|  |  |
| --- | --- |
| 년도-학기 | 2021년 1학기 |
| 과목명 | 임베디드시스템설계 |

|  |  |
| --- | --- |
| **LAB번호** | **제목** |
| 12 | Lab Cortex-M AI |

|  |  |
| --- | --- |
| 실험 일자 | 2021년 5월 28일 |
| 제출자 이름 | 강\*\* |
| 제출자 학번 | 201803\*\*\*\* |
| 팀원 이름 |  |
|  |  |

**Chapter 1. 프로그램의 동작 방식 설명**

1-1. mnist\_mlp.py 를 실행해 training 한다.

데이터셋을 생성한다.

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.reshape(60000, 784)

x\_test = x\_test.reshape(10000, 784)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

모델을 구성한다.

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(num\_classes, activation='softmax'))

모델의 학습과정을 설정하고 학습시킨다.

model.compile(loss='categorical\_crossentropy',

optimizer=RMSprop(),

metrics=['accuracy'])

history = model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test, y\_test))

모델을 평가한다.

score = model.evaluate(x\_test, y\_test, verbose=0)

모델을 저장한다.

model.save('mnist\_mlp\_model.h5')

1-2. aiSystemPerformance.c를 실행하고 보드에 빌드한다.

1-3. send\_test.py를 실행해 훈련된 모델로 테스트한다.

**Chapter 2. 결과**

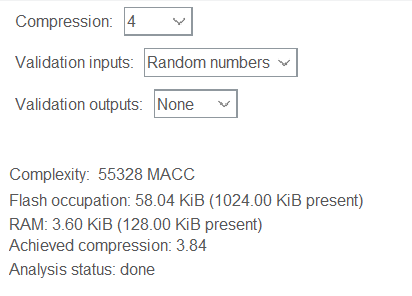
**2-1. 컴프레션 후 실행 결과**

Training 결과는 다음과 같다.

Test loss: 0.12700376453043655

Test accuracy: 0.9802

Compression을 하며 analyze한 결과로, 58.04 KiB 이다.

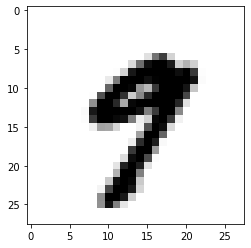


Test 결과는 모든 사진에 대해서 맞는 숫자가 예측되었다. 결과는 아래와 같다.

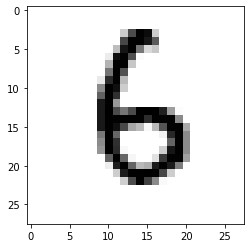
runfile('C:/work/Anaconda/send\_test.py', wdir='C:/work/Anaconda')

COM6 is open...

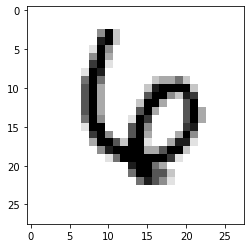
(10000, 28, 28)



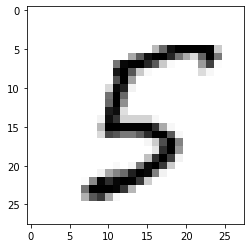
Recognized digit: 9.



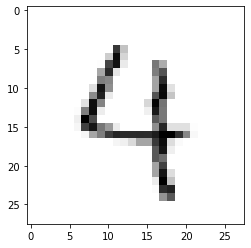
Recognized digit: 6.



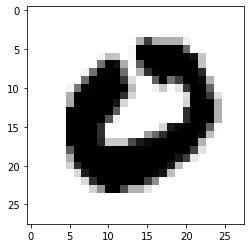
Recognized digit: 6.

￼

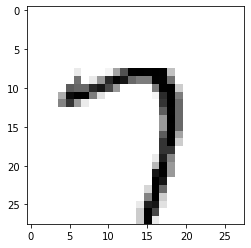
Recognized digit: 5.



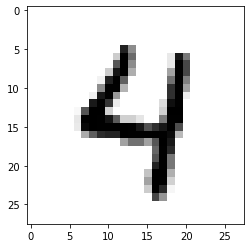
Recognized digit: 4.



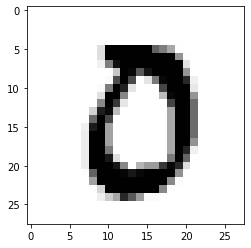
Recognized digit: 0.



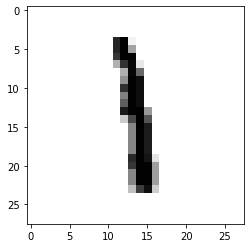
Recognized digit: 7.



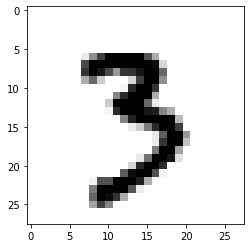
Recognized digit: 4.



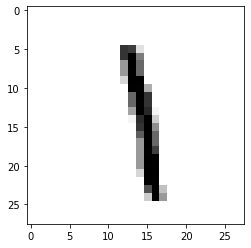
Recognized digit: 0.



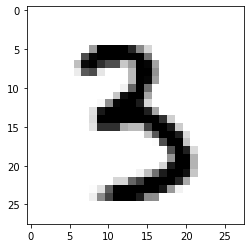
Recognized digit: 1.

￼

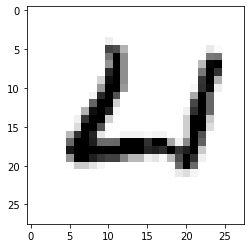
Recognized digit: 3.



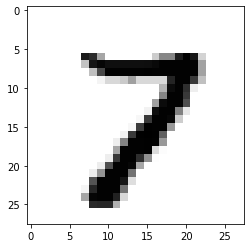
Recognized digit: 1.



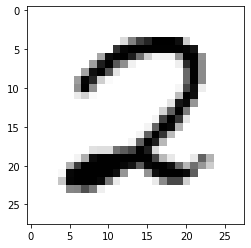
Recognized digit: 3.



Recognized digit: 4.



Recognized digit: 7.



Recognized digit: 2.

Results for "network", 16 inferences @168MHz/168MHz (complexity: 670880 MACC)

duration : 46.973 ms (average)

CPU cycles : 7891505 -436/+175 (average,-/+)

CPU Workload : 4% (duty cycle = 1s)

cycles/MACC : 11.76 (average for all layers)

used stack : 444 bytes

used heap : 0:0 0:0 (req:allocated,req:released) max=0 cur=0 (cfg=3)

observer res : 120 bytes used from the heap (6 c-nodes)

Inference time by c-node

kernel : 46.947ms (time passed in the c-kernel fcts)

user : 0.008ms (time passed in the user cb)

c\_id type id time (ms)

-------------------------------------------------

0 DENSE 0 27.129 57.79 %

1 NL 0 0.047 0.10 %

2 DENSE 2 19.463 41.46 %

3 NL 2 0.047 0.10 %

4 DENSE 4 0.245 0.52 %

5 NL 4 0.013 0.03 %

-------------------------------------------------

46.947 ms

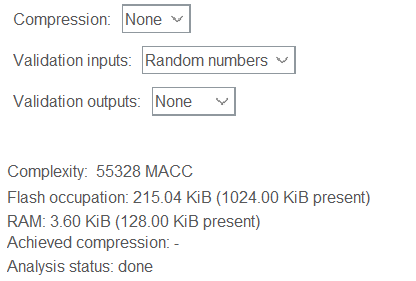
**2-2. 컴프레션 없이 뉴런 512에서 64로 감소**

Training 결과는 다음과 같다.

Test loss: 0.09571483579498599

Test accuracy: 0.9767

Compression을 하지 않고 analyze한 결과로, 215.04 KiB 이다.

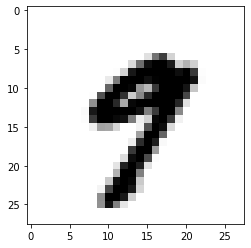


비교를 위해 이전과 같은 테스트 샘플로 test하였고 모든 사진에 대해서 맞는 숫자가 예측되었다. 결과는 아래와 같다. 추가적으로 3가지의 다른 샘플로 테스트해보았고, 모두 제대로 예측되었다.

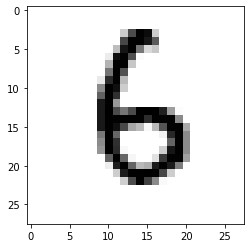
runfile('C:/work/Anaconda/send\_test.py', wdir='C:/work/Anaconda')

COM6 is open...

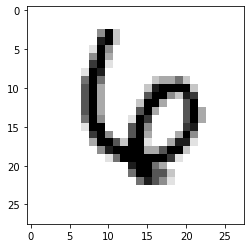
(10000, 28, 28)



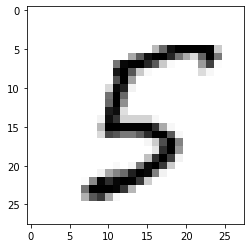
Recognized digit: 9.



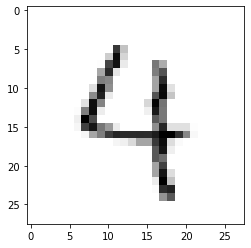
Recognized digit: 6.



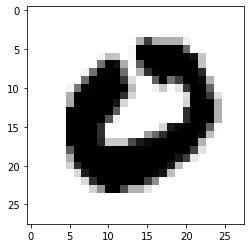
Recognized digit: 6.

￼

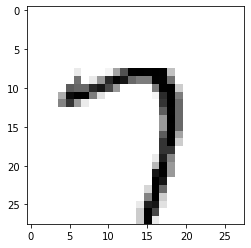
Recognized digit: 5.



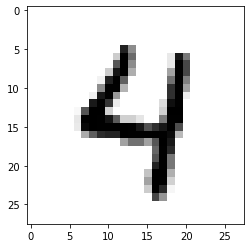
Recognized digit: 4.



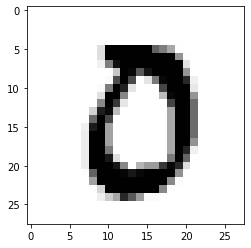
Recognized digit: 0.



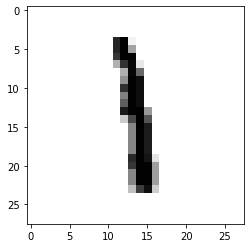
Recognized digit: 7.



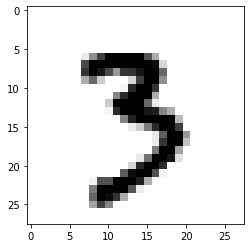
Recognized digit: 4.



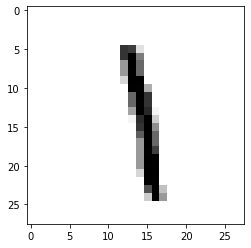
Recognized digit: 0.



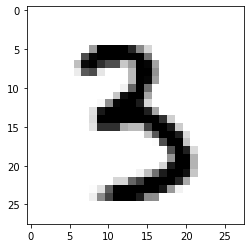
Recognized digit: 1.

￼

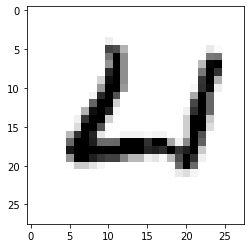
Recognized digit: 3.



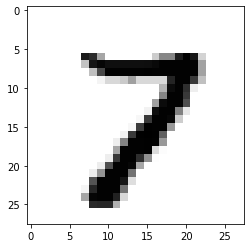
Recognized digit: 1.



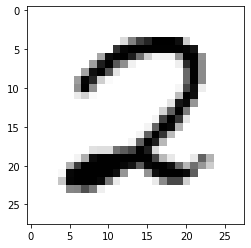
Recognized digit: 3.



Recognized digit: 4.



Recognized digit: 7.



Recognized digit: 2.

Results for "network", 16 inferences @168MHz/168MHz (complexity: 55328 MACC)

duration : 2.670 ms (average)

CPU cycles : 448687 -69/+78 (average,-/+)

CPU Workload : 0% (duty cycle = 1s)

cycles/MACC : 8.10 (average for all layers)

used stack : 412 bytes

used heap : 0:0 0:0 (req:allocated,req:released) max=0 cur=0 (cfg=3)

observer res : 120 bytes used from the heap (6 c-nodes)

Inference time by c-node

kernel : 2.645ms (time passed in the c-kernel fcts)

user : 0.008ms (time passed in the user cb)

c\_id type id time (ms)

-------------------------------------------------

0 DENSE 0 2.370 89.61 %

1 NL 0 0.007 0.29 %

2 DENSE 2 0.209 7.93 %

3 NL 2 0.007 0.29 %

4 DENSE 4 0.035 1.34 %

5 NL 4 0.014 0.55 %

-------------------------------------------------

2.645 ms

**Chapter 3. 결론 및 Discussion**

두 경우 모두 같은 테스트 샘플로 테스트했을 때 제대로 인식하였다. 뉴런이 64일 때 다른 샘플로도 테스트를 해보았는데 모두 제대로 숫자를 인식하였다. 100 퍼센트 정확함을 확인할 수는 없지만, 뉴런이 작음에도 인식이 잘 되는 것을 보니 이 과제에서는 뉴런을 512까지 쓸 필요는 없을 것 같다고 생각했다.

트레이닝 후 나온 결과를 표로 비교해보았다.

|  |  |  |
| --- | --- | --- |
|  | 컴프레션 O, 뉴런 512 | 컴프레션 X , 뉴런 64 |
| Test loss | 0.12700376453043655 | 0.09571483579498599 |
| Test accuracy | 0.9802 | 0.9767 |

뉴런이 512일 때 test loss 가 0.03정도 더 컸지만 test accuracy는 0.04 정도 더 컸다. 검색해본 결과 test loss가 크면 새로운 데이터에 대한 예측이 안 좋다고 한다. 뉴런이 많으면 test loss가 작고, test accuracy는 커야 할 것 같은데 오히려 test loss가 큰 것이 의아했다. 꼭 뉴런이 많을 필요가 없다는 것이 이런 이유 때문이지 않을까 생각했다.

메모리 사용량과 속도에 관련 있다고 생각한 부분을 표로 비교해 보았다.

|  |  |  |
| --- | --- | --- |
| Test 결과 성능 | 컴프레션 O, 뉴런 512 | 컴프레션 X, 뉴런 64 |
| duration | 46.973 ms (average) | 2.670 ms (average) |
| used stack | 444 bytes | 412 bytes |
| CPU Workload | 4% (duty cycle = 1s) | 0% (duty cycle = 1s) |

뉴런이 512로 많을 때 사용한 stack이 더 컸다. 그리고 cpu 작업량과 걸린 총 시간도 더 컸음을 확인할 수 있었다.

**부록**

프로그래밍를 한 것이 없습니다.

send\_test.py에서 포트 번호만 변경하였습니다.

# 0. 사용할 패키지 불러오기

from keras.datasets import mnist

import serial

port = "COM6"

baud = 115200

ser = serial.Serial(port, baud, timeout=1)

# open the serial port

if ser.isOpen():

print(ser.name + ' is open...')

# 1. 데이터셋 생성하기

# 훈련셋과 시험셋 불러오기

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

print(x\_test.shape)

for k in list(range(0,16)):

digit= x\_test[120+k]

import matplotlib.pyplot as plt

plt.imshow(digit, cmap=plt.cm.binary)

plt.show()

#print( digit)

for i in list(range(0,28)):

for j in list(range(0,28)):

digit\_string="{:3d}".format(digit[i][j])

ser.write(digit\_string.encode('ascii'))

#print(digit[i][j])

out = ser.read(2)

print('Recognized digit:',out.decode('utf-8'))

out = ser.read(2000)

print(out.decode('utf-8'))

ser.close()

#exit()