# Universal Adversarial Perturbations - Lab

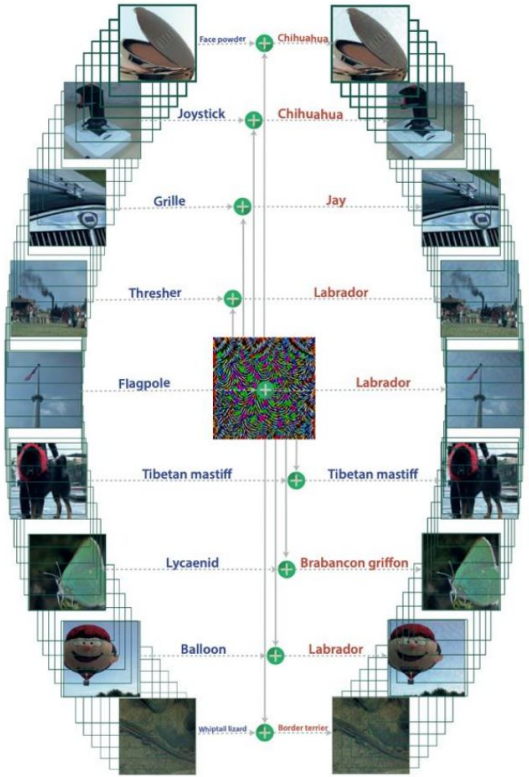
(Machine Learning Security - Fall 2025)

Dorjan Hitaj
Fabio De Gaspari

SAPIENZA
UNIVERSITÀ DI ROMA

# Universal adversarial perturbations

There exists a universal and small perturbation vector that causes images to be misclassified with high probability

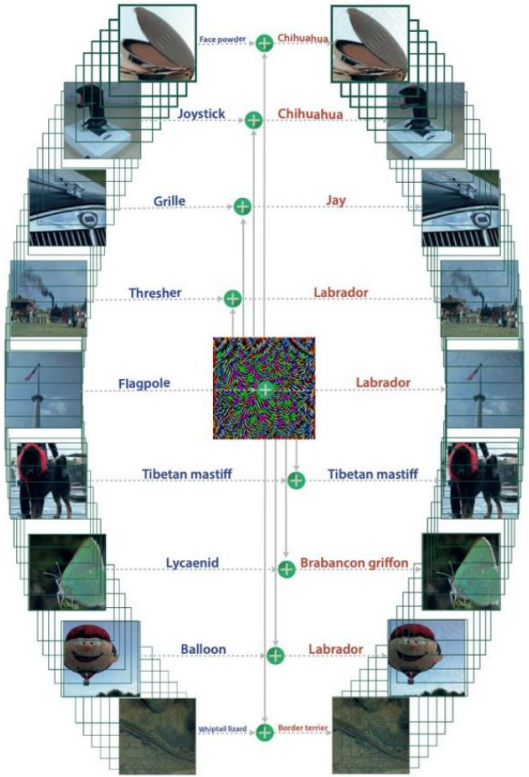


# Universal adversarial perturbations

There exists a universal and small perturbation vector that causes images to be misclassified with high probability.

In principle, same as before:

- Optimise the perturbation over different images until you get one that works on all intended classes



# Universal adversarial perturbations

**Algorithm 1** Computation of universal perturbations.

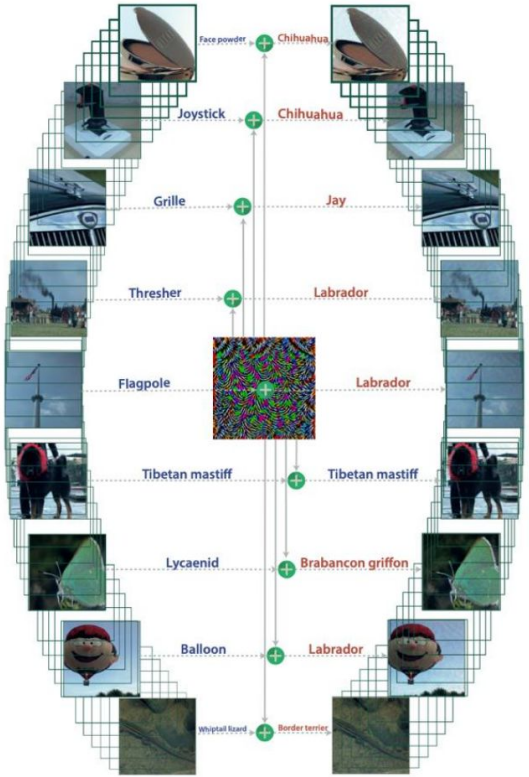
**input:** Data points $X$, classifier $\hat{k}$, desired $\ell_p$ norm of the perturbation $\xi$, desired accuracy on perturbed samples $\delta$.
**output:** Universal perturbation vector $v$.
Initialize $v \leftarrow 0$.
**while** $\text{Err}(X_v) \leq 1 - \delta$ **do**
    **for** each datapoint $x_i \in X$ **do**
        **if** $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
            Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

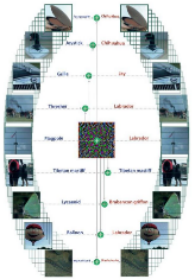
            Update the perturbation:

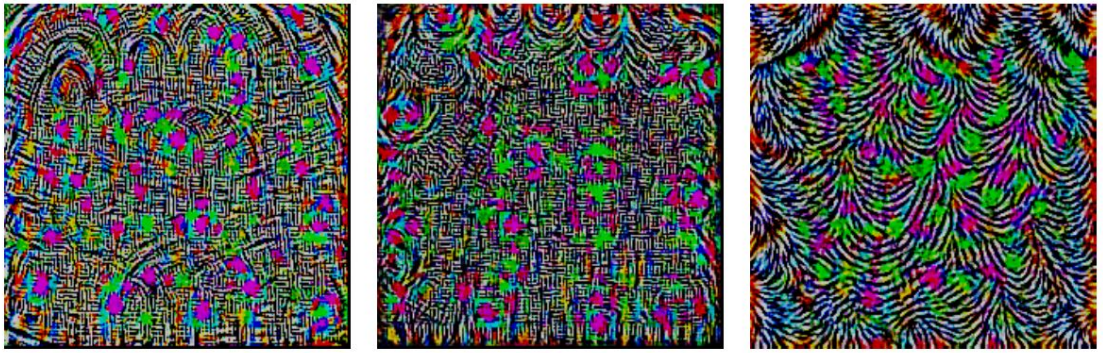
$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

        **end if**
    **end for**
**end while**



# Universal adversarial perturbations

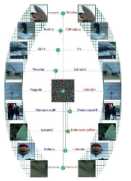




(a) CaffeNet (b) VGG-F (c) VGG-16

(d) VGG-19 (e) GoogLeNet (f) ResNet-152

# Universal adversarial perturbations



- A bit too slow and sometimes takes time to/does not converge

**Algorithm 1** Computation of universal perturbations.

**input:** Data points $X$, classifier $\hat{k}$, desired $\ell_p$ norm of the perturbation $\xi$, desired accuracy on perturbed samples $\delta$.
**output:** Universal perturbation vector $v$.
Initialize $v \leftarrow 0$.
**while** $\text{Err}(X_v) \leq 1 - \delta$ **do**
    **for** each datapoint $x_i \in X$ **do**
        **if** $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
            Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

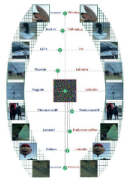
$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

            Update the perturbation:

$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

        **end if**
    **end for**
**end while**

# Universal adversarial perturbations - a faster variant



**Algorithm 1** Computation of universal perturbations.

1: **input:** Data points $X$, classifier $\hat{k}$, desired $\ell_p$ norm of

...rbed sam-

...ation that

$) \neq \hat{k}(x_i)$.

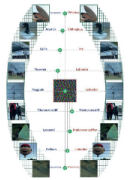
$v \leftarrow \ldots \; p,\xi(v + \Delta v_i)$.

9: **end if**
10: **end for**
11: **end while**

**Algorithm 2** Stochastic gradient for universal perturbation

**for** epoch $= 1 \ldots N_{ep}$ **do**
  **for** minibatch $B \subset X$ **do**
    Update $\delta$ with gradient variant $\delta \leftarrow \delta + g$
    Project $\delta$ to $\ell_p$ ball
  **end for**
**end for**

# Universal adversarial perturbations - a faster variant



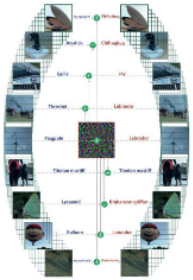
The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

**Universal Adversarial Training**

**Ali Shafahi,* Mahyar Najibi,* Zheng Xu,* John Dickerson, Larry S. Davis, Tom Goldstein**
Deparment of Computer Science
University of Maryland
College Park, Maryland 20742
{ashafahi, najibi, xuzh, john, lsd,tomg}@cs.umd.edu

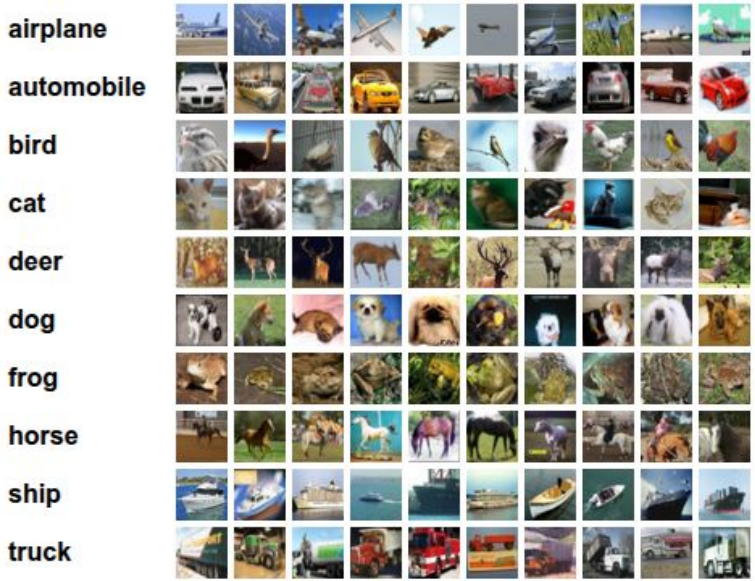
*Attached to this Lab materials

# Task at hand

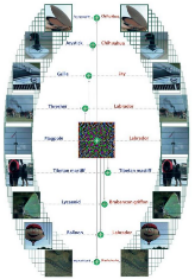


## CIFAR 10

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

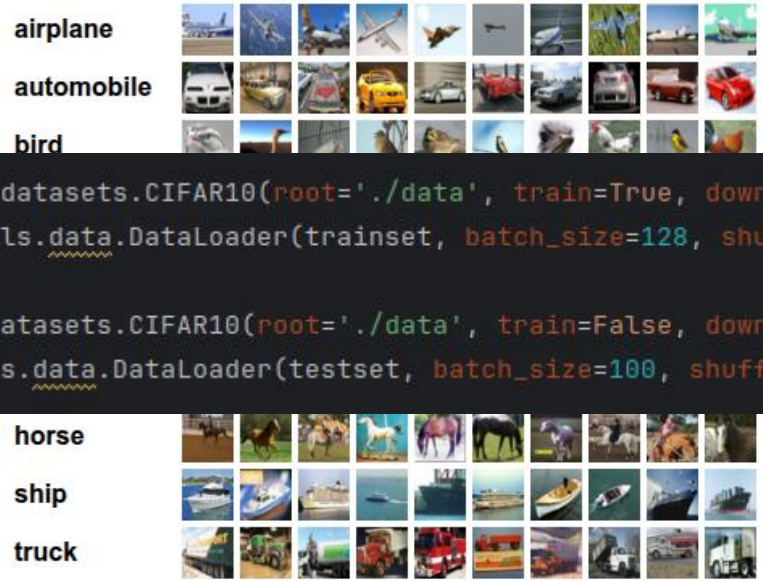


# Task at hand - Loading the data



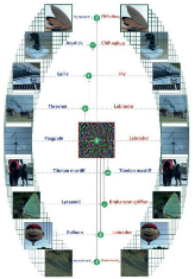
CIFAR 10

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.



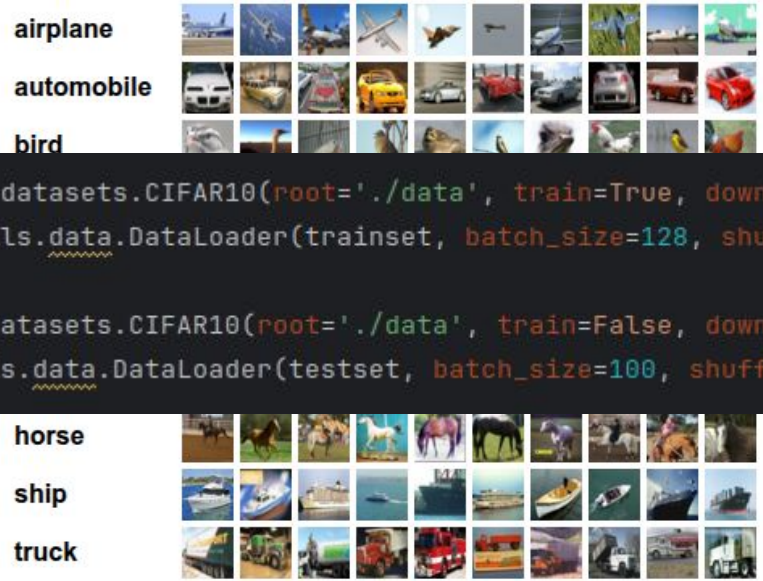

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False, num_workers=2)
```

# Task at hand - Loading the data



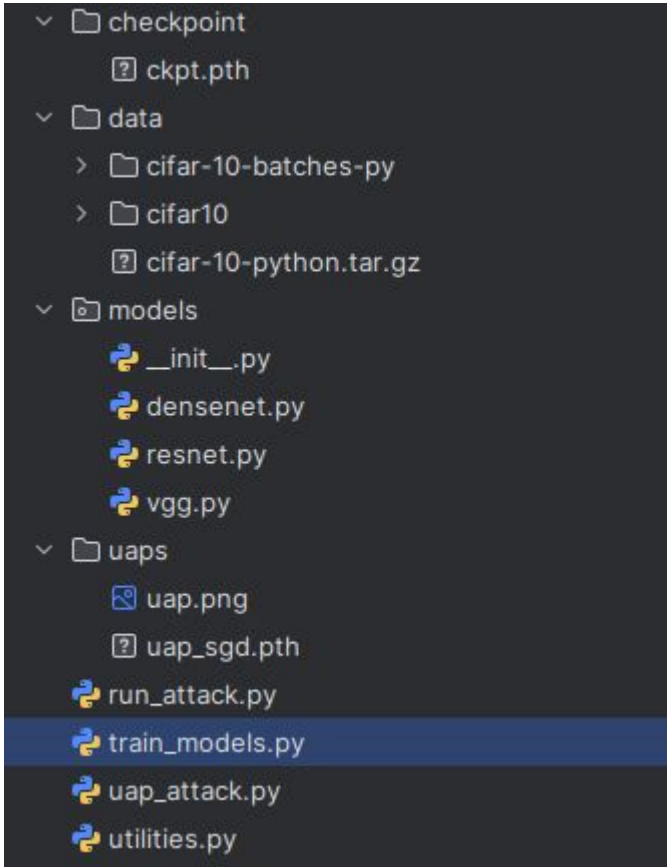
CIFAR 10

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.



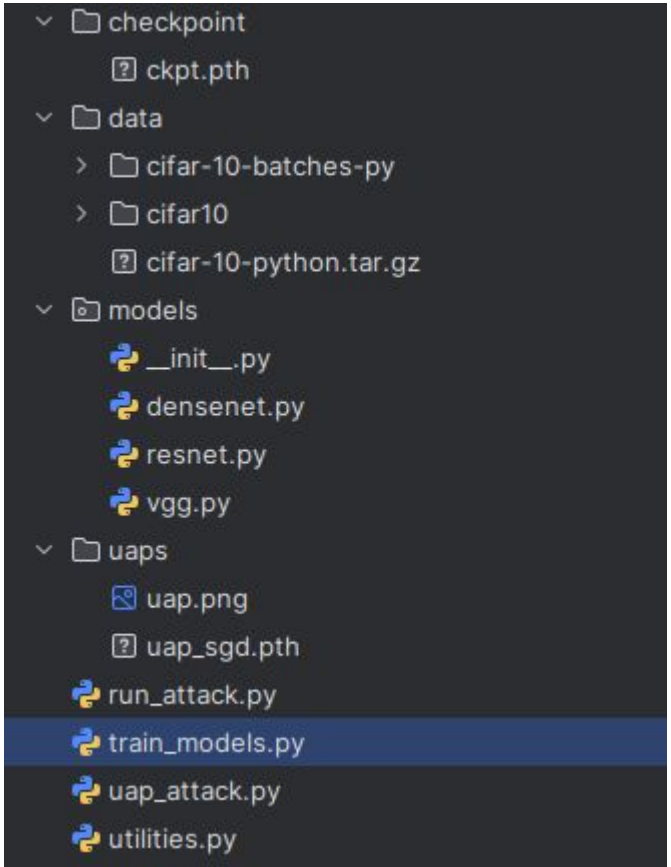

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False, num_workers=2)
```

# The overall code structure




- **models/** -3 different DNN architectures already implemented
- **data/** - the folder where cifar 10 data will be stored, automatically by the torch dataloader
- **uaps/** - the folder where you will store the different perturbations that you will generate for further use in evaluation.

# The overall code structure




- **run_attack.py -** short snippet of code to evaluate uaps
- **train_models.py -** contains the code to train different type of models on cifar dataset
- **utilities.py -** bunch of useful functions and definitions

- **uap_attack.py** - contains a method definition that you need to fill out

# What you will need to do in UAP method?

- Expand current universal perturbation to the batch.
- Compute the loss of the model on perturbed inputs (with per-sample losses clamped).
- Compute gradients of that loss w.r.t. the per-sample perturbations.
- Average the gradients across the batch, take the sign, and update the universal perturbation by a small step.
- Clip the universal perturbation to the allowed Linf ball (the bounds [-eps, eps]).

# What you will need to do in UAP method?

```python
def uap_sgd2(  2 usages
    model: nn.Module,
    loader: torch.utils.data.DataLoader,
    nb_epoch: int,
    eps: float,
    beta: float = 12.0,
    step_decay: float = 0.8,
    y_target: Optional[int] = None,
    loss_fn: Optional[nn.Module] = None,
    uap_init: Optional[torch.Tensor] = None,
    device: Optional[torch.device] = None,
) -> Tuple[torch.Tensor, List[float]]:
    """
    Universal Adversarial Perturbation (UAP) via sign-of-mean-grad updates.
    Args:
        model: pretrained classifier (expects inputs in [0,1]).
        loader: DataLoader yielding (images, labels).
        nb_epoch: number of epochs over the dataset.
        eps: Linf bound for the universal perturbation (e.g., 8/255).
        beta: clamp value for per-sample loss (scalar).
        step_decay: multiplicative decay factor applied per epoch: step = eps * (step_decay ** epoch).
        y_target: if provided, generate a targeted UAP towards this class (int).
        loss_fn: optional loss function (default CrossEntropyLoss(reduction='none')).
        uap_init: optional initial perturbation tensor of shape (C,H,W).
        device: torch.device or None (will use model device if None).
    Returns:
        delta: tensor (C,H,W) -- learned universal perturbation (detached).
        losses: list of scalar loss values recorded (one per batch).
    """
```

Ali Shafahi,  Mahyar Najibi,  Zheng Xu,  John Dickerson, Larry S. Davis, Tom Goldstein, Universal Adversarial Training

# Pseudocode

**Input:**

Pretrained model $f(\cdot)$
Dataset $D = \{(x_i, y_i)\}$
Epochs: $N$
Perturbation limit: $\varepsilon$
Step decay factor: $\alpha$
Clamping constant: $\beta$
(Optional) Target label y_target
(Optional) Loss function $L(\cdot)$ — default: Cross-Entropy

**Output:**

Universal perturbation $\delta$

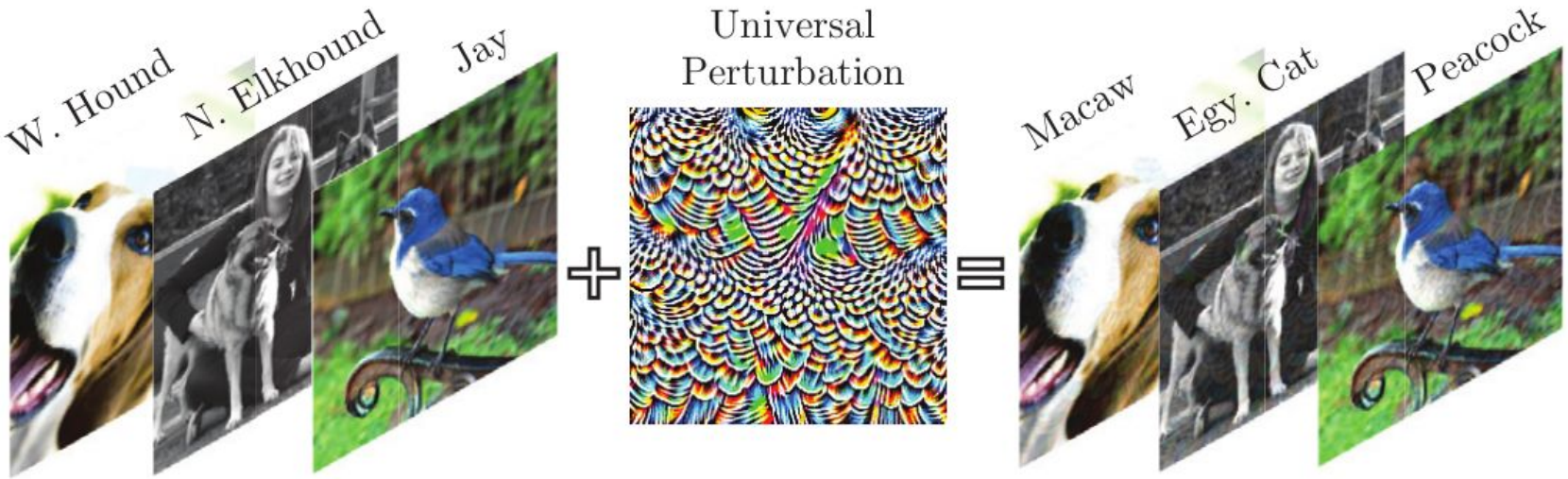
# Pseudocode

Set δ ← 0 (or provided initialization)
Define clamped loss:
L_clamp(x, y) = mean(min(L(f(x), y), β))

**For each epoch e = 1 … N:**
Set step size: η ← ε × (α)^e
For each mini-batch (x_b, y_b) ∈ D:
If targeted → replace y_b ← y_target
Compute perturbed inputs:
x_pert ← clip(x_b + δ, 0, 1)
Forward pass: p ← f(x_pert)
Compute loss:
ℓ ← L_clamp(p, y_b)
If targeted → ℓ ← -ℓ
Compute gradient of ℓ w.r.t δ: g ← ∇_δ ℓ
Average gradients and take sign:
g_sign ← sign(mean(g))
Update perturbation:
δ ← δ + η × g_sign
Project δ to valid range:
δ ← clip(δ, -ε, ε)

# Where you will need to end up?

```
def uap_sgd2(  2 usages
    model: nn.Module,
    loader: torch.utils.data.DataLoader,
    nb_epoch: int,
```




```
        delta: tensor (C,H,W) -- learned universal perturbation (detached).
        losses: list of scalar loss values recorded (one per batch).
    """
```

Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S. Davis, Tom Goldstein, Universal Adversarial Training (paper attached to this lab resources)

# Up next...

- Explainable AI and Guaranteed robustness towards adversarial examples [Next Lecture]
- Lab on explainable AI approaches for image classifiers and malware detectors.