Hack Object Detector is just Like Training Neural Networks

Jay Xiong, Wang Yang, Liu Yan, Hao Xin, Wei Tao

Presented at HITB GSEC 2019

Baidu X-Lab

About me

Baidu X-Lab

- **Jay Xiong,**
- Security Researcher
- League of Legends/Dead by Daylight Player

Agenda

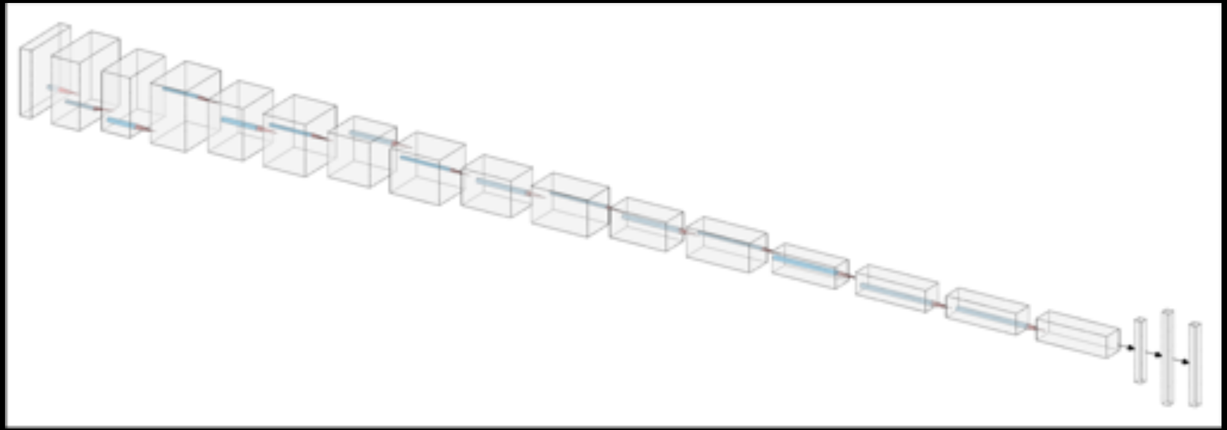
Baidu X-Lab

- Background on object detector deception(ODD)
- Object Detector Deception Analysis
- Use Trick to Enhance ODD
- One More Thing

Some concepts

- What is object detector?
- What is adversarial example?
- What is object detector deception?

What is object detector?



Faster YOLO v2 model architecture

```
def build_networks(self):
        if self.disp_console : print "Building YOLO_tiny graph..."
        self.x = tf.placeholder('float32',[None,448,448,3])
        self.conv_1 = self.conv_layer(1,self.x,16,3,1)
        self.pool_2 = self.pooling_layer(2,self.conv_1,2,2)
        self.conv_3 = self.conv_layer(3,self.pool_2,32,3,1)
        self.pool_4 = self.pooling_layer(4,self.conv_3,2,2)
        self.conv_5 = self.conv_layer(5,self.pool_4,64,3,1)
        self.pool_6 = self.pooling_layer(6,self.conv_5,2,2)
        self.conv_7 = self.conv_layer(7,self.pool_6,128,3,1)
        self.pool_8 = self.pooling_layer(8,self.conv_7,2,2)
        self.conv_9 = self.conv_layer(9,self.pool_8,256,3,1)
        self.pool_10 = self.pooling_layer(10,self.conv_9,2,2)
        self.conv_11 = self.conv_layer(11,self.pool_10,512,3,1)
        self.pool_12 = self.pooling_layer(12,self.conv_11,2,2)
        self.conv_13 = self.conv_layer(13,self.pool_12,1024,3,1)
        self.conv_14 = self.conv_layer(14,self.conv_13,1024,3,1)
        self.conv_15 = self.conv_layer(15,self.conv_14,1024,3,1)
        self.fc_16 = self.fc_layer(16,self.conv_15,256,flat=True,linear=False)
        self.fc_17 = self.fc_layer(17,self.fc_16,4096,flat=False,linear=False)
        #skip dropout_18
        self.fc_19 = self.fc_layer(19,self.fc_17,1470,flat=False,linear=True)
        self.sess = tf.Session()
        self.sess.run(tf.initialize_all_variables())
        self.saver = tf.train.Saver()
        self.saver.restore(self.sess,self.weights_file)
        if self.disp_console : print "Loading complete!" + '\n'
```

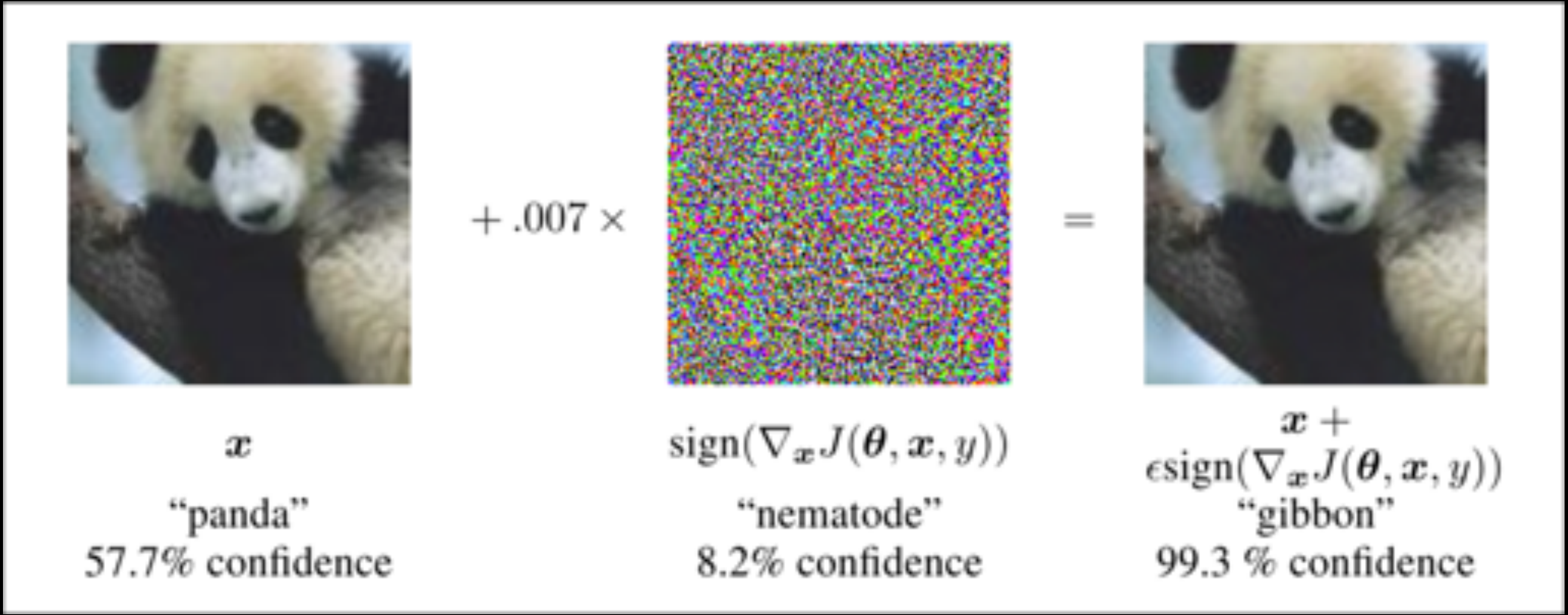
Cited from github user gliese581gg

https://github.com/gliese581gg/YOLO_tensorflow/blob/master/YOLO_tiny_tf.py

What is object detector?

Terminator 1, 1984,
https://www.youtube.com/watch?v=9MeaaCwBW28

What is adversarial example?



Ian J. Goodfellow, etc, EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES, ICLR 2015, https://arxiv.org/abs/1412.6572

Object detector deception(ODD)

object detector

combined with

adversarial example

Figure 4: Output of the extended RP_2 algorithm to attack YOLO v2 using poster and sticker attacks.

Kevin Eykholt, etc, Physical Adversarial Examples for Object Detectors, USENIX WOOT 2018

Baidu X-Lab

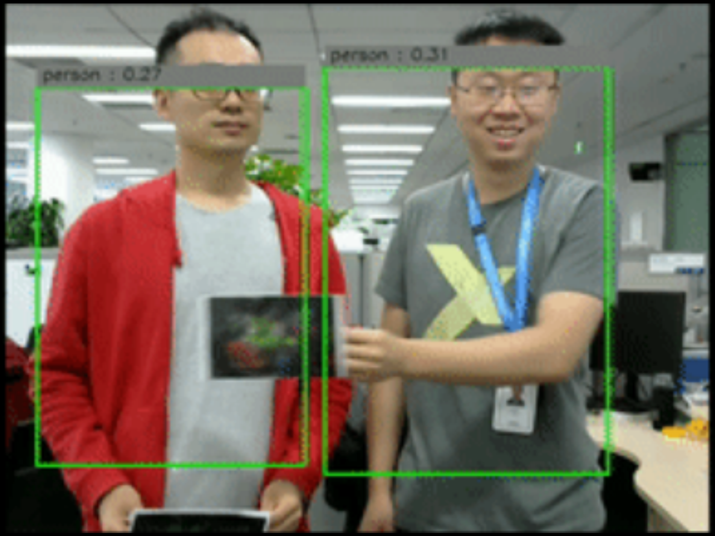
Adversarial Threat to DNN

- A real threat in deep learning application scenario

Physical attack against YOLO-V3 Baidu X-lab demonstrated on Blackhat Euro 2018

Related work on object detector deception(ODD)

CVPR Workshop 2019, [*], obtained from attacking test batch distribution
https://www.youtube.com/watch?v=MIbFvK2S9g8



Generated from attacking prediction score of a single image
https://github.com/advboxes/AdvBox

[*] Simen Thys, Wiebe Van Ranst, Toon Goedemé, Fooling automated surveillance cameras: adversarial patches to attack person detection. CVPR Workshop 2019,

Get down to what's happened

· The neural network training process.

The neural network training process is to search for a set of weights, which minimize the difference between model prediction and training/test label distribution. It can be summarized as follows:

Training Loss is $J(y_t = h_\theta(x), y_s)$, $y_s \sim D_s$, find θ, $y = h(\theta, x)$, st. $(x, y) \sim D_s$

· The ODD process

The ODD process is an "odd" process compared to how we usually a neural network, instead of trying to find a set of weights to make model prediction more precise, the ODD is a reversed process.

With model weights remain unchanged, we try to find a robust input δ to drift model prediction far from label distribution. It can be summarized as follows:

Attack loss is $J(y_t = h_\theta(x'), y)$, find a or a set of $\delta^{'}$, $y' = h_\theta\left(x + \delta^{'}\right)$, st. $(x', y') \sim D_{target}$
we would like δ to be as small as possible.

Object Detector Deception Modeling

Let's say:
convolution layer: $conv_{l(x_{l-1})} = w_{l-1}x_{l-1}+b_{l-1}$
leaky relu activation: $h_{l(x_{l-1})} = relu\_leaky(x_{l-1})$
maxpooling layer: $m_{l(x_{l-1})} = maxpool(x_{l-1})$
we can obtain their gradients by:

$$\frac{\partial conv_{l(x_{l-1})}}{\partial x_{l-1}} = {w_{l-1}}^T$$
$$\frac{\partial h_{l(x_{l-1})}}{\partial x_{l-1}} = np.where(x_{l-1} > 0, 1, 0.1x_{l-1}) = h_l$$
$$\frac{\partial m_{l(x_{l-1})}}{\partial x_{l-1}} = np.where(x_{l-1} > 0, 1, 0) = m_l$$

let's name y_t is the score to bend, we have the inference from the last conv layer.

$$y_t = w_l x_l + b_l$$

$$\begin{aligned}
dy_t &= dconv_{l+1(x_l)} = d(w_l x_l + b_l) \\
&= w_l dx_l = w_l dm_l(x_{l-1}) = w_l m_l \odot dx_{l-1} \\
&= w_l m_l \odot d\big(h_{l-1}(x_{l-2})\big) = w_l m_l \odot h_{l-1} \odot dx_{l-2} \\
&= w_l m_l \odot h_{l-1} \odot dconv_{l-2(x_{l-3})} \\
&\cdots \\
&= w_l m_l \odot h_{l-1} \odot w_{l-3} m_{l-3} \odot h_{l-4} \odot w_{l-6} \cdots dx_{img} \\
&= r_l r_{l-1} r_{l-2} \cdots r_1 dx_{img} \\
&= tr(r_l r_{l-1} r_{l-2} \cdots r_1 dx_{img})
\end{aligned}$$

➔ $\Delta y = tr\big(r_l r_{l-1} \ldots r_l \Delta x_{img}\big)$

➔ $\frac{\partial y_t}{\partial x_{img}} = (r_l r_{l-1} \ldots r_1)^T$

similarly:

➔ $\frac{\partial x_{img}}{\partial y_t} = (R_1 R_2 \ldots R_l)^T, R_i = {r_i}^{-1}$

Object Detector Deception Modeling

Δy  Δx_{img}

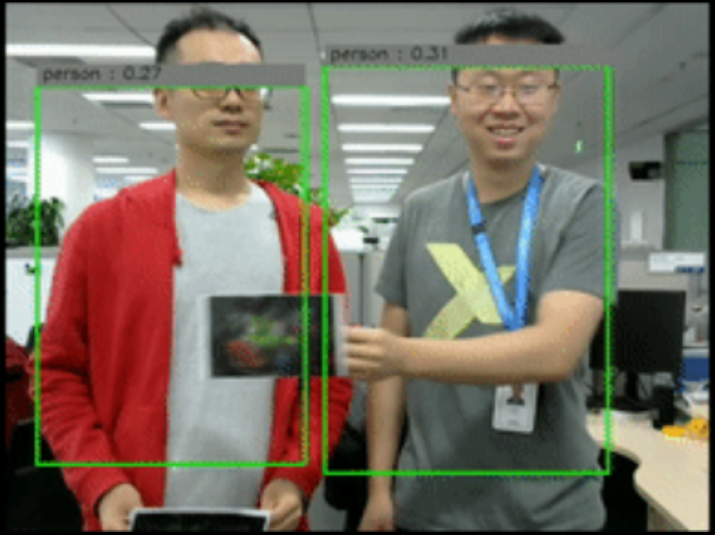
$$\frac{\partial y_t}{\partial x_{img}} = (r_l r_{l-1} \ldots r_l)^T$$

$$\frac{\partial x_{img}}{\partial y_t} = (R_1 R_2 \ldots R_l)^T$$

Object Detector Deception Modeling

Adversarial Sticker = Perturbation = $x' = x + \delta'$

VS



generated from attacking a set of images

⬇

Expectation over Testing Batch (eotb)

$$\boxed{\Delta y = tr(r_l r_{l-1} \dots r_l \Delta \overline{x}_{img})}$$

generated from attacking prediction score of a single image

⬇

Expectation of a single target case

$$\Delta y = tr(r_l r_{l-1} \dots r_l \Delta x_{img})$$

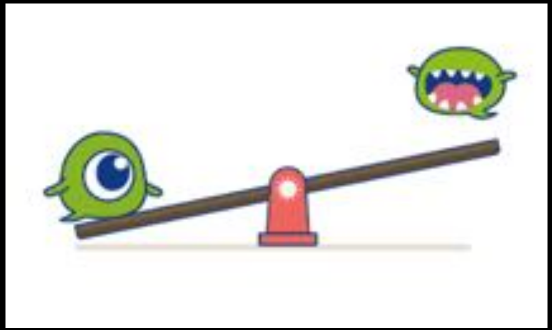
Why not use tricks in neural networks training for ODD?

Baidu X-Lab

Neural Networks Training		Object Detector Deception
Use proper weights initialization for better regression and local optimal?	inspire →	Use proper pixel value initialization for better physical deception robustness over environmental interference ?

The trick: Adversarial Initialization

Why does it work:



when training: $\uparrow \Delta y = tr(r_l r_{l-1} \ldots r_l \Delta\overline{x}_{img}) => \uparrow \Delta x_{img}$

when attacking: $\uparrow \Delta x_{img} => \uparrow \Delta y = tr(r_l r_{l-1} \ldots r_l \Delta\overline{x}_{img})$

note: Δx_{img}(perturbation) is the accumulation of model gradients

Use Adversarial Initialization to Enhance ODD

Experiment Setting:

```python
def build_graph(self, image, mode):
    assert mode=="init_model" or mode=="reuse_model"
    self.conv_1 = self.conv_layer(1,image,16,3,1,'Variable:0', 'Variable_1:0',mode=mode)
    self.pool_2 = self.pooling_layer(2,self.conv_1,2,2,mode=mode)
    self.conv_3 = self.conv_layer(3,self.pool_2,32,3,1,'Variable_2:0', 'Variable_3:0',mode=mode)
    self.pool_4 = self.pooling_layer(4,self.conv_3,2,2,mode=mode)
    self.conv_5 = self.conv_layer(5,self.pool_4,64,3,1,'Variable_4:0', 'Variable_5:0',mode=mode)
    self.pool_6 = self.pooling_layer(6,self.conv_5,2,2,mode=mode)
    self.conv_7 = self.conv_layer(7,self.pool_6,128,3,1,'Variable_6:0', 'Variable_7:0',mode=mode)
    self.pool_8 = self.pooling_layer(8,self.conv_7,2,2,mode=mode)
    self.conv_9 = self.conv_layer(9,self.pool_8,256,3,1,'Variable_8:0', 'Variable_9:0',mode=mode)
    self.pool_10 = self.pooling_layer(10,self.conv_9,2,2,mode=mode)
    self.conv_11 = self.conv_layer(11,self.pool_10,512,3,1,'Variable_10:0', 'Variable_11:0',mode=mode)
    self.pool_12 = self.pooling_layer(12,self.conv_11,2,2,mode=mode)
    self.conv_13 = self.conv_layer(13,self.pool_12,1024,3,1,'Variable_12:0', 'Variable_13:0',mode=mode)
    self.conv_14 = self.conv_layer(14,self.conv_13,1024,3,1,'Variable_14:0', 'Variable_15:0',mode=mode)
    self.conv_15 = self.conv_layer(15,self.conv_14,1024,3,1,'Variable_16:0', 'Variable_17:0',mode=mode)
    self.fc_16 = self.fc_layer(16,self.conv_15,256,'Variable_18:0', 'Variable_19:0',flat=True,linear=False,mode=mode)
    self.fc_17 = self.fc_layer(17,self.fc_16,4096,'Variable_20:0', 'Variable_21:0',flat=False,linear=False,mode=mode)
    #skip dropout_18
    self.fc_19 = self.fc_layer(19,self.fc_17,1470,'Variable_22:0', 'Variable_23:0',flat=False,linear=True,mode=mode)

    self.c = tf.reshape(self.fc_19[:,0:980],(self.fc_19[:,0:980].shape.as_list()[0],7,7,20))
    self.s = tf.reshape(self.fc_19[:,980:1078],(self.fc_19[:,980:1078].shape.as_list()[0],7,7,2))

    self.p1 = tf.multiply(self.c[:,:,:,14],self.s[:,:,:,0])
    self.p2 = tf.multiply(self.c[:,:,:,14],self.s[:,:,:,1])
    self.p = tf.stack([self.p1,self.p2],axis=0)
    self.batch_p = tf.reduce_max(self.p,2)
    self.Ctarget = tf.reduce_sum(self.batch_p)
    # self.Ctarget = tf.reduce_sum(self.p)

    return self.Ctarget
```

Cited from github user gliese581gg

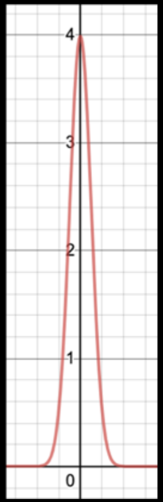
https://github.com/gliese581gg/YOLO_tensorflow/blob/master/YOLO_tiny_tf.py

Use Adversarial Initialization to Enhance ODD

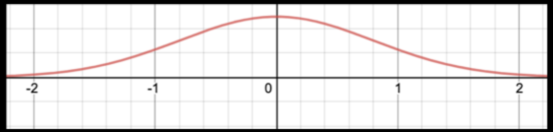
Experiment Setting:

- End to end differentiable object detector faster yolo v2
- 1200+ images containing people
- Adam optimizer lr = 1e-2
- Loss function $\text{Loss} = C_{people} + \lambda_1 \|\delta\|_2 + \lambda_2 \|d\delta\|_2$

```
# original
init_inter = tf.constant_initializer(0.001*np.random.random([1,448,448,3]))
```



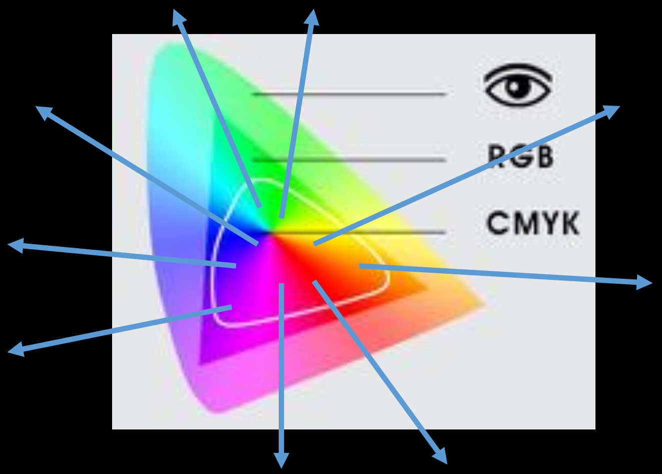

```
# improved
# think of it, we want ad sticker starts at somewhere easy
init_inter = tf.constant_initializer(0.7*np.random.normal(scale=0.8,size=[1,448,448,3]))
```

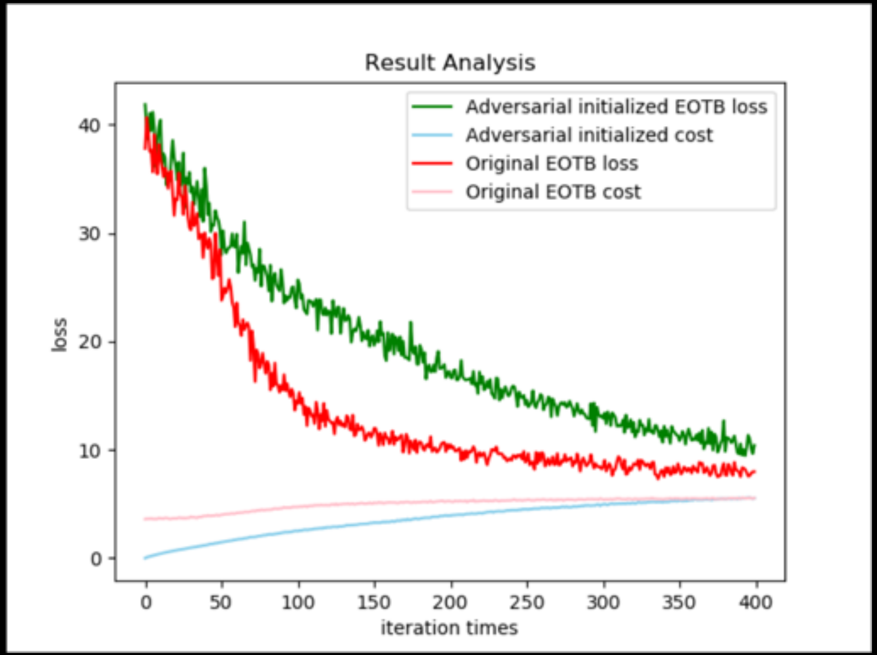


Compare Original & Adversarial Initialization

The physical adversarial robustness is not free.

- the cost of adversarial initialization: less optimal minimum
- at every iteration, larger y than original
- result in more color-saturated adversarial sticker

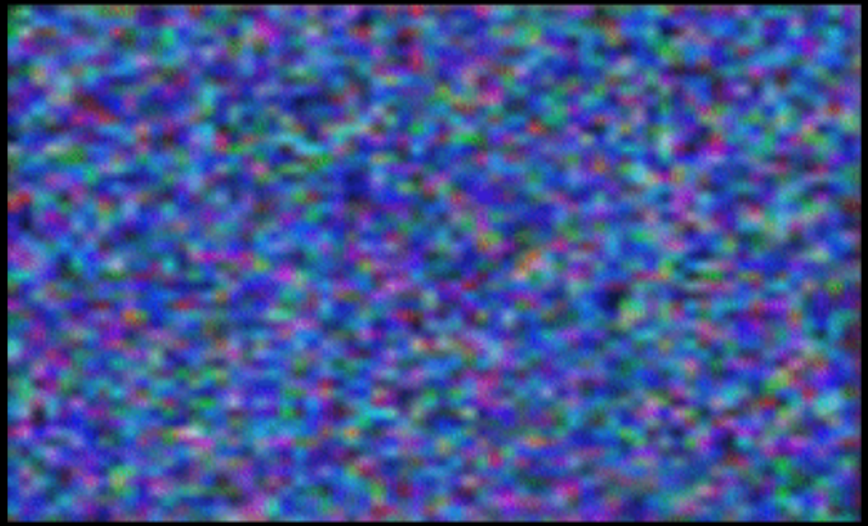




Compare Original & Adversarial Initialization

Time-lapse photography：

Original

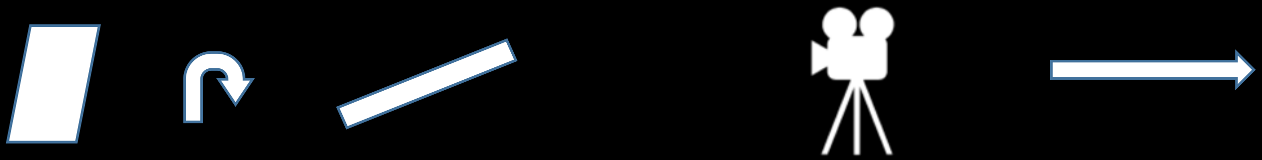


Improved

Compare Original & Adversarial Initialization

Effectiveness in physical world:

The experiment:



C_{people}

showing or blocking the sticker

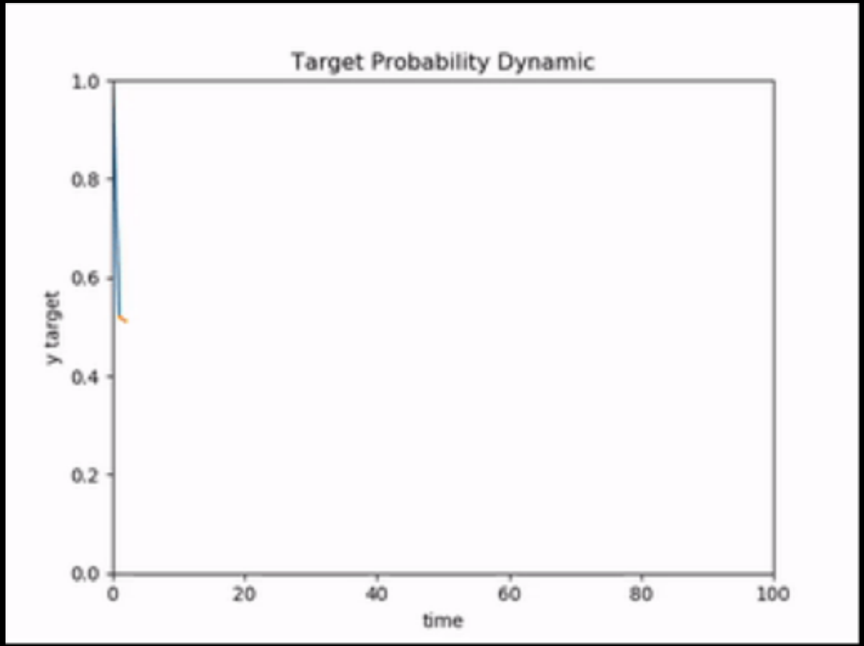
input image

faster yolo v2

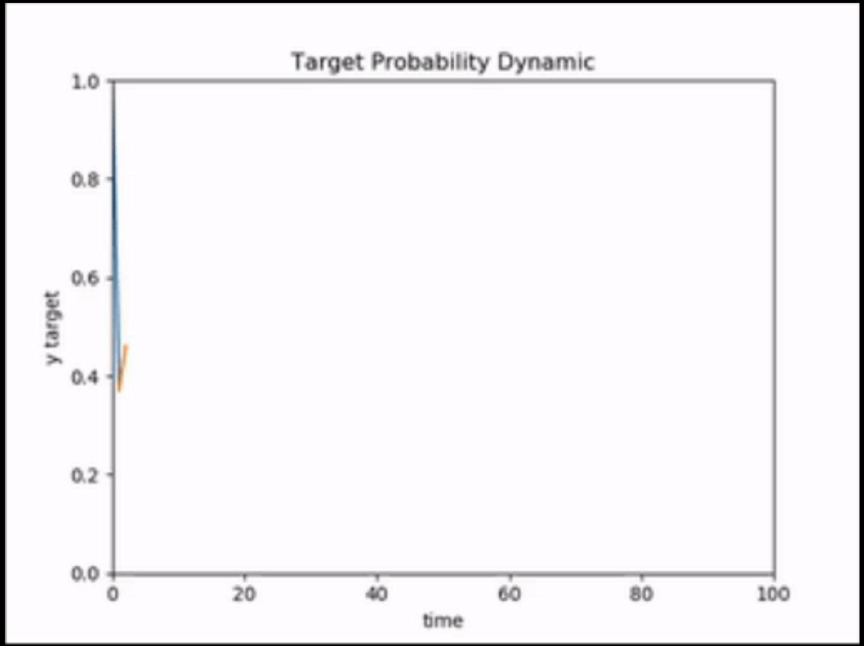
output confidence

Compare Original & Adversarial Initialization

Effectiveness in physical world:



Original



Improved

One More thing

How to craft your own adversarial sticker on your own object detector?

https://github.com/advboxes/AdvBox

THANKS

Presented at HITB GSEC 2019

Baidu X-Lab