# ===== 9/14 딥러닝 모델 만들기 예시 =====

# 인풋 레이어를 만듭니다

B0 = layers.Input(feature 갯수 입력)

# 학습시킬 Dense layer를 만듭니다. (자유는 구조)

# 32 개의 뉴런들이 다른 가중치를 가지고 선형 결합됩니다.

B1 = layers.Dense(32, activation="relu")(B0)

# 회귀 or 분류 레이어를 만듭니다.

layer\_regressor = layers.Dense(1)(B1)

# input layer + output layer 를 인자로 넣어 모델 생성.

model\_mlp = keras.Model(B0, layer\_regressor)

# 생성한 모델을 Compile 합니다.

model\_mlp.compile(loss=”mse”, optimizer=”adam”)

# ===== 공통 부분 =====

batch\_size 가 8 이고

전체 데이터 행이 1000개 이면

1000 / 8 = 125

1 epochs 에 돌아가는 횟수 iteration 은 약 125 번이다

# ===== 디폴트 옵션 (각자 조정) =====

tf.random.set\_seed(54321)

epochs = 200

batch\_size = 8

patientRate = 0.2

# callback 을 정의합니다

# 1. early stopping 콜백 : boosting 에서 쓰는 것과 같은 역할

# epochs 가 지정한 횟수 이상 성능개선이 없으면 학습 정지

cb\_earlystopping = tf\_callbacks.EarlyStopping(patience=int(np.floor(epochs \* patientRate)), restore\_best\_weights=True)

# 2. ReudeLROnPlateau 콜백 : learning rate 을 자동적으로 낮추도록 합니다

# epochs 가 지정한 횟수 이상 성능개선이 없으면 learning\_rate \* factor(내가 지정) 만큼 learning rate 을 감소시킵니다. (factor는 그래서 0~1 사이 값)

cb\_reduceLR = tf\_callbacks.ReduceLROnPlateau(patience=int(np.floor(epochs \* (patientRate \*\* 2))), factor=0.8, min\_lr=1e-5)

# 3. 위 두개의 콜백과 TqdmCallBack 을 합쳐 전체 callback 리스트를 만듭니다.

# TqdmCallback : 진행상황 표시줄?을 간단하게 만들어줌 모델이나 성능에 영향없음

cb\_lists = [cb\_earlystopping, cb\_reduceLR, TqdmCallback(verbose=0)]

model\_mlp.fit(train\_x\_oh, train\_y,

validation\_data=(val\_x\_oh, val\_y),

verbose=0, shuffle=False,

epochs = epochs,

batch\_size = batch\_size,

callbacks = cb\_lists)

# 주의! ReduceLROnPlateau 을 사용하려면

# complie 할 때 옵티마이저를 ‘adam’ 이라고 쓰는 것이 아니라 직접

# adam 옵티마이저 객체를 입력해주어야 합니다. (실행엔 문제 없으나 warnning을 안뜨게함)

# 다음과 같이 직접 ‘adam’말고 객체로 지정

# 또한 RMSE 도 확인하기위해 metrics 옵션도 지정

from tensorflow.keras import optimizers

from tensorflow.keras import metrics

ex. model\_mlp.compile(loss=”mse”, optimizer=optimizers.Adam(learning\_rate=0.003),

metrics=metrics.RootMeanSquaredError(name="rmse"))

# ================ §℗ⓔ☆₶ⓔ®−⁂ (DenseNet 구조) ====================

# MLP\_YJ

def createMLP\_DenseNet():

hiddenLayers = 512

dropoutRate = 0.2

B0\_input = layers.Input(shape=train\_x\_oh.shape[1], name="B0\_input")

B0\_embedding = layers.Dense(hiddenLayers \* 2, activation="relu",

kernel\_regularizer="l2", name="B0\_embedding")(B0\_input)

B1\_dense = layers.Dense(hiddenLayers, activation="relu", name="B1\_dense")(B0\_embedding)

B1\_dropout = layers.Dropout(rate=dropoutRate, name="B1\_dropout")(B1\_dense)

B1\_concat1 = layers.concatenate([B0\_embedding, B1\_dropout], name="B1\_concat1")

B2\_dense = layers.Dense(hiddenLayers, activation="relu", name="B2\_dense")(B1\_concat1)

B2\_dropout = layers.Dropout(dropoutRate, name="B2\_dropout")(B2\_dense)

B2\_concat2 = layers.concatenate([B0\_embedding, B1\_dropout, B2\_dropout], name="B2\_concat2")

B3\_dense = layers.Dense(hiddenLayers, activation="relu", name="B3\_dense")(B2\_concat2)

B3\_dropout = layers.Dropout(dropoutRate, name="B3\_dropout")(B3\_dense)

layer\_final = layers.Dense(int(hiddenLayers/2), activation="relu", name="layer\_final")(B3\_dropout)

layer\_regressor = layers.Dense(1, name="Regressor")(layer\_final)

model\_mlp = Model(B0\_input, layer\_regressor)

model\_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),

metrics=tf\_metrics.RootMeanSquaredError(name="rmse"))

# model\_mlp.summary()

return model\_mlp

tf.random.set\_seed(54321)

epochs = 200

batch\_size = 8

patientRate = 0.2

cb\_earlystopping = tf\_callbacks.EarlyStopping(patience=int(np.floor(epochs \* patientRate)), restore\_best\_weights=True)

cb\_reduceLR = tf\_callbacks.ReduceLROnPlateau(patience=int(np.floor(epochs \* (patientRate \*\* 2))), factor=0.8, min\_lr=1e-4)

cb\_lists = [cb\_earlystopping, cb\_reduceLR, TqdmCallback(verbose=0)]

model\_mlp = {}

model\_mlp['model'] = createMLP\_DenseNet()

model\_mlp['model'].summary()

model\_mlp['model'].fit(x=train\_x\_oh, y=train\_y,

epochs=epochs, batch\_size=batch\_size,

validation\_data=(val\_x\_oh, val\_y),

verbose=0, shuffle=False,

callbacks=cb\_lists)

model\_mlp["pred"] = model\_mlp["model"].predict(val\_x\_oh, batch\_size=batch\_size).flatten()

model\_mlp["performance"] = {"RMSE": np.sqrt(metrics.mean\_squared\_error(val\_y, model\_mlp["pred"])),

"R2": metrics.r2\_score(val\_y, model\_mlp["pred"])}

print(model\_mlp["performance"])

{'RMSE': 97.0149181374261, 'R2': 0.7968101169388544}

#========================[ ㄴㅏ▣ ⓔ⇸ Z O N E ]=========================

B1 = layers.Dense(512, activation="relu")(B0)

B2 = layers.Dense(512, activation="relu")(B1)

B3 = layers.Dense(256, activation="relu")(B2)

B4 = layers.Dense(128, activation="relu")(B3)

Mean squared error: 8522.406784431943

R2 score: 0.7868502392769979

#===========================<∑ⓞㅖ㈜Zon∈>===========================

B1= layers.Dense(2\*\*12,activation="relu")(B0)

B2= layers.Dense(2\*\*11, activation="relu")(B1)

B3= layers.Dense(2\*\*10, activation="relu")(B2)

B4= layers.Dense(2\*\*9, activation="relu")(B3)

B5= layers.Dense(2\*\*8, activation="relu")(B4)

B6= layers.Dense(2\*\*7, activation="relu")(B5)

B7= layers.Dense(2\*\*6, activation="relu")(B6)

Mean squared error: 116.52502828173681

R2 score: 0.6496400179456532

#========================<ㅈl여lZON∈>==========================

B1 = layers.Dense(2 \*\* 12, activation="relu")(B0)

B2 = layers.Dense(2 \*\* 11, activation="relu")(B1)

B3 = layers.Dense(2 \*\* 10, activation="relu")(B2)

**RMSE: 99.162**

**R2: 0.751**

dropoutRate=0.2

B1 = layers.Dense(2 \*\* 12, activation="relu")(B0)

B1\_dropout = layers.Dropout(rate=dropoutRate, name="B1\_dropout")(B1)

B2 = layers.Dense(2 \*\* 11, activation="relu")(B1\_dropout)

B2\_dropout = layers.Dropout(rate=dropoutRate, name="B2\_dropout")(B2)

B3 = layers.Dense(2 \*\* 10, activation="relu")(B2\_dropout)

**RMSE: 92.212**

**R2: 0.785**

# ===== Kaggle MLP (URL 참고) =====

def createMLP\_LP():

hiddenLayers = 256

dropoutRate = 0.2

# Source : Kaggle - Laurent Pourchot

# URL : https://www.kaggle.com/pourchot/simple-neural-network

# === input layers ===

B0\_input = layers.Input(shape=train\_x\_oh.shape[1], dtype="float32", name="B0\_input")

B0\_embedding = layers.Dense(units=hiddenLayers \* 2, activation="relu",

kernel\_regularizer="l2", name="B0\_dense")(B0\_input)

# === learning layers ===

B1\_dense = tfa.layers.WeightNormalization(

layers.Dense(

units=hiddenLayers, activation="selu"), name="B1\_dense"

)(B0\_embedding)

B1\_dropout = layers.Dropout(rate=dropoutRate, name="B1\_dropout")(B1\_dense)

B1\_concat = layers.Concatenate(name="B1\_concat")([B0\_embedding, B1\_dropout])

B2\_dense = tfa.layers.WeightNormalization(

layers.Dense(

units=hiddenLayers, activation="relu"), name="B2\_dense"

)(B1\_concat)

B2\_dropout = layers.Dropout(rate=dropoutRate, name="B2\_dropout")(B2\_dense)

B2\_concat = layers.Concatenate(name="B2\_concat")([B0\_embedding, B1\_dropout, B2\_dropout])

B3\_dense = tfa.layers.WeightNormalization(

layers.Dense(

units=hiddenLayers, activation="elu"), name="B3\_dense"

)(B2\_concat)

B3\_dropout = layers.Dropout(rate=dropoutRate, name="B3\_dropout")(B3\_dense)

# === top layers ===

layer\_final = layers.Dense(units=int(hiddenLayers / 2), activation="relu", name="layer\_final")(B3\_dropout)

layer\_regressor = layers.Dense(1, name="Regressor")(layer\_final)

model\_mlp = Model(B0\_input, layer\_regressor)

model\_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),

metrics=tf\_metrics.RootMeanSquaredError(name="rmse"))

# model\_mlp.summary()

return model\_mlp