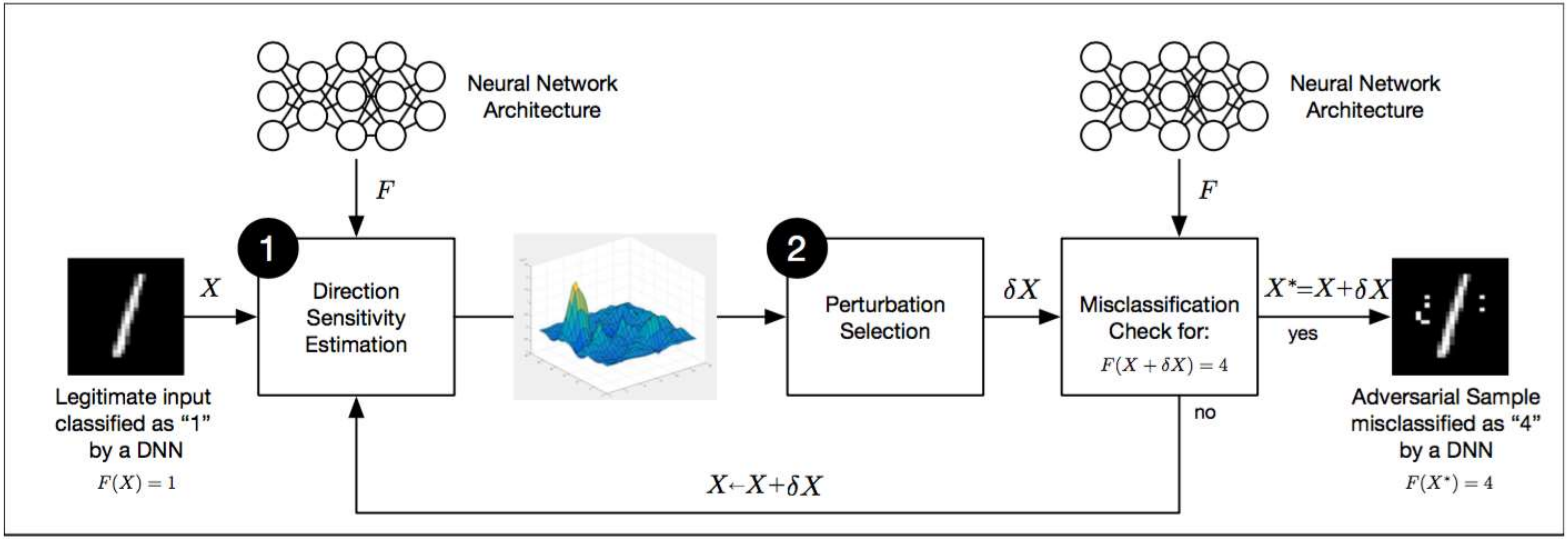


Fig. 3: **Adversarial crafting framework:** Existing algorithms for adversarial sample crafting [7], [9] are a succession of two steps: (1) *direction sensitivity estimation* and (2) *perturbation selection*. Step (1) evaluates the sensitivity of model F at the input point corresponding to sample X. Step (2) uses this knowledge to select a perturbation affecting sample X's classification. If the resulting sample $X + \delta X$ is misclassified by model F in the adversarial target class (here 4) instead of the original class (here 1), an adversarial sample X^* has been found. If not, the steps can be repeated on updated input $X \leftarrow X + \delta X$.

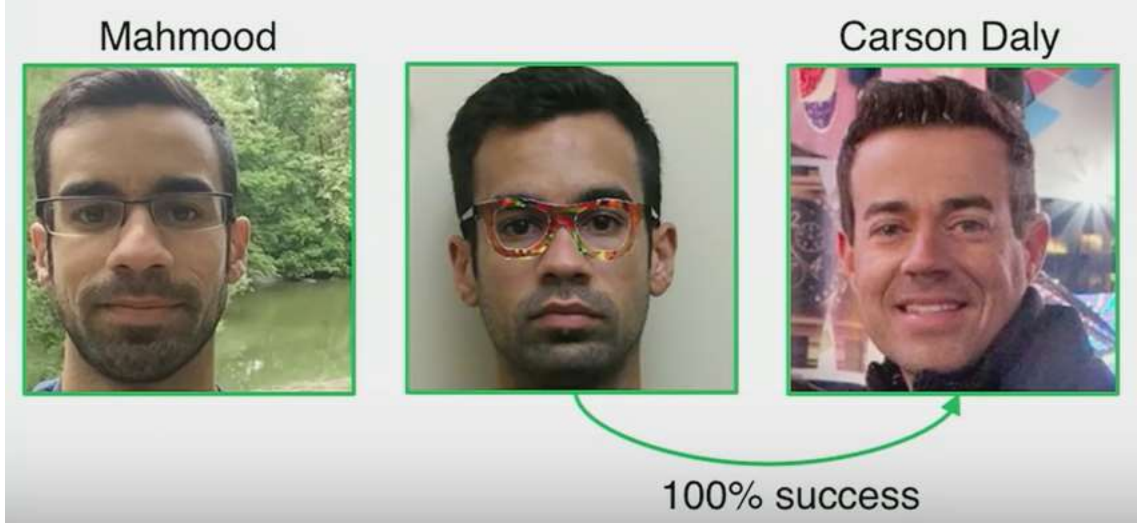
White-Box Evasion Attack

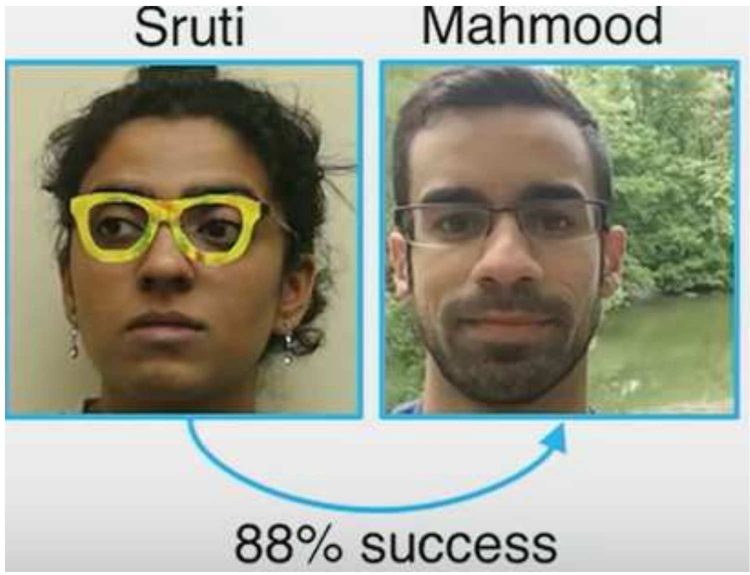
Perturbation selection :

this can small => L2 constraint

this can be sparse => L1 constraint

this can be local !!





Black-Box Evasion Attack

- **Adversarial Sample Transferability**

  Cross model transferability: The same adversarial sample is often misclassified by a variety of classifiers with different architectures

  cross training-set transferability: The same adversarial sample is often misclassified trained on different subsets of the training data.

- **Therefore, an attacker can**

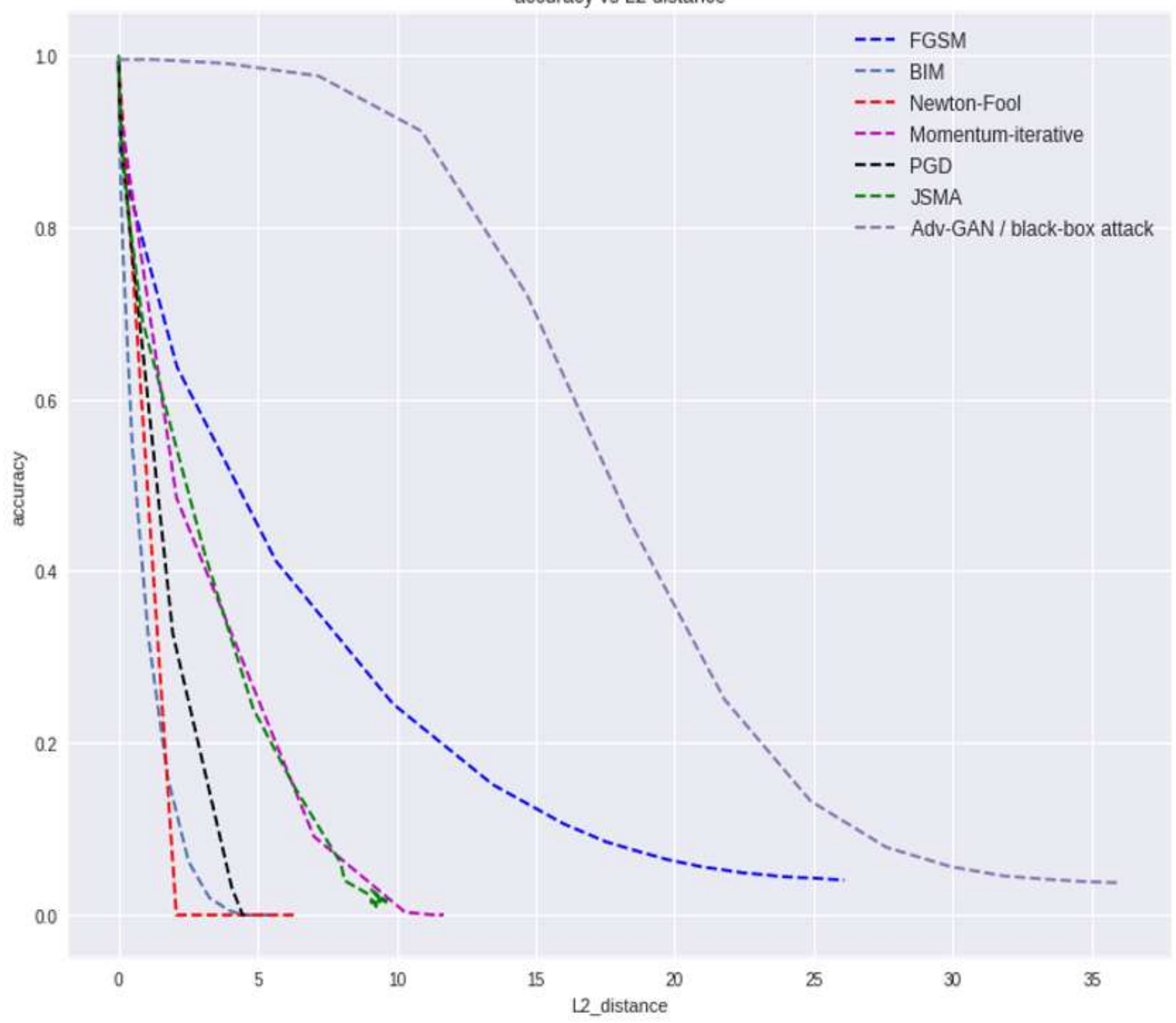
  First train his own (white-box) substitute model

  Then generate adversarial samples

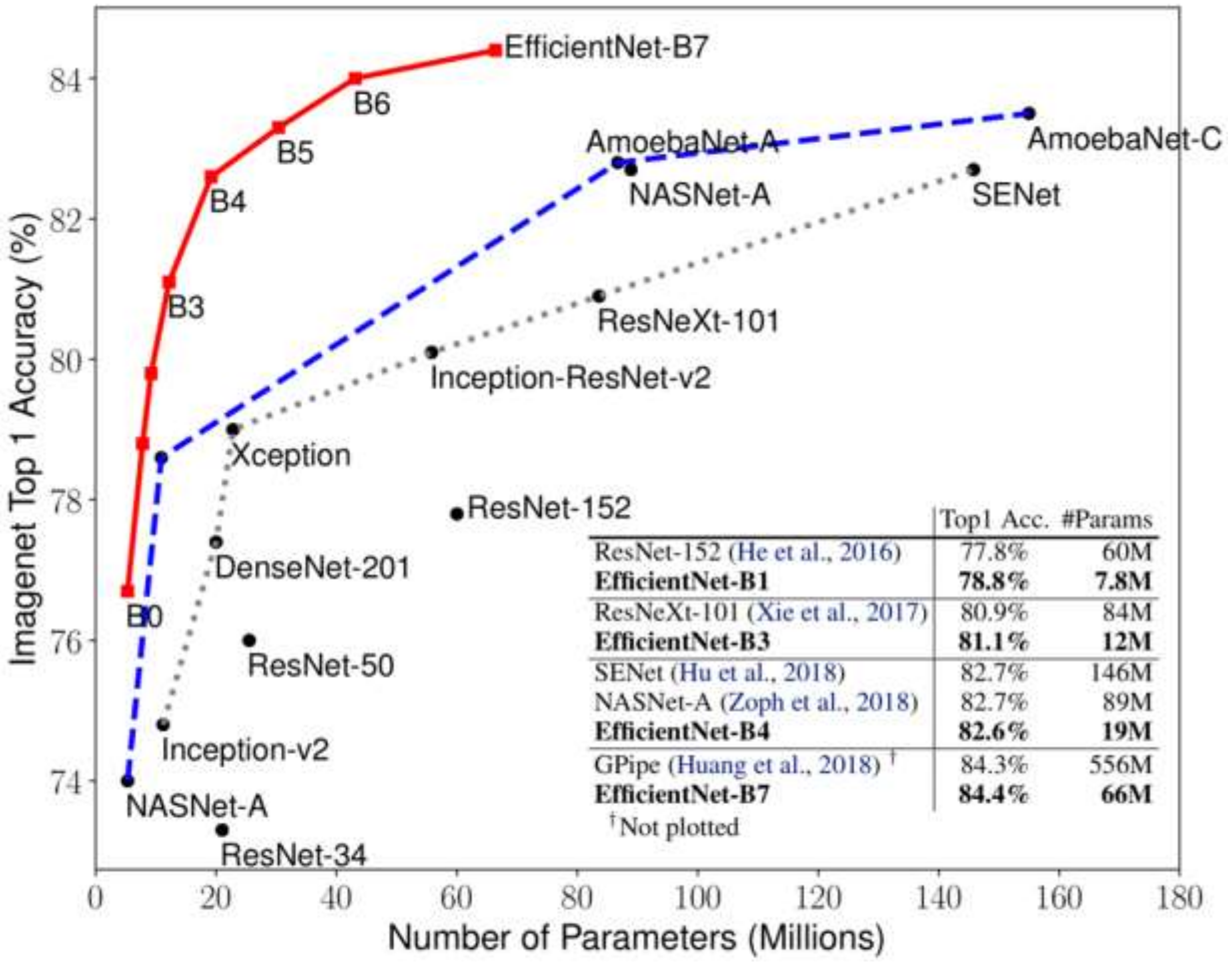
  Finally, apply the adversarial samples to the target ML model

Small Benchmark of Gradient Descent attack on MNist

Accuracy VS L2 distance



Attack on different DNN

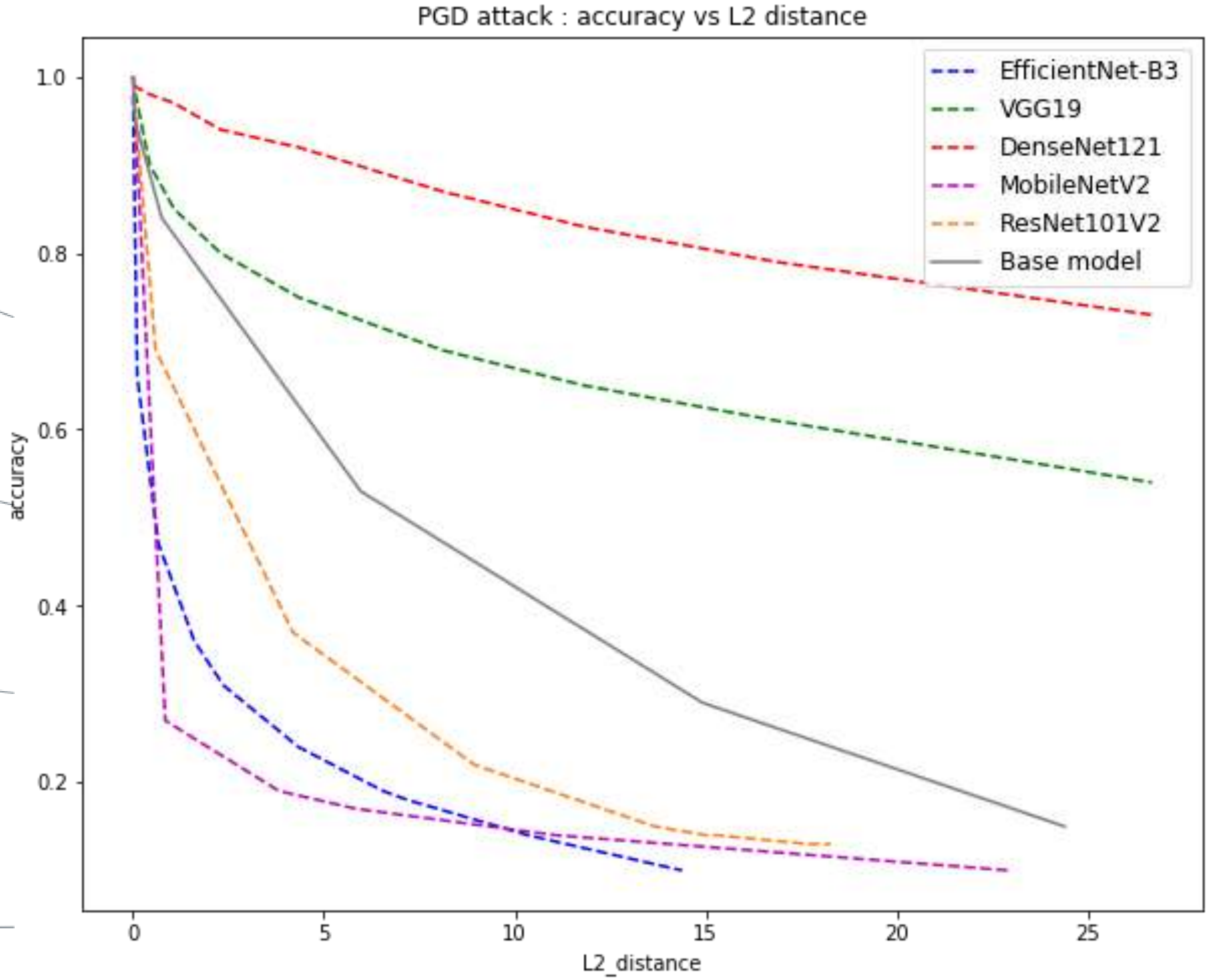


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Reference: (10 Juin 2019) *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.*
Mingxing Tan, Quoc V. Le

Attack on different DNN



EfficientNet-B3 and **MobileNetV2** have better performances but are more vulnerable.

Defense

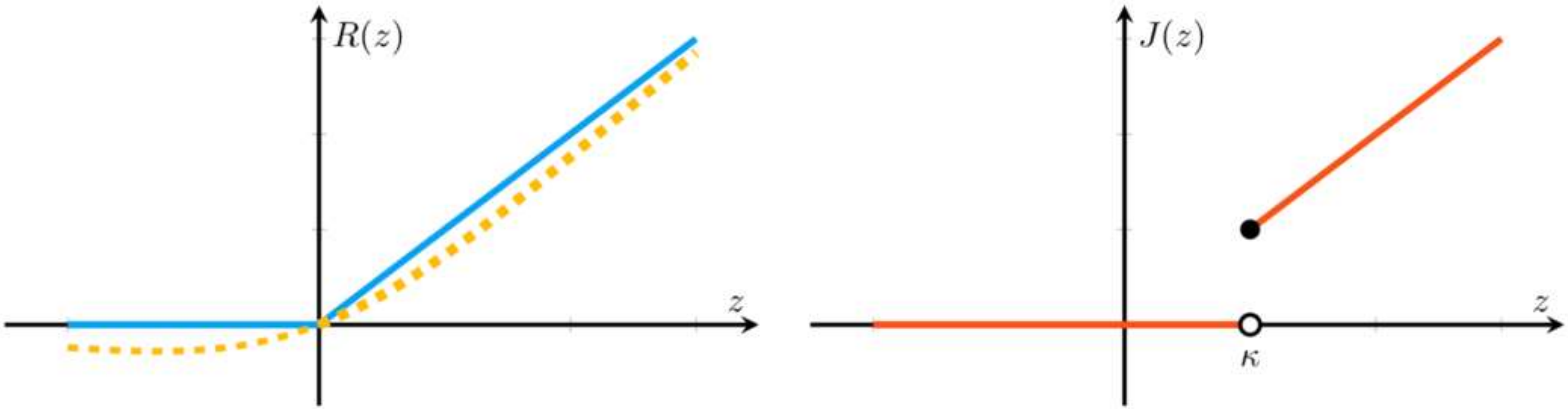
- Adversarial Training
- JumpReLU
- GAN

Defense through retraining

- **Start with original data**
- **Use *any learning algorithm* to learn a model *f***
- **For malicious instances, apply *any evasion method* to generate new instances *x'* to add to the dataset**
- **Repeat**
- **Stop when:**
    - No new instances to add
    - Iteration limit
    - Classifier changes small between successive iterations

JumpRelu

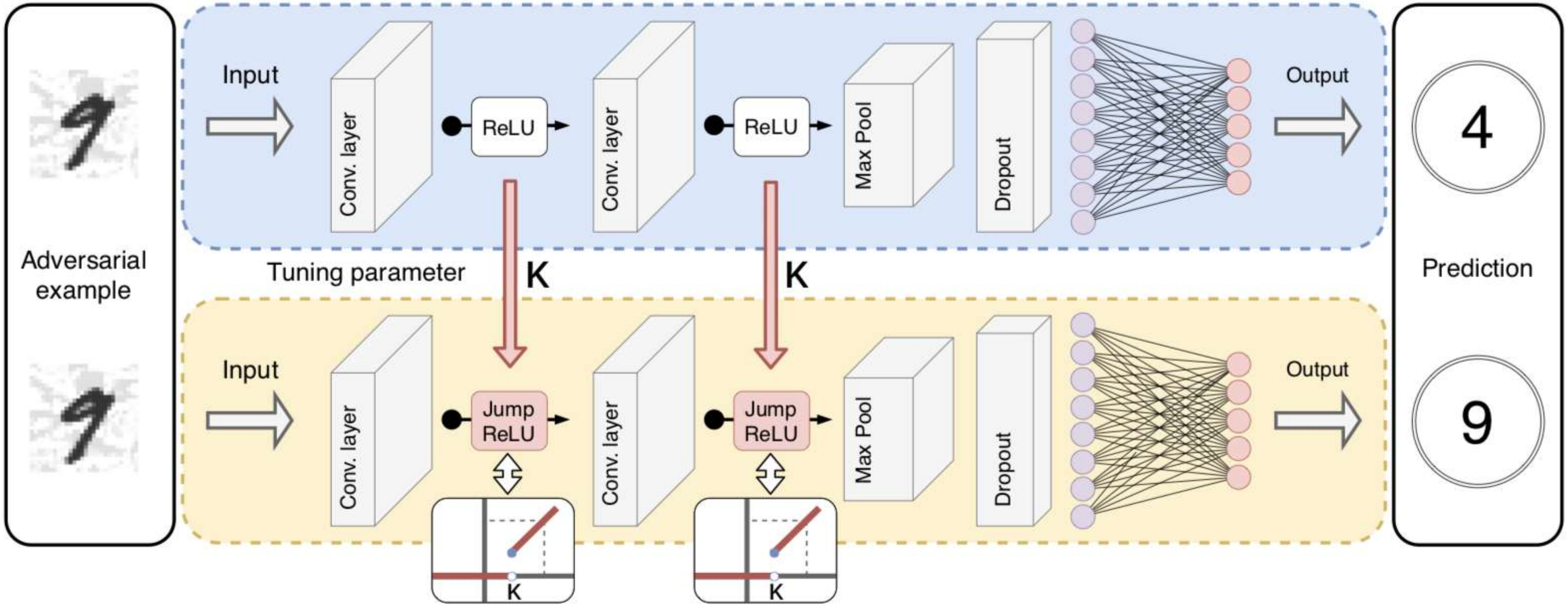
- The Jump ReLU (B) activation function provides robustness controlled by a jump value (threshold value) κ.
- JumpReLU suppresses weak positive signals.



(A) ReLU and Swish (dashed). (B) JumpReLU activation function.

Reference : (April 7th, 2019) *JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks.* N. Benjamin Erichson, Zhewei Yao, Michael W. Mahoney

JumpRelu



Reference : (April 7th, 2019) *JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks*. N. Benjamin Erichson, Zhewei Yao, Michael W. Mahoney

How to defense attacks using GAN

- **G() is pre-trained and has learned the underlying distribution of the training (image) dataset after training GAN**

Synthetic image **x'=G(**z*)**)**
(**Preserve low-dimensional manifold**)

Invert and Classify

Classifier C()

Original image **x**
(Could include high-dimensional manifold when noise enters)

$$z^* = \arg\min_z \|G(z) - x\|_2$$

Adversarial Task

Context :

- **The Deep Learning Wave is build on “Labeled Data”**
- **Labels do not always describe the target task**

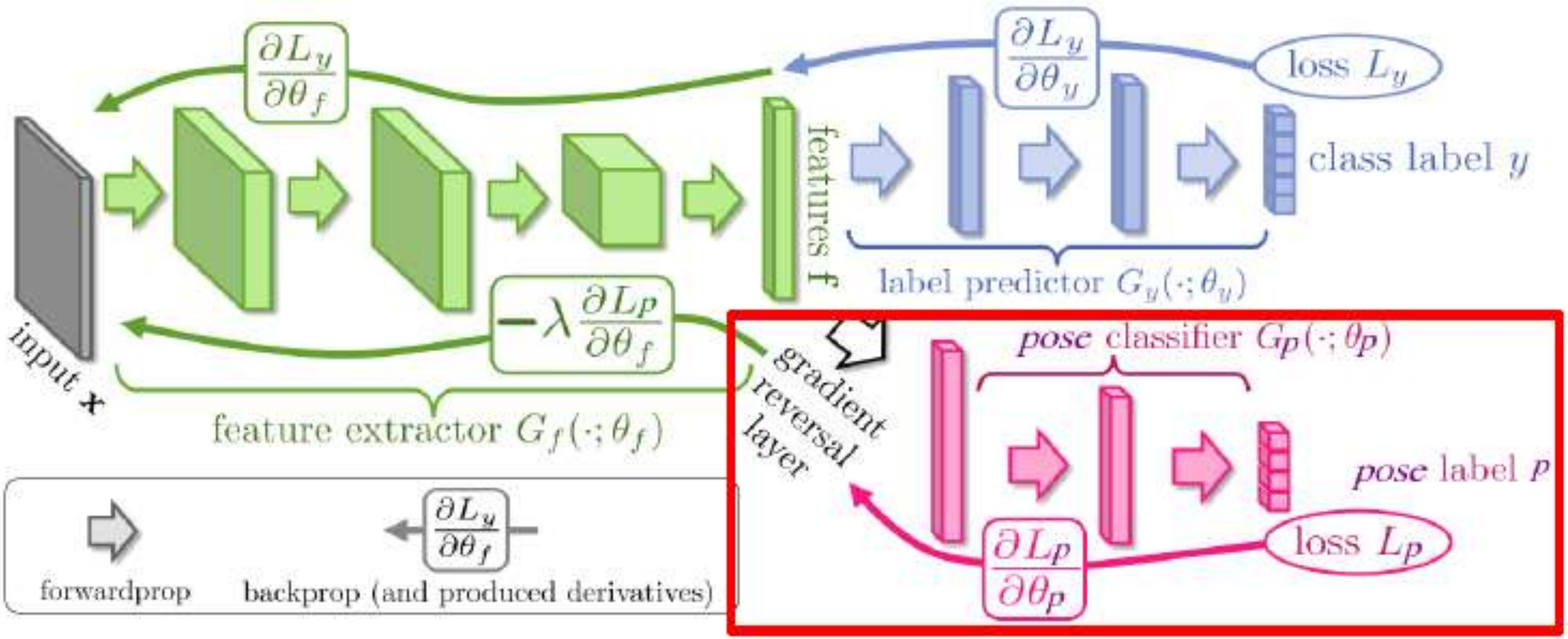
Multi-Task : Learning multiple task at the same time.

- **This can improve the performance of individual tasks when they share common information**

Adversarial-Task : Labels from undesired features can be use to improve internal representation

- **This is done with gradient reversal techniques**

Gradient Reversal



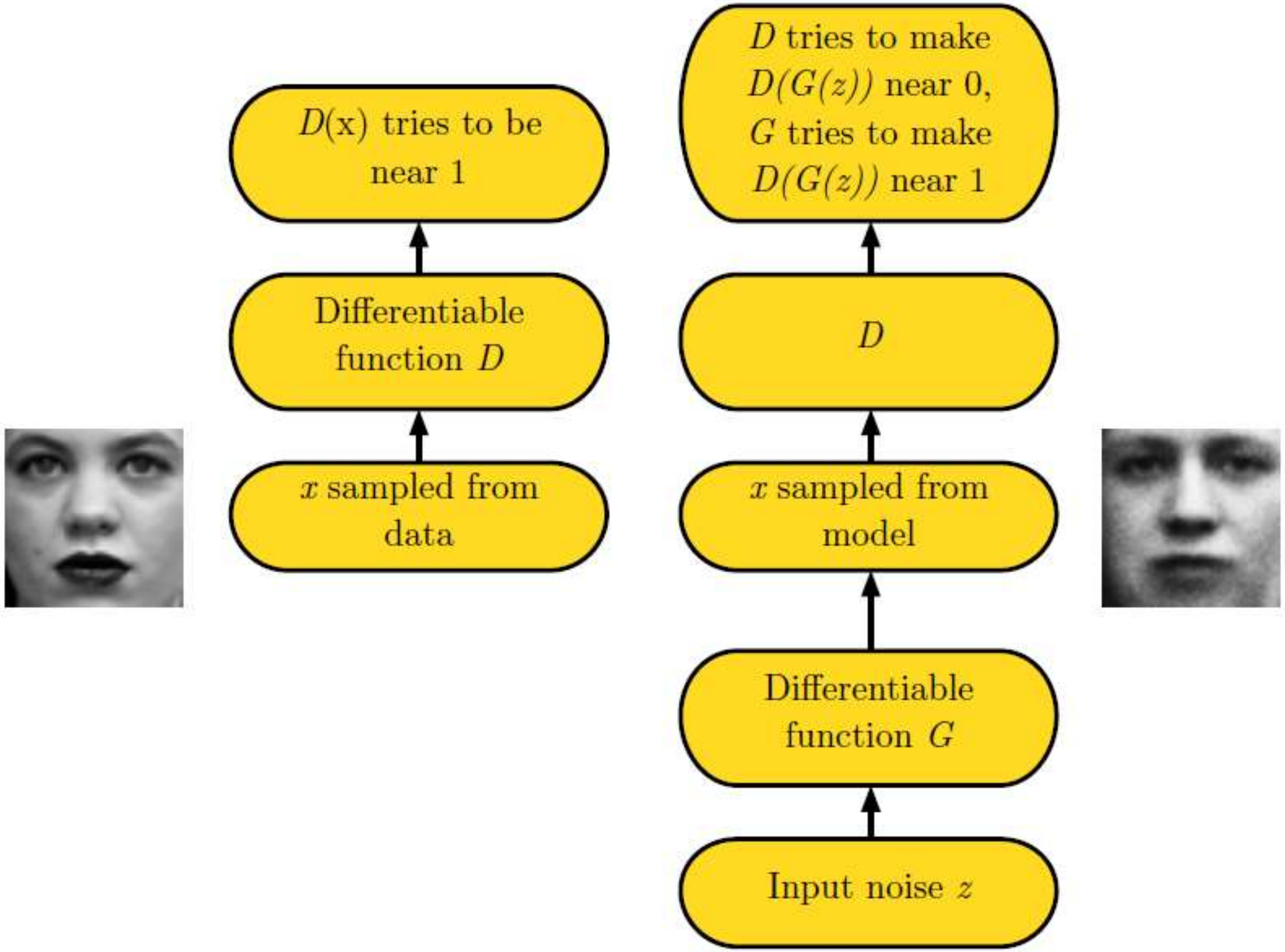
Generative Adversarial Network

- **System of two neural networks competing against each other in a zero-sum game framework.**
- **They were first introduced by Ian Goodfellow *et al.* in 2014.**
- **Can learn to draw samples from a model that is similar to data that we give them**

G, the generative model, is a multilayer perceptron (or a DNN) with some prior input noise with tunable parameter θ_g

D, the discriminative model, is a multilayer perceptron (or a DNN) that represents the probability of some x coming from the data distribution rather than the generative model.

Generative Adversarial Network

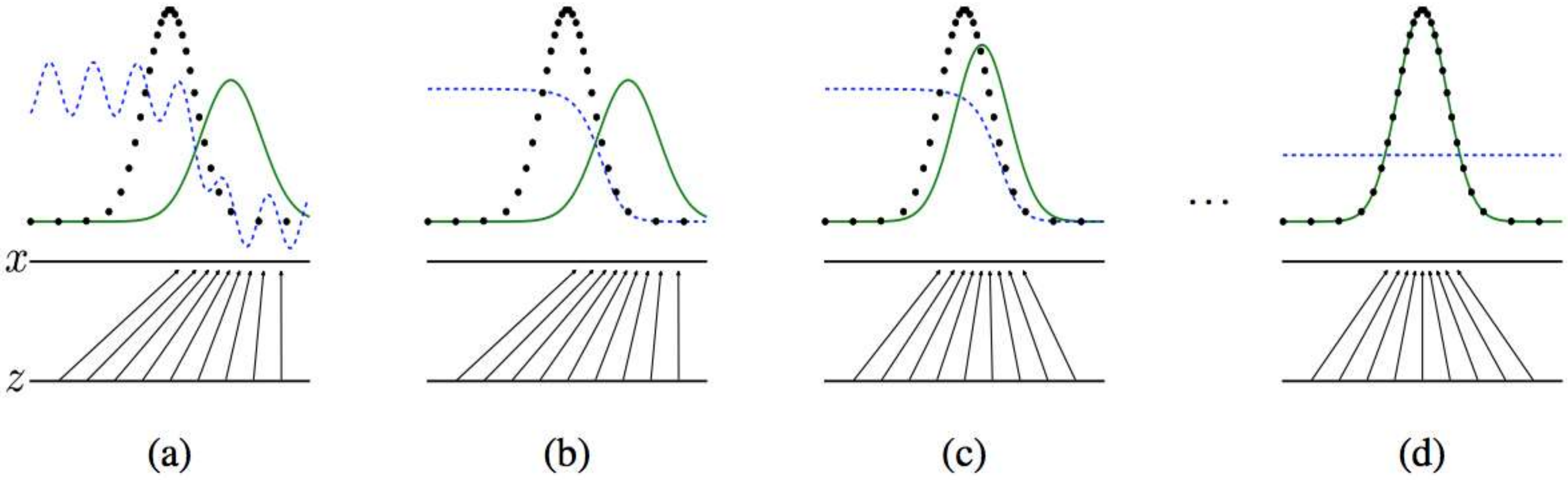


(Goodfellow 2016)

Generative Adversarial Network

MiniMax Game

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\sim p_z(z)} \log(1 - D(G(z)))$$



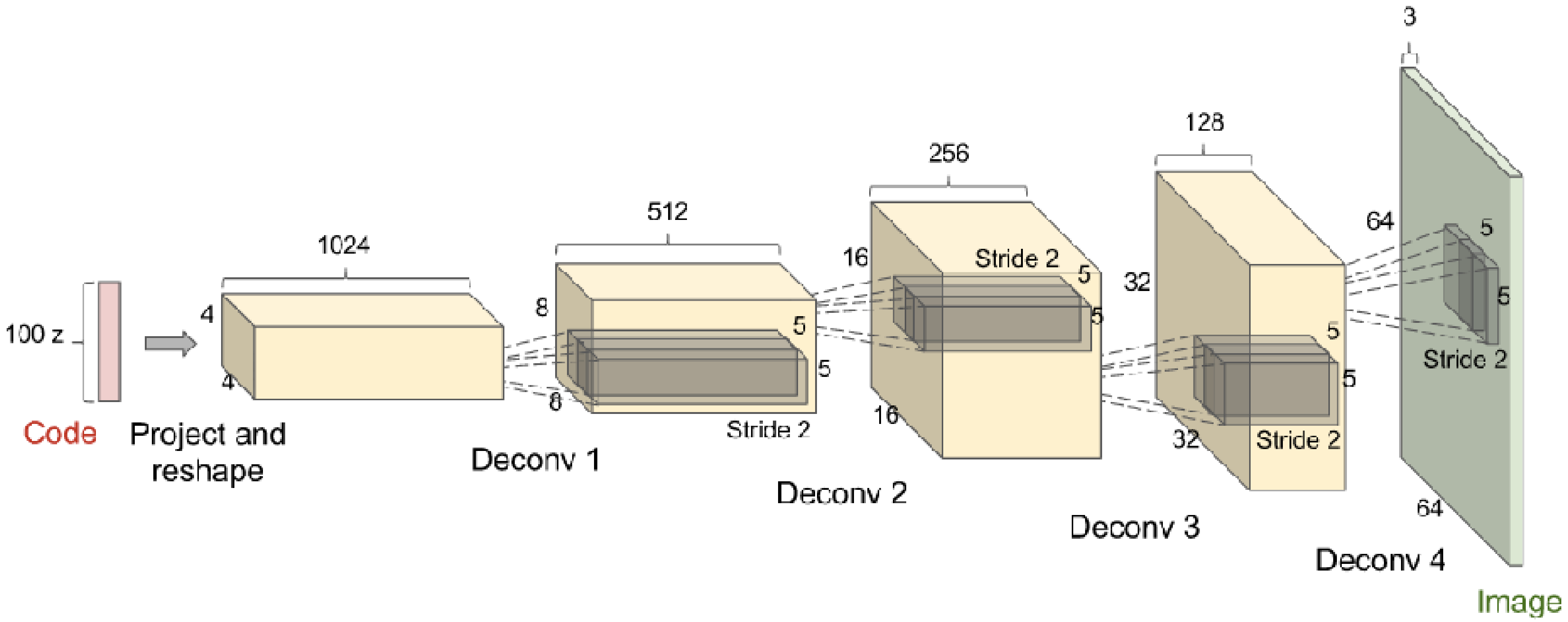
GAN : Balancing *G* and *D*

Usually the discriminator "wins"

- This is a good thing—the theoretical justifications are based on assuming *D* is perfect
- Usually *D* is bigger and deeper than *G*
- Sometimes train *D* more often than *G*.
- Do not try to limit *D* to avoid making it "too smart"
- Use gaussian distribution for Noise
- Use non-saturating functions (ReLu, MaxPool)
- Use Soft and Noisy Labels

Generative Adversarial Network

Example of generator : DCGAN (Deep Convolutional Generative Adversarial Network)



(Radford et al 2015)

Generative Adversarial Network

DCGANs for Bedrooms

(Radford et al 2015)

Progressive Growing of GANs for Improved Quality, Stability, and Variation

Submitted to ICLR 2018

IVI-GAN

Intra-class Variation Isolation in Conditional GANs. R.Marriot & All, arXiv:1811.11296v1, Idemia

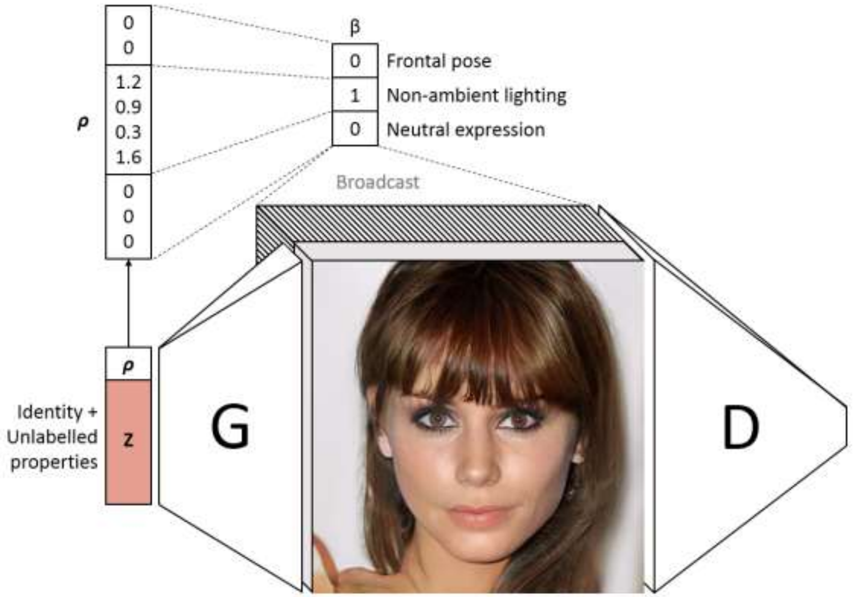


Figure 2. Illustration of the concept of intra-class variation isolation in the generator loss part of the IVI-GAN; i.e. equation (5).

Tips

My Neural Network isn't working! What should I do?

Created on Aug. 19, 2017, 5:56 p.m.

So you're developing the next great breakthrough in deep learning but you've hit an unfortunate setback: your neural network isn't working and you have no idea what to do. You go to your boss/supervisor but they don't know either - they are just as new to all of this as you - so what now?

Well luckily for you I'm here with a list of all the things you've probably done wrong and compiled from my own experiences implementing neural networks and supervising other students with their projects:

1. You Forgot to Normalize Your Data
2. You Forgot to Check your Results
3. You Forgot to Preprocess Your Data
4. You Forgot to use any Regularization
5. You Used a too Large Batch Size
6. You Used an Incorrect Learning Rate
7. You Used the Wrong Activation Function on the Final Layer
8. Your Network contains Bad Gradients
9. You Initialized your Network Weights Incorrectly
10. You Used a Network that was too Deep
11. You Used the Wrong Number of Hidden Units

Daniel Holden

http://theorangeduck.com/page/neural-network-not-working

Tips

- **Gather Data, more Data, use all you can**
  - If enough => deeper Networks
  - If not => Data Augmentation, Drop Out
- **Add small L2 regularization, it never hurts**
- **Use Batch Normalization**
- **Initialization with whitening**
- **Use the newest optimizer**
- **Define a cost function that solves your final problem**
- **Look at performances on different validations datasets**
- **Look at weights evolution**
- **Play with hyper-parameters**

Thank You

IDEMIA