

# VOneNet-FGSM report

김준호, 안병철

2021-04-03

# 목차

## 1. abstract

### 1.1. adversarial sample

### 1.2. FGSM

### 1.3. models

#### 1.3.1 AlexNet

#### 1.3.2 ConvNet

#### 1.3.3 Basic-CNN

#### 1.3.4 Linear-regression

## 2. generate adversarial images

### 2.1. samples

### 2.2. code

## 3. fine-tune

### 3.1. implementation

### 3.2. validation

## 4. VOneNet

### 4.1. VOneBlock 소개

### 4.2. validaion

### 4.3. VOneNet fine-tune

### 4.4. VOneNet의 adversarial image

## 1. abstract

### 1.1. adversarial sample

adversarial sample 은 model 을 혼란 시킬 목적으로 만들어진 특수한 입력으로, 신경망으로 하여금 잘못 추론하도록 한다. 사람이 보기에는 큰 차이가 없지만 신경망은 이를 제대로 식별하지 못한다.

이와 같은 신경망 공격에는 여러 종류가 있는데 white box 공격 기술에 속하는 FGSM 을 다루고자 한다.

white box 공격이란 공격자가 대상 모델의 모든 parameter 값에 접근할 수 있다는 가정 하에 이루어지는 공격을 일컫는다.<sup>1</sup>

### 1.2. FGSM

FGSM 은 pre-trained model 의 original image 에 대한 gradient 를 이용해 adversarial image 를 생성하는 기법이다. 만약 모델의 입력이 이미지라면, 입력 이미지에 대한 cost function 의 gradient 계산하여 그 손실을 최대화하는 이미지를 생성한다. 이처럼 새롭게 생성된 이미지를 adversarial image 라고 한다. 이 과정을 아래와 같은 수식으로 정리할 수 있다.

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

Figure 1

adv\_x : adversarial image

x : 원본 입력 이미지

y : 원본 입력 label

$\epsilon$  : 왜곡 정도

J : 손실 함수 (cost function)

여기서 흥미로운 사실은 입력 이미지에 대한 gradient 가 사용된다는 점이다. 여기서는 모델의 weight 는 수정하지 않는다. 입력으로 주어지는 이미지에 대한 gradient 를 구해 여기서 sign 함수를 적용하고 적절한 epsilon 을 곱해주어 더한다. 따라서 FGSM 의 궁극적인 목표는 이미 학습을 마친 상태의 모델을 혼란시키는 것이다. 아래 validation 에서 eps이라는 변수로 사용하였으며 0.35 로 통일하였다.

### 1.3. models

모델들은 모두 4 가지를 사용하였다. 하나는 AlexNet 을 조금 변형하였고, 두 개는 약간 다른 두 개의 기본적인 CNN 모델이다. 남은 하나는 가장 기초적인 linear regression 모델이다.

#### 1.3.1 AlexNet

---

<sup>1</sup> [https://www.tensorflow.org/tutorials/generative/adversarial\\_fgsm?hl=ko](https://www.tensorflow.org/tutorials/generative/adversarial_fgsm?hl=ko)

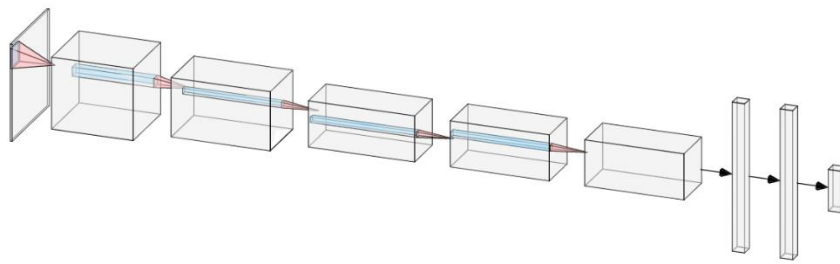


Figure 2 AlexNet

### 1.3.2 ConvNet

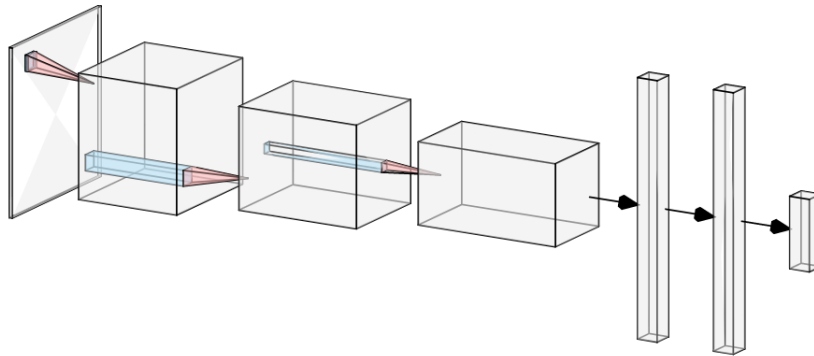


Figure 3 ConvNet

### 1.3.3 Basic-CNN

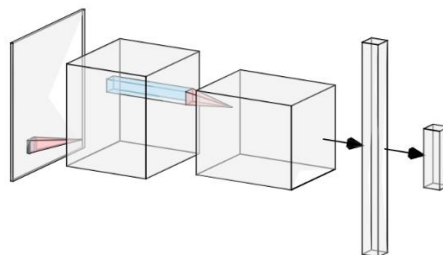


Figure 4 Basic-CNN

### 1.3.4 Linear Regression

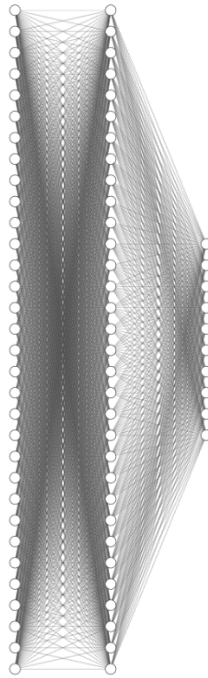


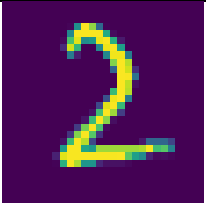
Figure 5 Linear Regression

이와 같은 hidden layer 가 하나인 기초 linear regression 이다.  
 자세한 코드는 아래 github 에 vonenet/back\_ends.py 에 모두 작성하였다.  
[https://github.com/comeeasy/FGSM\\_MNIST](https://github.com/comeeasy/FGSM_MNIST)

## 2. generate adversarial image

### 2.1. samples

위와 같은 방법으로 이미지를 생성하였는데 다음과 같이 사람은 알아볼 수 있지만 model 은 알아보기 어렵다.  
 왼쪽이 adversarial image 이고, 오른쪽이 original image 이다.

Adversarial image		Original image	
			
			

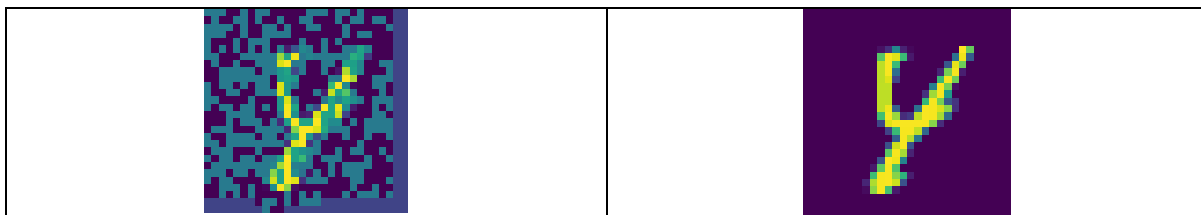


Figure 6 advarsay, original image

## 2.2. code

```

4  def generate_image_adversary(model, img_batch, target_batch, eps=0.35, device='cuda'):
5      img = img_batch.float().view(-1, 1, 28, 28).to(device)
6      label = target_batch.to(device)
7
8      img.requires_grad = True
9
10     model.zero_grad()
11     pred = model(img).to(device)
12
13     loss_fn = torch.nn.CrossEntropyLoss()
14     loss = loss_fn(pred, label).to(device)
15
16     # we need to calculate  $\nabla_x J(x, \theta)$ 
17     loss.backward()
18     img.requires_grad = False
19
20     tmp = img + eps*img.grad.data.sign()
21     tmp = torch.clamp(tmp, 0, 1)
22
23     return tmp

```

Figure 7 code

parameters 를 살펴보면, model, img\_batch, target\_batch, eps, device 가 주어진다. model 은 훈련된 모델이고, img\_batch 는 입력으로 주어지는 이미지, target\_batch 는 입력에 대한 label 이다. eps 는 얼마나 강하게 공격할 것인지에 대한 조정값이다.

line5, 6 를 보면 각 이미지를 model 에 forward 할 수 있게 적절히 변형 한다. img 에 대하여 cost 값의 gradient 를 구하므로 requires\_grad = True 로 해준다.

cost function 은 CrossEntropyLoss function 으로 하였다. 그리고 cost(loss)를 구한 뒤, 이를 backpropagation 한다. 그리고 requires\_grad = False 로 바꿔준다. 그리고 sign 함수를 적용한 뒤 eps 값을 곱하여 return 해준다.

## 2.3 accuracy

공격을 받았을 때의 이미지와 받지 않았을 때의 이미지의 accuracy 를 비교하였다.

각 모델은 AlexNet 을 약간 변형한 것과 기본적인 CNN 둘, 마지막으로 가장 기본적인 모델인 linear regression 모델을 사용하여 총 4 개의 모델로 검증하였다.

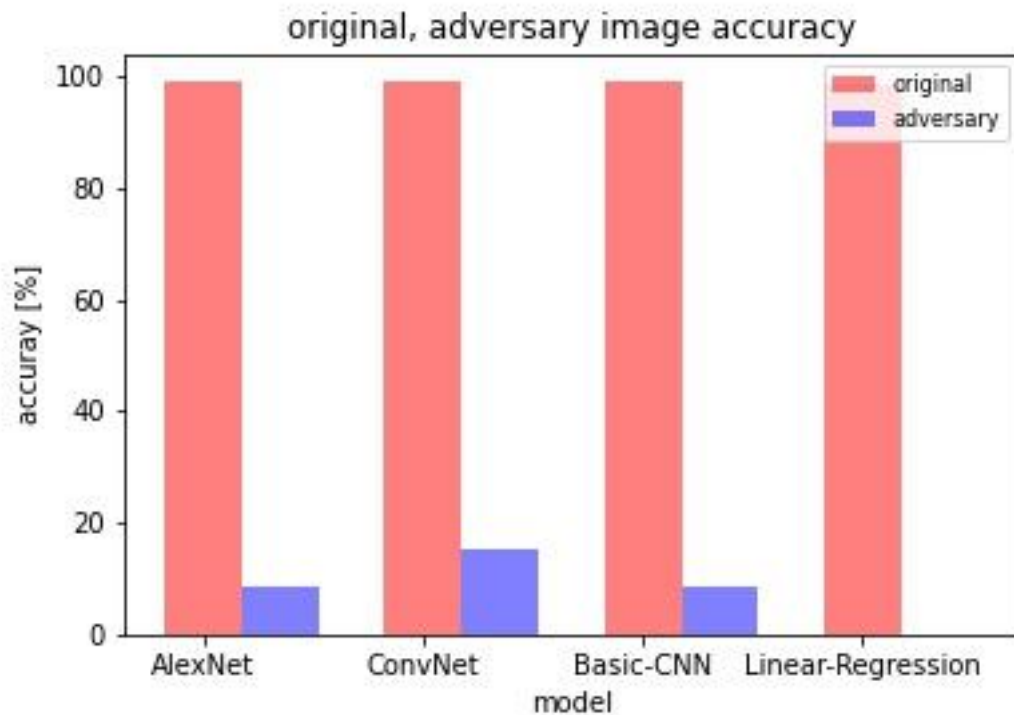


Figure 8 model accuracy 비교

위와 같이 CNN 모델들은 아주 조금 robust 한 그래프를 보여준다.

Model name	Original accuracy	Adv accuracy
AlexNet	99.048416%	8.703928%
ConvNet	98.898186%	15.074118%
Basic-CNN	99.045431%	8.473561%
Linear-Regression	98.367325%	0.060096%

### 3. fine-tune

#### 3.1. implementation

위에서 FGSM 방법으로 공격받은 이미지는 훈련된 모델이 제대로 inference 하지 못하였다.

따라서 공격받은 이미지를 기존의 훈련했던 learning rate 보다 더 작은 값으로 다시 한 번 모델을 훈련시킨다.

이를 fine-tuning 이라 하며, 이 방법을 사용해 original image 에 대한 accuracy 는 유지하고 공격받은 adversarial image 에 대한 accuracy 는 증가시키도록 하겠다.

1.3 에서 제시된 4 개의 모든 모델들은 초기에 train 할 때 learning rate 를 0.001 (1e-3)로 하였다. 이렇게 original 이미지에 대하여 10 epochs 씩을 훈련하였고, 위와 같은 결과를 얻었다.

따라서 fine-tuning 할 때는 learning rate 를 0.0001 (1e-4)로 하여 훈련을 시킬 것이고 fine-tuning 을 한 번씩 진행할 때마다 위에서 했던 방식과 똑같은 방식으로 validation 을 진행할 것이다.

## 3.2 validation

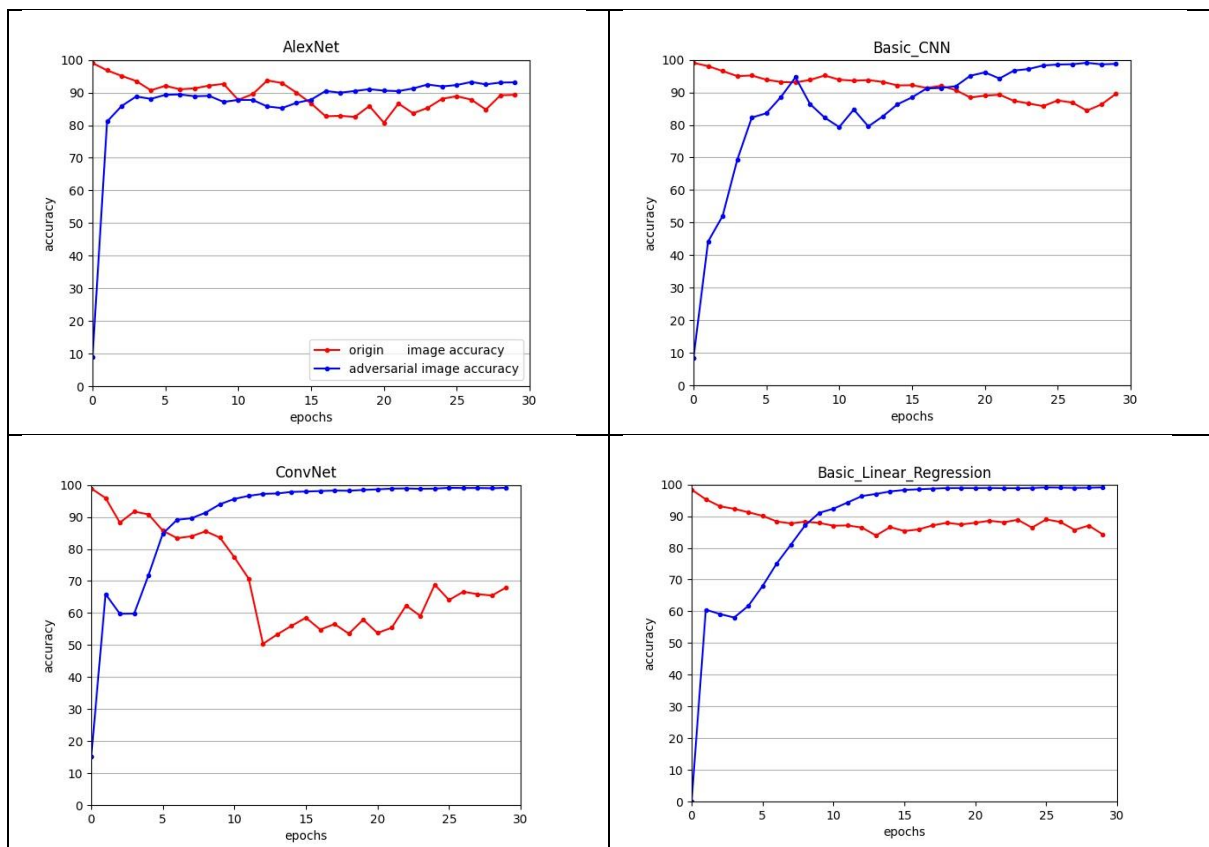


Figure 9 fine-tune validation

결과는 위와 같이 나타났다.

모두 공격받은 adversarial image 에 대해선 accuracy 가 상승하였지만 결국엔 original image 와 adversarial image 의 accuracy 가 cross 되어 original image 에 대해 제대로 인식하지 못하는 결과를 가져왔다.

## 4. VOneNet

VOneNet 은 기존의 모델의 앞에 VOneBlock 이 붙어있는 모델을 지칭한다.

### 4.1. VOneBlock 소개

VOneBlock 은 크게 gabor filter bank convolutional layer(이하 GFB)을 지나는 gabor\_f 부분과 noise 에 robust 하게 해주는 noise filter 를 지나는 noise\_f 가 있고 너무 커진 feature map 의 depth 를 줄여주는



bottleneck 으로 구성돼있다.

이는 아래와 같다.

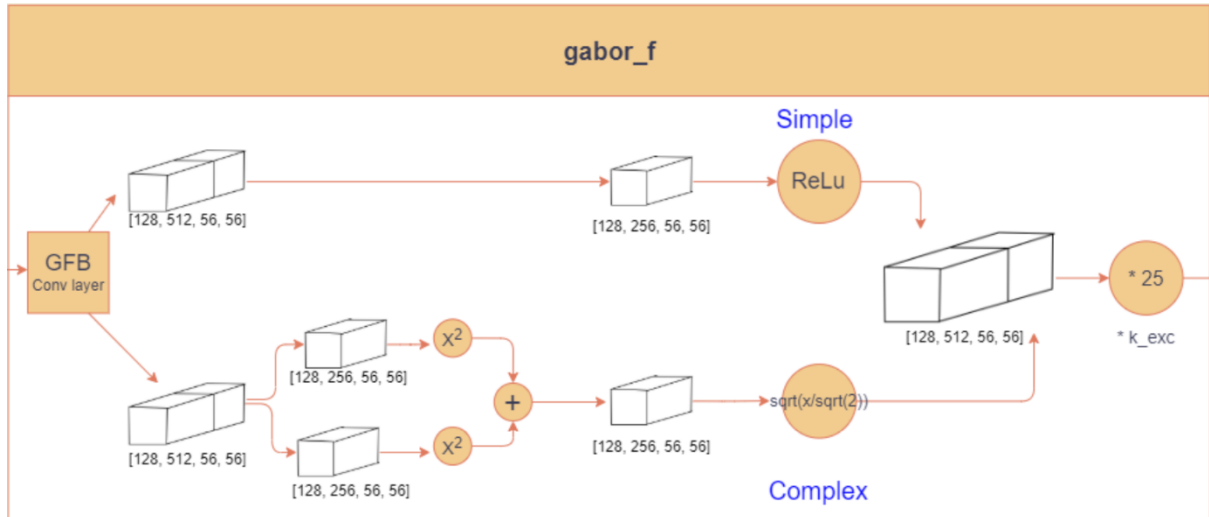


Figure 10 gabor\_f

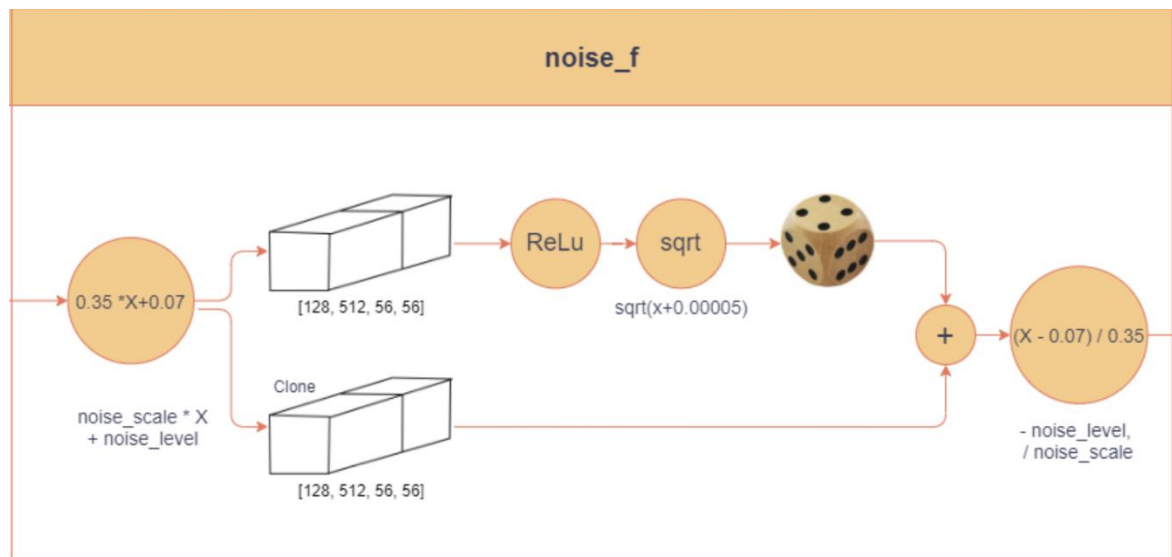


Figure 11 noise\_f

위 구조에서는 3x224x224 인 imageNet image 가 input 으로 들어가지만 MNIST 를 사용할 때는 GFB 의 kernel stride 를 1 로 수정하여 28x28 크기의 image 의 feature 도 잘 반영할 수 있도록 하였다. 따라서 최종 height, width 는 모두 28 이다.

noise\_f 를 거치며 나오는 최종 shape 은 [batch size, 512, 28, 28] 이다. 512 가 너무 크므로 아래와 같은 1x1 Convolution layer 를 거쳐 feature map 을 압축하였다.

```
bottleneck = nn.Conv2d(out_channels, in_channel, kernel_size=1, stride=1,
bias=False)
```

[batch size [in\\_channel](#), 28, 28]

위 shape 이 VOneBlock 을 통과한 feature 의 최종 shape 이며, 아래에서 진행할 validation 에서는 in\_channel 을 32 로 고정하였다.

## 4.2. validation

VOneBlock 을 붙인 1.3 에서 제시된 모든 모델을 original image 로만 각 10 epochs 씩 train 하였고, fine-tuning 을 전혀 하지 않은 상태에서 original image, adversarial image 의 accuracy 를 각각 validation 하였다.

결과는 아래와 같다.

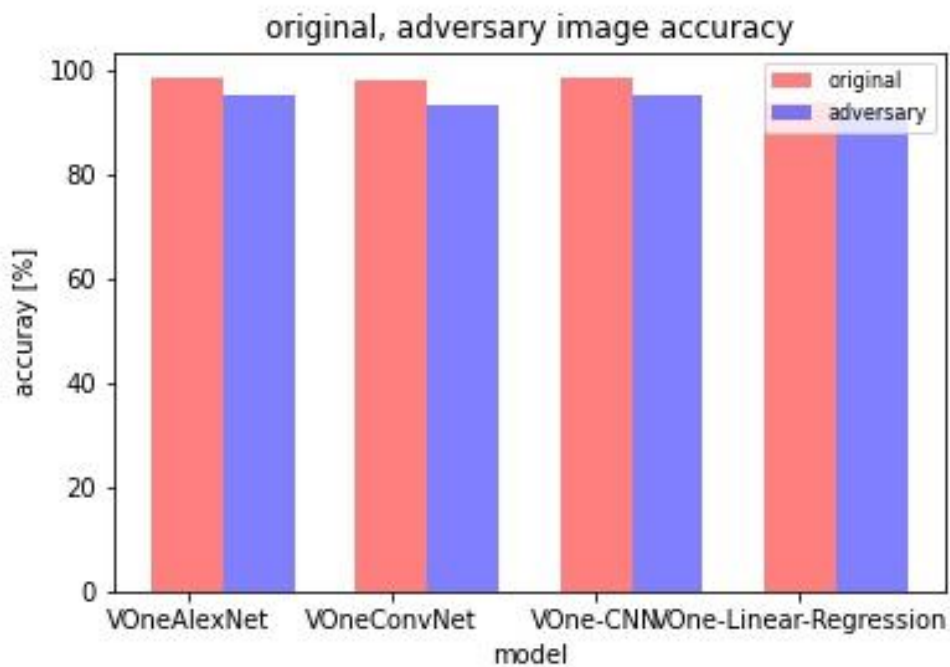


Figure 12 VOneNet validation

Model name	Original accuracy	Adv accuracy
VOneAlexNet	98.587715%	95.312485%
VOneConvNet	98.467514%	93.429474%
VOne-CNN	98.838081%	95.312469%
VOne-Linear-Regression	94.060471%	92.538033%

기존의 모델들보다 현저히 좋아진 결과를 볼 수 있다.

## 4.3. VOneNet finetune

위에서 시행하였던 fine-tuning 을 VOneBlock 을 붙인 VOneNet 들에 대해서도 시행하였다.  
결과는 아래와 같았다.

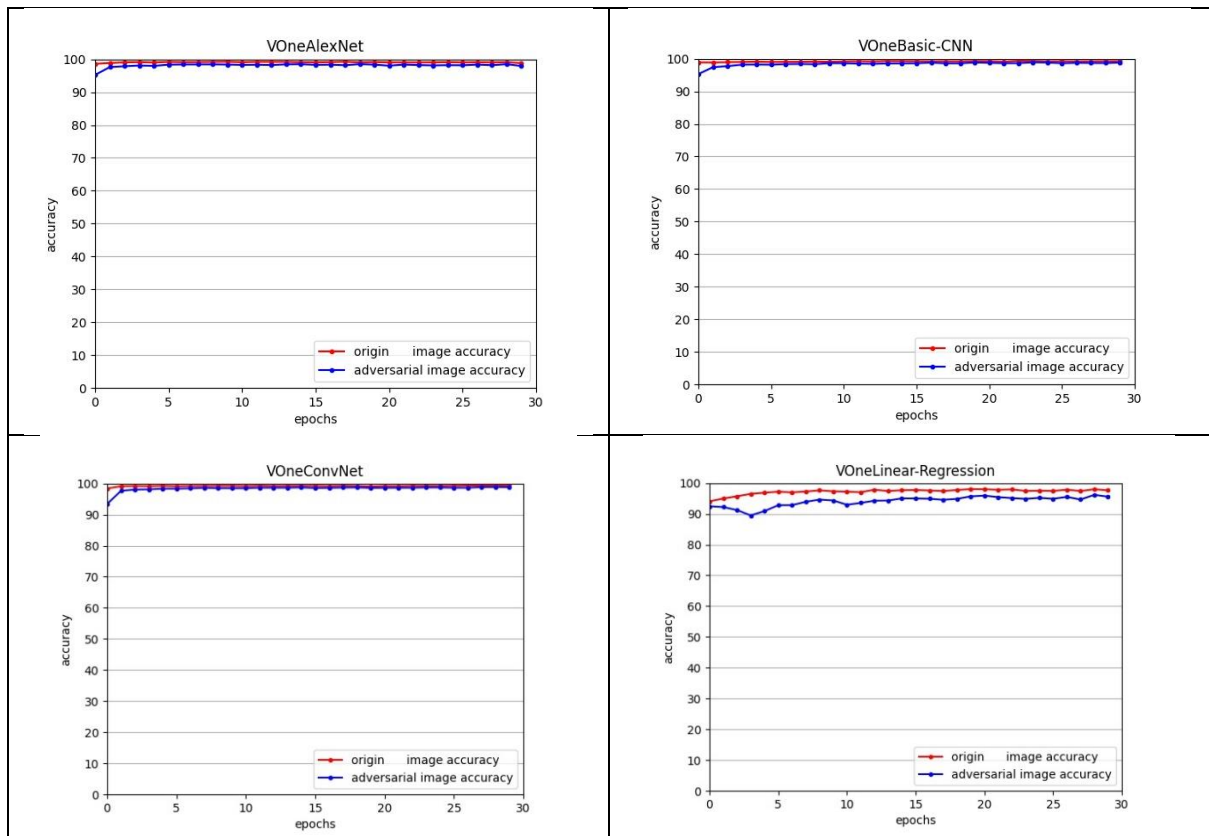


Figure 13 VOneNet finetune validation

처음에 제시하였던 traditional 한 모델들은 fine-tuning 을 거치며 original image 에 대한 accuracy 가 감소하였다. 그러나 VOneNet 들은 CNN model 들에 대해서는 거의 100%에 가까운 accuracy 를 보이며 original image, adversarial image 에 대한 accuracy 그 어느 것도 떨어지지 않는 것을 확인할 수 있다. Linear regression 모델 같이 기초적인 모델에 대해서는 accuracy 가 그렇게 좋지는 않지만 적어도 original image 의 accuracy 와 adversarial image 의 accuracy 가 cross 되지 않는다는 것에 주목할 필요가 있다.

#### 4.4. VOneNet 의 adversarial image

위와 같은 결과를 original image 로만 학습한, 즉 fine-tune 하지 않은 VOneNet 모델을 통해 adversarial image 를 위와 똑 같은 방식으로 얻어냈다.

결과는 아래와 같다. 위에서와는 달리 noise 가 사라졌거나 거의 없어진 것을 확인할 수 있다.

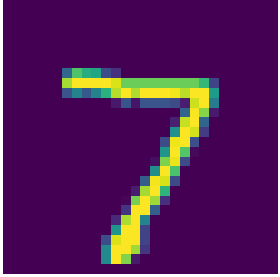
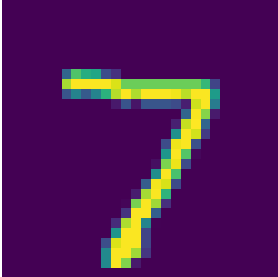
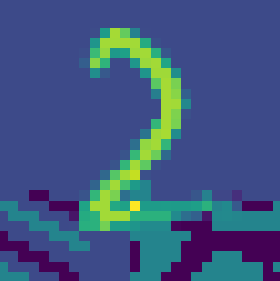
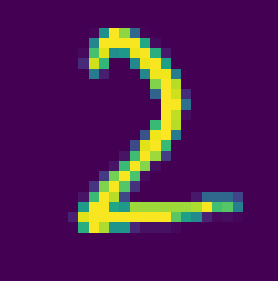
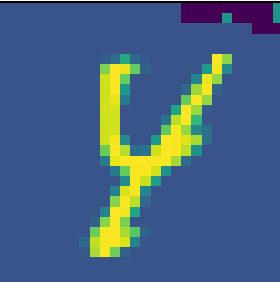
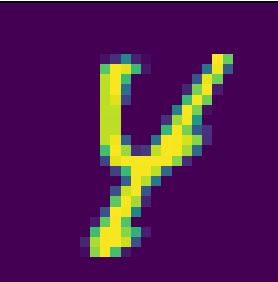
Adversarial image			Original image		
					
					
					

Figure 14 VOneNet adversarial image