머신러닝, 딥러닝 과제

C반 4조 이영은

1.진행상황

1시도.

```
]:
def CNN_BN(n_in, n_out):
    # Coding Time
    # Feature Extraction
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3, 3), padding='same', input_shape=n_in))
    model.add(BatchNormalization())
    model.add(ReLU())

    model.add(Conv2D(32, (3, 3), padding='same',strides=(2, 2)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())

    # Classfier
    model.add(Flatten())
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(Dense(n_out, activation='softmax'))
    return model
```

이걸로 하고 epochs = 40으로 하고 돌림
learning_rate = 0.001

98.6% 정확도로 나옴

2시도.
같은 상황에서 epochs = 50으로 하고 돌림
99.01% 정확도로 나옴
learning_rate = 0.0001로 수정

3시도.

모델 수정

epochs = 50으로 수정

정확도 99.21로 나옴

```
def CNN_BN(n_in, n_out):
    # Coding Time
    # Feature Extraction
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=n_in))
    model.add(BatchNormalization())
    model.add(ReLU())

    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(ReLU())

    model.add(Conv2D(32, (5, 5), padding='same',strides=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(Dropout(0.4))

    model.add(Conv2D(64, (3, 3), padding='same'))
#     model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())

    model.add(Conv2D(64, (3, 3), padding='same'))
#     model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())

    model.add(Conv2D(64, (3, 3), padding='same',strides=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())

#     model.add(Conv2D(64, (3, 3), padding='same',strides=(2, 2)))
#     model.add(BatchNormalization())
#     model.add(ReLU())

    model.add(Conv2D(128, (4, 4), padding='same'))
    model.add(BatchNormalization())
    model.add(ReLU())

    # Classifier
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(128))
#     model.add(BatchNormalization())
#     model.add(ReLU())
    model.add(Dense(n_out, activation='softmax'))
    return model
```

4시도.

모델을 수정하고 epochs = 100으로 바꿈
정확도가 99.5% 나옴

5시도.
모델학습하기만 다시하고 돌렸더니 정확도 99.49%로 나옴

6시도.
모델학습하기만 다시하고 돌렸더니 정확도 99.46%로 떨어짐

최종
99.46%

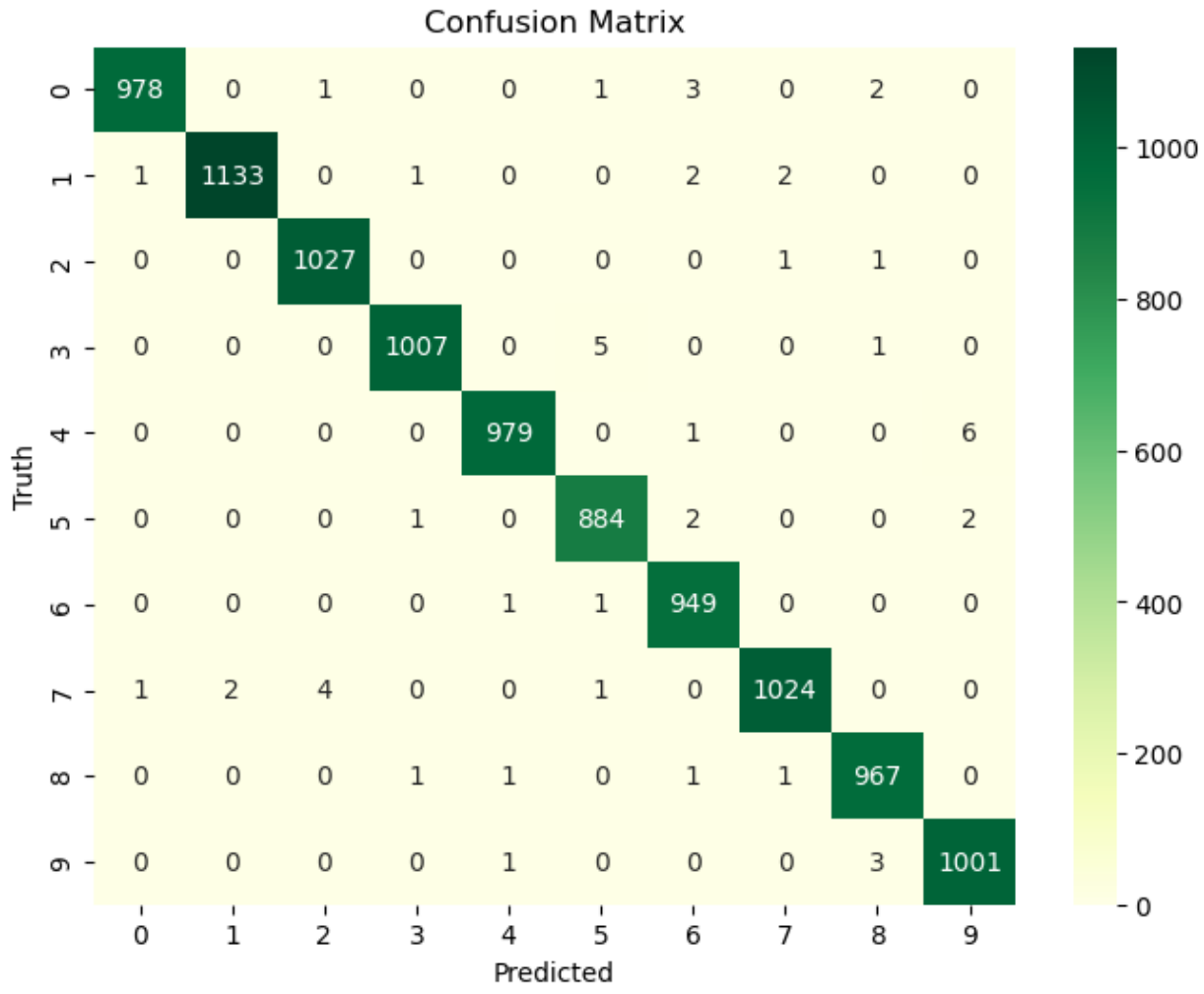
2.학습분석

```
Epoch 81/100
375/375 [==============================] - 4s 10ms/step - loss: 6.5713e-04 - accuracy: 0.9998 - val_loss: 0.0554 - val_accuracy: 0.9939
Epoch 82/100
375/375 [==============================] - 4s 10ms/step - loss: 4.2739e-04 - accuracy: 0.9998 - val_loss: 0.0502 - val_accuracy: 0.9946
Epoch 83/100
375/375 [==============================] - 4s 10ms/step - loss: 6.9556e-04 - accuracy: 0.9998 - val_loss: 0.0578 - val_accuracy: 0.9938
Epoch 84/100
375/375 [==============================] - 4s 10ms/step - loss: 7.8870e-04 - accuracy: 0.9997 - val_loss: 0.0541 - val_accuracy: 0.9937
Epoch 85/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0013 - accuracy: 0.9995 - val_loss: 0.0588 - val_accuracy: 0.9935
Epoch 86/100
375/375 [==============================] - 4s 10ms/step - loss: 4.3633e-04 - accuracy: 0.9999 - val_loss: 0.0636 - val_accuracy: 0.9933
Epoch 87/100
375/375 [==============================] - 4s 10ms/step - loss: 8.5503e-04 - accuracy: 0.9997 - val_loss: 0.0516 - val_accuracy: 0.9937
Epoch 88/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0011 - accuracy: 0.9996 - val_loss: 0.0578 - val_accuracy: 0.9933
Epoch 89/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.0592 - val_accuracy: 0.9929
Epoch 90/100
375/375 [==============================] - 4s 10ms/step - loss: 9.6543e-04 - accuracy: 0.9997 - val_loss: 0.0562 - val_accuracy: 0.9936
Epoch 91/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0010 - accuracy: 0.9997 - val_loss: 0.0560 - val_accuracy: 0.9934
Epoch 92/100
375/375 [==============================] - 4s 10ms/step - loss: 8.4171e-04 - accuracy: 0.9997 - val_loss: 0.0565 - val_accuracy: 0.9942
Epoch 93/100
375/375 [==============================] - 4s 10ms/step - loss: 4.9429e-04 - accuracy: 0.9998 - val_loss: 0.0520 - val_accuracy: 0.9944
Epoch 94/100
375/375 [==============================] - 4s 10ms/step - loss: 5.7889e-04 - accuracy: 0.9998 - val_loss: 0.0576 - val_accuracy: 0.9940
Epoch 95/100
375/375 [==============================] - 4s 10ms/step - loss: 7.2810e-04 - accuracy: 0.9997 - val_loss: 0.0570 - val_accuracy: 0.9942
Epoch 96/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 0.0564 - val_accuracy: 0.9943
Epoch 97/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0013 - accuracy: 0.9998 - val_loss: 0.0559 - val_accuracy: 0.9938
Epoch 98/100
375/375 [==============================] - 4s 10ms/step - loss: 6.7844e-04 - accuracy: 0.9998 - val_loss: 0.0526 - val_accuracy: 0.9943
Epoch 99/100
375/375 [==============================] - 4s 10ms/step - loss: 6.7138e-04 - accuracy: 0.9997 - val_loss: 0.0642 - val_accuracy: 0.9937
Epoch 100/100
375/375 [==============================] - 4s 10ms/step - loss: 0.0014 - accuracy: 0.9995 - val_loss: 0.0549 - val_accuracy: 0.9941
```

accuracy가 증가하는데, val_accuracy 가 떨어지면 과대 적합이다
따라서 accuracy가 일정 수에서 수렴하고 val_accuracy도 일정 수에서 수렴하므로 적정한 모델이라고 생각했다

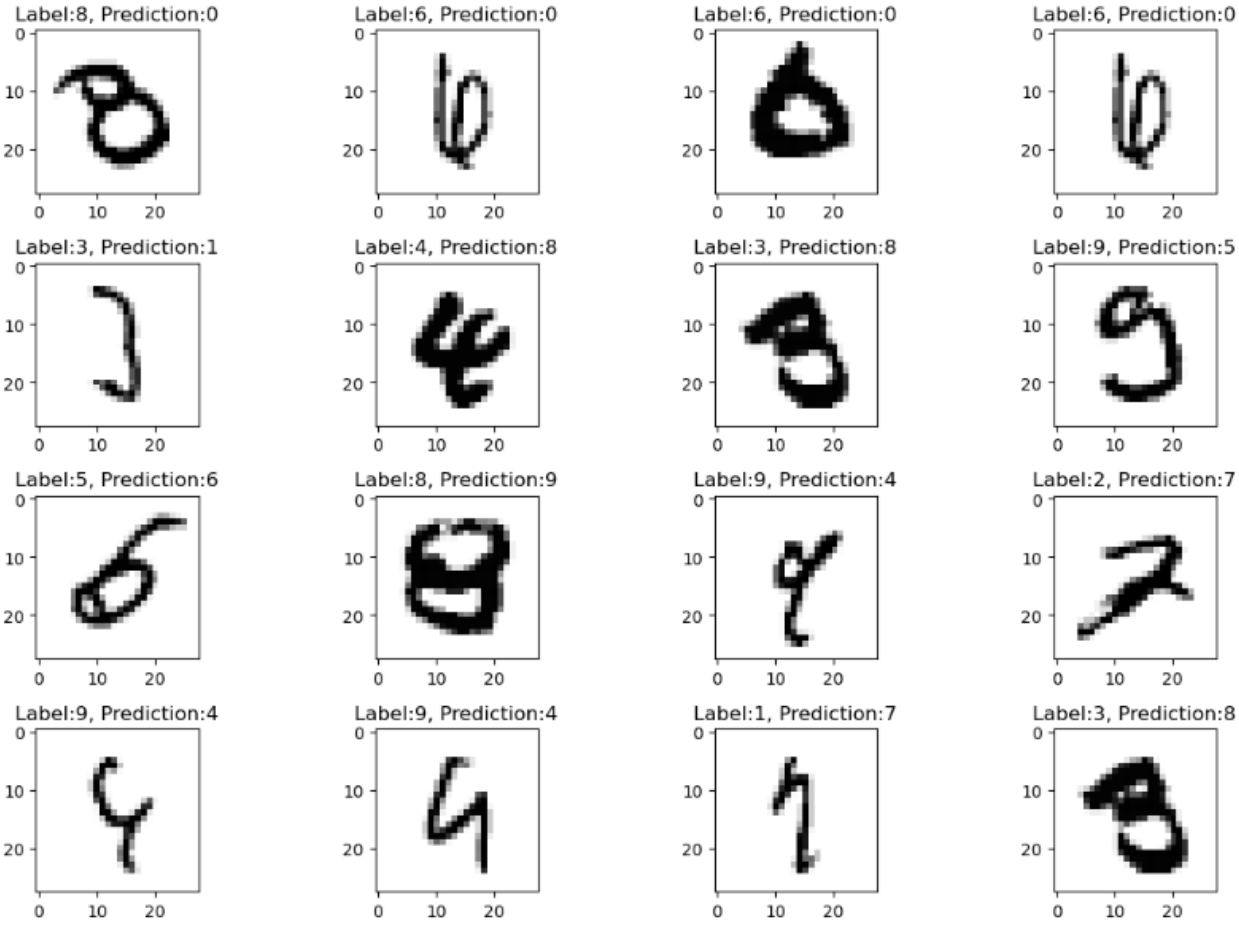
3.성능분석

```
# Coding Time
#모델의 예상값 리스트 생성
pred_y = model.predict(X_test, verbose=0)
Y_pred = [x.argmax() for x in pred_y]

# 정답 리스트 생성(인덱스)
Y_test_idx = [x.argmax() for x in Y_test]

#matrix 생성
data = {'Real' : Y_test_idx, 'Predict' : Y_pred}
df = pd.DataFrame(data, columns=['Real','Predict'])

#heatmap 생성

our_cmatrix = confusion_matrix(Y_pred, Y_test_idx)
plt.figure(figsize=(8, 6), dpi=99)
sns.heatmap(our_cmatrix, annot=True, fmt='g', cmap='YlGn').set(xlabel='Predicted'
                                                               , ylabel='Truth')
plt.title('Confusion Matrix')
plt.show()


# conf_mat = pd.crosstab(df['Real'], df['Predict'], rownames=['Real'], colnames=['Predict'])
# print(conf_mat)
```



1이 가장 분류하기 쉽고 5가 가장 분류하기 어려운 것을 알 수 있다

*랜덤하게 잘 못 분류된 값들을 뽑았다.



여기서 대부분 train data가 잘못된 것을 알 수 있다.