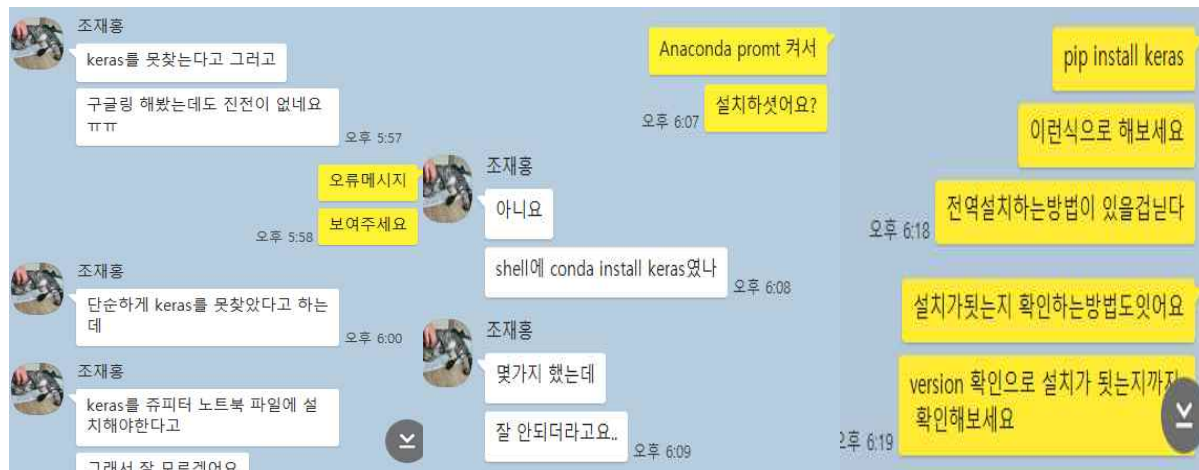
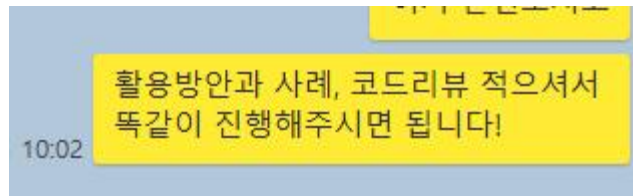


팀 프로젝트 지도 활동 보고서

교과목명	AI및데이터분석의 기초		
강사	한상호 교수		
활동 일자	2020년 09월 19일 (토요일)		
학과	성명	연락처	서명
스마트시스템SW학과	정학제	010-3895-1306	
스마트시스템SW학과	정재윤	010-9008-4847	
스마트시스템SW학과	정해문	010-7176-4289	
스마트시스템SW학과	조재홍	010-4062-9523	
활동 내용	<p>(미팅시 팀원들 사진 붙이기 - 선택사항)</p> <p>- 미팅 결과물에 대한 URL: https://github.com/Hott-J/AI-BigData-Basic 팀장 github 계정 아래 project repo를 하나 만들고 (repo 이름은 project 취지에 맞게 작명), 코드 파일들, PPT 파일 등을 commit 하고 해당 파일 접근을 위한 URL을 여기에 작성하기</p> <p>- 2페이지에 작성</p>		

< 정학제 팀장 >



과제 제시와 팀원들과 단톡방에서 오류 핸들링 및 전체적인 작업을 도와주면서 소통하였습니다. 이후, 팀원들이 보내준 내용을 바탕으로 보고서를 작성하였습니다.

< 조재홍 팀원 >

CNN의 활용

CNN은 딥러닝 알고리즘의 하나이며 피드포워드 신경망(feed-forward neural network) 같은 말로 인공신경망(artificial neural network)로부터 시작되었고 이는 사람들의 뇌가 뉴런들이 층을 지며 연결되어 있는 모습을 모방하여 만들어 졌다.

그러나 신경망의 층수가 늘어나면서 역전파 현상이 발생하여 신경망 내의 가중치들이 제대로 학습되지 않는 경사감소소멸 문제가 발생하였다. 이를 해결하기 위해 각층에 사전학습을 통해 가중치의 값을 해와 근접한 값으로 보정한 후에 최종 가중치를 계산하는 신뢰심층망 등의 방법이 도입되었다.

이후 사람의 시각인지 과정을 모방하여 지금의 CNN이 개발되었으며 CNN은 이미지 분류, 객체 검출, 사진을 분류하고 완벽히 이해하는 Semantic Segmentation 등 여러 기능을 가지고 있으며, 현재 실생활에서 자주 이용되고 있다.

인터넷 쇼핑을 할 때, 상품의 스타일, 취향 등을 구분하여 사용자에게 비슷한 취향의 상품을 추천해준다. 실제로 얼마전 MBN에서는 인공지능 앵커를 활용하여 도입하기도 하였으며 그 외에도 자율주행 자동차, 휴대폰 카메라, 얼굴 인식, 의학 분야 등 여러 곳에서 활용되고 있다.

< 정해문 팀원 >

CNN(Convolution Neural Networks) 의 활용

딥러닝에 가장 많이 사용되는 알고리즘 으로 이미지에서 객체, 얼굴, 장면을 인식하기 위해 패턴을 찾는데 유용함

<자율주행자동차>

자율주행차에 장착된 카메라에서 취득한 영상을 분석해 자동차/보행차/이륜차/신호등 등의 도로 위 사물을 검출하고 인식하거나, 도로의 차선 및 노면 표시, 자유공간 인식 등에 딥러닝 기술을 주로 사용

의료 영상 판독

환자들의 진료에 대한 방대한 의료 유전정보, 생활습관정보 등의 빅데이터를 바탕으로

이를 분석하고 활용하는데 있어 핵심기술

질병 예측과 예방, 웨어러블 장비에서 입수되는 센서 데이터와 이미지 인식 기술을 결합한

진단데이터를 분석하여 최적화된 진단 방안을 제공

인터넷 쇼핑



기존에는 키워드를 입력해 적당한 상품 검색 결과를 내놓을 수 있도록 하기 위해서는 해당 상품에 맞는 키워드를 일일이 입력해야 했지만, CNN은 이런 작업들 중 많은 부분을 자동화함. 예를 들어 원피스라는 태그를 붙인 이미지를 CNN에 수없이 반복 입력해서 공통점을 뽑아내게 하면 이후에 여기에 새로운 이미지를 입력했을 때 이것이 원피스인지 아닌지를 구분해 냄

Keras로 수행한 CNN을 이용해서 MNIST 손글씨 데이터 출력해보기

```
#필요한 모듈 import 하기
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import sys
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import numpy as np
np.random.seed(7)
```

```

#mnist 데이터 불러오기
img_rows = 28
img_cols = 28

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

input_shape = (img_rows, img_cols, 1)
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

batch_size = 128
num_classes = 10
epochs = 12

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples

```

#모델 프레임 설정

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), padding='same',
                activation='relu',
                input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 32)	832

max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0

conv2d_1 (Conv2D)	(None, 14, 14, 64)	8256

max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0

dropout (Dropout)	(None, 7, 7, 64)	0

flatten (Flatten)	(None, 3136)	0

dense (Dense)	(None, 1000)	3137000

dropout_1 (Dropout)	(None, 1000)	0

dense_1 (Dense)	(None, 10)	10010
=====		

Total params: 3,156,098

Trainable params: 3,156,098

Non-trainable params: 0

#모델 실행 환경 설정

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
hist = model.fit(x_train, y_train,
                 batch_size=batch_size,
                 epochs=epochs,
                 verbose=1,
                 validation_data=(x_test, y_test))
```

```
Epoch 1/12
469/469 [=====] - 26s 55ms/step - loss: 0.1897 - accuracy: 0.9419 - val_loss: 0.0451
- val_accuracy: 0.9871
Epoch 2/12
469/469 [=====] - 25s 53ms/step - loss: 0.0614 - accuracy: 0.9806 - val_loss: 0.0310
- val_accuracy: 0.9902
Epoch 3/12
469/469 [=====] - 25s 53ms/step - loss: 0.0430 - accuracy: 0.9867 - val_loss: 0.0302
- val_accuracy: 0.9898
Epoch 4/12
469/469 [=====] - 25s 53ms/step - loss: 0.0363 - accuracy: 0.9884 - val_loss: 0.0237
- val_accuracy: 0.9927
Epoch 5/12
469/469 [=====] - 25s 53ms/step - loss: 0.0303 - accuracy: 0.9897 - val_loss: 0.0262
- val_accuracy: 0.9913
Epoch 6/12
469/469 [=====] - 25s 53ms/step - loss: 0.0265 - accuracy: 0.9912 - val_loss: 0.0248
- val_accuracy: 0.9914
Epoch 7/12
469/469 [=====] - 25s 53ms/step - loss: 0.0241 - accuracy: 0.9922 - val_loss: 0.0226
- val_accuracy: 0.9932
Epoch 8/12
469/469 [=====] - 25s 53ms/step - loss: 0.0200 - accuracy: 0.9934 - val_loss: 0.0237
- val_accuracy: 0.9929
Epoch 9/12
469/469 [=====] - 25s 52ms/step - loss: 0.0183 - accuracy: 0.9941 - val_loss: 0.0243
- val_accuracy: 0.9920
Epoch 10/12
469/469 [=====] - 25s 53ms/step - loss: 0.0157 - accuracy: 0.9947 - val_loss: 0.0236
- val_accuracy: 0.9930
Epoch 11/12
469/469 [=====] - 25s 53ms/step - loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0233
- val_accuracy: 0.9929
Epoch 12/12
469/469 [=====] - 25s 54ms/step - loss: 0.0134 - accuracy: 0.9956 - val_loss: 0.0263
- val_accuracy: 0.9920
```

5]: #테스트 정확도 출력

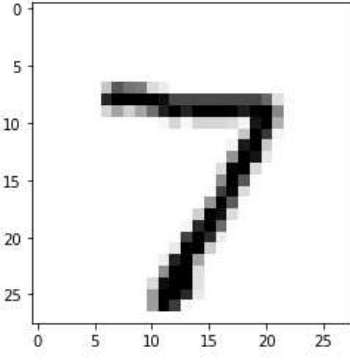
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.026275744661688805
Test accuracy: 0.9919999837875366
```



```
#이미지 불러오기
n = 0
plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()

print('The Answer is ', model.predict_classes(x_test[n].reshape((1, 28, 28, 1))))
```



```
WARNING:tensorflow:From <ipython-input-17-4e62bcab15ac>:5: Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead: * `np.argmax(model.predict(x), axis=-1)`, if your model does multi-class classification
(e.g. if it uses a `softmax` last-layer activation). * `(model.predict(x) > 0.5).astype("int32")`, if your mo
del does binary classification (e.g. if it uses a `sigmoid` last-layer activation).
The Answer is [7]
```

< 정재윤 팀원 >

Convolutional Neural Network

CNN은 자율주행자동차, 로봇틱스, 유명 화가의 작품 따라 하기, 몇개의 손글씨를 가지고 손글씨 생성하기, 빅스비 비전과 같은 사진을 찍으면 그 사진에 있는 텍스트를 추출해서 자동으로 번역해주기 등등 많은 곳에서 이미 쓰이고 있으며, CNN으로 나중에는 가짜 deepfake 영상까지 잡아내며 더욱 더 많은 분야에서 쓰일 예정이다.

오늘은 저번에 몰랐지만 정확도 향상을 위해서 새로 사용할 내용과 실습을 더욱 해보겠다

Batch normalization

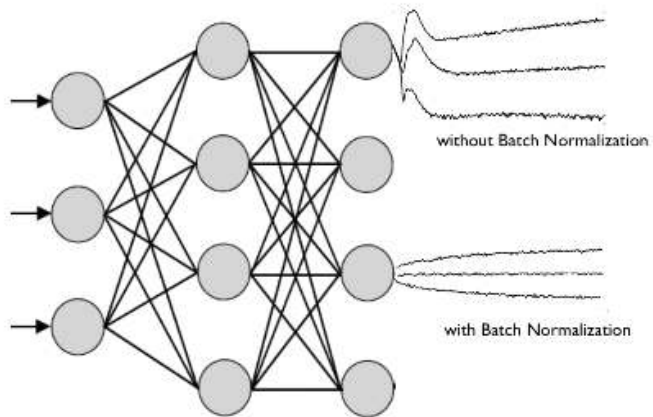
Dropout이라는 기법도 있지만 왜 배치 정규화를 사용하는가?

학습 속도가 개선된다 (학습률을 높게 설정할 수 있기 때문)
가중치 초깃값 선택의 의존성이 적어진다 (학습을 할 때마다 출력값을 정규화하기 때문)

과적합(overfitting) 위험을 줄일 수 있다 (드롭아웃 같은 기법 대체 가능)
Gradient Vanishing 문제 해결 등등의 이점이 있다고 한다.

입력 분포의 균일화

학습을 할 때 hidden layer의 중간에서 입력분포가 학습할 때마다 변화하면서 가중치가 엉뚱한 방향으로 갱신될 문제가 발생할 수 있다. 신경망의 층이 깊어질수록 학습 시에 가정했던 입력분포가 변화하여 엉뚱한 학습이 될 수 있다.



실습 및 실습 코드 (코드에 주석으로 설명이 되어있음)


```
[1] # 필요한 라이브러리 불러오기
import matplotlib.pyplot as plot
from keras.datasets import mnist
import tensorflow as tf
import keras
import tensorflow.keras.layers as Layers
import tensorflow.keras.activations as Actications
import tensorflow.keras.models as Models
import tensorflow.keras.optimizers as Optimizer
import tensorflow.keras.metrics as Metrics
import tensorflow.keras.utils as Utils
from keras.utils import to_categorical
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import numpy as np
from keras.utils import np_utils
```

```
[3] # 이미지 받아오기
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')/255.0
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')/255.0

y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)

print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)

x_train shape: (60000, 28, 28, 1)
y_train shape: (60000, 10)
```

[6] #커스텀 조금 깊은 CNN 및 batch normalization 추가

```
#모델 정의
model_2 = Models.Sequential()

#합성곱층 Conv2D(합성곱커널개수, 합성곱커널크기(높이,너비) - 3*3, 5*5를 많이 사용함,
# 매개변수에 렐루,패딩(없으면 기본값), 이미지의 높이, 너비, 컬러채널)
model_2.add(Layers.Conv2D(32, kernel_size=(3,3), input_shape = (28,28,1), activation='relu'))
model_2.add(Layers.Conv2D(64, kernel_size=(3,3), activation='relu'))

#풀링층(풀링의 (높이,너비), 스트라이드(없으면 기본값), 패딩(없으면 기본값)
model_2.add(Layers.MaxPool2D(pool_size = 2))
model_2.add(Layers.BatchNormalization()) #배치노멀리제이션 추가
model_2.add(Layers.Dropout(rate = 0.25)) #드롭아웃 추가

#완전 연결층에 연결하기 위해 Flatten 클래스를 수행해 배치 차원을 제외하고 일렬로 펼침 (2차원->1차원)
model_2.add(Layers.Flatten())

#마지막으로 완전 연결층을 추가(x개의 뉴런 사용, 렐루 활성화 함수 적용)
model_2.add(Layers.Dense(128, activation = 'relu'))

#맨 마지막에 클래스에 대응하는 개수의 뉴런 사용 및 소프트맥스 활성화 함수 적용
model_2.add(Layers.Dense(10, activation='softmax'))

#모델 컴파일(아담 옵티마이저, 정확도 관찰 위한 metrics에 accuracy), 옵티마이저 바꿔보기?
#compile() : 모델을 기계가 이해할 수 있도록 컴파일 합니다. 오차 함수와 최적화 방법, 메트릭 함수를 선택할 수 있습니다.
model_2.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'])

#모델 서머리 = 모델 구조 살펴보기
model_2.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
conv2d_3 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
batch_normalization (Batch Normalization)	(None, 12, 12, 64)	256
dropout (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 128)	1179776
dense_1 (Dense)	(None, 10)	1290
Total params: 1,200,138		
Trainable params: 1,200,010		
Non-trainable params: 128		

드롭아웃, 배치노멀리제이션은 오버피팅을 방지

[7] # fit() 메서드로 모델 훈련 시키기

```
trained = model_2.fit(x_train, y_train, epochs=20, batch_size = 128, verbose = 1, validation_split=0.30)
```

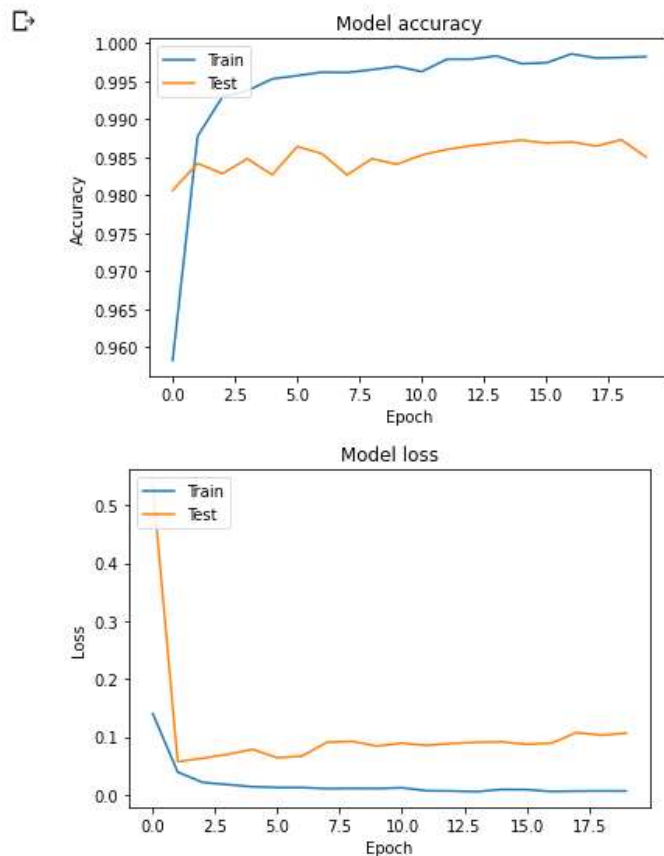
```
Epoch 1/20
329/329 [=====] - 2s 7ms/step - loss: 0.1401 - accuracy: 0.9582 - val_loss: 0.5347 - val_accuracy: 0.9806
Epoch 2/20
329/329 [=====] - 2s 7ms/step - loss: 0.0392 - accuracy: 0.9877 - val_loss: 0.0570 - val_accuracy: 0.9842
Epoch 3/20
329/329 [=====] - 2s 7ms/step - loss: 0.0214 - accuracy: 0.9929 - val_loss: 0.0631 - val_accuracy: 0.9828
Epoch 4/20
329/329 [=====] - 2s 7ms/step - loss: 0.0176 - accuracy: 0.9937 - val_loss: 0.0698 - val_accuracy: 0.9848
Epoch 5/20
329/329 [=====] - 2s 7ms/step - loss: 0.0139 - accuracy: 0.9953 - val_loss: 0.0785 - val_accuracy: 0.9827
Epoch 6/20
329/329 [=====] - 2s 7ms/step - loss: 0.0126 - accuracy: 0.9957 - val_loss: 0.0638 - val_accuracy: 0.9864
Epoch 7/20
329/329 [=====] - 2s 6ms/step - loss: 0.0126 - accuracy: 0.9962 - val_loss: 0.0671 - val_accuracy: 0.9854
Epoch 8/20
329/329 [=====] - 2s 6ms/step - loss: 0.0105 - accuracy: 0.9961 - val_loss: 0.0907 - val_accuracy: 0.9827
Epoch 9/20
329/329 [=====] - 2s 6ms/step - loss: 0.0112 - accuracy: 0.9965 - val_loss: 0.0923 - val_accuracy: 0.9848
Epoch 10/20
329/329 [=====] - 2s 6ms/step - loss: 0.0108 - accuracy: 0.9969 - val_loss: 0.0841 - val_accuracy: 0.9841
Epoch 11/20
329/329 [=====] - 2s 7ms/step - loss: 0.0122 - accuracy: 0.9962 - val_loss: 0.0891 - val_accuracy: 0.9853
Epoch 12/20
329/329 [=====] - 2s 6ms/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0853 - val_accuracy: 0.9860
Epoch 13/20
329/329 [=====] - 2s 7ms/step - loss: 0.0066 - accuracy: 0.9979 - val_loss: 0.0888 - val_accuracy: 0.9865
Epoch 14/20
329/329 [=====] - 2s 7ms/step - loss: 0.0054 - accuracy: 0.9983 - val_loss: 0.0905 - val_accuracy: 0.9869
Epoch 15/20
329/329 [=====] - 2s 7ms/step - loss: 0.0092 - accuracy: 0.9973 - val_loss: 0.0914 - val_accuracy: 0.9872
Epoch 16/20
329/329 [=====] - 2s 6ms/step - loss: 0.0090 - accuracy: 0.9974 - val_loss: 0.0872 - val_accuracy: 0.9868
Epoch 17/20
329/329 [=====] - 2s 6ms/step - loss: 0.0057 - accuracy: 0.9986 - val_loss: 0.0891 - val_accuracy: 0.9870
Epoch 18/20
329/329 [=====] - 2s 6ms/step - loss: 0.0064 - accuracy: 0.9980 - val_loss: 0.1075 - val_accuracy: 0.9864
Epoch 19/20
329/329 [=====] - 2s 6ms/step - loss: 0.0067 - accuracy: 0.9981 - val_loss: 0.1030 - val_accuracy: 0.9873
Epoch 20/20
329/329 [=====] - 2s 7ms/step - loss: 0.0065 - accuracy: 0.9982 - val_loss: 0.1064 - val_accuracy: 0.9850
```

```

plot.plot(trained.history['accuracy'])
plot.plot(trained.history['val_accuracy'])
plot.title('Model accuracy')
plot.ylabel('Accuracy')
plot.xlabel('Epoch')
plot.legend(['Train', 'Test'], loc='upper left')
plot.show()

plot.plot(trained.history['loss'])
plot.plot(trained.history['val_loss'])
plot.title('Model loss')
plot.ylabel('Loss')
plot.xlabel('Epoch')
plot.legend(['Train', 'Test'], loc='upper left')
plot.show()

```



```

[16] # 테스트 데이터로 정확도 측정하기
test_loss, test_acc = model_2.evaluate(x_test, y_test)
print('test_acc: ', test_acc)

313/313 [=====] - 1s 2ms/step - loss: 0.1000 - accuracy: 0.9872
test_acc: 0.9872000217437744

```

약 0.9872의 정확도가 나온것을 볼 수 있다.

팀프로젝트 지도 수행과 관련하여 상기와 같이 활동하였음을 확인합니다.

2020 년 11월 14 일

담당 교수 : 한 상 호 (인)