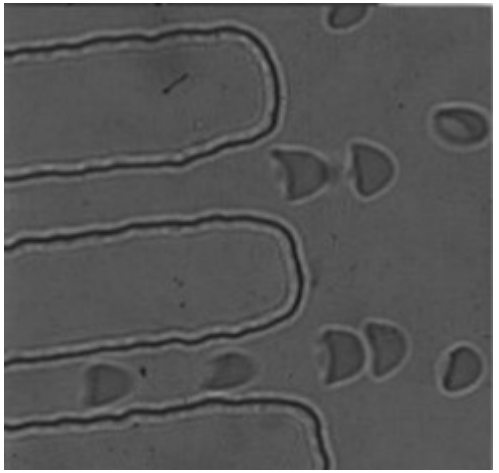
Week 11

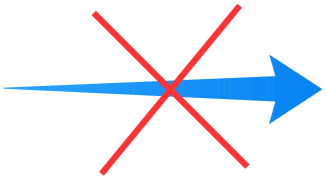
Deep learning limitations

Deep learning state of the art

Figure 1. *Four images showing the recognition, generative and potential misuse capabilities of deep learning. A) 3D pose prediction based on 2D images (Guler et al. 2018). B) Generated photos of non existing peoples faces (Karras et al. 2019). C) Example of super-resolution, the artificial increase of image resolution (Ledig et al. 2017). D) Screenshot of a video where the face of former president Obama (top) was controlled by an actor (below) (source: https://www.abc.net.au/news/2018-09-26/10308204)*

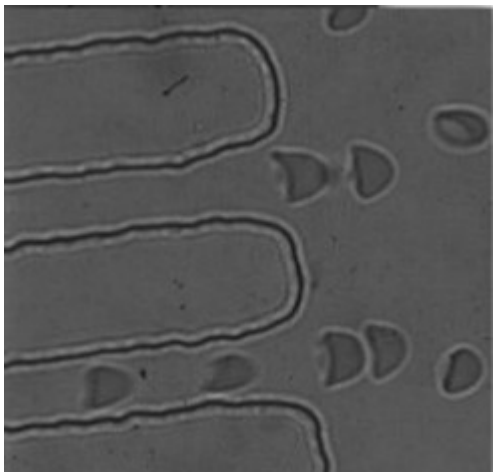
Can’t do very (for us) simple tasks



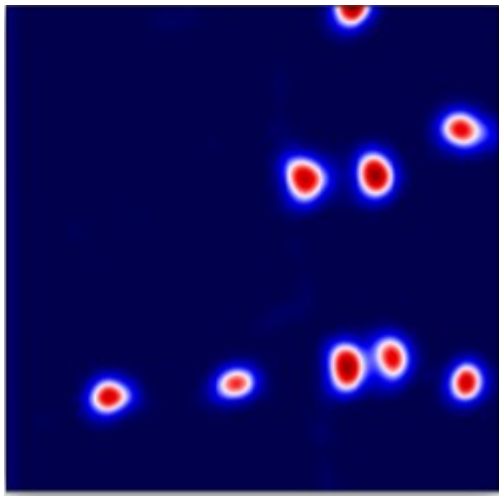


9

Can’t do very (for us) simple tasks



Deep learning

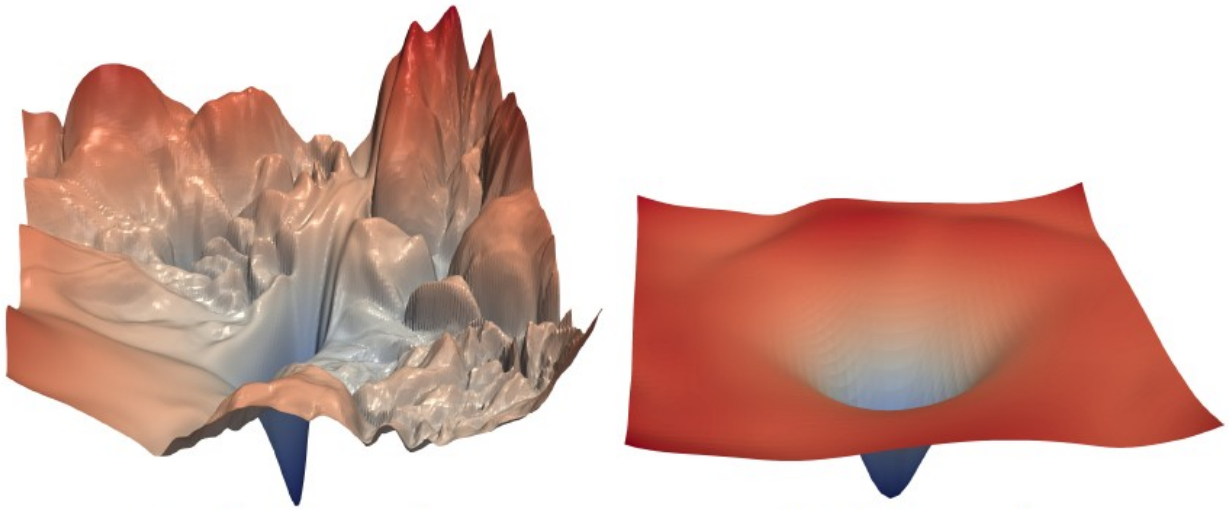


9

Many problems are already solved

BatchNorm/dropout → multiple layers

ResMet → More than 20 layers



(a) without skip connections

(b) with skip connections

1 obvious by now

Firstly, deep learning nearly always requires a large amount of annotated data!!!

1 obvious by now

Firstly, deep learning nearly always requires a large amount of annotated data!!!

Bypass it a bit with unsupervised learning.

1 obvious by now

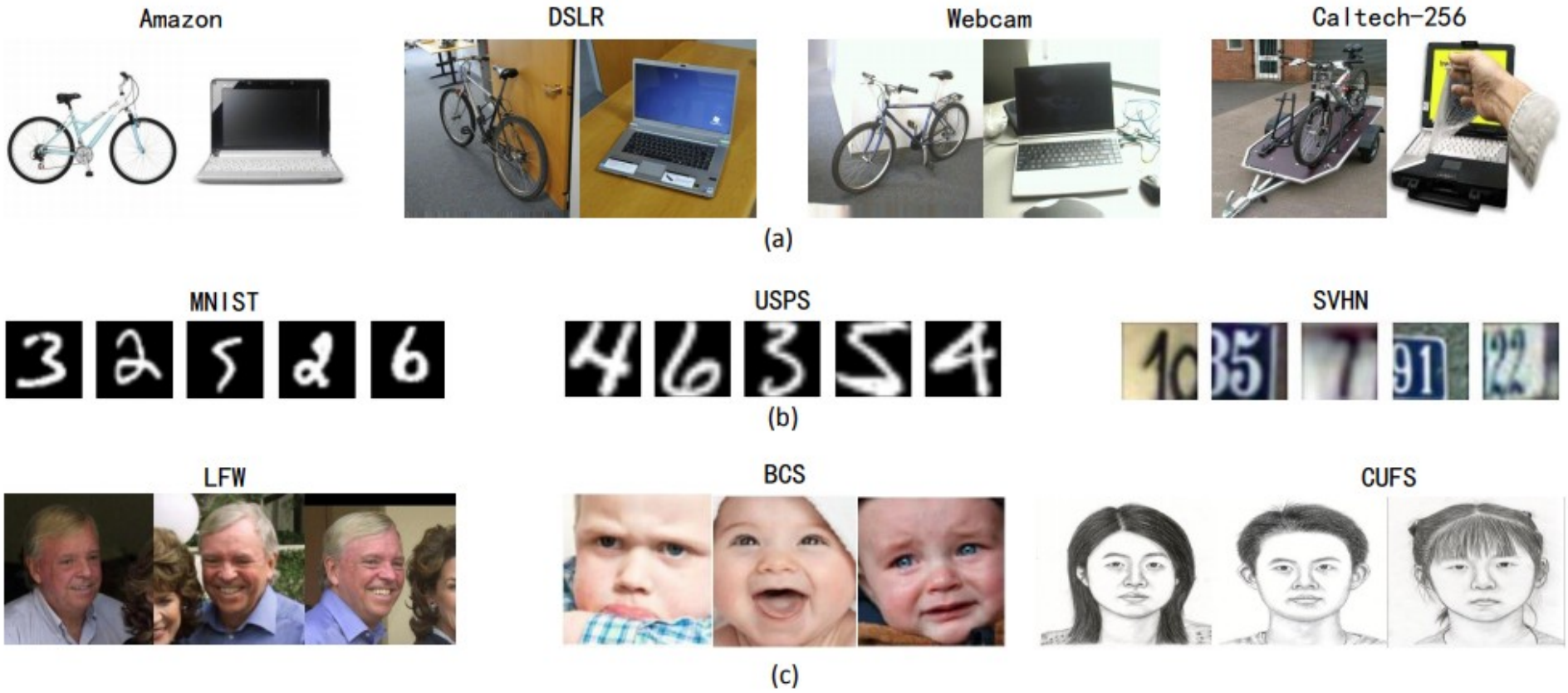
Firstly, deep learning nearly always requires a large amount of annotated data!!!

Bypass it a bit with unsupervised learning.

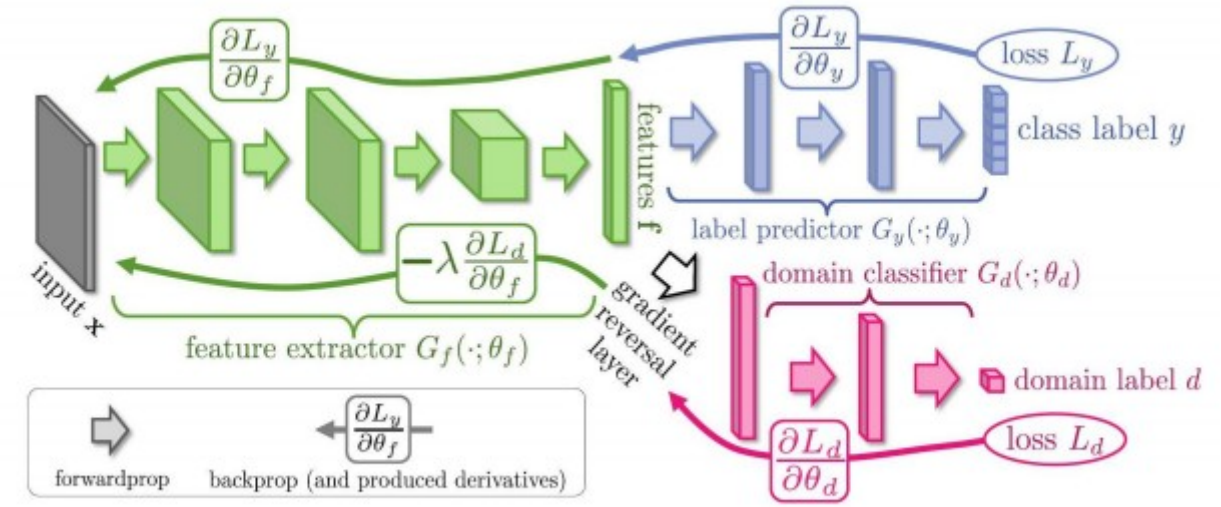
(but also with the use of synthetic data...)

Domain adaptation

Domain adaptation

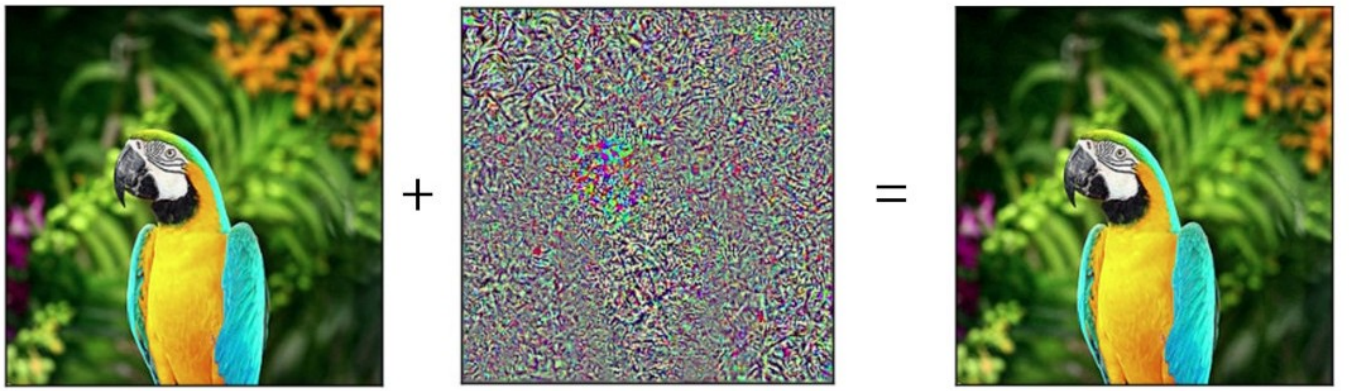


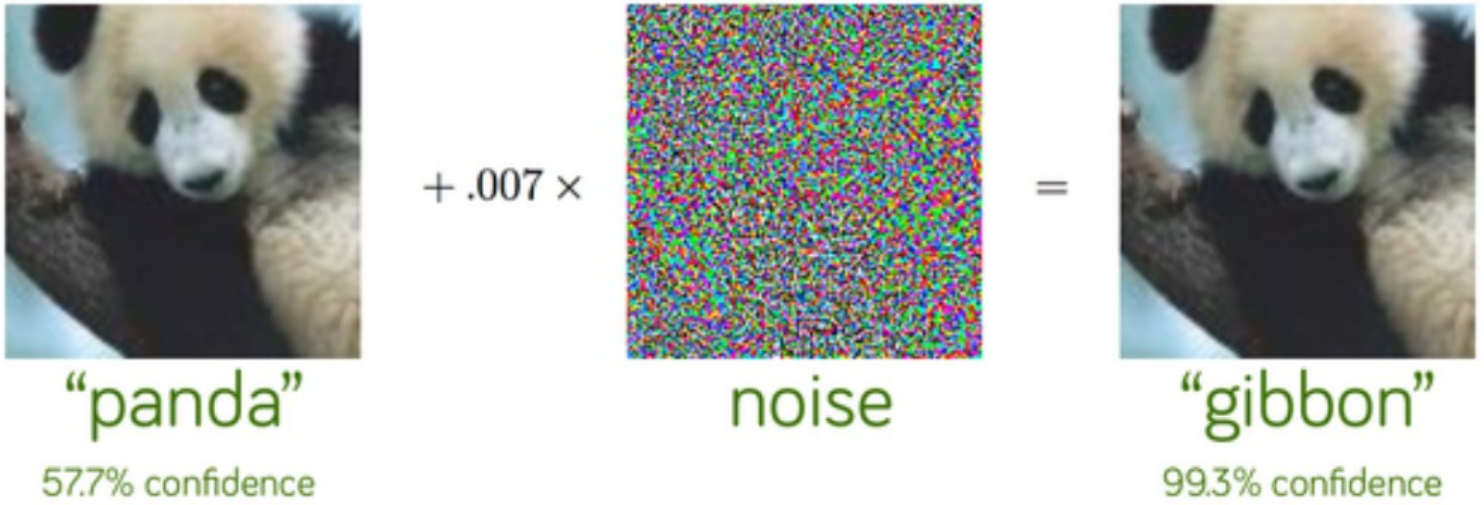
Solution


$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
$-\lambda\frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input x
gradient reversal layer
feature extractor $G_f(\cdot;\theta_f)$
domain label d
$\frac{\partial L_d}{\partial \theta_d}$
loss L_d
$\frac{\partial L_y}{\partial \theta_f}$
forwardprop
backprop (and produced derivatives)

Deep learning vs human perception

Deep learning vs human perception





Goodfellow et al. 2014

Deep learning vs human perception

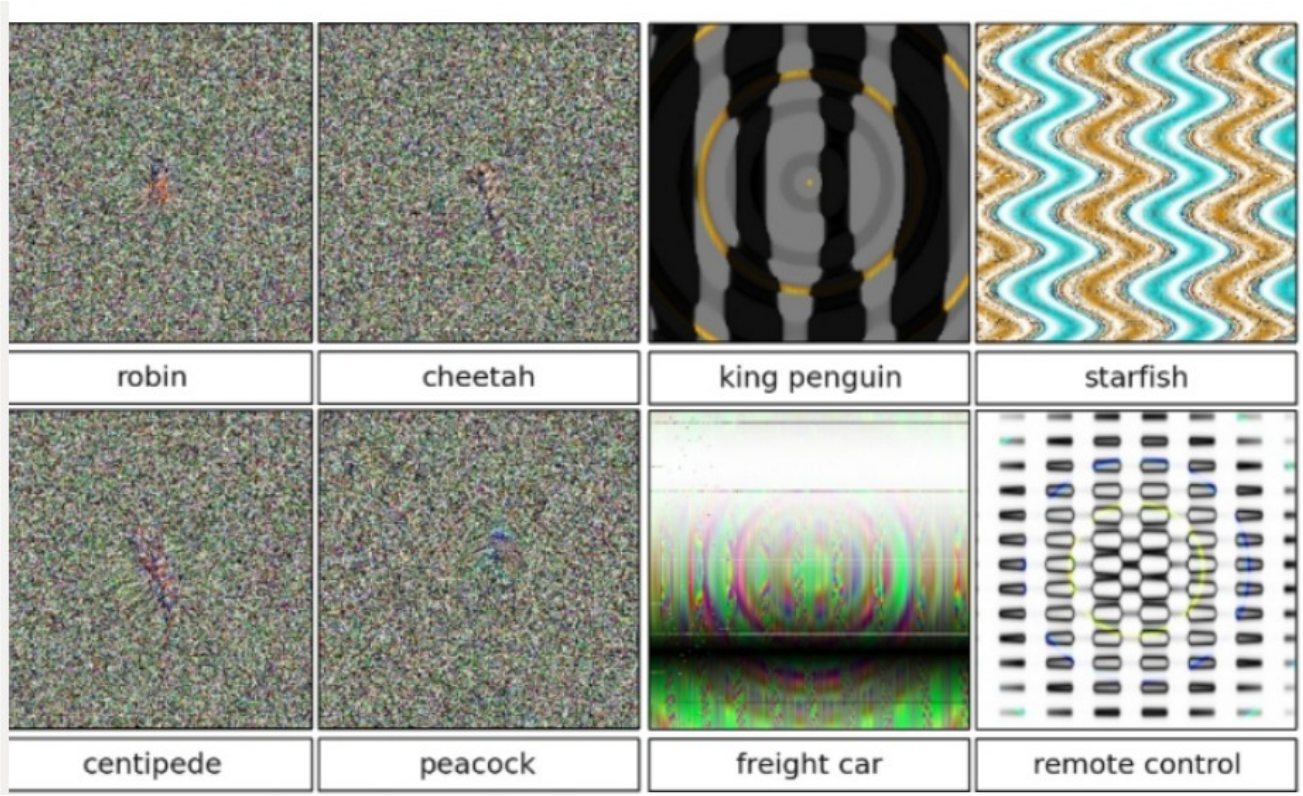


Figure 8. *"Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. " Source: Nguyen et al. 2015*

Even 1 single pixel

Figure 7. *"One-pixel attacks on photos. The modified pixels are highlighted with red circles. The original class labels are in black color while the target class labels and their corresponding confidence are given below."* *Source: Su et al. 2019.*

Deep learning vs human perception

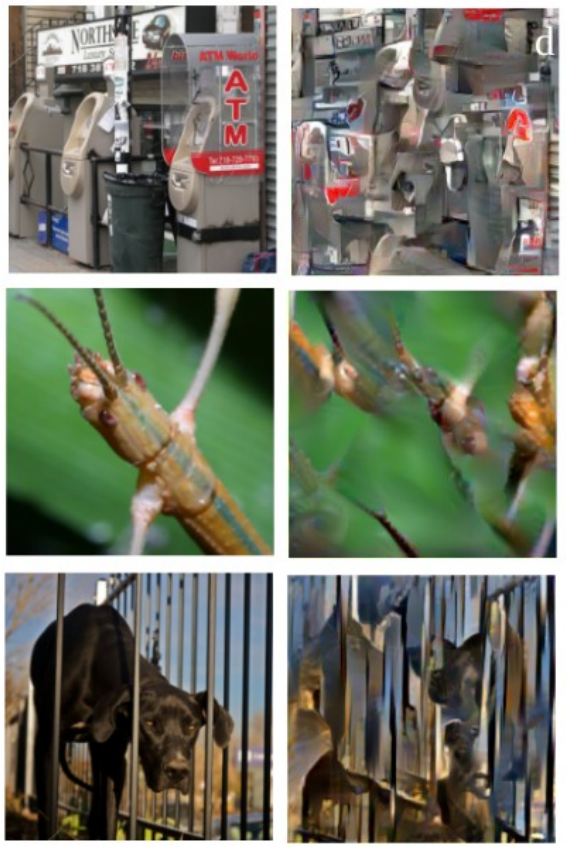


Figure 9. *Left, Original images. Right, mixed-up images. Trained deep learning models still reach high accuracy for the distorted images. Source: Brendel et al. (2019)*

Texture and not shape is the most dominant feature for object recognition by neural networks

Solution

Figure 11. *From left to right, a cat with elephant texture, a car with clock texture and a bear with a bottle texture. By changing the texture neural networks are forced to learn the shape of objects. Source: Geirhos et al. 2018*

Dataset bias

Bias example 1

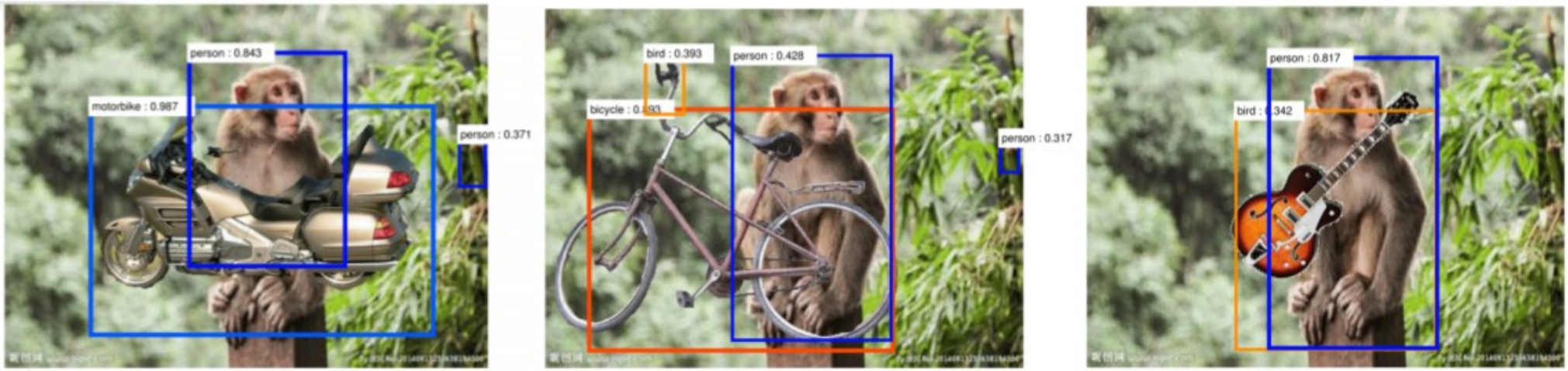
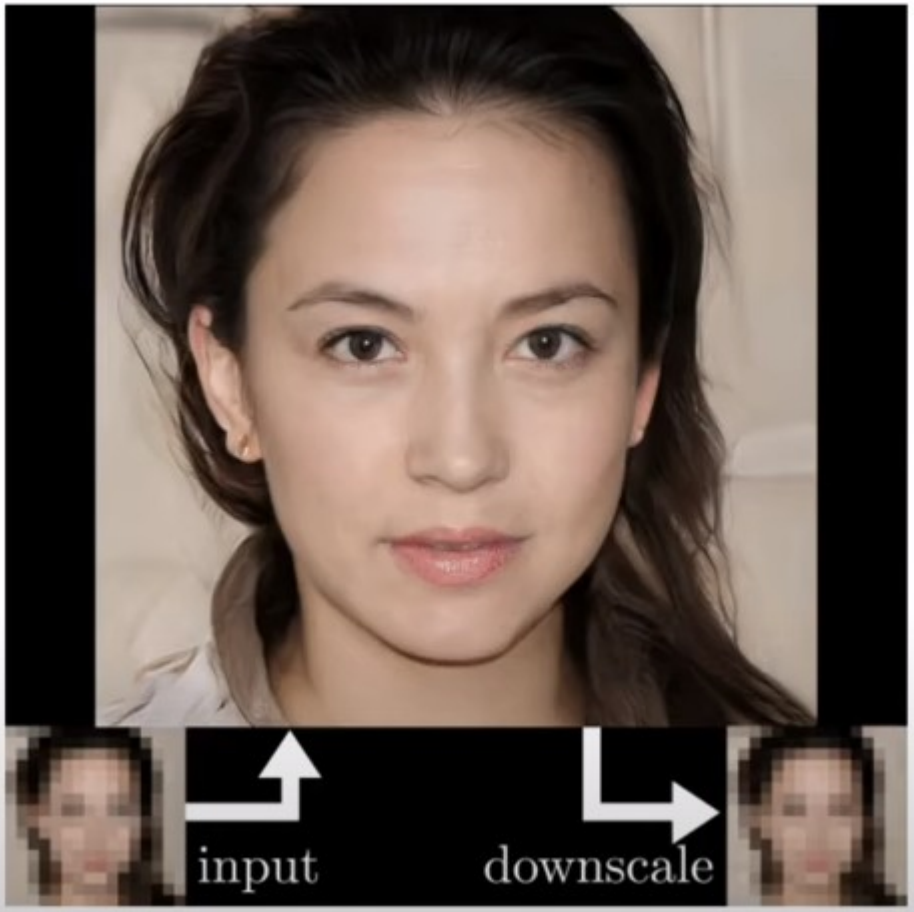


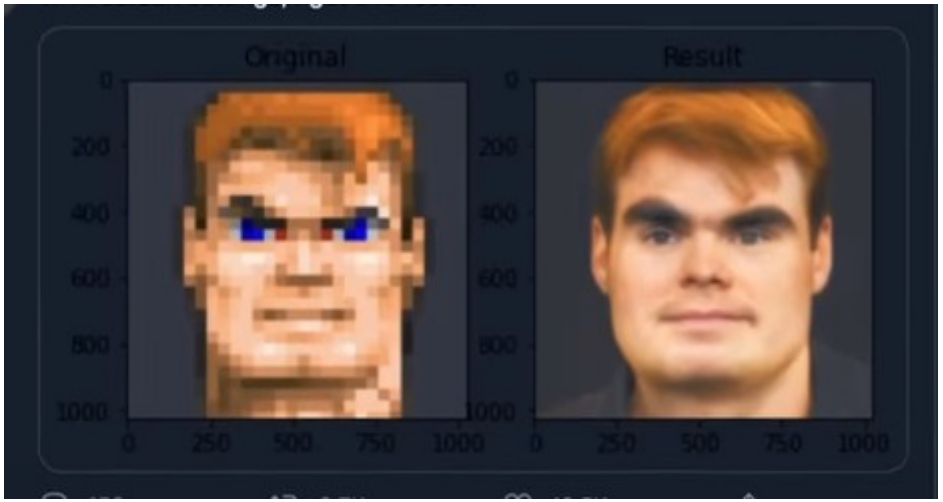
Figure 10. *"Adding occluders causes deep network to fail. We refer such examples as adversarial context examples since the failures are caused by misusing context/occluder info. Left Panel: The occluding motorbike turns a monkey into a human. Center Panel: The occluding bicycle turns a monkey into a human and the jungle turns the bicycle handle into a bird. Right Panel: The occluding guitar turns the monkey into a human and the jungle turns the guitar into a bird". Source: Wang et al. 2017*

New (last week)

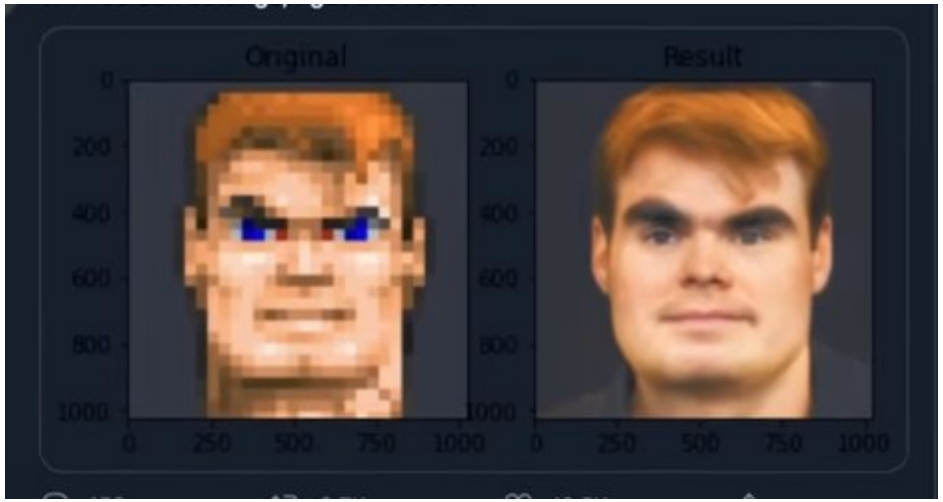
Low resolution → predict high resolution image

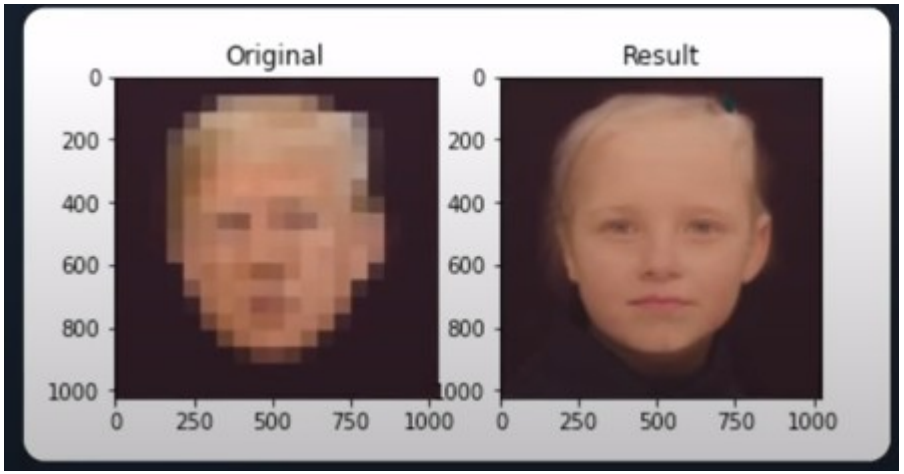


Funny results

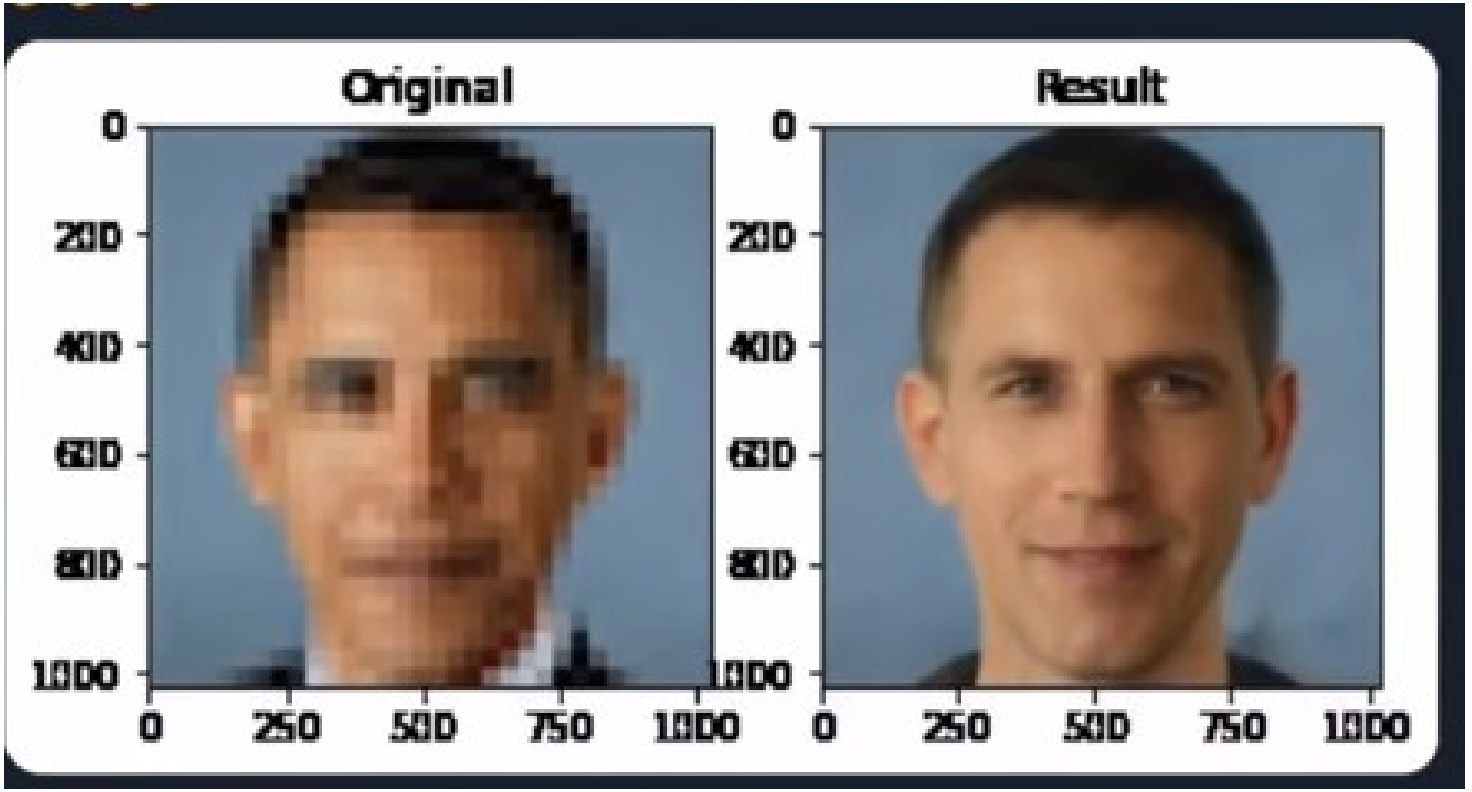


Funny results





For some people, not funny example



Input representation

Raw-input

Normalized input

One-hot

Embeddings

Input representation

Deep learning learn own features

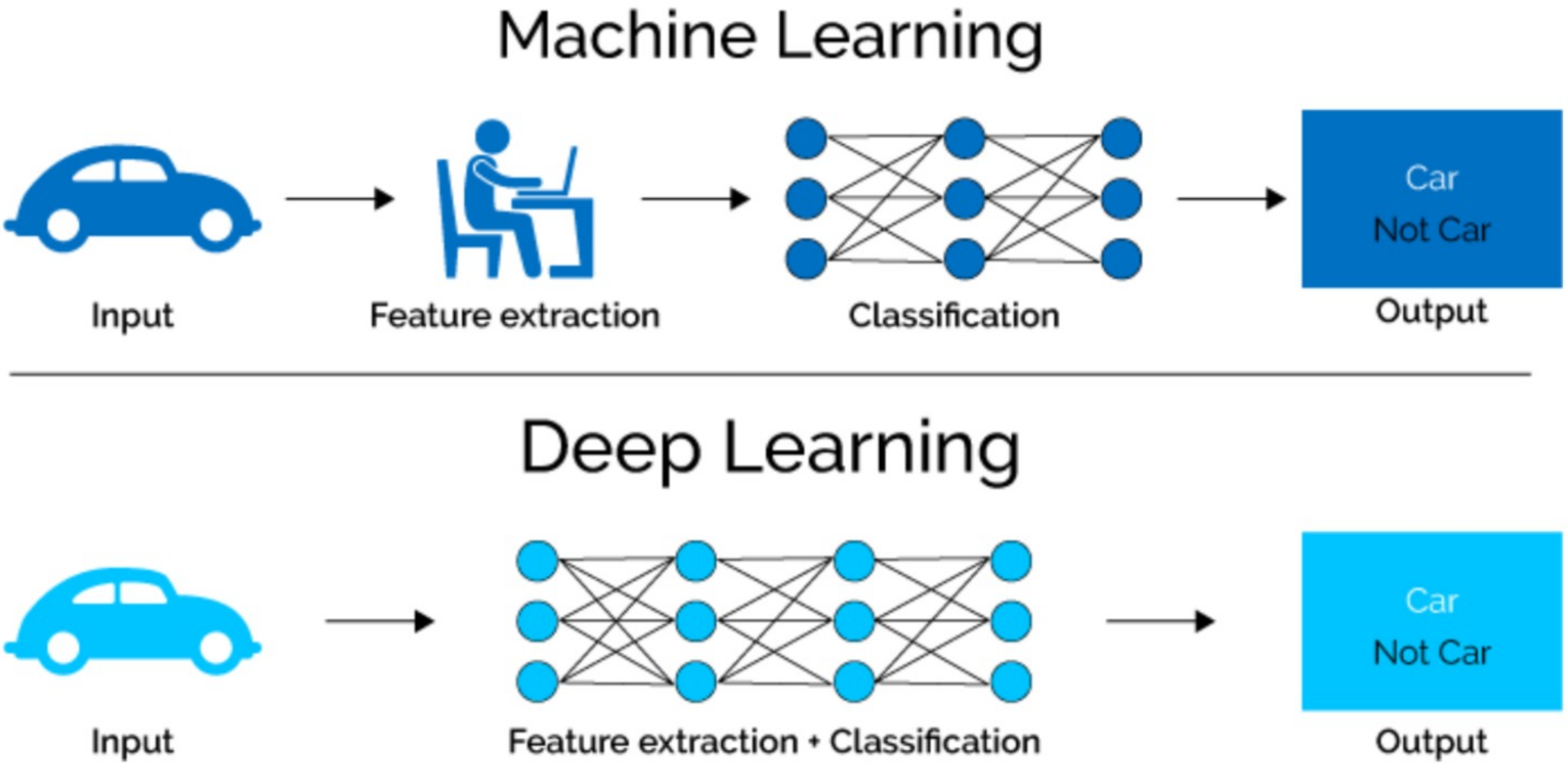
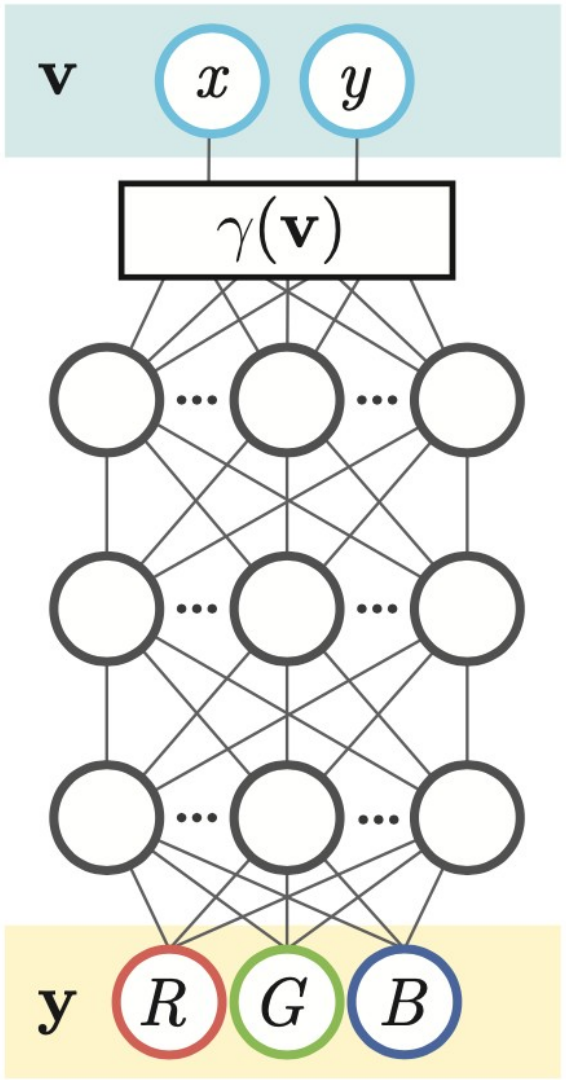


Figure 1: Machine Learning VS Deep Learning

Neural networks have a "spectral bias" towards being smooth

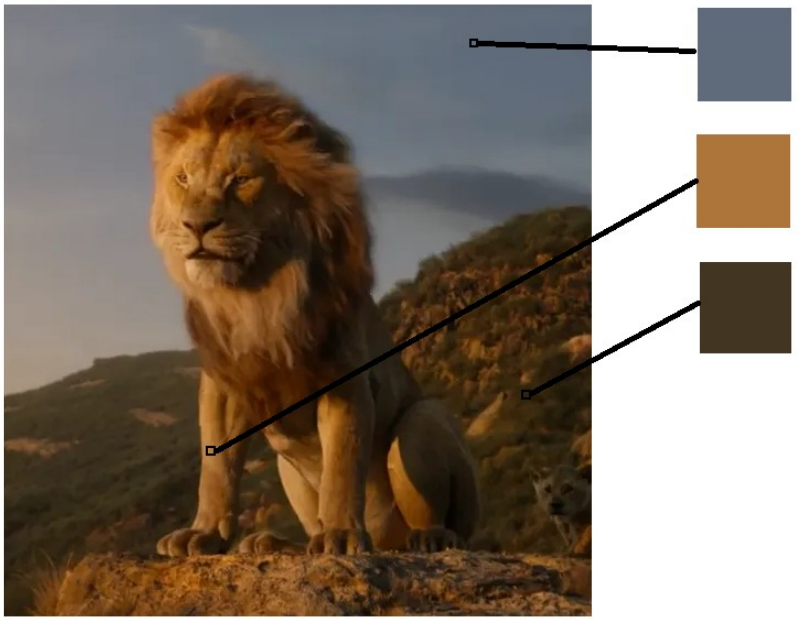
Neural networks have a "spectral bias" towards being smooth



Treat image as a ‘function’

Input: the x,y coordinates (normalize between 0-1)

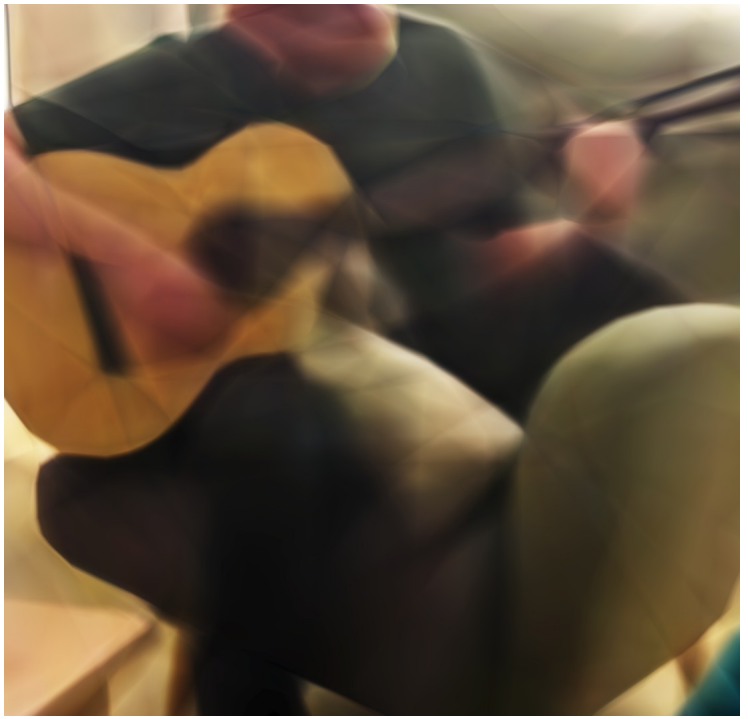
Output: the RGB value on that location



Example

GT

output



Example

Many many many neurons and parameters + weeks of training:

GT

output

paper

Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

Matthew Tancik[1*] **Pratul P. Srinivasan**[1,2*] **Ben Mildenhall**[1*] **Sara Fridovich-Keil**[1]

Nithin Raghavan[1] **Utkarsh Singhal**[1] **Ravi Ramamoorthi**[3] **Jonathan T. Barron**[2] **Ren Ng**[1]

[1]University of California, Berkeley [2]Google Research [3]University of California, San Diego

Abstract

We show that passing input points through a simple Fourier feature mapping enables a multilayer perceptron (MLP) to learn high-frequency functions in low-dimensional problem domains. These results shed light on recent advances in computer vision and graphics that achieve state-of-the-art results by using MLPs to represent complex 3D objects and scenes. Using tools from the neural tangent kernel (NTK) literature, we show that a standard MLP fails to learn high frequencies both in theory and in practice. To overcome this spectral bias, we use a Fourier feature mapping to transform the effective NTK into a stationary kernel with a tunable bandwidth. We suggest an approach for selecting problem-specific Fourier features that greatly improves the performance of MLPs for low-dimensional regression tasks relevant to the computer vision and graphics communities.

paper

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}}\mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}}\mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}}\mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}}\mathbf{v})\right]^{\mathrm{T}}. \quad (5)$$

Because $\cos(\alpha - \beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$, the kernel function induced by this mapping is:

$$k_\gamma(\mathbf{v}_1, \mathbf{v}_2) = \gamma(\mathbf{v}_1)^{\mathrm{T}}\gamma(\mathbf{v}_2) = \sum_{j=1}^{m} a_j^2 \cos\left(2\pi \mathbf{b}_j^{\mathrm{T}}(\mathbf{v}_1 - \mathbf{v}_2)\right) = h_\gamma(\mathbf{v}_1 - \mathbf{v}_2), \quad (6)$$

$$\text{where } h_\gamma(\mathbf{v}_\Delta) \triangleq \sum_{j=1}^{m} a_j^2 \cos(2\pi \mathbf{b}_j^{\mathrm{T}}\mathbf{v}_\Delta). \quad (7)$$

$$k_{\mathrm{NTK}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_{\theta \sim \mathcal{N}} \left\langle \frac{\partial f(\mathbf{x}_i;\theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}_j;\theta)}{\partial \theta} \right\rangle$$

$\mathcal{L} = \sum_{(\mathbf{x},y)\in\mathcal{D}} \ell(f_t^{\mathrm{lin}}(\mathbf{x}$ ... ...otes the vector training lab...

$$\ldots f_t^{\mathrm{lin}}(\mathbf{X}) \frac{1}{2} \left\|\mathbf{A}\left(f_t^{\mathrm{lin}}(\mathbf{X}) - \mathbf{y}\right)\right\|_2^2 \quad (15)$$

$$= \mathbf{A}^{\mathrm{T}}\mathbf{A}\left(f_t^{\mathrm{lin}}(\mathbf{X}) - \mathbf{y}\right). \quad (16)$$

Substituting this into the gradient flow dynamics of Eqn. 14 gives us:

$$\dot{f}_t^{\mathrm{lin}}(\mathbf{x}) = -\eta\hat{\Theta}_0(\mathbf{x}, \mathbf{X})\mathbf{A}^{\mathrm{T}}\mathbf{A}\left(f_t^{\mathrm{lin}}(\mathbf{X}) - \mathbf{y}\right), \quad (17)$$

with corresponding solution:

$$f_t^{\mathrm{lin}}(\mathbf{X}) = \left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y} + e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t} f_0(\mathbf{X}). \quad (18)$$

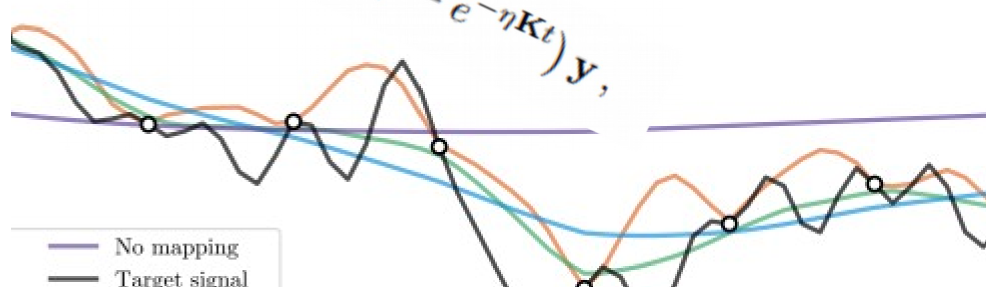
Finally, again following [20], we can decompose $f_t^{\mathrm{lin}}(\mathbf{x}) = \mu_t(\mathbf{x}) + \gamma_t(\mathbf{x})$ at any test point $\mathbf{x}$, where

$$\mu_t(\mathbf{x}) = \hat{\Theta}_0(\mathbf{x}, \mathbf{X})\hat{\Theta}_0^{-1}\left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)\mathbf{y}, \quad (19)$$

$$\gamma_t(\mathbf{x}) = f_0(\mathbf{x}) - \hat{\Theta}_0(\mathbf{x}, \mathbf{X})\hat{\Theta}_0^{-1}\left(\mathbf{I} - e^{-\eta\hat{\Theta}_0\mathbf{A}^{\mathrm{T}}\mathbf{A}t}\right)f_0(\mathbf{X}). \quad (20)$$

Assuming our initialization is small, *i.e.*, $f_0(\mathbf{x}) \approx 0\ \forall \mathbf{x}$, we can write our approximate linearized network output as:

$$\hat{\mathbf{y}}^{(t)} \approx \mathbf{K}_{test}\mathbf{K}^{-1}\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y},$$



In our previou... is fixed throu...

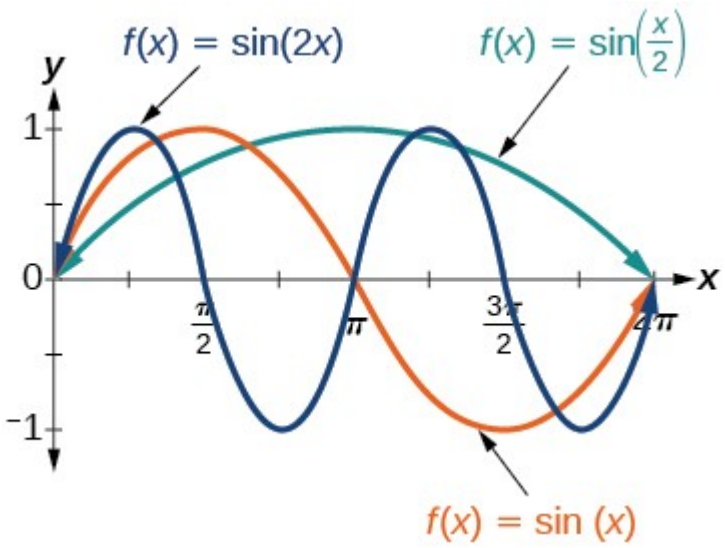
must be PSD, we can take its eigendecomposition $\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{Q}$ is orthogonal and $\mathbf{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues $\lambda_i \geq 0$ of $\mathbf{K}$. Then, since $e^{-\eta\mathbf{K}t} = \mathbf{Q}e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}$:

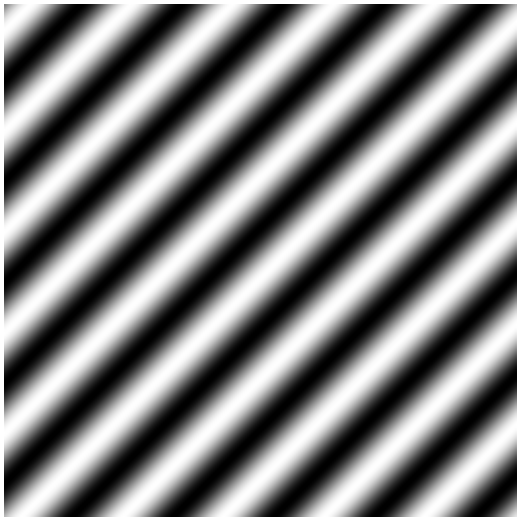
$$\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}_{\mathrm{train}}^{(t)} - \mathbf{y}) \approx \mathbf{Q}^{\mathrm{T}}\left(\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} - \mathbf{y}\right) = -e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}\mathbf{y}. \quad (4)$$

This is nearly

Paper conlcusion

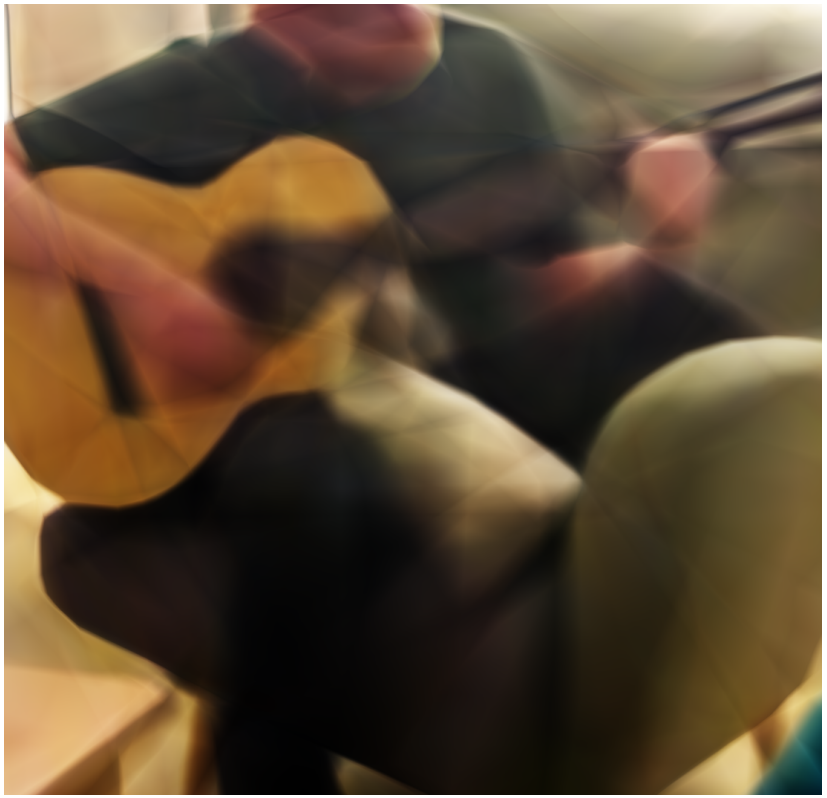
First map the **X** and **Y** coordinates to many Sine functions with different period





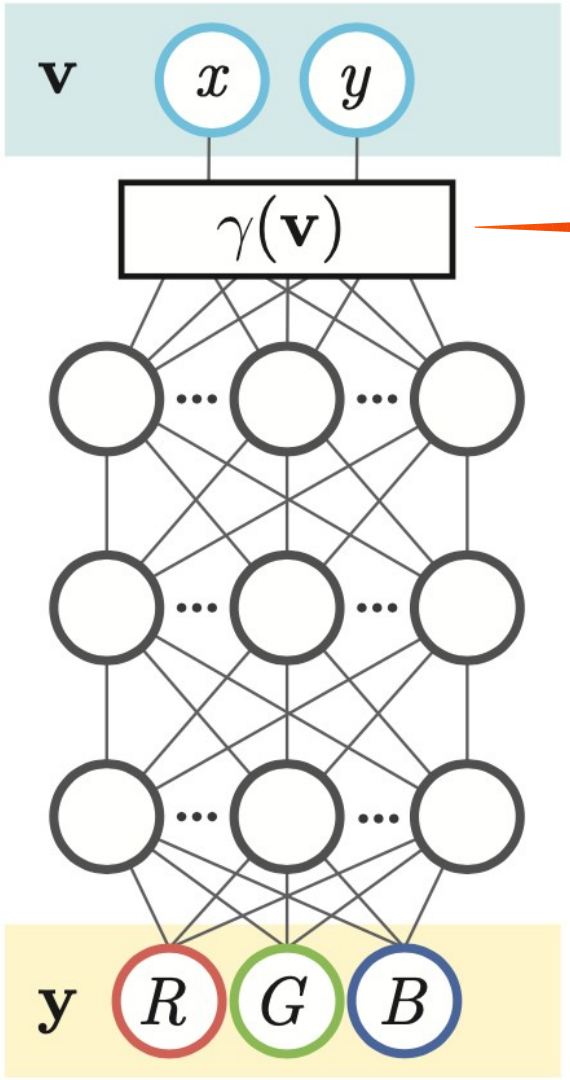
Works, simple MLP with 2 hidden layers

X, Y



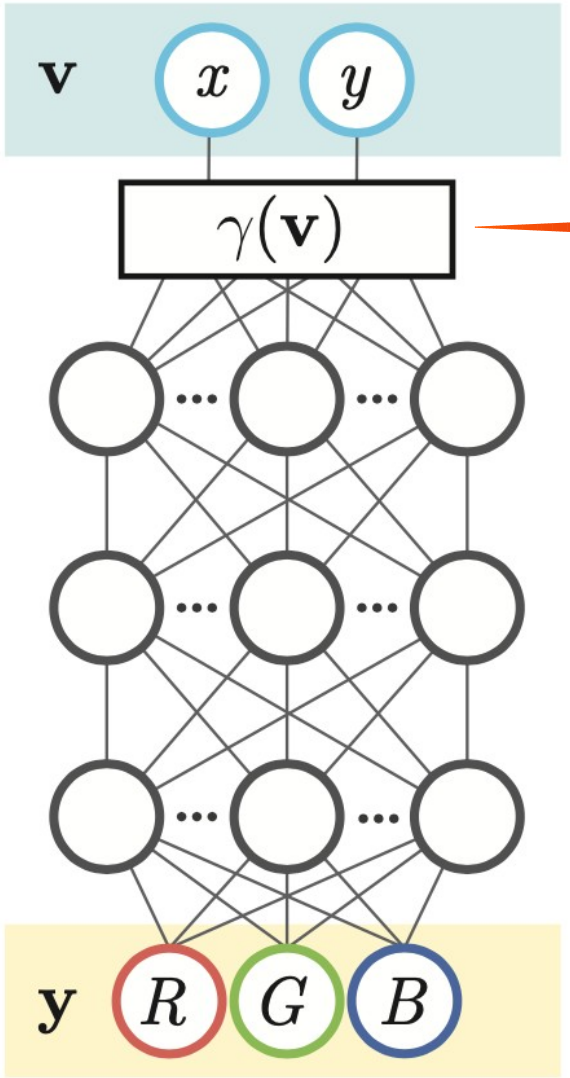
Fourier mapping

Paper summerized




```python
class Fourier(nn.Module):
    def __init__(self, inp, out, scale=10):
        super(Fourier, self).__init__()
        self.b = torch.randn(inp, out)*6.28*scale
    def forward(self, v):
        return torch.sin(torch.matmul(v, self.b))
```

Paper summerized

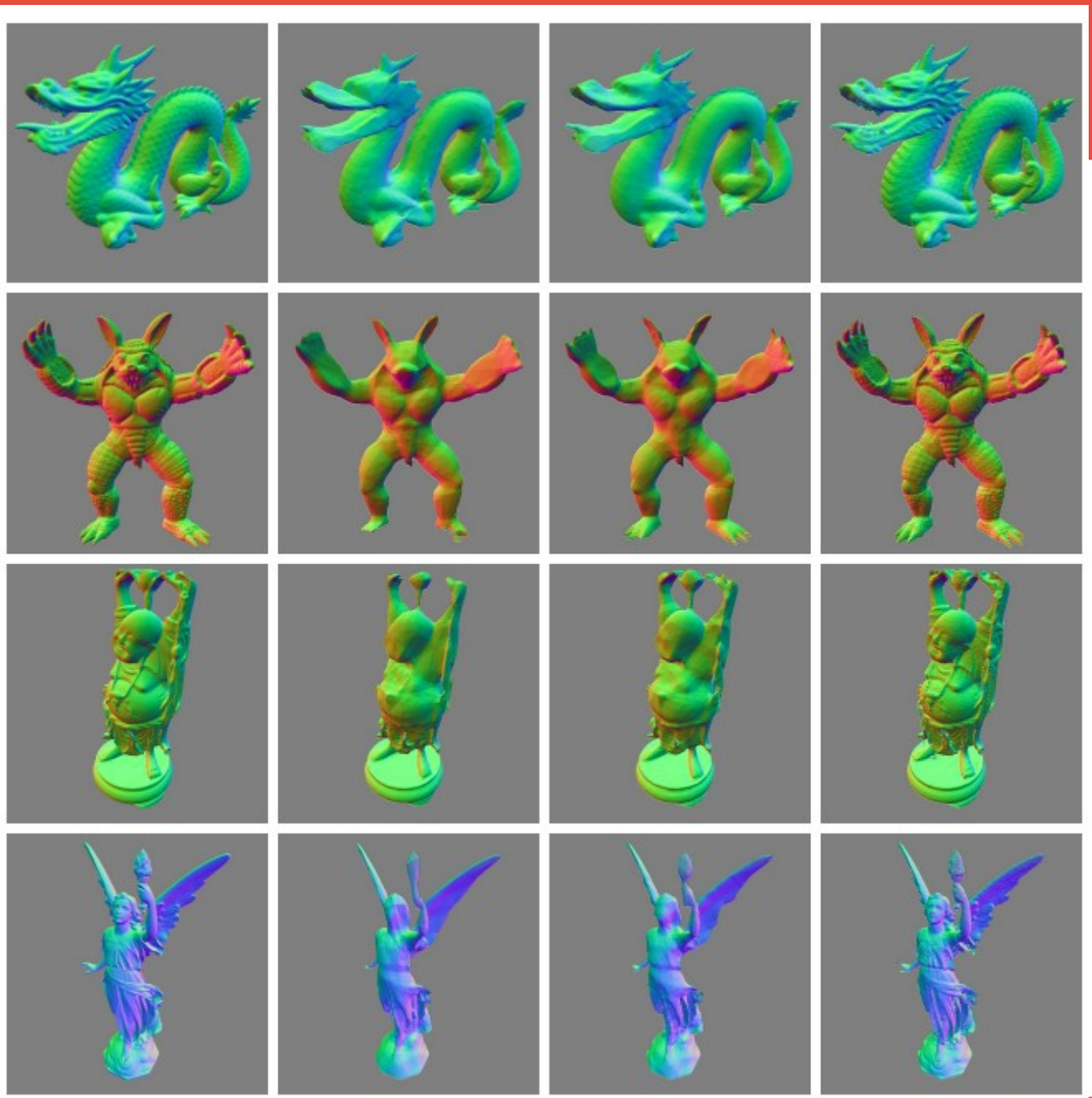



```python
class Fourier(nn.Module):
    def __init__(self, inp, out, scale=10):
        super(Fourier, self).__init__()
        self.b = torch.randn(inp, out)*6.28*scale
    def forward(self, v):
        return torch.sin(torch.matmul(v, self.b))
```

jnbrrn 1 point · 2 days ago

Glad we could help! And thanks for the pytorch implementation, very cool to see that the whole thing fits into a reddit comment :)

3D



(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc.

Loss funtion

Deep learning will ALWAYS try to cheat to optimize the loss!

https://www.youtube.com/watch?v=nKJlF-olKmg