Monthly Project4
CNN 기반 이미지 분류 모델의 강건성 평가

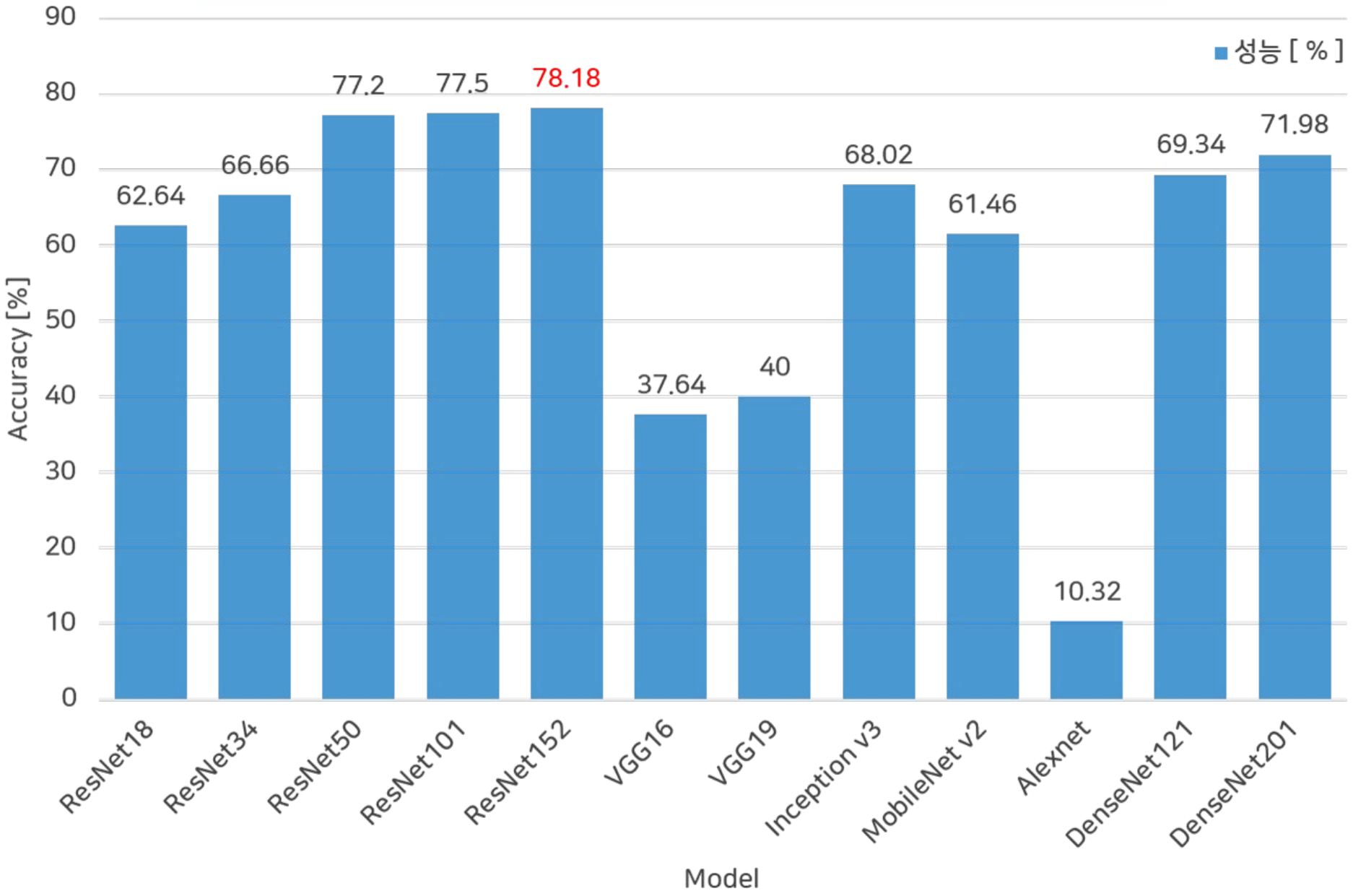
2조 - 고세규, 김예찬, 김수, 윤세환, 임동주, 정채윤

P1. 데이터셋 다운로드

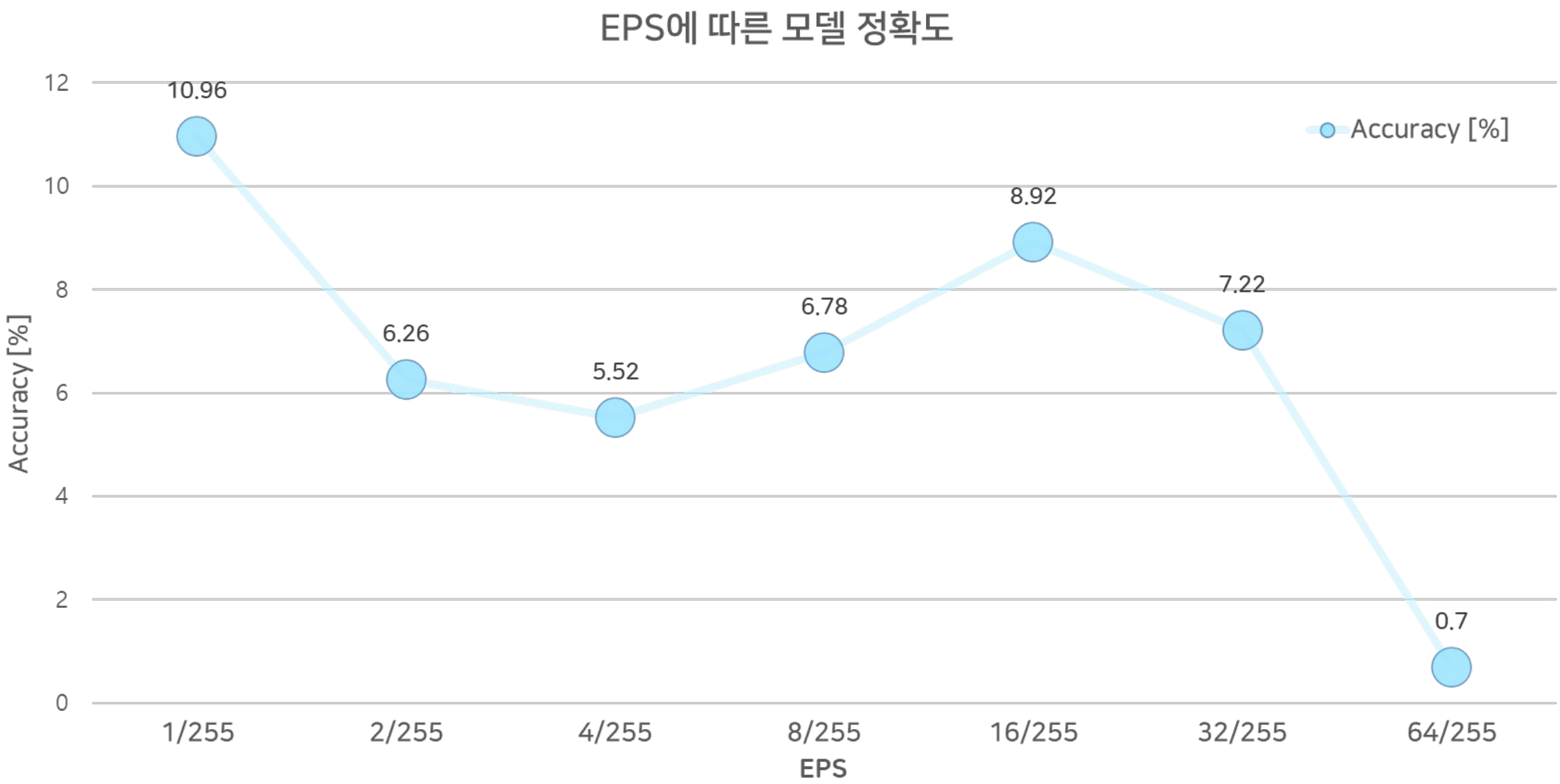
- Code

```
# 깃허브에서 데이터셋 다운로드하기
!git clone https://github.com/ndb796/Small-ImageNet-Validation-Dataset-1000-Classes
# 폴더 안으로 이동
%cd Small-ImageNet-Validation-Dataset-1000-Classes
```

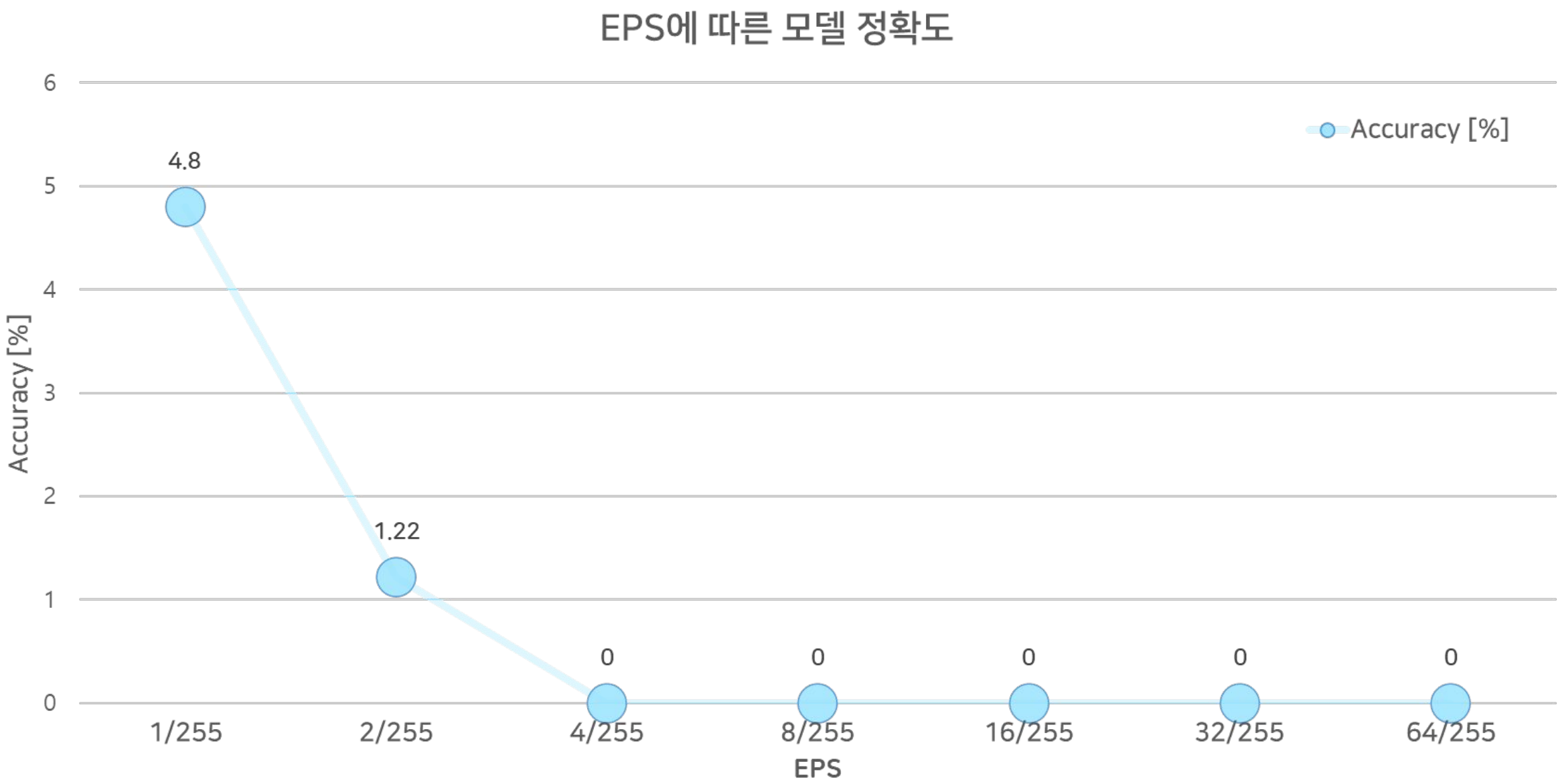
P2. 다양한 모델들의 성능 비교하기



P3. FGSM 공격 성능 분석하기



P4. PGD 공격 성능 분석하기

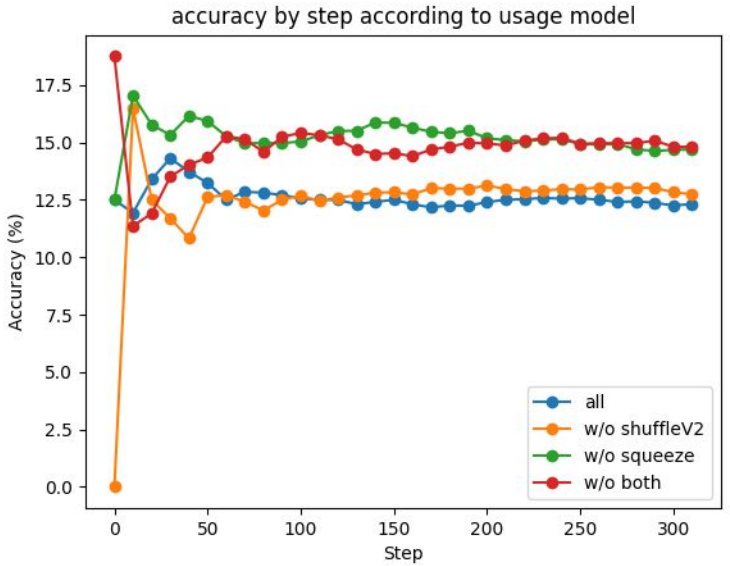


P5. 강력한 Black-box 공격 기법 설계하기

1. models에 모델의 리스트를 전달하여 다양한 모델을 학습
2. clamp를 기반으로 다양한 hyper parameter를 시도
3. momentum을 통해 poor local optima 회피
4. L1 정규화 실행

P5. 강력한 Black-box 공격 기법 설계하기

- **사용 모델 목록**
  (괄호는 imageNet에 대한 validation set 정확도)
  - efficientnet_b1 (76.88%)
  - shufflenetv2 (58.12%)
  - resnet50 (77.04%)
  - mobilenet_v2 (70.96%)
  - densenet121 (72.90%)
  - squeezenet1_1 (55.88%)



- **가설**
  - 정확도가 낮은 모델(shufflenetV2, squeezeNet) 제외 시 공격 성공률이 증가
- **결론**
  - 함께 사용하는 것이 **정확도 감소**에 효과적임.

-> 정확도가 낮은 모델이어도 해당 모델을 추가적으로 사용하는 것이 더 좋다고 판단되어,
사용하는 모델의 개수를 더 늘리는 방향으로 진행

P5. 강력한 Black-box 공격 기법 설계하기

```python
class CustomAttack_segyu:
  def __init__(self, eps, alpha, iters, beta=1.0) :
    self.eps = eps
    self.alpha = alpha
    self.iters = iters
    self.beta = beta
  def perturb(self, models, images, labels):
    # 이미지와 레이블 데이터를 GPU로 옮기기
    images = images.to(device)
    labels = labels.to(device)
    # 입력 이미지와 동일한 크기를 갖는 노이즈(perturbation) 생성
    # 노이즈 값은 음수가 될 수 있으므로, 평균이 0인 균등한(uniform) 랜덤 값으로 설정
    perturbation = torch.empty_like(images).uniform_(-self.eps, self.eps)
    perturbation.to(device)
    # 손실(loss) 함수 설정
    attack_loss = nn.CrossEntropyLoss()
    momentum =torch.zeros_like(images).detach().to(device)
    for i in range(self.iters):
      # required_grad 속성의 값을 True로 설정하여 해당 torch.Tensor의 연산을 추적
      # 모두 같은 가중치로 모델의 합을 구한다.
      if i <=1:
        used_beta = self.beta
      else:
        used_beta = self.beta/2
      for model in models:
        perturbation.requires_grad = True
        # 현재 공격 이미지(원래 이미지에 노이즈를 섞고, 그 결과가 [0, 1] 범위에 속하도록 자르기)
        current = torch.clamp(images + perturbation, min=0, max=1)
        outputs = model(current)# 모델의 판단 결과 확인
        # 손실 함수에 대하여 미분하여 기울기(gradient) 계산
        model.zero_grad()
        cost = attack_loss(outputs, labels).to(device)
        cost.backward()
        # 계산된 기울기(gradient)를 이용하여 손실 함수가 증가하는 방향으로 업데이트
        m_grad = self.beta * momentum + perturbation.grad/torch.sum(torch.abs(perturbation.grad))
        # 결과적으로 노이즈(perturbation)의 각 픽셀의 값이 [-eps, eps] 사이의 값이 되도록 자르기
        momentum += perturbation.grad
        perturbation = torch.clamp(perturbation + self.alpha * m_grad.sign(), min=-self.eps, max=self.eps).detach_()
    # (최종적으로 만들어진 공격 이미지, 노이즈) 반환
    current = torch.clamp(images + perturbation, min=0, max=1)
    return current, perturbation
```

P5. 강력한 Black-box 공격 기법 설계하기

- **Model**
  - Ensemble Model
    - ResNet50
    - ResNet101
    - ResNet152
    - ResNext50_32x4d
    - DenseNet121
    - DenseNet169
    - DenseNet201
    - EfficientNet
    - ShuffleNet v2
    - Regnet
    - MobileNet
  - Hyperparameters
    - eps : 16/255
    - alpha: 8/255
    - iters : 5
    - beta : 1.0(iter 3부터 0.5로 진행)

P5. 강력한 Black-box 공격 기법 설계하기

- **Results**
  - Accuracy: 2.88[%]
  - Average L0 distance: 143395.872
  - Average L2 distance: 22.3299
  - Average MSE: 0.0033
  - Average Linf distance: 0.0627

```
[Step #310] Loss: 0.9795 Accuracy: 2.8939% Time elapsed: 3028.3470s (total 4976 images)
[Validation] Loss: 0.9806 Accuracy: 2.8800% Time elapsed: 3044.5410s (total 5000 images)
[Size of Perturbation]
Average L0 distance (the number of changed parameters): 143395.872
Average L2 distance: 22.329934802246093
Average MSE: 0.0033337730780243872
Average Linf distance (the maximum changed values): 0.0627451241016388
```

Q&A

1. 논문과 달리 k개의 모델와 가중치의 합의 오류 역전파를 돌리는게 아닌 각각의 모델의 오류 역전파를 기반으로 perturbation을 수정하고 있는데 논문의 방법과의 차이가 어떤것인지?

2. 모델의 개수와 성능이 어느정도 비례하는 것으로 확인이 되었는데, 모델의 성능을 좌우하는 요인이 모델의 다양성이 절대적인 것인지, 아니면 다른 더 좋은 최적화 방법이 있는 것인지?