State Farm Distracted Driver Detection

- 데이터 셋 EDA
- 개발 환경 : Kaggle Notebook
- 참조 링크 : https://www.kaggle.com/code/pierrelouisdanieau/computer-vision-tips-to-increase-accuracy

학습 목표

- 데이터를 불러와 이를 전처리를 수행해 본다.
- 불러온 이미지를 시각화를 수행한다.
- CNN를 활용하여 기본 모델을 만든다.

대회 개요

- 문제 유형 : Multi-class Classification(다중 클래스 분류)
- 평가 척도 : Multi-class Logarithmic Loss
- 내용 : CDC 자동차 안전 본부에 따르면, 자동차 사건 5건 중 1건은 산만한 운전자로 인해 발생. 스테이트 팜은 대시보드 카메라를 통해 주의가 산만한 운전자들을 자동으로 감지하여 고객을 보호하고 사건을 예방하고자 함.
  - 학습용, 테스트 데이터 셋 양이 적절함.
  - 이미지 대회 중, 커널 공유 내용에 대해 저작권 이슈가 없다.
- 목표 : 차량내에 설치된 대시보드 카메라 이미지 세트를 기반으로 스테이트 팜은 캐글러들에게 각 운전자들의 행동을 분류해 주기를 바란다.
- 제공 데이터 셋
  - 2만개의 학습 데이터와 8만개의 테스트 데이터를 제공

01. 라이브러리 및 데이터 불러오기

In [ ]:

```
import os
from glob import glob
import random
import time
import tensorflow
import datetime

# 데이터 처리 및 시각화
import numpy as np
import pandas as pd
%matplotlib inline
from IPython.display import display, Image
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.image as mpimg
import cv2

# 경고 메시지
import warnings
warnings.filterwarnings('ignore')

# 머신러닝 라이브러리
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_files
from sklearn.utils import shuffle
from sklearn.metrics import log_loss
```

In [ ]:

```
os.environ['KERAS_BACKEND'] = 'tensorflow'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # 3 = INFO, WARNING, and ERROR
from tqdm import tqdm
from IPython.display import FileLink

# 딥러닝 라이브러리
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dro
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.utils import to_categorical
```

In [ ]:

```
# 데이터 셋 확인
base_path = "/kaggle/input/state-farm-distracted-driver-detection/"
dataset = pd.read_csv(base_path + 'driver_imgs_list.csv')
dataset.head(5)
```

Out[ ]:

	subject	classname	img
0	p002	c0	img_44733.jpg
1	p002	c0	img_72999.jpg
2	p002	c0	img_25094.jpg
3	p002	c0	img_69092.jpg
4	p002	c0	img_92629.jpg

In [ ]:

```
### 몇개의 subject로 이루어져 있는가?
print( len(dataset['subject'].unique()) )
dataset['subject'].unique()
```

```
26
```

Out[ ]:

```
array(['p002', 'p012', 'p014', 'p015', 'p016', 'p021', 'p022', 'p024',
       'p026', 'p035', 'p039', 'p041', 'p042', 'p045', 'p047', 'p049',
       'p050', 'p051', 'p052', 'p056', 'p061', 'p064', 'p066', 'p072',
       'p075', 'p081'], dtype=object)
```

드라이버당 몇 장의 평균 이미지가 있는가?

In [ ]:

```
# subject feature로 그룹화
by_drivers = dataset.groupby('subject')

# Groupby unique drivers
# 몇명의 드라이버가 있을까?
unique_drivers = by_drivers.groups.keys() # drivers id
print('몇명의 드라이버 : ',len(unique_drivers), '명')
mean_driver = round(dataset.groupby('subject').count()['classname'].mean())
print('한명의 드라이버당 평균 이미지 개수 :',mean_driver, ' ')
```

```
몇명의 드라이버 :  26 명
한명의 드라이버당 평균 이미지 개수 : 862
```

02. 데이터 전처리 및 정규화 (loading and normalization)

The 10 classes to classify are :

- c0: safe driving - 안전 운전
- c1: texting - right - 오른손으로 문자
- c2: talking on the phone - right - 오른손으로 전화
- c3: texting - left - 왼손으로 문자
- c4: talking on the phone - left - 왼손으로 전화
- c5: operating the radio - 라디오 조작
- c6: drinking - 음료수 섭취
- c7: reaching behind - 뒷자석에 손 뻗기
- c8: hair and makeup - 얼굴, 머리 만지기
- c9: talking to passenger - 조수석과 대화

In [ ]:

```
NUMBER_CLASSES = 10 # 10 classes
```

In [ ]:

```
# opencv를 이용하여 이미지 읽기
# path : 이미지 경로
# img_rows, img_cols : 이미지 행, 열
# color_type : 이미지
def get_cv2_image(path, img_rows, img_cols, color_type=3):
    """
    Function that return an opencv image from the path and the right number o
    """
    if color_type == 1: # Loading as Grayscale image - Gray이미지로 불러오기
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    elif color_type == 3: # Loading as color image - 컬러 이미지로 불러오기
        img = cv2.imread(path, cv2.IMREAD_COLOR)

    img = cv2.resize(img, (img_rows, img_cols)) # Reduce size
    return img
```

In [ ]:

```
# 학습용 데이터 셋 불러오기
def load_train(img_rows, img_cols, color_type=3):
    """
    지정된 데이터 셋 경로로 부터 이미지와 label들의 정보를 가져온다.
    """
    train_images = []
    train_labels = []

    # 학습용 데이터 셋 폴더를 살펴보기
    for classed in tqdm(range(NUMBER_CLASSES)):
        print('Loading directory c{}'.format(classed))

        base_cPath = "/kaggle/input/state-farm-distracted-driver-detection/im
        files = glob(os.path.join(base_cPath + str(classed), '*.jpg'))

        # 파일을 지정된 사이즈로 불러온다.
        for file in files:
            img = get_cv2_image(file, img_rows, img_cols, color_type)
            train_images.append(img)
            train_labels.append(classed)

    return train_images, train_labels
```

In [ ]:

```
# 주어진 데이터 셋을 label을 원핫 처리하고, 지정된 이미지로 변환
def read_and_normalize_train_data(img_rows, img_cols, color_type):
    """
    Load + categorical + split
    """
    # 학습용 데이터 가져오기
    X, labels = load_train(img_rows, img_cols, color_type)
    #y = np_utils.to_categorical(labels, 10) #categorical train label

    # 가져온 데이터 전처리(원핫)
    y = to_categorical(labels, 10) #categorical train label

    # 데이터 셋. 모델 평가를 위해 학습용 데이터 셋을 학습과 테스트 데이터셋으로 나누기
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 
    x_train = np.array(x_train, dtype=np.uint8).reshape(-1,img_rows,img_cols,
    x_test = np.array(x_test, dtype=np.uint8).reshape(-1,img_rows,img_cols,co

    return x_train, x_test, y_train, y_test
```

테스트 데이터 셋 불러오기

In [ ]:

```
# 검증 데이터 셋을 불러오기 - Loading validation dataset
def load_test(size=200000, img_rows=64, img_cols=64, color_type=3):
    """
    Same as above but for validation dataset
    """


    base_cPath = "/kaggle/input/state-farm-distracted-driver-detection/imgs/te
    path = os.path.join(base_cPath, '*.jpg')

    files = sorted(glob(path))
    X_test, X_test_id = [], []
    total = 0

    files_size = len(files)
    for file in tqdm(files):
        if total >= size or total >= files_size:
            break
        file_base = os.path.basename(file)
        img = get_cv2_image(file, img_rows, img_cols, color_type)
        X_test.append(img)
        X_test_id.append(file_base)
        total += 1

    return X_test, X_test_id
```

In [ ]:

```
# 검증용 데이터 셋 전처리
# 자료형 변환(uint8), 자료형 형변환
def read_and_normalize_sampled_test_data(size, img_rows, img_cols, color_type
    test_data, test_ids = load_test(size, img_rows, img_cols, color_type)
    test_data = np.array(test_data, dtype=np.uint8)
    test_data = test_data.reshape(-1,img_rows,img_cols,color_type)
    return test_data, test_ids
```

In [ ]:

```
# 이미지 크기 (64, 64)
img_rows = 64
img_cols = 64
color_type = 1 # grey
nb_test_samples = 200

# train images 불러오기
x_train, x_test, y_train, y_test = read_and_normalize_train_data(img_rows, im

# validation images 불러오기 - 200개
test_files, test_targets = read_and_normalize_sampled_test_data(nb_test_sampl
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
Loading directory c0
 10%|█         | 1/10 [00:21<03:14, 21.59s/it]
```

```
Loading directory c1
 20%|██        | 2/10 [00:41<02:44, 20.59s/it]
Loading directory c2
 30%|███       | 3/10 [01:02<02:24, 20.62s/it]
Loading directory c3
 40%|████      | 4/10 [01:22<02:02, 20.46s/it]
Loading directory c4
 50%|█████     | 5/10 [01:42<01:41, 20.28s/it]
Loading directory c5
 60%|██████    | 6/10 [02:02<01:21, 20.36s/it]
Loading directory c6
 70%|███████   | 7/10 [02:23<01:01, 20.45s/it]
Loading directory c7
 80%|████████  | 8/10 [02:41<00:39, 19.57s/it]
Loading directory c8
 90%|█████████ | 9/10 [02:58<00:18, 18.77s/it]
Loading directory c9
100%|██████████| 10/10 [03:16<00:00, 19.70s/it]
  0%|          | 200/79726 [00:02<13:44, 96.47it/s]
```

03. 데이터 EDA

In [ ]:

```
base_tr_path = "/kaggle/input/state-farm-distracted-driver-detection/imgs/tra
base_test_path = "/kaggle/input/state-farm-distracted-driver-detection/imgs/t

names = [item[0:19] for item in sorted(glob(base_tr_path))]
print(names)
test_files_size = len(np.array(glob(os.path.join(base_test_path, '*.jpg'))))

x_train_size = len(x_train)
categories_size = len(names)
x_test_size = len(x_test)

print()
print('전체 이미지 : %s ' % (test_files_size + x_train_size + x_test_size))
print('학습용 이미지 : %d ' % x_train_size)
print('총 학습용 이미지 종류(c0~c*) : %d ' % categories_size)
print('검증용 이미지 : %d ' % x_test_size)
print('테스트 이미지 : %d '% test_files_size)
```

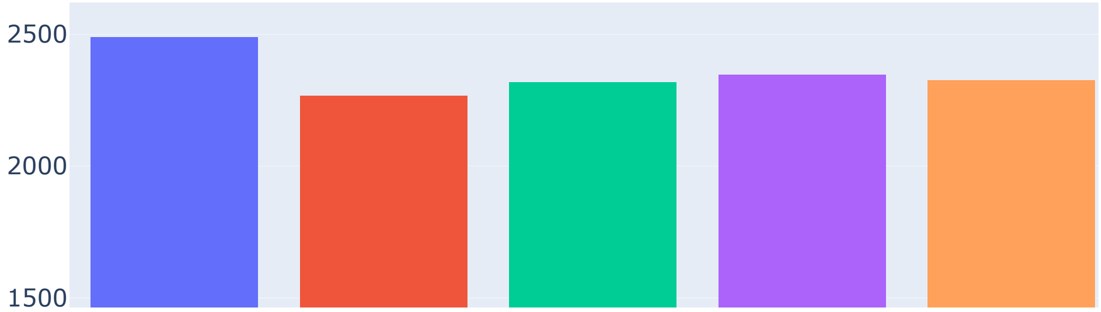
```
['/kaggle/input/state', '/kaggle/input/state', '/kaggle/input/state', '/kaggl
e/input/state', '/kaggle/input/state', '/kaggle/input/state', '/kaggle/input/s
tate', '/kaggle/input/state', '/kaggle/input/state', '/kaggle/input/state']

전체 이미지 : 102150
학습용 이미지 : 17939
총 학습용 이미지 종류(c0~c*) : 10
검증용 이미지 : 4485
테스트 이미지 : 79726
```

데이터 시각화

In [ ]:

```
import plotly.express as px

px.histogram(dataset, x="classname", color="classname", title="Number of imag
```

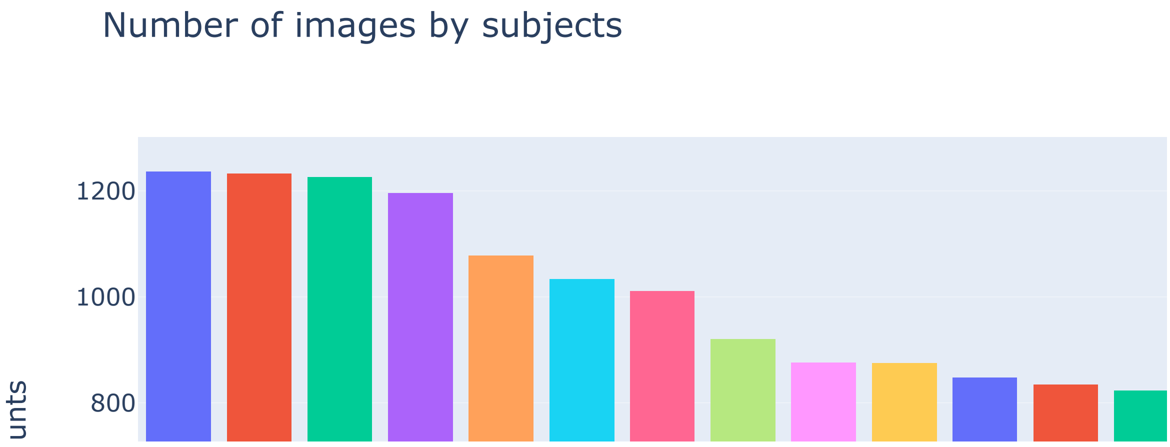
Number of images by categories



- 각 이미지의 분포는 좋은 편이다.

In [ ]:

```
# 각 드라이버 당 이미지의 빈도 확인
drivers_id = pd.DataFrame((dataset['subject'].value_counts()).reset_index())
drivers_id.columns = ['driver_id', 'Counts']
px.histogram(drivers_id, x="driver_id",y="Counts" ,color="driver_id", title="
```

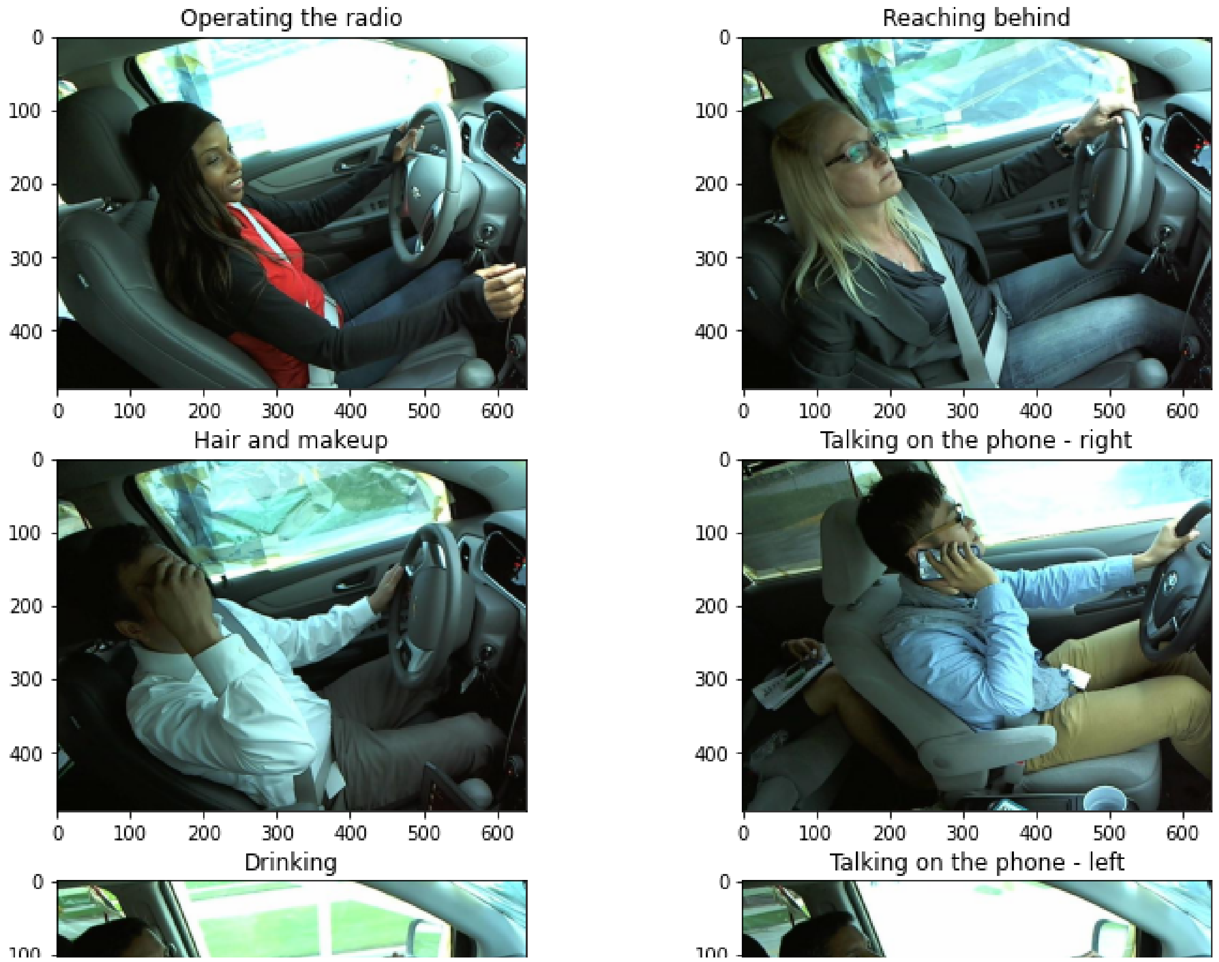


이미지 살펴보기

- C0 ~ C9는 어떤 이미지인지 직접 확인해 보기

In [ ]:

```
activity_map = {'c0': 'Safe driving',
                'c1': 'Texting - right',
                'c2': 'Talking on the phone - right',
                'c3': 'Texting - left',
                'c4': 'Talking on the phone - left',
                'c5': 'Operating the radio',
                'c6': 'Drinking',
                'c7': 'Reaching behind',
                'c8': 'Hair and makeup',
                'c9': 'Talking to passenger'}

plt.figure(figsize = (12, 20))
image_count = 1
BASE_URL = '../input/state-farm-distracted-driver-detection/imgs/train/'
for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(5, 2, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(activity_map[directory])
```



Texting - left

Texting - right

Talking to passenger

Safe driving

```
In [ ]: # 제출용 파일 만들기
        def create_submission(predictions, test_id, info):
            result = pd.DataFrame(predictions, columns=['c0', 'c1', 'c2', 'c3', 'c4',
            result.loc[:, 'img'] = pd.Series(test_id, index=result.index)

            now = datetime.datetime.now()

            if not os.path.isdir('kaggle_submissions'):
                os.mkdir('kaggle_submissions')

            suffix = "{}_{}".format(info,str(now.strftime("%Y-%m-%d-%H-%M")))
            sub_file = os.path.join('kaggle_submissions', 'submission_' + suffix + '.

            result.to_csv(sub_file, index=False)

            return sub_file
```

04. CNN 모델

구조 :

- 3 Convolutionnal layers (with Relu, Maxpooling and dropout)
- A flatten layer
- 2 Dense layers with Relu and Dropouts
- 1 Dense layer with softmax for the classification

In [ ]:

```
# 배치 사이즈(batch size)와 에폭(epochs)
batch_size = 40 #40
nb_epoch = 6 #10
```

In [ ]:

```
# 학습시, 모델 저장하기
models_dir = "saved_models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

checkpointer = ModelCheckpoint(filepath='saved_models/weights_best_vanilla.hd
                               monitor='val_loss', mode='min',
                               verbose=1, save_best_only=True)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
#callbacks = [checkpointer, es]
```

모델 구축

In [ ]:

```
def create_model():
    model = Sequential()

    ## CNN 1
    model.add(Conv2D(32,(3,3),activation='relu',input_shape=(img_rows, img_co
    model.add(BatchNormalization())
    model.add(Conv2D(32,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 2
    model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 3
    model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.5))

    ## Output
    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(10,activation='softmax'))

    return model
```

In [ ]:

```
model = create_model()

# 상세 모델 정보
model.summary()

# 모델 컴파일(Compiling the model)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        320       
_________________________________________________________________
batch_normalization (BatchNo (None, 62, 62, 32)        128       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 62, 62, 32)        9248      
_________________________________________________________________
batch_normalization_1 (Batch (None, 62, 62, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 31, 31, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 31, 31, 64)        256       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 31, 64)        36928     
_________________________________________________________________
batch_normalization_3 (Batch (None, 31, 31, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856     
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       512       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       147584    
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 128)       512       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               4194816   
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 4,552,042
Trainable params: 4,550,122
Non-trainable params: 1,920
_________________________________________________________________
```

모델 학습

In [ ]:

```
%%time

history = model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=nb_epoch, batch_size=batch_size, verbose=1,
          callbacks = [checkpointer, es])
```

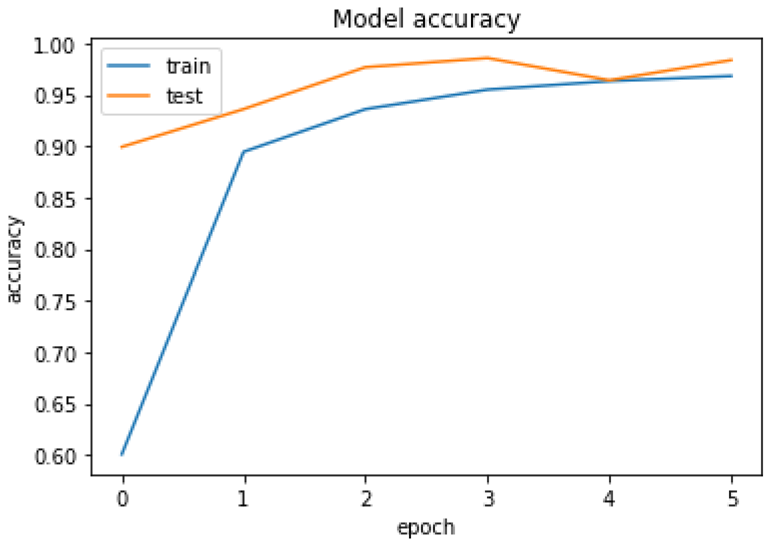
```
# model.load_weights('saved_models/weights_best_vanilla.hdf5')
print('History of the training',history.history)
```

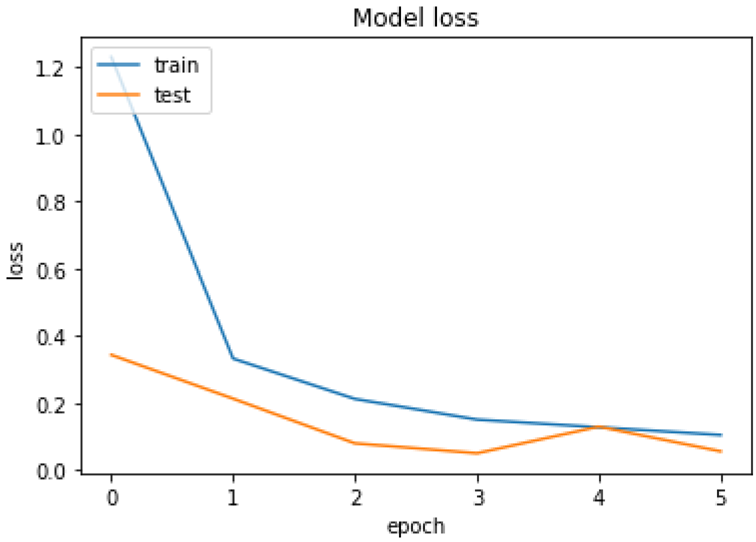
```
Epoch 1/6
449/449 [==============================] - 16s 16ms/step - loss: 1.2295 - accu
racy: 0.6005 - val_loss: 0.3431 - val_accuracy: 0.8992

Epoch 00001: val_loss improved from inf to 0.34307, saving model to saved_mode
ls/weights_best_vanilla.hdf5
Epoch 2/6
449/449 [==============================] - 7s 15ms/step - loss: 0.3321 - accur
acy: 0.8944 - val_loss: 0.2126 - val_accuracy: 0.9360

Epoch 00002: val_loss improved from 0.34307 to 0.21259, saving model to saved_
models/weights_best_vanilla.hdf5
Epoch 3/6
449/449 [==============================] - 7s 15ms/step - loss: 0.2118 - accur
acy: 0.9359 - val_loss: 0.0794 - val_accuracy: 0.9768

Epoch 00003: val_loss improved from 0.21259 to 0.07943, saving model to saved_
models/weights_best_vanilla.hdf5
Epoch 4/6
449/449 [==============================] - 7s 15ms/step - loss: 0.1501 - accur
acy: 0.9550 - val_loss: 0.0498 - val_accuracy: 0.9857

Epoch 00004: val_loss improved from 0.07943 to 0.04982, saving model to saved_
models/weights_best_vanilla.hdf5
Epoch 5/6
449/449 [==============================] - 7s 15ms/step - loss: 0.1273 - accur
acy: 0.9633 - val_loss: 0.1292 - val_accuracy: 0.9641

Epoch 00005: val_loss did not improve from 0.04982
Epoch 6/6
449/449 [==============================] - 7s 15ms/step - loss: 0.1044 - accur
acy: 0.9682 - val_loss: 0.0556 - val_accuracy: 0.9837

Epoch 00006: val_loss did not improve from 0.04982
Epoch 00006: early stopping
History of the training {'loss': [1.229490041732788, 0.3320949673652649, 0.211
79638803005219, 0.15011906623840332, 0.127284973859787, 0.10437912493944168],
'accuracy': [0.6004794239997864, 0.8944199681282043, 0.9359496235847473, 0.955
0142288208008, 0.963320136070251 5, 0.9682256579399109], 'val_loss': [0.3430742
32339859, 0.21258755028247833, 0.07943043112754822, 0.04982385411858559, 0.129
21278178691864, 0.05558828264474869], 'val_accuracy': [0.8992196321487427, 0.9
360089302062988, 0.9768115878105164, 0.9857302308082581, 0.964102566242218, 0.
983723521232605]}
CPU times: user 41.9 s, sys: 2.95 s, total: 44.9 s
Wall time: 49.9 s
```

- cpu 학습 시간
  - CPU times: user 1h 15min 22s, sys: 35.6 s, total: 1h 15min 57s
  - Wall time: 21min 35s

In [ ]:

```
# 학습 결과 시각화
def plot_train_history(history):
    """
    Plot the validation accuracy and validation loss over epochs
    """
    # Summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # Summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

plot_train_history(history)
```





테스트 셋. 결과 예측

In [ ]:

```
def plot_test_class(model, test_files, image_number, color_type=1):
    """
    Function that tests or model on test images and show the results
    """
    img_brute = test_files[image_number]
    img_brute = cv2.resize(img_brute,(img_rows,img_cols))
    plt.imshow(img_brute, cmap='gray')

    new_img = img_brute.reshape(-1,img_rows,img_cols,color_type)

    y_prediction = model.predict(new_img, batch_size=batch_size, verbose=1)
    print('Y prediction: {}'.format(y_prediction))
    print('Predicted: {}'.format(activity_map.get('c{}'.format(np.argmax(y_pr
    plt.show()
```

In [ ]:

```
score1 = model.evaluate(x_test, y_test, verbose=1)
score1
```

```
141/141 [==============================] - 1s 4ms/step - loss: 0.0556 - accura
cy: 0.9837
```

Out[ ]:

```
[0.05558827146887779, 0.983723521232605]
```

In [ ]:

```
print('Loss: ', score1[0])
print('Accuracy: ', score1[1]*100, ' %')
```

```
Loss:  0.05558827146887779
Accuracy:  98.3723521232605  %
```