

# 임베디드 소프트웨어: 손글씨 인식 개인 과제

Park Jonghyeon  
컴퓨터정보통신공학과  
전남대학교 공과대학  
jonghyeon@jnu.ac.kr

*Abstract*—CNN 실습 과정 중의 MNIST 데이터 학습 실습 내용을 참고하여 손글씨 인식 모델을 새로 학습시켰습니다. 이어서 직접 작성한 0부터 9까지 10개 수를 직접 작성하여 모델의 성능 실험에 사용하였습니다.

## I. 모델 정의 및 학습

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	832
conv2d_1 (Conv2D)	(None, 28, 28, 32)	25,632
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18,496
conv2d_3 (Conv2D)	(None, 14, 14, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401,536
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

## II. 성능 실험

이렇게 학습시킨 모델의 성능을 확인하기 위해 직접 0부터 9까지 10개 수를 4회 작성하였습니다. 몇 개 수는 필체를 조금씩 변주하였으나, 대부분의 경우 동일한 필체입니다.

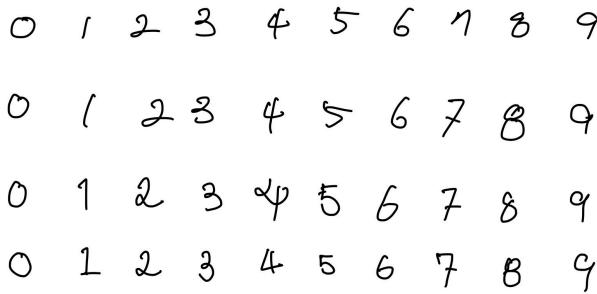


Fig. 1: 추정 테스트에 사용한 손글씨 데이터

### A. 전처리

이렇게 수기 작성한 40개 수는 이미지 편집 프로그램을 이용하여 40개의 이미지 파일로 분할하였습니다. 이후에 코드에서 추가 전처리를 수행할 수 있으므로, 이미지 파일의

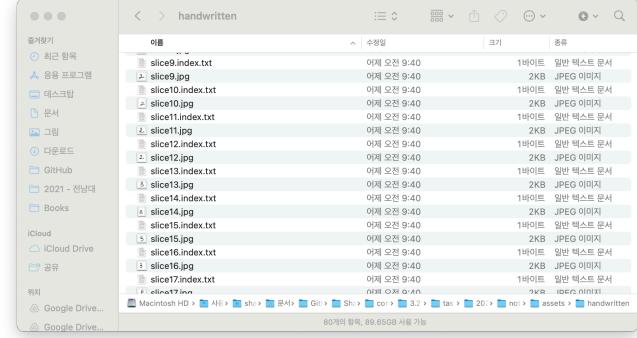


Fig. 2: 손글씨 이미지 분할 및 정답 데이터 생성

해상도와 종횡비를 고려하지 않았습니다. 각 파일의 해상도는 일정하지 않습니다.

```
for i in range(1, 41):
    with open(f'slice{i}.index.txt') as f:
        f.write(f'{(f - 1) // 4}')
```

Listing 1: 각 수에 대한 정답 데이터 생성

이와 같이 생성한 데이터는 성능 평가 과정에서 그레이스케일,  $28 \times 28$  픽셀 해상도로 변환하여 모델에 입력합니다. 이와 같이 변환한 것은 1채널  $28 \times 28$  벡터로 모델의 입력 층을 설정하였기 때문입니다.

## III. 성능 평가

성능 평가 결과는 Section IV.B 와 Section IV.C 를 참조해주시십시오.

최초의 성능 평가는 부적절한 전처리를 거쳐 제대로 추정하지 못했습니다. 모델 학습에 사용한 MNIST 데이터는 배경을 0, 획을 1로 처리하였습니다. 이에 반해 성능 부적절한 전처리를 거쳐 성능 평가를 수행할 때는 배경을 1, 획을 0으로 처리하였기 때문에 모든 시도에서 결과를 제대로 추정하지 못했습니다.

이 전처리 문제를 해결한 뒤에는 시도 #17 #22 #27 등과 같이 5개 시도를 제외하고는 정확한 결과를 도출하여 87.5%의 정확도를 보였습니다.

#### IV. 부록

##### A. 손글씨 데이터



Fig. 3: 시도 #2에 사용한 손글씨 데이터



Fig. 4: 시도 #3에 사용한 손글씨 데이터



Fig. 5: 시도 #4에 사용한 손글씨 데이터



Fig. 6: 시도 #5에 사용한 손글씨 데이터



Fig. 7: 시도 #6에 사용한 손글씨 데이터



Fig. 8: 시도 #7에 사용한 손글씨 데이터



Fig. 9: 시도 #8에 사용한 손글씨 데이터



Fig. 10: 시도 #9에 사용한 손글씨 데이터



Fig. 11: 시도 #10에 사용한 손글씨 데이터



Fig. 12: 시도 #11에 사용한 손글씨 데이터



Fig. 13: 시도 #12에 사용한 손글씨 데이터



Fig. 14: 시도 #13에 사용한 손글씨 데이터



Fig. 15: 시도 #14에 사용한 손글씨 데이터



Fig. 16: 시도 #15에 사용한 손글씨 데이터

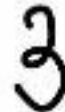


Fig. 17: 시도 #16에 사용한 손글씨 데이터



Fig. 18: 시도 #17에 사용한 손글씨 데이터



Fig. 19: 시도 #18에 사용한 손글씨 데이터



Fig. 20: 시도 #19에 사용한 손글씨 데이터



Fig. 21: 시도 #20에 사용한 손글씨 데이터



Fig. 22: 시도 #21에 사용한 손글씨 데이터



Fig. 23: 시도 #22에 사용한 손글씨 데이터



Fig. 24: 시도 #23에 사용한 손글씨 데이터



Fig. 25: 시도 #24에 사용한 손글씨 데이터



Fig. 26: 시도 #25에 사용한 손글씨 데이터



Fig. 27: 시도 #26에 사용한 손글씨 데이터



Fig. 28: 시도 #27에 사용한 손글씨 데이터

## B. 적절한 전처리를 거친 데이터의 평가 결과

6

Fig. 29: 시도 #28에 사용한 손글씨 데이터

7

Fig. 30: 시도 #29에 사용한 손글씨 데이터

7

Fig. 31: 시도 #30에 사용한 손글씨 데이터

7

Fig. 32: 시도 #31에 사용한 손글씨 데이터

7

Fig. 33: 시도 #32에 사용한 손글씨 데이터

8

Fig. 34: 시도 #33에 사용한 손글씨 데이터

8

Fig. 35: 시도 #34에 사용한 손글씨 데이터

8

Fig. 36: 시도 #35에 사용한 손글씨 데이터

8

Fig. 37: 시도 #36에 사용한 손글씨 데이터

9

Fig. 38: 시도 #37에 사용한 손글씨 데이터

9

Fig. 39: 시도 #38에 사용한 손글씨 데이터

9

Fig. 40: 시도 #39에 사용한 손글씨 데이터

9

Fig. 41: 시도 #40에 사용한 손글씨 데이터

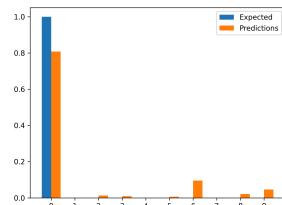


Fig. 42: 시도 #2의 추정 결과

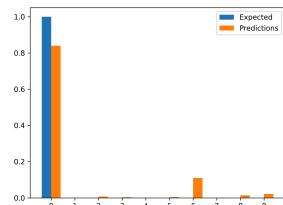


Fig. 43: 시도 #3의 추정 결과

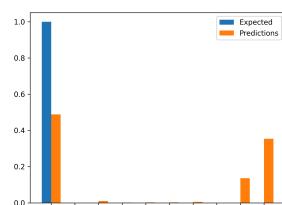


Fig. 44: 시도 #4의 추정 결과

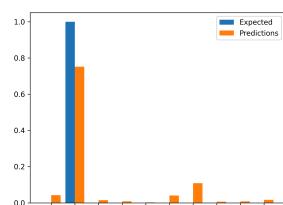


Fig. 45: 시도 #5의 추정 결과

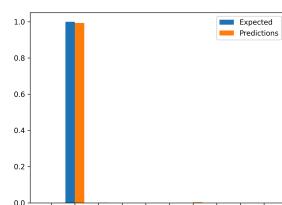


Fig. 46: 시도 #6의 추정 결과

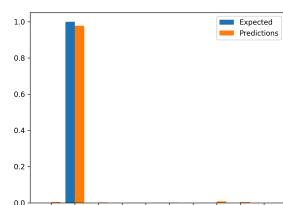


Fig. 47: 시도 #7의 추정 결과

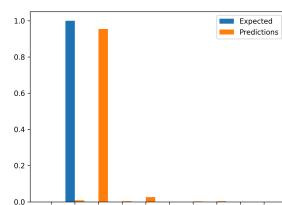


Fig. 48: 시도 #8의 추정 결과

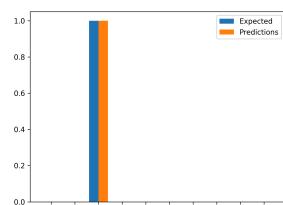


Fig. 49: 시도 #9의 추정 결과

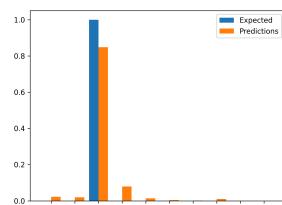


Fig. 50: 시도 #10의 추정 결과

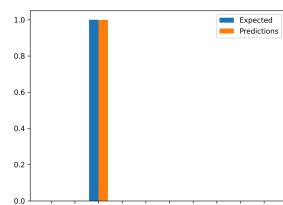


Fig. 51: 시도 #11의 추정 결과



Fig. 52: 시도 #12의 추정 결과



Fig. 53: 시도 #13의 추정 결과



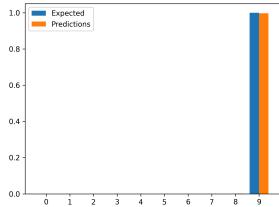


Fig. 78: 시도 #38의 추정 결과

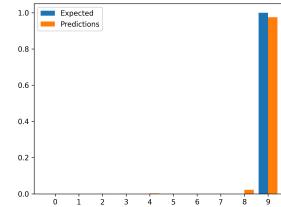


Fig. 79: 시도 #39의 추정 결과

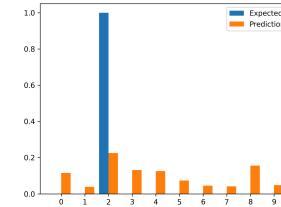


Fig. 89: 시도 #10의 추정 결과

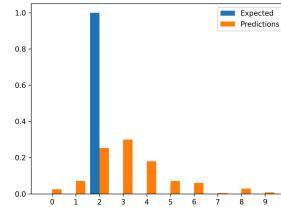


Fig. 90: 시도 #11의 추정 결과

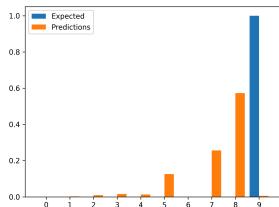


Fig. 80: 시도 #40의 추정 결과

### C. 부적절한 전처리를 거친 데이터의 평가 결과

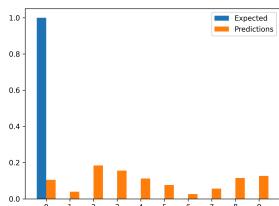


Fig. 81: 시도 #2의 추정 결과

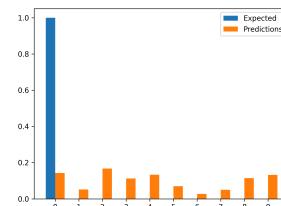


Fig. 82: 시도 #3의 추정 결과

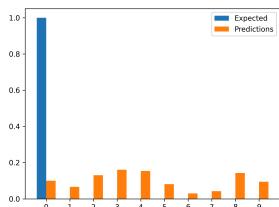


Fig. 83: 시도 #4의 추정 결과

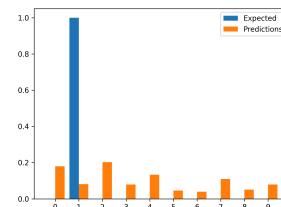


Fig. 84: 시도 #5의 추정 결과

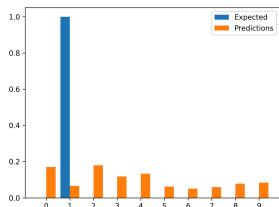


Fig. 85: 시도 #6의 추정 결과

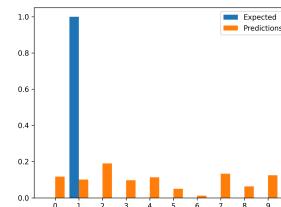


Fig. 86: 시도 #7의 추정 결과

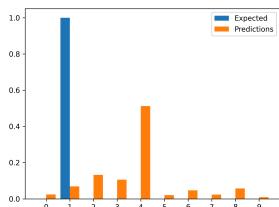


Fig. 87: 시도 #8의 추정 결과

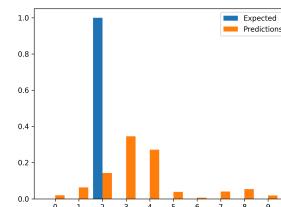


Fig. 88: 시도 #9의 추정 결과

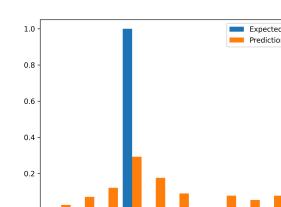


Fig. 91: 시도 #12의 추정 결과

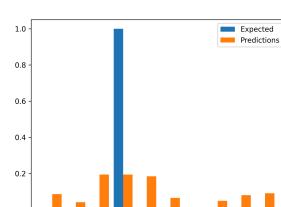


Fig. 92: 시도 #13의 추정 결과

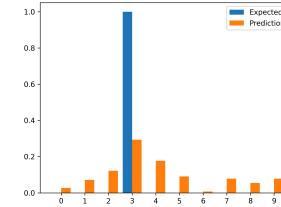


Fig. 93: 시도 #14의 추정 결과

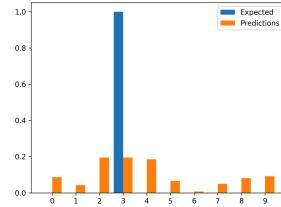


Fig. 94: 시도 #15의 추정 결과

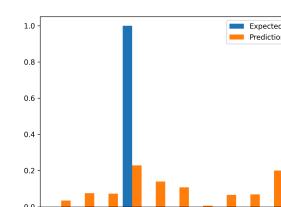


Fig. 95: 시도 #16의 추정 결과

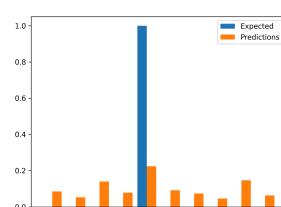


Fig. 96: 시도 #17의 추정 결과



Fig. 97: 시도 #18의 추정 결과



Fig. 98: 시도 #19의 추정 결과



Fig. 99: 시도 #20의 추정 결과



Fig. 100: 시도 #21의 추정 결과



#### D. 소스코드

```
a) use.py:
import tensorflow as tf
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

model = tf.keras.models.load_model('model.keras')

def load_and_predict(image_path):
    img = Image.open(image_path).convert('L').resize((28, 28))

    img_array = np.array(img)
    img_array = img_array.astype('float32') / 255.0
    img_array = 1 - img_array

    img_array = np.expand_dims(img_array, axis=0)

    predictions = model.predict(img_array)

    return predictions

import pathlib

def main():
    basepath = pathlib.Path('assets/handwritten')
    for i in range(1, 41):
        image_path = basepath / f'slice{i}.jpg'
        index_path = basepath / f'slice{i}.index.txt'
        result = load_and_predict(image_path)
        print(f'Image {i}:')
        index = -1
        with open(index_path, 'r') as f:
            index = int(f.read())
            print(f'Expected: {index}')
        print(f'Predictions: {tf.math.argmax(result, axis=1)[0]}')
        print(result)
        print(tf.nn.softmax(result))

        x_axis = np.arange(10)
        plt.xticks(range(10), range(10))
        plt.bar(x_axis - .2, [0 if i != index else 1 for i in range(10)], .4, label='Expected')
        plt.bar(x_axis + .2, result[0], .4, label='Predictions')
        plt.legend()
        plt.savefig(f'assets/out/preprocessed/result{i}.png', dpi=300)
        plt.clf()

if __name__ == '__main__':
    main()
```