CS5489
Lecture 12.2: Neural Networks and Deep Learning Part VIII: Advanced Topics Cont.

Kede Ma

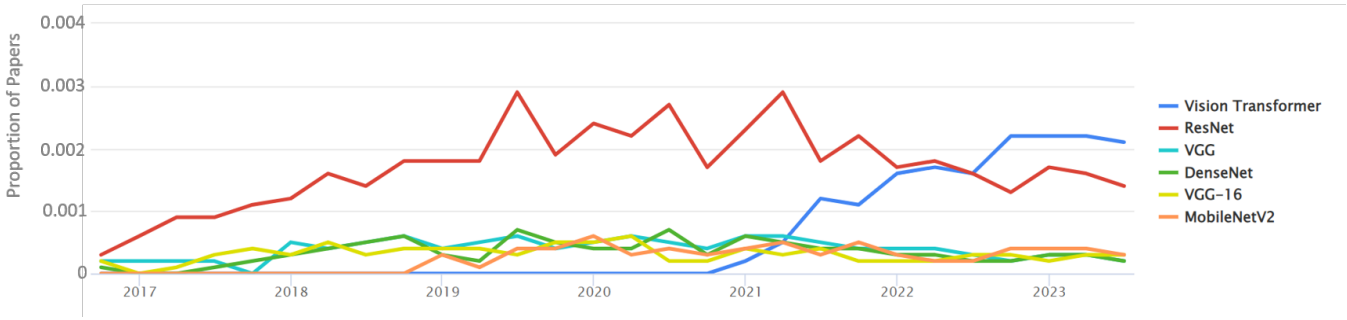
City University of Hong Kong (Dongguan)

Slide template by courtesy of Benjamin M. Marlin

Outline

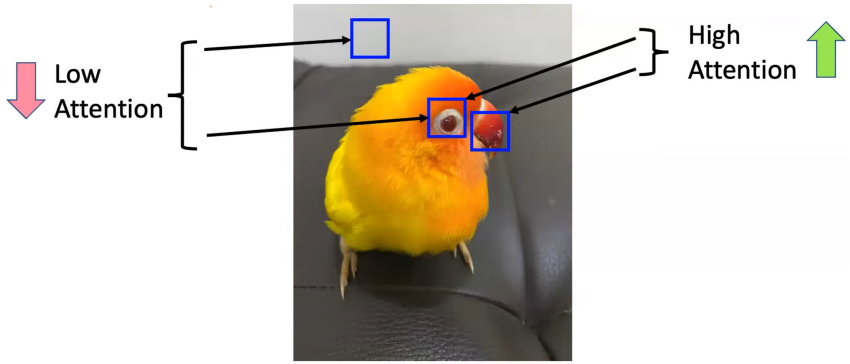
Background

- Transformers represent a family of very successful neural network architectures for natural language processing, and have penetrated the field of computer vision in the past five years
  - Vision Transformers (ViTs)



Alexey Dosovitskiy et al., "An Image is Worth 16 $\times$ 16 Words: Transformers for Image Recognition at Scale," *International Conference on Learning Representations*, 2021.

Visual Attention



- In human visual attention, we can quickly process which areas belong to the bird and which belong to the background
- Attention between any two patches that correspond to different parts of the bird is typically high
- Attention between a background patch and a bird patch is typically low
  - If it happens to be high, we encounter a *spurious* correlation (e.g., a sea bird against the sky/sea background)

An Overview of ViT

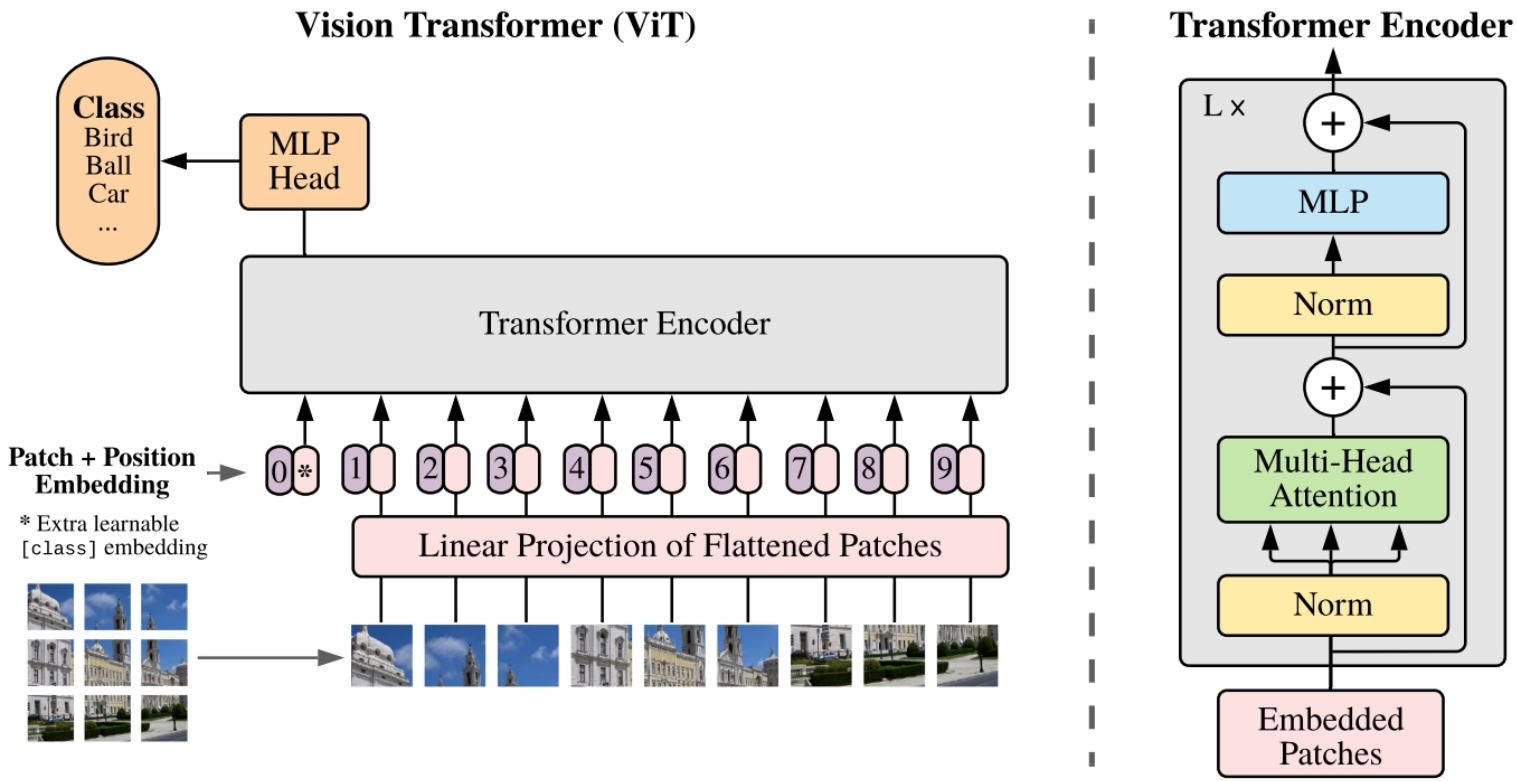
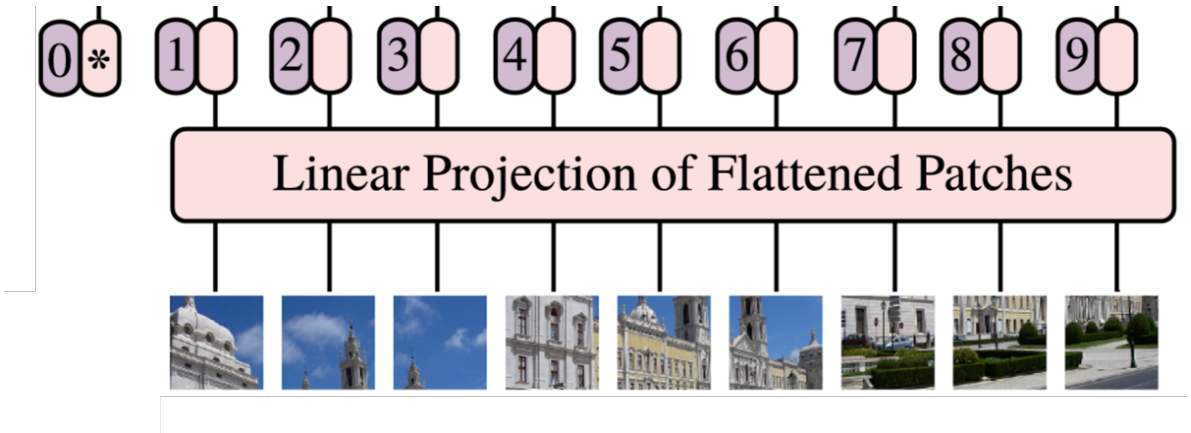


Image to Patches (im2col in MATLAB)

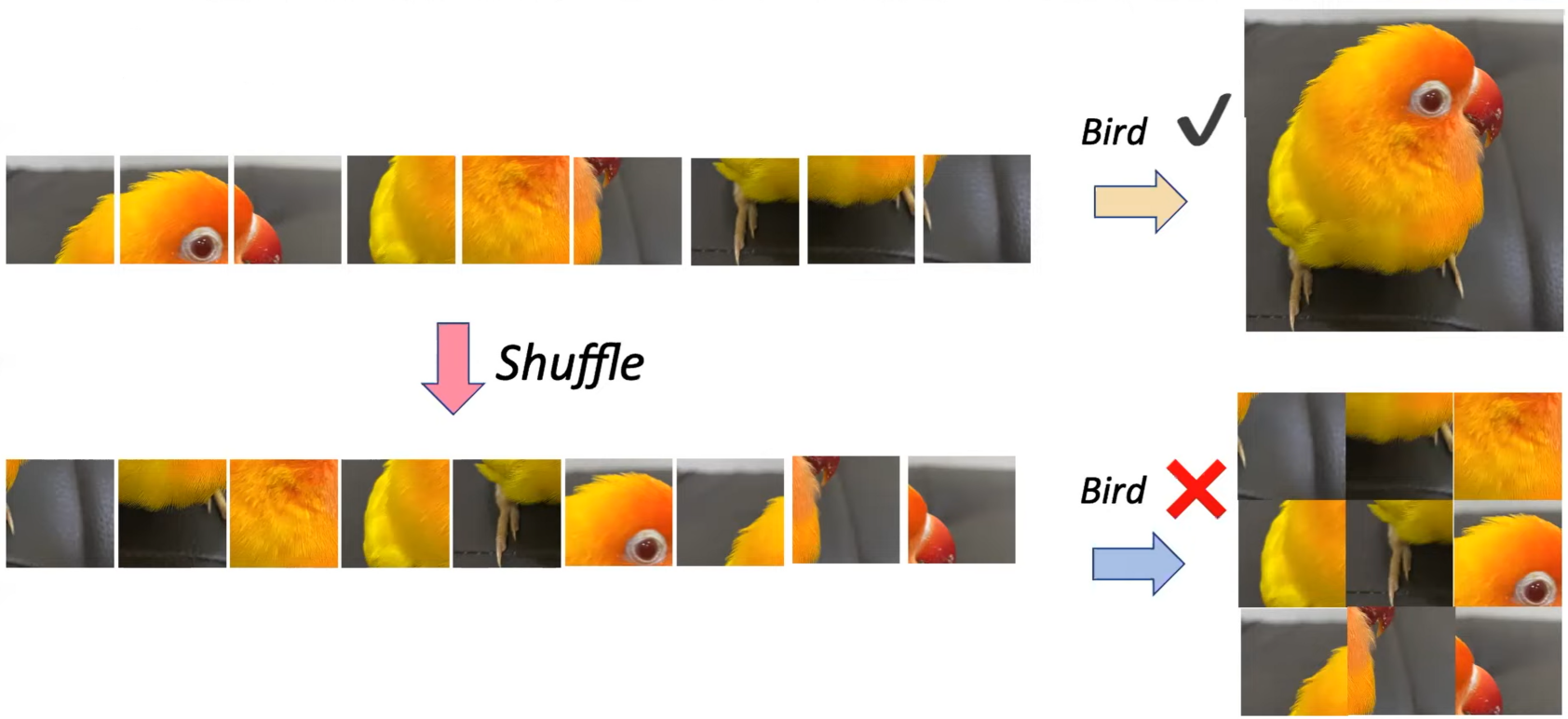
- Reshape an image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened patches $\mathbf{p} \in \mathbb{R}^{N \times (P^2 C)}$
  - (H, W) is the resolution of the original image
  - C is the number of channels
  - (P, P) is the resolution of each image patch
  - $N = HW/P^2$ is the resulting number of patches
    - This indicates that patches are non-overlapping

Linear Projection

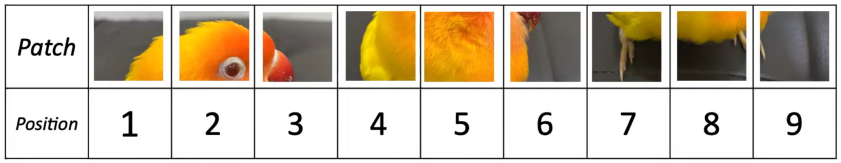


- ViT uses a constant latent vector size of D through all of its layers, so we flatten the patches and map them to D dimensions using a learnable linear projection
  - $\mathbf{z}_{0,i} = \mathbf{p}_i\mathbf{E}$, where $\mathbf{E} \in \mathbb{R}^{(P^2C)\times D}$
    - 0 indicates that it is the input to the first layer of ViT

Position Embedding



Position Embedding



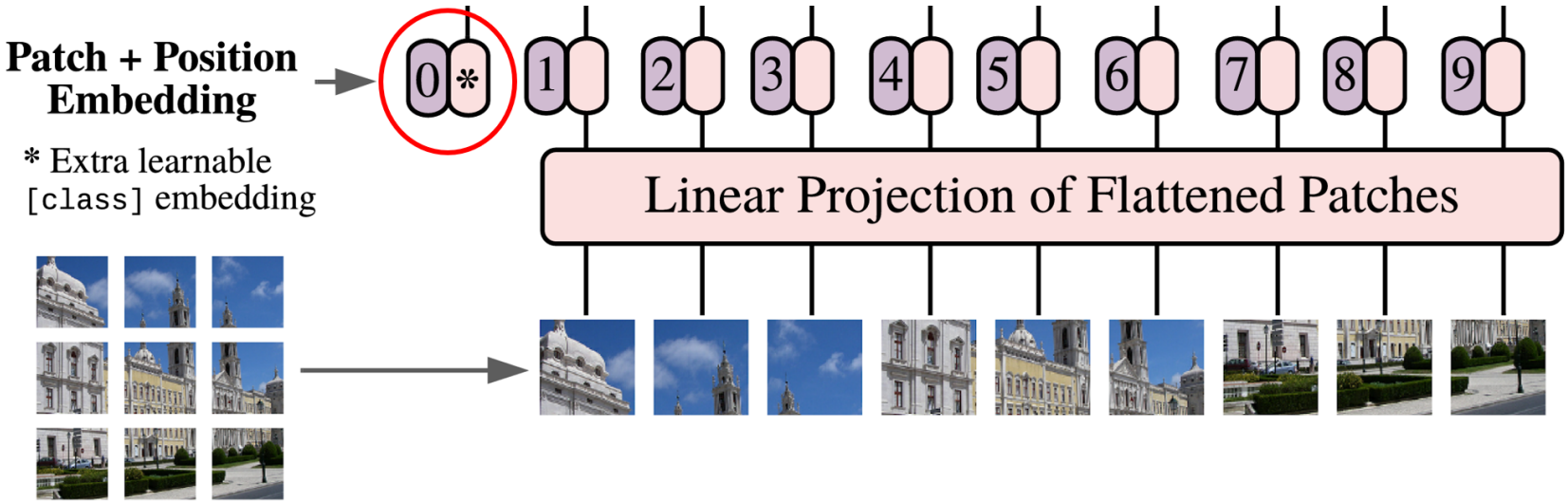
- Sinusoidal function

$$\mathrm{PE}_{(\mathrm{pos},2j)} = \sin\left(\mathrm{pos}/10000^{2j/D}\right)$$

$$\mathrm{PE}_{(\mathrm{pos},2j+1)} = \cos\left(\mathrm{pos}/10000^{2j/D}\right)$$

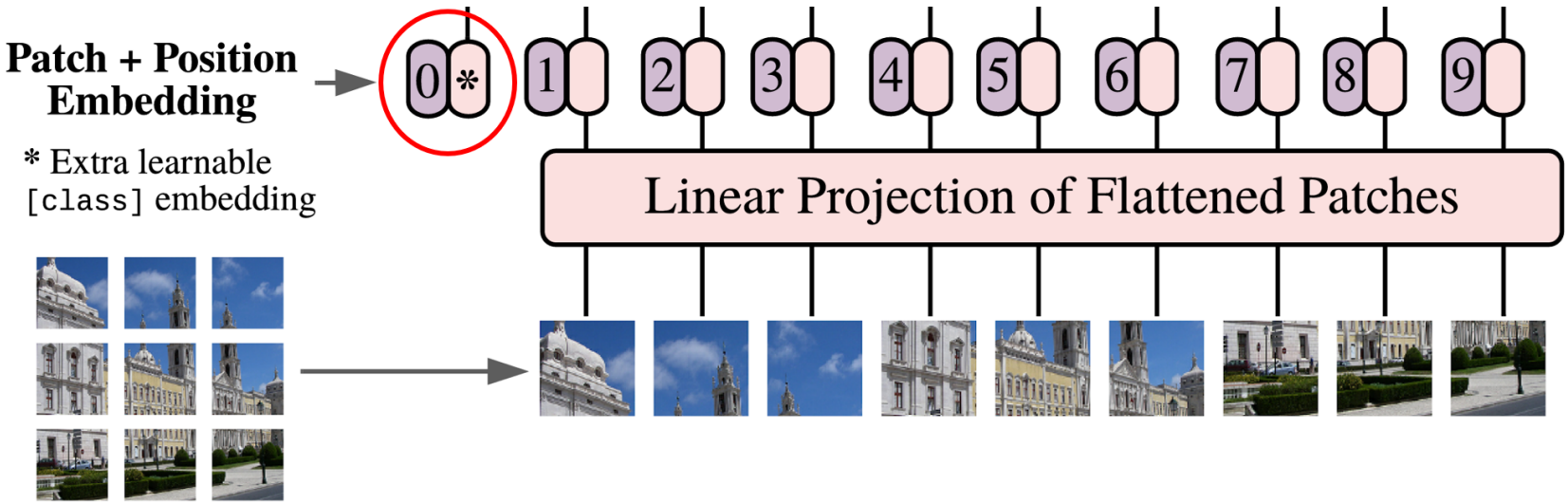
  - *pos* is the position and *j* is the index of the feature dimension
- Learnable position embeddings (most common in ViT)
  - Can be absolute or relative

Learnable Class Embedding



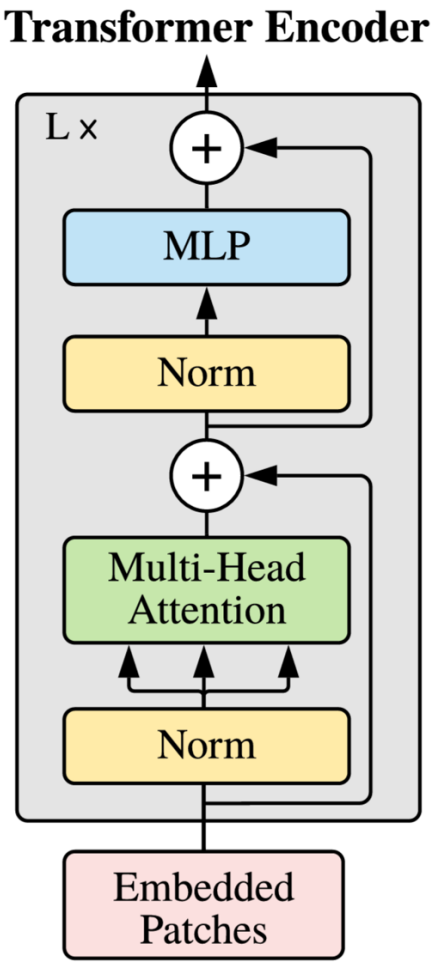
- We prepend a learnable embedding to the sequence of embedded patches ($\mathbf{z}_{0,0} = \mathbf{p}_{\text{class}}$), which will be used for image classification
  - We may as well discard this `[class]` embedding, and directly work with the patch embeddings for classification

Patch + Position Embedding



- $\mathbf{z}_0 = [\mathbf{p}_{\text{class}}; \mathbf{p}_1\mathbf{E}; \mathbf{p}_2\mathbf{E}; \ldots, \mathbf{p}_N\mathbf{E}] + \mathbf{E}_{\text{pos}}$
  - $\mathbf{E} \in \mathbb{R}^{(P^2C)\times D}$, $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$

Transformer Encoder



Transformer Encoder

- Scaled dot-product attention
  - Given an input sequence $\mathbf{z}$, project to the sets of queries, keys, and values using linear transforms

$$\mathbf{Q} = \mathbf{W}^Q\mathbf{z}, \mathbf{K} = \mathbf{W}^K\mathbf{z}, \mathbf{V} = \mathbf{W}^V\mathbf{z}$$

    where $\mathbf{W}^Q \in \mathbb{R}^{(N+1)\times(N+1)}$, $\mathbf{W}^K \in \mathbb{R}^{(N+1)\times(N+1)}$, and $\mathbf{W}^V \in \mathbb{R}^{(N+1)\times(N+1)}$

  - The dot-product attention is calculated as follows

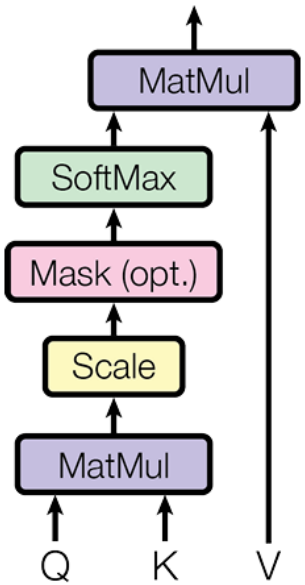
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D}}\right)\mathbf{V}$$

    - The purpose of scaling by $\sqrt{D}$ is to prevent gradient vanishing

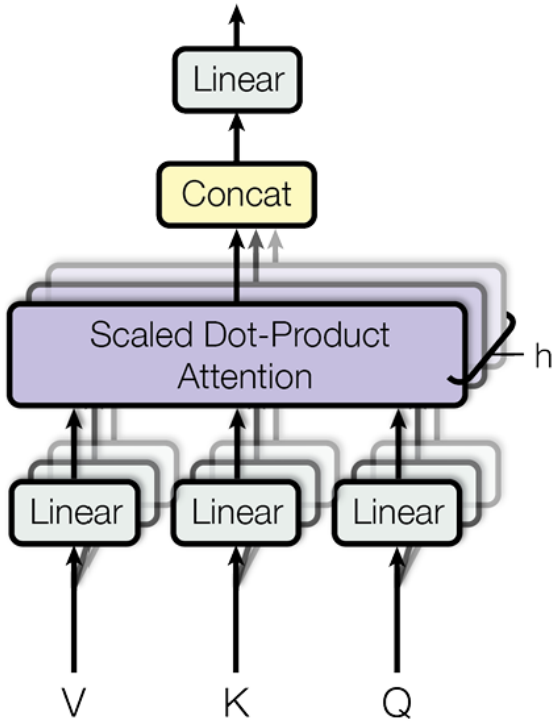
Transformer Encoder

- Multi-head attention

Scaled Dot-Product Attention



Multi-Head Attention



Transformer Encoder

- Multi-head attention
  - Given the query, key, and value matrices, we transform those into sub-queries, sub-keys, and sub-values, which are passed through the scaled dot-product attention independently. Afterward, we concatenate the heads
  - Mathematically,

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

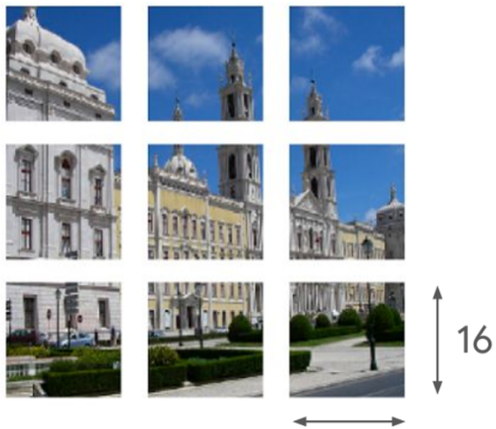
    where

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

    - $\mathbf{W}^O \in \mathbb{R}^{D \times D}$
    - h is the number of heads and $d_i = D/h$
    - $\mathbf{W}_i^Q \in \mathbb{R}^{D \times d_i}$, $\mathbf{W}_i^K \in \mathbb{R}^{D \times d_i}$, and $\mathbf{W}_i^V \in \mathbb{R}^{D \times d_i}$
  - Different learned representations can improve the performance

ViT Models

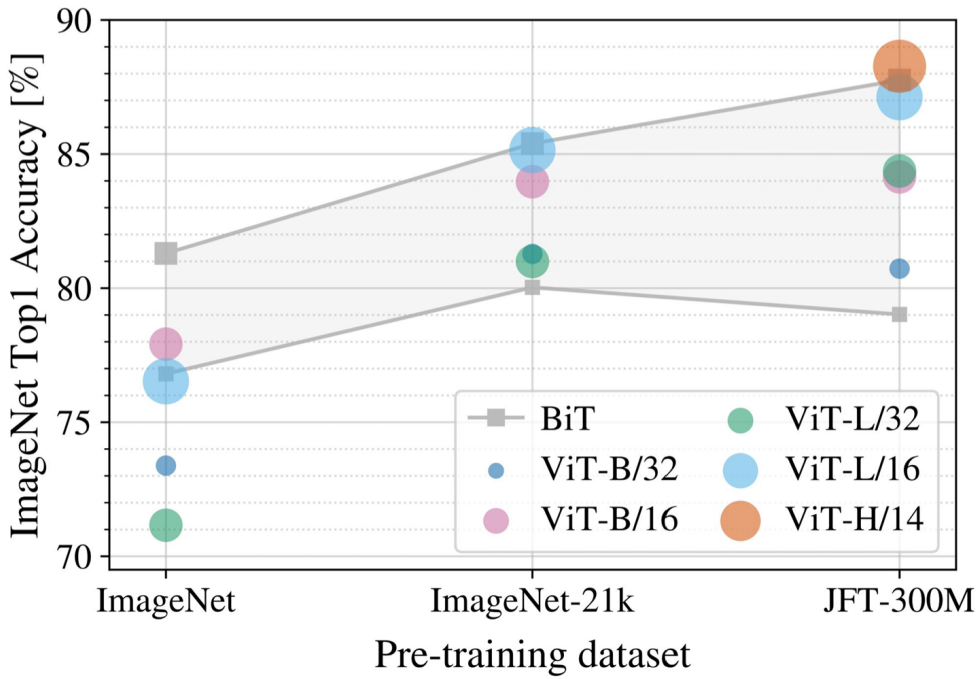
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M



Notations: ViT-L/16

ViTs are Effective at Scale

- ViTs have less inductive biases than CNNs
- So they need more data to train



Acceleration Techniques

- Standard ViTs compute global self-attention, which leads to **quadratic** complexity with respect to the number of patches
- Not suitable for vision tasks requiring an immense set of patches for dense prediction or to represent a high-resolution image
- Acceleration techniques:
  - Window attention

    Ze Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," *International Conference on Computer Vision*, 2021.
  - Linear attention

    Han Cai et al., "EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction," *International Conference on Computer Vision*, 2023.
  - Sparse attention

    Xuanyao Chen et al., "SparseViT: Revisiting Activation Sparsity for Efficient High-Resolution Vision Transformer," *IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

Outline

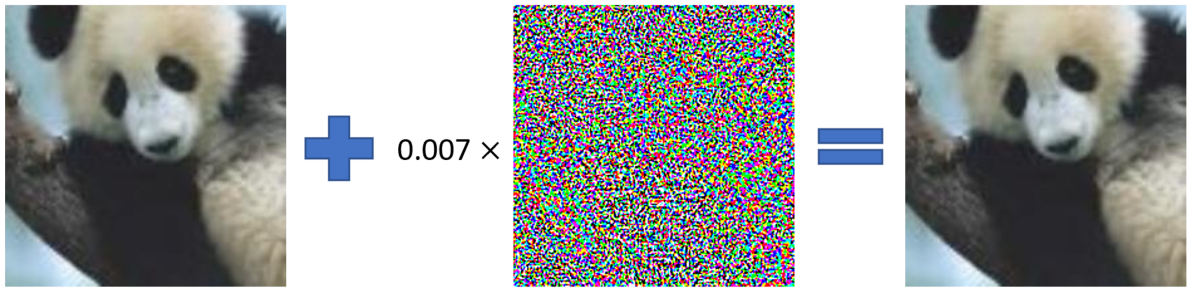
Background

- Machine learning (ML) algorithms in real-world applications mainly focus on increasing accuracy
- The security and robustness of ML models should also be considered
- Adversarial ML aims to study the security of ML models
- A subfield of research is to design adversarial examples to fool ML models

What are adversarial examples?

Adversarial Examples

- Natural (or normal) examples
  - Natural inputs of ML models
- Adversarial examples
  - Intentionally designed
  - Cause the model to make mistakes
  - Visually indistinguishable from natural examples
    - This constraint is not necessary as long as the added perturbation to the input is *label-preserving*
- A visual example:



Physical Adversarial Examples

- Manipulate a stop sign with physical adversarial patches
  - The ML model of a self-driving car classifies it as a speed limit sign

Lab Test

Physical road signs with adversarial perturbation

Cropping, Resizing

Stop Sign → Speed Limit Sign

Field (Drive-by) Test

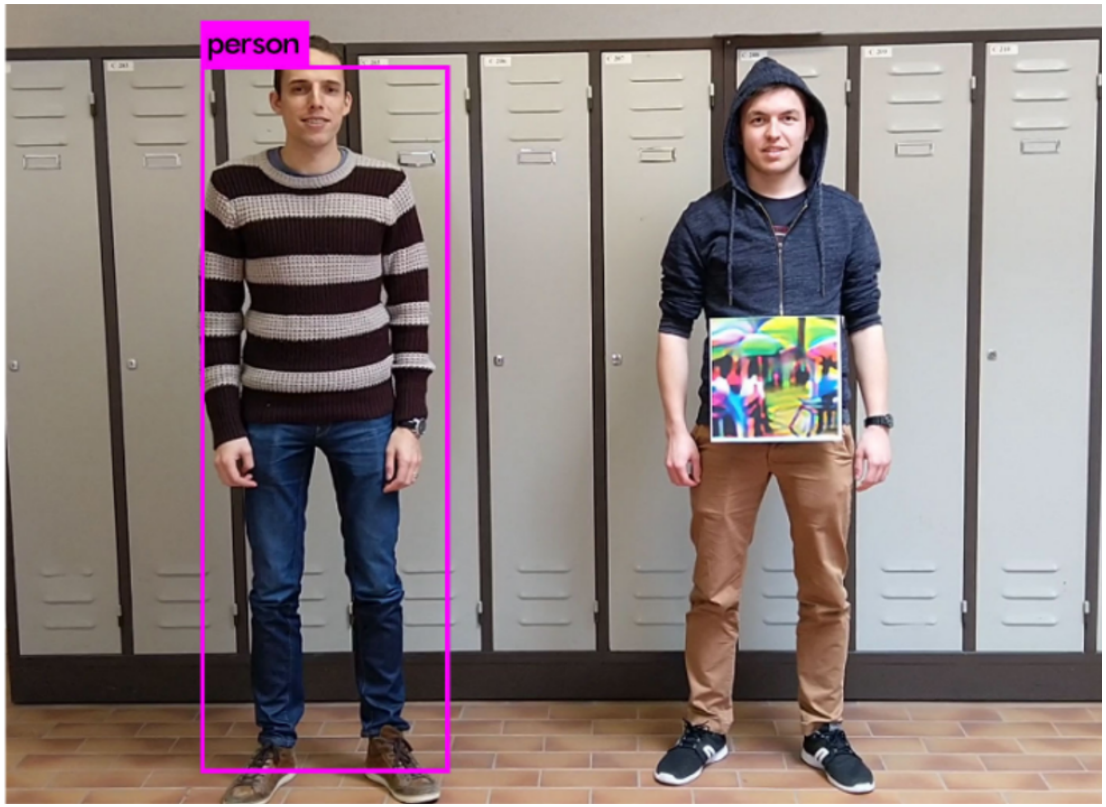
Physical road signs with adversarial perturbation

Sample per K frames, Cropping, Resizing

Stop Sign → Speed Limit Sign

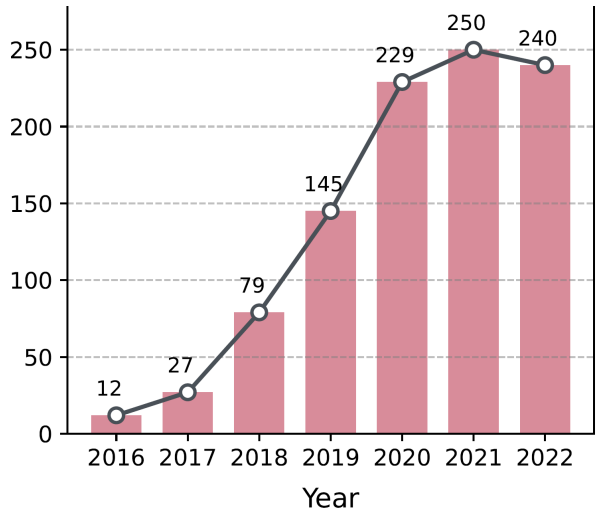
Physical Adversarial Examples

- The person on the right hanging the patch is ignored by the person detector based on YOLOv2



Adversarial Examples

- These examples suggest that ML models could make mistakes on images perturbed by imperceptible and malicious noise
- Adversarial ML is an emerging sub-area to study these adversarial phenomena



Question

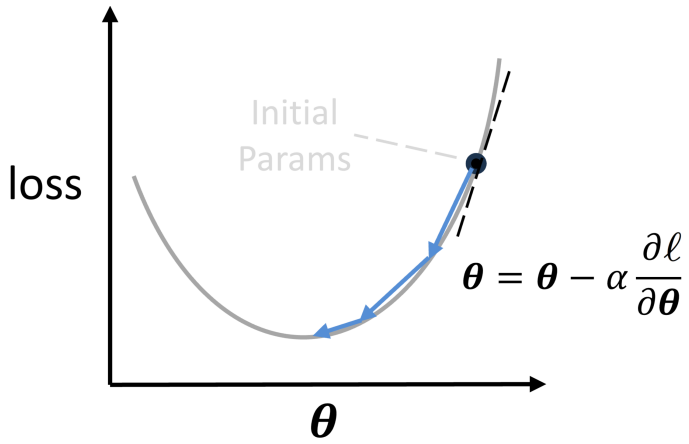
How do we identify adversarial examples?

Problem Description

- Given an image $\mathbf{x}$, the classifier to be fooled (e.g., SVM or CNN) predict its label as Class y
  - $f(\mathbf{x}) = y$
- Create an adversarial image $\mathbf{x}_{\text{adv}}$ by adding a small perturbation to the original image $\mathbf{x}$
  - I.e., the distance between $\mathbf{x}$ and $\mathbf{x}_{\text{adv}}$, $D(\mathbf{x}, \mathbf{x}_{\text{adv}})$, shall be small
- The classifier assigns a different label to the adversarial image
  - I.e., $f(\mathbf{x}_{\text{adv}}) = y_{\text{adv}} \neq y$
- If y_{adv} is randomly chosen, the attack is *untargeted*; if y_{adv} is pre-determined for some specific purposes, the attack is *targeted*

Gradient Descent Revisited

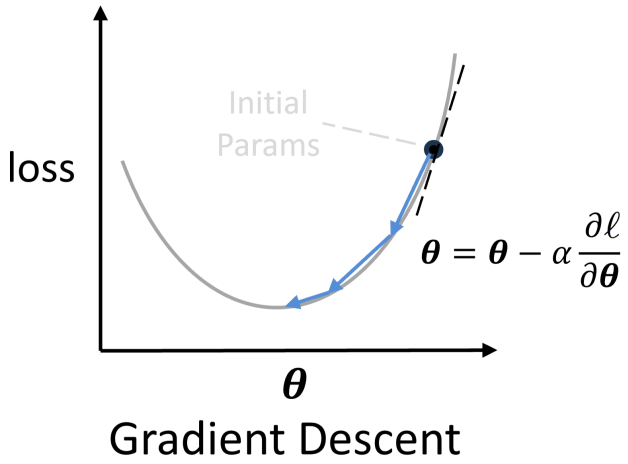
- Recall the gradient descent algorithm to train CNNs
  - The gradients of the loss function with respect to the parameters $\boldsymbol{\theta}$, $\frac{\partial \ell}{\partial \boldsymbol{\theta}}$, give the direction and magnitude for updating $\boldsymbol{\theta}$
  - The model parameters $\boldsymbol{\theta}$ are iteratively updated in the opposite direction of the gradients until a (local) minimum of the loss function is reached
    - $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial \ell}{\partial \boldsymbol{\theta}}$
    - α is the learning rate

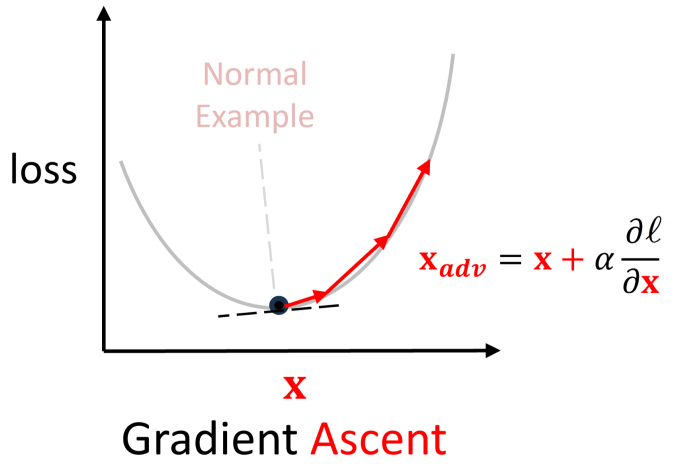


Gradient Descent

Method

- To create adversarial examples, the basic idea is to adopt *gradient ascent* to iteratively increase the loss
  - Given a pre-trained ML model, we freeze its parameters and update the input image by gradient ascent to increase the loss
  - This equals adding perturbation noise to the input, which tries to cause the model to make mistakes as much as possible





Methods

We mainly introduce three adversarial attack methods:

- Fast gradient sign method (FGSM)
- Projected gradient descent (PGD)
- Physical attacks

FGSM Attack

Fast gradient sign method (FGSM) attack was proposed by Goodfellow et al. in 2015

- The adversarial image $\mathbf{x}_{\text{adv}}$ is created by adding perturbation noise to the natural input $\mathbf{x}$

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \cdot \text{sign}(\frac{\partial \ell}{\partial \mathbf{x}})$$

- ϵ is the magnitude of the noise and $\text{sign}(\cdot)$ is sign function

$$\text{sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

- This increases the loss for the true class of $\mathbf{x}_{\text{adv}}$

Ian J. Goodfellow et al., "Explaining and Harnessing Adversarial Examples," *International Conference on Learning Representations*, 2015.

FGSM Attack

- FGSM obtains adversarial image with only one step, and cannot well guarantee visual imperceptibility

Original image

Prediction: car mirror

Adversarial image

Prediction: sunglasses

PGD Attack

Projected gradient descent (PGD) was proposed by Madry et al. in 2017

- PGD creates an adversarial image $\mathbf{x}_{\text{adv}}$ via iterative optimization. After each step of perturbation, the adversarial image is projected back onto the ℓ_∞-ball of radius ϵ centered at $\mathbf{x}$ using a projection function $\text{Proj}(\cdot)$

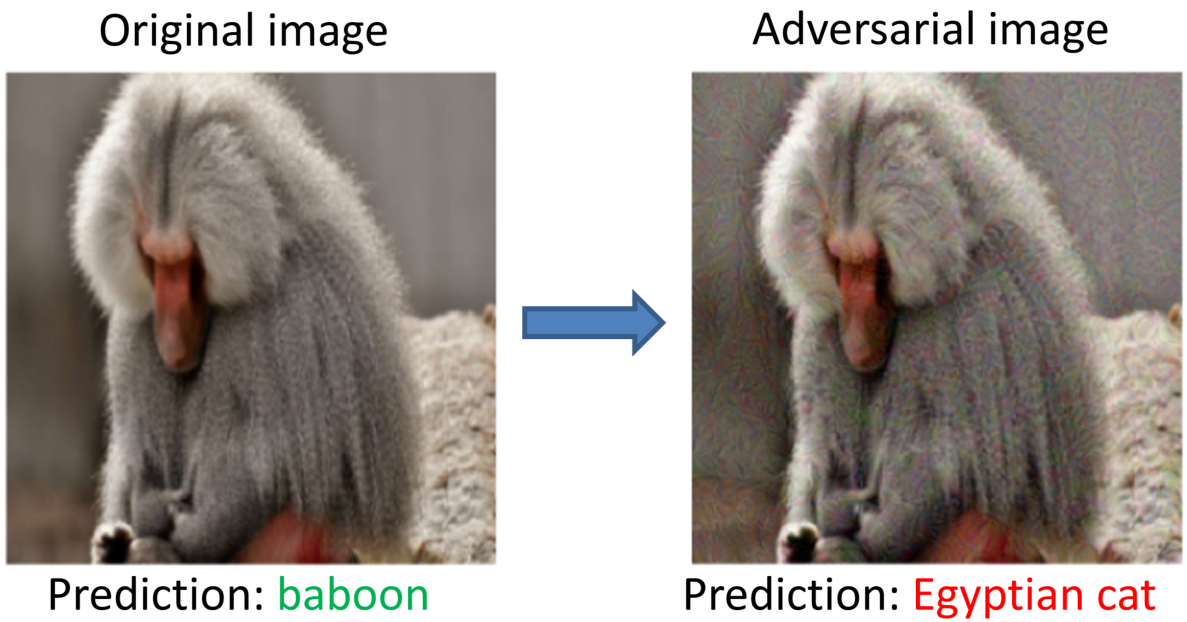
$$\mathbf{x}_{\text{adv}}^{(t)} = \text{Proj}\left(\mathbf{x}_{\text{adv}}^{(t-1)} + \alpha \cdot \text{sign}\left(\frac{\partial \ell}{\partial \mathbf{x}}\right)\right)$$

where α is the learning rate and t is the iteration step

Aleksander Madry et al., "Towards Deep Learning Models Resistant to Adversarial Attacks," *International Conference on Learning Representations*, 2018.

PGD Attack

- PGD is one of the strongest first-order attack methods
  - First-order attack means that the adversary uses only the gradients of the loss function with respect to the input



Physical Attack

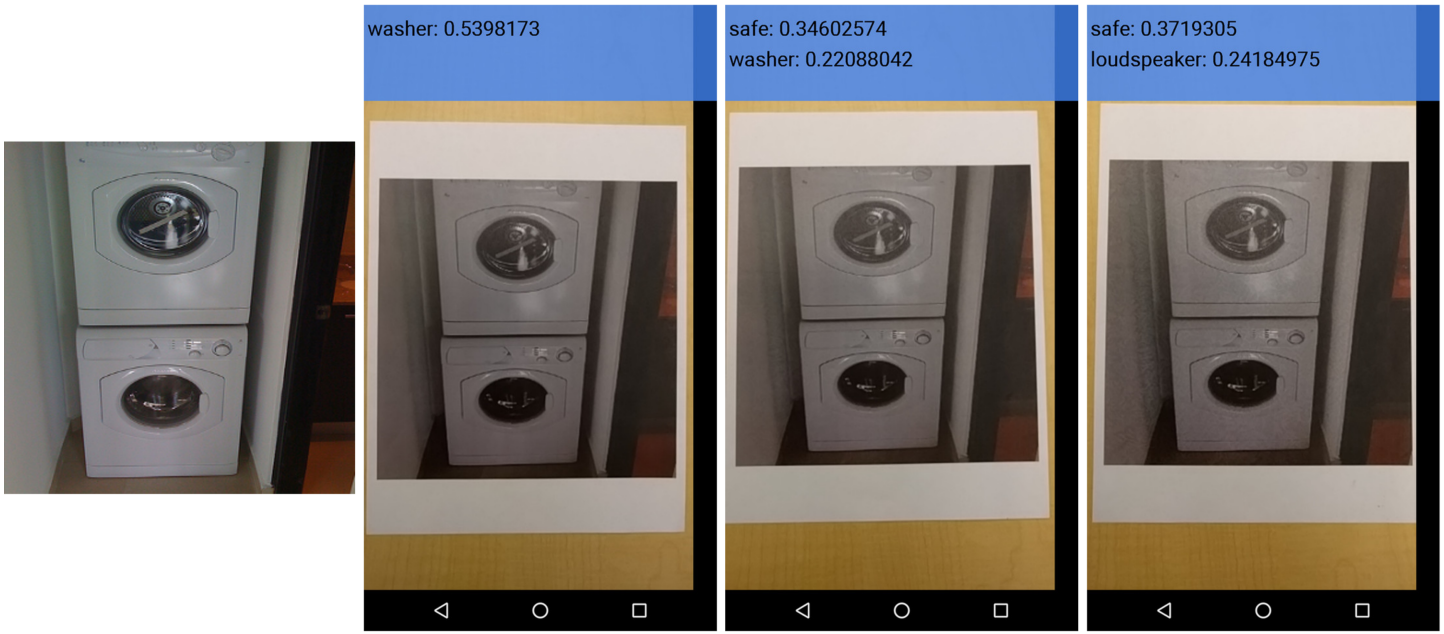
Physical attacks are for ML models deployed in physical scenarios. The whole attack procedure consists of three steps:

1. Generate adversarial examples in the digital space via methods like FGSM and PGD
2. Transform the digital perturbation to the object in the physical world via printing or other manners
3. Transform the physical perturbation back into the digital space via cameras/scanners and then conduct inference

Alexey Kurakin et al., "Adversarial Examples in the Physical World," *International Conference on Learning Representations Workshop*, 2017.

Physical Attack

- Example of physical attack



Original image | Clean image | Adv. Image, $\epsilon = 4$ | Adv. Image, $\epsilon = 8$

Defense

- Adversarial samples can cause ML algorithms to fail, so how can we defend them?
- Many defense strategies against adversarial attacks have been proposed, including:
  - Detecting and excluding adversarial examples
  - Adversarial training
  - Robust optimization (regularization, certified defenses)
  - ...

Defense

Training or fine-tuning the model using adversarial samples is referred to as **adversarial training**

- The training dataset is augmented with adversarial examples produced by known types of attacks
- Train the model from scratch using natural and adversarial examples
- Or train the model on natural examples and afterward fine-tune with adversarial examples