실험 2 MNIST hyper-parameter optimization 실습

1. 실험 목적

Neural Network를 구축하고 training할 때 선택할 수 있는 Hyper-parameter 가 무엇인지 이해하고 Bayesian optimization 방법을 통해 최적의 Hyper-parameter optimize되 MNIST model을 생성한다.

- Hyper-parameter optimization을 위해 scikit-optimize API를 사용
- learning-rate, layer 개수, node 개수, activation function, 등 hyper-parameter
- MNIST dataset을 training, validation, test 비율
- Hyper-parameter optimization model 생성
- Bayesian optimization을 이요한 최적의 MNIST model을 생성
- Evaluate Best Model on TestSet

2. 실험 전에 준비해야할 내용

a) Hyper-Parameter Optimization

TensorFlow 에서 Neural Network 를 구축하고 training 할 때 선택할 수 있는 parameter 는 다양하다. 이를 흔히 hyper parameter optimization 이라고 합니다. 네트워크의 layer 수에 관한 hyper parameter, layer 당 node 수에 관한 네트워크의 계층 수에 대한 hyper parameter, 사용할 activation function 에 관한 hyper parameter 등 다양한 예시가 있다. 최적화 방법 또한 learning rate 와 같이 하나 혹은 여러개의 hyper parameter 를 가질 수 있다.

hyper parameter 를 찾는 한 가지 좋은 방법은 하나하나 직접 실험해보며 최적의 performance 를 내는 것을 찾는 hand tuning 이다. 하지만 이것은 시간이 오래걸릴 뿐만 아니라 optimal parameter 가 사람의 직관과 어긋나 찾지 못할 경우도 있다.

b) Bayesian Optimizer using Gaussian Process

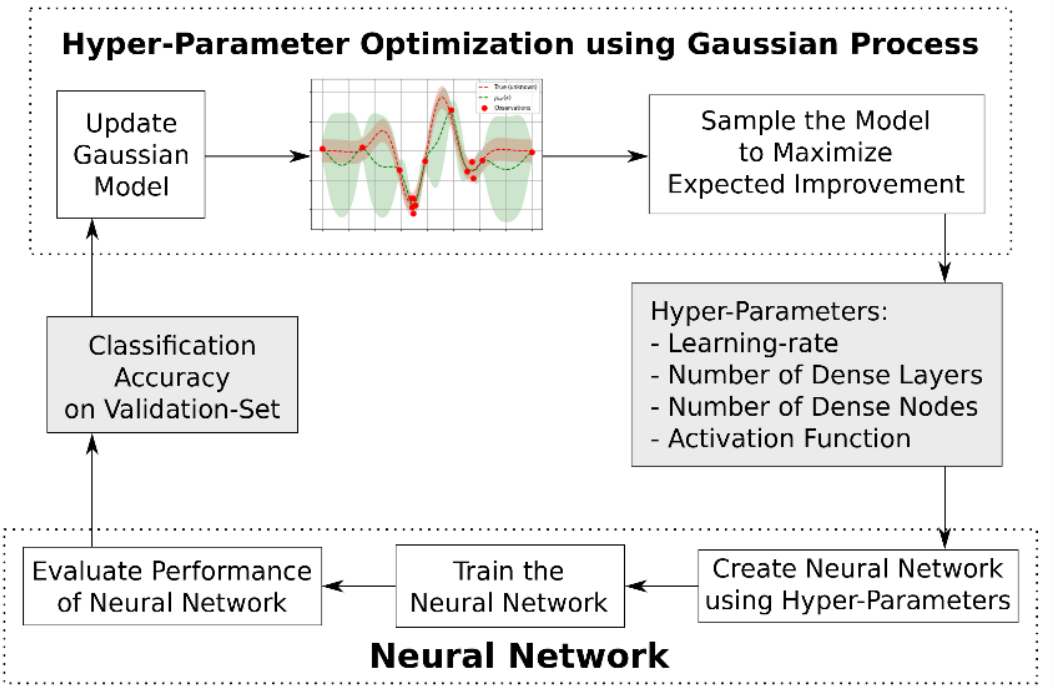
좋은 hyper parameter 값을 찾는 또 다른 방법은 hyper parameter에 넣을 수 있는 값들을 순차적으로 입력한 뒤에 가장 높은 성능을 보이는 hyper parameter를 찾는 탐색 방법이다. 이것은 Grid Search라고 부른다. 이 모든 것은 컴퓨터에서 돌아감에도 불구하고 시간이 많이 소요된다. 왜냐하면 hyper parameter가 추가될때 마다 parameter들의 조합은 기하급수적으로 늘어나기 때문이다. 이 문제는 curse of dimensionality라고 알려져있다. 예를 들어 각각 10가지 값을 갖는 hyper parameter 4개를 가지고 있다면 총 조합의 수는 10^4개이다. 하지만 하나의 hyper parameter만 추가하면 그 조합은 총 10^5개로 늘어난다.

Hyper parameter optimization의 문제점은 한 가지 parameter set의 성능을 확인하는데 굉장히 번거롭다는 것이다. 먼저 그에 맞는 신경망을 구축하고 훈련시키고 마지막으로 test set을 이용하여 그 성능을 확인해야하기 때문이다. 이 튜토리얼에서는 training을 빨리 끝내기 위해 작은 MNIST를 사용할 것이다. 그러나 실생활에서 다뤄지는 문제들은 아주 빠른 컴퓨터에서도 training이 적게는 수시간에서 몇 주까지도 걸린다. 그래서 우리는 필요한 순간에만 성능을 계산하는 최대한 효율적인 hyper parameter optimizaton 방법이 필요하다.

Bayesian optimization은 hyper parameters를 위한 search-space 모델을 만드는 것이다. 이 중 한가지는 Gaussian Process이다. 이것은 hyper parameter의 변화에 따라 성능이 어떻게 달라지는지 예측할 수 있게한다. hyper parameter의 실제 성능을 계산할 때 마다 우리는 약간의 노이즈만 제외하고 성능을 확실히 알 수 있다. 그 다음 Bayesian optimizer를 이용해 우리가 아직 확인하지 않은 search-space에서 더 나은 성능의 hyper parameter를 추천받는다. Bayesian optimizer가 충분히 좋은 모델을 구축할 때까지 이 과정을 몇 차례 반복한 후 우리는 최적의 parameter를 선택할 수 있

다.

Wiki page : https://en.wikipedia.org/wiki/Hyperparameter_optimization



3. 실습 실험을 진행하기 위한 환경 세팅

이 실습은 Host 서버에 접속하기 위해 terminal 프로그램을 설치하여 미리 만들어 난 VItis-AI Docker 환경에서 실습을 진행한다. Host 서버는 서울대학교 전기정보공학보 실습용 서버이며 세팅된 환경에서 실습을 진행한다.

a) Terminal 설치 및 사용

Host 서버에 접속하기 위해 MobaXterm terminal 프로그램을 설치하는 방법.

MobaXterm 다운로드 주소 https://mobaxterm.mobatek.net/

b) Vitis-AI docker 실행

VItis-AI Docker 환경을 실행하는 명령어

$cd Vitis-AI

$docker pull xilinx/vitis-ai-cpu:latest

$./docker_run.sh xilinx/vitis-ai-cpu:latest

```
ai_system10@ECE-util1:~$ cd Vitis-AI/
ai_system10@ECE-util1:~/Vitis-AI$ ls
LICENSE    data  docker_run.sh  dsa       external    models  tools
README.md  demo  docs           examples  index.html  setup
ai_system10@ECE-util1:~/Vitis-AI$ docker pull xilinx/vitis-ai-cpu:latest
latest: Pulling from xilinx/vitis-ai-cpu
Digest: sha256:1d568b1b77601a4e9989f969a74dfd9fd61102b713cb137edb83d76db11cea91
Status: Image is up to date for xilinx/vitis-ai-cpu:latest
docker.io/xilinx/vitis-ai-cpu:latest
ai_system10@ECE-util1:~/Vitis-AI$ ./docker_run.sh xilinx/vitis-ai-cpu:latest
NOTICE:  BY INVOKING THIS SCRIPT AND USING THE SOFTWARE INSTALLED BY THE
SCRIPT, YOU AGREE ON BEHALF OF YOURSELF AND YOUR EMPLOYER (IF APPLICABLE)
TO BE BOUND TO THE LICENSE AGREEMENTS APPLICABLE TO THE SOFTWARE THAT YOU
INSTALL BY RUNNING THE SCRIPT.

Press any key to continue...
```

```
Do you agree to the terms and wish to proceed [y/n]? y
Setting up ai_system10 's environment in the Docker container...
Running as vitis-ai-user with ID 0 and group 0


==========================================


__      ___ _   _                 /\   |___ \
\ \    / (_) | (_)               /  \    __) |
 \ \  / / _| |_ _ ___ ______    / /\ \  |__ <
  \ \/ / | | __| / __|______|  / ____ \ ___) |
   \  /  | | |_| \__ \        /_/    \_\____/
    \/   |_|\__|_|___/

==========================================

Docker Image Version:  1.4.916
Build Date: 2021-07-20
VAI_ROOT: /opt/vitis_ai

For TensorFlow 1.15 Workflows do:
     conda activate vitis-ai-tensorflow
For Caffe Workflows do:
     conda activate vitis-ai-caffe
For PyTorch Workflows do:
     conda activate vitis-ai-pytorch
For TensorFlow 2.3 Workflows do:
     conda activate vitis-ai-tensorflow2
Vitis-AI /workspace >
```

Vitis-AI Docker 환경에서 사용하는 library에 따라 콘다 환경을 세팅되어 있다.

다음의 명령어를 이용해 tensorflow2 콘다 환경을 실행할 수 있다.

$conda activate vitis-ai-tensorflow2

$pip install scikit-optimize

Scikit-Optimize(skopt)는 비싸고 noisy한 black-box function을 최소화하기 위한 간단하고 효율적인 라이브러리이다. Seqential 모델 기반 최적화를 위해 몇 가지 방법을 구현한다. 그리고 Skopt는 많은 상황에서 접근가능하고 재사용 가능하다.

```
(vitis-ai-tensorflow2) Vitis-AI /workspace > pip install scikit-optimize
Defaulting to user installation because normal site-packages is not writeable
Collecting scikit-optimize
  Downloading scikit_optimize-0.8.1-py2.py3-none-any.whl (101 kB)
     |████████████████████████████████| 101 kB 6.7 MB/s
Requirement already satisfied: numpy>=1.13.3 in /opt/vitis_ai/conda/envs/vitis-a
i-tensorflow2/lib/python3.7/site-packages (from scikit-optimize) (1.21.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /opt/vitis_ai/conda/envs/
vitis-ai-tensorflow2/lib/python3.7/site-packages (from scikit-optimize) (0.24.2)
Collecting pyaml>=16.9
  Downloading pyaml-21.8.3-py2.py3-none-any.whl (17 kB)
Requirement already satisfied: joblib>=0.11 in /opt/vitis_ai/conda/envs/vitis-ai
-tensorflow2/lib/python3.7/site-packages (from scikit-optimize) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in /opt/vitis_ai/conda/envs/vitis-a
i-tensorflow2/lib/python3.7/site-packages (from scikit-optimize) (1.5.3)
Requirement already satisfied: PyYAML in /opt/vitis_ai/conda/envs/vitis-ai-tenso
rflow2/lib/python3.7/site-packages (from pyaml>=16.9->scikit-optimize) (5.4.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/vitis_ai/conda/envs/
vitis-ai-tensorflow2/lib/python3.7/site-packages (from scikit-learn>=0.20.0->sci
kit-optimize) (2.2.0)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-21.8.3 scikit-optimize-0.8.1
(vitis-ai-tensorflow2) Vitis-AI /workspace >
```

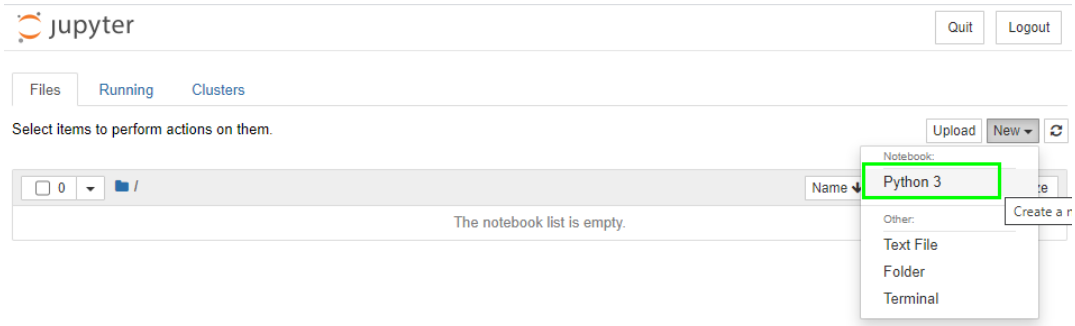
c) Jupyter Notebook 실행

주피터 노트북 앱은 웹 브라우저를 통하여 노트북 문서들을 편집하고 실행하게 하는 서버-클라이언트 애플리케이션이다. 인터넷 접근없이 로컬 데스크탑에서 실행될 수 있다. 또는 인터넷을 통하여 리모트 서버에 설치될 수 있고 접근가능한다.

이번 실습에서 주피터 노트북 앱을 이용해 코드 작성을 진행 한다.

$ jupyter notebook

Python script 작성



4. Jupyter Notebook을 통해 실습 실험

a) 필요한 library 추가 및 초기화 세팅

a) Import Libraries

```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import math
```

We need to import several things from Keras.

```
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Input
from tensorflow.keras.layers import Reshape, MaxPooling2D
from tensorflow.keras.layers import Conv2D, Dense, Flatten
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
```

```
import skopt
from skopt import gp_minimize, forest_minimize
from skopt.space import Real, Categorical, Integer
from skopt.plots import plot_convergence
from skopt.plots import plot_objective, plot_evaluations
from skopt.plots import plot_histogram, plot_objective_2D
from skopt.utils import use_named_args
```

b) Hyper-Parameters

For this demonstration we want to find the following hyper-parameters:

- The learning-rate of the optimizer.
- The number of fully-connected / dense layers.
- The number of nodes for each of the dense layers.

- Whether to use 'sigmoid' or 'relu' activation in all the layers.

c) Search-ranges for hyper-parameters

```
# search-ranges for hyper-parqameter

range_learning_rate = Real(low=1e-6, high=1e-2, prior='log-uniform', name='learning_rate')
range_num_dense_layers = Integer(low=1, high=5, name='num_dense_layers')
range_num_dense_nodes = Integer(low=5, high=512, name='num_dense_nodes')
activation_function = Categorical(categories=['relu', 'sigmoid'], name='activation')

dimensions = [range_learning_rate, range_num_dense_layers, range_num_dense_nodes, activation_function]
```

d) Default parameters 세팅

Before we run the hyper-parameter optimization, let us first check that the various functions above actually work, when we pass the default hyper-parameters.

```
default_parameters = [1e-5, 1, 16, 'relu']
```

e) Log 저장 function

```
# Log the training-progress for all parameter-combinations
def log_dir_name(learning_rate, num_dense_layers, num_dense_nodes, activation):

    # The dir-name for the TensorBoard log-dir.
    s = "./logs/lr_{0:.0e}_layers_{1}_nodes_{2}_{3}/"

    log_dir = s.format(learning_rate, num_dense_layers,
                       num_dense_nodes, activation)
    return log_dir
```

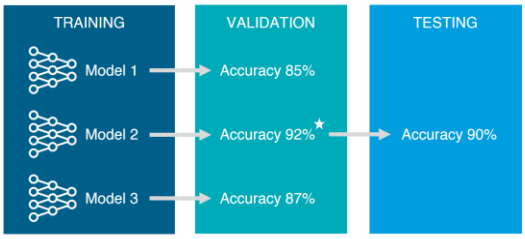
f) Load MNIST Dataset

Training set(훈련 데이터)은 모델을 학습하는데 사용된다. Training set으로 모델을 만든 뒤 동일한 데이터로 성능을 평가해보기도 하지만, 이는 cheating이 되기 때문에 유효한 평가는 아니다. 마치 모의고사와 동일한 수능 문제지를 만들어 대입 점수를 매기는 것과 같다. Training set은 test set이 아닌 나머지 데이터 set을 의미하기도 하며, training set 내에서 또 다시 쪼갠 Validation set이 아닌 나머지 데이터 set을 의미하기도 한다. 문맥상 Test set과 구분하기 위해 사용되는지, validation과 구분하기 위해 사용되는지를 확인해야 한다.

Validation set(검정 데이터)은 training set으로 만들어진 모델의 성능을 측정하기 위해 사용된다. 일반적으로 어떤 모델이 가장 데이터에 적합한지 찾아내기 위해서 다양한 파라미터와 모델을 사용해보게 되며, 그 중 validation set으로 가장 성능이 좋았던 모델을 선택한다.

Test set(테스트 데이터)은 validation set으로 사용할 모델이 결정 된 후, 마지막으로 딱 한번 해당 모델의 예상되는 성능을 측정하기 위해 사용된다. 이미 validation set은 여러 모델에 반복적으로 사용되었고 그중 운 좋게 성능이 보다 더 뛰어난 것으로 측정되어 모델이 선택되었을 가능성이 있다. 때문에 이러한 오차를 줄이기 위해 한 번도 사용해본 적 없는 test set을 사용하여 최종 모델의 성능을 측정하게 된다.

아래 예시에서는 training set : validation set : test set 비율을 7 : 2: 1 으로 나누져 있지만, 이번 실험에서 dataset 비율도 hyper-parameter이기 때문에 dataset 비율에 따라 model 정확도가 달라지는지 확인해 본다.



70.000 images and class-numbers for the images.

- Training-set: 48999
- Validation-set: 14000
- Test-set: 7001

```
from sklearn.model_selection import train_test_split
DATASET_SIZE = 70000
TRAIN_RATIO = 0.7
VALIDATION_RATIO = 0.2
TEST_RATIO = 0.1
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000,784)).astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train)

x_test = x_test.reshape((10000,784)).astype('float32') / 255.0
y_test = tf.keras.utils.to_categorical(y_test)

x = np.concatenate([x_train, x_test])
y = np.concatenate([y_train, y_test])

x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=(1-TRAIN_RATIO))
x_val, x_test, y_val, y_test = train_test_split(
    x_val, y_val, test_size=((TEST_RATIO/(VALIDATION_RATIO+TEST_RATIO))))

print('Training data: {}. {}'.format(x_train.shape, y_train.shape))
print('Validation data: {}. {}'.format(x_val.shape, y_val.shape))
print('Test data: {}. {}'.format(x_test.shape, y_test.shape))
```

```
Training data: (48999, 784). (48999, 10)
Validation data: (14000, 784). (14000, 10)
Test data: (7001, 784). (7001, 10)
```

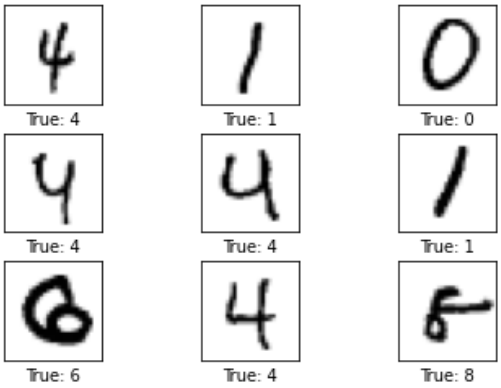
```
img_size = 28
img_size_flat = 784
img_shape = (28, 28)
img_shape_full = (28, 28, 1)
num_classes = 10
num_channels = 1
validation_data = (x_val, y_val)
```

g) Helper-function for plotting images

```
def plot_images(images, cls_true, cls_pred=None):
    assert len(images) == len(cls_true) == 9

    # Create figure with 3x3 sub-plots.
    fig, axes = plt.subplots(3, 3)
    fig.subplots_adjust(hspace=0.3, wspace=0.3)

    for i, ax in enumerate(axes.flat):
        # Plot image.
        ax.imshow(images[i].reshape(img_shape), cmap='binary')

        # Show true and predicted classes.
        if cls_pred is None:
            xlabel = "True: {0}".format(cls_true[i])
        else:
            xlabel = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

        # Show the classes as the label on the x-axis.
        ax.set_xlabel(xlabel)

        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    # Ensure the plot is shown correctly with multiple plots
    # in a single Notebook cell.
    plt.show()
```

h) 데이터 출력

```
y_test_cls = [np.argmax(y, axis=None, out=None) for y in y_test]
images = x_test[0:9]
cls_true = y_test_cls[0:9]
plot_images(images=images, cls_true=cls_true)
```



i) Function for plotting examples of images from the test-set that have been mis-classified.

```
def plot_example_errors(cls_pred):
    # Boolean array whether the predicted class is incorrect.
    incorrect = (cls_pred != y_test_cls)

    # Get the images from the test-set that have been
    # incorrectly classified.
    images = x_test[incorrect]

    # Get the predicted classes for those images.
    cls_pred = cls_pred[incorrect]

    # Get the true classes for those images.
    cls_true = y_test_cls[incorrect]

    # Plot the first 9 images.
    plot_images(images=images[0:9],
                cls_true=cls_true[0:9],
                cls_pred=cls_pred[0:9])
```

j) Model 생성

```
def create_model(learning_rate, num_dense_layers, num_dense_nodes, activation):

    model = Sequential()
    # Add an input layer which is similar to image-size.
    model.add(InputLayer(input_shape=(img_size_flat,)))
    model.add(Reshape(img_shape_full))
    # First convolutional layer. Using strides2 and max pooling 2
    model.add(Conv2D(kernel_size=5, strides=1, filters=16, padding='same',
                     activation=activation, name='layer_conv1'))
    model.add(MaxPooling2D(pool_size=2, strides=2))
    # Second convolutional layer. Using strides2 and max pooling 2
    model.add(Conv2D(kernel_size=5, strides=1, filters=36, padding='same',
                     activation=activation, name='layer_conv2'))
    model.add(MaxPooling2D(pool_size=2, strides=2))
    # Flatten output of the convolutional layers
    model.add(Flatten())
    # Add fully-connected / dense layers.
    for i in range(num_dense_layers):
        name = 'layer_dense_{0}'.format(i+1)
        model.add(Dense(num_dense_nodes, activation=activation, name=name))
    # Last fully-connected / dense layer with softmax-activation
    model.add(Dense(num_classes, activation='softmax'))

    # Use the Adam method for training the network.
    optimizer = Adam(learning_rate=learning_rate)
    #optimizer = sel_optimizer(learning_rate=learning_rate)

    # In Keras we need to compile the model so it can be trained.
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

path_best_model = 'best_trained_model.h5'
best_accuracy = 0.0
```

k) Training Function

Training fuction은 주어진 hyper-parameter로 신경망을 만들고 훈련시키는 함수이다. 그 다음 validation set에서 성능을 평가한다. 그리고 이 함수는 validation set에서의 negative classification accuracy인 fitness value를 반환시킨다. Skopt가 최대화 대신 최소화를 하기 때문에 negative이다.

```
@use_named_args(dimensions=dimensions)
def fitness(learning_rate, num_dense_layers, num_dense_nodes, activation):
    # Print the hyper-parameters.
    print('learning rate: {0:.1e}'.format(learning_rate))
    print('num_dense_layers:', num_dense_layers)
    print('num_dense_nodes:', num_dense_nodes)
    print('activation:', activation)
    print()

    # Create the neural network with these hyper-parameters.
    model = create_model(learning_rate=learning_rate, num_dense_layers=num_dense_layers,
                         num_dense_nodes=num_dense_nodes, activation=activation)


    # Dir-name for the TensorBoard log-files.
    log_dir = log_dir_name(learning_rate, num_dense_layers, num_dense_nodes, activation)

    # saves the log-files for TensorBoard.
    callback_log = TensorBoard(log_dir=log_dir, histogram_freq=0, write_graph=True,
                               write_grads=False, write_images=False)

    # Use Keras to train the model.
    history = model.fit(x=x_train, y=y_train,
                        epochs=3, batch_size=128,
                        validation_data=validation_data,
                        callbacks=[callback_log])

    # Get the classification accuracy on the validation-set
    accuracy = history.history['val_accuracy'][-1]
    print()
    print("Accuracy: {0:.2%}".format(accuracy))
    print()

    # Save the new model
    global best_accuracy
    if accuracy > best_accuracy:
        model.save(path_best_model)
        best_accuracy = accuracy

    # Delete the Keras model with these hyper-parameters from memory.
    del model
    K.clear_session()
    return -accuracy
```

l) Run Test for default hyper parameters

```
fitness(x=default_parameters)
```

```
learning rate: 1.0e-05
num_dense_layers: 1
num_dense_nodes: 16
activation: relu

Epoch 1/3
  1/383 [..............................] - ETA: 0s - loss: 2.3072 - accuracy:
0.0703WARNING:tensorflow:From /opt/vitis_ai/conda/envs/vitis-ai-tensorflow2/li
b/python3.7/site-packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (f
rom tensorflow.python.eager.profiler) is deprecated and will be removed after 2
020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/383 [..............................] - ETA: 13s - loss: 2.3052 - accuracy:
0.0781WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0168s vs `on_train_batch_end` time: 0.0530s).
Check your callbacks.
383/383 [==============================] - 7s 18ms/step - loss: 2.2262 - accura
cy: 0.2432 - val_loss: 2.1286 - val_accuracy: 0.3896
Epoch 2/3
383/383 [==============================] - 7s 18ms/step - loss: 1.9724 - accura
cy: 0.4740 - val_loss: 1.7971 - val_accuracy: 0.5587
Epoch 3/3
383/383 [==============================] - 7s 17ms/step - loss: 1.5918 - accura
cy: 0.6270 - val_loss: 1.3890 - val_accuracy: 0.6832

Accuracy: 68.32%


-0.6832143068313599
```

m) Run the Hyper-Parameter Optimization

이제 실제 scikit-optimize 패키지에서의 Bayesian optimization을 이용한 hyper parameter optimization를 실행할 준비가 되었다. 손으로 직접 찾은 Staring point로 default_parameters와 fitness()를 먼저 호출한다. 이것은 hyper parameter의 위치를 빨리 잡는데 도움이 된다. 15로 설정한 fitness() function의 call 숫자 등 실험 해볼만한 parameter들이 많다. 그러나 fitness()는 평가하기 비싸므로 특히 큰 신경망과 데이터셋에서 너무 많이 실행해서는 안된다. Bayesian optimizer의 internel model에서 새로운 hyper parameter 셋을 찾는 Acquisition function을 통해서도 실험할 수 있다.

```
%%time
search_result = gp_minimize(func=fitness, dimensions=dimensions,
                            acq_func='EI', # Expected Improvement.
                            n_calls=15, x0=default_parameters)
```

```
learning rate: 1.0e-05
num_dense_layers: 1
num_dense_nodes: 16
activation: relu

Epoch 1/3
  2/383 [..............................] - ETA: 15s - loss: 2.3123 - accuracy:
0.0859WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0154s vs `on_train_batch_end` time: 0.0671s).
Check your callbacks.
383/383 [==============================] - 7s 18ms/step - loss: 2.2687 - accura
cy: 0.1564 - val_loss: 2.2145 - val_accuracy: 0.1904
Epoch 2/3
383/383 [==============================] - 7s 17ms/step - loss: 2.1311 - accura
cy: 0.1794 - val_loss: 2.0390 - val_accuracy: 0.1899
Epoch 3/3
383/383 [==============================] - 7s 17ms/step - loss: 1.9321 - accura
cy: 0.2612 - val_loss: 1.8279 - val_accuracy: 0.3619

Accuracy: 36.19%

learning rate: 1.1e-06
num_dense_layers: 5
num_dense_nodes: 48
activation: sigmoid

Epoch 1/3
  2/383 [..............................] - ETA: 21s - loss: 2.4235 - accuracy:
0.0781WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0159s vs `on_train_batch_end` time: 0.0986s).
Check your callbacks.
383/383 [==============================] - 7s 19ms/step - loss: 2.4204 - accura
cy: 0.1002 - val_loss: 2.4314 - val_accuracy: 0.0974
Epoch 2/3
383/383 [==============================] - 7s 18ms/step - loss: 2.4134 - accura
cy: 0.1002 - val_loss: 2.4242 - val_accuracy: 0.0974
Epoch 3/3
383/383 [==============================] - 7s 18ms/step - loss: 2.4067 - accura
cy: 0.1002 - val_loss: 2.4174 - val_accuracy: 0.0974
```

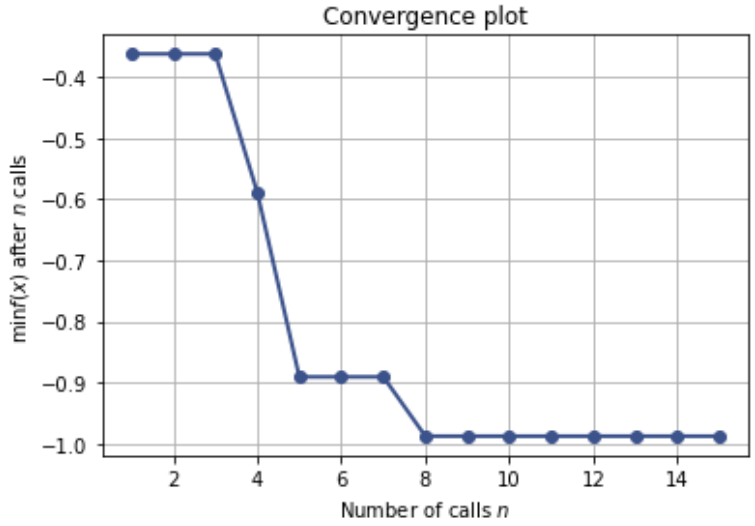
n) 최적화 과정

Hyper-parameter optimization의 과정은 쉽게 표기할 수 있다. 가장 좋은 fitness value는 y축에 있다. 이것은 validation set의 negated classification 정확도라는 것을 기억하라.

상당한 개선을 찾기까지 그렇게 많은 hyper parameter 테스트를 해보지 않았음을 확인해라.

```
plot_convergence(search_result)
```

```
<AxesSubplot:title={'center':'Convergence plot'}, xlabel='Number of calls $n$',
ylabel='$\\min f(x)$ after $n$ calls'>
```



o) Best Hyper-Parameters

The best hyper-parameters found by the Bayesian optimizer are packed as a list because that is what it uses internally.

```
# Find the best Hyper-Parameters
search_result.x
```

```
[0.0014181810091091485, 2, 165, 'relu']
```

```
def point_to_dict(point):
    return {"lat": point.y, "lng": point.x}
```

```
space = search_result.space
#space.point_to_dict(search_result.x)
```

```
# This is a negative number because the Bayesian optimizer performs minimization
search_result.fun
```

```
-0.9896000027656555
```

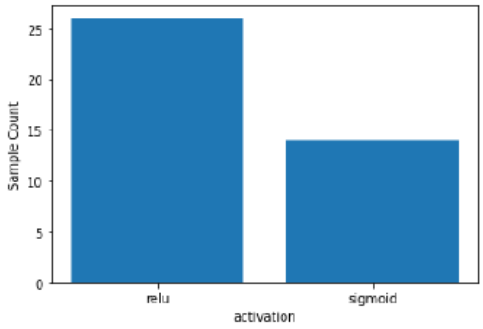
p) see all the hyper-parameters tried by the Bayesian optimizer

```
sorted(zip(search_result.func_vals, search_result.x_iters))
```

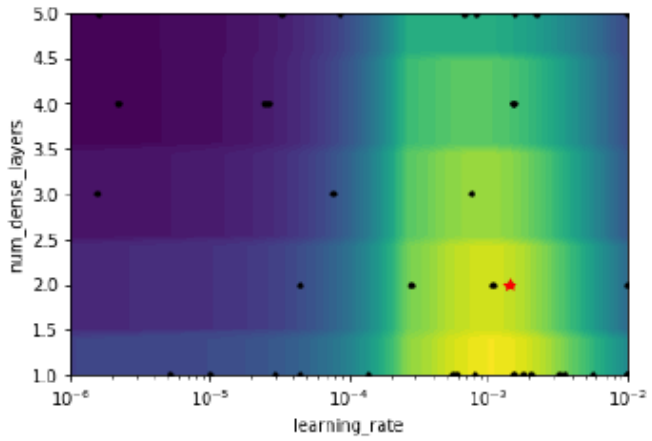
```
[(-0.9880714416503906, [0.0011100588934016583, 2, 398, 'relu']),
 (-0.9842857122421265, [0.0004405714106394455, 3, 219, 'relu']),
 (-0.9743571281433105, [0.002551188734752213, 1, 154, 'sigmoid']),
 (-0.9673571586608887, [0.00010169029918334655, 4, 139, 'relu']),
 (-0.9667142629623413, [0.001361590903001132, 1, 156, 'sigmoid']),
 (-0.890999972820282, [3.517801871872421e-05, 4, 33, 'relu']),
 (-0.7867143154144287, [2.9398534327893408e-06, 2, 428, 'relu']),
 (-0.6447857022285461, [1.8689563959773055e-06, 4, 398, 'relu']),
 (-0.5889285802841187, [2.2471392166156066e-06, 5, 193, 'relu']),
 (-0.3619285821914673, [1e-05, 1, 16, 'relu']),
 (-0.10499999672174454, [5.068785267184029e-06, 2, 196, 'sigmoid']),
 (-0.10499999672174454, [2.0705277834728642e-05, 5, 292, 'sigmoid']),
 (-0.103071428835392, [0.01, 1, 255, 'sigmoid']),
 (-0.1022857129573822, [4.768376669122543e-06, 3, 15, 'sigmoid']),
 (-0.0974285677075386, [1.0920438354715786e-06, 5, 48, 'sigmoid'])]
```

q) Visualization of Hyper-parameter Search

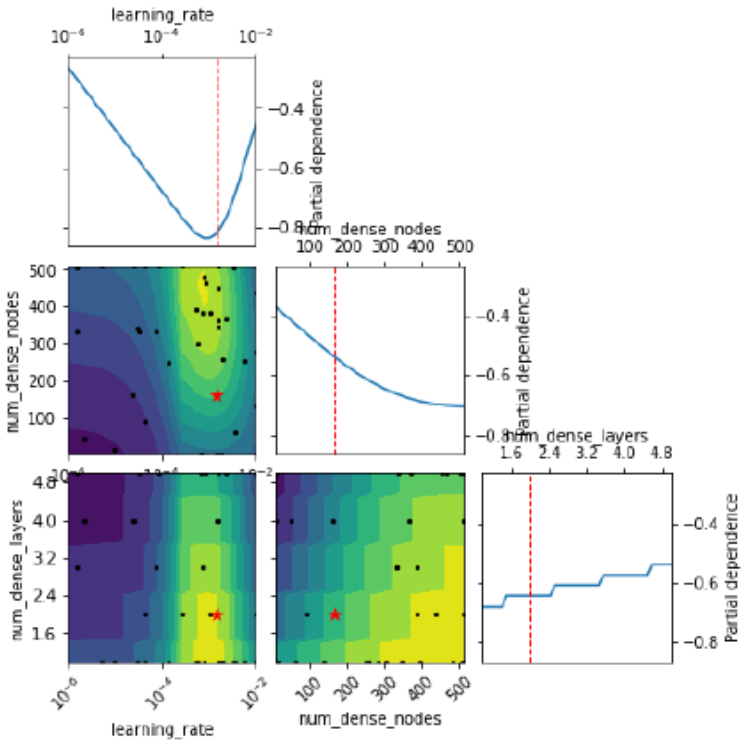
```
# activation parameter, which shows the distribution of samples during the hyper-parameter optimization.
fig, ax = plot_histogram(result=search_result, dimension_identifier='activation')
```



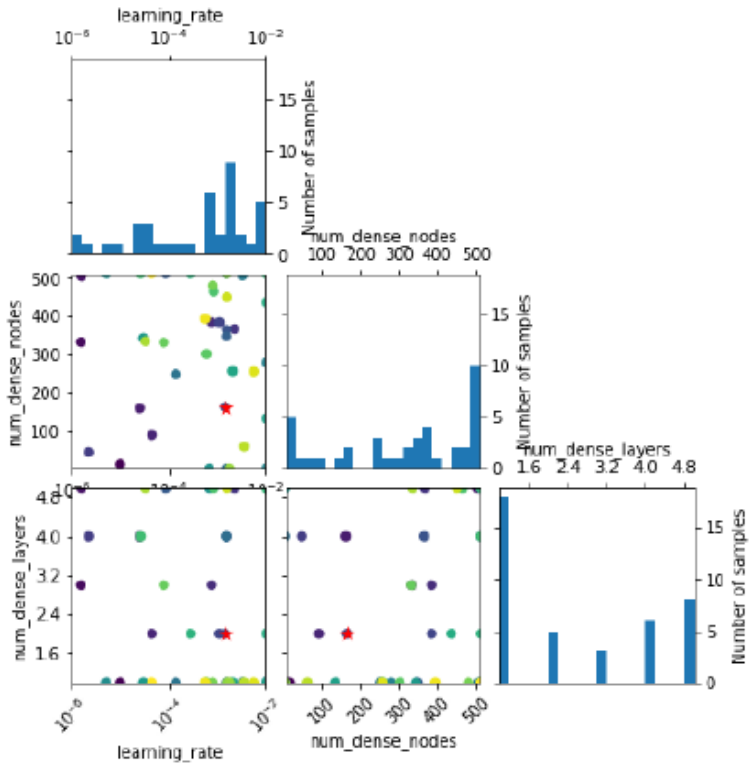

```
# learning_rate and num_dense_layers
fig = plot_objective_2D(result=search_result,
                        dimension_identifier1='learning_rate',
                        dimension_identifier2='num_dense_layers',
                        levels=50)
```



```
dim_names = ['learning_rate', 'num_dense_nodes', 'num_dense_layers']

fig, ax = plot_objective(result=search_result, plot_dims =dim_names)
```




```
fig, ax = plot_evaluations(result=search_result, plot_dims=dim_names)
```



r) Evaluate Best Model on Test Set

```
model = load_model(path_best_model)
result= model.evaluate(x=x_test, y=y_test)
```

```
219/219 [==============================] - 1s 3ms/step - loss: 0.0487 - accuracy: 0.9854
```

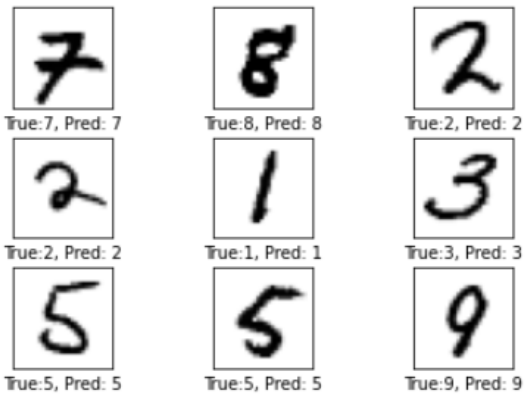
```
for name, value in zip(model.metrics_names,result):
    print(name, value)
```

```
loss 0.04866045340895653
accuracy 0.9854306578636169
```

```
print("{0}: {1:.2%}".format(model.metrics_names[1],result[1]))
```

```
accuracy: 98.54%
```

```
images = x_test[0:9]
cls_true=y_test_cls[0:9]
y_pred = model.predict(x=images)
cls_pred = np.argmax(y_pred, axis=1)

plot_images(images=images, cls_true=cls_true, cls_pred=cls_pred)
```



5. 실험 후 보고서에 포함될 내용

i. 최적의 accuracy를 갖는 hyper-parameter 값은 무엇인지 찾아보기

ii. Hyper-parameter를 이용하여 최적의 accuracy를 가지는 Batch size 값을 구하시오

iii. Hyper-parameter를 이용하여 최적의 accuracy를 가지는 dataset 비율

iv. 실험 1에 Tensorflow Keras API를 이용한 model accuracy와 최적의 model accuracy를 비교하고 어떤 방법이 가장 좋은건지, 그 이유가 무엇인지 대해 작성

v. Hyper-parameter를 수행하는 목적은 무엇인지 장단점에 대해 작성