2024 Fall
20190741 김지수

인공지능과 마케팅 코딩과제2

:Mnist data

MNIST Data

손글씨로 작성된 숫자(0~9) 데이터 셋으로 분류 목적으로 자주 사용되는 이미지 데이터셋

- 이미지 크기: 28*28 픽셀의 흑백 이미지
- 데이터 분포: 총 70,000장의 이미지 (학습용으로 60,000장과 테스트용으로 10,000장)
- 분류: 0~9까지 총 10개의 숫자 클래스로 구성됨
- 이미지 분류 문제를 다루는 기초적인 데이터셋으로 CNN 성능 확인에 많이 사용됨

Import library

```
# LOAD LIBRARIES
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
# USE KERAS WITH DEFAULT TENSORFLOW BACKEND\
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,
BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from keras.datasets import mnist
```

Load Dataset & Split Dataset

```
# LOAD MNIST DATASET AS 60K TRAIN AND 10K TEST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

데이터 전처리

```
# PREPARE DATA FOR NEURAL NETWORK
X_train = x_train / 255.0
X_test = x_test / 255.0
X_train = X_train.reshape(-1,28,28,1)
X_test = X_test.reshape(-1,28,28,1)
Y_train = to_categorical(y_train, num_classes = 10)
```

1. **입력 이미지 데이터를 255.0으로 나누어 정규화 작업을 수행한다.**
2. **X_train과 X_test 데이터를 CNN이 처리할 수 있는 형식으로 변환한다.**

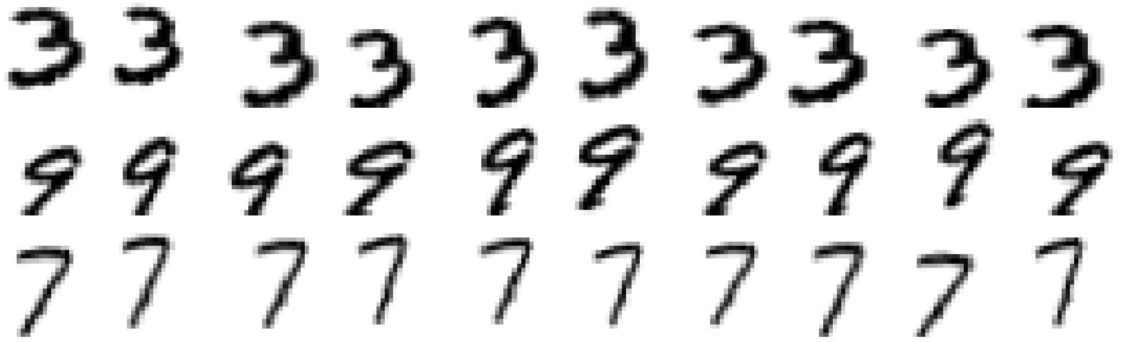
: MNIST의 각 이미지가 28*28의 크기이며, 1은 흑백을 나타내고(채널이 1개), -1은 샘플 개수를 자동으로 맞추라는 설정이다.

3. **분류 결과인 Y에 대해서는 범주형 데이터로 변환한다.**

: MNIST 데이터는 0부터 9까지 10개의 클래스가 있으므로 num_classes를 10으로 설정하여 CNN이 분류 문제를 다룰 수 있도록 y_train을 원-핫 인코딩을 사용하여 변환한다.

Data Augmentation (데이터 증강)

```
# CREATE MORE IMAGES WITH DATA AUGMENTATION
datagen = ImageDataGenerator(
        rotation_range=15,
        zoom_range = 0.15,
        width_shift__range=0.1,
        height_shift_range=0.1)
```



1. 더 많은 데이터를 생성하여 CNN의 학습 성능을 높인다.

: 원본 데이터셋을 확대(zoom), 회전(rotate), 이동(shift)하는 등의 방법으로 변형하여 새로운 이미지를 만들고 이를 학습 데이터로 활용하여 모델이 더 다양한 패턴을 학습해 과적합(overfitting)을 방지한다.

위의 코드에서는 이미지를 최대 +- 15도까지 회전시키고, 최대 15%까지 확대하거나 축소하고, 가로 방향으로 최대 10%까지 이동시키며, 세로 방향으로 최대 10%까지 이동시킨다. 이는 입력되는 이미지가 중심에서 벗어나거나 때로는 크게 혹은 작게, 조금 회전되어서 나타날 수 있음을 고려한 것이다.

Train the model

모델 생성

```
# BUILD CNN
nets = 7
model = [0] *nets
for j in range(nets):
    model[j] = Sequential()

    model[j].add(Conv2D(32, kernel_size = 3, activation='relu', input_shape = (28, 28, 1)))
    model[j].add(BatchNormalization())
    model[j].add(Conv2D(32, kernel_size = 3, activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Conv2D(32, kernel_size = 5, strides=2, padding='same', activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Dropout(0.4))

    model[j].add(Conv2D(64, kernel_size = 3, activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Conv2D(64, kernel_size = 3, activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Conv2D(64, kernel_size = 5, strides=2, padding='same', activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Dropout(0.4))

    model[j].add(Conv2D(128, kernel_size = 4, activation='relu'))
    model[j].add(BatchNormalization())
    model[j].add(Flatten())
    model[j].add(Dropout(0.4))
    model[j].add(Dense(10, activation='softmax'))

    # COMPILE WITH ADAM OPTIMIZER AND CROSS ENTROPY COST
    model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Train the model

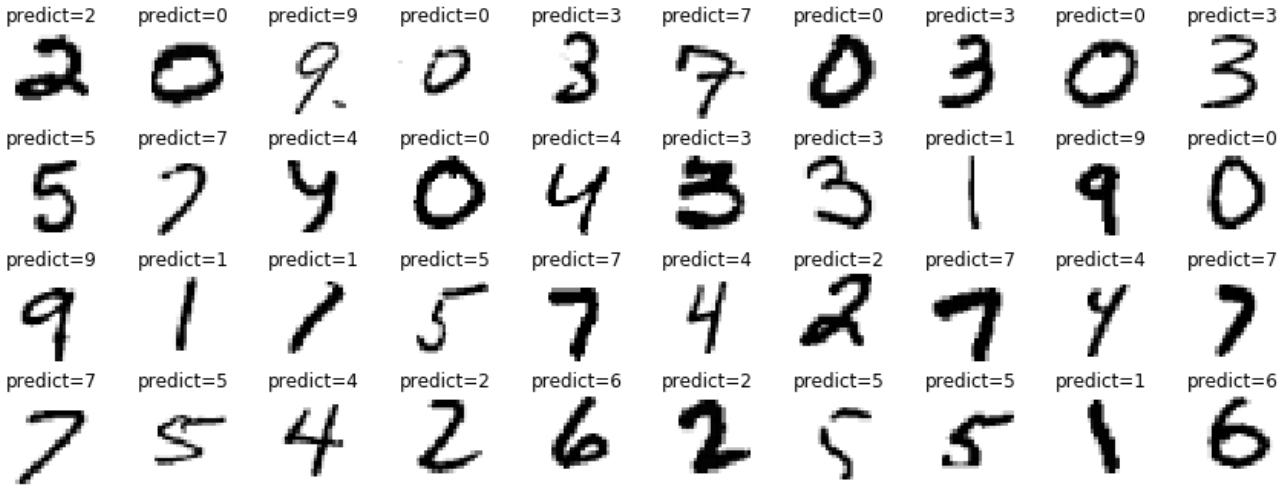
모델 학습

```
# Learning Rate Scheduler: 매 epoch마다 learning rate를 0.95배로 감소
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

# TRAIN CNNs AND DISPLAY ACCURACIES
epochs = 40
history = [0] * nets
results = [0] * nets

for j in range(nets):
    # 데이터를 훈련 세트와 검증 세트로 분할
    X_train2, X_val2, Y_train2, Y_val2 = train_test_split(X_train, Y_train, test_size=0.1)

    # CNN 훈련 및 진행 상태 출력
    history[j] = model[j].fit(datagen.flow(X_train2, Y_train2, batch_size=64),epochs=epochs,steps_per_epoch=X_train2.shape[0] // 64,
                              validation_data=(X_val2, Y_val2),callbacks=[annealer],verbose=1)  # 진행 상황을 출력하도록 verbose=1 설정
    # 훈련 및 검증 정확도 출력
    print("CNN {0:d}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}".format(
        j + 1, epochs, history[j].history['accuracy'][epochs - 1],  # 'acc' 대신 'accuracy'로 수정
        history[j].history['val_accuracy'][epochs - 1]  # 'val_acc' 대신 'val_accuracy'로 수정
    ))
    # PREDICT DIGITS FOR CNN J ON MNIST 10K TEST
    results[j] = model[j].predict(X_test)
    results2 = np.argmax(results[j], axis=1)

    # CNN J의 MNIST 10K 테스트 정확도 계산
    c = 0
    for i in range(10000):
        if results2[i] != y_test[i]:
            c += 1
    print("CNN %d: Test accuracy = %f" % (j + 1, 1 - c / 10000.))
```

Evaluate the Dataset

앙상블 및 모델 평가

```
# PREDICT DIGITS FOR ENSEMBLE ON MNIST 10K TEST
results2 = np.zeros( (X_test.shape[0],10) )
for j in range(nets):
    results2 = results2 + results[j]
results2 = np.argmax(results2,axis = 1)

# CALCULATE ACCURACY
c=0
for i in range(10000):
    if results2[i]!=y_test[i]:
        c +=1
print("Ensemble Accuracy = %f" % (1-c/10000.))
```



Ensemble Accuracy = 0.997200

앙상블 및 모델 평가

CNN 1: Epochs=40, Train accuracy=1.00000, Validation accuracy=0.99550

CNN 1: Test accuracy = 0.995600

CNN 2: Epochs=40, Train accuracy=1.00000, Validation accuracy=0.99683

CNN 2: Test accuracy = 0.995700

CNN 3: Epochs=40, Train accuracy=0.98438, Validation accuracy=0.99633

CNN 3: Test accuracy = 0.996100

CNN 4: Epochs=40, Train accuracy=1.00000, Validation accuracy=0.99633

CNN 4: Test accuracy = 0.996900

CNN 5: Epochs=40, Train accuracy=0.98438, Validation accuracy=0.99383

CNN 5: Test accuracy = 0.996200

CNN 6: Epochs=40, Train accuracy=1.00000, Validation accuracy=0.99617

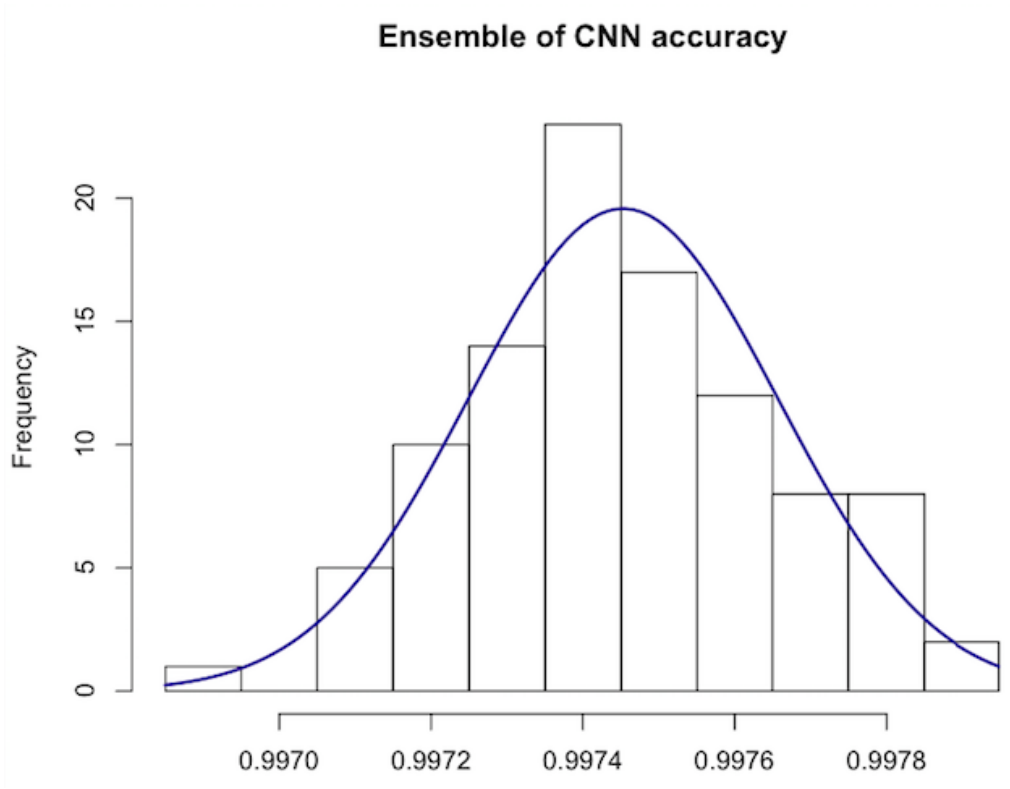
CNN 6: Test accuracy = 0.996300

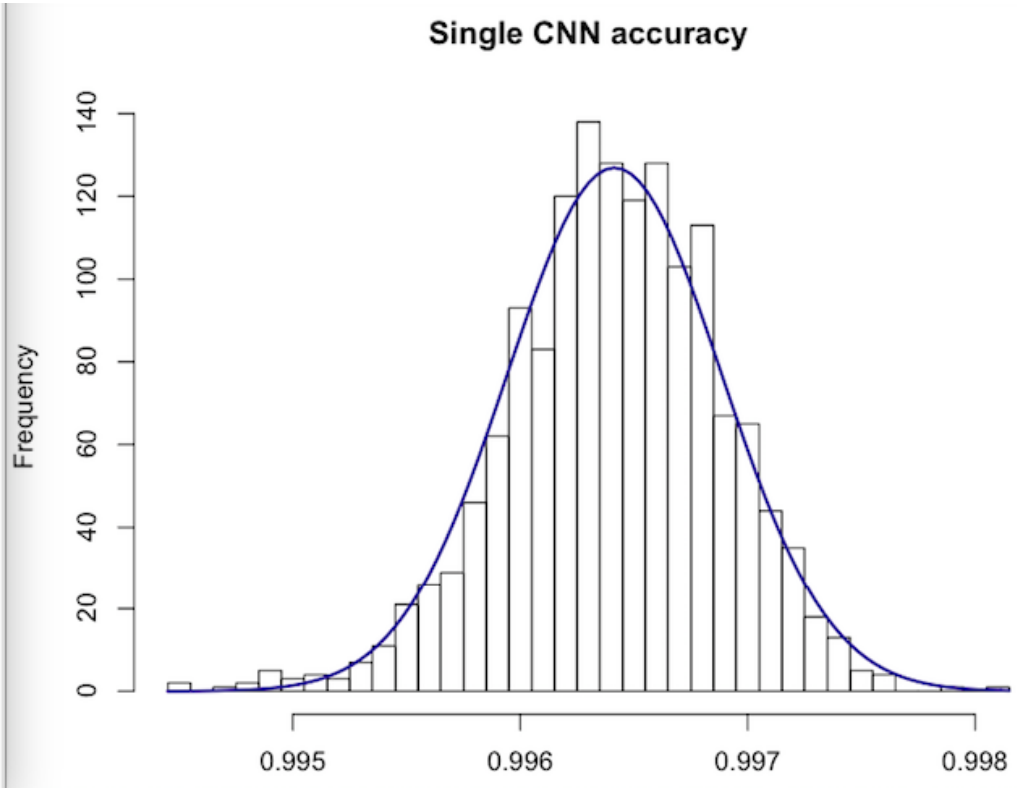
CNN 7: Epochs=40, Train accuracy=0.98438, Validation accuracy=0.99650

CNN 7: Test accuracy = 0.996000

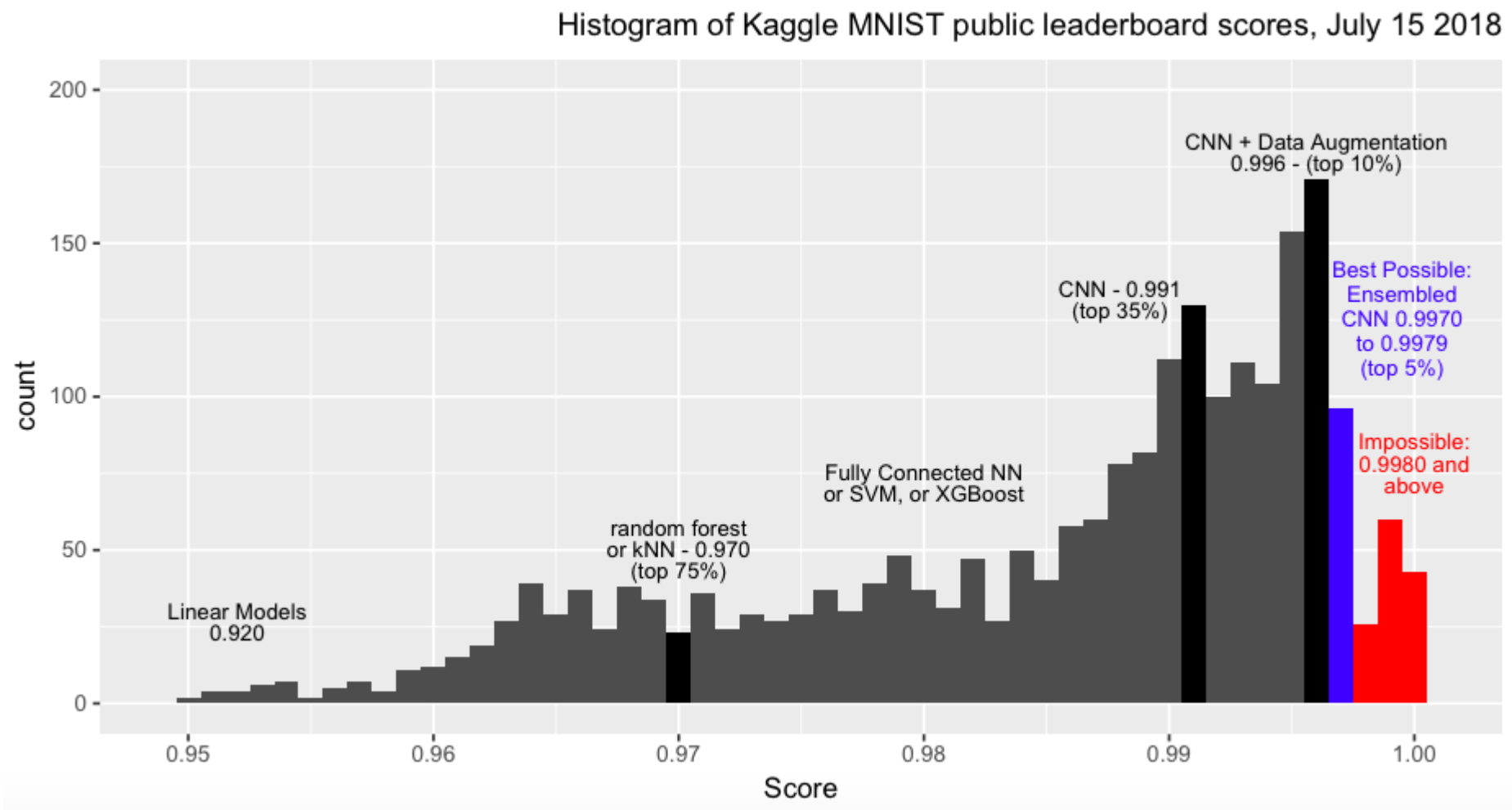
Ensemble Accuracy = 0.997200

Evaluate the Dataset



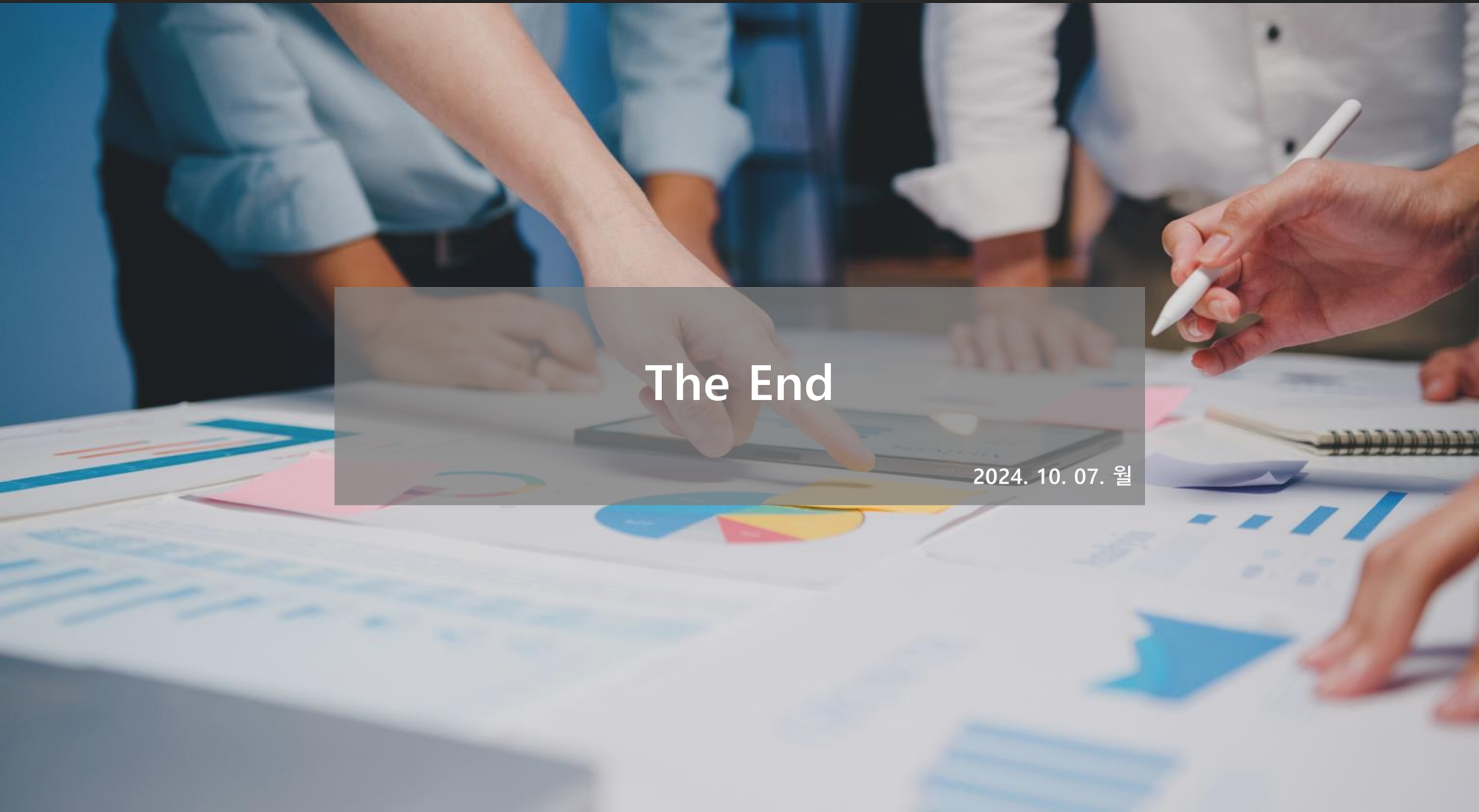


Evaluate the Dataset



Conclusion

1. 7만개의 데이터를 사용하는데도 과적합의 방지를 위해 추가적인 데이터를 생성하는 것을 알게되었다.
2. 기존 코드에서는 CNN 모델 15개를 생성하여 앙상블하였는데, 해당 코드에서는 모델 7개만을 만들고 이를 앙상블하였는데도 각 개별 CNN 모델의 정확도보다 높은 것을 알 수 있었다.
3. 모델을 여러 개를 학습시키고 epoch를 늘릴 수록 학습에 필요한 시간이 오래 걸리기 때문에 목표하는 성능 수준을 고려하여 모델을 설계하는 것이 좋겠다고 생각했다.



The End

2024. 10. 07. 월