다중 사용자 정의 클래스 동시 학습 알고리즘 개발

2019.6

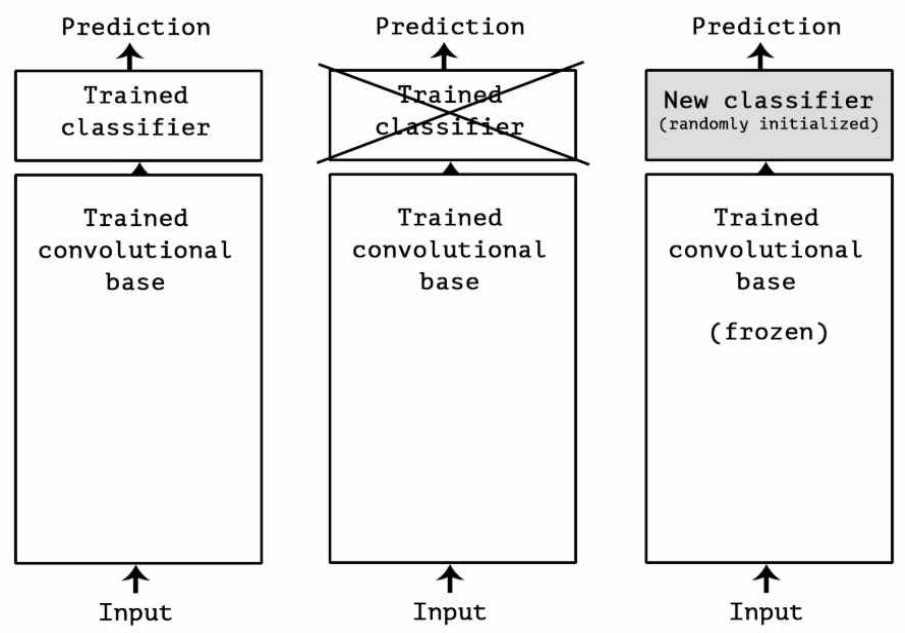
github: https://github.com/joyoon1110/my_classification

연구 노트

연구 목표

예측 모델을 구현하기 위한 사전 훈련된 컨브넷에 대한 자료 수집 및 실험 설계를 수행한다.

개념



사전 훈련된 네트워크()는 일반적으로 대규모 이미지 분류 분제를 위해 대량의 데이터셋에서 미리 훈련되어 저장된 네크워크이다. 작은 이미지 테이터 셋에 딥러닝을 적용하는 일반적이고 매우 효과적인 방법으로 알려져 있음. 데이터 증시 기법 사용여부에 따라 2가지로 수행할 수 있음.

[요약]

1. 특성추출1(CPU) - 데이터 증식 기법 사용 불가능
2. 특성추출2(GPU) - 데이터 증식 기법 사용 가능

연구 노트

연구 목표

사전 훈련된 컨브넷 네크워크 구성하여 classification(분류) 작업을 위한 실험을 수행한다.

개념

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 256)	1179904
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 9)	2313

Total params: 1,182,217
Trainable params: 1,182,217
Non-trainable params: 0

```
Epoch 30/30
32312/32312 [==============================] - 2s 72us/step - loss: 0.3346 - acc: 0.8791 - val_loss: 0.4027 - val_acc: 0.8721
```

캐런 시몬연과 앤드류 지서먼이 2014년에 개발한 VGG16 구조를 사용하여 분류 작업을 수행하였다. VGG16은 간단하고 ImageNet 데이터셋에 널리 사용되는 컨브넷 구조이다. VGG16은 조금 오래되었고 최고 수준의 성능에는 못 미치며 최근의 다른 모델 보다는 조금 무거운 단점이 있다.

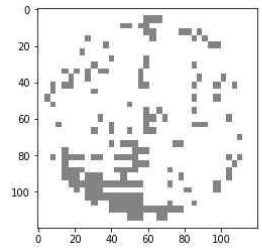
왼쪽 네트워크는 이미지의 특징을 추출하기위한 네트워크를, 오론쪽 네트워크는 추출한 특징을 훈련시키기 위한 네크워크를 나타내며 training accuracy와 validation accuracy에서 각각 87.9%와 87%의 정확도를 달성하였다.

연구 노트

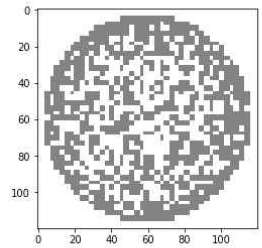
연구 목표

훈련된 모델을 사용하여 확률값에 따라 test sample을 분류하고, 사용자 정의 라벨링을 수행한다.

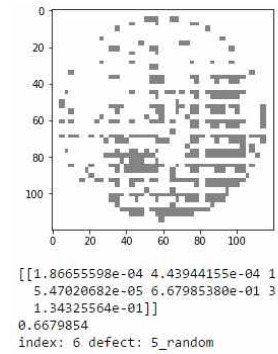
개념



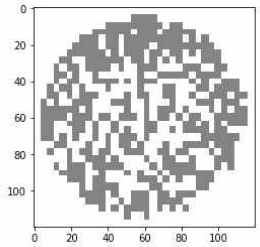
```
[[5.8373087e-04 3.7649053e-04 4.7165900e-01 3.9465260e-03 6.4623155
  8.5929892e-04 2.7768436e-01 1.6087389e-03 2.4327536e-01]]
0.471659
```



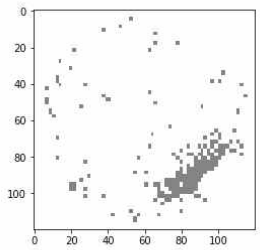
```
[[7.7467468e-20 2.5928501e-17 4.2207732e-10 8.5599767e-17 2.2467582
  1.0000000e+00 5.9603219e-25 6.0612802e-24 1.3400412e-14]]
1.0
index: 6 defect: 5_random
```



```
[[1.86655598e-04 4.43944155e-04 1.96855232e-01 5.60161141e-07
  5.47020682e-05 6.67985380e-01 3.21275547e-05 1.15841845e-04
  1.34325564e-01]]
0.6679854
index: 6 defect: 5_random
```



```
[[2.1227181e-14 8.9924589e-11 8.3709267e-05 7.4275322e-11 4.100012
  9.9991596e-01 1.8226710e-16 1.4212770e-13 2.0349578e-09]]
0.99991596
index: 6 defect: 5_random
```



```
[[3.1202301e-06 7.0025402e-07 3.2423118e-01 2.2918899e-08 2.915399
  5.2366141e-07 1.7665925e-02 1.8964958e-04 6.5790886e-01]]
0.65790886
index: 9 defect: 8_loc
```

각 그림은 웨이퍼의 결함 종류별 대표 이미지를 나타내며 숫자는 소속 클래스 확률값을 나타낸다. 가장 높은 숫자가 해당 클래스 의미하고, index 번호로 defect에 해당하는 라벨을 확인할 수 있도록 작성하였다.

연구 노트

연구 목표

사전 훈련 모델에 데이터 증식기법을 사용하여 훈련 정확도 및 예측정확도를 올리기 위한 작업을 수행한다.

개념

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(120, 120),
    batch_size=128,
    class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(120, 120),
    batch_size=128,
    class_mode='binary'
)
```

```
In [22]: history = model.fit_generator(
             train_generator,
             steps_per_epoch=30,
             epochs=30,
             validation_data=validation_generator,
             validation_steps=50,
             verbose=2)
```

```
Epoch 1/30
 - 22s - loss: 0.5878 - acc: 0.7982 - val_loss: 0.5478 - val_acc: 0.8086
Epoch 2/30
 - 17s - loss: 0.6080 - acc: 0.7867 - val_loss: 0.5775 - val_acc: 0.7984
Epoch 3/30
```

데이터 증식을 사용한 특성 추출 실험으로 훈련이 훨씬 느리고 비용이 많이 든다. 여기에서 비용은 연산 비용을 말하여 GPU를 사용할 수 있을 때 시도하는 점을 주의해야 한다. 학습에 적은양의 데이터를 사용하여 증식했으나 네트워크의 구성으로 인하여 훈련 정확도와 예측 정확도의 값이 향상되는 결과는 얻지 못했다.

연구 노트

연구 목표

사전 훈련 모델을 바꾸고 데이터 증식기법을 사용하여 훈련 정확도 및 예측정확도를 올리기 위한 작업을 수행한다.

개념

```
Train on 3116 samples, validate on 396 samples
Epoch 1/30
3116/3116 [==============================] - 1s 217us/step - loss: 0.4529 - acc: 0.8030 - val_loss: 0.2613 - val_acc: 1.0000
Epoch 2/30
3116/3116 [==============================] - 0s 65us/step - loss: 0.1895 - acc: 0.9676 - val_loss: 0.1120 - val_acc: 1.0000
Epoch 3/30
3116/3116 [==============================] - 0s 72us/step - loss: 0.1073 - acc: 0.9811 - val_loss: 0.0649 - val_acc: 1.0000
Epoch 4/30
3116/3116 [==============================] - 0s 81us/step - loss: 0.0706 - acc: 0.9878 - val_loss: 0.0309 - val_acc: 1.0000
Epoch 5/30
3116/3116 [==============================] - 0s 73us/step - loss: 0.0504 - acc: 0.9891 - val_loss: 0.0229 - val_acc: 1.0000
Epoch 6/30
3116/3116 [==============================] - 0s 71us/step - loss: 0.0423 - acc: 0.9901 - val_loss: 0.0145 - val_acc: 1.0000
Epoch 7/30
3116/3116 [==============================] - 0s 69us/step - loss: 0.0341 - acc: 0.9901 - val_loss: 0.0103 - val_acc: 1.0000
Epoch 8/30
3116/3116 [==============================] - 0s 62us/step - loss: 0.0293 - acc: 0.9923 - val_loss: 0.0108 - val_acc: 1.0000
Epoch 9/30
3116/3116 [==============================] - 0s 59us/step - loss: 0.0277 - acc: 0.9933 - val_loss: 0.0062 - val_acc: 1.0000
Epoch 10/30
3116/3116 [==============================] - 0s 62us/step - loss: 0.0215 - acc: 0.9942 - val_loss: 0.0049 - val_acc: 1.0000
Epoch 11/30
3116/3116 [==============================] - 0s 61us/step - loss: 0.0207 - acc: 0.9936 - val_loss: 0.0050 - val_acc: 1.0000
Epoch 12/30
3116/3116 [==============================] - 0s 60us/step - loss: 0.0202 - acc: 0.9945 - val_loss: 0.0037 - val_acc: 1.0000
Epoch 13/30
3116/3116 [==============================] - 0s 59us/step - loss: 0.0180 - acc: 0.9949 - val_loss: 0.0036 - val_acc: 1.0000
Epoch 14/30
3116/3116 [==============================] - 0s 61us/step - loss: 0.0148 - acc: 0.9961 - val_loss: 0.0031 - val_acc: 1.0000
Epoch 15/30
3116/3116 [==============================] - 0s 59us/step - loss: 0.0150 - acc: 0.9971 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 16/30
3116/3116 [==============================] - 0s 61us/step - loss: 0.0150 - acc: 0.9958 - val_loss: 0.0019 - val_acc: 1.0000
Epoch 17/30
3116/3116 [==============================] - 0s 64us/step - loss: 0.0129 - acc: 0.9949 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 18/30
3116/3116 [==============================] - 0s 59us/step - loss: 0.0126 - acc: 0.9978 - val_loss: 0.0014 - val_acc: 1.0000
Epoch 19/30
3116/3116 [==============================] - 0s 60us/step - loss: 0.0127 - acc: 0.9974 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 20/30
3116/3116 [==============================] - 0s 62us/step - loss: 0.0102 - acc: 0.9974 - val_loss: 0.0013 - val_acc: 1.0000
Epoch 21/30
3116/3116 [==============================] - 0s 71us/step - loss: 0.0093 - acc: 0.9981 - val_loss: 0.0012 - val_acc: 1.0000
Epoch 22/30
3116/3116 [==============================] - 0s 76us/step - loss: 0.0096 - acc: 0.9974 - val_loss: 0.0012 - val_acc: 1.0000
```


P100 기준	시간(s)
패키지 로드	1.93
이미지 로드	0.1(241장)
데이터 증식	11.4
특징 추출	7.78
트레이닝	7.23
	28.44

사전 훈련 모델을 바꾼 결과 훈련 정확도와 테스트 정확도의 값이 크게 향상된 결과를 얻을 수 있었다. 뿐만 아니라 각각의 과정에서 소요되는 시간을 확인하였다.

연구 노트

연구 목표

사용자 정의 분류(classification)를 위해 데이터 소스의 서식과 가공방법을 알아본다.

개념

1. SQLite, MySQL

테이블 생성, 데이터 추가, 데이터 추출

```python
# 라이브러리 읽어 들이기 --- (※1)
import MySQLdb
# MySQL 연결하기 --- (※2)
conn = MySQLdb.connect(
    user='root',
    passwd='test-password',
    host='localhost',
    db='test')
# 커서 추출하기 --- (※3)
cur = conn.cursor()
# 테이블 생성하기 --- (※4)
cur.execute('DROP TABLE items')
cur.execute('''
    CREATE TABLE items (
        item_id INTEGER PRIMARY KEY AUTO_INCREMENT,
        name TEXT,
        price INTEGER
    )
    ''')
# 데이터 추가하기 --- (※5)
data = [('Banana', 300),('Mango', 640), ('Kiwi', 280)]
for i in data:
    cur.execute("INSERT INTO items(name,price) VALUES(%s,%s)", i)
# 데이터 추출하기 --- (※6)
cur.execute("SELECT * FROM items")
for row in cur.fetchall():
    print(row)
```

03장: 데이터 소스의 서식과 가공

3-1. 웹의 다양한 데이터 형식

__텍스트 데이터와 바이너리 데이터 / XML 분석 / JSON 분석 / YAML 분석 / CSV/TSV 분석 / 엑셀 파일 분석

3-2. 데이터베이스

데이터베이스 / 데이터 저장에는 어떤 데이터베이스를 사용해야 할까? / SQLite - 가볍게 파일 하나로 사용할 수 있는 데이터베이스 / MySQL 사용하기 / TinyDB 사용하기

데이터에 머신러닝을 원활하게 적용하기 위한 데이터를 가공하는 방법을 DB 추출 및 조회 방법을 사용하여 구현하였음.

[참고문헌]

파이썬을 이용한 머신러닝, 딥러닝 실전 개발 입문

연구 노트

연구 목표

사용자 정의 분류(classification)를 위해 DB에서 사용자 정의 클러스터를 조회하는 방법을 다룬다.

개념

1. Database(MySQL)

- 테이블 생성, CRUD, Insert, Join

1) 데이터베이스의 목적

2) MySQL 설치

bitnami wamp 검색

MySQL, Apache, PHP 동시 설치

'Manage Servers' 탭에서 확인!

*'dir' 디렉토리 파일 확인 명령어

cmd에서 'C:₩Bitnami₩wampstack-7.1.29-0₩mysql₩bin' 으로 이동후

mysql -uroot -p 실행

3) MySQL의 구조

4) MySQL의 서버 접속

5) MySQL 스키마(schema)의 사용

'C:₩Bitnami₩wampstack-7.1.29-0₩mysql₩data'에 DATABASE 생성됨

ex) CREATE DATABASE opentutorials

* DROP DATABASE opentutorials

* SHOW DATABASES;

* USE opentutorials;

6) SQL과 테이블 구조

7) MySQL 테이블의 생성

7-1

*create table in mysql cheat sheet

```
CREATE TABLE topic(
  id INT(11) NOT NULL AUTO_INCREMENT,
```

7-2

```
CREATE TABLE topic(
  id INT(11) NOT NULL AUTO_INCREMENT,
  title VARCHAR(100) NOT NULL,
  description TEXT NULL,
  created DATETIME NOT NULL,
```

```
    author VARCHAR(30) NULL,
    profile VARCHAR(100) NULL,
    PRIMARY KEY(id));

8) MySQL의 CRUD
  Create
  Read
  Update
  Delete
9) SQL의 INSERT 구문
* SHOW TABLES;
* DESC topic;

  INSERT INTO topic (title,description,created,author,profile) VALUES('MySQL','MySQL is ...',NOW(),'egoing','developer');
  SELECT * FROM topic;
  INSERT INTO topic (title,description,created,author,profile) VALUES('ORACLE','ORACLE is',NOW(),'egoing','developer');
  SELECT * FROM topic;
  INSERT INTO topic (title,description,created,author,profile) VALUES('SQL Server','SQL Server is ...',NOW(),'duru','data administrator')
  INSERT INTO topic (title,description,created,author,profile) VALUES('PostgreSQL','PostgreSQL is ...',NOW(),'taeho','data scientist, developer')
  INSERT INTO topic (title,description,created,author,profile) VALUES('MongoDB','MongoDB is ...,NOW(),'egoing','developer')
  SELECT * FROM topic;
10) SQL의 SELECT 구문
  SELECT * FROM topic;
  SELECT id,title,created,author FROM topic;
  SELECT "egoing";
  SELECT "egoing", 1+1;
  SELECT id,title,created,author FROM topic WHERE author='egoing';
  SELECT id,title,created,author FROM topic WHERE author='egoing' ORDER BY id DESC;
  SELECT id,title,created,author FROM topic WHERE ahthor='egoing' ORDER BY id DESC LIMIT 2;

11) SQL의 UPDATE 구문
  DESC topic;
  SELECT * FROM topic;
```

```
  UPDATE topic SET description='Oracle is ...', title='Oracle' WHERE id=2;
  SELECT * FROM topic;
12) SQL의 DELETE 구문
  SELECT * FROM topic;
  DELETE FROM topic WHERE id = 5;
  SELECT * FROM topic;
13) 수업의 정상
14) 관계형 데이터베이스의 필요성
  SELECT * FROM author;
  SELECT * FROM topic;
  SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id;
15) 테이블 분리하기
  SHOW TABLES;
  RENAME TABLE topic TO topic_backup;
  SHOW TABLES;
  SELECT * FROM topic_backup;

  CREATE TABLE topic(
    id INT(11) NOT NULL AUTO_INCREMENT,
    title VARCHAR(30) NOT NULL,
    description TEXT,
    created DATETIME NOT NULL,
    author_id INT(11) DEFAULT NULLM,
    PRIMARY KEY(id))
  DESC topic;

  CREATE TABLE author(
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    profile VARCHAR(200) DEFAULT NULL,
    PRIMARY KEY(id))



  INSERT INTO author (id, name, profile) VALUES(1, 'egoing', 'developer');
  SELECT * FROM author;
  INSERT INTO author (id, name, profile) VALUES(2, 'duru', 'data administrator');
  SELECT * FROM author;
  INSERT INTO author (id, name, profile) VALUES(3, 'taeho', 'data scientist,
developer');
  SELECT * FROM author;
```

```
  INSERT INTO topic (id, title, description, created, author_id) VALUES(1, 'MySQL', 'MySQL is ...', '2018-1-1 12:10:11', 1);
  SELECT * FROM topic;
  INSERT INTO topic (id, title, description, created, author_id) VALUES(2, 'Oracle', 'Oracle is ...', '2018-01-03 13:01:10', 1);
  SELECT * FROM topic;
  INSERT INTO topic (id, title, description, created, author_id) VALUES(3, 'SQL Server', 'SQL Server is ...', '2018-01-20 11:01:10', 2);
  SELECT * FROM topic;
  INSERT INTO topic (id, title, description, created, author_id) VALUES(4, 'PostgreSQL Server', 'PostgreSQL Server is ...', '2018-01-23 1:3:3', 3);
  SELECT * FROM topic;
  INSERT INTO topic (id, title, description, created, author_id) VALUES(5, 'MongoDB', 'MongoDB is ...', '2018-01-30 12:31:3', 1);
  SELECT * FROM topic;

  SELECT * FROM author;
  SELECT * FROM topic;
16) 관계형 데이터베이스의 꽃 JOIN
  SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id;

  SELECT * FROM topic;
  SELECT * FROM topic LEFT JOIN author;
  SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id;
  SELECT id,title,description,created,name,profile FROM topic LEFT JOIN author ON topic.author_id = author.id;
  SELECT topic.id,title,description,created,name,profile FROM topic LEFT JOIN author ON topic.author_id = author.id;
  SELECT topic.id AS topic_id,title,description,created,name,profile FROM topic LEFT JOIN author ON topic.author_id = author.id;

17) 인터넷과 데이터베이스
  database server

   database    --->        database
   client      Internet    server
               <---
18) MySQL 클라이언트
19) MySQL Workbench
```

mysql workbench

mysql -uroot -p -hlocalhost

20) 수업을 마치며

index

modeling

backup - mysqldump, binary log

cloud - AWS RDS, Google Cloud SQL for MySQL, AZURE Database for MySQL

programming - Python mysql api, PHP mysql api, Java mysql api, ...

```
mysql> SELECT * FROM topic;
+----+------------+-------------------+---------------------+--------+--------------------------+
| id | title      | description       | created             | author | profile                  |
+----+------------+-------------------+---------------------+--------+--------------------------+
|  1 | MySQL      | MySQL is ...      | 2019-05-20 11:47:32 | egoing | developer                |
|  2 | ORACLE     | ORACLE is         | 2019-05-20 11:55:22 | egoing | developer                |
|  3 | SQL Server | SQL Server is ... | 2019-05-20 12:01:01 | duru   | data administrator       |
|  4 | PostgreSQL | PostgreSQL is ... | 2019-05-20 12:02:07 | taeho  | data scientist, developer |
|  5 | MongoDB    | MongoDB is ...    | 2019-05-20 12:02:55 | egoing | developer                |
+----+------------+-------------------+---------------------+--------+--------------------------+
5 rows in set (0.00 sec)
```

2. Image database 저장

[참고문헌]

https://www.opentutorials.org/course/3161

연구 노트

연구 목표

사전 훈련된 모델을 이용하여 wm-811k 데이터 셋의 특징을 추출하고, fully-connected layer의 부분을 수정하여 분류 모델을 만드는 작업을 수행한다.

개념

1. 사전 훈련 모델

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

	Xception	VGG16	VGG19	ResNet50
1.패키지로드	4.38 s	4.75 s		4.2 s
2.이미지로드	0.303 s	0.126 s		0.310 s
3.데이터생성	24.9 s	18 s		28.9 s
4.특징추출	8m 56 s	4m 40 s		3m 55s
5.Training		14 s		

- 두 개의 표 중에서 위에 나타낸 표는 딥러닝을 이용하여 이미지 분류를 위한 분류기를 만들 때 많이 사용하는 모델을 나타낸다. 네트워크의 깊이와 학습 파라미터에 따라 모델의 크기와 정확도가 다른 것을 알 수 있다.
- 아래에 나타낸 표는 사전 훈련 모델 중 일부인 Xception과 VGG16, ResNet50을 이용한 사전 훈련 모델을 사용하여 mk-811k에 대한 스크립트 실행 시간을 측정하였다.

연구 노트

연구 목표

사전 훈련 모델로 사용한 Xception, VGG16, ResNet50 외에 다양한 사전 훈련 모델이 존재하며, 이에 따라 다른 사전 훈련 모델을 사용하여 스크립트 실행 시간을 측정하는 작업을 수행한다.

개념

1. 사전 훈련 모델2

	Xception	VGG16	VGG19	ResNet50	ResNet101	ResNet152	ResNet50 v2	ResNet101 v2	ResNet152 v2	ResNetXt50
1.패키지로드	4									
2.이미지로드	0.303 s	0.126 s		0.310 s						
3.데이터생성	24.9 s	18 s		28.9 s						
4. 특징추출 (Batch 64)	3m 56 s	4m 40 s	5m 40 s	3m 55 s	9m 14 s		4m 21 s			9m 43 s
5. Training (30 epochs)		14 s								

	ResNEtXt 101	InceptionV3	Inception ResNetV2	MobileNet	MobileNetV2	DenseNet 121	DenseNet 169	DenseNet 201	NASNet Mobile	NASNet Large
1.패키지로드										
2.이미지로드		0.323 s								
3.데이터생성		38.5 s								
4. 특징추출 (Batch 64)		2m 15 s	3m 42 s	~~2m 17 s~~ ~~2m 18 s~~ ~~2m 18 s~~ 2m 17.6 s	~~2m 55 s~~ ~~2m 56 s~~ ~~2m 56 s~~ 2m 54.7 s	7m 16 s	8m 32 s		input_shape 고정	input_shape 고정
5. Training (30 epochs)		28.5 s	22.2 s	~~25.1 s~~ ~~26.4 s~~ ~~25.0 s~~ 25.5 s	~~32.6 s~~ ~~31.0 s~~ ~~30.8 s~~ 31.5 s	27.5 s				

173 서버 (CPU)

I	패키지 로드	이미지 로드	데이터 생성	특징추출 (Batch 128)	Training (30 epochs, Batch 128)	SUM
4,000 장	2.31 -> 2.45 (0.14 G)	2.45 -> 2.47 (0.02 G)	2.58 ->2.60 (0.02 G)	2.66 -> 3.06 (0.51 G)	2.92 -> 3.05 (0.13 G)	0.82 G

- 앞서 사전 훈련 모델 Xception, VGG16, ResNet50 외에 다양한 사전 훈련 모델을 이용하여 학습 및 스크립트를 수행하는 동안의 시간을 측정하였다.
- 사전 훈련 모델중 일부는 입력 데이터 셋의 고정된 크기로 인하여 사용할 수 없는 모델이 존재하였으며, 네트워크의 크기에 따라 소요되는 시간이 다양하게 분포되는 것을 알 수 있다. 그중 네트워크의 크기 및 스크립트의 수행속도가 가장 적게 소요되는 모델을 잠정적 사전 훈련 네트워크로 사용할 것을 결정하였다.

연구 노트

연구 목표

DB에서 쿼리된 데이터를 가정하여, 앞에서 결정한 사전 훈련 모델을 이용하여 분류기를 만드는 작업을 수행한다.

개념

1. DB 연동 Script 실행

```
# 가정: "SELECT * FROM new_cluster WHERE nc_userid = 'kms' AND nc_flag = FALSE")
data = [('1', 'kms','uc_1','C:/Users/hbee.ysjo/mindflow/UserDefineClassification/kms/uc_1/', 'False', '2019-05-24 09:04:31.320887'),
        ('2', 'kms','uc_2','C:/Users/hbee.ysjo/mindflow/UserDefineClassification/kms/uc_2/', 'False', '2019-05-24 09:04:31.320887'),
        ('3', 'kms','uc_3','C:/Users/hbee.ysjo/mindflow/UserDefineClassification/kms/uc_3/', 'False', '2019-05-24 09:04:31.320887'),
        ('4', 'kms','uc_4','C:/Users/hbee.ysjo/mindflow/UserDefineClassification/kms/uc_4/', 'False', '2019-05-24 09:04:31.320887'),
        ('5', 'kms','uc_5','C:/Users/hbee.ysjo/mindflow/UserDefineClassification/kms/uc_5/', 'False', '2019-05-24 09:04:31.320887')]
```

```
Train on 4557 samples, validate on 1133 samples
Epoch 1/10
4557/4557 [==============================] - 2s 492us/step - loss: 0.0061 - acc: 0.9989 - val_loss: 0.4270 - val_acc: 0.9170
Epoch 2/10
4557/4557 [==============================] - 2s 482us/step - loss: 0.0055 - acc: 0.9993 - val_loss: 0.4448 - val_acc: 0.9126
Epoch 3/10
4557/4557 [==============================] - 2s 486us/step - loss: 0.0045 - acc: 0.9998 - val_loss: 0.4081 - val_acc: 0.9188
Epoch 4/10
4557/4557 [==============================] - 2s 481us/step - loss: 0.0048 - acc: 0.9991 - val_loss: 0.4370 - val_acc: 0.9162
Epoch 5/10
4557/4557 [==============================] - 2s 508us/step - loss: 0.0046 - acc: 0.9993 - val_loss: 0.4417 - val_acc: 0.9135
Epoch 6/10
4557/4557 [==============================] - 2s 508us/step - loss: 0.0044 - acc: 0.9991 - val_loss: 0.4398 - val_acc: 0.9170
Epoch 7/10
4557/4557 [==============================] - 2s 532us/step - loss: 0.0040 - acc: 0.9993 - val_loss: 0.4482 - val_acc: 0.9144
Epoch 8/10
4557/4557 [==============================] - 2s 483us/step - loss: 0.0037 - acc: 1.0000 - val_loss: 0.4492 - val_acc: 0.9188
Epoch 9/10
4557/4557 [==============================] - 2s 487us/step - loss: 0.0042 - acc: 0.9998 - val_loss: 0.4638 - val_acc: 0.9162
Epoch 10/10
4557/4557 [==============================] - 2s 499us/step - loss: 0.0036 - acc: 0.9996 - val_loss: 0.4505 - val_acc: 0.9135
Wall time: 5min 38s
```

- DB에서 조회된 data는 순서대로 'index', 'userid', 'cluster_name', 'path', 'training true/false', '추가 시간'을 의미한다.
- 사전 훈련된 모델로 MobileNet을 사용한 훈련 모델의 정확도는 훈련 정확도는 99.96%, 테스트 정확도는 91.35%를 달성하였다. 훈련에 사용한 사전 훈련 모델의 유효성을 확인할 수 있는 실험이었으며 또한, GPU사용이 제한된 환경에서도 충분히 성능을 발휘하는 것을 확인하였다.

연구 노트

연구 목표

클래스 추가에 따른 스크립트 수행 속도를 측정하고, 사전 훈련 모델을 사용한 저장될 추출된 특징의 크기를 확인하는 실험을 수행한다.

개념

1. 클래스 추가 실험

2 class: 3:00

3 clsas: 3:30

4 class: 3:22

5 class: 5:38

- 위에서 클래스의 숫자와 시간은 앞서 수행했던 사전 훈련 모델을 이용하여 데이터 셋의 종류 및 숫자에 따라 훈련된 시간을 의미한다. 클래스 숫자의 증가에 따라 4개의 클래스까지는 시간의 변화가 크게 있지는 않았지만 클래스의 숫자가 5개부터 특징 추출로 인한 시간으로 인하여 시간의 증가를 확인하였다.

2. 추출된 특징의 저장 공간

1. **이미지** vs **npy** vs **pyarrow**

(3class 6000장 기준)

이미지 - train: 23.6 MB test: 6MB **SUM**: 30 MB

---

npy - train: 159 MB test: 40MB **SUM**: 200MB

prarrow - train: 55 MB test: 15MB **SUM**: 70 MB

- 스크립트를 수행하는 서버에서 저장 공간의 크기를 확인하기 위한 실험을 수행하였다. 3개의 클래스를 기준으로 원본 데이터 셋의 크기는 30MB, 추출된 특징의 크기는 npy와 pyarrow를 이용하여 저장했을 경우 각각 200MB, 70MB로 확인되었다.

연구 노트

연구 목표

사용자 정의 데이터 셋 클래스 추가에 대한 상황을 반영하여 스크립트를 작성하여 훈련 및 테스트 정확도를 측정한다.

개념

1. 추출된 특징을 저장하기 위한 디렉토리 작업

```
1. 데이터 증식 -> 특징 추출 -> Training
                    save feature
ex) ./UserDefineClassification/kms/tmp/tmp_train/
    ./UserDefineClassification/kms/tmp/tmp_train/uc_1
    ./UserDefineClassification/kms/tmp/tmp_train/uc_2
                          :
    ./UserDefineClassification/kms/tmp/tmp_train/uc_n
ex2) one-hot encoding
     2 class: 0 or 1
     3 class: 100 010 001
```

- 위에 나타낸 디렉토리 경로는 사전 훈련 모델에 사용하는 추출된 특징이 저장될 공간을 의미하며 기존에 특징 추출에서 데이터 증식을 위한과정을 추가하기 위한 디렉토리 작업을 수행한다.

2. 서버에서 스크립트로 전달될 파라미터

```
In [2]:
using_dictionary = {'uc_userid': 'jyp2',
                    'data': {
                             'uc_index':[0,4,2,3],
                             'uc_name':['uc1','uc1','uc2','uc2'],
                             'uc_features_flag':[False,False, False,False]
                            }
                   }
using_json = json.dumps(using_dictionary)
```

- DB에서 쿼리된 데이터를 json파일 형태로 변환하여 스크립트를 수행하기 위해 파라미터를 위 그림과 같이 바꾸는 작업을 수행하였다. 크게 userid와 data 부분으로 나뉘어지며, data 안에는 index, class_name, 특징 추출 유무를 확인하기 위한 파라미터 받는다.

3. 기존 클래스에 추가된 클래스 반영하여 전달될 파라미터

- 앞의 사용자 정의 클래스에 클래스 추가 및 삭제 상황을 위한 스크립트 작성을 수행

Step2 - index추가!!

```
## Step2 Index ADD
using_dictionary ={'uc_userid': 'jyp2',
                   'data': {
                            'uc_index':[0,4,2,3,5,6,7],
                            'uc_name':['uc1','uc1','uc2','uc2','uc3','uc3','uc3'],
                            'uc_features_flag':[True, True, True, True, False, False, False]
                           }
                  }
using_json = json.dumps(using_dictionary)
```

```
In [23]: model_training(train_feature_array, train_label_array, test_feature_array,test_label_array ,
                        cluster_class_dictionary,flag_false, 10)
Epoch 2/10
457/457 [==============================] - 0s 380us/step - loss: 0.2909 - acc: 0.9081 - val_loss: 0.0824 - val_acc: 0.9667
Epoch 3/10
457/457 [==============================] - 0s 388us/step - loss: 0.2000 - acc: 0.9322 - val_loss: 0.1319 - val_acc: 0.9583
Epoch 4/10
457/457 [==============================] - 0s 421us/step - loss: 0.1125 - acc: 0.9737 - val_loss: 0.0753 - val_acc: 0.9667
Epoch 5/10
457/457 [==============================] - 0s 436us/step - loss: 0.0597 - acc: 0.9803 - val_loss: 0.0686 - val_acc: 0.9833
Epoch 6/10
457/457 [==============================] - 0s 386us/step - loss: 0.0573 - acc: 0.9803 - val_loss: 0.0595 - val_acc: 0.9833
Epoch 7/10
457/457 [==============================] - 0s 384us/step - loss: 0.0338 - acc: 0.9847 - val_loss: 0.0636 - val_acc: 0.9833
Epoch 8/10
457/457 [==============================] - 0s 410us/step - loss: 0.0443 - acc: 0.9847 - val_loss: 0.0655 - val_acc: 0.9750
Epoch 9/10
457/457 [==============================] - 0s 386us/step - loss: 0.0193 - acc: 0.9934 - val_loss: 0.0646 - val_acc: 0.9750
Epoch 10/10
457/457 [==============================] - 0s 511us/step - loss: 0.0411 - acc: 0.9847 - val_loss: 0.0566 - val_acc: 0.9833

Out[23]: {'success': True}
```

한다. 클래스가 추가된 경우 추출된 특징이 존재하지 않으므로 features_flag의 값은 False를 가지게 되며, 이때 기존 추출된 특징은 그대로 사용하고, 추출하지 않은 특징만 사전훈련 모델을 사용하여 특징을 추출하게 된다.

- 기존의 상황에서 클래스가 추가되어 학습을 다시 할 경우 훈련 정확도 및 테스트 정확도에서 실무에서 활용할 수 있는 만큼의 정확도를 얻을 수 있음을 확인하였다.

연구 노트

연구 목표

사용자 정의 클래스가 훈련된 다음, 하나의 클래스가 추가 되었을 경우의 수행속도를 측정하고 테스트 데이터에서 확률값을 구하여 나타낸다.

개념

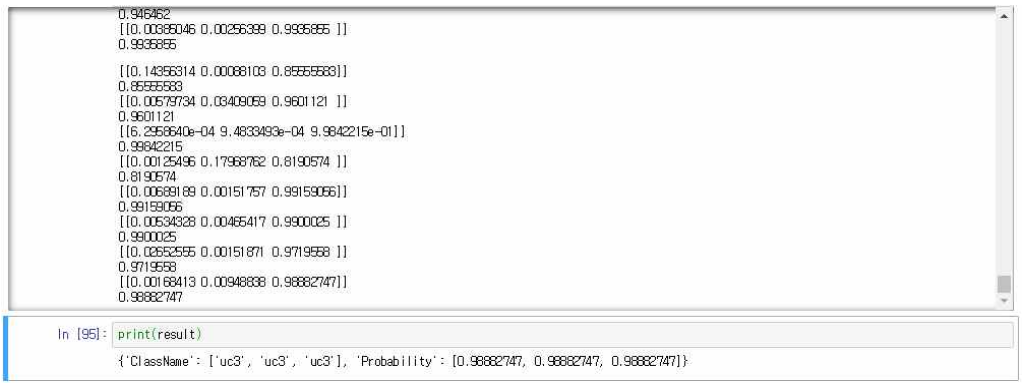
1. CPU서버에서의 수행속도 측정

2class, 4000 장 55 s

3class, 6000 장 36 s

- 기존의 2개의 클래스를 훈련할 경우 55s의 시간이 소요되었으며, 기존의 클래스의 추출된 특징은 그대로 사용하고 새로운 클래스를 훈련할 때, 36s의 시간이 소요되었음.

2. 추가된 클래스의 확률값



- 위에 실행된 스크립트는 테스트 데이터에 대한 클래스 추정과 확률값을 구하는 것으로 입력으로 사용한 데이터의 클래스를 추정하고 가장 확률값이 높은 값을 구하여 출력하게 된다.

연구 노트

연구 목표

스크립트를 실행하기 위해 standard input과 standard output을 정의하고 출력한다.

개념

1. standart input, standart output

- DB에서 조회한 데이터를 바탕으로 웹에 머신러닝 알고리즘 출력 결과를 보기 위해서 standard input과 standard output을 정의하는 것이 필요함.

```python
classifier = build_model()
classifier.load_weights(os.path.join(feature_path, 'model.h5'))
result = predict_images(classifier, image_list)

def default(o):
    if isinstance(o, np.int64): return int(o)
    if isinstance(o, np.float32): return float(o)
    raise TypeError
result_json = json.dumps(result, default=default)
sys.stdout.write(result_json)
```

```python
print(result)
```

```
{'ClassName': ['uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc3', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc
3', 'uc3', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'u
c1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1',
'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc3', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3',
'uc1', 'uc1', 'uc3', 'uc3', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc3', 'uc1', 'uc1',
'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1',
'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc3', 'uc1', 'uc2', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1',
'uc3', 'uc1', 'uc1', 'uc1', 'uc1', 'uc3', 'uc1', 'uc1', 'uc1', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc3', 'uc2', 'uc2', 'uc2', 'uc2',
'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2',
'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2',
'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2',
'uc2', 'uc2', 'uc3', 'uc2', 'uc2', 'uc2', 'uc3', 'uc3', 'uc2', 'uc3', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2', 'uc2',
'uc2', 'uc2', 'uc2', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1',
'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3',
'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc2', 'uc1',
'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc1', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc1', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3', 'uc2', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc3',
'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3',
'uc3', 'uc3', 'uc3', 'uc1', 'uc3', 'uc1', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3', 'uc3',
'uc3'], 'Probability': [0.900075, 0.903072, 0.93392545, 0.8898064, 0.7599701, 0.76734275, 0.5030881, 0.9909388, 0.92413133, 0.8255448, 0.49
7556, 0.985365, 0.9540008, 0.97894764, 0.7277377, 0.97839487, 0.998324, 0.570522, 0.76383454, 0.9158143, 0.95897436, 0.74661887, 0.9423744,
```

- 앞에서 수행했던 실험과 마찬가지로 실행된 스크립트는 입력으로 사용한 데이터의 클래스를 추정하고 가장 확률값이 높은 값을 구하여 출력하게 되는데 입력과 출력의 형태가 standard input과 standard output의 형태로 바꾸는 작업을 수행하였음.

연구 노트

연구 목표

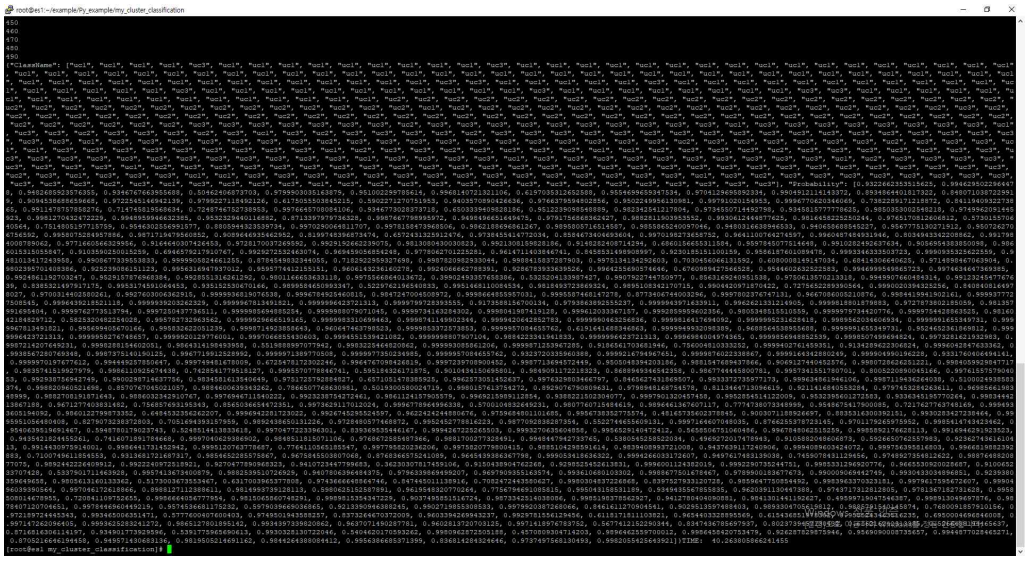
실제 스크립트가 실행될 서버에서 스크립트를 실행하여 standard input과 standard output의 출력값을 확인한다.

개념

1. 서버에서 스크립트 실행

* 498장 / Storm 서버(215)

* 시간: 40 s



- 실제 서버에서 작동하는 시간을 측정하였고, standard output의 형태로 출력됨을 확인하였음. 또한 이를 json 확장자 파일로 저장하여 웹 상에서 출력이 가능하도록 스크립트를 작성하였음.

연구 노트

연구 목표

스크립트 실행으로 인한 부하 테스트 작업을 수행한다.

개념

1. 부하 테스트

1. 부하테스트

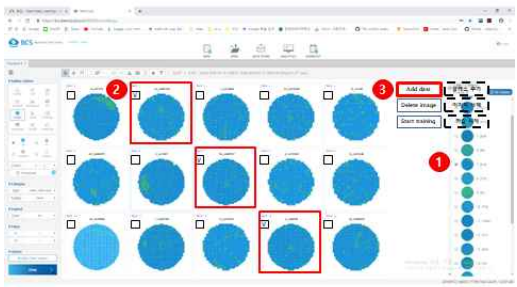
		2개 동시	3개 동시
2 Class	메모리	2.3 G	3.8 G
	시간	110 s	160 s
3 Class	메모리	2.4 G	3.7 G
	시간	140 s	200 s

- 서버에서 실행될 스크립트는 한명의 사용자가 아닌 여러 명의 사용자에 의해 실행된다. 그에 따라서 부하 테스트가 필요하며 실험을 수행하였음. 2개의 클래스와 3개의 클래스가 처음 수행될 때, 2개와 3개 동시 실행 테스트를 진행하였고 그 때 메모리와 시간은 위 표와 같음을 확인하였음. 클래스의 증가로 인한 메모리와 시간의 소비는 크게 차이가 없으나 동시 실행량으로 인한 메모리와 시간의 소비가 크게 늘어나는 것을 알 수 있었음.

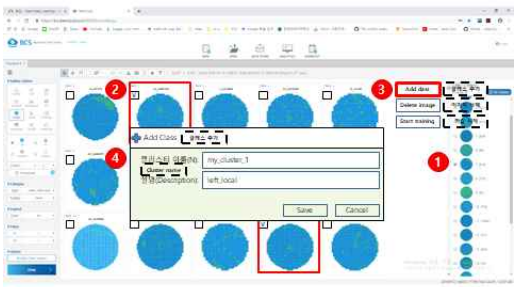
2. Flow chart

- 웹상에 적용하기 위한 UI의 구상도는 아래 그림과 같음. 사용자가 선택한 이미지로 새로운 클래스를 구성하며 DB에 사용자의 정보가 남게 되는데 이 부분을 조회하여 스크립트를 실행하게 된다.

1. 사용자관리 화면 디자인(추가)



1. 사용자관리 화면 디자인(추가)



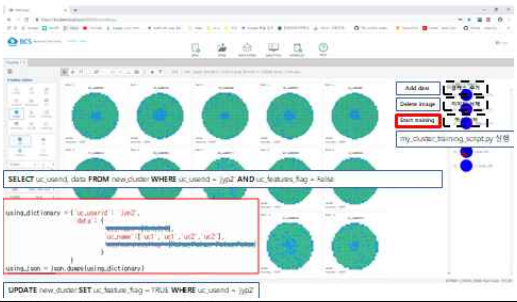
2. 사용자관리 항목

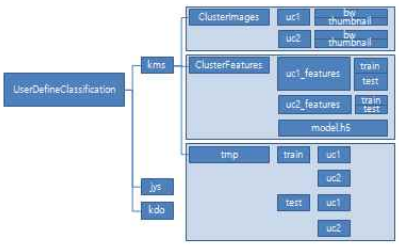
인덱	아이디	인덱스	클러스터이름	클러스터설명	Flag	
	jyp2	1	uc1	center	False	
	jyp2	2	uc2	donut	False	
	jyp2	3	uc2	Donut	False	
	jyp2	4	uc1	center	False	

2. 사용자관리 항목

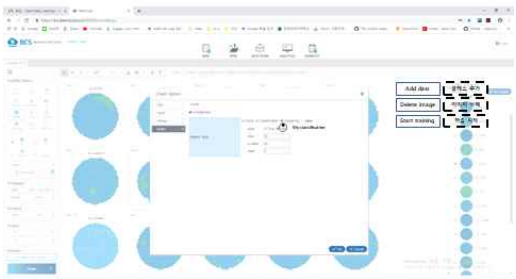
항목명	변수명	데이터 컴포넌트	필수	비고
아이디	uc_userid	text	Y	
인덱스	uc_index	int	Y	
클러스터이름	uc_index	text	Y	
클러스터설명	uc_description	text	-	
Flag	uc_features_flag	bool	Y	

3. 사용자 정의 클래스 training

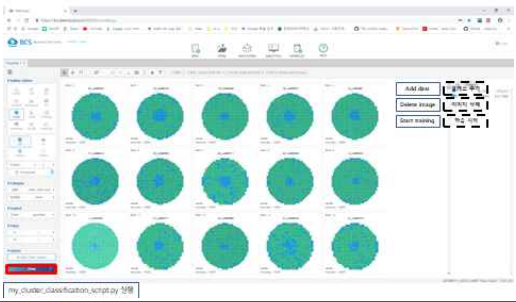




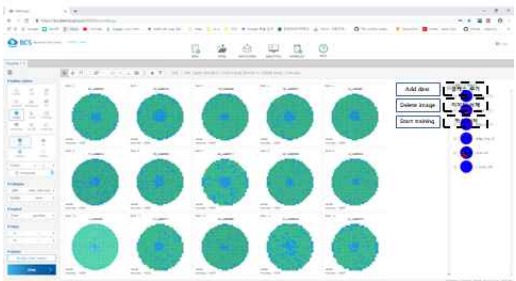
3. 사용자 정의 클래스 training



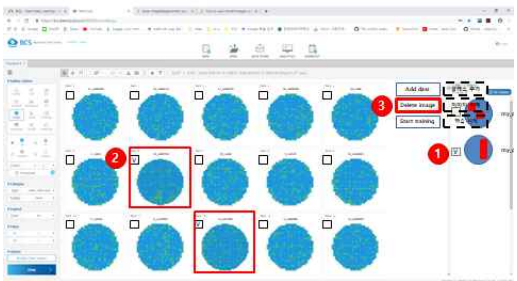
3. 사용자 정의 클래스 training



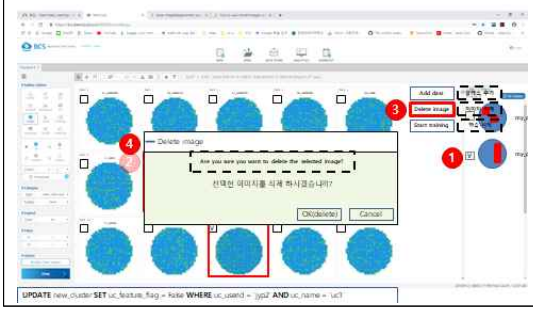
3. 사용자 정의 클래스 training



4. 사용자관리 화면 디자인(삭제)



4. 사용자관리 화면 디자인(삭제)



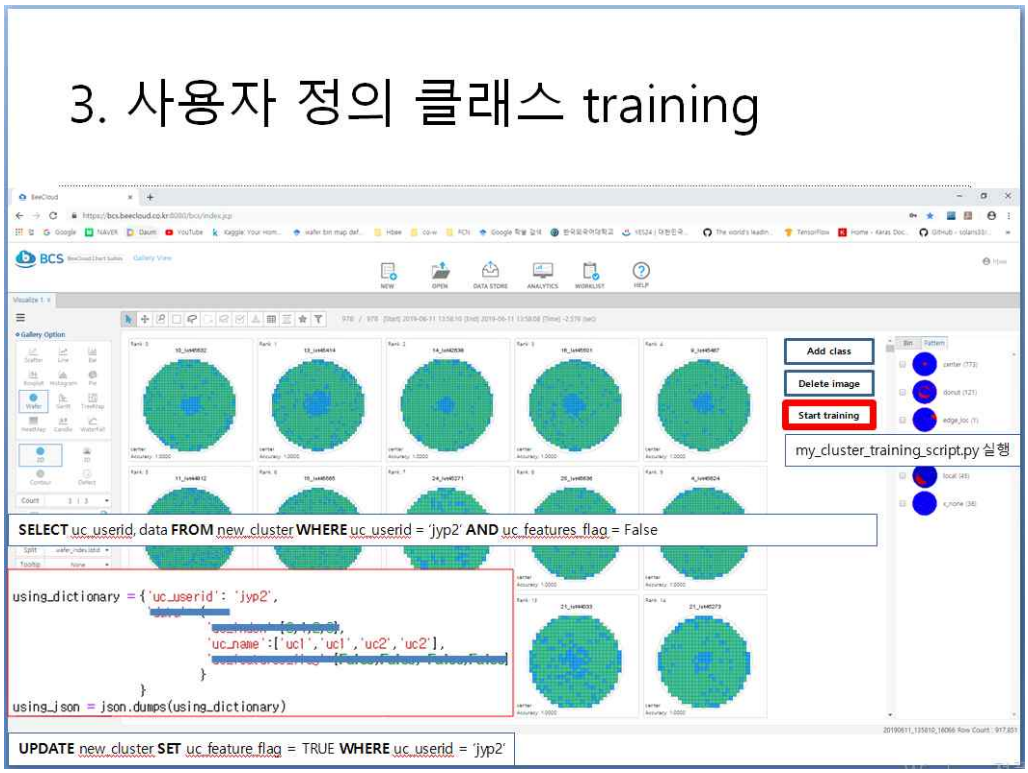
연구 노트

연구 목표

사용자 정의 클래스 분류를 위한 Script 동작을 기반으로 UI 디자인을 수정하고, Script를 동작시키기 위한 명령어를 작성한다. 또한, Sciprt의 동작을 확인하여 성능평가(수행속도, 정확도, 에러율)를 수행한다.

개념

1. 아래 그림은 사용자 정의 클래스 분류를 위한 UI 디자인으로 'Start training'을 클릭함으로써 학습을 위한 진행 과정을 시작한다. 'SELECT' 문과 'UPDATE' 문을 DB에서 쿼리하여 조건부 학습을 진행한다.



```
using_dictionary ={'uc_userid': 'hbee.ysjo',
                   'uc_name': ['uc1', 'uc2']}
```

```
Train on 3103 samples, validate on 651 samples
Epoch 1/10
3103/3103 [==============================] - 2s 508us/step - loss: 0.2625 - acc: 0.9120 - val_loss: 0.5813 - val_acc: 0.9078
Epoch 2/10
3103/3103 [==============================] - 1s 332us/step - loss: 0.0324 - acc: 0.9878 - val_loss: 0.7051 - val_acc: 0.9094
Epoch 3/10
3103/3103 [==============================] - 1s 329us/step - loss: 0.0263 - acc: 0.9900 - val_loss: 0.8895 - val_acc: 0.9078
Epoch 4/10
3103/3103 [==============================] - 1s 333us/step - loss: 0.0109 - acc: 0.9965 - val_loss: 0.8962 - val_acc: 0.9094
Epoch 5/10
3103/3103 [==============================] - 1s 329us/step - loss: 0.0056 - acc: 0.9987 - val_loss: 0.9335 - val_acc: 0.9094
Epoch 6/10
3103/3103 [==============================] - 1s 338us/step - loss: 0.0049 - acc: 0.9984 - val_loss: 1.0242 - val_acc: 0.9094
Epoch 7/10
3103/3103 [==============================] - 1s 339us/step - loss: 0.0040 - acc: 0.9981 - val_loss: 1.1072 - val_acc: 0.9094
Epoch 8/10
3103/3103 [==============================] - 1s 328us/step - loss: 0.0022 - acc: 0.9994 - val_loss: 1.2320 - val_acc: 0.9094
Epoch 9/10
3103/3103 [==============================] - 1s 329us/step - loss: 0.0017 - acc: 0.9994 - val_loss: 1.2026 - val_acc: 0.9094
Epoch 10/10
3103/3103 [==============================] - 1s 339us/step - loss: 0.0016 - acc: 0.9997 - val_loss: 1.2556 - val_acc: 0.9094
```

```
using_dictionary2 ={'uc_userid': 'hbee.ysjo',
                    'uc_name': ['uc3']}
```

```
Train on 4537 samples, validate on 913 samples
Epoch 1/10
4537/4537 [==============================] - 2s 504us/step - loss: 0.6936 - acc: 0.7758 - val_loss: 0.7346 - val_acc: 0.8653
Epoch 2/10
4537/4537 [==============================] - 2s 332us/step - loss: 0.3225 - acc: 0.8907 - val_loss: 0.7694 - val_acc: 0.8751
Epoch 3/10
4537/4537 [==============================] - 2s 339us/step - loss: 0.2337 - acc: 0.9121 - val_loss: 0.7612 - val_acc: 0.8708
Epoch 4/10
4537/4537 [==============================] - 2s 334us/step - loss: 0.1940 - acc: 0.9253 - val_loss: 0.7990 - val_acc: 0.8642
Epoch 5/10
4537/4537 [==============================] - 2s 332us/step - loss: 0.1580 - acc: 0.9403 - val_loss: 0.7108 - val_acc: 0.8751
Epoch 6/10
4537/4537 [==============================] - 2s 339us/step - loss: 0.1295 - acc: 0.9480 - val_loss: 0.7389 - val_acc: 0.8784
Epoch 7/10
4537/4537 [==============================] - 2s 331us/step - loss: 0.1227 - acc: 0.9526 - val_loss: 0.8331 - val_acc: 0.8740
Epoch 8/10
4537/4537 [==============================] - 1s 327us/step - loss: 0.1056 - acc: 0.9572 - val_loss: 0.8310 - val_acc: 0.8751
Epoch 9/10
4537/4537 [==============================] - 2s 334us/step - loss: 0.0968 - acc: 0.9592 - val_loss: 0.8733 - val_acc: 0.8773
Epoch 10/10
4537/4537 [==============================] - 1s 327us/step - loss: 0.0846 - acc: 0.9689 - val_loss: 1.0099 - val_acc: 0.8697
```

위의 그림은 Script를 동작시키기 위해 쿼리하여 딕셔너리 행태로 Script에 인자를 전달하여 사용자 정의 클래스 분류 학습 모델의 훈련을 시작한다. 'uc_name'으로부터 사용자가 정의한 클래스가 전달됨을 알 수 있으며 기존과 다른 클래스의 이름이 들어올 경우, 기존 클래스의 특징을 그대로 두고 새로운 클래스에 대해서만 특징을 추출한 뒤, 기존클래스와 결합하여 학습을 수행함을 알 수 있다.

각 Script의 훈련 정확도와 테스트 정확도로부터 클래스의 분류 정확도는 클래스의 종류에 영향을 받는 다는 것을 확인하였다.

연구 노트

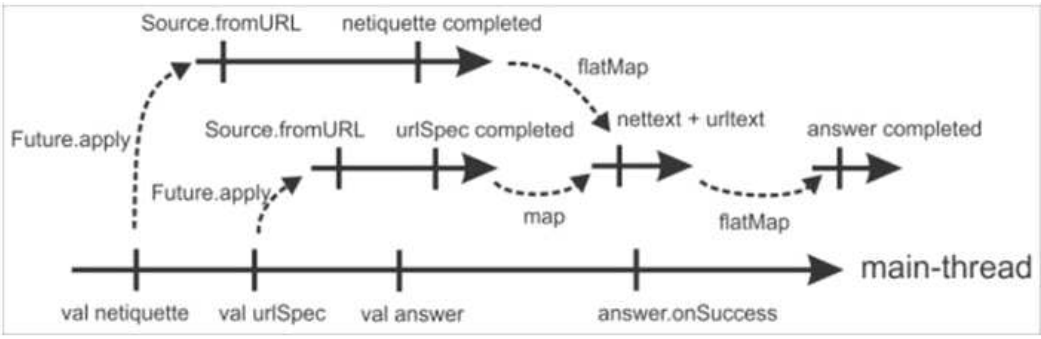
연구 목표

사용자 정의 클래스의 숫자 및 Script 동시 실행 개수를 측정하여 서버에서 정상 작동이 유효함을 확인한다.

개념

1. 동시성

이것은 구성의 문제로 여러 가지의 일들이 어떤 식으로 전개되어 있느냐, 어떻게 구조화할 것이냐에 관한 것이다. 두 가지 일이 같이 어우러져 발생하며 그 일에 대한 순서를 블록킹/시퀀셜하지 않게 만드느냐에 대한 문제이다. 현실적으로 가장많이 사용되는 것은 I/O가 일어나는 곳임.



2. 병렬성

이것은 계산의 문제로 동일한 시간에 여러 계산들이 동시에 이루지는 것을 말한다. 예를 들어 1부터 1억을 더하는데, 하나의 쓰레드에서 모두 더 할 수도 있지만, 10개의 쓰레드에서 양을 나누어서 계산한 후 최종적으로 더 할 수 있다. 그 때 10개의 쓰레드는 병렬성을 가지고 있다고 함.

		2개 동시	3개 동시	4개 동시	5개 동시	6개 동시	7개 동시	8개 동시	9개 동시	10개 동시
2 Class	메모리	2.3 G	3.8 G	4.8 G	6.5 G	7.5	8.5	9.5 G	10.5	11.5 G
	시간	110 s	160 s	200 s	240 s	280	320	360 s	400	450 s
3 Class	메모리	2.4 G	3.7 G							11.8 G
	시간	140 s	200 s	260	320	380	440	500	560	620 s

동시성과 병렬성의 개념을 활용하여 부하테스트에 대한 실험을 진행하였음.

연구 노트

연구 목표

앞서 실험했던 사용자 정의 클래스의 숫자 및 Script 동시 실행 개수를 늘려 CPU 점유율 및 메모리 사용량, 실행 속도 측정하여 서버에서 정상 작동이 유효함을 확인한다.

개념

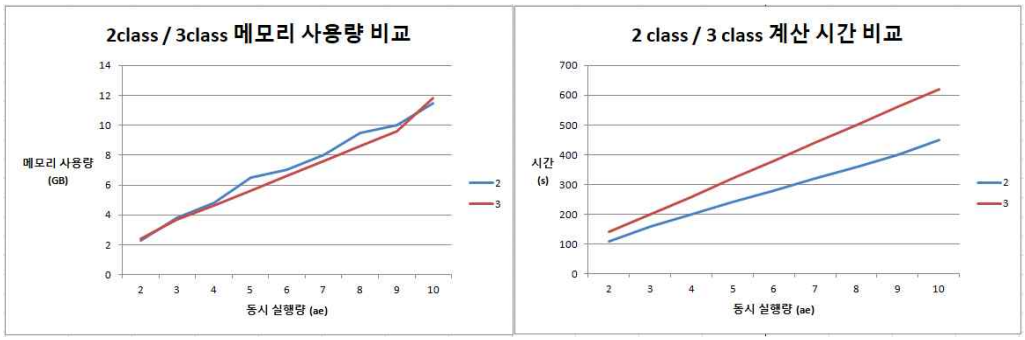
1. 동시 실행 테스트

		2개 동시	3개 동시	4개 동시	5개 동시	6개 동시	7개 동시	8개 동시	9개 동시	10개 동시
2 Class	메모리	2.3 G	3.0 G	4.0 G	6.5 G	7.5	8.5	9.5 G	10.5	11.5 G
	시간	110 s	160 s	200 s	240 s	280	320	360 s	400	450 s
3 Class	메모리	2.4 G	3.7 G	4.6	5.6	6.6	7.6	8.6	9.6	11.8 G
	시간	140 s	200 s	260	320	380	440	500	560	620 s

		2개 동시	3개 동시	4개 동시	5개 동시	6개 동시	7개 동시	8개 동시	9개 동시	10개 동시
4 class	메모리									11.6 G (9.6 G) swap 49.5-76
	시간									850 s
5 class	메모리									11.7 G swap 76-132
	시간									1040 s
6 class	메모리									12.2 G swap 132-185
	시간									1260 s

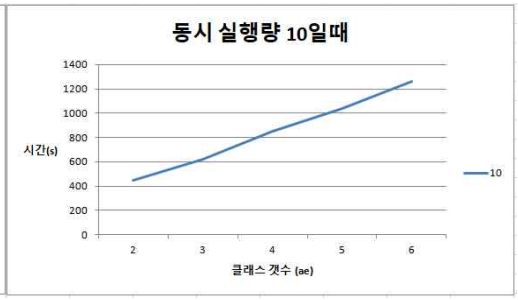
기존에 동시 실행 테스트에서 사용자 정의 클래스 2개와 3개에서의 CPU점유율과 메모리 사용량이 여유가 있음을 확인하였고, 그에 따라 클래스의 개수민 동시 실행량을 메모리 허용 범위 내에서 최대범위를 사용하여 테스트를 진행함.

2. 메모리 및 계산시간 측정



위의 표는 사용자 정의 클래스가 2개와 3개 일 때 동시 실행횟수를 2~10까지 변화시키며 메모리 사용량을 측정한 것으로 동시 실행량의 증가에 따라 메모리 사용량이 비례적으로 늘어난다는 것을 알 수 있음.

클래스의 개수가 2~6으로 늘어날 때 메모리 사용량의 변화는 크지 않았고, 시간이 클



래스의 개수에 비례하여 점진적으로 늘어난다는 것을 확인할 수 있었음.

연구 노트

연구 목표

스크립트 호출의 불확실성을 제거하고, class_name을 호출하는 부분을 수정하여 스크립트의 정상 작동 유무를 검사한다.

개념

1. 스크립트 실행 조건 변경

```
!python kang_to_jo_combine_script7.py --uc_userid="jys" --uc_name="uc1|uc2"
 896/3100 [=====>........................] - ETA: 0s - loss: 0.0028 - acc: 0.9978
1152/3100 [======>.......................] - ETA: 0s - loss: 0.0023 - acc: 0.9983
1408/3100 [=======>......................] - ETA: 0s - loss: 0.0019 - acc: 0.9986
1664/3100 [========>.....................] - ETA: 0s - loss: 0.0017 - acc: 0.9988
1920/3100 [=========>....................] - ETA: 0s - loss: 0.0018 - acc: 0.9990
2176/3100 [==========>...................] - ETA: 0s - loss: 0.0016 - acc: 0.9991
2432/3100 [===========>..................] - ETA: 0s - loss: 0.0014 - acc: 0.9992
2688/3100 [============>.................] - ETA: 0s - loss: 0.0013 - acc: 0.9993
2944/3100 [===========================>..] - ETA: 0s - loss: 0.0012 - acc: 0.9993
3100/3100 [==============================] - 1s 342us/step - loss: 0.0016 - acc: 0.9990 - val_loss: 0.0892 - val_acc: 0.9835
```

사용자 정의 클래스 이름의 중복값 허용은 스크립트 호출의 불확실성을 초래함. 그에 따라 스크립트 실행 명령어 중에 아래와 같이 '--uc_name' 부분의 클래스 이름을 '|'로 구분하여 불확실성을 제거함.

2. 사용자 정의 클래스 호출 방법 변경

```
cluster_class_dictionary = {}

for inx, aname in enumerate(uc_name):
    cluster_class_keys = list(cluster_class_dictionary.keys())
    cluster_class_number = len(cluster_class_dictionary)
    if not uc_name[inx] in cluster_class_keys:
        cluster_class_dictionary[uc_name[inx]] = cluster_class_number

with open(os.path.join(features_dir, 'cluster_class.json'), mode='w', encoding='utf-8') as js:
    js.write(json.dumps(cluster_class_dictionary))

print(cluster_class_dictionary)
```

사용자 정의 클래스 호출은 앞서 스크립트 불확실성의 제거를 통해 위와 같은 코드로 진행이 된다. 위와 마찬가지로 불확실성의 제거부분을 검사 및 스크립트 실행 정상작동 유뮤를 파악하였음.