Master Lecture:
Competitive Problem Solving with Deep Learning
Framework, Visualization, Competitive Problem

Dr. Haojin Yang
Internet Technologies and Systems
Hasso Plattner Institute, University of Potsdam

Content

- Overview of deep learning frameworks
- Visualization
  - **Data visualization, *PCA, t-SNE***
  - Neural network visualization
- Image recognition challenge

Feature Distill

- Richness of features indicates the information richness
- This assumption is based on the precondition that the features themselves are not related to each other → **independent**
- Feature B is useless for a given problem, if
  - Feature B is derived from another feature A,
  - or feature B is not correlated to the problem
  - or feature B and feature A describe a same property just in different forms
- Generally speaking, using more features means that we also need more training samples.
- Feature distill

Principal Components Analysis (PCA)

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly **correlated** variables into a set of values of **linearly uncorrelated** variables called **principal components**.

- Simplifying data set
- **Reduce the number of dimensions**
- Maintaining the features in the data set that contribute most to the variance (maintain the "*important*" features)
- Inhibit overfitting
- Visualization of data set
- Linear transformation

Principal Components Analysis (PCA)

PCA workflow:

- Data normalization
- Find the covariance matrix of the sample features
- Select *k* largest eigenvalues
- Build the matrix of selected eigenvectors
- Project the data sample onto the matrix of eigenvectors

Principal Components Analysis (PCA)

An example: Compress *n*-dim features to *k*-dim

- *m* (10) samples
- Each has *n*=2 features

$x^{(1)}$	$x^{(2)}$
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Principal Components Analysis (PCA)

Step 1: Normalization

$$\bar{x}^i = \frac{1}{n-1}\sum_{j=1}^{n} x_j^i$$

$$x_j^i = x_j^i - \bar{x}^i, j = 1, \dots n$$

$$\bar{x}^{(1)} = 1.81$$

$$\bar{x}^{(2)} = 1.91$$

$x_j^{(1)} - \bar{x}^{(1)}$	$x_j^{(2)} - \bar{x}^{(2)}$
0.69	0.49
-1.31	-1.21
0.39	0.99
0.09	0.29
1.29	1.09
0.49	0.79
0.19	-0.31
-0.81	-0.81
-0.31	-0.31
-0.71	-1.01

Principal Components Analysis (PCA)

Step 2: Calculate covariance matrix

$$Var(X) = \frac{1}{n-1}\sum_{j=1}^{n}(x_j^1 - \bar{x}^i)(x_j^1 - \bar{x}^i)$$

$$Cov(X^1, X^2) = \frac{1}{n-1}\sum_{j=1}^{n}(x_j^1 - \bar{x}^1)(x_j^2 - \bar{x}^2)$$

Matrix:

$$Cov(Z) = \begin{pmatrix} Cov(X^1, X^1) & Cov(X^1, X^2) \\ Cov(X^2, X^1) & Cov(X^2, X^2) \end{pmatrix}$$

$$= \begin{pmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{pmatrix}$$

$x_j^{(1)} - \bar{x}^{(1)}$	$x_j^{(2)} - \bar{x}^{(2)}$
0.69	0.49
-1.31	-1.21
0.39	0.99
0.09	0.29
1.29	1.09
0.49	0.79
0.19	-0.31
-0.81	-0.81
-0.31	-0.31
-0.71	-1.01

Principal Components Analysis (PCA)

Step 3.1: eigenvalues

$$Cov(Z) = \begin{pmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{pmatrix}, \mathbf{determinant\ (Cov - \lambda I) = 0}$$

$$\mathrm{Det}\begin{pmatrix} 0.616556 - \lambda & 0.615444 \\ 0.615444 & 0.716556 - \lambda \end{pmatrix} = 0,$$

Calculates the determinant: $(0.616556 - \lambda)(0.716556 - \lambda) - 0.615444^2 = 0$

$$= 0.441797 - 1.333112\lambda + \lambda^2 - 0.378771 = \underset{a}{1}\lambda^2 - \underset{b}{1.333112}\lambda + \underset{c}{0.063026}$$

$$\boldsymbol{D = b^2 - 4ac > 0 \rightarrow \text{two eigenvalues } \lambda}$$

$$= (-1.333112)^2 - 4 * 1 * 0.063026 = \mathbf{1.525084 > 0}$$

$$\lambda_1 = \frac{-b - \sqrt{D}}{2a} = \frac{1.333112 - 1.234943}{2} = \mathbf{0.0490854}$$

$$\lambda_2 = \frac{-b + \sqrt{D}}{2a} = \frac{1.333112 + 1.234943}{2} = \mathbf{1.2840275}$$

Principal Components Analysis (PCA)

Step 3.2: eigenvectors

$$\text{For } \lambda_1: \begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = 0.0490845 * \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}$$

$$\Rightarrow \begin{cases} 0.616556x_{11} + 0.615444x_{12} = 0.049085x_{11} \\ 0.615444x_{11} + 0.716556x_{12} = 0.0490845x_{12} \end{cases}$$

$$\Rightarrow \begin{cases} 0.567471x_{11} = -0.615444x_{12} \\ 0.615444x_{11} = -0.6674715x_{12} \end{cases} \Rightarrow x_{11} = -1.08454x_{12} \Rightarrow \begin{pmatrix} \mathbf{-0.73518} \\ \mathbf{0.677873} \end{pmatrix}$$

$$\text{For } \lambda_2: \begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = 1.2840275 * \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}$$

$$\Rightarrow x_{11} = 0.922053x_{12} \Rightarrow \begin{pmatrix} \mathbf{-0.677875} \\ \mathbf{-0.73518} \end{pmatrix}$$

Step 3: eigenvalues and eigenvectors

$$Cov(Z) = \begin{pmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{pmatrix}$$

$$eigenvalues = \begin{pmatrix} 0.049085 \\ 1.284028 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -0.73518 & -0.677875 \\ 0.677873 & -0.73518 \end{pmatrix}$$

Principal Components Analysis (PCA)

Step 4: Build the matrix of eigenvectors

- Select *k* largest eigenvalues, (in this case *k*=1)

$$eigenvalues = \begin{pmatrix} 0.049085 \\ \mathbf{1.284028} \end{pmatrix}$$

- Use the corresponding eigenvectors as the columns of the matrix

$$eigenvectors = \begin{pmatrix} -0.73518 & \mathbf{-0.677875} \\ 0.677873 & \mathbf{-0.73518} \end{pmatrix}$$

$$\begin{bmatrix} \mathbf{-0.677875} \\ \mathbf{-0.73518} \end{bmatrix}$$

Principal Components Analysis (PCA)

Step 5: Project the data sample onto the matrix of eigenvectors

- We have m data samples, n features, after normalization: $DataA(m \times n)$
- $Cov(Z)$: $(n \times n)$, select k eigenvectors as matrix:
- $EigenVectors(n \times k)$
- Data samples after projection:

$$FinalData(m \times k) = DataA(m \times n) \cdot EigenVectors(n \times k), \quad F_n \rightarrow F_k$$

Principal Components Analysis (PCA)

Step 5: Project the data sample onto the matrix of eigenvectors

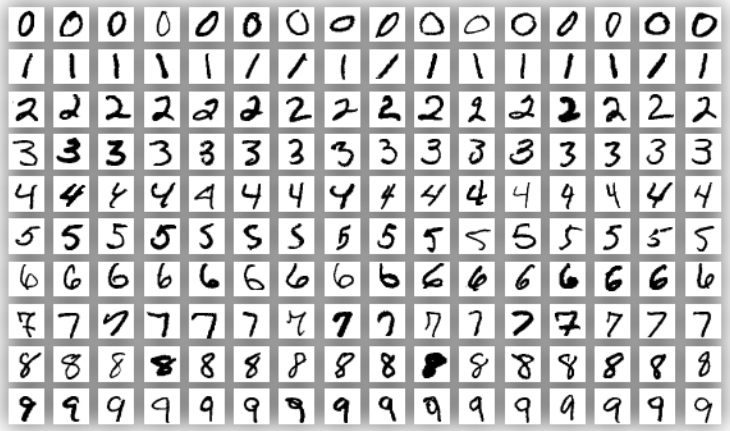
$$FinalData(m \times k) = DataA(m \times n) \cdot EigenVectors(n \times k), \quad F_n \rightarrow F_k$$

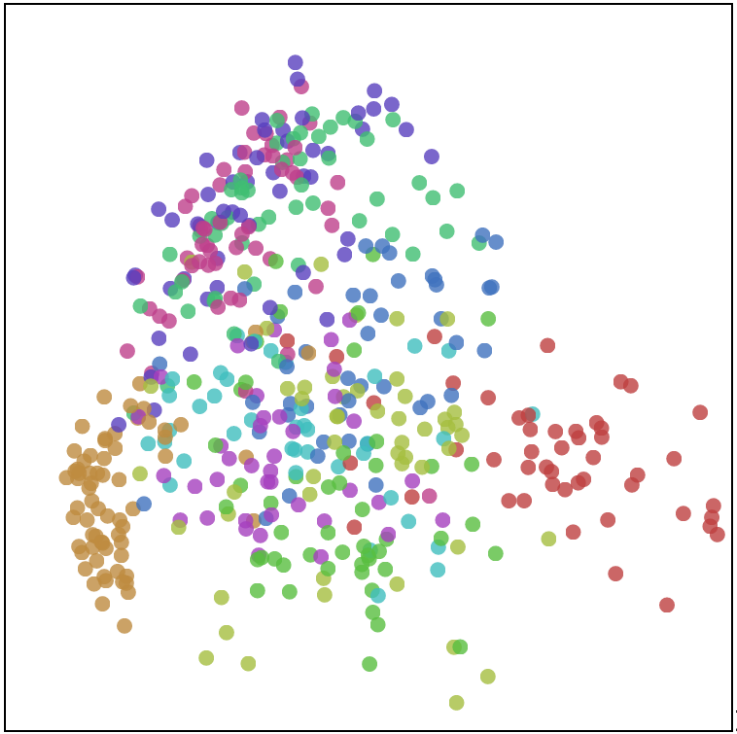
$x^{(1)}$	$x^{(2)}$
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

x
-0.8279
1.7776
-0.9922
-0.2742
-1.6758
-0.9129
0.0991
1.1146
0.438
1.2238

Principal Components Analysis (PCA)

Visualizing MNIST with PCA





Source: http://colah.github.io

SNE (Stochastic Neighbor Embedding)

- SNE (Stochastic Neighbor Embedding, *Hinton and Roweis, 2002*)
  - SNE constructs a probability distribution among high-dimensional samples
  - SNE constructs the probability distribution of these samples in the low-dimensional space, and
  - making the two probability distributions as similar as possible

SNE (Stochastic Neighbor Embedding)

- In **high-dimensional space** R^x, convert **Euclidean Distance to Conditional Probability** to express the similarity between samples,
- e.g. $x_i and\, x_j$, x_i choose its neighbors x_j based on $p_{j|i}$

$$p_{j|i} = \frac{e^{(-\|x_i - x_j\|^2 / 2\sigma_i^2)}}{\sum_{k \neq i} e^{(-\|x_i - x_k\|^2 / 2\sigma_i^2)}}$$

- When we map the data to **low-dimensional** space R^y, should get similar correlations between samples
- $x_i and\, x_j \rightarrow y_i and\, y_j$ and set $\sigma^2 = \frac{1}{\sqrt{2}}$

$$q_{j|i} = \frac{e^{(-\|y_i - y_j\|^2)}}{\sum_{k \neq i} e^{(-\|y_i - y_k\|^2)}}$$

SNE (Stochastic Neighbor Embedding)

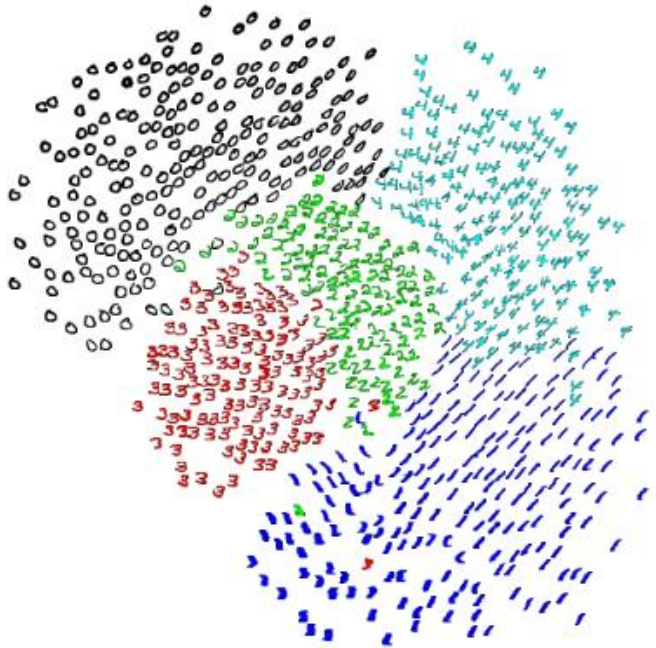
- SNE uses gradient descent to optimize the cost function *C* based on **KL** distance (Kullback-Leibler Divergence) of P_i and Q_i:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

- Gradient wrt. y_i:

$$\frac{dC}{dy_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

- Disadvantages
  - KL is an asymmetric metric, hard to optimize
  - Crowding problem, the boundaries are too blurred



Hinton and Roweis, 2002

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- t-SNE *(Maaten and Hinton 2008)*
  - A nonlinear dimension reduction algorithm
  - From high dimension to 2 or 3 dimension, data visualization
- t-SNE improved two problems of SNE
  - **Asymmetric SNE**
  - **The Crowding Problem**

t-SNE

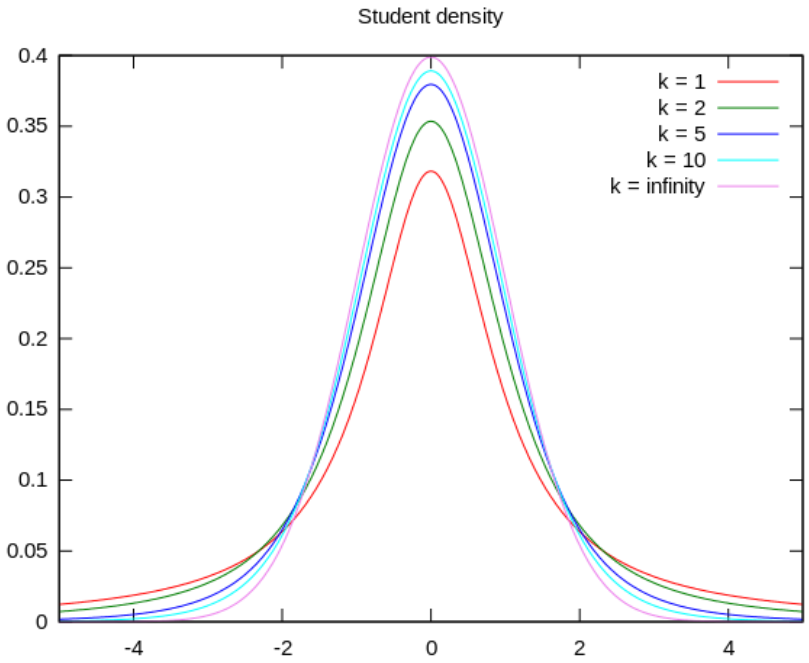
Symmetric SNE

- Uses **joint probability distribution**, $\forall i,j: p_{ij} = p_{ji}, q_{ij} = q_{ji}$
- Low dimensional space: $q_{ij} = \frac{e^{(-\|y_i - y_j\|^2)}}{\sum_{k \neq i} e^{(-\|y_i - y_k\|^2)}}$
- High dimensional space, simply: $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$ where *n*: number of samples
- Cost function $C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$
- Gradient: $\frac{dC}{dy_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$
- Simple, more efficient computation

t-SNE

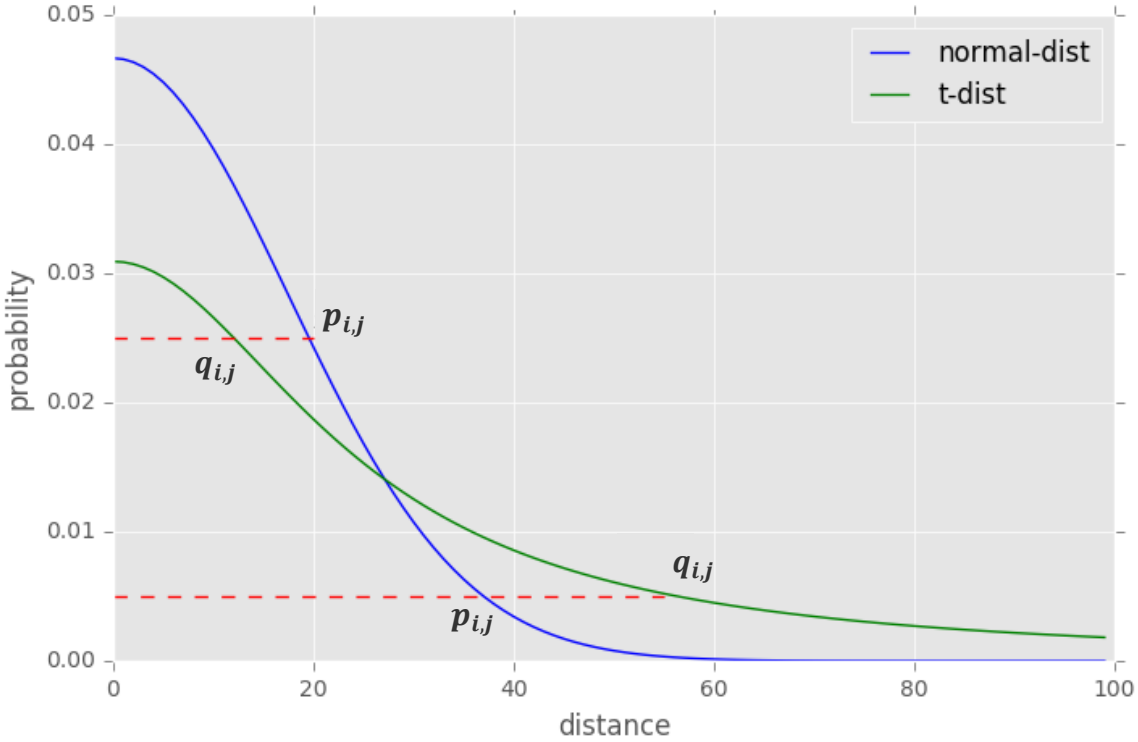
Solve the Crowding Problem

- In the visualization, different types of clusters crowded together, can not be distinguished clearly
- t-Distribution
  - More suitable than normal distribution for dealing with small sample set and outliers



t-SNE

- **t-Distribution**



t-SNE

- Redefine q_{ij} for low-dimensional space with freedom degree equals 1

$$q_{ij} = \frac{\left(1+\|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq i}\left(1+\|y_i - y_j\|^2\right)^{-1}}$$

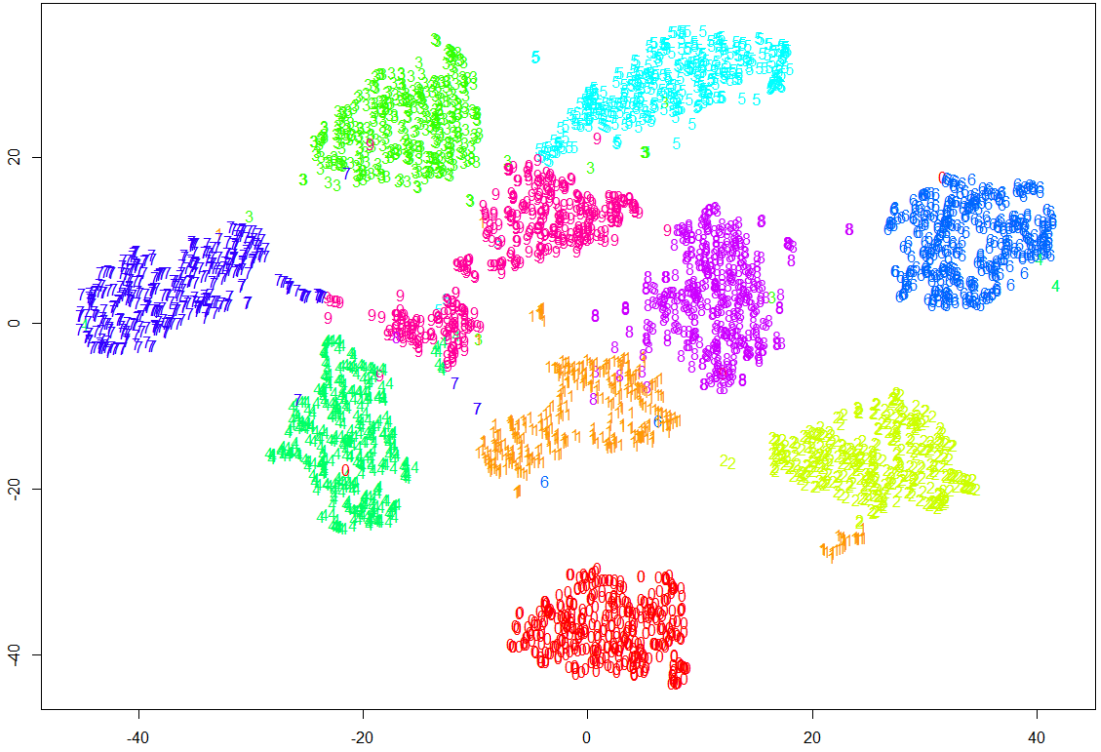
- Gradient wrt. y_i:

$$\frac{dC}{dy_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

- Disadvantages:
  - Time and space complexity $O(N^2)$ wrt. number of samples
  - Neither classification nor regression

t-SNE

- **Visualizing MNIST with t-SNE**



Content

- Overview of deep learning frameworks
- Visualization
  - Data visualization, *PCA, t-SNE*
  - **Neural network visualization**
- Image recognition challenge

Weight Filters and Activations Visualization

Learned weights filter

Activations from data

Activation gradients

Conv1

ReLu1

Pool1

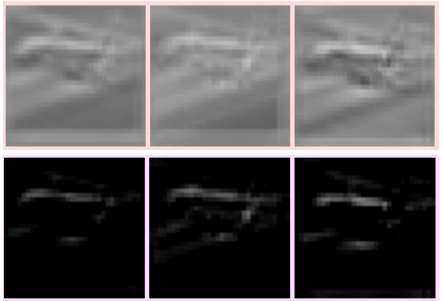
Conv2

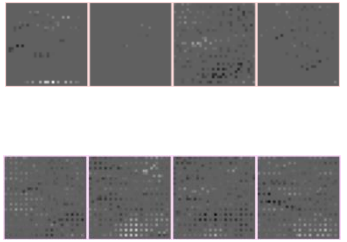
...

Pool4

fc 5

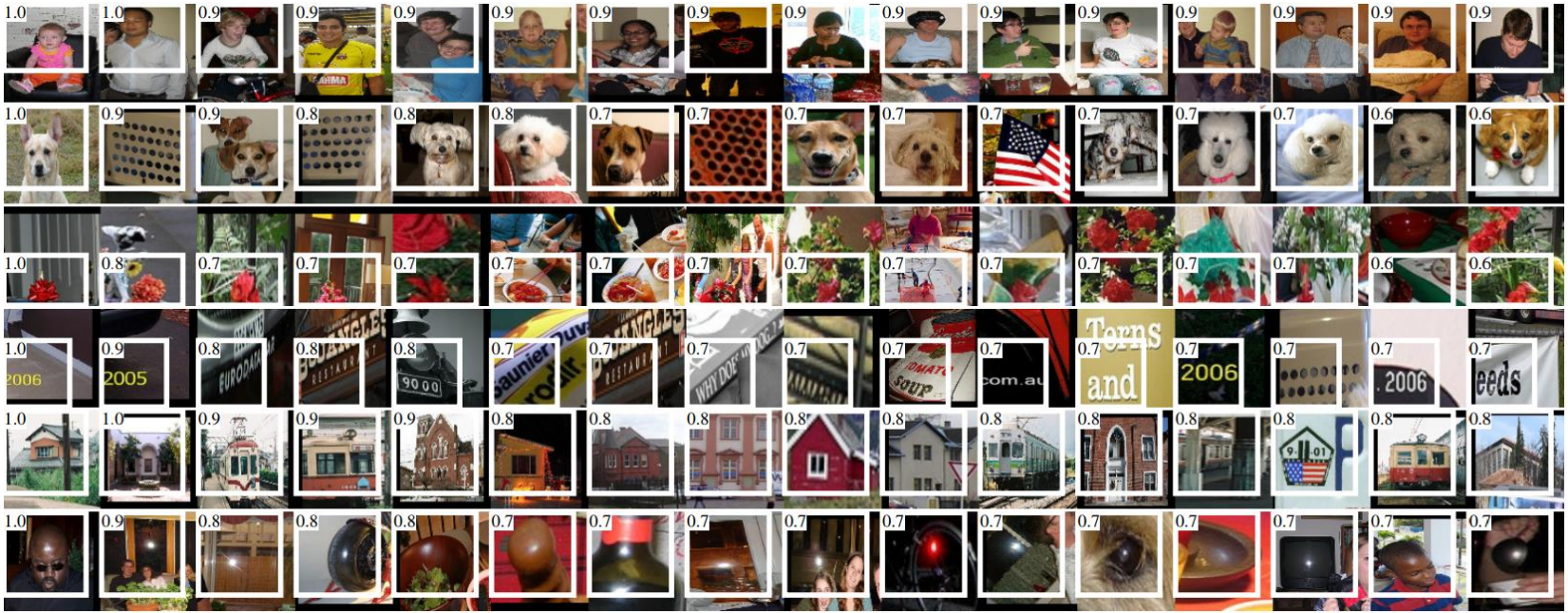
softmax





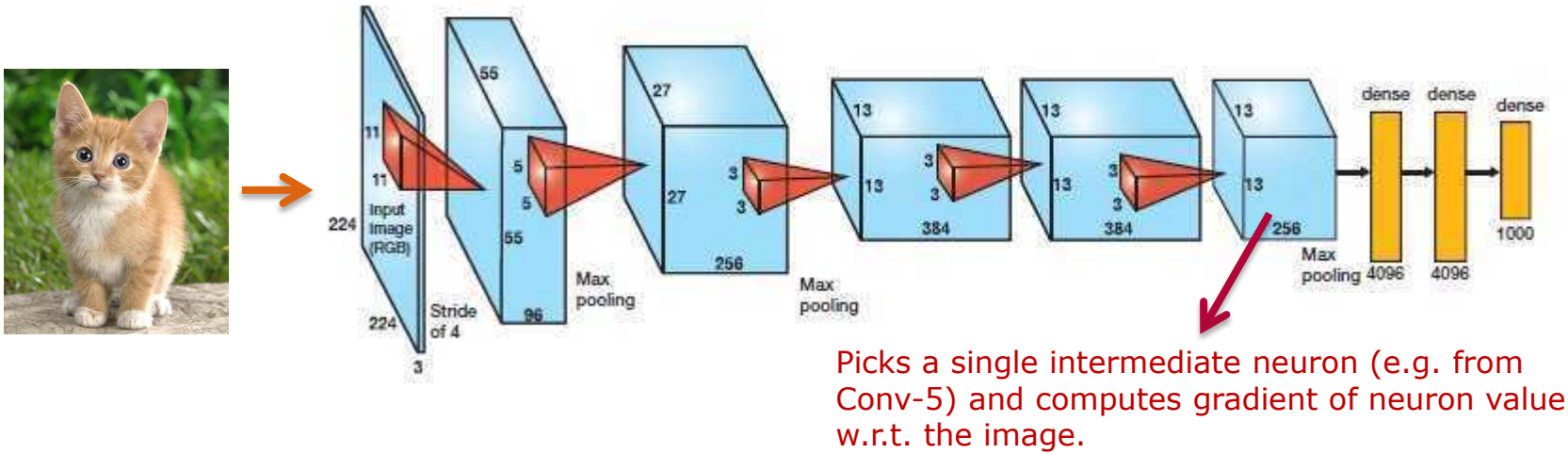
Source: ConvNetJS

Visualization of Receptive Fields



Girshick et al., "Rich feature hierarchies for accurate object detection and semantic segmentation" 2014

Gradient Based Approach

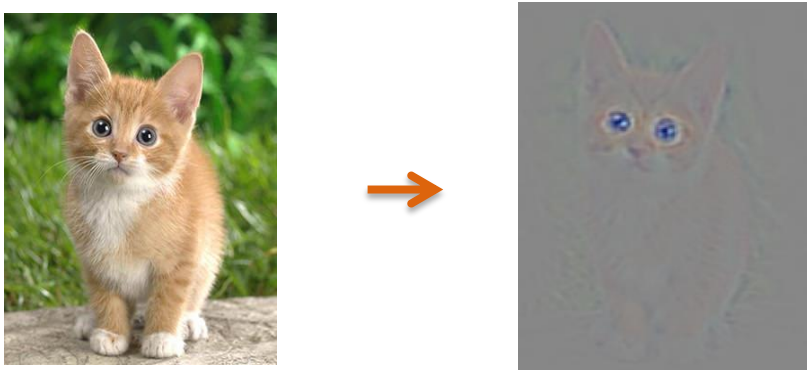


Picks a single intermediate neuron (e.g. from Conv-5) and computes gradient of neuron value w.r.t. the image.

- Visualize the image pixels that mostly activate a neuron in a deeper layer
  - Forward input up to the target layer e.g. conv-5
  - Set all gradients to 0
  - Set gradient for the specific neuron to 1
  - Backpropagate to get reconstructed image, showing gradient on the image

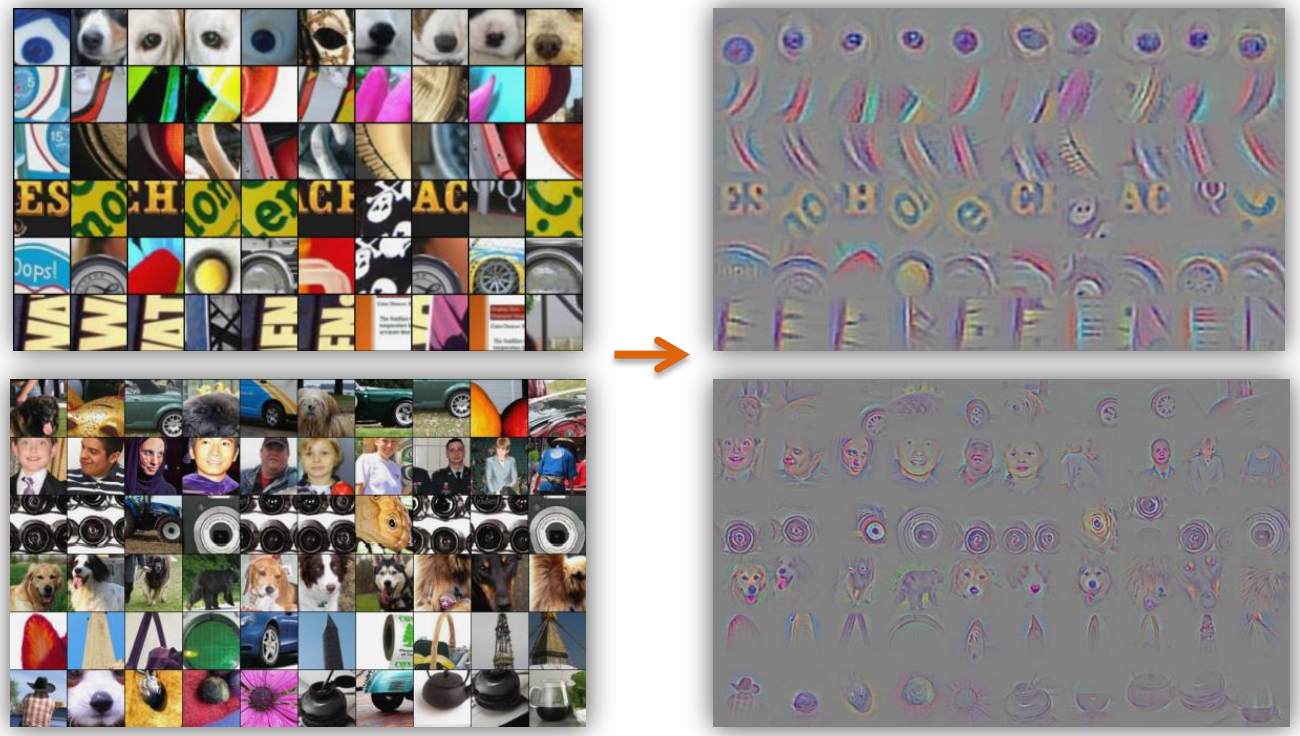
Gradient Based Approach

- Forward input up to the target layer e.g. conv-5
- Set all gradients to 0
- Set gradient for the specific neuron to 1
- Backpropagate to get reconstructed image, showing gradient on the image



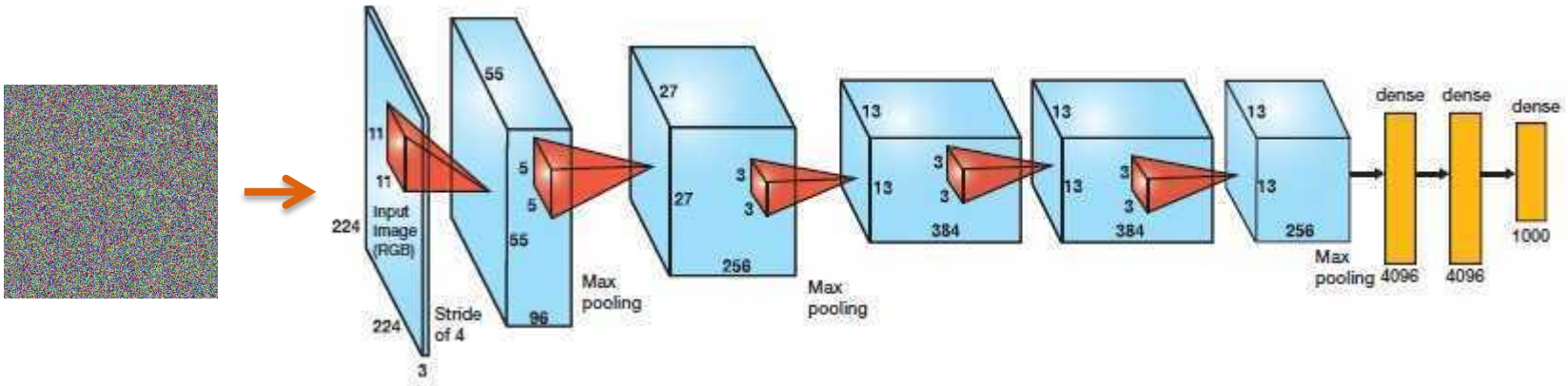
Springenberg et al., "Striving for Simplicity: The All Convolutional Net" 2015

Gradient Based Approach



Springenberg et al., "Striving for Simplicity: The All Convolutional Net" 2015

Optimization Based Approach



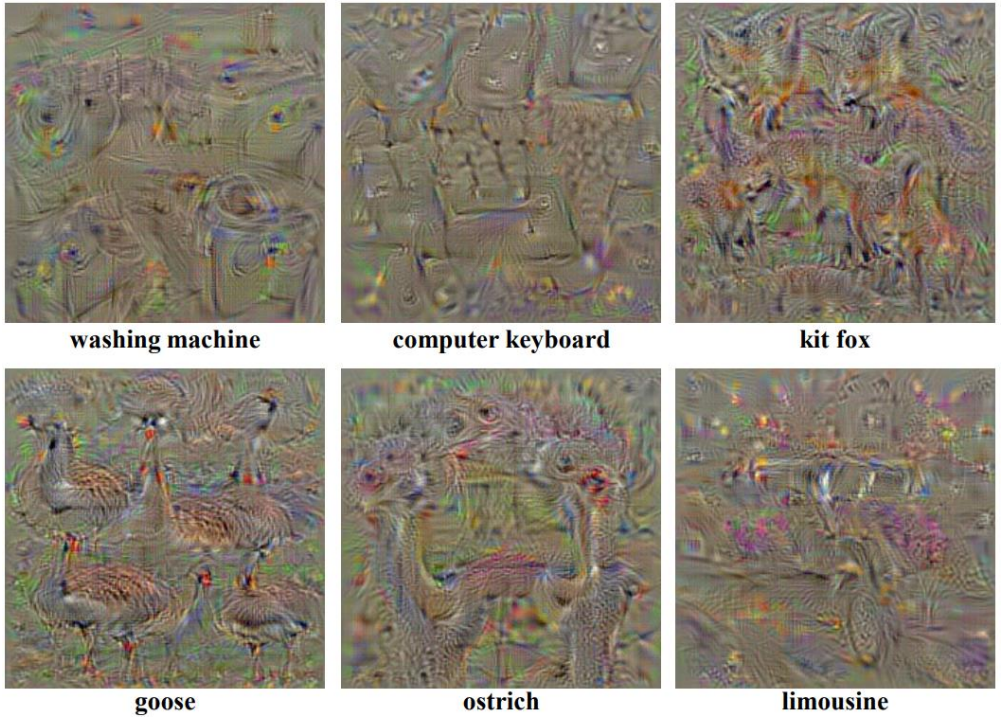
- **Generate an image that maximizes a class score** (or a neuron activation)
- Forward a random image
- Repeat{
  - Set the gradient of the scores vector to be [0,0,…,1,…,0]
  - Backprop to get gradient on the image
  - Update image with a small step in the gradient direction

  }

Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and aliency Maps" 2014

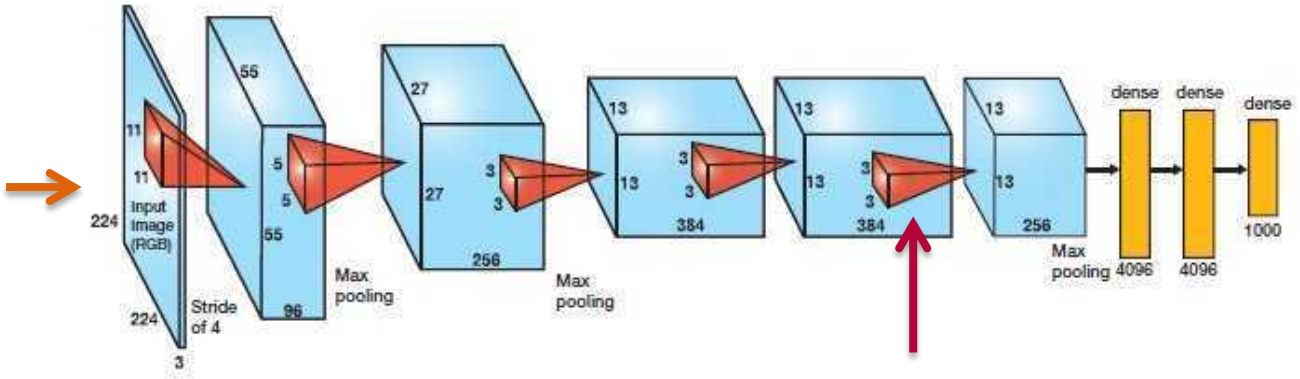
Optimization Based Approach

- **Generate an image that maximizes a class score** (or a neuron activation)



Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and aliency Maps" 2014

Deep Dream



- Repeat{
  - Forward image up to a specific layer e.g. Conv-5
  - **Set the gradients to equal the layer activations**
  - Backprop to get gradient on the image
  - Update image with small step

  }

https://github.com/google/deepdream 2015

Deep Dream

Set the gradients to equal the layer activations

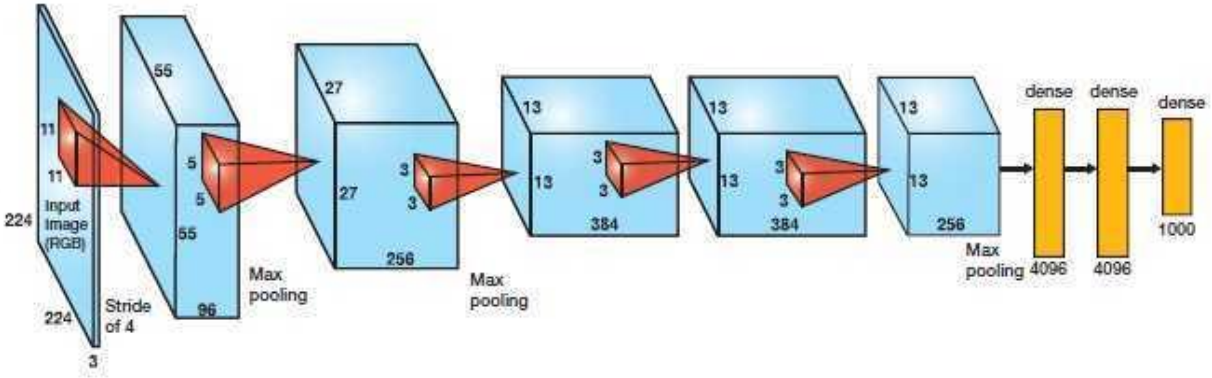
- Rather than synthesizing an image to maximize a specific neuron, at each iteration, the image is updated to **boost all features that activated in that layer in the forward pass**

Inception_4a output

Deep Dream

https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Neural Style

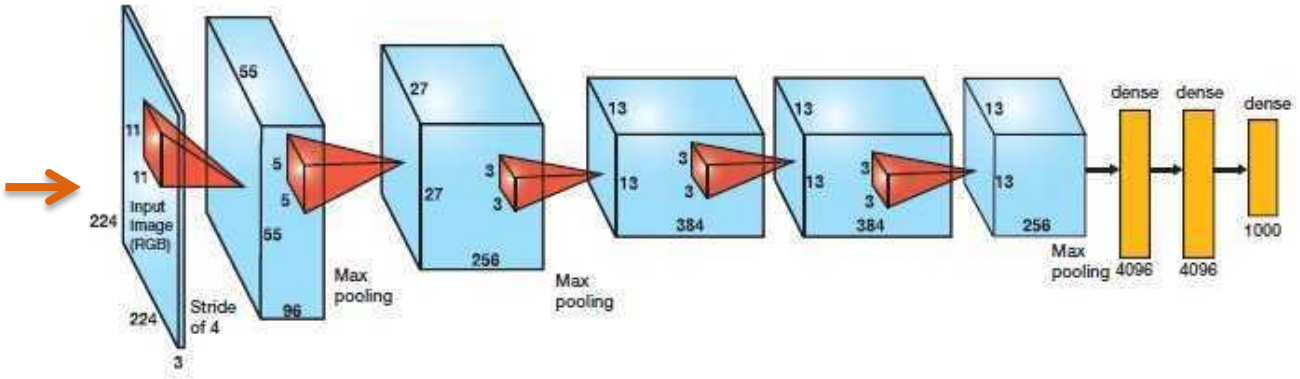


Content image

- Extracts **raw activations** in all layers
- Activations represent the image content

Gatys et al., "Image Style Transfer Using Convolutional Neural Networks", 2016

Neural Style

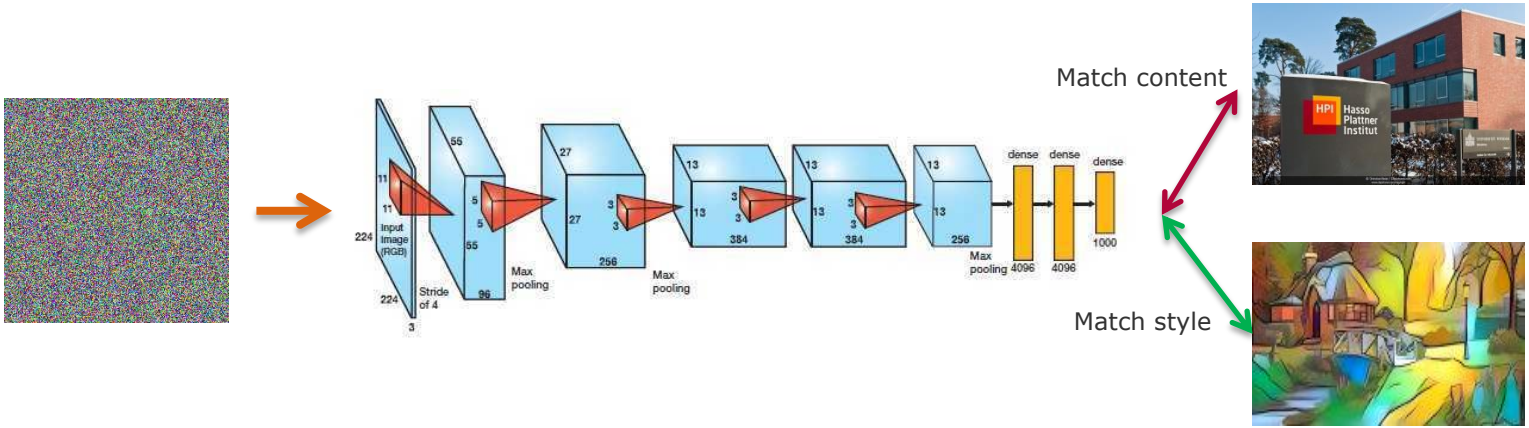


Style image

- Extracts **activations** from style image in all layers
- Instead raw activations, computes **Gram Matrix** G at each layer depicting style
  - $G = V^T V$, where V has dimension $[W * H, D]$
  - Gram matrix G gives the **correlations** between channel responses

Gatys et al., "Image Style Transfer Using Convolutional Neural Networks", 2016

Neural Style



- Simultaneously matches the content representation of $\vec{p}$ and the style representation of $\vec{a}$
- Thus jointly minimize:

$$\boldsymbol{L_{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

Match activations from content image

Match Gram matrices from style image

Gatys et al., "Image Style Transfer Using Convolutional Neural Networks", 2016

Neural Style

Gatys et al., "Image Style Transfer Using Convolutional Neural Networks", 2016

Neural Style

Content image

+

Style image

"Starry night" by Vincent Van Gogh

Neural Style

Luan et al., "Deep Photo Style Transfer", 2017

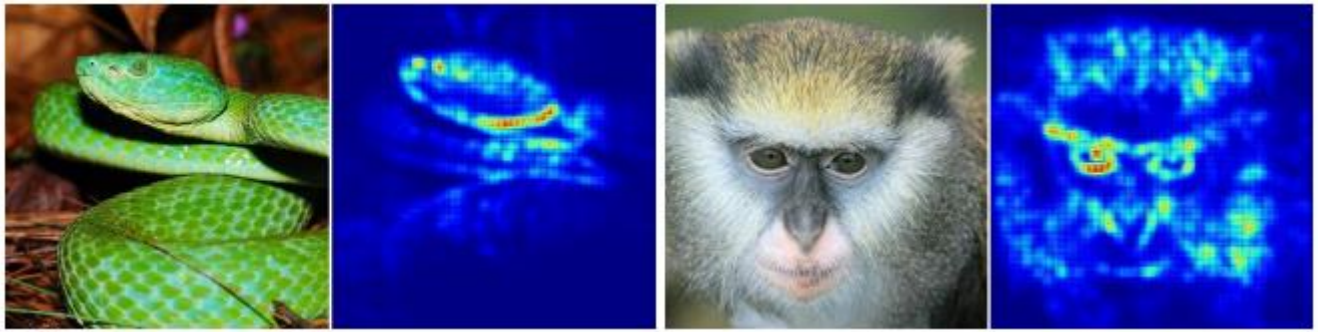
VisualBackProp

Feature maps in a CNN model

- In deeper layers: contains higher abstraction, more important for the prediction results, but with very low resolution
  - e.g. the last feature map of VGG-16 is 14x14
- In shallow layers: low level abstraction, but higher resolution
- Can we take advantages of higher abstraction from deeper layer and higher resolution from shallow layer?
  - *Bojarski et al., "VisualBackProp: efficient visualization of CNNs" 2017*

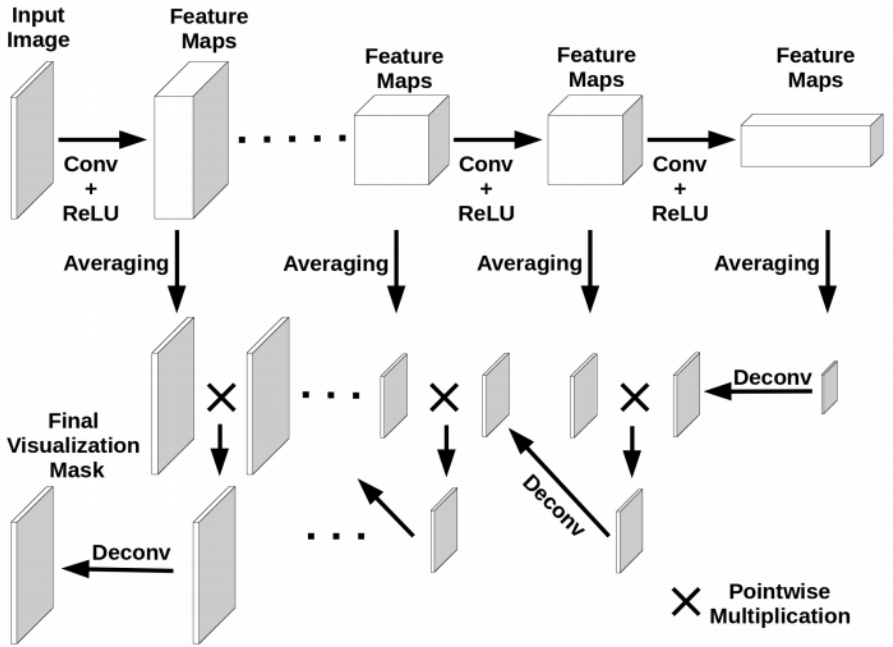
VisualBackProp

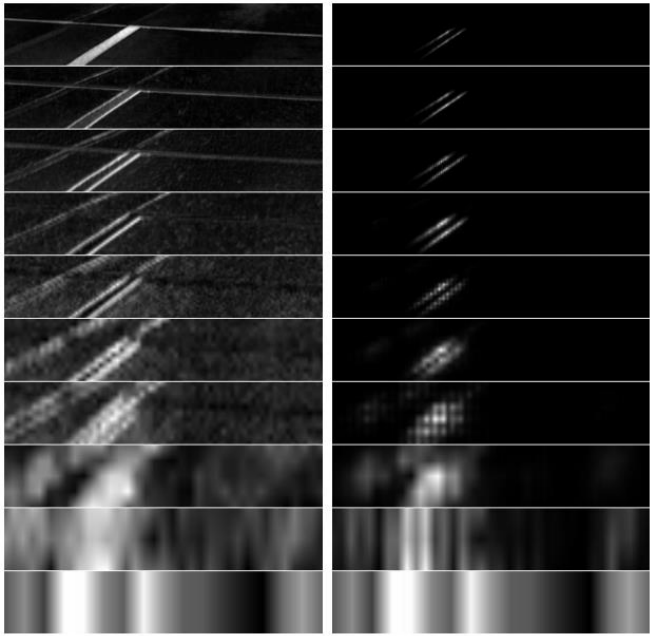
- For visualizing which sets of pixels of the input image contribute most to the predictions
- As a debugging tool for the development of CNN-based systems
- High efficient, thus can be used during both training and inference
- VisualBackProp is a value-based method:
  - Backpropagate values (images) instead of the gradients



Bojarski et al., "VisualBackProp: efficient visualization of CNNs" 2017

VisualBackProp

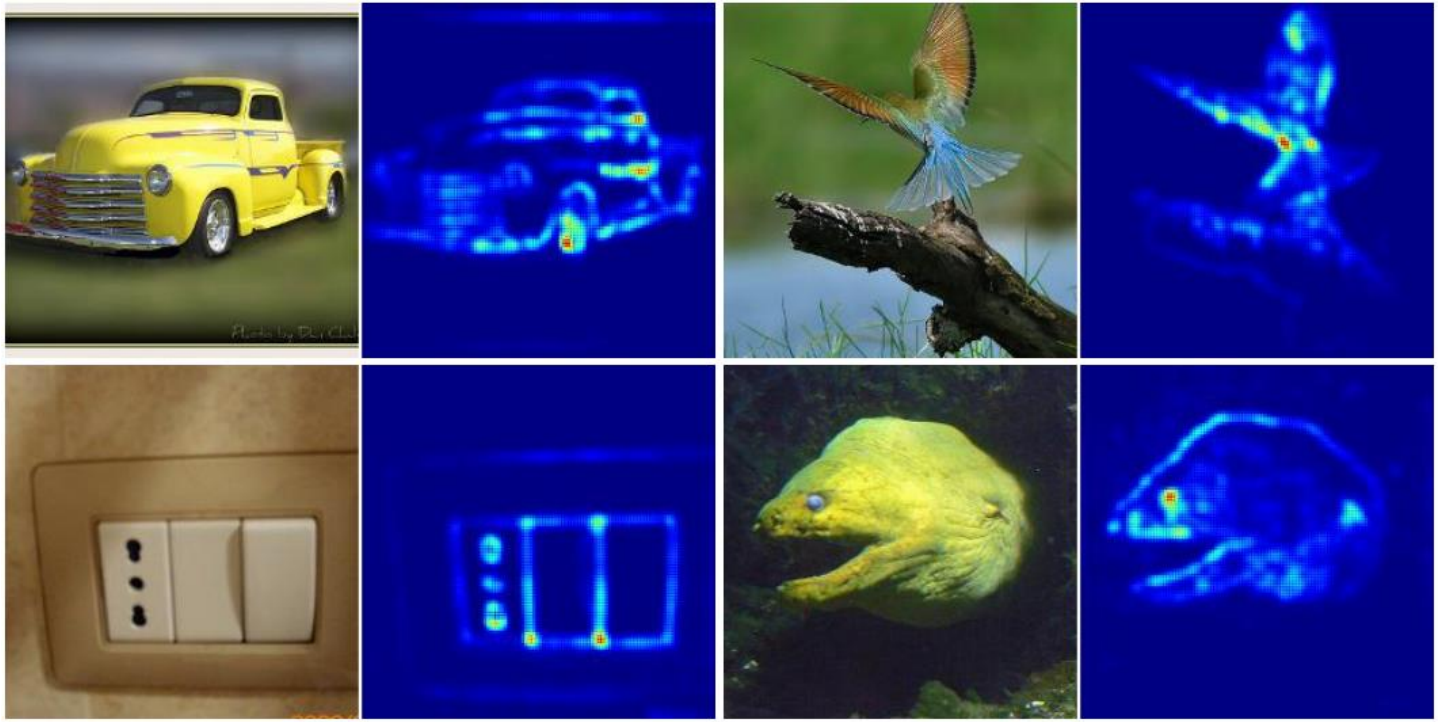




Averaged feature maps intermediate masks

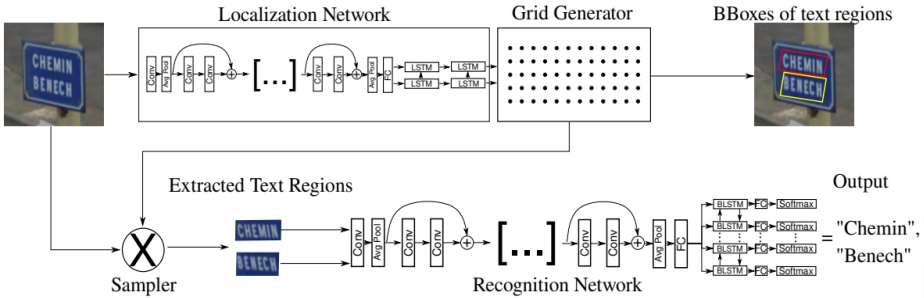
Bojarski et al., "VisualBackProp: efficient visualization of CNNs" 2017

VisualBackProp



Bojarski et al., "VisualBackProp: efficient visualization of CNNs" 2017

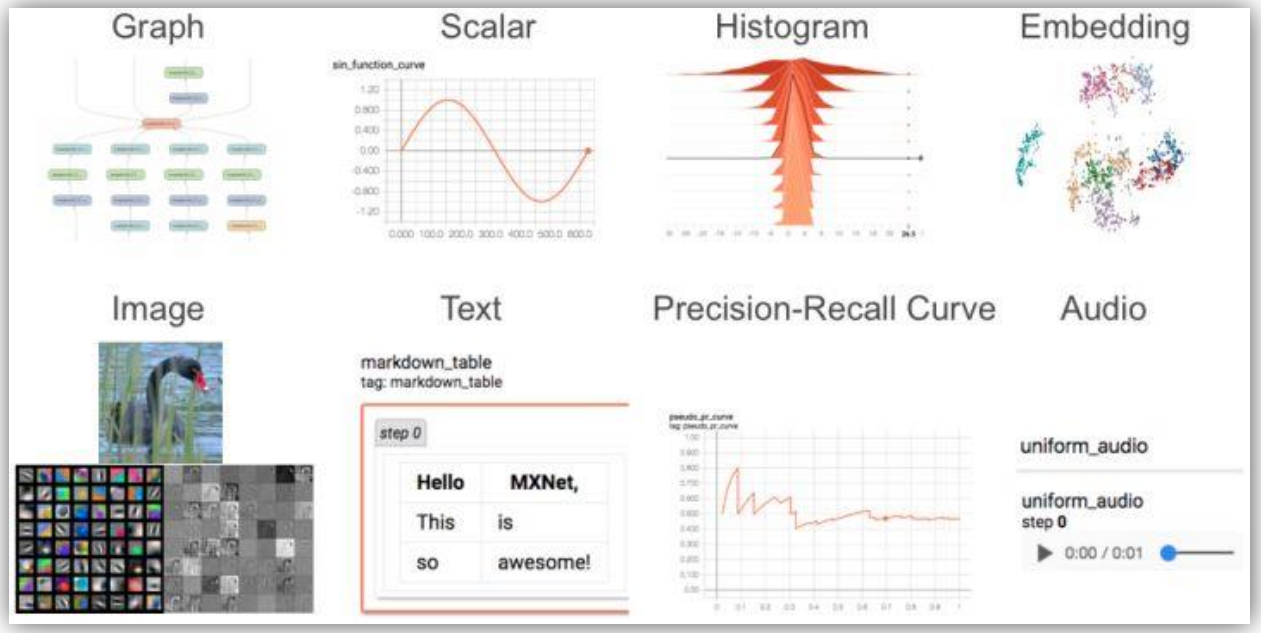
VisualBackProp



Semi-supervised end-to-end scene text recognition (Bartz et al., 2017)

Visualization Tool

- Visualization with TensorBoard
- tensorflow/**tensorboard**
- awslabs/**mxboard**
- **tensorboard-pytorch**
- **tensorboard-chainer**

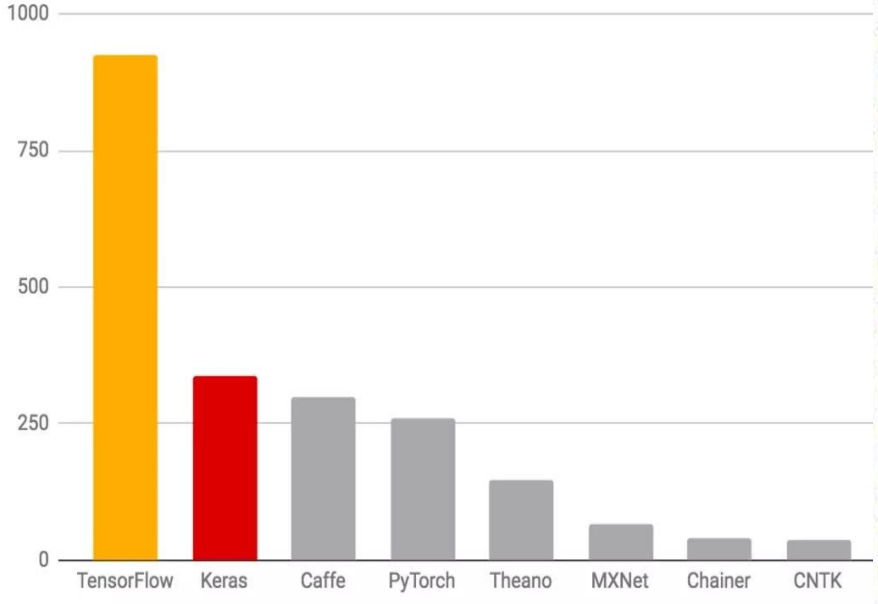


Content

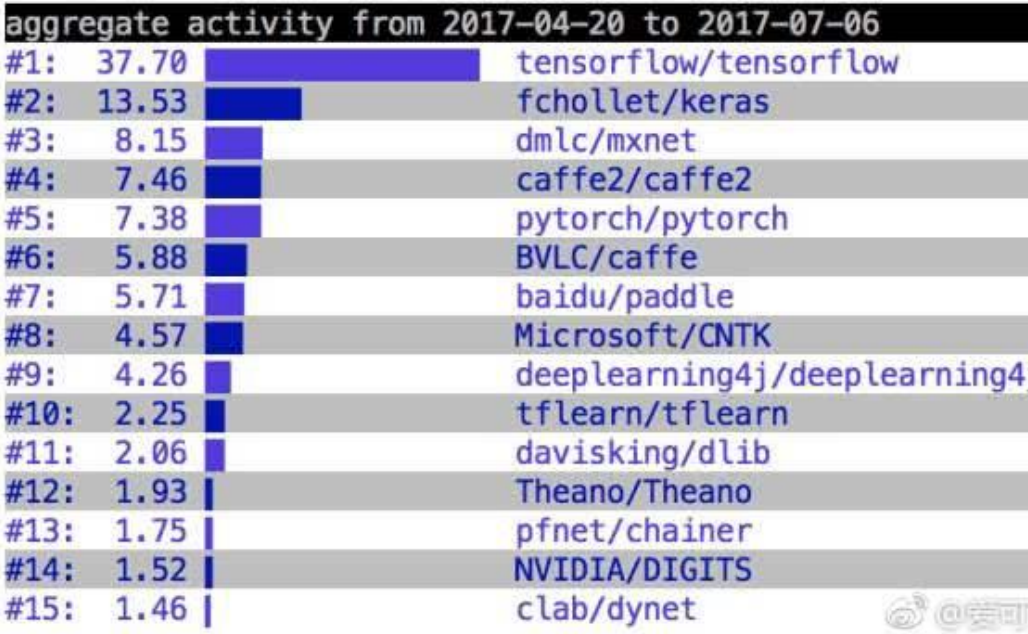
- **Brief overview of deep learning frameworks**
- Visualization
  - Data visualization, *PCA, t-SNE*
  - Neural network visualization
- Image recognition challenge

Deep Learning Frameworks

Activity of Deep Learning Frameworks



arXiv mentions as of 2018/03/07 (past 3 months)



Github aggregate activity April-July 2017

DL Framework Features

Framework	Core language	Platform	Interface	Distributed training	Model ZOO	Multi-GPU	Multi-threaded CPU
Caffe	C++	Linux, MacOS, Windows	Python, Matlab	No	yes	Only data parallel	yes
Tensorflow	C++	Linux, MacOS, Windows	Python, Java, Go	yes	yes	Most flexible	yes
MXNet	C++	Linux, MacOS, Windows, Devices	Python, Scala, R, Julia, Perl	yes	yes	yes	yes
Pytorch	Lua	Linux, MacOS, Windows	Python	yes	yes	yes	yes
Chainer	Python	Linux	Python	yes	yes	Yes	Via openblas
CNTK	C++	Windows, Linux	Python, C#	yes	yes	yes	yes

Deep Learning Frameworks Benchmarking

- Data set CIFAR-10,
- Task: average time for 1,000 images using ResNet-50 for feature extraction

DL Library	K80/CUDA8/cuDNN6	P100/CUDA8/cuDNN6
Caffe2	**148**	54
Chainer	162	69
CNTK	163	53
Gluon	152	62
Keras(CNTK)	194	76
Keras(TF)	241	76
Keras(Theano)	269	93
Tensorflow	173	57
Lasagne(Theano)	253	65
MXNet	**145**	**51**
PyTorch	169	**51**
Julia-Knet	159	n/a

Source: https://analyticsindiamag.com/evaluation-of-major-deep-learning-frameworks/

Content

- Overview of deep learning frameworks
- Visualization
  - Data visualization, *PCA, t-SNE*
  - Neural network visualization
- **Image recognition challenge**

Image Captioning

- Roadmap
- Data set
- Evaluation
- Grouping method
- GPU usage schedule

Project - Roadmap

- **Competitive problem solving: An image recognition challenge**
  - 11.06.2018 Challenge open: Release training und validation data, grouping
  - 02.07.2018 Release test set
  - 09.07.2018 Release pre-ranking result
  - 09-10.07.2018 Model submission: Tutors will run the models using a secret test dataset
  - 16.07.2018 **Final presentation**, release final ranking result, awards granting (**20%**)
  - Until 31. August Final submission: **Implementation + Paper** (**40%**)
- **Weekly individual meeting with your tutors during the project**

Data set

- Each photo with 5 captions
- Training and validation data
  - /data/dl_lecture_data/TrainVal on each server


```
{
  "license":5,
  "file_name":"COCO_val2014_000000003310.jpg",
  "coco_url":"http://images.cocodataset.org/val2014/
              COCO_val2014_000000003310.jpg",
  "height":640,
  "width":480,
  "date_captured":"2013-11-20 08:09:40",
  "flickr_url":"http://farm5.staticflickr.com/4020/4667
              015053_4566d4aaa0_z.jpg",
  "id":3310
}
{
  "image_id":3310,
  "id":621942,
  "caption":"A woman is taking a bite out of a hot dog."
},
{
  "image_id":3310,
  "id":625008,
  "caption":"A person with a red hat bites into a hot dog."
},
{
  "image_id":3310,
  "id":625014,
  "caption":"The woman is enjoying her very large hot dog."
}
...
```

Model Submission

- Nvidia-Docker image (*ready-to-run)*
- Well written documentation for your docker file

Evaluation

- Bleu :A Method for Automatic Evaluation of Machine Translation
  - analyzes the co-occurrences of n-grams between the candidate and reference
- Meteor: Automatic Machine Translation Evaluation System
  - is calculated by generating an alignment between the words in the candidate and reference
- ROUGE: A Package for Automatic Evaluation of Summaries
- CIDEr: Consensus-based Image Description Evaluation
  - measures consensus in image captions by performing TF-IDF weighting for each n-gram
- SPICE: Semantic Propositional Image Caption Evaluation
- Evaluation toolkit: coco-caption

Grouping

- Up to 8 groups
  - Each group accepts maximal 6 people
- Please add your name on Doodle

GPU usage schedule

- We offer 3 servers with totally **8** GPUs
- Each team can select **2** time slots for setup, installation etc. (1 day per slot)
- Each team can select **4** time slots for experiments (3 days per slot)
- For each time slot **2** GPUs can be used
- Add your time slots use this link or QR code

Thank you for your Attention!
Have fun with the project!

Reference

- [Goodfellow15] Ian Goodfellow et al., „Expaining and harnessing adversarial examples“, ICLR 2015
- [Brown17] Tom B. Brown et al., „Adversarial Patch“, *arXiv preprint arXiv:1712.09665*
- [Gary17] Gary Marcus, „Deep Learning: A Critical Appraisal”
- [Hebb] Hebb D. O., The organization of behaviour.New York: Wiley & Sons.
- [Ronsenb58] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

Contact

Dr. Haojin Yang

Office: H-1.22

Email: haojin.yang@hpi.de

Dr. Xiaoyin Che

Office: H-1.22

Email: xiaoyin.che@hpi.de

Christian Bartz, M.sc

Office: H-1.11

Email: chrisitan.bartz@hpi.de

Mina Rezaei, M.sc

Office: H-1.22

Email: mina.rezaei@hpi.de

Goncalo Mordido, M.sc

Office: H-1.22

Email: Goncalo.Mordido@hpi.de

Joseph Bethge, M.sc

Office: H-1.21

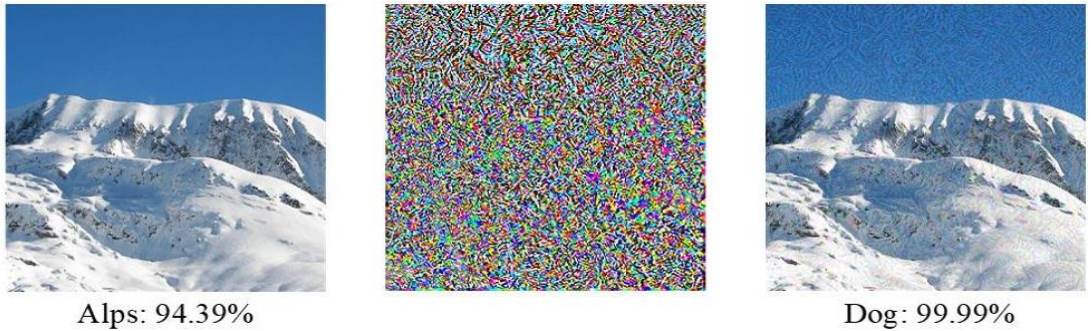
Email: joseph.bethge@hpi.de

Content

- Overview of deep learning frameworks
- Visualization
  - Data visualization, *PCA, t-SNE*
  - Neural network visualization
- **Adversarial examples**
- Image recognition challenge

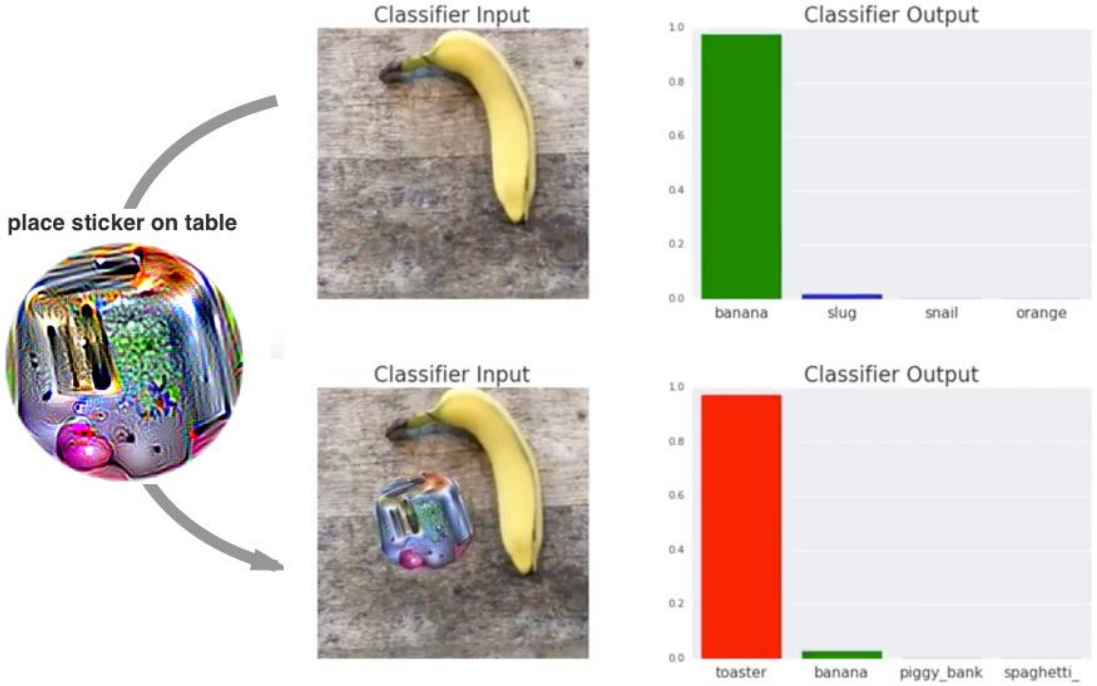
Deep Learning - Current Limitations
Adversarial Examples

Adversarial examples is a class of samples that are maliciously designed to attack machine learning models



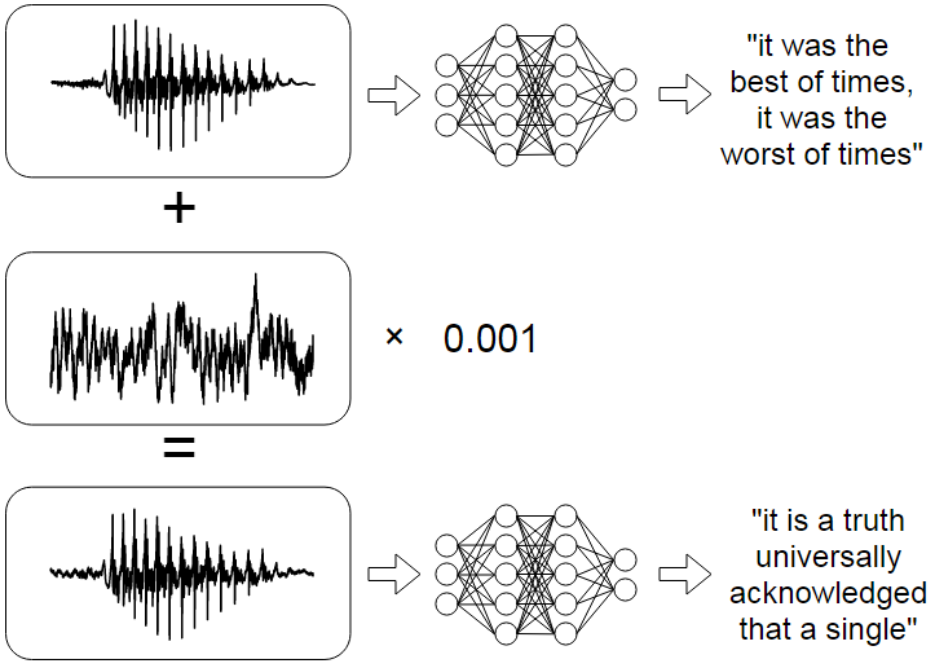
Deep Learning - Current Limitations
Adversarial Examples

Adversarial Patch is one of the latest research results from Google [Brown17]



Deep Learning - Current Limitations
Adversarial Examples

Adversarial examples is a class of samples that are maliciously designed to attack machine learning models



"without the dataset the article is useless"

"okay google browse to evil dot com"

Samples from : https://nicholas.carlini.com/code/audio_adversarial_examples/

Adversarial Examples

Reason?

- Non-linearity , uneven distribution, overfitting
- Linearity
  - *Goodfellow et al. "Explaining and Harnessing Adversarial Examples"*
  - If the model has a large enough input resolution
  - Example: a binary classifier $Score = W^T X$, add a small noise n, then $Score' = W^T X + n$

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1

Adversarial Examples

- Example: a binary classifier $Score = W^T X$, add a small noise n,
  Then $Score' = W^T X + n$
- Let $n = 0.5$
- $P(y = 1|x; w) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$, where $b = 0$

x	2	-1	3	-2	2	2	1	-4	5	1	$\sum$	$P(y = 1\|x; w)$
w	-1	-1	1	-1	1	-1	1	1	-1	1		
wx	-2	1	3	2	2	-2	1	-4	-5	1	-3	**0.04743**
$wx + n$	-1.5	1.5	3.5	2.5	2.5	-1.5	1.5	-3.5	-4.5	1.5	2	**0.8808**

- We improved the class 1 probability from **4.7%** to **88%**

Adversarial Examples

Adversarial samples

- Almost impossible to distinguish the difference between real and adversarial examples with naked eyes
- Will lead to wrong judgment of the model, but not the human
- Not specific images
- Not specific deep neural networks (discriminative ML models)
- Attacks and defenses of adversarial samples is active research field

Adversarial Examples

Tutorials:

- Tricking Neural Networks: Create your own Adversarial Examples link
- Adversarial Examples and Adversarial Training – YouTube link
- Adversarial Examples and their implications link

Paper:

- Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks[J]. arXiv preprint arXiv:1312.6199, 2013.
- Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world[J]. arXiv preprint arXiv:1607.02533, 2016.
- Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples[J]. arXiv preprint arXiv:1412.6572, 2014.

Challenge:

- NIPS 2017 Adversarial learning competition link