**차량지능기초 HW02**

# <Pytorch(torchvision)에서 제공되는 Pre-trained Model 성능 비교>

학부: 소프트웨어학부 학번: 2019260 이름: 심혜린

Github Link: https://github.com/ShimHyerin/2021-VehicleIntelligence/tree/main/HW2

본 과제에서는 Pytorch에서 제공되는 Pre-trained Model의 성능 비교를 목표로 하며, 성능 비교에 사용할 Dataset은 ImageNet Validation Set이다. 먼저 각 모델에 대한 특징에 대한 설명 후 성능 비교 과정과 결과를 첨부하였다.

## <모델 특징>

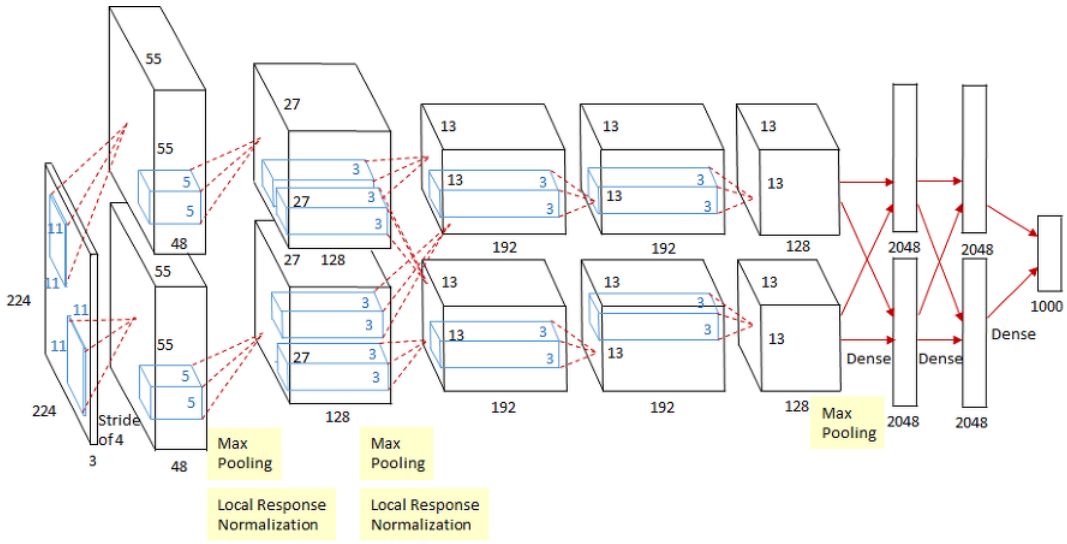
본 과제에서 사용 및 조사된 모델은 다음과 같다.

1. AlexNet
2. VGG16
3. ResNet18
4. GoogLeNet

위 모델들은 CNN(Convolutional Neural Network, 합성곱 신경망)이며, CNN은 크게 Convolution layer(합성곱층)와 Pooling layer(풀링층)의 조합으로 구성된다. 이런 구성을 가지고 있는 모델을 통틀어 CNN 모델이라고 할 수 있다. CNN은 정규화와 같은 효과를 내기 위해 데이터에서 계층적 패턴을 활용하며 더 작고 간단한 패턴을 사용하여 더욱 복잡한 패턴을 표현한다. CNN 연결구조의 복잡성은 다층 퍼셉트론에 비교하면 극단적으로 낮고, CNN을 활용한 영상 분류는 전처리를 거의 사용하지 않는다는 장점이 있다.

### *# AlexNet*

CNN의 모델 중 하나인 AlexNet은 인공지능의 Classification 대회인 ILSVRC에서 2012년에 당시 오차율 16.4%로 다른 모델들을 압도하고 우승한 모델이다. 당시에는 굉장한 정확도와 성능을 보여주었다. AlexNet의 'Alex'는 모델 논문의 저자인 Alex Khrizevsky의 이름을 따온 것이다.




**# 구조**

AlexNet은 총 8개의 층으로 구성되어 있다. 첫 5개 층은 Convolution, 그 뒤 3개 층은 Fully-Connected 층이다. 1, 2층은 Max Pooling층으로 데이터의 중요한 요소들만 요약하여 추출하고 3, 4, 5층은 중간다리역할로 서로 연결되어 있다. 여기까지 Convolution층이다. 6층부터는 Max Pooling층이 뒤따르는데, 1층부터 5층까지 학습된 데이터들의 중요한 요소만 요약하여 2개로 구성된 Fully Connected층에서 분류된다.

**# 특징**

**Overlapping Pooling Layer**: Stride를 좁혀 Overlapping 하는 구조로 정확도가 상승시킨다. 그러나 overlapping사용으로 인해 연산량은 증가한다.

**ReLU Activation Function**: 활성화함수로, 학습 속도를 최대 6배 상승시킨다.

**Local Response Normalization(LRN)**: AlexNet에서 처음 도입된 기법으로, 특정 특징을 부각시켜주어 정확도를 향상시킨다.

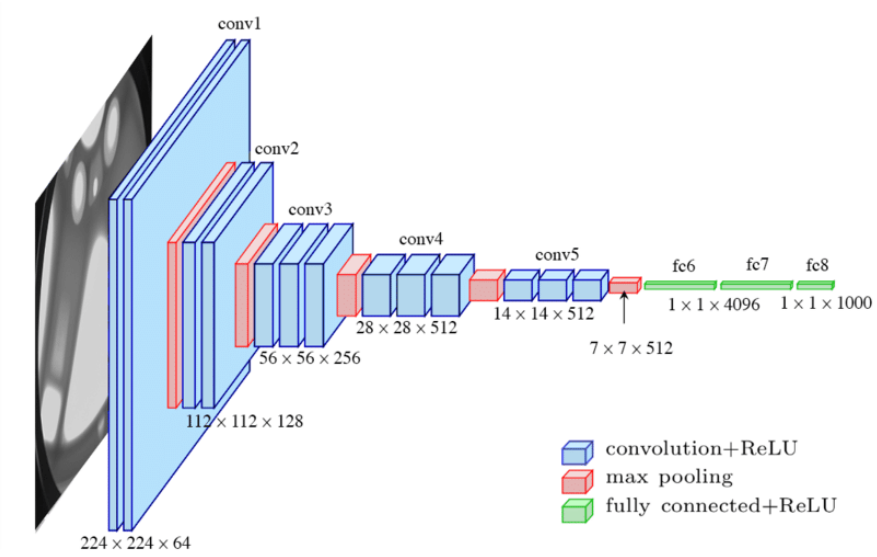
**Dropout**: 신경망 사이의 연결을 랜덤하게 끊어 overfitting을 줄여주어 에러 확률을 줄였으나 학습시간은 약 2배 증가되었다.

**Training on Multiple GPUs**: 학습 시간을 줄이기 위해 사용되었다. 여러 개의 GPU를 사용하는 병렬처리 방식이다.

## *# VGG16*

VGGNet은 옥스포드 대학의 연구팀 VGG에 의해 개발된 모델이며, 2014년 이미지넷 이미지 인식 대회에서 준우승을 한 모델로 16개층으로 구성된 VGG16, 19개 층으로 구성된 VGG19가 있다.

**# 구조**




Convolution층 사이사이 Max Pooling층이 존재하여 각 층의 중요 데이터를 추출한 후 Fully Connected층에서 분류된다.

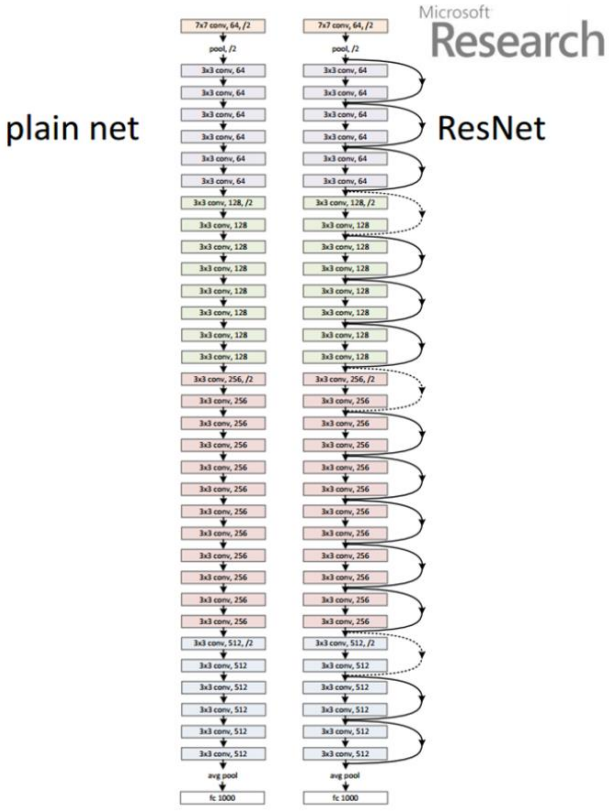
**# 특징**

VGG16의 모든 필터는 3*3, stride = 1으로 고정되어 있다. 필터의 사이즈를 작게 만들면 더 큰 크기의 필터와 같은 효과를 가지지만 더 적은 파라미터가 나와 학습의 효율이 올라간다는 장점이 생긴다. 필터의 사이즈가 클수록 이미지의 크기가 줄어드는 것이 빠르기 때문에 이를 방지하기 위함도 있다.

## *# ResNet18*

ResNet은 2015년 ILSVRC에서 우승을 차지한 모델로 마이크로소프트에서 개발했다. 기본적으로 VGG19의 구조를 기반으로 하고 있으며, VGG19에 Convolution층을 추가하여 더욱 깊게 만들고 shortcut를 추가하여 만들어졌다.

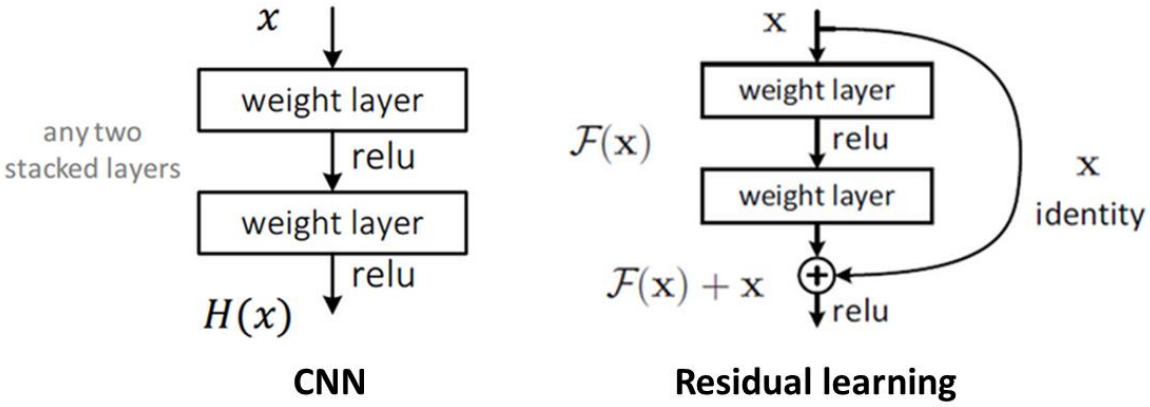
**# 구조**




점선과 실선은 shortcut 연결로, 점선은 특성맵의 크기를 줄일 때 사용되는 shortcut, 실선은 입력으로 들어온 특성맵의 크기를 변화시키지 않도록 Stride와 Zero padding을 1로 하고 (1x1) 크기의 필터를 사용하는 shorcut다.

**# 특징**

**Residual Learning**

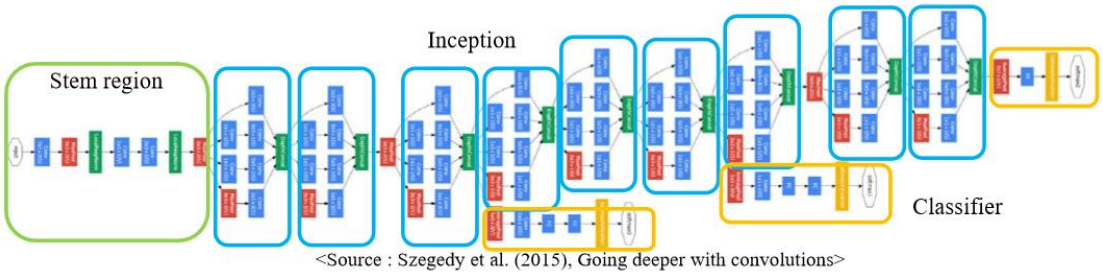



Shortcut 연결로 정보의 전달이 한 층을 건너 뛰어 연결되지 않은 2개의 층을 연결해준다. 그림의 F(x) + x를 최소화하는 것이 목적으로, F(x)를 0에 수렴하세 만드는 것이 최종 목표다.

## *# GoogLeNet*

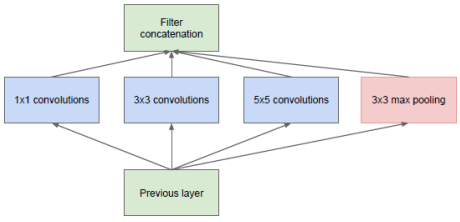
GoogleNet은 2014년 이미지넷 이미지 인식 대회에서 우승을 차지한 모델이다. 22층으로 구성되어 있으며, 구글에서 개발되었다.

**# 구조**

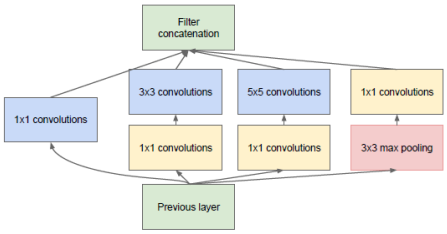


<Source : Szegedy et al. (2015), Going deeper with convolutions>

저층구간(사진에서 Stem region 부분)은 인셉션 모듈은 저층에서의 학습이 비효율적이기 때문에, 인셉션 모듈을 효율적으로 사용할 수 있을 때까지 일반적인 CNN기법을 활용하여 학습을 하는 구간이다.



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

인셉션 구간은 일반적인 CNN의 convolution과 pooling층으로 이루어지지 않고 local receptive field라는 개념이 추가된다. 동일한 사이즈의 필터 커널을 이용해서 convolution을 하지 않아 다양한 종류의 특성이 도출된다. 1*1 convolution 또한 포함되어 연산량도 매우 줄어들게 된다.

**#특징**

**Auxiliary classifier**

네트워크 중간에 2개의 보조 분류기를 추가해주었다. 네트워크의 깊이가 깊어질 때, 역전파 과정에서 가중치를 업데이트 할 때 gradient가 0으로 수렴하는 vanishing gradient 문제를 극복하기 위함이다.

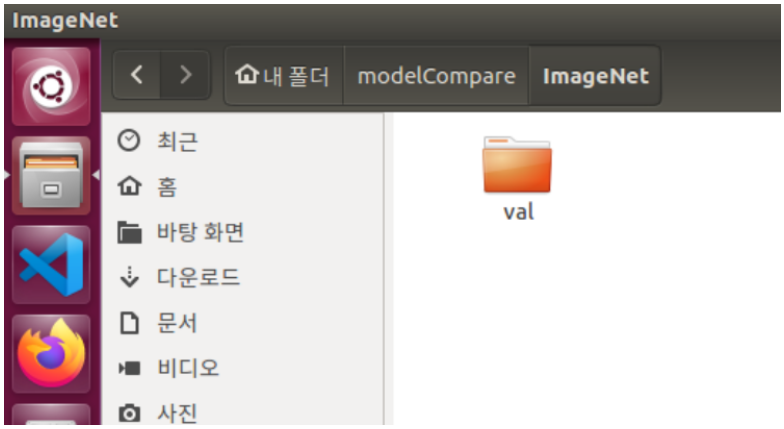
**Global average pooling**

전 층에서 산출된 특성맵을 각각 평균내고, 그것을 이어 1차원 벡터를 만들어 주는 기법으로 가중치의 개수를 상당히 많이 없애준다는 큰 장점이 있다. 여기서 1차원 벡터로 만드는 이유는 최종적으로 이미지 분류를 위한 softmax층은 1차원 벡터로 연결할 수 있기 때문이다. 구글넷은 일반적인 방법인 Fully connected층이 후반후에 연결되어 있는 방식과 달리 Global average pooling 방식을 사용한다.

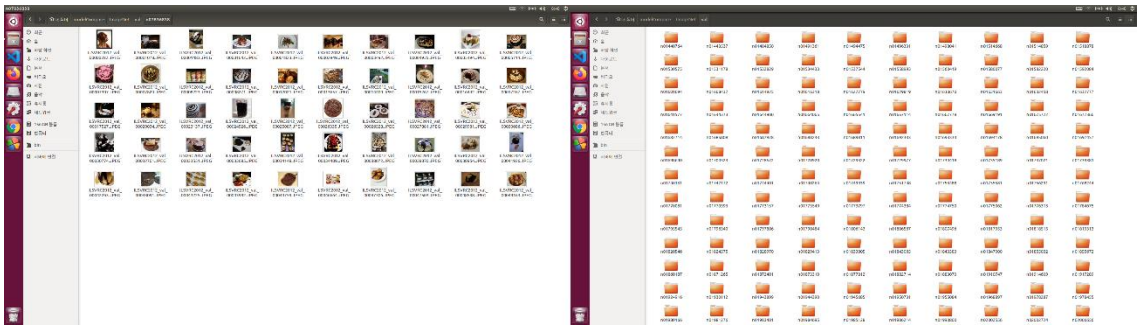
***<모델 성능 비교>***

**# DataSet**

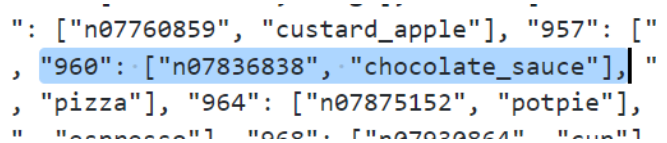
성능 비교를 위해 ImageNet Validation Set을 사용하였다. DataSet 다운로드 후 압축을 풀면 5만장의 이미지 파일이 다운로드 된다. 이후 이 경로에서 스크립트 파일을 다운로드 받고 스크립트를 실행하면 1000개의 클래스로 분류가 된다. 폴더 명에 따른 실제 클래스의 이름 label은 JSON 파일에 저장해 두었다.




(ImageNet Validation Set 다운로드 후 압축 해제한 모습)



(이미지 클래스별로 폴더에 분류)




(imagenet_class_index.json 파일 일부 캡쳐)

분류한 이미지 폴더명에 맞게 JSON 파일에 라벨링 된 것을 확인할 수 있다. 예시로 n07836838 폴더에 있는 이미지 데이터들이 초콜릿 소스가 뿌려진 이미지임을 확인하였다.

## <Code>

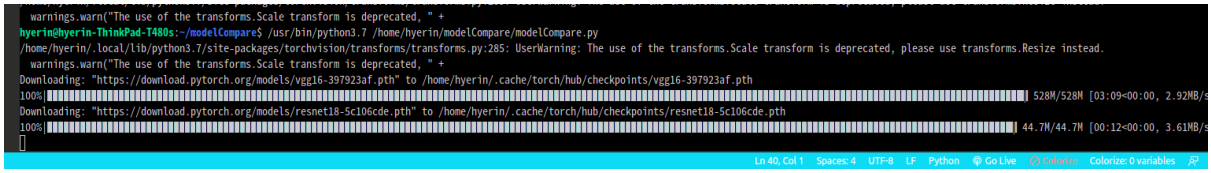
성능 비교 지표는 Top-1 Accuracy(정확도)이다. Top-5 Accuracy도 함께 계산하였으며 Top-1 accuracy는 softmax의 output에서 제일 높은 수치를 가지는 값이 정답인 경우에 대한 지표로, 예측한 class가 정답인 비율이고 Top-5 accuracy는 softmax의 output에서 높은 5개의 class 중에 예측한 클래스가 존재하는 비율이다. 성능 비교를 위해 사용된 코드는 다음과 같다. 결과 값은 txt 파일 및 그래프 이미지로 저장하였다.

```
import torch
import torchvision
import torch.utils.data as data
import torchvision.transforms as transforms
import os
import sys
from matplotlib import pyplot as plt

sys.stdout = open('modelCompareResFin.txt', 'w')

if __name__ == "__main__":
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])

    # dataset path
    valdir = '/home/hyerin/modelCompare/ImageNet/val'
    # dataset load
    val_set = torchvision.datasets.ImageFolder(valdir, transforms.Compose([
            transforms.Scale(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize,
            ]))
    val_loader = torch.utils.data.DataLoader(val_set, batch_size=128, shuffle=True
, num_workers=4)
```

```
    resTop1 = []
    resTop5 = []

    # models
    alexnet = torchvision.models.alexnet(pretrained=True).to(device)
    vgg16 = torchvision.models.vgg16(pretrained=True).to(device)
    resnet18 = torchvision.models.resnet18(pretrained=True).to(device)
    googlenet = torchvision.models.googlenet(pretrained=True).to(device)
    models_name = ['AlexNet', 'VGG16', 'ResNet', 'googleNet']
    models = [alexnet, vgg16, resnet18, googlenet]


    for i in range(4):
        print("\n---------Model :: {} ---------\n".format(models_name[i]))
        model = models[i]
        model.eval()

        top1 = 0
        top5 = 0
        total = 0

        with torch.no_grad():
            for idx, (images, labels) in enumerate(val_loader):

                images = images.to(device)
                labels = labels.to(device)
                outputs = model(images) # batch_size eval

                # rank 1
                _, pred = torch.max(outputs, 1)
                total += labels.size(0)
                top1 += (pred == labels).sum().item()


                # rank 5
                _, rank5 = outputs.topk(5, 1, True, True)
                rank5 = rank5.t()
                correct5 = rank5.eq(labels.view(1, -1).expand_as(rank5))
                correct5 = correct5.contiguous()

                for k in range(6):
                    correct_k = correct5[:k].view(-1).float().sum(0, keepdim=True)
                top5 += correct_k.item()

                print("step : {} / {}".format(idx + 1, len(val_set)/int(labels.siz
e(0))))
                print("top-1 percentage :  {0:0.2f}%".format(top1 / total * 100))
                print("top-5 percentage :  {0:0.2f}%".format(top5 / total * 100))
```

```
        print("\n---------Result :: {} ---------\n".format(models_name[i]))
        print("top-1 percentage :  {0:0.2f}%".format(top1 / total * 100))
        print("top-5 percentage :  {0:0.2f}%".format(top5 / total * 100))
        print("----------------------------------------\n\n")

        # res store
        resTop1.append(top1/total*100)
        resTop5.append(top5/total*100)

    sys.stdout.close()

    # output store
    f = open('resFinal.txt', 'w')

    for i in range(4):
        print('model :: {}'.format(models_name[i]), file=f)
        print('------------------------------\n',file=f)
        print('top-1 accuracy :: {0:0.2f}%'.format(resTop1[i]), file=f)
        print('top-5 accuracy :: {0:0.2f}%'.format(resTop5[i]), file=f)
        print('\n------------------------------\n\n\n',file=f)

    f.close()

# draw Graph
def create_x(t, w, n, d): # numberOfData, BarWidth, numOfCurrentData, numOfDataLen
    return [t*x + w*n for x in range(d)]
top1_x = create_x(2, 0.8, 1, 4)
top5_x = create_x(2, 0.8, 2, 4)
ax = plt.subplot()
ax.bar(top1_x, resTop1, color='salmon', label='top-1 accuracy')
ax.bar(top5_x, resTop5, color='silver', label='top-5 accuracy')
x = [(a+b)/2 for (a,b) in zip(top1_x, top5_x)]
ax.set_xticks(x)
ax.set_xticklabels(models_name)
ax.set_xlabel('Model')
ax.set_ylabel('Accuracy percentage')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.1), ncol=2)
plt.savefig('modelCompareGraph.png', format='png', dpi=300)
plt.show()
```

처음 코드를 실행하면 밑의 사진과 같이 model이 다운로드 된다.




이후 코드가 실행되면서 txt 파일에 모델의 성능을 조사한 결과 값을 저장해 나간다. 자세한 성능 조사 수행 과정을 기록한 txt 파일은 Github의 이 링크에서 확인할 수 있다.

```
---------Model :: AlexNet ---------

step : 1 / 390.625
top-1 percentage :  52.34%
top-5 percentage :  76.56%
step : 2 / 390.625
top-1 percentage :  52.34%
top-5 percentage :  77.34%
step : 3 / 390.625
top-1 percentage :  54.95%
top-5 percentage :  76.82%
step : 4 / 390.625
top-1 percentage :  55.08%
top-5 percentage :  77.34%
step : 5 / 390.625
top-1 percentage :  54.53%
top-5 percentage :  78.28%
step : 6 / 390.625
top-1 percentage :  54.43%
top-5 percentage :  78.12%
step : 7 / 390.625
top-1 percentage :  54.58%
top-5 percentage :  78.46%
step : 8 / 390.625
top-1 percentage :  55.47%
top-5 percentage :  79.39%
```

```
---------Model :: VGG16 -------

step : 1 / 390.625
top-1 percentage :  75.00%
top-5 percentage :  90.62%
step : 2 / 390.625
top-1 percentage :  75.00%
top-5 percentage :  91.41%
step : 3 / 390.625
top-1 percentage :  73.18%
top-5 percentage :  90.62%
step : 4 / 390.625
top-1 percentage :  69.92%
top-5 percentage :  90.23%
step : 5 / 390.625
top-1 percentage :  69.69%
top-5 percentage :  89.84%
step : 6 / 390.625
top-1 percentage :  70.18%
top-5 percentage :  90.23%
step : 7 / 390.625
top-1 percentage :  71.21%
top-5 percentage :  90.18%
step : 8 / 390.625
top-1 percentage :  71.58%
top-5 percentage :  90.53%
```

```
---------Model :: ResNet ---------

step : 1 / 390.625
top-1 percentage :  72.66%
top-5 percentage :  92.19%
step : 2 / 390.625
top-1 percentage :  70.70%
top-5 percentage :  89.06%
step : 3 / 390.625
top-1 percentage :  68.75%
top-5 percentage :  88.02%
step : 4 / 390.625
top-1 percentage :  69.14%
top-5 percentage :  88.28%
step : 5 / 390.625
top-1 percentage :  69.69%
top-5 percentage :  88.59%
step : 6 / 390.625
top-1 percentage :  68.88%
top-5 percentage :  88.80%
step : 7 / 390.625
top-1 percentage :  68.42%
top-5 percentage :  88.39%
step : 8 / 390.625
top-1 percentage :  68.95%
top-5 percentage :  88.77%
```

```
---------Model :: googleNet ---------

step : 1 / 390.625
top-1 percentage :  65.62%
top-5 percentage :  87.50%
step : 2 / 390.625
top-1 percentage :  65.23%
top-5 percentage :  88.67%
step : 3 / 390.625
top-1 percentage :  68.23%
top-5 percentage :  89.84%
step : 4 / 390.625
top-1 percentage :  69.92%
top-5 percentage :  91.41%
step : 5 / 390.625
top-1 percentage :  69.84%
top-5 percentage :  90.78%
step : 6 / 390.625
top-1 percentage :  69.14%
top-5 percentage :  90.36%
step : 7 / 390.625
top-1 percentage :  69.42%
top-5 percentage :  90.18%
step : 8 / 390.625
top-1 percentage :  68.85%
top-5 percentage :  89.36%
```

**# 결과**

4개의 모델의 성능을 조사한 결과 다음과 같은 결과가 나타났다.

model :: AlexNet

---

top-1 accuracy :: 56.52%

top-5 accuracy :: 79.07%

model :: VGG16

---

top-1 accuracy :: 71.59%

top-5 accuracy :: 90.38%

model :: ResNet

---

top-1 accuracy :: 69.76%

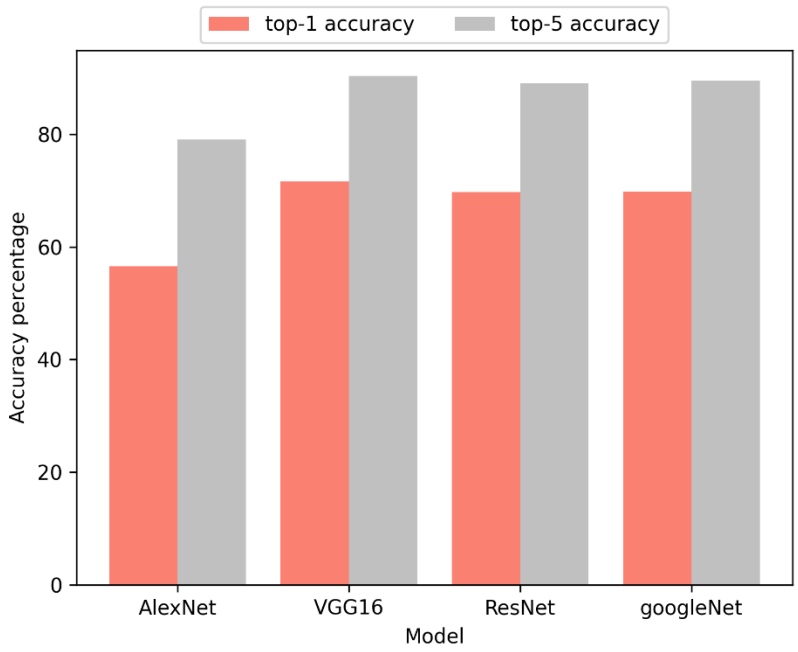
top-5 accuracy :: 89.08%

model :: googLeNet

---

top-1 accuracy :: 69.78%

top-5 accuracy :: 89.53%




top-1 accuracy Result

VGG16 (71.59%) > googleNet (69.78%) > ResNet(69.76%) > AlexNet (56.52%)

위의 결과를 그래프로 그리면 다음과 같다. 그래프는 python의 matplotlib을 사용하였다. Alexnet은 다른 model에 비해 조금 낮은 정확도를, 나머지 model은 VGG16이 가장 높지만 서로 유사한 정확도를 보여주고 있는 것을 확인할 수 있다.