CPEG 589 – Advanced Deep Learning
Lecture 10

Outline

- Vision Transformer
- Positional Encoding
- Inductive Bias
- Adversarial Machine Learning

Positional Encoding in Transformer

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

Transformers for Computer Vision

- AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE-ICLR 2021 (https://arxiv.org/pdf/2010.11929.pdf)
- “We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks.”
- “Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.”
- Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP).
- The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019).
- “Thanks to Transformers’ computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters. With the models and datasets growing, there is still no sign of saturating performance.”

Transformer for Computer Vision

- Split an image into patches and provide the sequence of linear embeddings of these patches as an input to a Transformer.
- Image patches are treated the same way as tokens (words) in an NLP application.
- When trained on mid-sized datasets such as ImageNet, Tansformer models yield modest accuracies of a few percentage points below ResNets of comparable size.
- Reason: Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data.
- “However, the picture changes if the models are trained on larger datasets (14M-300M images). We find that large scale training trumps inductive bias.”
- Naive application of self-attention to images would require that each pixel attends to every other pixel.

Vision Tranformer

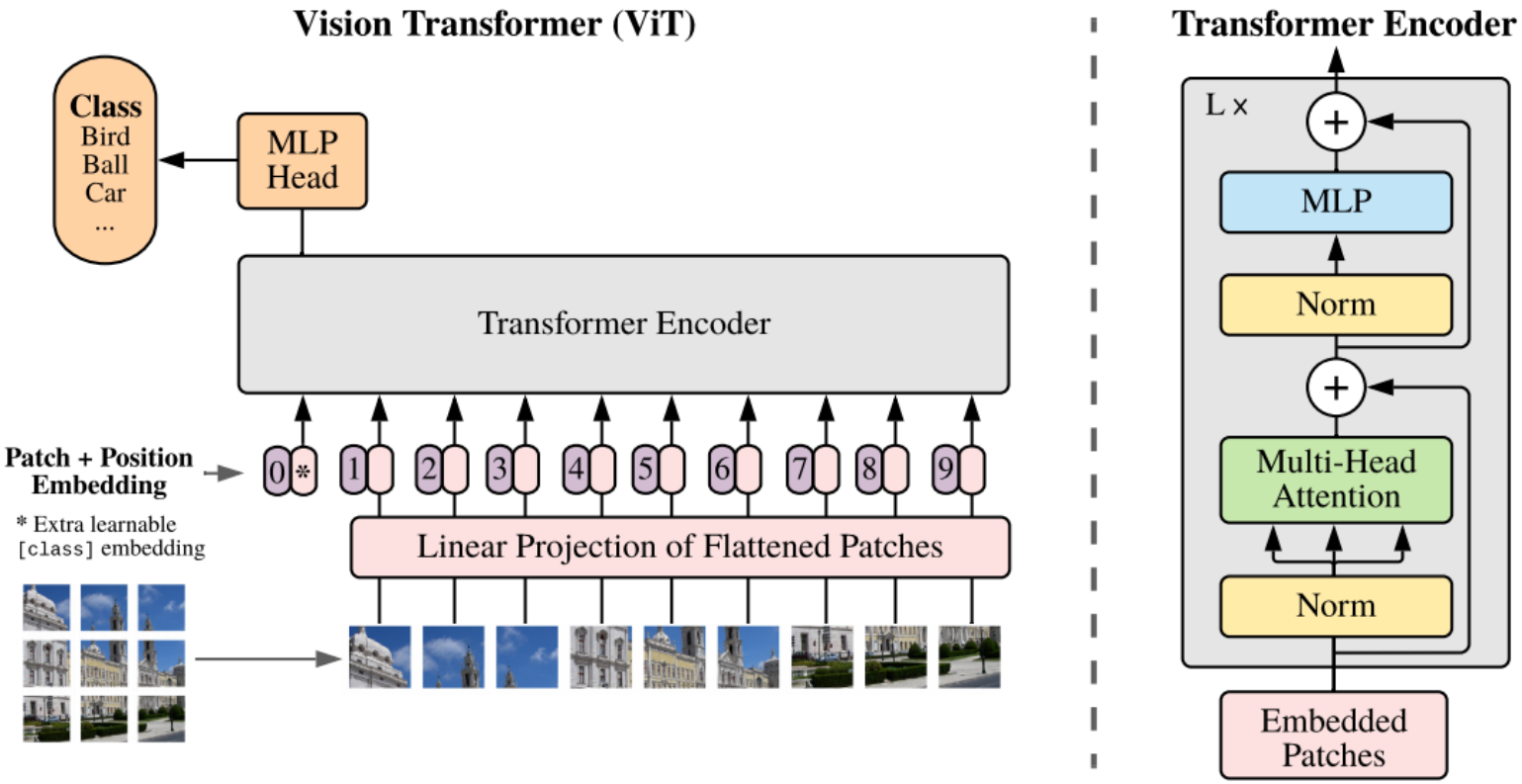


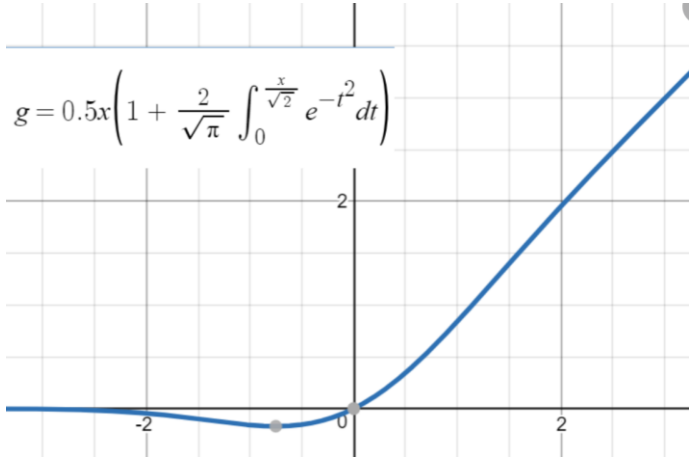
Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

ViT - Details

- The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images.
- Reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$

  where (H, W) is the resolution of the original image, C is the number of channels,

  (P, P) is the resolution of each image patch, and N = HW/P^2 is the resulting number of patches
- The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.
- GELU is used in MLP layers - Gaussian Error Linear Unit. This activation function is used in the most recent Transformers – e.g., BERT, GPT-2, GPT-3



ViT Formulation

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

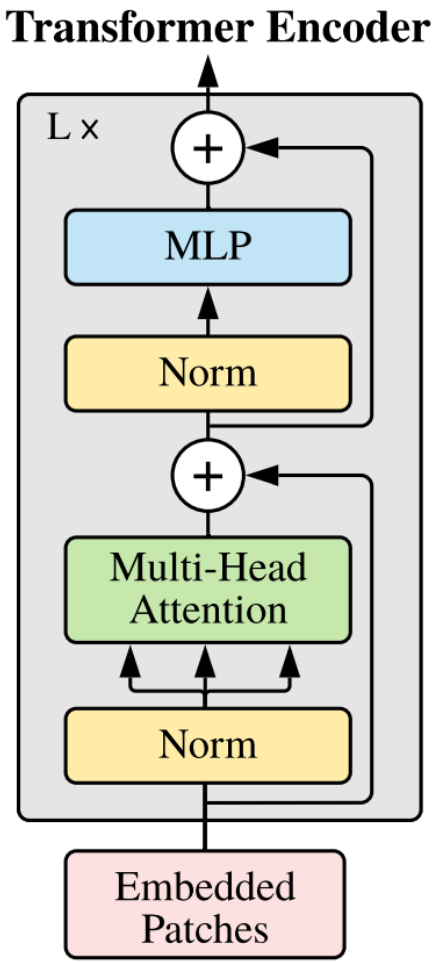
$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

MSA = Multi-Headed Self Attention

MLP = Multi Layer Perceptron

LN = Layer Norm



ViT Results

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

Table 1: Details of Vision Transformer model variants.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

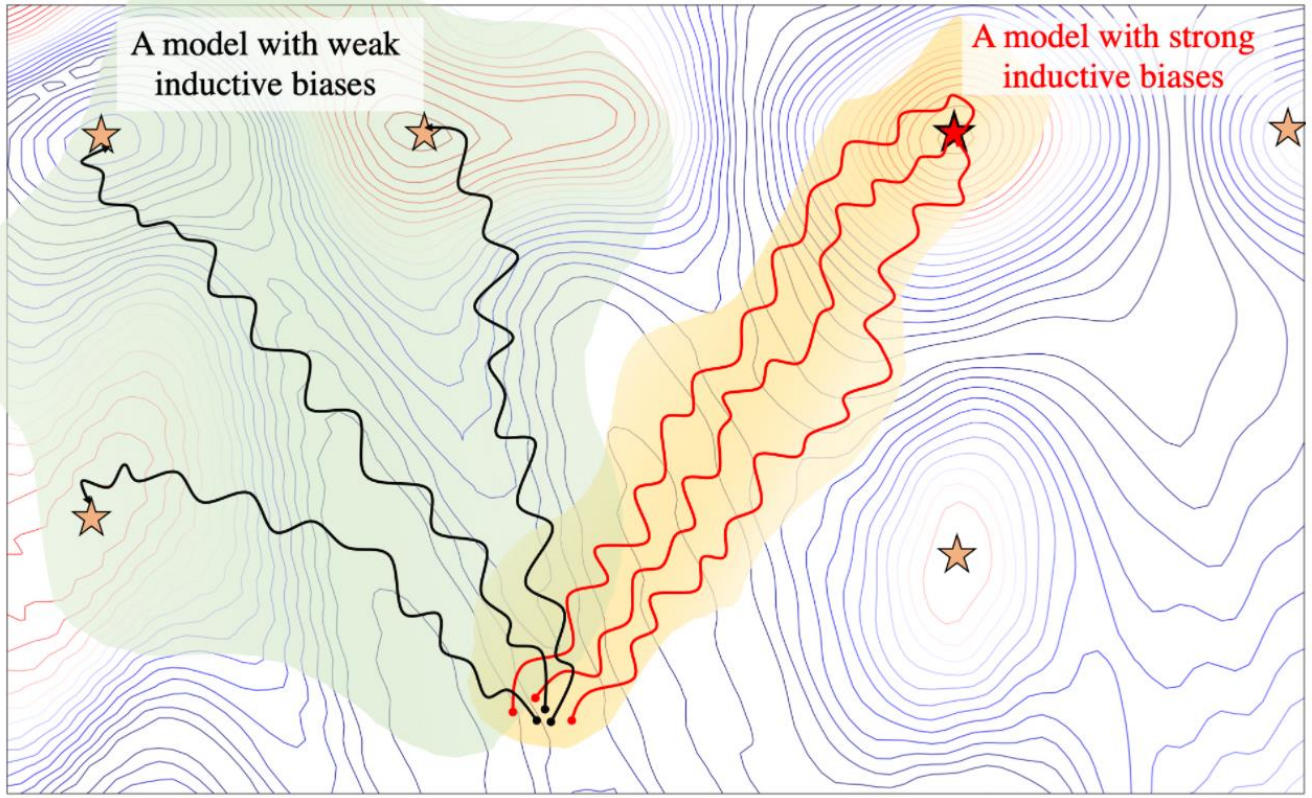
Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in Touvron et al. (2020).

Effect of Training Data Size

- Does Vision Transformer Always perform well.
- The Vision Transformer performs well when pre-trained on a large JFT-300M dataset.
- With fewer inductive biases for vision than ResNets, how crucial is the dataset size?
- Only with JFT-300M, do we see the full benefit of larger models.

Inductive Bias

- Inductive biases are the characteristics of learning algorithms that influence their generalization behavior, independent of data. They drive learning algorithms toward particular solutions.
- Without strong inductive biases, a model can be equally attracted to several local minima on the loss surface; the solution can be arbitrarily affected by random variations, e.g., the initial state or the order of training examples.



Inductive Bias

- The convolution operation in combination with max pooling makes CNNs approximately invariant to translation.
- Note that convolution is Translation equivariant (sensitive to translation), but pooling makes it translation invariant
- Translation invariance of CNNs improves their generalization and makes them data efficient compared to fully connected networks.
- In the lack of this inductive bias, the model needs to see examples of an image at different positions to be able to correctly classify them at test time.
- On the other hand, this translation invariance can hurt the performance of CNNs in cases where the position of the objects in the image matters.
- Transformers struggle to generalize on tasks that require capturing hierarchical structures when training data is limited.
- Can we combine the strengths of CNNs and Transformers?
- In a Transformer, we can transfer the effect of inductive bias through knowledge distillation.

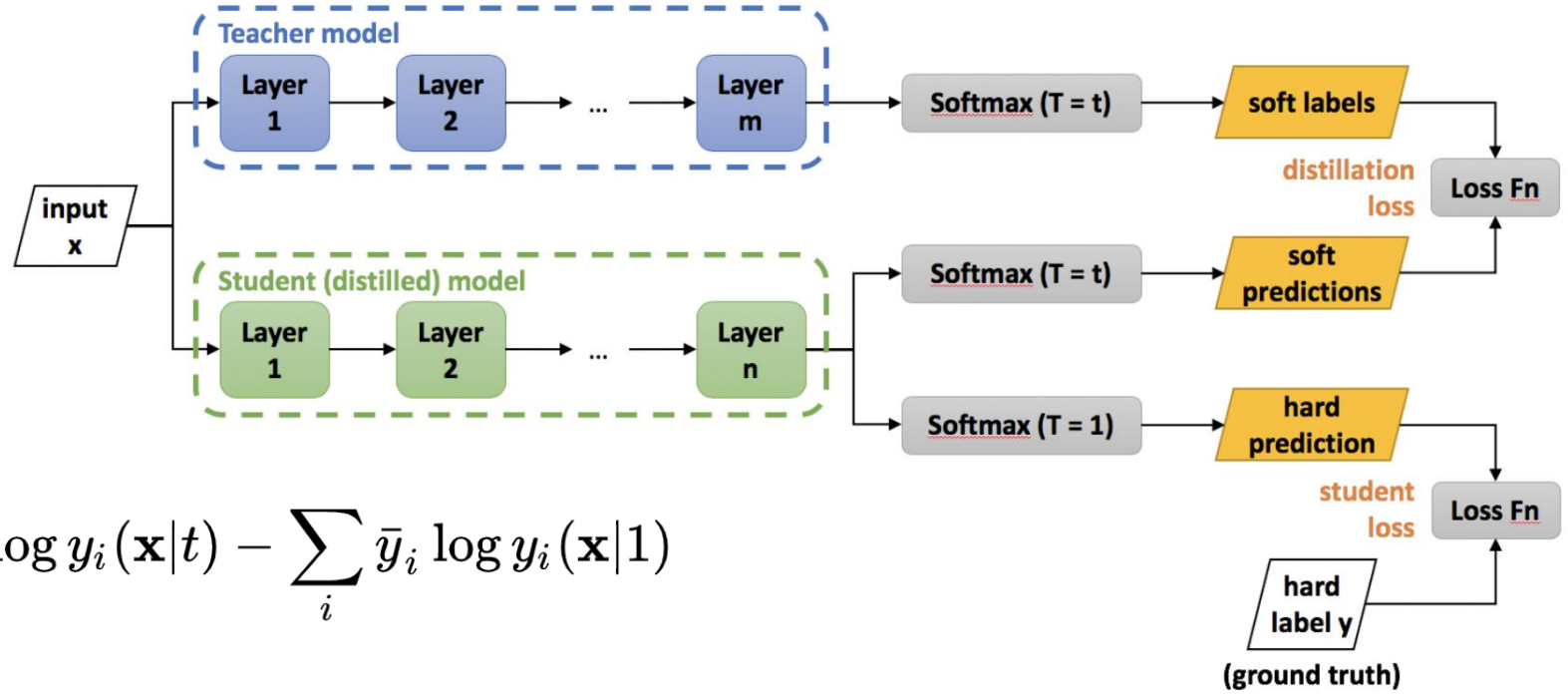
Inductive Bias Transfer – Knowledge Distillation ?

- Knowledge Distillation (Teacher-Student Model)

Neural networks typically produce class probabilities by using a "softmax" output layer that converts the logit, z_i, computed for each class into a probability, q_i, by comparing z_i with the other logits.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \quad (1)$$

- where *T* is a temperature that is normally set to 1. Using a higher value for *T* produces a softer probability distribution over classes.



$$E(\mathbf{x}|t) = -t^2 \sum_i \hat{y}_i(\mathbf{x}|t) \log y_i(\mathbf{x}|t) - \sum_i \bar{y}_i \log y_i(\mathbf{x}|1)$$

Adversarial Machine Learning

- **Whitebox Attack**
  - Input Image is adjusted in the direction of increasing Loss
  - Requires Knowledge of Model Parameters (weights and biases)
- **Blackbox Attack** (Transfer Attack)
  - No Knowledge of Model is needed as long as we can query the model – input->output label
  - Pure Blackbox Attack
    - Needs access to the training data
  - Mixed Black Box Attack
    - Attack Model is iteratively improved by using the training data as well as the query input-output label data

Computing Gradient w/r Input Image

- Tensorflow
- Gradient Tape Allows us to Access the Gradients with respect to the Loss and a particular parameter
- Example:

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as g:
  g.watch(x)
  y = x * x
  z = y * y
dz_dx = g.gradient(z, x)  # 108.0 (4*x^3 at x = 3)
dy_dx = g.gradient(y, x)  # 6.0
del g  # Drop the reference to the tape
```

Fast Gradient Sign Method - FGSM

- Whitebox Attack – Modifies the Input such that it causes increase in Loss

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

```
def adversarial_pattern(self, model, image, label):
        image = tf.cast(image, tf.float32)

        with tf.GradientTape() as tape:
            tape.watch(image)
            prediction = model(image)
            loss = tf.keras.losses.MSE(label, prediction)

        gradient = tape.gradient(loss, image)
        signed_grad = tf.sign(gradient)
       return signed_grad

    def create_adversarial_example(self, model, img_rows, img_cols, image, channels, image_label,
epsilon=0.1):
        perturbations = self.adversarial_pattern(model, image.reshape((1, img_rows, img_cols, channels)),
image_label).numpy()
        adversarial = image + perturbations * epsilon
        return adversarial
```

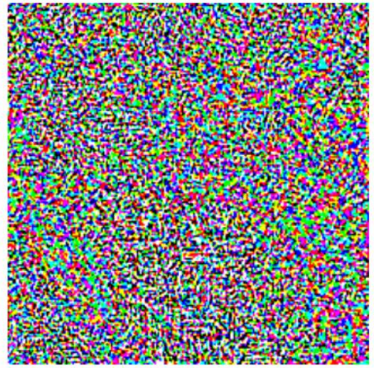
FGSM Attack

- Examples:

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



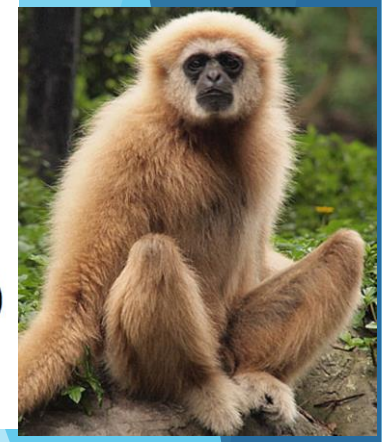
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

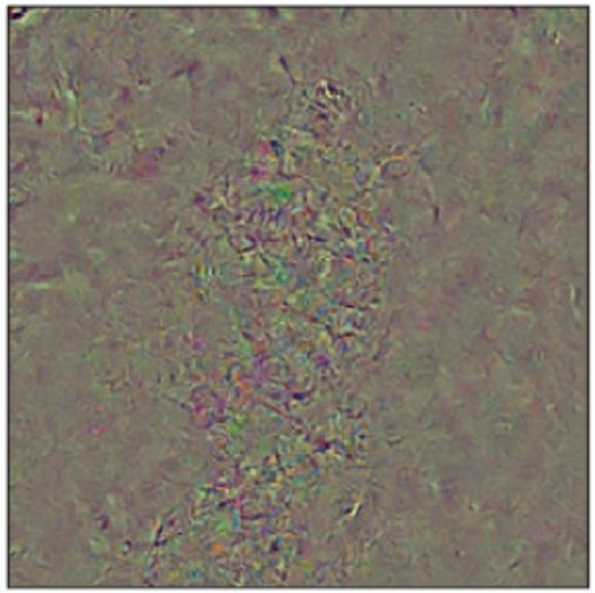
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence



P(robin) = 0.65

+



P(cleaver) = 0.02

=

P(waffle iron) = 1.00

FGSM in PyTorch

```
for data, target in test_loader:

    # Send the data and label to the device
    data, target = data.to(device), target.to(device)

    # Set requires_grad attribute of tensor. Important for Attack
    data.requires_grad = True

    # Forward pass the data through the model
    output = model(data)
    loss.backward()

    # Collect datagrad
    data_grad = data.grad.data
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
```

How to Safeguard Against Whitebox Attacks?

- Do Adversarial Training.
- One of the Most Popular Techniques is Called Madry Training Using the Projective Gradient Descent (PGD) Algorithm.
- PGD Objective is a Min-Max Problem (Relationship to GAN?)

$$\underset{\theta}{\text{minimize}} \frac{1}{|S|} \sum_{x,y \in S} \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y).$$

$$\theta := \theta - \alpha \frac{1}{|B|} \sum_{x,y \in B} \nabla_\theta \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y).$$

Projected Gradient Descent Attack - Training

- PGD Algorithm (Defense - Training)

Repeat:

1. Select minibatch B, initialize gradient vector $g := 0$
2. For each (x, y) in B:
   a. Find an attack perturbation $\delta^\star$ by (approximately) optimizing

$$\delta^\star = \underset{\|\delta\| \leq \epsilon}{\operatorname{argmax}} \ell(h_\theta(x + \delta), y)$$

   b. Add gradient at $\delta^\star$

$$g := g + \nabla_\theta \ell(h_\theta(x + \delta^\star), y)$$

3. Update parameters θ

$$\theta := \theta - \frac{\alpha}{|B|} g$$

Madry PGD Training

- CIFAR 10 - Attacks are allowed to perturb each pixel of the input image by at most epsilon=8.0 on a 0-255 pixel scale.
- PGD with 7 steps of size 2 to train the model, while using 20 steps of size 1 to attack it?

			Standard Accuracy			Robust Accuracy		
	Norm	ε	Standard	Half-half	Robust	Standard	Half-half	Robust
MNIST	ℓ_∞	0	99.31%	-	-	-	-	-
		0.1	99.31%	99.43%	99.36%	29.45%	95.29%	95.05%
		0.2	99.31%	99.22%	98.99%	0.05%	90.79%	92.86%
		0.3	99.31%	99.17%	97.37%	0.00%	89.51%	89.92%
	ℓ_2	0	99.31%	-	-	-	-	-
		0.5	99.31%	99.35%	99.41%	94.67%	97.60%	97.70%
		1.5	99.31%	99.29%	99.24%	56.42%	87.71%	88.59%
		2.5	99.31%	99.12%	97.79%	46.36%	60.27%	63.73%
CIFAR10	ℓ_∞	0	92.20%	-	-	-	-	-
		$2/255$	92.20%	90.13%	89.64%	0.99%	69.10%	69.92%
		$4/255$	92.20%	88.27%	86.54%	0.08%	55.60%	57.79%
		$8/255$	92.20%	84.72%	79.57%	0.00%	37.56%	41.93%
	ℓ_2	0	92.20%	-	-	-	-	-
		$20/255$	92.20%	92.04%	91.77%	45.60%	83.94%	84.70%
		$80/255$	92.20%	88.95%	88.38%	8.80%	67.29%	68.69%
		$320/255$	92.20%	81.74%	75.75%	3.30%	34.45%	39.76%

Blackbox Attack

- In white-box attacks the architecture of the classifier, and trained weights are assumed to be known.
- In black-box attacks, the architecture and trained weights are assumed to be secret.
- if the defense conceals gradient information, white-box attacks will fail but black-box attacks may succeed.
- Blackbox Attack variations
- **Target Model Based Attack**
  - Score-based Blackbox Attack
  - Decision Based Blackbox Attack
- **Synthetic Model Based Attacks (Transfer Attacks)**
  - Pure Blackbox Attack
  - Oracle Based Blackbox Attack
  - Mixed Blackbox Attack

Blackbox Attack

- The synthetic model is sometimes referred to as Carlini Network.
- Create another Network (simple CNN – Carlini Network) that is trained on the same training data (pure back box) as the Network we want to attack.
- Create Adversarial Examples (using PGD) on the Carlini Network, then attack the real Network with these examples.
- Thus Blackbox Attack uses the Transfer Learning to come up wih the Adversarial Examples.
- CNNs have the drawback of being Extremely susceptible to Transfer or Blackbox Attacks – even if the complexity of the target and the attack networks differ greatly.
- Reference: Kaleel Mahmood, Deniz Gurevin, Marten van Dijk, Phuong Ha Nguyen, “Beware the Black-Box: on the Robustness of Recent Defenses to Adversarial Examples", https://arxiv.org/pdf/2006.10876.pdf

Defense vs Clean Accuracy Tradeoff

- Better Adversarial Defense comes at the cost of reduction in Clean Accuracy.
- Can we make a model robust to White or Blackbox Attacks?
- Can we make a model robust to White or Blackbox attacks without sacrificing Accuracy?
- One recent Approach Friendly Adversarial Training (FAT).

Jingfeng Zhang, Xilie Xu, Bo Han, Gang Niu, Lizhen Cui, Masashi Sugiyama, Mohan Kankanhalli , “Attacks Which Do Not Kill Training Make Adversarial Learning Stronger”, https://arxiv.org/pdf/2002.11242.pdf (ICML 2020)