IT 실험 프로젝트

CNN(convolutional neural network)기법을 활용한 버스 분류

Bus classification by using CNN method

지도교수: 임종태 교수님

작성자 : B515171 임형구

gudrn7869@naver.com

참고: 학위 논문으로 제출함.

1. 연구 배경

1.1 시각 장애인들의 불편함

✓ 일상 생활 중 버스를 탈 때 가장 많이 불편함을 느낌

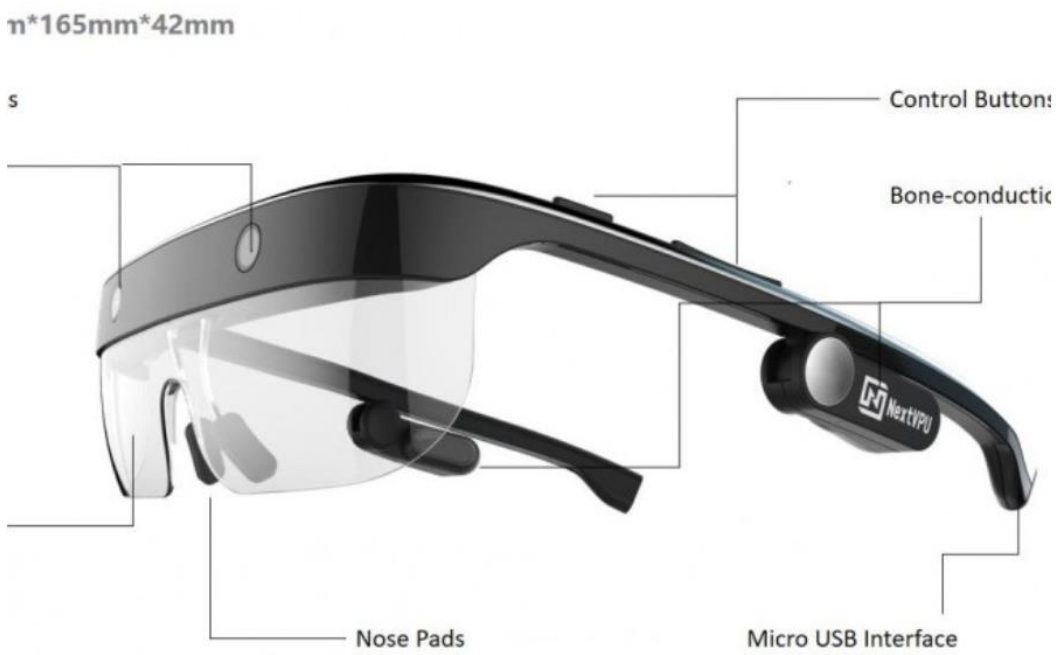
교통수단별 장애인 이용가능 비율

구분	버스	지하철	택시
시각장애인	21.1% (안내정보 부족)	63.2% (승차 후 착석 힘듦)	49.2% (택시 승하차 곤란)

*서울시정개별연구원(교통약자 이동편의 증진 계획)

1. 연구 배경

1.2 웨어러블 디바이스의 발전



*Orcam 사의 스마트 안경

2. 연구 주제

2.1 스마트 디바이스에 함께 활용될 버스 안내 기능 제안

- ✓ 버스의 존재 여부 확인(실험 1 busornot)
- ✓ 시각 장애인이 이용 할 특정 버스의 존재 여부 확인(실험 2 mapo08ornot)

*머니투데이 강남역 사진

2. 연구 주제

2.2 실험1 버스의 존재 여부 (busornot)

✓ 버스가 도착한 경우

✓ 버스가 도착하지 않은 경우

2. 연구 주제

2.3 실험2 특정버스의 도착 여부 확인 (mapo08ornot)

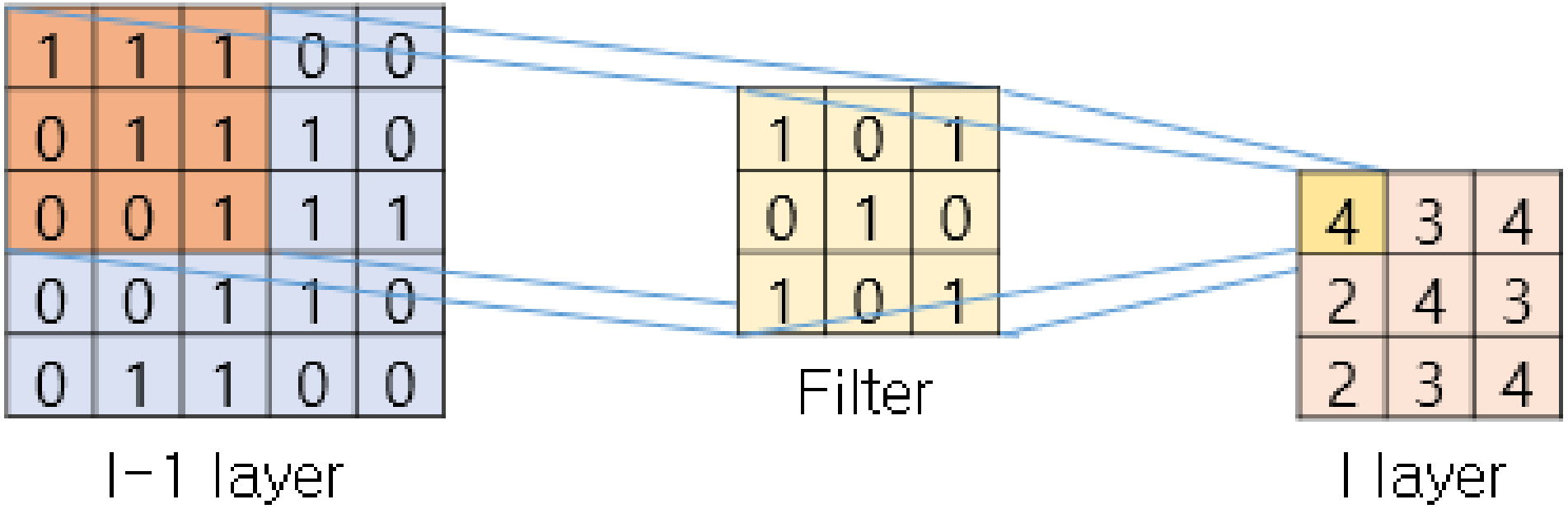
✓ 특정 버스가 도착한 경우(마포 08)

✓ 다른 버스가 도착한 경우

3. 배경 지식

3.1. Convolutional Neural Network(CNN)이란?

✓ 이미지 인식에 뛰어난 신경망
✓ 컨볼루션 연산과 풀링 연산을 반복하는 형태

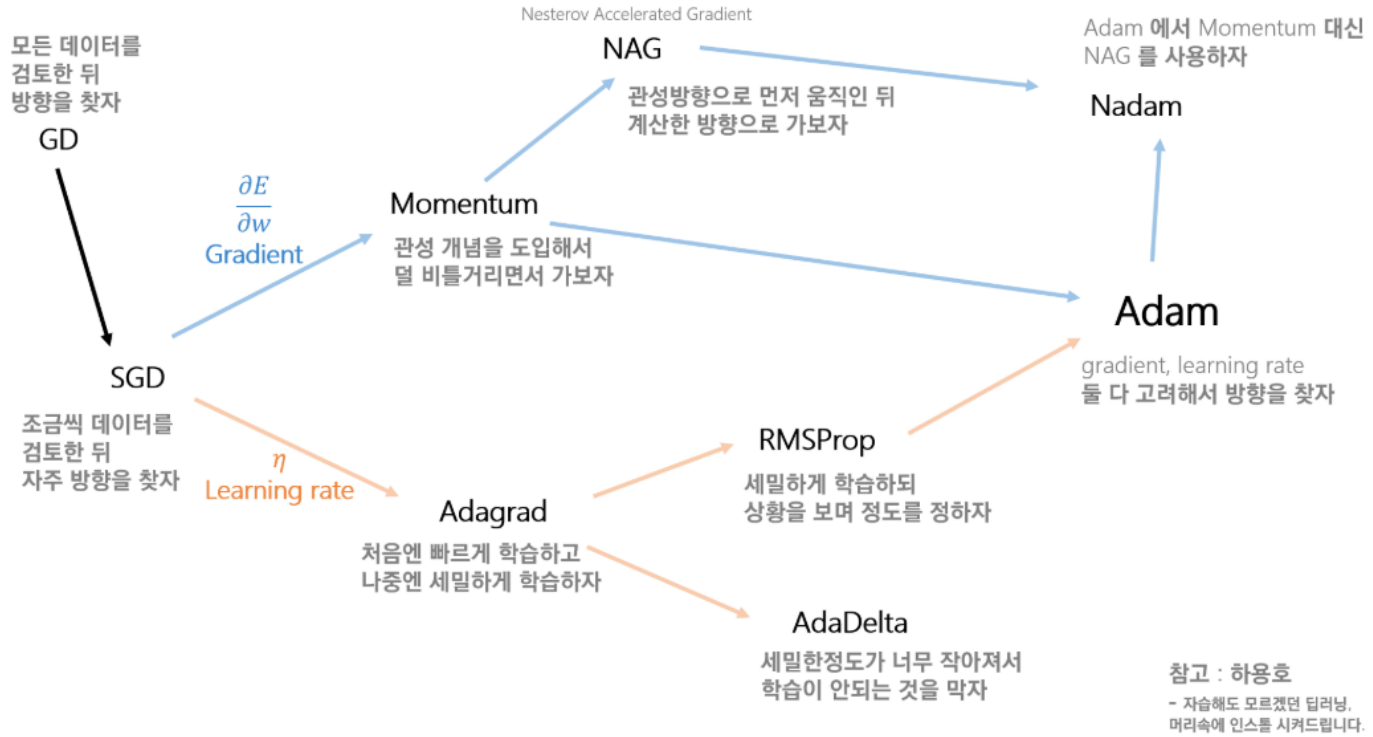


*Evaluation of Food Image Recognition Using CNN in Smart Refrigerator-논문

3. 배경 지식

3.2 Optimizer & Learning Rate & Dropout

- ✓ 손실함수(loss function)는 훈련을 진행하는 동안 최소로 만들어야 하는 값
- ✓ Optimizer는 손실함수를 기반으로 네트워크가 어떻게 업데이트 될지 결정

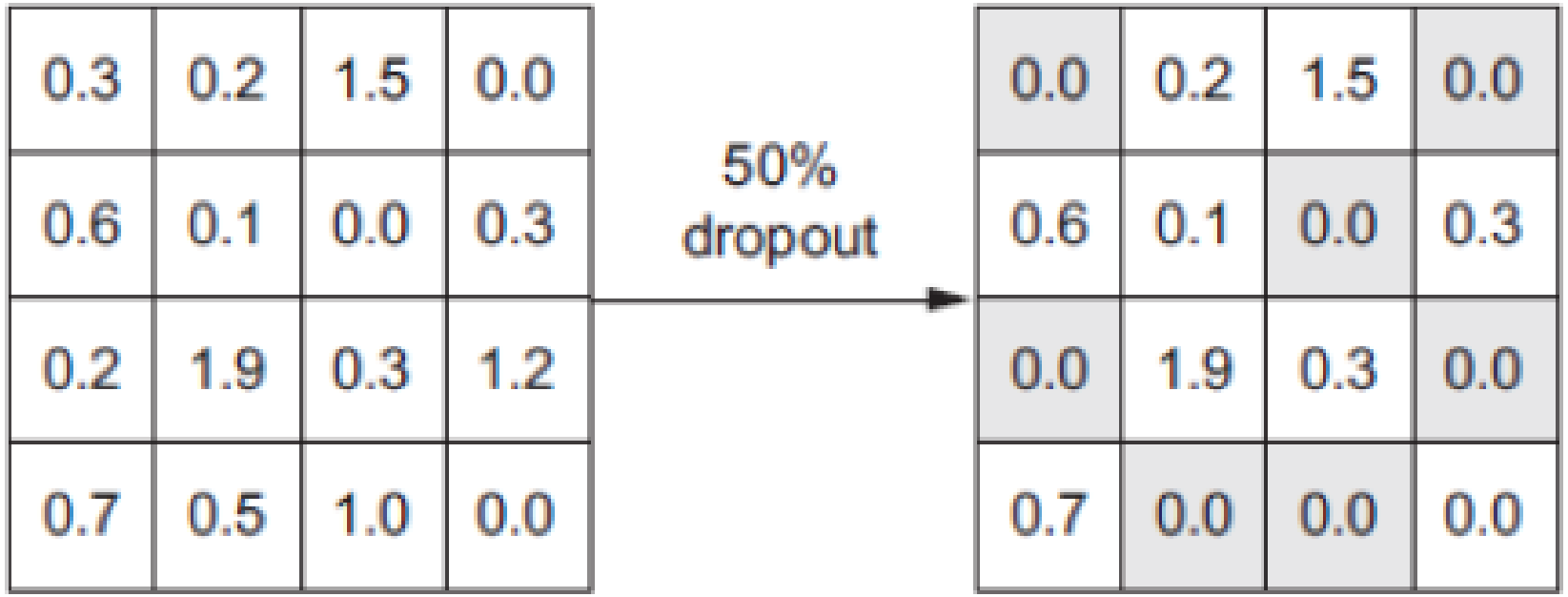


https://gomguard.tistory.com/187

3. 배경 지식

3.3 소규모 데이터 셋을 위한 다양한 기법들

- ✓ Learning Rate은 옵티머이저로 어떻게 움질일지를 결정했다면, 얼만큼 움직일지를 결정
- ✓ Dropout은 오버슈팅을 방지하기 위한 방법 중에 하나, 훈련 중 무작위로 일부 출력 특성 제외

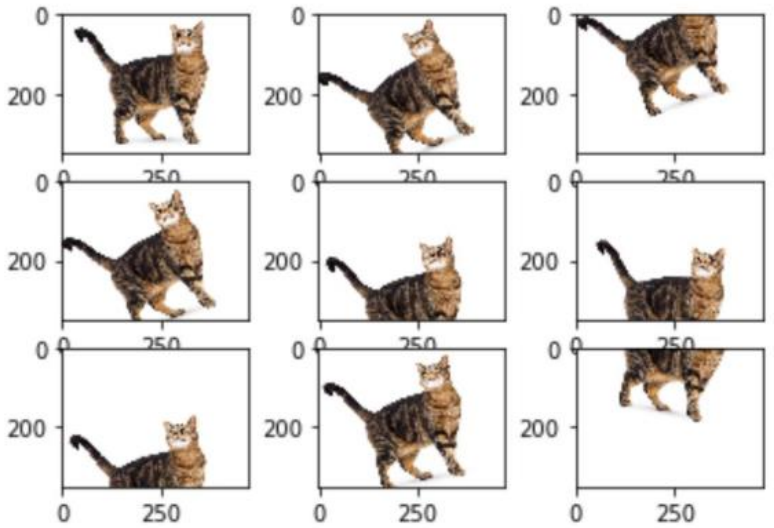


https://www.google.com/search?q=dropout+example&tbm=isch&ved=2ahUKEwjss6DUoO3pAhWjG6YKHXwuDJcQ2-cCegQIABAA&oq=dropout+example&gs_lcp=CgNpbWcQAzIECAAQGDIECAAQGDoCCAA6BAgAEB46BAgAEBM6CAgAEAUQHhATUKRXWlhpYONpaABwAHgAgAGoAYgBrQeSAQMwLjeYAQCgAQGqAQtnd3Mtd2l6LWltZw&sclient=img&ei=wJTbXuzTJaO3mAX83LC4CQ&bih=670&biw=809#imgrc=0A0eJw1961ApBM

3. 배경 지식

3.4 소규모 데이터 셋을 위한 다양한 기법들

- ✓ 데이터 증식은 이미지를 수평,수직 평행이동 확대 등으로 이미지를 변환시키는 것
- ✓ 목적에 따라서 적절히 선택하여 증식하는 것이 중요

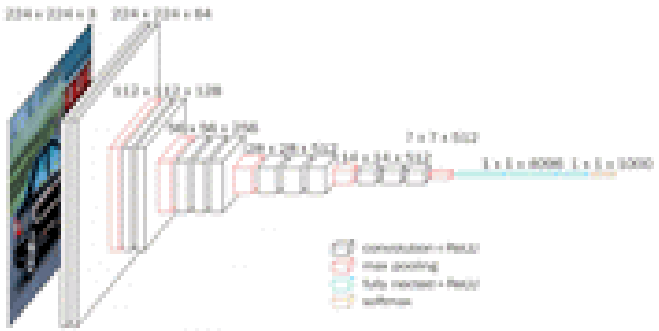


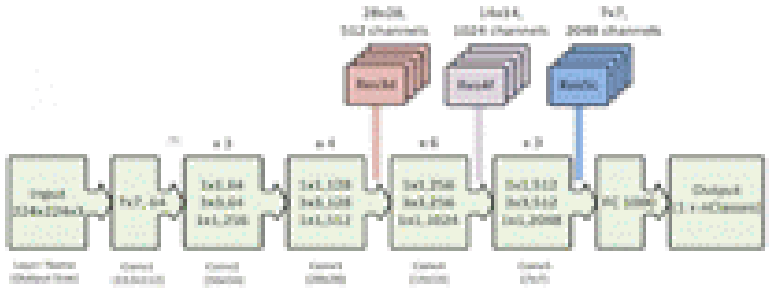
https://stackoverflow.com/questions/44819922/data-augmentation-is-shifting-needed

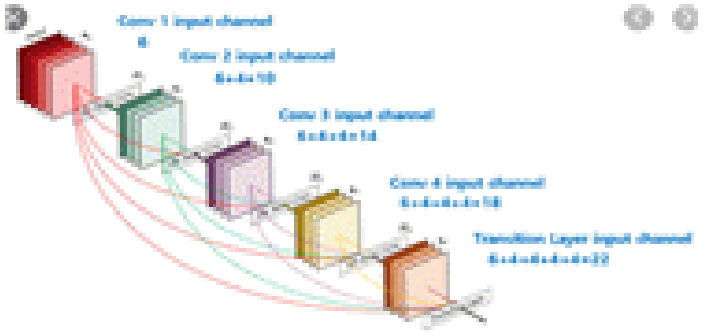
3. 배경 지식

3.5 특징 추출하기

- ✓ 사전에 학습된 네트워크의 표현을 사용하여 샘플에서 흥미로운 특징을 뽑아내는 기법
- ✓ ImageNet 데이터 셋에 훈련된 (VGG16, resnet50, Xeception) 네트워크를 새로운 분류에 적용







http://samediff.kr/wiki/index.php/DenseNet

4. 실험 1 busornot

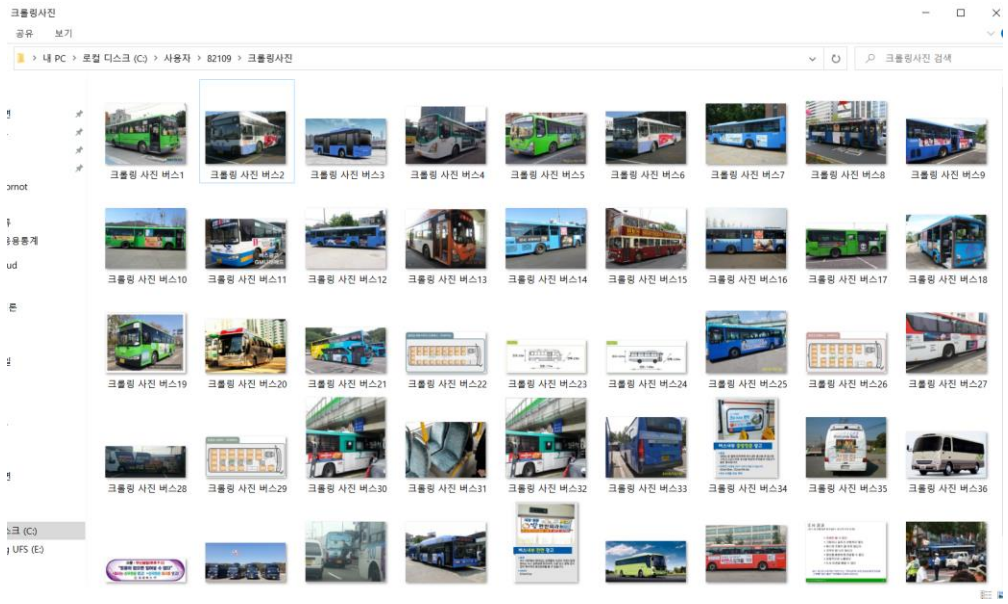
4.1 이미지 수집

✓ 쥬피터에서 코드를 통해 네이버 이미지 다운로드

In [1]:
```
from urllib.request import urlopen
from urllib.parse import quote_plus
from bs4 import BeautifulSoup
```

In [3]:
```
baseurl = "https://search.naver.com/search.naver?where=image&sm=tab_jum&query="
plusurl = input("크롤링 할 검색어를 입력하세요:")

url = baseurl + quote_plus(plusurl)
html = urlopen(url).read()
soup = BeautifulSoup(html,'html.parser')
img = soup.find_all(class_='_img')
n = 1
for i in img:
    imgurl = i['data-source']
    with urlopen(imgurl) as f:
        with open('./크롤링 사진 '+plusurl+str(n)+'.jpg','wb') as h:
            img = f. read()
            h.write(img)
    n += 1
```

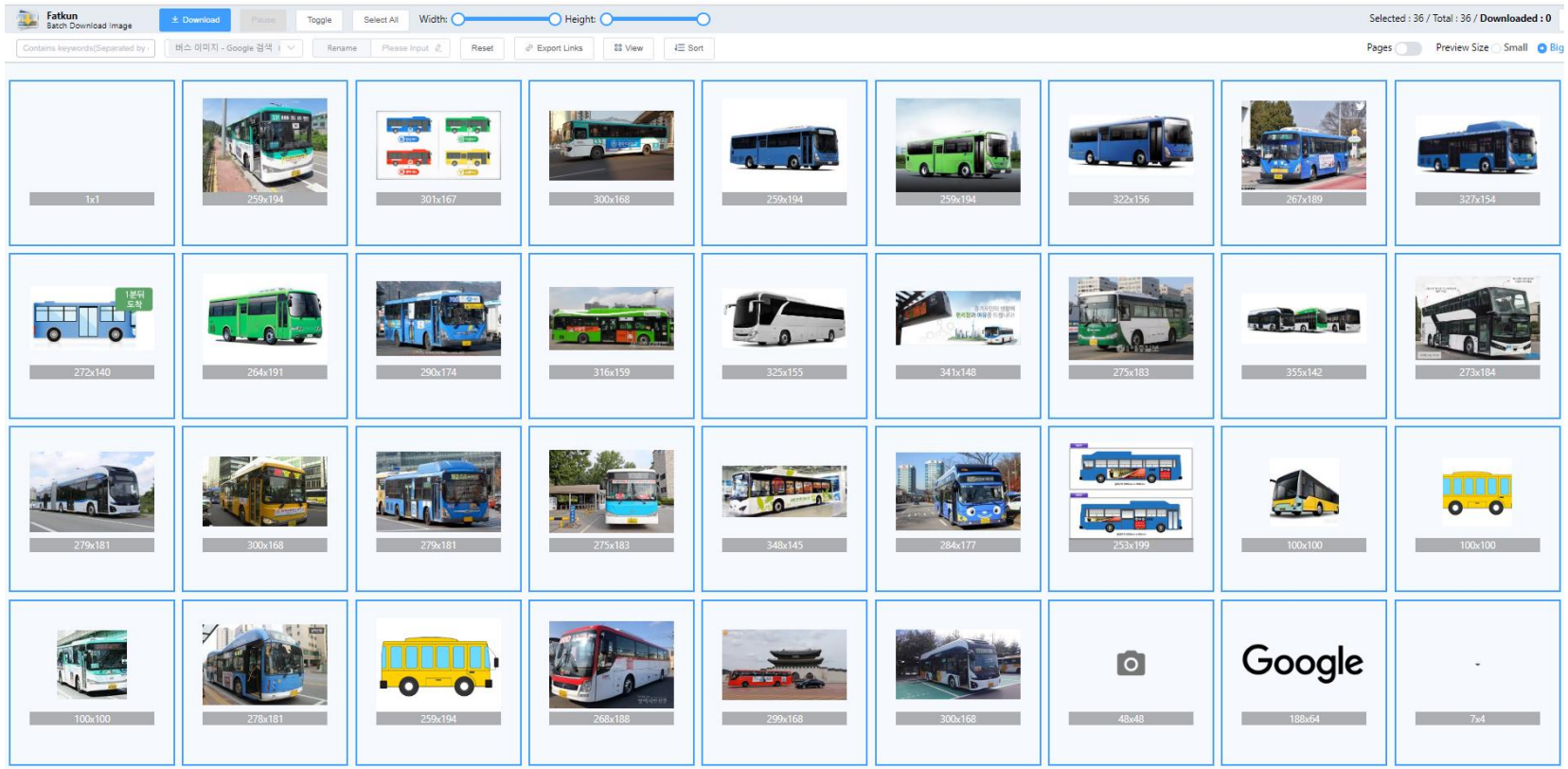
크롤링 할 검색어를 입력하세요:버스



4. 실험 1 busornot

4.1 이미지 수집

✓ 구글 이미지 다운로드 모듈(Fatkun) 활용



4. 실험 1 busornot

4.1 이미지 수집

✓ 실험 1 busornot 이미지에 추출에 사용된 키워드들

BUS	버스	bus	buses	public bus	서울버스
NOT	bulidings	background	배경화면	도시	city

✓ 실험 1 busornot 에 사용된 이미지 데이터의 수

Train		Validation		Test	
BUS	3278	BUS	819	BUS	819
NOT	2091	NOT	522	NOT	522

4. 실험 1 busornot

4.2 이미지 전처리

- ✓ 총 이미지가 5000장 정도로, ImageDataGenerator를 이용하여 이미지를 증식
- ✓ 회전, 가로 세로 이동, 확대 등을 parameter를 사용

In [1]:

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,rotation_range =40,width_shift_range= 0.2
                                   ,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2
                                   ,horizontal_flip=True,fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = train_datagen.flow_from_directory("desktop/busornot/train",target_size =(150,150),
                                                    batch_size =20, class_mode = 'binary')
validation_generator = test_datagen.flow_from_directory("desktop/busornot/validation",target_size =(150,150),
                                                    batch_size =20, class_mode = 'binary')
```

```
Using TensorFlow backend.
```

```
Found 5369 images belonging to 2 classes.
Found 1341 images belonging to 2 classes.
```

4. 실험 1 busornot

4.3 신경망 생성

- ✓ 크기 (150,150), 가장 기본적인 신경망 이용
- ✓ 이미지 증식에도 상당히 안 좋은 결과 (0.6075)

In [5]:
```
from keras import models, layers

# CNN layers
CNN = models.Sequential()
CNN.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape=(150,150,3)))
CNN.add(layers.MaxPooling2D((2,2)))
CNN.add(layers.Conv2D(64, (3,3), activation = 'relu'))
CNN.add(layers.MaxPooling2D((2,2)))
CNN.add(layers.Conv2D(128, (3,3), activation = 'relu'))
CNN.add(layers.MaxPooling2D((2,2)))
CNN.add(layers.Conv2D(64, (3,3), activation = 'relu'))
CNN.add(layers.MaxPooling2D((2,2)))

CNN.add(layers.Flatten())
CNN.add(layers.Dropout(0.5))
CNN.add(layers.Dense(512, activation = 'relu'))
CNN.add(layers.Dense(1, activation = 'softmax'))
```

In [6]:
```
from keras import optimizers
CNN.compile(loss='binary_crossentropy', optimizer='RMSprop', metrics=['acc'])
```

In [7]:
```
hist = CNN.fit_generator(
        train_generator,
        steps_per_epoch=15,
        epochs=50,
        validation_data= validation_generator,
        validation_steps=5)
```

In [13]:
```
test_loss ,test_acc =CNN.evaluate_generator(test_generator,steps =20)
print('test_loss:   ',test_loss)
print('test_acc:   ',test_acc)
```

```
test_loss:    7.666619777679443
test_acc:    0.6075000166893005
```

➤ test_acc = 0.6075

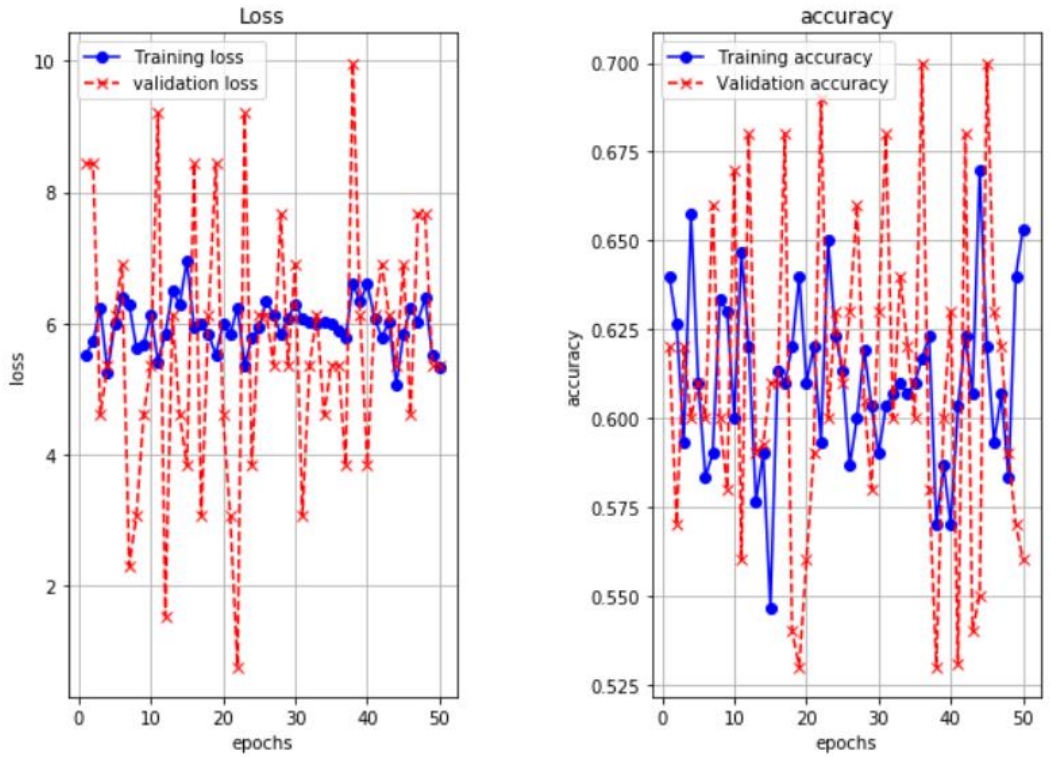
```
Epoch 1/50
15/15 [==============================] - 8s 508ms/step - loss: 5.5200 - acc: 0.6400 - val_loss: 8.4333 - val_acc: 0.6200
Epoch 2/50
15/15 [==============================] - 9s 588ms/step - loss: 5.7244 - acc: 0.6267 - val_loss: 8.4333 - val_acc: 0.5700
```

4. 실험 1 busornot

4.3 신경망 생성

- ✓ Loss가 전혀 줄어들지 않음
- ✓ 정교한 방식의 필요성을 느낌

```
Out[8]: <matplotlib.legend.Legend at 0x1ae0047a1c8>
```



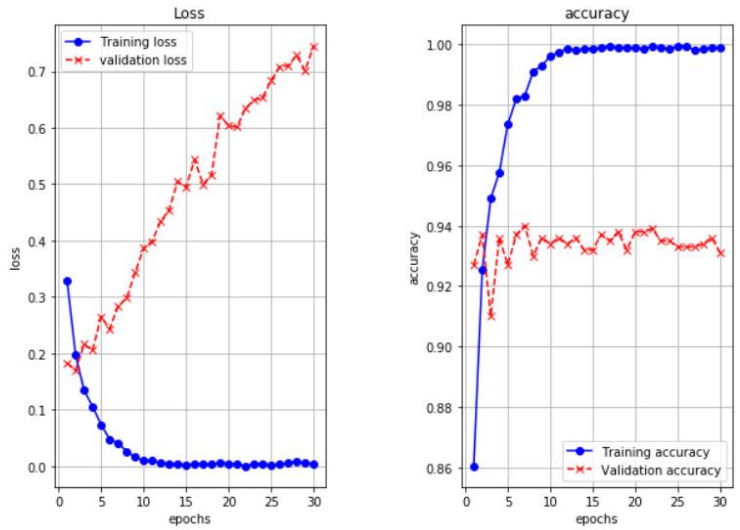
4. 실험 1 busornot

4.4 특징 추출하기

- ✓ Loss와 accuracy가 이상적인 모양
- ✓ Test accuracy 0.9169 상당히 개선된 모습 확인

In [4]:

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
base_dir = 'desktop/busornot'
train_dir = os.path.join(base_dir,'train')
validation_dir = os.path.join(base_dir,'validation')
test_dir = os.path.join(base_dir,'test')

datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count,4,4,512),dtype='uint8')
    labels= np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150,150),
        batch_size=batch_size,
        class_mode='binary')

    i=0
    for inputs_batch,labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i*batch_size : (i+1) * batch_size] = features_batch
        labels[i*batch_size:(i+1)*batch_size]=labels_batch
        i+=1
        if i*batch_size >= sample_count:
            break
    return features,labels

train_features, train_labels = extract_features(train_dir,2000)
validation_features,validation_labels = extract_features(validation_dir,1000)
test_features,test_labels = extract_features(validation_dir,1000)
```



In [10]:

```
test_generator = test_datagen.flow_from_directory(
        test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
print('test acc',test_acc)
```

```
Found 1341 images belonging to 2 classes.
1000/1000 [==============================] - 0s 117us/step
test acc 0.9169999957084656
```

➤ test_acc = 0.9166

4. 실험 1 busornot

4.5 최고 성능 만들기

- ✓ Optimizer 의 변경, 저장 형식의 변경
- ✓ Test accuracy 0.9499 로 개선

변수	변경 전	변경 후
optimizer	RMSprop	adam
특징들 저장형식	uint8	uint16

```
In [15]: test_generator = test_datagen.flow_from_directory(
             test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

         test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
         print('test acc',test_acc)

         Found 1341 images belonging to 2 classes.
         1000/1000 [==============================] - 0s 160us/step
         test acc 0.9449999928474426
```

➢ test_acc = 0.9499

5. 실험 2 mapo08ornot

5.1 이미지 수집

- ✓ 네이버와 구글은 실험1 busornot과 같은 방식 이용
- ✓ 마포08은 직접 촬영한 이미지를 활용

5. 실험 2 mapo08ornot

5.2 특징 추출 기법

✓ 0.9499로 높은 test accuracy 확인

In [50]:
```
import numpy as np
train_features =np.reshape(train_features,(500,4*4*512))
validation_features = np.reshape(validation_features,(200,4*4*512))
test_features = np.reshape(test_features,(200,4*4*512))
```

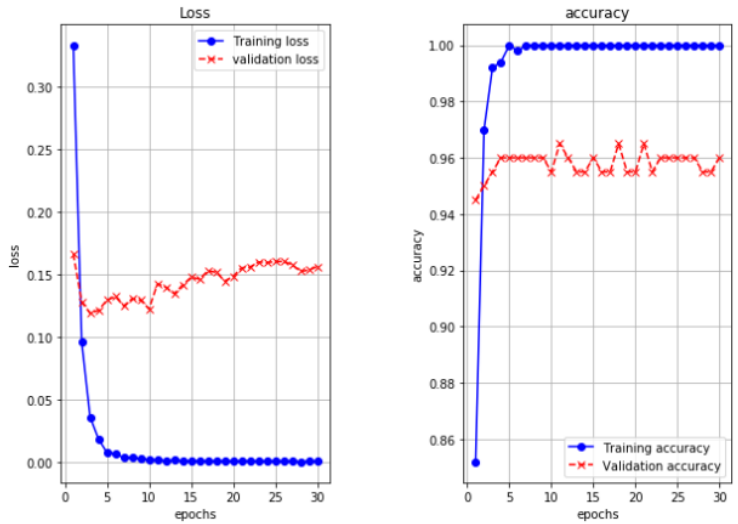
In [51]:
```
from keras import models
from keras import layers
from keras import optimizers

CNN = models.Sequential()
CNN.add(layers.Dense(256, activation = 'relu', input_dim=4*4*512))
CNN.add(layers.Dropout(0.5))
CNN.add(layers.Dense(1,activation = 'sigmoid'))
```

In [52]:
```
from keras import optimizers
CNN.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [53]:
```
hist = CNN.fit(train_features,train_labels,
               epochs=30,batch_size=20,
               validation_data = (validation_features,validation_labels))
```

```
Train on 500 samples, validate on 200 samples
Epoch 1/30
500/500 [==============================] - 1s 2ms/step - loss: 0.3332 - accuracy: 0.8520 - val_loss: 0.1663 - val_accuracy: 0.9450
Epoch 2/30
500/500 [==============================] - 1s 1ms/step - loss: 0.0958 - accuracy: 0.9700 - val_loss: 0.1278 - val_accuracy: 0.9500
Epoch 3/30
500/500 [==============================] - 1s 1ms/step - loss: 0.0356 - accuracy: 0.9920 - val_loss: 0.1187 - val_accuracy: 0.9550
Epoch 4/30
500/500 [==============================] - 1s 1ms/step - loss: 0.0181 - accuracy: 0.9940 - val_loss: 0.1212 - val_accuracy: 0.9600
Epoch 5/30
500/500 [==============================] - 1s 1ms/step - loss: 0.0073 - accuracy: 1.0000 - val_loss: 0.1292 - val_accuracy: 0.9600
Epoch 6/30
500/500 [==============================] - 1s 2ms/step - loss: 0.0065 - accuracy: 0.9980 - val_loss: 0.1321 - val_accuracy: 0.9600
Epoch 7/30
500/500 [==============================] - 1s 2ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.1246 - val_accuracy: 0.9600
Epoch 8/30
500/500 [==============================] - 1s 2ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.1304 - val_accuracy: 0.9600
Epoch 9/30
500/500 [==============================] - 1s 2ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.1293 - val_accuracy: 0.9600
Epoch 10/30
500/500 [==============================] - 1s 2ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.1222 - val_accuracy: 0.9550
Epoch 11/30
```

Out[55]: <matplotlib.legend.Legend at 0x2ac211b3c48>



In [56]:
```
test_generator = test_datagen.flow_from_directory(
        test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
print('test acc',test_acc)
```

```
Found 814 images belonging to 2 classes.
200/200 [==============================] - 0s 143us/step
test acc 0.949999988079071
```

➢ test_acc = 0.9499

5. 실험 2 mapo08ornot

5.3 resnet 50

- ✓ 굉장히 느림
- ✓ Validation 부터 성능이 좋지 않음

In [ ]:
```
from keras.applications import ResNet50
from keras.layers import Dense, Input, Activation
from keras.models import Model
from keras.callbacks import EarlyStopping
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
```

In [5]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,rotation_range =40,width_shift_range= 0.2
                                   ,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2
                                   ,horizontal_flip=True,fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = train_datagen.flow_from_directory("desktop/mapo08ornot/train",target_size =(150,150),
                                                    batch_size =20, class_mode = 'binary')
validation_generator = test_datagen.flow_from_directory("desktop/mapo08ornot/validation",target_size =(150,150),
                                                    batch_size =20, class_mode = 'binary')
test_generator = test_datagen.flow_from_directory("desktop/mapo08ornot/test",target_size =(150,150),
                                                    batch_size =20, class_mode = 'binary')
```

```
Found 3257 images belonging to 2 classes.
Found 754 images belonging to 2 classes.
Found 814 images belonging to 2 classes.
```

In [9]:
```
input = Input(shape=(150, 150, 3))
model = ResNet50(input_tensor=input, include_top=False, weights=None, pooling='max')

x = model.output
x = Dense(256, name='fully', init='uniform')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dense(126, init='uniform')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dense(1, activation='softmax', name='softmax')(x)
CNN = Model(model.input, x)
```

```
C:₩Users₩82109₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(256, name="fully", kernel_initializer="uniform")`
  """
C:₩Users₩82109₩anaconda3₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(126, kernel_initializer="uniform")`
```

5. 실험 2 mapo08ornot

5.4 Densenet 121

In [43]:
```
from keras.applications import DenseNet121
conv_bas = DenseNet121(include_top=False, weights='imagenet', input_tensor=None, input_shape=(150,150,3), pooling=None, classes=1000)
```

In [*]:
```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
base_dir = 'desktop/mapo08ornot'
train_dir = os.path.join(base_dir,'train')
validation_dir = os.path.join(base_dir,'validation')
test_dir = os.path.join(base_dir,'test')

datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count,4,4,1024),dtype='uint16')
    labels= np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150,150),
        batch_size=batch_size,
        class_mode='binary')

    i=0
    for inputs_batch,labels_batch in generator:
        features_batch = conv_bas.predict(inputs_batch)
        features[i*batch_size : (i+1) * batch_size] = features_batch
        labels[i*batch_size:(i+1)*batch_size]=labels_batch
        i+=1
        if i*batch_size >= sample_count:
            break
    return features,labels

train_features, train_labels = extract_features(train_dir,500)
validation_features,validation_labels = extract_features(validation_dir,200)
test_features,test_labels = extract_features(validation_dir,200)
```

```
Found 3257 images belonging to 2 classes.
```

In [45]:
```
import numpy as np
train_features =np.reshape(train_features,(500,4*4*1024))
validation_features = np.reshape(validation_features,(200,4*4*1024))
test_features = np.reshape(test_features,(200,4*4*1024))
```

In [46]:
```
from keras import models
from keras import layers
from keras import optimizers

CNN = models.Sequential()
CNN.add(layers.Dense(512, activation = 'relu', input_dim=4*4*1024))
CNN.add(layers.Dropout(0.5))
CNN.add(layers.Dense(1,activation = 'sigmoid'))
```

In [47]:
```
from keras import optimizers
CNN.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [48]:
```
hist = CNN.fit(train_features,train_labels,
               epochs=50,batch_size=20,
               validation_data = (validation_features,validation_labels))
```

```
Epoch 42/50
500/500 [==============================] - 4s 9ms/step - loss: 0.0156 - accuracy: 0.9940 - val_loss: 0.3324 - val_accuracy: 0.9400
Epoch 43/50
500/500 [==============================] - 5s 10ms/step - loss: 0.0118 - accuracy: 0.9980 - val_loss: 0.3145 - val_accuracy: 0.9450
Epoch 44/50
500/500 [==============================] - 5s 10ms/step - loss: 0.0027 - accuracy: 0.9980 - val_loss: 0.3303 - val_accuracy: 0.9300
Epoch 45/50
500/500 [==============================] - 5s 11ms/step - loss: 0.0161 - accuracy: 0.9980 - val_loss: 0.5190 - val_accuracy: 0.9500
Epoch 46/50
500/500 [==============================] - 5s 10ms/step - loss: 0.0251 - accuracy: 0.9940 - val_loss: 0.4171 - val_accuracy: 0.9350
Epoch 47/50
500/500 [==============================] - 3s 6ms/step - loss: 1.5170e-04 - accuracy: 1.0000 - val_loss: 0.3572 - val_accuracy: 0.95
00
```

In [50]:
```
test_generator = test_datagen.flow_from_directory(
        test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
print('test acc',test_acc)
```

```
Found 814 images belonging to 2 classes.
200/200 [==============================] - 0s 1ms/step
test acc 0.9700000286102295
```

➤ test_acc = 0.9700

6. 결론

6.1 실험 정확도 결과

✓ VGG16 특징 추출 기법 0.9449

```
In [15]: test_generator = test_datagen.flow_from_directory(
                test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

         test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
         print('test acc',test_acc)

         Found 1341 images belonging to 2 classes.
         1000/1000 [==============================] - 0s 160us/step
         test acc 0.9449999928474426
```

➢ test_acc = 0.9499

✓ DenseNet 121 특징 추출 기법 0.9700

```
In [50]: test_generator = test_datagen.flow_from_directory(
                test_dir,target_size=(150,150),batch_size =20,class_mode='binary')

         test_loss ,test_acc =CNN.evaluate(test_features,test_labels)
         print('test acc',test_acc)

         Found 814 images belonging to 2 classes.
         200/200 [==============================] - 0s 1ms/step
         test acc 0.9700000286102295
```

➢ test_acc = 0.9700

6. 결론

6.2 실험의 한계 & 향후 연구

- 실제 제공해야 하는 데이터의 차이 존재

  ✓ 시각 장애인에게 필요한 "버스와의 거리" 정보 결여
  ✓ "버스와의 거리 " 측정은 object-detection기법을 활용
  ✓ 향후 연구로는 CNN과 object-dectection방식 결합하여 추가 정보에 대한 연구

- 모형,크기는 같고 숫자만 다른 버스에 대한 구별 실험 필요성

  ✓ 실제 버스정류에는 모양,크기, 색이 동일하고 버스번호만 다른 경우가 다수 존재
  ✓ 이런 버스들을 분류해내는 더 정교한 신경망의 필요
  ✓ 숫자를 인식하는 OCR 방식과 정확도 차이 비교 및 OCR과 CNN 신경망 결합 가능성연구

7. 참고 문헌

- 케라스 창시자에게 배우는 딥러닝 (프랑소와 숄레 저)
- https://blog.naver.com/fkdldjs60/221867405574
- 모두의 딥러닝 1 (Youtube from sung kim)
- CS231n (youtube lecture from stanford university)
- 웹 크롤링과 스크레이핑 (카토 코타 저)
- https://github.com/codeMinst/ML-SimplerAPI/blob/master/blog/add_layer.py
- https://keras.io/api/applications/densenet/

8. 이미지 출처

[그림 1-1]	*서울시정개별연구원(교통약자 이동편의 증진 계획)
[그림 1-2]	http://www.seoulilbo.com/news/articleView.html?idxno=187194
[그림 1-3]	Evaluation of Food Image Recognition Using CNN in Smart Refrigerator-논문
[그림 1-4]	Evaluation of Food Image Recognition Using CNN in Smart Refrigerator-논문
[그림 1-5]	https://gomguard.tistory.com/187
[그림 1-6]	https://www.google.com/search?q=dropout+ example&tbm=isch&ved=2ahUKEwjss6DUoO3pAhWjG6YKHXwuDJcQ2-cCegQIABAA&oq=dropout+ example&gs_lcp=CgNpbWcQAzIECAAQGDIECAAQGDoCCAA6BAgAEB46BAgAEBM6CAgAEAUQHhATUKRXWIhpYONpaABwAHgAgAGoAYgBrQeSAQMwLjeYAQCgAQGqAQtnd3Mtd2l6LWltZw&sclient=img&ei=wJTbXuzTJaO3mAX83LC4CQ&bih=670&biw=809#imgrc=0A0eJw1961ApBM
[그림 1-7]	https://stackoverflow.com/questions/44819922/data-augmentation-is-shifting-needed
[그림 1-8]	http://samediff.kr/wiki/index.php/DenseNet